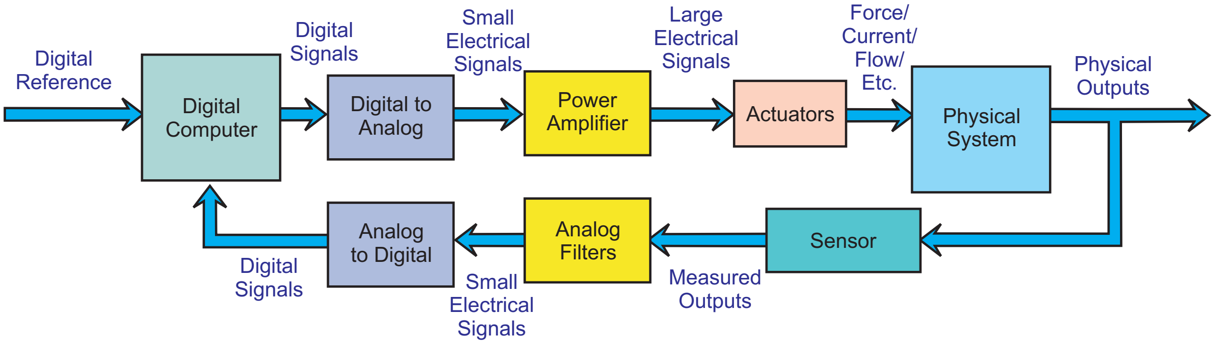
# Practical Methods for Real World Control Systems



Daniel Y. Abramovitch

System Architect, Agilent Technologies

**E-mail: abramovitch@ieee.org**

December 31, 2022; 9:46 PM

# Disclaimer

This set of materials started as a companion to the Practical Methods for Real World Control Systems Workshop. It has grown consistently in the five years that we have been doing the workshops as I have found new things to explain and cleaned up some of the older explanations. I would love it to be complete, but it will not be, at least in this first cut attempt. It will be information rich, and I believe that most of it will be accurate and useful. That being said, the reader should exercise engineering judgment in the use of this material. No warranties expressed or implied here, just the best thinking I can give you at this point, and a promise to try to fix any mistakes that I or others might find.

In its present form, the book is far beyond being a companion to the workshop but it is not ready for publication in a salable form. It is, though, time to put a zeroth edition out so that I can get more feedback on the ideas and presentations here. The hope is that by the time I get to the first edition, it will be something that is worth a few bucks.

In order to make things clear in this document, I will heavily rely on what can best be called sketches. Basically, these are like sketches in notes, but done on the computer. They allow me to convey concepts without physical constants, but are not meant to be exact. Whenever possible, I will back these up with plots from examples and when I see that the sketches vary too far from reality, I will try to make adjustments over time. What I will guarantee you is that I will only present material that I have actually gone through and understood thoroughly. Of course, this will limit the amount of material in the book, but as I am long winded, I believe I can make up for it.

Finally, the language is informal since I find that I communicate ideas far more effectively with a casual writing style. (I finally have something in common with Cervantés.)

**Created: May 9, 2018**

**Last Updated: December 31, 2022**

**Edition: 0.5a**

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Forward

This book was thrown together quickly in the spring of 2018 as a companion to the first run of the Practical Methods for Real World Control Systems workshop held before the 2018 American Control Conference (ACC) in Milwaukee, WI, USA. It took as its starting point, material I had produced for talks at other conferences, largely ACC 2016 in Boston, ACC 2015 in Chicago, and the Multi- Conference on Systems and Control (MSC) 2015 in Sydney, Australia. It was the invitation by Reza Mohamani (now at UT Dallas) that sent me on the path of trying to tie together the advanced methods I was starting to fully develop and understand with the seemingly simple methods that are so often used in practical control applications. As General Chair of ACC 2016 in Boston, I championed a structure which left Friday afternoon (the last half of the last day) open for tutorials on applications. The idea was that reduced registration costs for applications specific material would be popular with local practicing engineers. Well, the sessions were quite popular, but mostly with folks already attending the conference. Two of the tutorials that I produced (one on PID tuning with Sean Andersson and one on measurements for control system design) became the basis for the above workshop's first two hours, and for Chapters 3 and 4

Throwing these papers together created a base of material, starting with discussions of simple models and measurements and then branching into PID controller design. From there it was a run to connect that material with the filtering work of 2015 and practical state space representations. Plus, there was an opportunity to Spackle in work on noise measurement (the PES Pareto area) as well as things that I'd learned about feedforward from my interactions with Lucy Pao of the University of Colorado at Boulder and some of her former students, particularly Jeff Butterworth. I have also benefited greatly from my interactions with Sean Andersson and his students at Boston University. Sean has provided a great sanity check on the practicality of my work: if I see it in his lab, I must have done something partway right.

As one might guess, slapping a bunch of old documents together leaves a lot to be desired, in matters of overlapping materials, different notation, and generally providing a consistent theme and feel to the work. By the summer of 2018, it had started to look semi-organized, and I believe there were a lot of areas of unique treatment of material, but it was still a hot, sticky mess.

This update attempts to fill in a lot of areas where I left myself notes in red letters. Along the way, I sent some copies out to some of my friends. Among the most thorough has been Russ Rhinehart of Oklahoma State, who has given me great insights into the assumptions I am making, coming from the mechatronic world. Russ comes from the world of chemical process control, and because of his helpful comments, I am trying to make sure that that perspective shows up in the book.

That being said, 2019 had a lot more material, is better organized, and has helpful book-like features, such as a table of contents, a list of figures, a list of tables, and an index. Let's just say it's still a work in progress. In 2020, I was able to clean up some of the discussions on systems, modeling, and measurements, but also to vastly expand the chapter dealing with noise analysis (via PES Pareto) and with the understanding and use of demodulation methods with respect to servo signals.

## 1.1 The Purpose of the Workshop

The proverbial "gap" between control theory and practice has been discussed since the 1960s, but it shows no signs of being any smaller today than it was back then. Despite this, the growing ubiquity of powerful and inexpensive computation platforms, of sensors, actuators and small devices, the "Internet of Things", of automated vehicles and quadcopter drones, means that there is an exploding application of control in the world. Any material that allows controls researchers to more readily apply their work and/or allows practitioners to improve their devices through best practices consistent with well understood theory, should be a good contribution to both the controls community and the users of control. This workshop is intended as a small but useful step in that direction.

**The goal, the purpose of the workshop and the book can best be summarized as to provide essential context about algorithms – basic or advanced. This context helps us know which algorithms may or may not be useful in a class of problems and what adjustments, if any,**

**need to be made to fit that algorithm to the problem.**

## 1.2 The Purpose of this Book

This book was originally intended to be a companion document to the workshop: something that the workshop participant can take home with them and refer back to to go into more depth on the workshop material. However, there is more in the book than can ever be described in a one day workshop. Thus, the second purpose of this book is to be a handbook for bridging the gap between control theory and control practice. As the years have passed since the first workshop at ACC 2018 in Milwaukee, WI, USA, the book has grown and taken on a life of its own. Chapters are no longer collections of old tutorials or slides with a lot more words. They are calling to stand on their own. Over time, I have found new ways of describing things more cleanly and the material has tightened up. This would often have a sequence where I am struggling on explaining some stuff, then I start to unify a few ideas and it suddenly seems simple to me. Not sure of myself, I fire off an email to my workshop partners, Sean Andersson at Boston University and Craig Buhr at Mathworks. Once they haven't said anything too bad about my intelligence, I work it into a few slides and slip it into this tome.

I don't expect to replace any of the many classic books on control theory or digital control. They go into the math in much greater detail than I can here without triggering reader depression (or my own). Instead, this book takes up where those books often leave off: I discuss the mechanics of making good measurements, but I also discuss which measurements can be made in a practical way on a given system. I discuss PID controllers, but I also discuss the different forms that they take in different engineers' hands, and how to translate between those representations. I discuss filter shapes that I might use, but I also give sample code on how to implement the filter subroutine. The idea is to be a guide on how to get from the physical system into the control theory math and back again (hopefully with an improved controller for the physical system). The idea is to be able to see what limits the behavior of real systems so that I don't end up with optimal solutions that never see the light of day.

The idea is to be access the wonderful world of control theory in a way that is practical for understanding and improving real-world systems. The final value theorem (FVT) shows up in Chapter 4 to motivate integral control. Simple Bode plots for different types of systems can explain why PID controllers show up in different forms for different applications.

I shape two chapters with Bode's integral theorem, or rather Gunter Stein's brilliant explanation of the importance of Bode's integral theorem [1], both to show us the limits of loop shaping in Chapter 6 and in to lead us into a method of analyzing noise effects on the loop in a much more coherent way (PES Pareto [2]) in Chapter 7.

PES Pareto and sensitivity to both noise and time delay/latency lead us into a discussion of removing the noise from sensor signals, both those that operate in the baseband and those that are modulated onto a carrier. Modulated sensor signals lead us o a discussion of demodulation, both non-coherent and coherent, for servo systems. This is a topic usually reserved for communication theory texts, but they lack our sensitivity to time delay and that changes everything.

Chapter 8 tries to bring feedforward control into the discussion. This is generally made harder by the fact that feedforward control can mean a half dozen different things, depending upon who is talking about it. What is cool is that if you take a step back, almost all of these methods share something in common: the latency with which correction is applied is far less than when feedback is applied on the same problem.

There is a chapter on state-space methods (Chapter 9) that seems to grow every time I read it. The first have tries to give context for what is in many control theory texts, as well as adding in many topics that seem to not make the cut: the transfer functions of state-space controllers, adding in integration as a state space analog to the integrator in a PID, and adding in reference signals. From there, I poke a pin in many of the balloons holding potential applications of state-space controllers, returning (repeatedly) to the need for a good model to use those "model-based" methods. The last half of the chapter largely deals with my own inventions of state space, the biquad state-space (BSS) [3, 4] and the bilinear state-space (BLSS) [5] structures. Not only do these have better numerical properties than the standard canonical forms, but they largely preserve physical intuition, which is incredibly helpful in trying to debug a real world system.

There is a chapter on computation (Chapter 10) for control systems. This is a difficult chapter to write because the technology changes every time someone sneezes. What doesn't change are the principles that we use to pick a particular technology solution and so this chapter tries to focus on that.

One of the repeated focal themes that transcribes most of the material here is the notion of hidden assumptions buried in most problems. These are the assumptions that seem to be hidden in any problem area and taken for granted by the people who have been working a five or more years in that area (long enough to be viewed as seasoned). New entrants into an area, not knowing these assumptions, can often feel like they are missing context. That being said, as James Burke points out in The Day the Universe Changed[6], breakthroughs often come to any area when someone switches career paths into that new area and – not knowing those hidden assumptions – goes in a different direction. The point, which Burke's examples from global technological history emphasize, is that these assumptions often bound the problem. The case I try to make throughout the book is that we need to state these assumptions out loud, so that we know the boundaries we are using, and so that we can consider what happens if we modify them. Keeping those assumptions hidden is often what limits moving a methodology between different application areas (both with their own hidden assumptions). Where possible, e.g. say with why PID controllers are almost always discretized using a backwards rectangular rule, I will point these out and give my best guess as to why they are there.

Another repeated focal theme in this book is that of the importance of the quality of the model of the physical system we are trying to control. Let's be honest: on a lot of "easy to control" systems, where only nominal performance is needed, the model need not be that great. In fact, a huge percentage of practical control systems can be viewed as having some sort of basic first or second-order model without much parameter sensitivity. For those, matching anything other than the gross (approximate, not disgusting) system behaviors. At the other end of the spectrum are systems that truly need a lot of feedback and feedforward intervention, either to simply operate or to press the performance. In these cases, the cases in which people most often claim the need for advanced control methods, the quality of the model is critical to any application of those methods. Fundamentally, the model must do what Stephen Hawking wrote about in A Brief History of Time[7]: it must explain what we have seen with a small number of arbitrary constants and it must allow us to make predictions about what we will measure. I will add that to be useful for our practical controls problems, the model must do these things in the operational ranges that we care about. If we only care about basic performance, a good model need only match basic performance. If we care about pushing the system behavior using our control methods, then the model has to be accurate in these high performance regimes. What is often the case is that folks try to apply advanced methods which work extremely well in simulation (where the model matches itself) but fail on the real problem when the model match is poor. Thus, I will keep coming back to modeling and to making enough high-quality measurements to make those models valid.

None of these are complete, either in theory or in every use case, but the hope is that they give context, a framework by which we can place everything in a more appropriate place. And as Trevor Noah said on his last night hosting "The Daily Show", "Context is everything."

## 1.3 The Style of this Book

When my oldest son, Michael, was an undergraduate engineering student at UCSB (The University of California at Santa Barbara), he became righteously angry with the costs of engineering texts. He and his friends were paying US $150–$200 for engineering texts that they might use for only one quarter. He was quick to point out, “It’s ridiculous, and most of them aren’t very good beyond the one class. I know that my parents can afford to pay for mine, but a lot of my friends are the first ones in their families to go to college. Some of them have a lot of student loans to be there and making them pay that much for textbooks is absurd.” At the same time, I spoke with a friend who is co-author on one of the most popular automatic control texts there is, and asked him how much the latest edition costs and how much of that he gets. I was guessing US $150, but he corrected me higher. Then I asked what his royalties were and he and his co-authors split about US $15. Well, how much did it cost to manufacture a book? US $8. This is the, “Are you f***ing kidding me?!?” moment. The students are paying a huge amount but the money is not going to the producers of the content. I get that there are editors and typesetters to be paid, but it cannot account for a ten times markup. I swore to Michael then that this would never be that kind of a book. However it gets published in the end, the electronic version will be less that buying lunch at most coffee shops and the print version will cost the extra amount needed to turn a PDF into something physical.

There are pluses and minuses to this choice. I won’t be working with paid professional editors who can legitimately clean up badly written and overly ponderous text that I will inevitably create. On the other hand, I don’t have to worry about someone telling me to make a figure fewer colors to minimize printing costs, or to not repeat a figure in three different places. Oh, and why would I repeat a figure in three different places? Well, maybe because I don’t want the poor reader to have to go flipping back and forth between where the figure was originally defined and where it is being referenced again 200 pages later. I won’t have folks pushing the books at conferences, but then again, students and other buyers won’t be paying for sales folks to fly to exotic conference locations and set up book booths, so that some professor adopts the book and forces 30 students a quarter to pay US $200 each. This is what I will try to avoid if at all possible. I want this book to be accessible and inexpensive. I would rather you finding it so useful and affordable that you check for the update every couple of years.

I also get to use the informal conversational style that folks who come to my presentations and workshops seem to like. I use contractions. I make jokes. I drop in pop culture references that date back to before when the average reader was born. I call that culture. You can call it what you want if it makes the book easier to digest. My my point is that when most of us watched Peter Jackson’s “Return of the King”, only our bladders told us the movie was longer than 3 hours. When we watched JJ Abrams “The Rise of Skywalker”, we noticed. At some point folks just broke down, whining “Okay, bring back

Jar-Jar. I don't care anymore! Just let me out of here!" All of which is another way to say that I would rather make something long and easy to read than compact and inscrutable.

The most important thing you can get out of this book is context: context for theory, context for applications. Understanding why successful engineers make certain choices in certain domains over and over again frames the theory and allows us to have intuition about where we will hit limits. Knowing that may allow us to build those physical limits into our models and then extend where we think we can push the theory. There is a line I love from the movie, "A Bridge Too Far,"[8, 9] in which the general asks one of his lead subordinates, "What is the best way to take a bridge?" Answer, "Both ends at once." To bridge this theory/practice gap, to take that bridge, coming from one side or the other is likely to fail. We need to come at it from both ends at once.

## 1.4 Intended Audience for the Workshop and Book

I believe that this workshop will be of great interest to three types of audience members:

1) Academic researchers who are well versed in control theory but would like to learn more about issues practicing control engineers often encounter as well as techniques and methods often used outside of standard textbook solutions to enhance their students' experience in the classroom and laboratory.

2) Practicing engineers who work on physical control systems and products that use control with an interest in connecting their work to "best practices" motivated by theory.

3) Students who may be interested in adding laboratory experiments to their research or want to know how to make what they have learned applicable in industry. For each of these groups – and those that are somewhere in the intersection of them – this workshop will address the gap from both sides, so as to give the participant

## 1.5 Prerequisites

Every workshop, any technical book has to start with some assumption about what the audience and/or reader knows. In this, I assume that the participant(s) have had at least an undergraduate knowledge of feedback systems, of the digital sampling of data, and the basics of programming. Furthermore, this book (and the workshop) are intended for folks with an honest interest in being able to translate control theory into physical control systems. There are many fine books that introduce control theory. This is not intended to be one of them, although I hope that in understanding the issues involved in making the theory work, the reader will have a deeper understanding of the theory.

By undergraduate knowledge, I am generally talking about the first course in feedback control, in which the basic concepts of dynamic response, feedback, and stability are taught. Almost always, this includes ideas of Routh-Hurwitz stability and root loci. For electrical and mechanical engineers, this also includes a few weeks spent on frequency response methods. For chemical engineers, these methods are taught less frequently and used sparingly, in large part due to the slower time constants. In this book I will rely quite a bit on frequency response methods, although I have put a lot of effort into showing when these methods don't make sense. Still, for large classes of problems, having a strongly coupled understanding of the signal domain and the frequency domain is important. Undergraduate classes in control usually get to the proportional plus integral plus derivative (or proportional-integral-derivative i.e. PID) controller. While it is the bane of graduate students in control, it is remarkably effective and versatile for almost any control problem that can be reasonably modeled as having first or second order physical system model (a.k.a. plant). However, instead of presenting these as a separate topic – as done in many control books – or as a be all and end all – as in other control books, this work will attempt to classify the PID in its proper place: as the starting point controller for the low frequency, rigid body or spring-mass-damper behavior of the physical system. We will also spend more than the usual time on the discretization of PID controllers and what one can expect when applying part or all of a PID to different simple systems.

Depending upon the class, state space might be introduced at the end. It is rarely tied in well with the transfer function methods (some will argue this), and the students are often left with the "what just happened" look on their faces. I don't go into state space until the end, but throughout the book, I am trying to build up methods for getting better models from physical systems. After all, state space methods are model based methods, and model based methods require . . . a good model. (The difficulty of this last statement cannot be overemphasized, and yet it is often ignored in most first year controls classes.)

Sampling is another matter. While much of advanced control theory takes place in continuous time, the modern implementation of most control systems is in discrete time. That is, there are plenty of analog/continuous time components, but the implementation of the bulk of the control law (the decision making and most of the math) is done in a digital computer. Because of this, it is critical to have some understanding of sampled data systems. I won't expect a lot of derivations of different discretization methods, but it is important to understand the trade-offs of different discrete-time representations. This material will be most easily understood by a reader that has taken a class in digital control, but the approach here will be quite different from the standard textbook treatment in several ways:

- I stay away from a "set it and forget it" view of discretization. I think that we need to go back and forth to get make sure we understand the discrete models in the compute domain and how our sampled system interacts with the physical world. This is often overlooked.
- I will provide pseudo-code for some of the major block types. I think it is important to pay attention to the programming aspect of digital systems.
- I will try to explore the computational pictures of not only fast mechatronic systems but slow process systems. It is no exaggeration to say that many folks working at one end of the sampling spectrum act as if the other end does not exist. Instead, I will try to use these differences to show us common frameworks and how differences get exposed, so that we are ready when Moore's law moves something that was once only possible in analog circuitry into the domain of "slow" processors.

Still, some experience in digital filtering, digital control, or simply programming embedded systems helps to motivate the reader towards the turns in the text. It makes understanding my obsession with digital representations more understandable. A reader can get a lot out of the first few chapters without having taken a digital controls class, but most of the discussions of implementation assume that the control law is being executed by a digital computer.

Finally, the implementation of control laws, of filtering, and of any other type of decision making on a digital computer requires some (usually a lot of) programming. I believe that a control engineer cannot really tune their algorithms unless they are the ones involved in the implementation, and that means programming and debugging, making measurements in the lab, checking code and comparing physical measurements to what was predicted in their simulations. To paraphrase something that was once said about Willie Mays, the only person who truly knows what measurements need to be made is the person who needs them for their control algorithm. There is a lot of confusion about what engineers need to know for this, and I will try to explain some of the salient points.

In the end, I hope that this allows more control engineers to try to implement the stuff they simulate in Matlab and its relatives. I hope that practicing engineers working on control problems find enough useful material in here to allow them to better understand common structures across the systems that they see. Maybe they will see that some of the advanced methods can be useful, if the devilish details of implementation are taken care of.

One more thing about the style of this book. I will never be mistaken for Calvin Coolidge (a famously Laconic American President) nor do I have a fetish for Occam's Razor (or at least the interpretation that says the shortest explanation is the best). I am much more a fan of Einstein's notion that things should be made as simple as possible and no simpler. To that end, if I believe that repeating the same idea, diagram, justification in multiple places if it means that the reader doesn't have to flip back and forth between chapters, I will. To paraphrase Dolly Parton's classic, "It costs a lot of money to look this cheap," I may take a lot of pages to make something simple. There are no constraints from some brick and mortar publisher, so I will feel free to be informal, use color, and use repetition and a lot of graphics. I'm more likely to follow the old preacher sermon method of "Tell them what you're gonna tell them, tell them, and tell them what you told them," than the KISS method (keep it simple, stupid). In fact, I've never thought that the KISS method was appropriate for many technical situations. Instead, I prefer the KICK method (keep it clear, knucklehead) . Some things are just complicated, but we can work hard to make the explanations clear. It doesn't bother me and shouldn't bother you if that makes the PDF 20% longer, so long as it makes the understanding 30% easier.

Finally, after all these years, I have learned that I truly am an engineer down to my soul and what this means is that there is always something that can be improved. This book is a work in progress, and I expect that no matter how (or how often) it gets published, it will always be getting tweaked and optimized, just to make it a little bit better. Your comments and suggestions are always welcome. Your rants, maybe a little bit less. (I believe that's what Twitter is for.) Thank you for taking the time to read this. I hope it's helpful.

Danny Abramovitch, December 20, 2022

# Chapter 2

# Introduction

While Chapter 1 gave you, the reader, the style, purpose, and context of this book, this chapter is intended as a technical introduction and overview of the material. As stated in Chapter 1 (and later in this chapter), this book is absolutely not intended as a replacement for all the fine "introduction to control theory" texts out there. It is intended as a missing supplement, a companion book, a provider of context to the material in these other books. This book – like the workshop that birthed it – is intended to help engineers who want to apply control theory in the real world bridge the gap between theory and practice. With any luck, we can help control engineers answer the two most likely questions that occur in trying to implement real systems:

- "Why didn't that work?" and
- "What can I adjust to give it a chance of working?"

## 2.1 Introduction to the Introduction

"Practical Methods for Control" refers to the stuff one has to do when one wants to do controls in the real world. It is the crossing of the proverbial gap between theory and practice to get theory that can be used in practice and practical systems that are far more empowered by analysis and design. The difference between idealized analysis and the grubby details of implementation can be visualized in

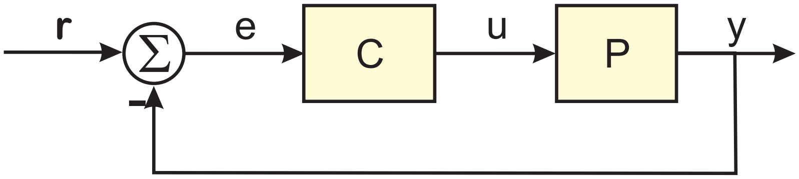


Figure 2.1: A generic control loop.

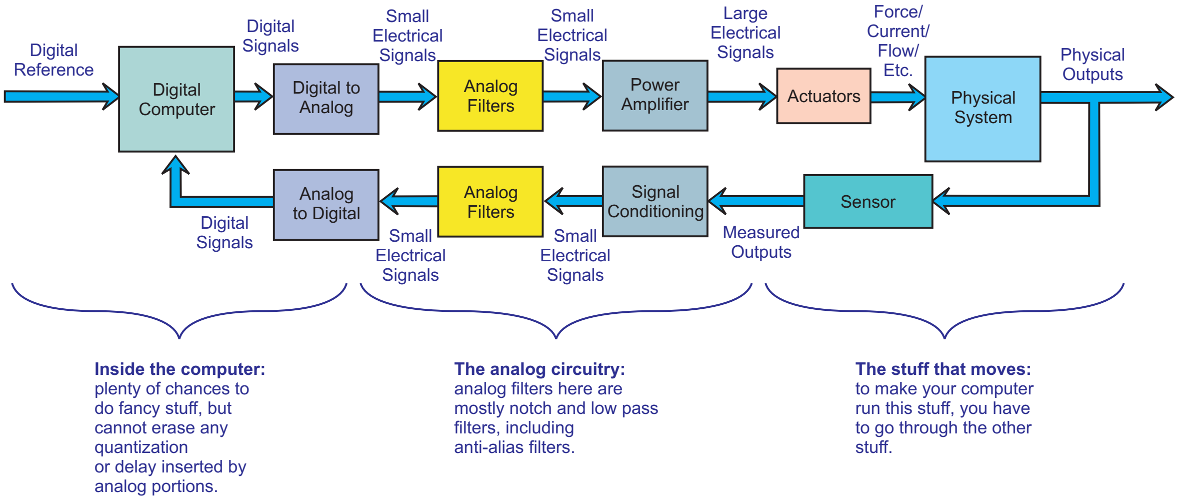

Digital Reference
Digital Computer
Digital Signals
Digital to Analog
Small Electrical Signals
Analog Filters
Small Electrical Signals
Power Amplifier
Large Electrical Signals
Actuators
Force/ Current/ Flow/ Etc.
Physical System
Physical Outputs
Sensor
Measured Outputs
Signal Conditioning
Small Electrical Signals
Analog Filters
Small Electrical Signals
Analog to Digital
Digital Signals
Inside the computer: plenty of chances to do fancy stuff, but cannot erase any quantization or delay inserted by analog portions.
The analog circuitry: analog filters here are mostly notch and low pass filters, including anti-alias filters.
The stuff that moves: to make your computer run this stuff, you have to go through the other stuff.

Figure 2.2: A more detailed, but still generic, digital control loop.

comparing Figures 2.1 and 2.2. Figure 2.1 is the type of diagram we would associate with analysis, but to implement a control system, we need to consider all the components in Figure 2.2. The main difference, conceptually, is that there are a lot more pieces between the physical system and the controller implemented in the digital computer. Typically, to get from Figure 2.2 to Figure 2.1, a lot of those pieces get either swept into $P$ or $C$. For example, $C$ might include the Digital Computer, and the converters. The power electronics, actuators, and sensors may very well be wrapped up in $P$. The analog filters might go in either direction, depending upon the design style.

That being said, while compressing all those blocks into $P$ and $C$ might help with analysis, that involves an assumption that this simplification doesn't significantly affect the response. If that were always the case, we would not need to worry about practical methods. Another way to look at the two sides of the great divide is to consider them as a form of responsive reading.

One side is the **academic/theoretical (AT)** side, where one starts with analytical models, CAD tools (e.g. MATLAB), and no end of advanced algorithms. Here, all the cool kids use advanced control (e.g. state space), use online identification and learning algorithms, use optimization methods on sophisticated models that may or may not represent reality.

On the other side is the **implementation/industrial (II)** world, where experience often is considered more important than analysis, where models – if they show up – are first or second order (plus some crap to be eliminated separately) where the kinds of controllers used are "Both kinds: PID and three-term." This world is inhabited by practicing control engineers, often many years past that one-semester college class on automatic control. Sometimes, students stumble in, attempting to apply what they have spent the previous term deriving. Mostly though, it is about getting something to work well enough and not to hinder the rest of the system.

Our goal here is not to close this proverbial, 60-year gap, but to use some "simple tricks and nonsense" [10] to get a few rope bridges across it. Let's throw out a few ideas. As we will sometimes have to do, getting the general idea will involve some broad generalizations and simplifications. So, follow along with Brother Danny. The following list is best read as a call/response type reading that one might hear in a religious service or rock concert. It helps if one's voice changes with each color.

- In **academic/theoretical (AT)** problems, we start with a model.
- In **implementation/industrial (II)** problems, getting to a reasonable model is 85–95% of the control design.
- In academic/theoretical (AT) problems, we often analyze in continuous time (CT).
- In implementation/industrial (II) problems, we almost always implement using a digital computer. It may – and usually does – connect to the real world via analog circuitry, but the control law, parameters, decision making, and intelligence are almost all implemented digitally these days.
- In AT problems, the models are "mostly" linear time-invariant (LTI), but can be quite large. If there is a dominant nonlinearity in the problem, then dealing with this is usually the main focus of the work.
- In II problems, the models – if the are written down – are first or second order. When the physical system presents higher order properties, a lot of effort is made to beat down, segment, divide up, etc. the problem into second order chunks.

There are exceptions, mostly in the high-end “fighter-plane” or large refinery problems. These problems usually have the characteristic that multiple engineers are needed to tune/adjust a single instance of the design.

- In AT problems, the filtering of extra dynamics is often folded into the control design.
- In II problems, filter blocks are added along the analog/digital path in a piecemeal way to remove imperfections in the signal path close to where they occur, thereby isolating them from the rest of the system.
- In AT problems, a more complex algorithm that produces a slightly better response on some metric leads to a publication. #Winning
- In II problems, the goal is to make a reliably working system, a device that operates, a product that sells. Adding ten times the complexity for a 2% increase in some metric is not something that will make it to market.
- In AT problems, the nonlinearities are often added in at the end to make the problem “more real”.
- In II problems, the nonlinearities usually frame and limit the scope of the control design.
- People working on AT problems would love to get their advanced algorithms into hardware on physical systems, but it’s hard to make that stuff work.
- People working on II problems would love to add advanced algorithms to their physical systems, but it’s hard to make that stuff work.

As with any modeling, we have to generalize in order to extract common threads. Many of these should seem obvious once stated, but they are often quite hidden until someone writes them down or says them out loud. I will try to write them down here as much as I can. We will follow up this set of broad, oversimplified, massively generalized assumptions with a few premises:

- State space filtering and control are model based filtering and control, and model based control and filtering require (wait for it . . . ) a good model.
- Extracting models from measurements is imperfect and depends strongly on the device/system being measured.
  - Certain physical systems admit only certain types of measurements.

  - Each measurement class only gives a certain amount of information about a system and no one measurement gives a complete picture.

- Virtually all the quantities measured from a physical system start out as analog quantities. This means that the signal conversion is important.

- Even the internal digital signals inside the controller are usually digital facsimiles of analog quantities.

- All the measurements are eventually digital, in that they result in sampled data that gets fed into a computer. For this book, when we say computer with no modifications, we mean digital computers – what anyone born after 1970 likely thinks of exclusively as computers. There are lots of other forms of computing: analog, hybrid, biological, quantum, etc., but we will specify them if we talk about them. Analog computing, while a great concept and useful from a circuitry implementation perspective, is no longer mainstream. The rest have not yet become mainstream. Most of the time in this book, “computer” means digital computer, a device base on approximating real numbers and differential equations in binary logic.

- Most physical systems are engineered to present a low order model – at least in their basic, low frequency behavior.

Therefore, in trying to understand practical control methods:

- We will pick some representative low order models that demonstrate what we see in the physical world (Section 2.3).

- We will talk about what kinds of measurements we can make to determine the parameters of these models.

- More importantly, we will talk about how to write code (or what type of code needs to be written) to make the most of these measurements.

- We will discuss why for any first or second order model, a well tuned digital PID (proportional-integral-derivative controller) and some filters have all the needed degrees of freedom to implement excellent (not optimal) control.
  - We will give a common framework for PID controllers.
  - We will discuss the different ways to design PID controllers based upon what model measurements are available.

  - We will discuss the hows and whys of PID discretization.
  - We will talk about what to expect from a closed-loop PID response when applied to one of our canonical physical models.

- We will then move beyond the simple models to discuss the hows and whys of compensating extra dynamics. In particular, much is made of filter design with little thought to how to implement compensating filters. Someone has to go there. It might as well be us.

- A key factor in being able to intelligently design control systems is the ability to rapidly iterate between measurements, modeling, design, and implementation. We will discuss the types of data connections needed between these devices, and give examples of how this can be done.

  - The lack of data connectivity in labs is one of the key limitations in applying more advanced methods to practical problems. It make it hard to connect measurements to models, makes designs hard to connect to real-time controllers.
  - The issue is that getting this connectivity is a slow, tedious pain-in-the-butt (technical term) during which time "real results" are not being produced. Because of this, the "infrastructure building" step is often shunned or at least minimized both in academia and industry. However, while you may be able to get from (a) to (b) over a dirt road, you cannot go fast or move much. To move truckloads of data or even sports car loads at high speed, you need that highway. Taking the time to build the infrastructure gives engineers the ability to iterate rapidly, which is one of the things we do best.

- We cannot understand, create, or modify computer control implementations without knowing something about computers, and how to get data in and out of them, so we will discuss computational models and different methods of getting control implemented on digital computers. While there are discussions of this throughout the book, Chapter 10 focuses on this.

It makes no sense to discuss modern control systems without some discussion of the effect of sampling on physical system models. In particular, the obfuscation of physical parameters created in any sampling scheme should make us reconsider most identification methods for use in practical systems.

In Section 2.3, we will present a set of low order models that can be considered representative of the types of practical systems that get controlled with simple controllers, such as PIDs. These models are helpful because in initially looking here, we are able to frame the discussion of model identification from measurements, the subject of Chapter 3. In particular, we can see what parameters can be extracted from step responses, and how practical use of step response must be coupled with an assumption of one of these model types [11].

Having simple, linear models can help us get a “best we can do” idea for our control systems. In our discussions of practical identification, we will see how the best information we can get out of step responses depends on assuming one of these ideal models. In Section 4.11 of our chapter on simple controllers and PIDs, we will see what we can say by assuming one of these models and picking one of the PID forms [12].

Of course, knowing the ideal response would be great except that “stuff happens”, and that stuff makes things worse that the “best we can do”. We will start an ongoing discussion of these in Section 2.5. This stuff limits our use of simple models and forces us to think about a lot of other issues.

Because discretization has such a strong effect on almost all our control systems, we will then give a useful introduction in Section 2.5.1. Of course, topics of discretization will pop up gain in almost every chapter. Instead of discussing discretization in one place and then assuming everything stays in one domain or the other (continuous or discrete), this book will pop back and forth between the two representations of the same model, trying to give insight into which issues are inherent in the problem and which show up simply when we discretize. We also are not content to stay with a “one size fits all” discretization method and will pick and choose the ones that seem best for a particular problem or part of a problem.

Again, this book was not intended to replace a first course in feedback control or one of the many fine books on control theory [13, 14] and digital control [15, 16]. we will assume here that the readers (a.k.a. you) have already taken a course in feedback control and maybe even in digital control. We (a.k.a. I) assume that the readers (once again, you) are familiar with stability, of poles, zeros, and gains (“Oh, my!”), of Bode plots, of gain and phase margins, of discretization, Laplace, Fourier, and Z transforms, of sampling, and of time delay. A good place to add more controls textbook references.

In any event, there are many books to explain these, some of them quite readable. We (a.k.a. I) are not going to waste your time trying to replace these. Instead, this book is about the things that limit the implementation of real control systems; about things that must be dealt with if we want to see theory come to life. To get there, a large part of what we have to do is put all those theoretical topics in better context with each other and with implementation methods and constraints.

Time and again, when all the control levers have been applied, the limiting factors are latency (a.k.a. time delay) and noise. We also need to realize that while we like to deal with linear models, most systems have some amount of nonlinearity. There are nonlinearities that limit the range of a variable, nonlinearities that distort a signal, nonlinearities that are repeatable, and nonlinearities that drift over time. We need to understand the limits imposed by nonlinearities, bot understand the operating ranges of our linear model based designs. When there is a distortion, can we reliably and repeatably undo that numerically?

Even for linear time-invariant (LTI) systems, we come back to latency and noise. We need to understand the sources and impacts of each of these. We need to see where we can limit the noise at its source, where we can minimize the latency through our implementation of circuits or computer programs. At the same time, we need to understand how these factors limit what we can do with the control loop. The latter might be about loop dynamics, but he former involves understanding the system components. Time and again, we will see the need for component knowledge. We may not need to be experts everywhere, but we should be conversant everywhere so that we can work with component experts to get what we need.

## 2.2 Use the Digital, Luke

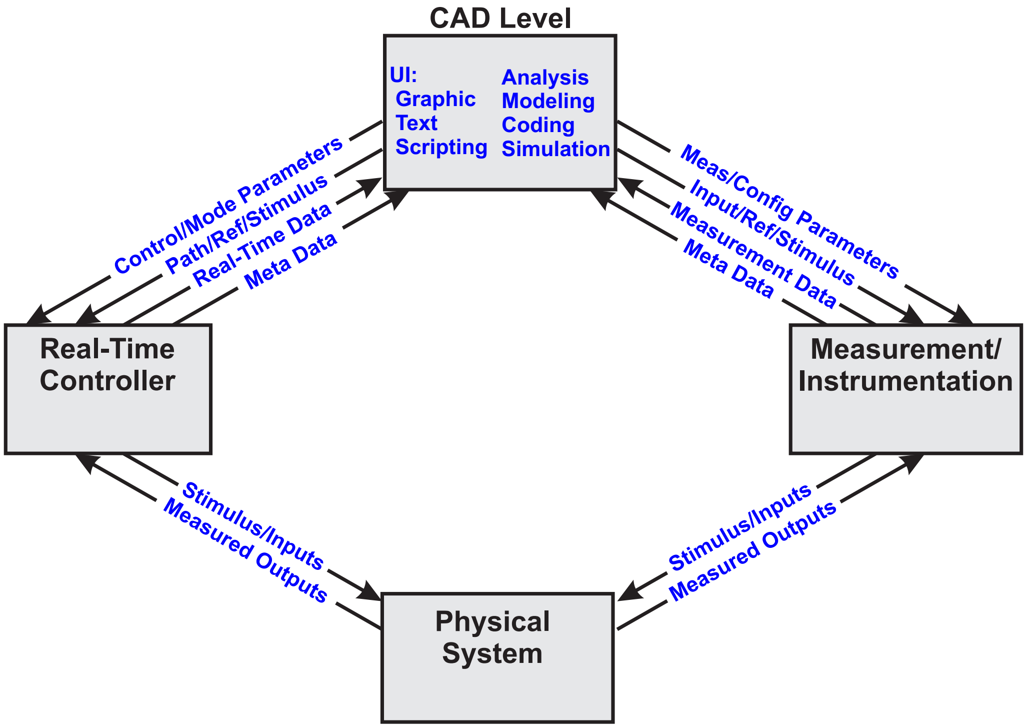


Figure 2.3: A view of test, analysis, and design for a real-time system.

One of the ideas that gets little discussion in a lot of controls texts are all the things beyond the implementation of a control law that one gets out of digital control methods[17]. Digital methods give us more than a chance to do control implementation in embedded software; they allow us to do things that do not lend themselves easily to circuits and continuous-time analysis. Sanity checking results, switching modes depending upon the inputs from auxiliary sensors, updating parameters, and applying repetitive control methods or iterative learning control are all things that really can only be done in the digital world. This has been understood since the success of the Apollo Program [18]. However, the teaching digital control often focuses on how close we can get to analog behavior. Thus, the bureaucracy of the data handling needed to do these things needs to be accepted as part of enabling the vastly superior performance – not due to an improved implementation of an approximated differential equation, but due to the ability to switch between multiple methods depending upon what the measurements are saying. In our opinion, this gets far too little discussion in standard controls texts.

We will try to correct for some of this in Chapter 10 where we deal with real-time computation for control, but we can get a preview by taking a look at Figure 2.3. Figure 2.3 presents a conceptual schematic of a development environment for control design. At the bottom we have a physical system. At the top we have our “CAD (Computer Aided Design) Level”. The physical system is reality, and the top layer represents where we do our thinking and have the bulk of our analysis and design tools. The question then is what goes between them to connect them? Well, since we are talking about implementing control systems, there must be some sort of real-time controller. We will assume for most of the work described in this book that the real-time controller is digital computer based, but historically this was not the case. The other often-ignored component is the measurement engine/instrumentation. The diagram shows the type of data that gets shared between these two intermediate stages and the physical system on one side, and the intermediate stages and the top level. This data is almost always in digital form these days, and that implies that we need to be able to convert between the different data formats, sample rates, measurement lengths, etc. This is a at its core a bookkeeping problem. It is boring and tedious to get right. And it is absolutely necessary and often ignored.

The increasing universality of digital implementations of control have also exposed another issue: a needed change in how measurements for control design are accomplished. Measurements for control design are an often yet another under-discussed subject, despite their strong effect on any reasonable effort to identify the plant model from actual signals. After all, we can measure anywhere within a **Simulink**environment, but the same is clearly not true in the physical world.

Looking back to Figure 2.3, we see that we push parameters, stimuli, and mode changing information down towards our real-time layer (the controller and the instrumentation), while gathering the data and meta-data they gather from the physical system. In the days before computers, the mechanism to do this was the human operator, perhaps with the aid of some analog measurement devices. Still, it was up to the human to combine the various sources of data, do the analysis, and implement the design, then once again measure the resulting system. The digitization of these layers have provided the potential to automate much of this, to make the data exchange and conversion much more automated. However, someone has to write the code to make those transfers and conversions nearly automatic. The connectivity allows for rapid iteration between measurement, analysis, design, implementation, and more measurement. I think this is woefully neglected by most controls researchers.

## 2.3 Low Order Models

In this section, we present a handful of low order models that describe basic behaviors we see in practical control systems. These models will be used both in extracting parameters from measurements and in understanding the closed-loop behavior of various PID forms (P, PI, PD, and PID) on these models. Moreover, these models are quite representative of ones seen in practice, and so we can use them to get insights into what can and cannot be extracted from different measurement types to quantify our systems. How do we get from first principles (science) to models that help us do better control designs? What parts of our models can we verify from actual measurements? How do we work knowledge of current working loops into modeling for improved performance of the same system?

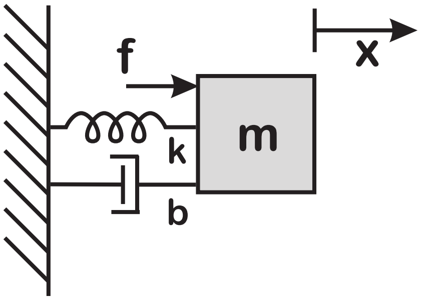


Figure 2.4: Spring-mass-damper system represents the typical second order mechatronics system.

It turns out that a lot of the simple, practical control problems that are solved using a PID or lead/lag controller can be classified as one of a handful of first or second order models. Of course, a closer look often reveals a lot of extra dynamics, but a vast number of problems end up being modeled this way. For this reason, a lot of practical control designs can be considered to be using one of these or a combination of them.

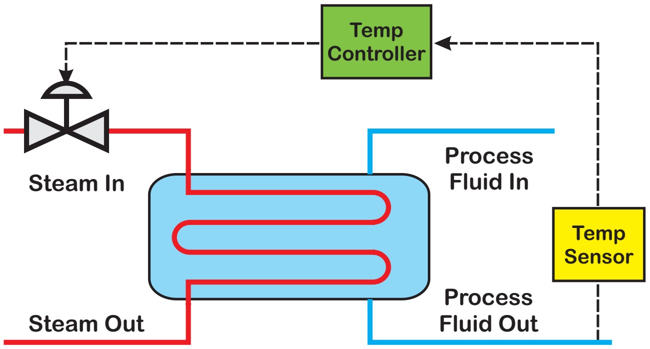


Figure 2.5: The heat exchanger is a classic first order system with delay.

In particular, we can consider the following simple systems. They can be considered as the “rigid body mode” or baseband portion of more complex systems. When possible, we will discuss where these models arise. We will also show a typical Bode plot and note the key characteristics of that plot for each model

Note that we are giving physical models in continuous time although most controllers that we will discuss and most measurements that we will make are done with computers in discrete time. For our purposes, the continuous time is far closer to the physical system and therefore gives cleaner understanding. We will spend some time on how to reconcile the continuous time models with discrete time measurements.

### 2.3.1 Integrator

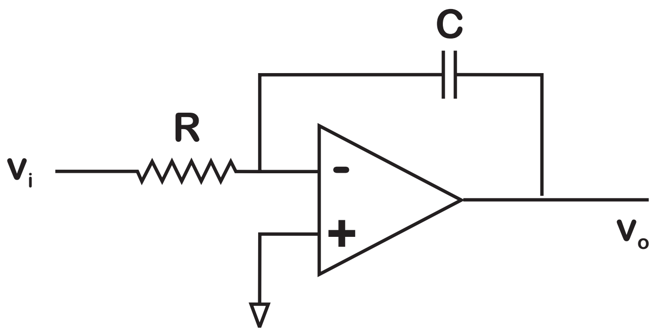


Figure 2.6: An op-amp circuit that implements an integrator.

Figure 2.6 shows a simple analog circuit that implements an integrator function. The integrator response is given by:

$$\frac{X(s)}{F(s)} = H(s) = \frac{K}{s}. \tag{2.1}$$

A note about circuits for my analog designer friends: Yes, I realize that nobody would build circuits the way they are diagrammed here. The idea is to present the fundamental circuit so that we can extract the Laplace transform model and get some back of the envelope analysis going.

An integrator may well be the easiest physical system to control [19]. In fact, many controller designs first try to make the open loop look like an integrator before closing the loop on that integrator. From a PID perspective, any of the combinations, P, PI, PD, and PID, can control this loop with (theoretically) no upper limit on gain. Furthermore, when the open loop response looks like an integrator, then the open loop has infinite gain margin and $90^\circ$ phase margin.

### 2.3.2 Integrator with Delay

If the transport delay in the system is a significant portion of the response, then we modify Equation 2.1 to include this time delay factor.

$$\frac{X(s)}{F(s)} = H(s) = \frac{K}{s}e^{-sT_D} \tag{2.2}$$

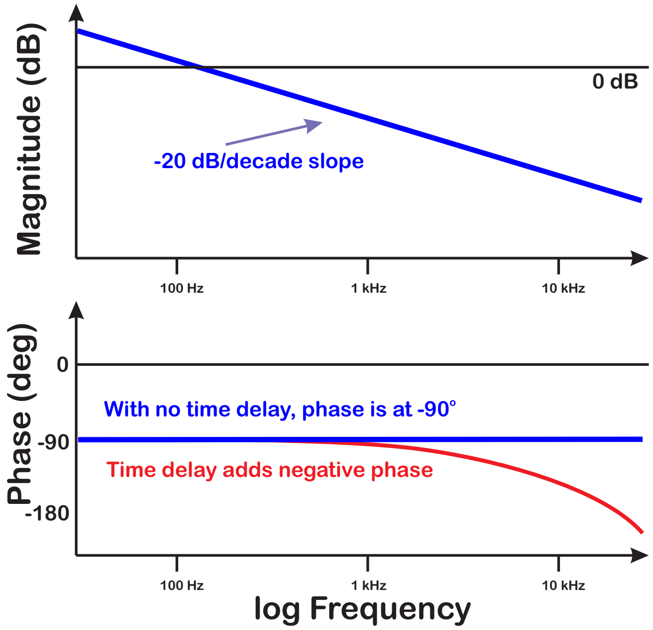


Figure 2.7: A schematic Bode plot response of a single integrator model. It is characterized by a phase that is flat at $-90°$ and a magnitude that drops off at $-20$ dB/decade. It is stable. The addition of delay has no effect on the magnitude response but adds "negative phase" that gets more negative as the frequency gets higher.

If one generates a Bode plot of Equation 2.2, then the magnitude response is identical to that of Equation 2.1, but the phase response is significantly different. In time this corresponds to a system with impulse response of

$$\begin{array}{ll} h(t) = K1(t - T_D) & \text{for } t - T_D \geq 0 \\ 0 & \text{otherwise} \end{array} \tag{2.3}$$

Including delay always makes things worse in a system model, because eventually the growing negative phase takes away all the phase margin. Thus, while all PID variants can control this model, there is an upper limit on the gain for any of them.

### 2.3.3 First Order Low Pass with Delay

Pure integrators are nice constructs in theory, but hard to find in the physical world. Often they are leaky, which means that rather than the stead response to a step, their impulse response is

$$\begin{array}{ll} h(t) = Kae^{-a(t-T_D)} & \text{for } t - T_D \geq 0 \\ 0 & \text{otherwise} \end{array} \tag{2.4}$$

which has the transfer function of

$$\frac{X(s)}{F(s)} = \frac{Ka}{s+a} e^{-sT_D}. \tag{2.5}$$

This equation in one form or another is typically used to model problems such as flow problems or heat exchangers [20, 21, 22], diagrammed in Figure 2.5. As such it is a fundamental system in chemical process control. Many systems, especially in heat flow, biology, and chemical process control, are adequately described by this model. As with the integrator with delay, all the PID variants can control this model, but there is an upper bound on the gain determined by the variant used, the delay, and the pole of the filter.

### 2.3.4 Bilinear Filter

When there is differentiation as well as integration, then there are dynamics in the numerator of the transfer function, as seen in Equation 2.6:

$$\frac{X(s)}{F(s)} = K\left(\frac{a_{1c}}{b_{1c}}\right)\left(\frac{s+b_{1c}}{s+a_{1c}}\right) \tag{2.6}$$

This is the form that analog lead circuits, used to stabilize systems by providing band limited differentiation. It is also the form that an analog lag filter takes. The key difference is that for a lead, $b_{1c} < a_{1c}$ and for a lag $b_{1c} > a_{1c}$.

An simple, generic op-amp circuit is shown in Figure 2.9. If we do standard op amp circuit analysis, we have a single current going from $V_o$ to ground and from ground to $V_i$ so that

$$i = \frac{V_o}{Z_o} = -\frac{V_i}{Z_i}, \text{ or} \tag{2.7}$$

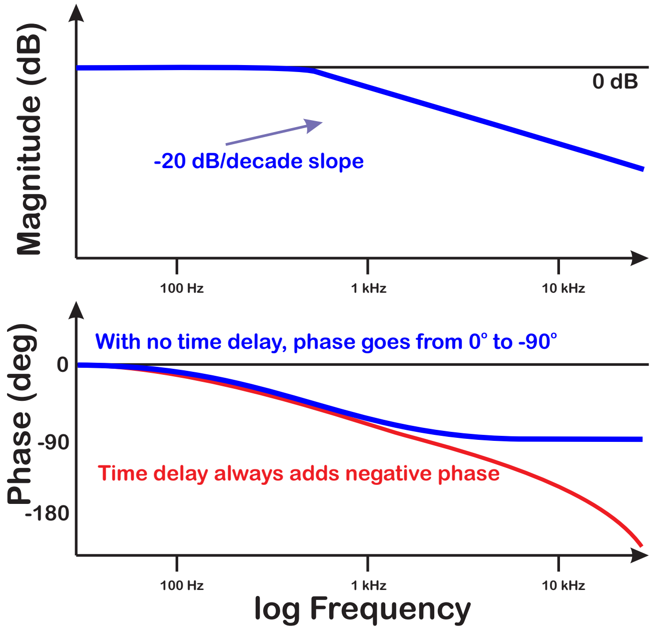


Figure 2.8: A schematic Bode plot response of a first order model. It is characterized by a phase that goes from 0° down to −90° and a magnitude that is flat and then drops off at −20 dB/decade. It is stable. The addition of delay has no effect on the magnitude response but adds "negative phase" that gets more negative as the frequency gets higher.

$$\frac{V_o}{V_i} = -\frac{Z_o}{Z_i}. \tag{2.8}$$

We have

$$Z_o = \frac{\frac{R_2}{sC_2}}{R_2 + \frac{1}{sC_2}} = \frac{\frac{1}{C_2}}{s + \frac{1}{R_2C_2}}. \tag{2.9}$$

Likewise

$$Z_i = \frac{\frac{R_1}{sC_1}}{R_1 + \frac{1}{sC_1}} = \frac{\frac{1}{C_1}}{s + \frac{1}{R_1C_1}}, \tag{2.10}$$

so that

$$\frac{Z_o}{Z_i} = \frac{\frac{\frac{1}{C_2}}{s+\frac{1}{R_2C_2}}}{\frac{\frac{1}{C_1}}{s+\frac{1}{R_1C_1}}} = \left(\frac{C_2}{C_1}\right)\left(\frac{s + \frac{1}{R_1C_1}}{s + \frac{1}{R_2C_2}}\right). \tag{2.11}$$

Finally,

$$\frac{V_o}{V_i} = -\left(\frac{C_2}{C_1}\right)\left(\frac{s + \frac{1}{R_1C_1}}{s + \frac{1}{R_2C_2}}\right). \tag{2.12}$$

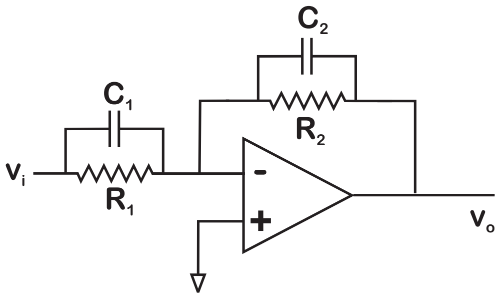


Figure 2.9: A generic simple circuit diagram of a bilinear filter.

Comparing Equations 2.6 and 2.12, we see that $b_{1c} = \frac{1}{R_1C_1}$ and $a_{1c} = \frac{1}{R_2C_2}$, so we can get a lead if $R_1C_1 < R_2C_2$ and a lag if $R_1C_1 > R_2C_2$. We also have

$$K\left(\frac{a_{1c}}{b_{1c}}\right) = -\left(\frac{C_2}{C_1}\right) \tag{2.13}$$

although making this exact would depend upon finding the right resistors and capacitors. The simple diagram of Figure 2.9 is not a very good circuit from an analog design point of view, but it does make for an easy to understand analysis. The other thing is that this simple circuit is inverting, so we would likely follow it with an inverting voltage follower circuit to buffer the signal and change the sign (Figure 2.10) or use a different configuration going into the non-inverting input of the op-amp.

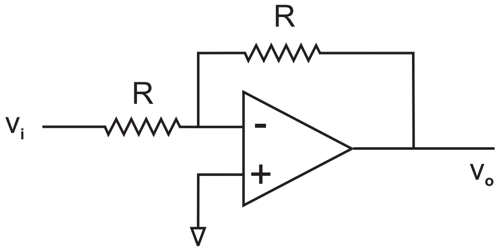


Figure 2.10: A generic simple circuit diagram of an inverter.

### 2.3.5 Double Integrator

Another model that shows up in countless idealized control problems is the double integrator (Figure 2.11), so named because it is described by the solution to the differential equation,

$$\begin{array}{ll} \ddot{x}(t) = 0 + Kf & \text{for } t \geq 0 \\ 0 & \text{otherwise} \end{array} \tag{2.14}$$

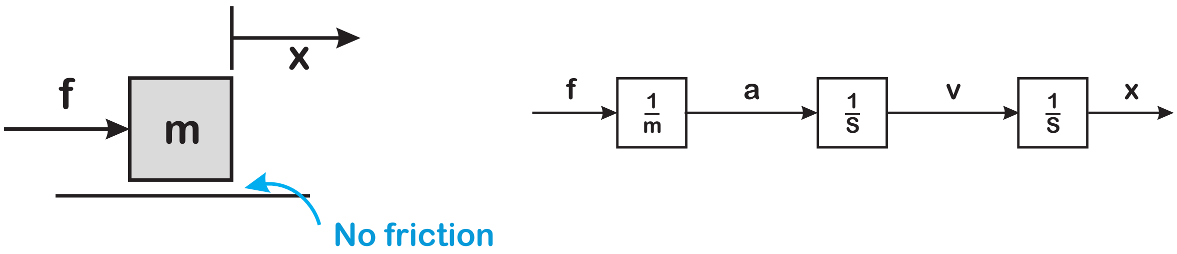


Figure 2.11: A pure mass on a frictionless plane is one physical system that results in a double integrator model.

where $f$ is the input to the system. This has the transfer function of

$$\frac{X(s)}{F(s)} = \frac{K}{s^2}, \tag{2.15}$$

and is seen in most physics problems as a mass on a frictionless plane. If $x(t)$ is displacement, then we get back Newton's $f = ma$ where $\ddot{x} = a$ and $K = 1/m$.

This model is the starting point for many simple electro-mechanical systems. In fact, some papers and texts really never get to the point of adding extra dynamics because they are hard to deal with. However, they do represent the behavior of a 3-axis stabilized satellite (where each axis behaves as a double integrator), since there is no friction in space or wind to deal with. However, this model does not describe the more flexible spacecraft parts (such as the robot arms used on the NASA's space shuttle or on the International Space Station).

We will see that even from such a simple model we can predict when each of the forms of PID will work. P and PI will not suffice, but PID can and PD may be the best of all.

### 2.3.6 Double Integrator with Delay

If we model the transport delay along with the double integrator, we get:

$$\frac{X(s)}{F(s)} = \frac{K}{s^2} e^{-sT_D} \tag{2.16}$$

Everything gets worse and more limited with delay, so while our PD controller may work for whatever

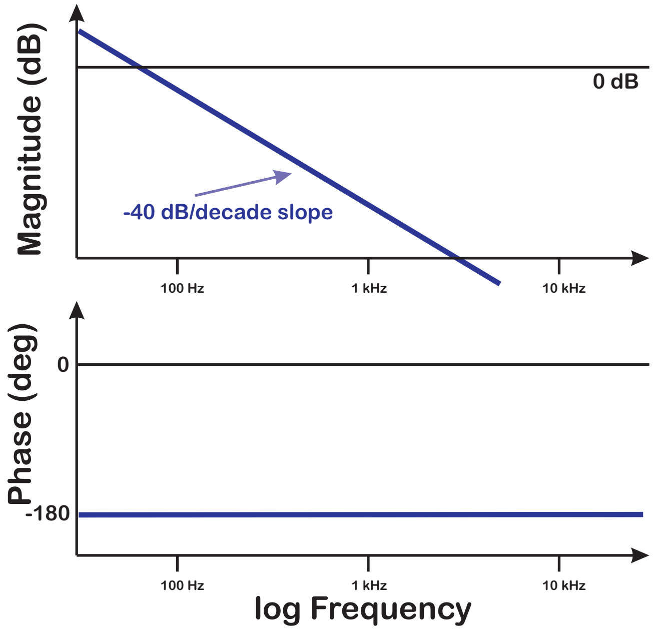


Figure 2.12: A schematic Bode plot response of a double integrator model. It is characterized by a phase that is flat at −180° and a magnitude that drops off at −40 dB/decade. It is not stable in the sense of bounded output to a bounded input, but it is relatively easy to control.

gain we wish in the double integrator case, the addition of delay will cause us to have to limit the gain based on the negative phase that the delay provides.

### 2.3.7 Pure Delay

We can also have a model that is a pure time delay, that is, the output reproduces the input but delayed $T_D$ seconds. The transfer function for this is:

$$\frac{X(s)}{F(s)} = Ke^{-sT_D}, \tag{2.17}$$

and it has a gain of $K$ for all frequencies, but an ever decreasing phase. Signals passing through lines without attenuation, or simply data passing through a computer system exhibits this kind of behavior. Two examples of such signals are shown on the left of Figure 2.13. On the right is sketched the resulting Bode plot for the pure delay with gain $K$.

Delay is a weird thing, like that relative in the penitentiary. Everyone knows it's there, but discussing it

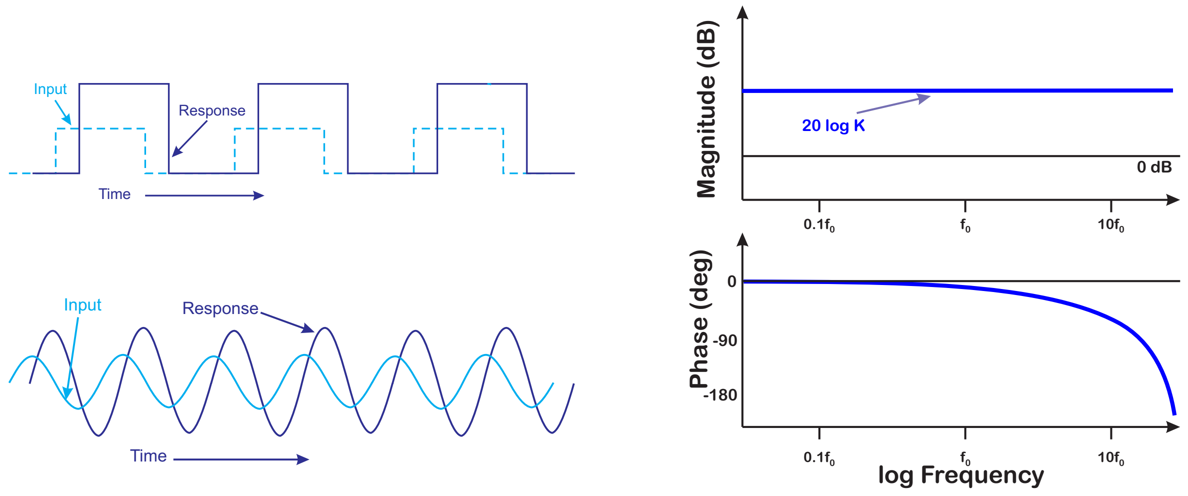


Figure 2.13: On the left, simple examples of pure time delay in a system. The output reproduces the input, with perhaps some scaling, but no other distortion. On the right is a sketch of the Bode plot this produces: flat gain ($20logK$) and phase going from $0$ to increasingly negative with high frequency.

is complex (no pun intended). It is easy to consider on a Bode plot, but not so much on a root locus. Furthermore, if the delay is not an integer number of sample periods, life gets a lot more complicated for filter and state space analyses.

However, unlike that incarcerated relative, time delay is certain to show up again. Now, it may not matter: the relative amount of time delay might be so small compared to the system dynamics that it has no practical effect. Still, it is worth having the tools to understand time delay and put it in the context of its affect on the system response.

As shown in Figure 2.17, delay has no affect on the response magnitude, but generates phase lag. The phase lag is proportional to the time delay and the frequency of the signal. It looks curved in Figure 2.17 only because the frequency axis is logarithmic. Eventually, time delay will put a limit on bandwidth. Fractional sample time delay will mess up our ability to make reasonable measurements for modeling.

However, what might be the most difficult part of understanding time delay is that for many classes of systems, it just doesn't matter. For example thermal, pressure, and systems have such slow dynamics that virtually any real-time computing system is far faster than the dynamics and any delay in the dynamics. In this case people often get confused about which delay is being discussed, which delay dominates.

### 2.3.8 Simple Resonance with No Zeros

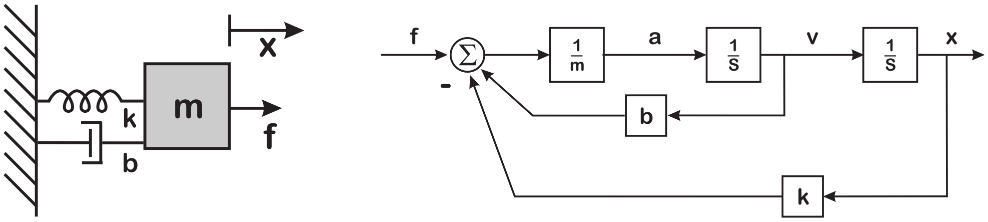


Figure 2.14: Simple spring-mass-damper problem. On the left is a schematic diagram and on the right is the resulting system block diagram. The schematic diagram can also be obtained by augmenting a double integrator with feedback from both the position and velocity term, resulting in a second order, stable system. If the ratio between the velocity feedback term and the spring feedback term is small enough, we get the complex roots of a resonant system.

The spring-mass-damper system of Figure 2.4 provides an easy to visualize model for a lot of simple electromechanical systems as well as for a lot of analog circuits. The block diagram of such a physical system is derived from adding position feedback ($k$, corresponding to the spring term) and velocity feedback ($b$, corresponding to the damping term) to the diagram of Figure 2.11 to get Figure 2.14. The transfer function is:

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}}, \tag{2.18}$$

which can be put into resonance terms:

$$\frac{X(s)}{F(s)} = K\frac{\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}, \tag{2.19}$$

by setting $\omega_d^2 = \frac{k}{m}$, $2\zeta_d\omega_d = \frac{b}{m}$, and $K\omega_d^2 = \frac{1}{m}$.

This is a fundamental structure [14] that shows up in physics tests, filter models, etc. In Figure 2.14, we see that it can be represented by a direct force acting on a mass, connected to an immovable wall. This is known as a spring-mass-damper system.

### 2.3.9 Simple Resonance with One Zero

$$\frac{X(s)}{F(s)} = K\frac{s + b_{1,c}}{s^2 + 2\zeta_d\omega_d s + \omega_d^2} \tag{2.20}$$

We will see that even from such a simple model we can predict when each of the forms of PID will work, P and PI if the overall gain is low enough, PD if the gain is high enough, and PID if we have an accurate model.

The presence of a zero simplifies the control problem in some ways because the differentiation adds phase lead and eventually limits the system's negative phase. In the model above, the high frequency phase is $-90°$. However, we should keep in mind that every physical system is eventually low pass. That is, eventually, all physical systems have a frequency response that attenuates at high frequency. (This helps keep the universe from blowing up, since signals continuing to have amplitude at infinite frequency probably causes problems.) That means eventually, everything has lots of negative phase. However, this is far beyond the limits of where the model above might be applied.

### 2.3.10 Resonance with Anti-Resonance (notch)

We do not usually see this system in isolation since most physical systems exhibit some sort of low pass behavior when we get to high enough frequencies, and some sort of rigid body behavior at the lowest frequencies. Mechatronics engineers are used to seeing these responses out beyond those rigid body behaviors.

$$\frac{X(s)}{F(s)} = K\left(\frac{\omega_d^2}{\omega_n^2}\right)\left(\frac{s^2 + 2\zeta_n\omega_n s + \omega_n^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}\right), \tag{2.21}$$

This model also shows up as a system component. For example, analog conditioning circuits often have notches (anti-resonances in the numerator of (2.21)) to eliminate problem frequencies. Flexible mechanical systems may have many, many resonance, anti-resonance pairs in them, which would end up as a series of models like Equation 2.21.

Figure 2.15 shows the biquad as the $i^{th}$ member of a chain of biquads in which $a_{i1} = 2\zeta_{id}\omega_{id}$, $a_{i2} = \omega_{id}^2$, $\tilde{b}_{i1} = 2\zeta_{in}\omega_{in}$, $\tilde{b}_{i2} = \omega_{in}^2$, and $b_{i0} = K\frac{\omega_{id}^2}{\omega_{in}^2}$.

The modeling of such systems in state space using chains of analog (or digital) biquad filters has been described by the author in [4] and [3].

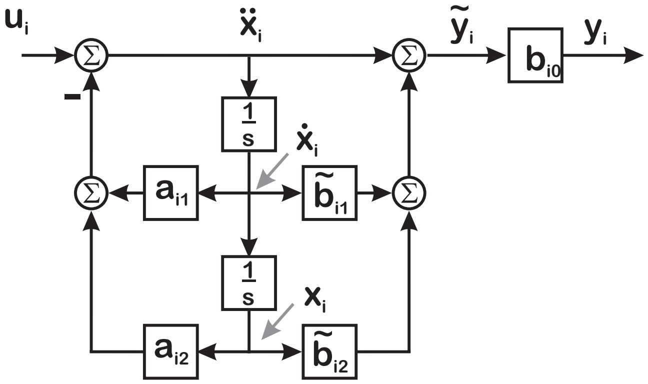


Figure 2.15: A block diagram of an analog biquad filter, which can implement Equation 2.21.

### 2.3.11 Some General Ideas

Why does this matter? At a zeroth order level, it is simply that systems which are essentially dominated by these models show up time and again in practical work. An engineer is looking at a problem and realizes, “Oh, this guy again!” Some of that is a self-fulfilling prophecy. Other engineers will design out/in features so that eventually the bulk of what the feedback controller is is one of these models.

1) Because each of these base models has properties that determine what kind of control can be easily and practically be designed and implemented for it.

2) Because we can take a simple parameterization of the most general PID model and apply it to these, we can see the effects of different tuning schemes.

## 2.4 The Filtering Framework versus the Feedback Framework

This section presents a discussion of filtering contexts versus feedback contexts. It is a recent addition [23], but once I got through finding a simple explanation, I found that it showed up again and again and explained many of the technical mis-communications to which I had been privy over the years. I will introduce it here, but it will show up again and again in this book. Understanding the difference between these two contexts is important for modeling and identification (Chapter 3), in how we apply

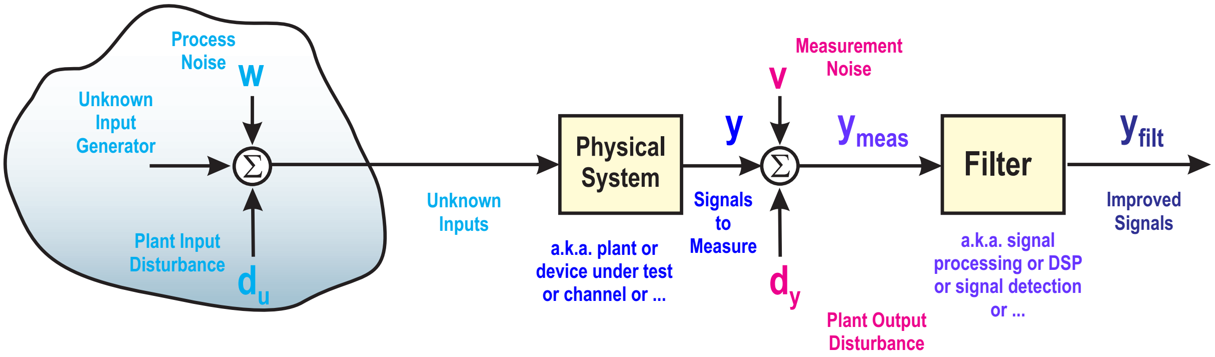


Figure 2.16: A filtering structure for looking at processes.

filters (Chapter 6), how we deal with input noise (Chapter 7), and how we think about computation for feedback control systems (Chapter 10). Even how one implements an estimator/observer (e.g. a Kalman Filter) changes dramatically with the change in these contexts (Chapter 9).

Filtering and feedback are akin to cousins that both fight and get along. Despite many similar features, there are core differences in the way the two frameworks approach similar looking problems. Understanding those differences helps us apply them towards business decisions.

A filtering framework/perspective is depicted in Figure 2.16. Somewhere beyond our direct access is a physical process generating an input. Perhaps this unknown input is passing through some physical process or channel that shapes it (and for which we may have some form of a model). That unknown input can be corrupted by noise and disturbances (what we would call process noise and plant input disturbances). While both noise and disturbances are forms of uncertainty, we define noise as driven by a random process which we cannot predict (but we can characterize). We consider disturbances a form of uncertainty that has some predictable, repetitive, or structural component in it, so that with the right algorithm and/or extra sensors it would be measurable. The key point of the filtering framework is that we do not have access to any of these signals as they flow into a physical system, process, or communication channel only the measured outputs. Those outputs are corrupted by uncertainty (what we in the controls community would call measurement noise and plant output disturbance). It is on this latter type of signal that the filtering framework applies.

The lack of access to the original physical signal (the reason why we need to do filtering in the first place) places fundamental limits on the modeling that can be done to generate a filter. Learning systems, such as an adaptive filter or supervised machine learning (ML), would require “ground truth” (known as the desired signal in adaptive signal processing [24]) to train the model or digital filter. Without this ML is limited to unsupervised methods, which are far more limited.

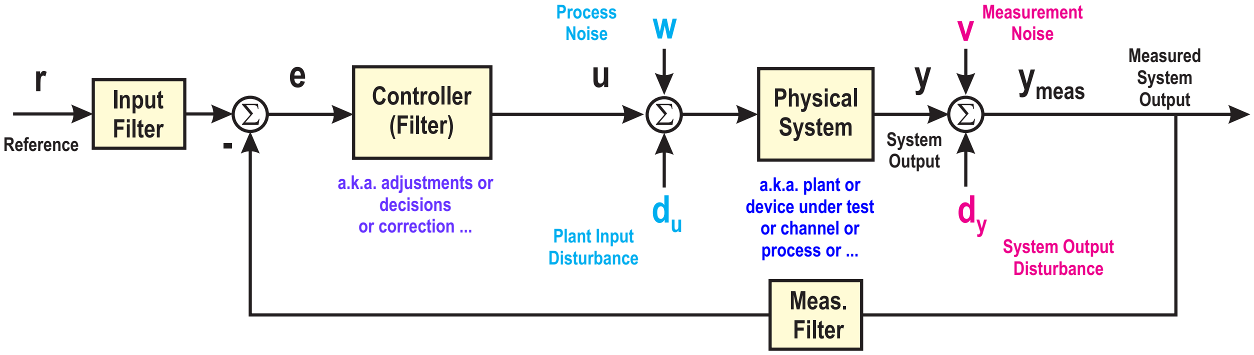


Figure 2.17: A feedback structure for physical processes.

The feedback framework, of Figure 2.17, considers access to the physical system input (at least some of them) as fundamental – else we cannot do feedback. The feedback framework allows us to affect the inputs to the physical process and so is in some ways far more relevant for decision making in business processes. After all, in business we would want to improve the process behavior by our adjustments. We still do filtering, but this can be in one or more locations including the feedback path (to filter measurement data), the input path (to filter the input commands or references) and in the controller itself (which is often implemented as yet another filter).

The filtering framework beckons as a simpler metaphor because it appears to have fewer perils. As we cannot affect the physical process, we do not take into account time delay or input disturbances. In fact, we must assume that the process is stable and relatively well behaved on its own, or there would be nothing to filter. We cannot get input-output behavior models from input-output measurements since we cannot generate inputs.

The filtering framework blinds us to the danger posed by latency, how a good correction done too late is a bad correction. It lacks the notion that measurement noise travels straight through to any corrected system output – a fundamental takeaway from that first controls class. Thus, without the feedback framework, decision makers would lack some fundamental intuition of how strongly measurement uncertainty limits their decision making.

The feedback framework tells us that we can use what we have measured to correct the process but a broader understanding would tell us that there are inputs to the process that we do not or cannot affect and outputs of the process that we do not or cannot measure. In this framework we would then be able to ask if more sensing and/or actuation is helpful in improving the behavior of the original process. On the other side, we could ask ourselves if there are outputs that – if ignored – can cause “Dark Data” problems [25]. Similarly, are there parts of the process that are far more easily adjusted and improved by adding another input to the system?

Even these intuitive arguments tell us that feedback is a far more complete framework than the filtering one. To be certain, as with physical systems, there are many processes that are so well behaved that only cleaned up observation and analysis a.k.a. filtered measurements are needed. To keep this interesting, we are focusing on those that could benefit from feedback.

## 2.5 Stuff Happens

We will see in later sections how the closed-loop responses of these low order models under the control of various forms of PID controllers (P, PI, PD, PID) and be predicted. However, if linear continuous-time analysis of simple models and simple controllers were sufficient to build great controllers in the real world, there would be no need for a book or a workshop on practical methods. We would simply apply the right theory, do some optimization, and real-world devices would work. Instead, we know that in the real world, “stuff” happens. Sampling, delay, and noise. (“Oh my!”) This section will point out how those turn this workshop into a two-day affair, crammed into one day. they are the reason this tome keeps needing new sections and new explanations. Without that stuff, good theory would be enough to generate good practical control systems.

### 2.5.1 Sampling

As mentioned in Section 2.1, we will assume that the control law will be implemented in a digital computer (or a computer for anyone born after 1980). By digital computer we may be discussing anything from a field-programmable gate array (FPGA) chip connected to the physical system with analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) , to MicroControllers, to digital signal processor (DSP) chips, to full processors, to Linux and Microsoft Windows systems. What we are excluding is the implementation of control laws via analog circuitry or analog computers (the “other” computers). While there are some clear sampling advantages to analog implementation, and these still exist at very high frequencies – where it is hard to sample fast enough and actually do control calculations between the samples – these are increasingly the domain of a small niche. An understanding of analog circuit design is extremely helpful in understanding the interface circuit that connect sensors to the ADCs and DACs to driver electronics and actuators. (Chapter 7 begins a discussion of some relevant analog circuits. Chapter 10 on computation discusses these in terms of signal chains into and out of the computer.

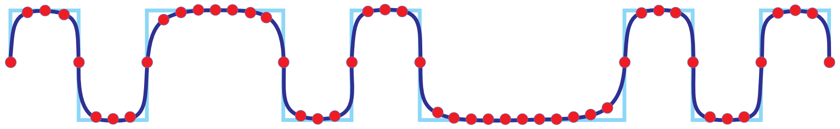


Figure 2.18: A diagram of real-time sampling of a signal. The ideal signal (cyan) has sharp transitions, but passage through the physical system often has a low-pass filter (LPF) nature to it resulting in a rounded curve (blue). The sampled signal (red) typically (but not always) is produced at regular sample intervals and with a certain level of quantization in the conversion. One can say that the real goal of a sampling system is to recover or infer the behavior of the ideal signal, but this simple diagram points out that even the physical filtering must be taken into account in any attempt to do so.

Realizing that most modern implementations are on digital computers, how do we discuss thinking in analog while implementing in digital? When is “sampling fast” enough and what insights can we gain from paying attention to how we discretize things?

The fact that controllers will be digital means that we should want a very physical, intuitive understanding of how sampling affects our perception of the physical signal (Figure 2.18), and how it limits what we can do with the controller. In particular, sampling adds delay to the system, delay results in negative phase, and negative phase reduces phase margin. Thus, the simple act of sampling the data limits our top end bandwidth before anything else can.

The second thing we must understand about sampling is that no one discretization model fully captures our ability to control a system digitally. The workhorse of most discretization in control is the zero-order hold (ZOH) equivalent [15], but even the most cursory look at this method reveals that all physical intuition about any system more complex than a double integrator is lost using this method. We will discuss how using other discretization methods may preserve physicality with a small (and often negligible) loss of mathematical exactness in the model.

A simple example of what is wrong with our understanding of discretization is seen in discretizing the double integrator of Section 2.3.5 and Figure 2.11. In the continuous time drawing on the right, we can easily pick off the velocity term, which makes using both position and velocity measurements available for feedback. Let’s remember that for a second order system, access to position and velocity for feedback is full state feedback, and this is the 800 pound gorilla of control theory – it puts the poles

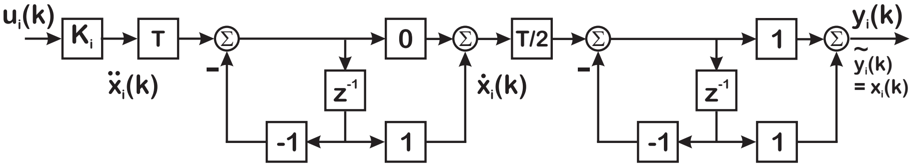


Figure 2.19: Discrete double integrator BLSS model (ZOH equivalent). In this drawing, we've chosen to make the index, i, as with a biquad stage, but we are explicitly labeling the different integration levels.

anywhere it wants. The situation gets much worse as soon as we discretize. If we use the standard Zero-Order Hold equivalent model and use a bilinear state space (BLSS) form (see Section 9.28) [5] shown in Figure 2.19, we can access the position, but the intermediate result does not do a good job of representing velocity, since the integrators are not interchangeable. It seems highly illogical to go to the trouble of generating a state-space realization on the basis that state-space gives us access to all the states and somehow lose access to velocity in one of the simplest state-space realizations available [5].

Consider the earlier example of the second order resonance, of Equations 2.18 and 2.19:

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}} = \frac{K\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}, \tag{2.22}$$

or to the more general analog biquad model:

$$\ddot{x} = -a_1\dot{x} - a_2 x + u \tag{2.23}$$

$$y = b_0\ddot{x} + b_1\dot{x} + b_2, \tag{2.24}$$

with the transfer function description:

$$\begin{gathered}\left(s^2 + a_1 s + a_2\right) = U(s)\\ Y(s) = \left(b_0 s^2 + b_1 s + b_2\right),\end{gathered} \tag{2.25}$$

yielding

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}. \tag{2.26}$$

A discrete transfer function version of (2.26)

$$\frac{Y(z)}{U(z)} = \frac{b_{0,D}z^2 + b_{1,D}z + b_{2,D}}{z^2 + a_{1,D}z + a_{2,D}} = \frac{b_{0,D} + b_{1,D}z^{-1} + b_{2,D}z^{-2}}{1 + a_{1,D}z^{-1} + a_{2,D}z^{-2}}. \tag{2.27}$$

The question is what the meaning of the new coefficients in relation to the old ones, and this is entirely related to both the original parameters and the discretization method. Exact discretization methods obscure the coefficient meaning and couple states in a very non-intuitive way. Some other approximations, in which the original transfer function is broken into a cascade of second order sections (biquads) can preserve much physical intuition.

There are lots of discretization methods and even when one does the "exact" math, one doesn't get a satisfying answer. In fact, the exact math can give an answer that is so convoluted as to obscure any hope of physical intuition and this is bad. The Trapezoidal Rule, also known as Tustin's Rule or a bilinear equivalent[15], substitutes discrete time operators (based on the Z transform) for the continuous time operator (based on the Laplace transform). Using the Trapezoidal Rule, we make the substitution:

$$s \longleftarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) \tag{2.28}$$

and if we substitute for $s$ in (2.26) to get to (2.27) then we end up with the following mappings:

$$\begin{array}{lcllcl} \Delta & = & 1 + a_1 \frac{T}{2} + a_2 \frac{T^2}{4} & & & \\ b_{0,D} & = & \frac{1}{\Delta}\left(b_0 + b_1 \frac{T}{2} + b_2 \frac{T^2}{4}\right) & a_{0,D} & = & 1 \\ b_{1,D} & = & \frac{2}{\Delta}\left(b_2 \frac{T^2}{4} - b_0\right) & a_{1,D} & = & \frac{2}{\Delta}\left(a_2 \frac{T^2}{4} - 1\right) \\ b_{2,D} & = & \frac{1}{\Delta}\left(b_0 - b_1 \frac{T}{2} + b_2 \frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1 - a_1 \frac{T}{2} + a_2 \frac{T^2}{4}\right) \end{array} \tag{2.29}$$

For the simple spring-mass-damper system of (2.22), we end up with

$$\begin{array}{lcllcl} \Delta & = & 1 + \frac{b}{m}\frac{T}{2} + \frac{k}{m}\frac{T^2}{4} & & & \\ b_{0,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{0,D} & = & 1 \\ b_{1,D} & = & \frac{2}{\Delta}\left(\frac{2}{m}\frac{T^2}{4}\right) & a_{1,D} & = & \frac{2}{\Delta}\left(\frac{k}{m}\frac{T^2}{4} - 1\right) \\ b_{2,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1 - \frac{b}{m}\frac{T}{2} + \frac{k}{m}\frac{T^2}{4}\right) \end{array} \tag{2.30}$$

The point of the discussion is: if it looks complicated, it's supposed to be. The physical parameters get lost in the shuffle a lot and we need to fight to keep them in terms that are meaningful in our discrete time model. The $b$, $k$, and $m$ parameters are spread all over Equation 2.30. This also spells bad news for trying to extract physical parameters from time domain ID with discrete time models. The accuracy to which we need to identify the coefficients in Equation 2.27 in order to back out the physical coefficients using Equation 2.29 is tremendous. The problems get worse when the system order gets higher.

What I've come to realize is that breaking the problem down into blocks and discretizing the blocks makes a lot of sense in the sense that each block has it's own discretization error, but it also preserves the physical meaning of the original block (if you do it right, which still isn't trivial).

### 2.5.2 Delay

The discussion of sampling brings into focus a discussion of time delay, and an understanding of time delay should be part of any control system implementation. After all, time delay manifests itself as a pure negative phase in the frequency domain. That is for a delay, $T_D$, the Fourier transform pair [26] is:

$$f(t - T_D) \supset F(s)e^{-j2\pi f T_D}, \tag{2.31}$$

which means once again that even if everything else is done perfectly, the time delay will eventually drive the open-loop phase below $-180^\circ$ at some frequency, $f = f_{lim}$.

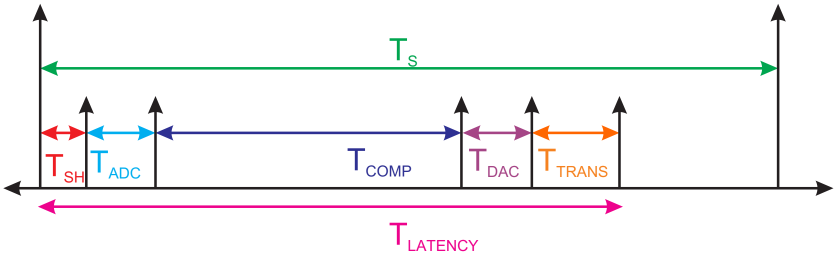


Figure 2.20: A diagram of delays in computing a control signal for a sampled data system.

We will talk about time delay much more extensively in Section 5.6 of our chapter on loop shaping and in Section 10.5 of our chapter on computation.

While we think about time delay from sampling, where typically one assumes an average delay of:

$$T_D \approx \frac{T_S}{2}, \tag{2.32}$$

and $T_S = 1/f_S$ is the sample period, there are many other components of delay in the system. One can consider the delay from the time that a signal is sampled until a control signal responding to that signal is produced, as diagrammed in Figure 2.20. Even if the sampling delay was $0$ there would be physical transport delays in a system, the time it takes for fluid to flow from an actuator value to the flow sensor, the time that it takes for the effects of a heating coil to be sensed at a thermocouple that is displaced from the heater, the time for ions to travel from the source of a mass spectrometer to the detector, or the time for packets to be transmitted across a network, just to name a few.

As humans, we like to simplify, and this often means that we focus on one dominant delay. For fast mechatronic systems such as atomic force microscopes (AFMs, [27]), the computational delay diagrammed in Figure 2.20. For chemical process control [20, 28], the physical delays swamp the computational delays allowing for a slower sample rate (longer $T_S$) and therefore making the relative effect of $T_{LATENCY}$ in the diagram.

### 2.5.3 Time Constants

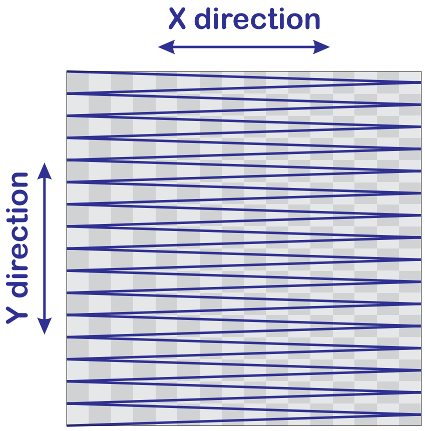
Both discussions of sampling and delay bring up the discussion of time constants. Generally, the term time constant is associated with the solution to a first order, linear differential equation,

$$\dot{t} = at = \frac{1}{\tau}t, \tag{2.33}$$

where $\tau$ is called the time constant and is the time for an initial condition to decay to $e^{-1}$ of it initial condition value, $V(t) = V_0 e^{-1}$. In common usage though, it is generally considered the amount of time it takes for some response to really start rolling off. In this context, time constants can be thought of a how much time it takes for a signal to get through a system, or alternately, how quickly or slowly a system can respond to sudden changes in input. Time constants then can tell us how relatively important our delay is. After all, if the constants of the system are super slow relative to the sampling delay, then the sampling delay can be ignored. (This is also known as “sampling fast”, but really should be fast relative to the time constants in question.) Likewise, transport delay only matters relative to the time constants of the system in question.

One of the reasons that control engineers working in chemical process control (CPC) can use techniques like model predictive control (MPC) [29] is that their system time constants are incredibly slow, when compared to high speed electronics or mechatronics. In particular, almost any real-time processing system is so much faster than the slow system time constants that a lot of complex algorithms can be run in between samples. In fast systems, such as atomic force microscopes (AFMs) [27] or phase-locked loops (PLLs) [30] (used in communication, clocking, and synchronization), current computation technology is not fast enough to allow that.

We often don’t consider what is obvious in retrospect: that a lot of our frequencies come from the movements of the system. Consider the movements of an AFM [31], generated by a raster scan to



| Rate | Definition |
|---|---|
| sample rate | frequency of data sampling |
| scan rate | frequency of 1 line scan in each direction |
| frame rate | scan rate/lines per frame |
| pixel rate | scan rate · pixels per line |
| samples per pixel | sample rate/pixel rate |
| scan speed | scan line width ($\mu$m/line) · scan rate (lines/s) |

Figure 2.21: A raster scan generates the initial movements in an AFM. The blue line represents the scan, which is generally faster in one direction – commonly denoted as the X axis. The checkerboard pattern underneath schematically shows pixels that might be formed from such a scan pattern. Each direction of scanning along the X axis is used to form a separate image. For each scan, the sampling of data along the Y axis will be considerably slower than the sampling of data along the X axis. Typically, some sort of decimation or averaging is done to reduce the X data samples to the pixel rate in the X direction.

create an image. Figure 2.21 shows the X-Y scanning of a sample area that is typical in generating an image. On the right of the figure is a table that shows the different rates in the system. One of the areas of potential confusion is that any one of the rates listed can be what any engineer – or more frighteningly, any marketing person – calls the "rate". If one makes certain assumptions about the control design, including the phase margin, the scan size, the number of pixels, and the width of a line, one can assess the desired closed-loop bandwidth and acceptable system latency. This then allows us to back out the needed sample rate and electronics switching speed.

Try to find original calculations as they are instructive.

We will find that it is useful to paraphrase Sun Tzu [32] here, "Know your time constants, and know your dynamics, and you can close 100 loops without disaster."

### 2.5.4 Nonlinearities

All of these can create issues for making a system work, but as they say in late night commercials, "Wait! There's more!" This is because every real system has nonlinearities, and these nonlinearities often make the quantization we see in the ADCs and DACs look like small potatoes. Often this describes slew rate limits on flow, or simply a valve being able to only open to 100% and close to 0% flow (rather than towards $+\infty$ or $-\infty$). The thing about nonlinearities is that they break our linear analysis tools and so the question is not about whether a system has nonlinearities or is nonlinear, but really about how nonlinear a system actually is. Does this account for 90% of the response or 0.9%? Is a system governed by different equations in one region of operation than in another? The ability to detect these conditions and change the controller's behavior is one of the great driving forces behind using computer control. Of course, we still have to build that.

Nonlinearities take the form of limiting nonlinearities which limit the range of some variable (e.g. saturation, signum (sign) function, sigmoid function) and distorting functions which alter the input-output relationship (e.g. square root, square, signal harmonic, or quantization).

The saturation nonlinearity is one of the most common, showing up all along the signal chain, from limits on valves to limits on voltages, to limits on the size of signals in fixed point computation. How much of a signal's nominal response exists in between the limits? What does the rest of the system do once those limits are hit.

- In double integrator problems, the effects of saturation do not affect the reachability of any state and therefore has little effect on closed-loop stability. However, it is the saturation that sets up the time optimal, bang-bang control solution.

- For some systems, the effective gain reduction due to saturation can lead to instability. One of the simplest systems that display this is the triple integrator where for small enough feedback gain, the root locus reveals closed-loop poles in the right half plane. Thus, for a large enough input signal, the gain reduction due to saturation results in instability., The details depend upon the specific feedback compensator and on the system. In the case of an unstable system, the saturation means that for certain large deviations, the system might not have enough input range to get back to a zero state.

- Saturation also creates problems with integrators, which end up accumulating error signals. When the system comes out of saturation, the accumulated error signal in the integrator may be way out of proportion with the actual error in the system. This phenomenon is called wind-up and is discussed in some depth in Section 4.18 of the chapter on simple controllers and PID tuning.

- One of the most nefarious and hardest to debug locations for saturation is in notch/anti-notch filters. A notch can be thought of working by creating a canceling signal that matches the input signal at a particular frequency of interest. If that input signal is saturated coming into the notch, the generated canceling signal will also be clipped, resulting in a notch that doesn't really work.
- In fixed point math, not only can signals be clipped, but also filter coefficients. Returning to our notch example, it turns out that for high Q filters, the limits on coefficients can take stable, minimum-phase filters and make them unstable and/or non-minimum phase [33].

Lest we forget ADCs and DACs provide quantization nonlinearities in each digital control system. While we are accustomed to our double precision floating point numbers, ADCs and DACs have fixed quantization with the quantization level inversely related to the sample period and the chip cost. The fastest ADCs are the 8-bit converters used in multi-Gigahertz scopes (cite some Keysight,Techtronix scopes). The highest accuracy in products in broad distribution in the market as of this writing is around 24 bits. Typical quantization levels in control applications are typically between 12 – 18 bits.

### 2.5.5 Noise

Finally, there is noise, which is a term used to describe anything from shot noise to biases due to cables or friction. Noise can be easy to model (e.g additive white Gaussian noise) and not describe what we are observing or may be perfectly descriptive and hard to handle analytically. As with all of the above, it places a limit on what information we can cleanly extract from a system, either in our attempts to model the system or to properly control it.

Less common is the understanding of the effects of noise through the feedback loop, or how to back noise measurements out to their sources. In the legendary "Respect the Unstable" Bode Lecture of 1989 [1] Gunter Stein educated us to the idea that loops do not eliminate noise, they merely move it around, as he brilliantly illustrated with a dirt digging problem, reconstructed from memory on the right side of Figure 2.22.

If one uses that as a starting point and works backwards, one can establish what the "input noise" must have looked like, as shown in Figure 2.22. This became the basis for the PES Pareto methodology of

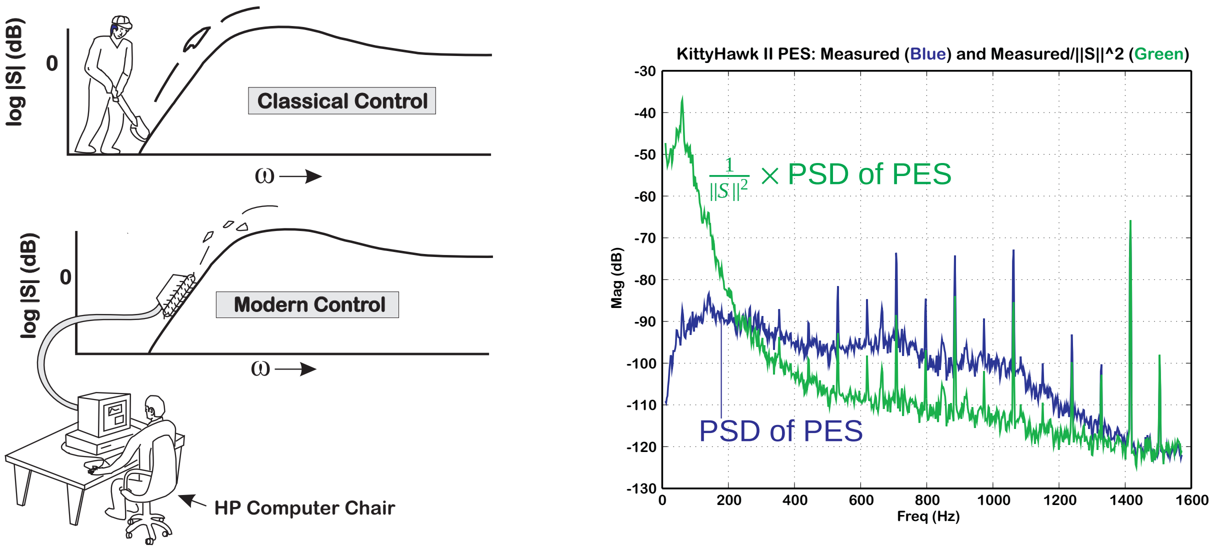


Figure 2.22: On the left, Gunter Stein's dirt digging analogy, recreated from memory circa 1994. On the right, KittyHawk 1.3" disk drive: PSD of PES, and PSD of PES filtered by $\frac{1}{\|S\|^2}$.

analyzing the effects of noise on a system [34, 35, 36, 37]. This divide and conquer analysis method is one that pays many dividends because we go after the "right" noises that have the most affect on the system. This is one of the main topics of Chapter 7.

Even then, when we are trying to make practical measurements, we must default to some idealizations. The Widrow model [38] of quantization, it assumes quantization error as uniform white noise on the interval, $[-q/2, q/2]$. To analyze noise through a linear filter requires an assumption of additive, Gaussian, white noise. We step away from analytical purity and mathematical exactitude to be able to gain understanding. That is, in PES Pareto, we might start with the Widrow quantization model and mumble a few incantations before setting the uniform variance of $q^2/12 = \sigma^2$ in an additive, white, Gaussian noise (AWGN) distribution.

We will return to methods for analyzing and minimizing the effects of noise, because as will be repeated many times in this work, even in an LTI system, with perfect modeling and digital compensation, eventually noise and delay our our fundamental limiting factors.

## 2.6 Skill Sets

Beyond a technical risk, there is an organizational risk posed by the necessity to involve multiple designers with varied skill sets int he design. If one walks around a typical digital control loop of Figure 2.2, one sees that each block brings with it a particular set of skill requirements.

**Digital Computer:** In order to do control, we will need to do digital control. Besides the obvious needs to understand some digital control theory, we also need to program the computer to implement the control law. This involves understanding of the physical system time constants and the power of computation we have available. Can we run in a multi-tasking operating system or do we need to be close to a pure processor or FPGA solution? In which language will we program? What levels of abstraction do we need? What do we gain and what do we give up when we let the advanced tool handle all these abstractions?

**Physical System:** A fictional engineer once made the epic statement to his commanding officer, "I can't defeat the laws of physics." In fact no one and no algorithm can defeat the laws of physics. It's akin to a surfer claiming they can change the shape of a big wave with their board. Instead, good algorithms ride physics of the problem. They read the fine print of the laws of science. They understand the fracture points of the different problems. This is the key to difficult problems and why one might approach "magic algorithms" which claim that you don't need to know anything about the system with a load of skepticism.

The first step to doing any control is to have some understanding of the physical system. That requires domain expertise in the system to be controlled. It is a sad, but true case that physical systems do not come with their own third order analytical models. Perhaps we can fix that after the next Big Bang, but until then we are stuck trying to model physical systems from a combination of first principles (science), measurements, and simulation. Even measurements require some system knowledge to know what must be measured and what one can hope to understand.

**Sensor:** The sensors need to interface with the physical system to return an analog of the physical property being measured. As such, there is a lot of science tied to circuitry tied to machine design.

**Actuators:** On the other side of sensors are the actuators that tie electrical signals into physical action. Again, they involve a combination of machine design, science, and signal knowledge.

**Analog Filters:** Most conversion circuits require some sort of signal filtering to minimize noise entering the loop. Before the signal can go into an ADC or go from a DAC to the physical system, we want to clean up the edges, remove noise spikes, notch out harmonics, etc. These are done with analog circuits and the art of analog design is almost a rarity these days. What are the phase effects of different low-pass or anti-alias filters? Do power line harmonics need to be removed? All of these require some knowledge of analog design.

**ADCs and DACs:** The choice of signal converters is rarely simply about picking the number of bits, or we would simply pick the converters with the highest resolution. Instead, we must contend with delays, both in the interface between converter and computer (parallel or a variety of serial formats) as well as delays in the conversion method (SAR, pipelined, etc.).

**Power Electronics:** Before we can send a DAC signal to actuate a backhoe arm, we need some power electronics to scale things up. Before that engine temperature can be tracked, it must pass through a thermocouple. Signals in kilovolts need to be scaled down to that of the ADC, typically under 5V these days. For the electrical grid to get smart, 3-phase measurements at high voltage levels need to be made. All of these require a knowledge of power electronics, which are often not taught in many collegiate Electrical Engineering curricula anymore.

**Control Design:** At the end, we come back to control design, but control design in the context of these other factors. In the real world, optimality sucks [19]. That is, optimality, which can only exist in the context of an abstract, usually simplified, mathematical model, cannot exist in a real world system where all that stuff is happening. However, our idealized, simplified models can be used to tell us the best we can do, even in the real world. Guided by understanding how to apply our theory and optimization to real world systems, we can't get to optimal control, but we can get to excellent control. Or to be simplified to a bumper sticker, Optimality sucks . . . but excellence rocks[19].

## 2.7 Introduction Summary

To quote Winston Churchill [39], “Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.” This introduction is intended to motivate the dives in the sections to come. The difference between an idealized system in analysis and a real working system involves addressing these issues, extracting a workable system model in the face of sampling, time delays, nonlinearities, and noise, then applying that model in a way that does not violate the limitations imposed by those issues.

The rest of this book's chapters will be organized as follows.

- **System Models and Characterizing Them with Measurements:** We will first delve into that fundamental need for applying advanced control methods to physical systems: how to extract usable models from available measurements. For these, and particularly when only step response methods are available, we will find that we need to assume something akin to one of the models described in Section 2.3. Using ideas from Section 2.5.1, we will discuss the difficulties and limitations of trying to use discrete-time, time domain identification on models for which we want to extract physical understanding. (Chapter 3.)

- **Simple Controllers for Simple Models (or why so many controllers are PIDs):** A dual to extracting a simple model is understanding what can be done with simple control schemes. Thus, we will put PIDs into a unified framework [40] and then describe what we can tell about these controller for our canonical simple models of Section 2.3.

- These two segments will provide some intuition into what we can expect from simple models and controllers, and they will form a backdrop for the other aspects of practical systems. In particular, they can be seen as the ideal that we are trying to get back to when other system features interfere with our plans.

- **Practical Loop Design, Or Why Most Open Loops Should Be an Integrator, and How to Get There:** We will discuss a relatively simple design objective, where it shows up without much discussion in many engineering problems, and what we have to do to apply it in practical problems. We will also discuss the ramifications of loop shaping, as explained in Gunter Stein's explanation of Bode's Integral Theorem [1]. (Chapter 5.)

- **Resonances, Anti-Resonances, Filtering, and Equalization:** As Maarten Steinbuch writes on the board in his control systems class, mechatronic systems are "-2 plus stuff." (The word, ”stuff” is a diplomatic substitution here, just as in the expression, ”Stuff happens.”) For flexible, mechatronic systems, that stuff consists of one or more high Q resonances and anti-resonances. If one wishes to not be limited by these, we must understand practical application of filters for equalization. (Chapter 6.)

- **Signal Detection, Sensors, Sample Rates, and Noise (Oh My):** When everything else is said and done, we can't do feedback control without sensing the data and how quickly and cleanly we can do this determines a lot of what any algorithm can do in a control system. (Sensor noise goes right through.) Thus, it's worth understanding sensors, sample rates, and noise propagation through the system. Once we have understood how to relate measured noise to an input, we will look at filtering and demodulation methods to minimize it before it enters the loop. (Chapter 7.)

- **Integrating in Feedforward Control:** As the Cheshire Cat pointed out to Alice [41], it's a lot easier to get where we want to go if we actually know where we want to go. In control, that is called feedforward. Sometimes this is not available to us, but how to we make use of it when it is? (Chapter 8.)

- **State Space: The Good, the Bad, and the Practical:** State space control is model based control and model based control requires a . . . model. If we've kept up to this point, we may very well believe that the greatest limiting factor in using state space control on physical systems is our ability (or not) to extract accurate models of the actual system and to understand the system based limitations of the accuracy of those models. We will try to develop a mental checklist of the things we need to get to so that we can do what the cool kids do. (Chapter 9.)
- **Real-Time Computing Issues for Control Systems:** Finally, while Silicon is cheap, finding the right Silicon takes some investment. We can only scratch the surface here, but this section will give the listener a start on understanding a three layer computation model for real-time systems and how to make use of this in the chips we may chose to use for our control systems. (Chapter 10.)
- And we will have some **Closing Thoughts**, because every workshop/book/paper should have something like that, and "conclusions" seemed like a bad name. This will be in Chapter 11. I suppose it could have had something about putting the philosophy back into Ph.D.

# Chapter 3

# System Models and Characterizing Them with Measurements

## 3.1 In This Chapter

This chapter returns to the idea of modeling dynamic systems and how to accurately parameterize these using measurements. We will primarily stick with models one might use for controller design and/or for construction of a state-space model.

The literature tends to break down models from either “first principles” (a.k.a. science) or data centric. We will argue that this is a false dichotomy. (Translation: I think it’s some BS.) A great explanation for this view was stated by Stephen Hawking at the start of A Brief History of Time [7]:

> I shall take the simple minded view that a theory is just a model of the universe, or a restricted part of it, and a set of rules that relate quantities in the model to observations that we make. It exists only in our minds and does not have any other reality (whatever that might mean). A theory is a good theory if it satisfies two requirements. It must accurately describe a large class of observations on the basis of a model that contains only a few arbitrary elements, and it must make definite predictions about the results of future observations.

**The point of a theory is to describe measurements in a concise way and then let us use that compressed representation to predict what we might measure later on.**

We also need to understand what we can and cannot measure. Our inability to look at all signals at infinite sample rate with infinite precision – it sucks to be us – means that we must discern models and their parameters from measurements limited by what the system allows. Again, there is a disconnect if we simply think we can make measurements outside the context of any model at all. James Burke offered the view in The Day the Universe Changed [6]:

> Science, therefore . . . is not objective and impartial, since every observation it makes of nature is impregnated theory. Nature is so complex and so random that it can only be approached with a systematic tool that presupposes certain facts about it. Without such a pattern it would be impossible to find an answer to questions even as simple as 'What am I looking at?'

Even with all these constraints, Moore's law [42] allows us to do much more than we might think. The thing is – as in all other chapters in this book – there is a disconnect between what the official theory teaches us and what works in many control systems.

We will start with discussing the discrete-time linear model regression taught in many system identification classes and describe which problems for which these methods might work well, and when they might have very limited success.

We will then divert into problems for which step response methods can work well. Though there are severe limits on how many parameters can be extracted using step response methods, we will see that we can use them to gain some information from most systems.

Finally, we will branch into frequency response methods. While there are some restrictions on which types of systems can be measured with these methods, they still provide a wealth of information about those systems that allow us to extract parameters.

We want to keep in mind that extracting a model and its parameters from data is an exercise in data compression. If successful we are able to represent a huge amount of data in a relatively small number of parameters. Like all data compression problems, model regression takes a large amount of data to reliably extract a model and its parameters – assuming one exists at all. In each of the methods above, there is a dance between measurements and models. Which models apply to which systems, which measurements are useful for which models, and which models allow for which measurements. It is important to keep in mind that measurements without model lack any context from which to draw insight. There are simply a set of phenomenological descriptions. At the same time, models without measurements are pure conjecture, and as such disconnected from reality. Once again, our goal is to meet somewhere in the middle.

## 3.2 Chapter Ethos

“Measurements, filtering, and noise, oh my!” For any physical control system, measurements are made to characterize noise in the system, to characterize the physical system, and/or to characterize the system’s performance. Likewise, all physical systems have behaviors that cannot be fully accounted for by a tractable model. These are often called unmodeled dynamics when the they follow some pattern and noise when they do not. Because all measurements are affected by noise and unmodeled dynamics, we need to consider this when we make measurements to characterize the systems we are trying to control.

In this chapter, we will talk about system modeling and how to extract models of our physical systems from measurements that we can make on them. We cannot even start this without discussing discretization and discrete models of physical systems. This is not the standard textbook approach of, “We start with a continuous-time transfer function in $s$ and end up with a discrete-time one in $z$. They both have coefficients.” In our case, for many of our physical systems, the journey of the physical parameters into those discrete transfer functions or state-space realizations actually matters.

Once we have discouraged everyone properly, we will ask what measurements can be made to identify discrete-time models, and how those might be mapped back to physical parameters. From there, we will take a different tack; found often in industry, but not often unified. We will return to our simple second order models models and ask what parameters can be extracted from step response measurements. One of the mental changes in this is that we will be extracting physical parameters of continuous-time models from discrete-time measurements. Finally, we will turn our attention to something that is far more common in mechatronic control systems than in chemical process control or biological control systems: frequency response methods.

Several issues are worth highlighting from a top level view. The first of these are the fundamental questions of what constitutes a good model and how good does that model have to be to meet our needs? (While we are at it, do we really understand what we want and what we need from our system?) There are models that describe nature in great detail but cannot give us much insight, much view into the fundamental structure of things. There are models that are supremely accurate at capturing the behavior of a physical system, but are useless for generating a control or filter design. For our purposes, we are looking for models that both capture the essential behavior of our physical system and allow us to do something with that information. Preferably, this involves an improved control design. Still, one might ask how closely this model has to match the system behavior to be useful in a feedback design. When do we need to capture 99.9% of the behavior, and when is 95%, 90%, 80%, or even 60% of the physical system behavior sufficient?

Honestly, the general answer to this question is more philosophy than hard science. However, there does seem to be an inverse relationship between how cleanly and rapidly one can measure signals and apply feedback for linear time-invariant (LTI) systems): and how close the model has to be. Two factors show up again and again (at least the sharpness of the resonances and other dynamic features (but mostly resonances) and the relative speed of the controller to the highest-frequency significant dynamic feature of the physical system. For digital controllers (and most practical controllers) this ties directly to the sample rate and the latency of the controller.

## 3.3 System Models and Measurements: Introduction

In 2008, Karl Åström invited me to give a talk to his advanced controls class at the University of California at Santa Barbara (UCSB) where he spent many winters. I generated the talk that became “A Tale of Three Actuators” [31]. At the end of the talk, one graduate student (GS below) seemed pretty upset and asked, “So, you’re telling us that all this state-space stuff we’ve been learning all these years is useless!?!” My internal monologue was going wild.

Me: (“Think, Danny! Think!”) I started with “Well, um, state-space methods are . . . model based methods, right?”

GS: Gives a distrustful head nod, and mutters, “Yeah.”

Me: “And model-based methods need a good model, right?”

GS: Gives a distrustful head nod, and mutters, “Yeah.”

Me: (Now, if you are old enough to remember the movie, Animal House, this is essentially my version of the moment when Bluto asks, ‘Was it over when the Germans bombed Pearl Harbor?’ and Boon wonders ‘Germans?’ and Otter responds, ‘Forget it, he’s rolling.’ At this point, I’m rolling.) “Well, to tell the truth, most of the models you all use . . . suck. You assume simple models for analysis but in the real world I by the time I get to that simple model, I’m 85% of the way home. Getting models from measurements of real-world systems, especially lightly damped mechatronic systems, is really, really hard and so it happens a lot less in industry than it happens in academia.”

It’s been over a decade since that conversation, and so I may not remember the details, but the essence of it is captured in what is above. The graduate student was not wrong to be frustrated. I recall the same frustration trying to apply all the things I thought I knew coming out of graduate school to optical and magnetic disk drives [19]. What that student had done with his insightful question was forced me to state in a relatively brief way, what was one of the main disconnects between academic and practical applications in the field of control.

**The painful truth is, most people optimize the crap out of lousy models.**

Model-based methods require a good model. We can go from first principles, but at some point we need to correlate those physical models with measurements. Ideally, this would be simple, but unfortunately (or fortunately depending upon the employment situation) there are issues.

The first is that while we can apply any measurement we want to a simulation model, different physical world systems allow only certain measurements. Furthermore, nonlinearities, quantization, noise, time variation, data memory limits, all mean that no one measurement can give universal information about a system.

The second issue is that for all intents and purposes, our measurements come from discrete-time measurements of a physical system. Setting aside quantization, saturation, and time-delay, we are either applying discrete measurements to identify a discrete-time model (and either doing direct control design from that identified model or converting that model to a continuous-time model) or using the discrete-time measurements to approximate continuous signals and applying these to identify a continuous-time model. The former is by far the most in vogue method taught in academia, but as the examples from the Introduction (Chapter 2) show, the discrete-time model obscures the physical parameters of the system. Much of how identification and control design has been taught since the 1960s has involved working with the discrete models and not worrying about the loss of physical model parameters. My experience in the past 30 years has shown that for control of physical systems, one has the best shot of success if one understand the physical parameters. Not only are those the places where models change, but also having “physicality” in our controller designs allows us to debug our design in a far more intuitive way.

There is a huge literature of time domain identification of discrete-time models. Most involve some sort of least squares or gradient method to match a moving-average (MA) (a.k.a. a finite impulse response (FIR) filter ) [24, 43] or auto-regressive, moving-average (ARMA) (a.k.a. an infinite impulse response (IIR) filter ) [44, 45, 46, 47, 48, 49] to input/output data of the physical system. Of course, things get more sophisticated with attempts to model how the noise is shaped, but for the point I am trying to get across here, the simple model will do.

Most of this involves some version of a model:

$$y(k) = \theta^T \phi = [b_0, b_1, b_2, \ldots, b_N, a_1, a_2, \ldots, a_N] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \ldots \\ u_{k-N} \\ -y_{k-1} \\ -y_{k-2} \\ \ldots \\ -y_{k-N} \end{bmatrix}, \tag{3.1}$$

for the system and an attempt to estimate the unknown $\theta$ parameters. This would correspond to a transfer function in $z^{-1}$ of

$$\frac{Y(z)}{U(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots a_N z^{-N}} \tag{3.2}$$

The key mental break between the academic/theoretical (AT) models and what is done in practice is that identification of an unknown $\theta$ vector might be fine in theory, but is far less common in practice. In practice, time and again, we see attempts to work with physical parameters. Some of the reason behind this should become apparent in the section of this chapter on discretization 3.6: most of the methods obscure the physical parameters so much that it is hard to localize actual physical parameter uncertainty on a few coefficients of the discrete-time model.

Too be clear, there are plenty of examples of these methods working. If this were not the case, far fewer books on adaptive methods would be published. It appears that if one looks closely at these examples, they include:

- Adaptive filtering as popularized by the classic book, **Adaptive Filtering**, by Bernard Widrow and Sam Stearns [24]. This book has been on my shelf since graduate school, when it was in draft form. If one looks at the problems described here, they are filtering problems, where time delay does not matter, and where the thing to be filtered is stable. Furthermore, all the problems described there essentially have a low pass behavior, well described by a discrete transfer function. The methods in the book describe using an FIR) (transversal) filters to approximate these discrete transfer functions and then using the least-mean-squares (LMS) adaptive algorithm to search for the best fit. Now, this works in a lot of situations, and it helps that latency and stability are not issues when one isn't closing the loop. However, from a modeling perspective, it seems that the fits work best (and with the shortest FIR models) when the system being modeled is low order, and/or well damped, and certainly stable.
- Rigid body systems, such as spacecraft. In such systems, there are few flexible modes and we are really looking at adapting to constants of a lot of forms of double integrators.
- Chemical process control systems, which are often modeled as first order, second order, or first order plus time delay (FOPTD) These are alternately known as first order plus dead time (FOPDT) .
- Ship steering problems. Need I point out that these systems are slow, the resonances are well damped and at extremely low frequency relative to any sample rate, and so the modeling, while possibly multi-input, multi-output (MIMO) does not involve lightly damped or unstable modes.

The common themes here are that any dynamics are well damped, that the sampling is far faster than any dynamics (e.g. 20–1000 times the fastest dynamic), and that when one is dealing with higher order systems, it seems to matter little which physical feature/parameter is the one that is unknown or changing.

It is when the system exhibits more exciting dynamics, e.g. instabilities or simply features with very low damping, that these methods seem to hit severe limits. We will see that in these cases, practicing engineers have moved away from the methods above to step and frequency response methods. That will be the focus of this chapter. At some later point in time, a later chapter will delve into making adaptive methods practical.

Measurements made to characterize a system can be delineated by several choices depending upon what is available in the physical system:

- Time domain versus frequency domain: This is a bit deceptive because measurements must be made in the operational (or time) domain. However, some are only displayed after being transformed into the frequency domain. This is true of both spectrum measurements (where only the output data is measured and transformed) and network measurements (where the input and output data are measured and transformed).

- Operational data versus special test data: Operational data measurements rely solely on data that would be present during normal system operation. Some systems have this limitation, while others allow for specially generated signals to be used that allow for more dynamics to be identified. Examples of the latter include:
  - impulsive inputs,
  - step inputs, although these are more likely square waves with a half-period longer than the essential dynamics of the system (Section 3.11),
  - pseudo-random inputs,
  - chirped sinusoidal inputs,
  - stepped-sine inputs (also called swept-sine in industry), and
  - multi-sine inputs.
- Signal extraction methods, especially with respect to frequency domain measurements: Are broadband methods such as fast Fourier transforms (FFTs) used? What are the benefits and downsides of coherent demodulation?
- Which of these are available often depends upon the type of system under test. Chirped sinusoids may be easy to apply to mechatronic systems, but may be completely impractical for a characterizing a chemical process control (CPC) problem.

Likewise, a variety of instruments may be available to accomplish some very similar tasks, including:

- Digital Oscilloscopes: These were also called digital storage oscilloscopes (DSO) in the past and are generally called digital scopes today. They can be viewed as an instrument to capture time traces of sampled data. Over the years, the sample rates and storage capabilities of these have expanded so much that they can do much of the processing once found in spectrum analyzers, especially simple FFTs of the captured data. Note that the scaling of the digital scope FFT is likely different from the scaling of say an FFT from Matlab or NumPy in Python.
- Spectrum Analyzers: These are instruments that capture time data and then compute a Fourier transform type of calculation on it, to reveal the frequency domain content of the signal over

a specified frequency range. Old instruments accomplished this by slowly scanning a narrowband filter (often a Gaussian filter) across frequencies and capturing the filter output at that frequency. Modern spectrum analyzers compute an FFT of the data, but this functionality is also now part of modern digital oscilloscopes. What are currently produced under the title spectrum analyzers now can analyze a lot of digital patterns.

- Network Analyzers: These are spectrum analyzers that can also inject test signals into the device under test (DUT) . By injecting test data, the idea is to stimulate particular elements of the system and so as to be able to analyze the input-output behavior of the system. The resulting measurement can be captured in time, but classically these instruments are looking to measure frequency response functions (FRFs) , often known as empirical transfer function estimates (ETFEs) in the academic literature. Network analyzers were particularly popular for analyzing analog circuits. Very specifically, they are used for generating S-parameter measurements [50]. Modern network analyzers incorporate features to stimulate and measure complex digital patterns.

- Dynamics Analyzers: These are low frequency network analyzers, specifically optimized to work with electro-mechanical systems. As they evolved they were optimized to make measurements that did not make sense in simple analog circuits. These have largely been replaced in the marketplace by similar functionality ported into the real-time control systems. We will discuss this in detail later in this chapter.

- The real-time system running the control loop. For all but the highest-speed control systems (or the lowest end analog loop), these are always digital computer systems of some type or another. Moore's law [42] has meant that we can do more and more of the above instrumentation functions on our real-time computing system.

Each of these has it's own advantages and issues, including but not limited to the sample rates that they can achieve, the number of bits in their ADCs (and if present DACs ), the signal processing that is available, the necessary test points to enable their functionality, the physical limitations of the device, whether they make measurements on open or closed-loop systems, and whether the measurements are supposed to reveal analog or digital parameters.

This tutorial will focus on two methods that can be readily applied in practice: step response measurements (usually involving the real-time digital controller) and frequency response function measurements (typically done using a Dynamic Signal Analyzer). We will lay out both the benefits and limitations of these methods in practical applications.

## 3.4 A Brief, Practical, and Incomplete Discussion of Domains

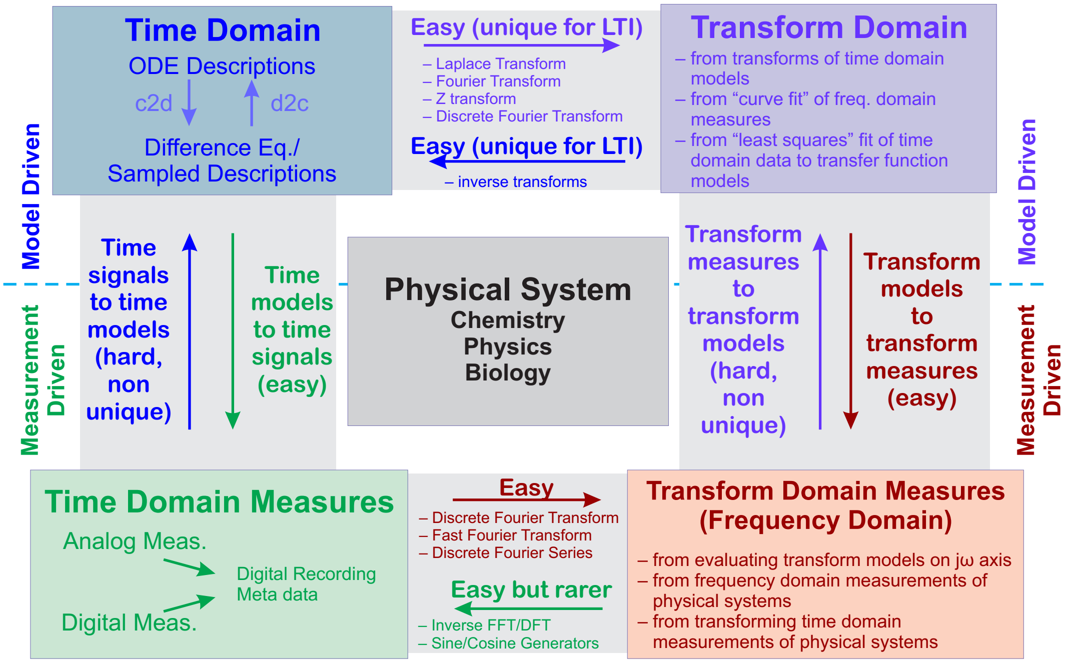

Time Domain
ODE Descriptions
c2d
d2c
Difference Eq./
Sampled Descriptions
Easy (unique for LTI)
– Laplace Transform
– Fourier Transform
– Z transform
– Discrete Fourier Transform
Easy (unique for LTI)
– inverse transforms
Transform Domain
– from transforms of time domain models
– from "curve fit" of freq. domain measures
– from "least squares" fit of time domain data to transfer function models
Model Driven
Measurement Driven
Time signals to time models (hard, non unique)
Time models to time signals (easy)
Physical System
Chemistry
Physics
Biology
Transform measures to transform models (hard, non unique)
Transform models to transform measures (easy)
Model Driven
Measurement Driven
Time Domain Measures
Analog Meas.
Digital Recording
Meta data
Digital Meas.
Easy
– Discrete Fourier Transform
– Fast Fourier Transform
– Discrete Fourier Series
Easy but rarer
– Inverse FFT/DFT
– Sine/Cosine Generators
Transform Domain Measures
(Frequency Domain)
– from evaluating transform models on jω axis
– from frequency domain measurements of physical systems
– from transforming time domain measurements of physical systems

Figure 3.1: **A diagram of the different domains.**

This section and its cousins will likely be repeated throughout this book in various forms. Since I don't have a publisher setting page limits, I don't have to send you flipping pages to find it again when you're in a different chapter. You can thank me later.

One of the things that becomes quasi-religious in the intensity of the discussions is the domain that engineers choose to work in. The cheat sheet version of these discussions is that the best domain is whatever domain the speaker likes to use. For the sake of argument, I'll assert that they all have different uses and in any given application, one may have more advantages. Still, what we really want to understand is the back and forth between domains so that we know what part of the understanding that we get from one domain can be applied to the other. Figure 3.1 is an attempt to put some of these ideas into a single diagram.

We start with the signal domain. For most control problems, this would be the time domain, $t$, the domain in which the signal variables move. However, as we get into spatial domains, the signals are all spatial (e.g. $x$, $y$, and/or $z$). Furthermore, we usually have problems where the signals move in some direction with respect to time, but we can also have wave type systems where the signals are also moving with respect to one of the spatial variables. Mostly, for control, we are talking about signals in the time domain, $t$, and in our digital control problems, there is sampling of $t$, usually but not always at some fixed sample rate $f_S$.

The next domain is the transform domain, and I want to speak generally here, but for control problems we are usually talking about Laplace transforms ($s$), Fourier transforms ($j\omega$), or Z transforms ($z$, which honestly doesn't have a very creative name). Also note that while $z$ is the transform variable in the discrete domain, it is often used interchangeably with the unit advance and its inverse, $z^{-1}$ is used even more often with the unit delay. There is some formalism that since the Z Transform is only unique for linear time-invariant (LTI) systems, the unit delay, $q^{-1}$ is used sometimes when filter/model parameters are changing so as to show independence from the Z Transform.

In any event, we get from the signal (time) domain into one of the transform domains via an integral done over infinite time. This means that we cannot actually ever compute a compete transform from measured data unless we have infinite time, and I've got other stuff to do. We get an approximation by going out "far enough", assuming that there is a far enough, which is a mathematical construct. Still, the thought experiment of the transform calculation tells us something. As Professor Bernard Widrow of Stanford explained many years ago, the integral essentially takes the function in time describing the signal and multiplies it at each step by a single frequency value (e.g. $j\omega_i$ for Fourier Transforms) and integrates that for all time. What we get out is the component of that signal that occurs at the single frequency, $j\omega_i$. If the functions are nice mathematically, then we can integrate for all time and get a transform domain description of the signal, showing its magnitude and phase at any frequency we choose to plug into the transformed result. If we have a function, $f(t)$ and the Fourier transform is given by:

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{3.3}$$

then for any value of $\omega$ we know what part of the signal is at that frequency. In the case of signal and transform domains, we move between then with different integrals, but on either side, we expect there to be some sort of analytical function.

The third domain is the frequency domain, which is often confused with the transform domain for some very good reasons, but is actually different in usage. For example, many systems are transformed using Laplace Transforms to give a transfer function, but this is still a formula. We often think of the frequency domain as evaluating the Laplace variable (which spans the entire complex plane) simply on the imaginary axis ($s = j\omega$), where this gives the signal at a never damped oscillation frequency. In usage, though, we actually perform this evaluation, often generating a complex response for each frequency on the imaginary axis (or each positive frequency) and we get insight from the shape of the response. There is a whole sub-domain of control design that gets to this point and works here.

The point of all of this discussion is to point out that while these domains are not the same, they should be more tightly coupled. This lack of coupling is apparent in discussions of transform domain and frequency domain. These terms are often used interchangeably, but I won't do this and neither should the reader.

Transform methods and models usually start with an LTI system model which gets transformed into a transform domain model. For example, for a continuous-time LTI system modeled by a set of differential equations, a Laplace transform puts the model into the transform domain, where the frequency variable is $s$, a complex variable that can range over an entire complex plane. Causal signals (signals that move forward in time) but die off map to the left half of the $s$ plane, while causal signals that grow without bound are mapped to the right half of the $s$ plane. (In contrast, acausal signals move backwards in time.) Signals that go on forever at a fixed amplitude are mapped to the $j\omega$ axis. For LTI systems described by LTI differential equations, the Laplace transform results (ignoring initial conditions) in a transfer function relating the output behavior to the input behavior with the free variable being $s$. From the complex plain and the transfer function, we can do designs based on the placement of poles and zeros, using methods such as root locus. We can evaluate stability using the Routh-Hurwitz criterion.

What are called frequency domain methods are brought about in these models by evaluating them along a single line of the $s$ plane, namely the imaginary ($j\omega$) axis . Typically, the set of frequencies used have $\omega = 2\pi f$ ranging from $0$ to $+\infty$. The frequency response function then is not an analytic function in the same way as a transfer function is, but instead is a vector of ordered pairs of numbers, a real frequency and its corresponding complex response. In this domain, working from these plots, we can synthesize the response of filter/controller components we might add to see how they affect the curves. That is, we take a proposed filter parameterization and we evaluate it along the $j\omega$ axis to synthesize the frequency response function that this filter element would give and to combine it with other system elements. From here, we can look at the overall open-loop response which gives us great intuition through margins such as gain and phase margin. We can synthetically close the loop and evaluate the projected closed-loop response, from which we can evaluate the closed-loop bandwidth and peaking.

Going from a transfer function (TF) to a frequency response function (FRF) is computationally easy: we simply evaluate the TF at all the values of $j\omega$ that we wish to visualize. The resulting vector pairs can be plotted in Nyquist Plots , Bode plots , or Nichols charts . I recall that the last of these was very cool, but I haven't done one in years. Mostly, the visualization is done in Bode plots, which are far more intuitive than the other two, but have some limitations.

On the other hand, going from FRFs to TFs is incredibly hard. The computational level is in some ways similar to the computer password generation problem, where the operation to generate a password is easy, but the operation to decode a password is incredibly difficult. Going from FRFs to TFs, through curve fits of rational functions of polynomials has a relatively simple version (which works until there is any noise on the FRF) and a more difficult version (which I am trying to explain and expand upon in this book). The reason why this matters – why this is such a breakpoint in the work of getting usable models from measurements is that some of the most powerful methods of measuring complicated system responses are the frequency domain measurements we will discuss later in this chapter. To be certain, there is a lot to do to get clean measurements of these systems, but even when we have done that, even when we have beautifully clean frequency response functions, we still need to turn those into parametric models if we want access to our most powerful design algorithms. The difficulty of this step is often why so many people pontificate about model based methods while rarely having access to an accurate model of a physical system.

Looking once more at Figure 3.1, a few things are worth noting. All interactions with the physical system are via the time and/or transform measures. However, those measurements are informed/shaped/limited by the models we assume.

All understanding of the physical system comes from our abstraction and modeling. However, to paraphrase Stephen Hawking [7], these models are useless unless they describe what we can observe and predict what we might observe in the future. For any real model to do that, it must be based on measurements of/interactions with the physical system.

The reason for the use of different domains is that each gives some insight that is not as accessible in one of the others. For example, the frequency domain allows us to visually examine the behavior of transform domain models in a way that is amazingly helpful. The effects of poles and zeros, of delay, of damping factors, all become very obvious. On the other hand, design for the frequency domain takes place in the transform domain. Even when someone realizes they want a lead, or a notch, or some other device to change a behavior in the frequency domain, they return to the transform domain, generate a prototype component and evaluate this across the $j\omega$ axis so as to be able to evaluate its response. That is because the path from the transform domain to frequency domain is so much easier than the reverse path that it is often more advantageous to iteratively design in the transform domain and "measure" the projected improvement in the frequency domain.

To get the most complete understanding, we really want to move between these measurements and models for any physical system. Some directions are "easy" so those transitions are made often. For example, going from the time domain to the transform domain involves a Laplace Transform (perhaps tedious, but straightforward) while going from the transform domain to the transform measure (frequency domain) involves evaluating the transform model at a set of frequencies along the $j\omega$ axis. Going from a time domain model to a time domain measure involves evaluating the model at successive time steps, i.e. a simulation.

Other transitions are hard and hard to reliably automate. A transform measure, i.e. a frequency domain plot being transformed to a transform domain model requires a curve fit of the complex response to a small set of parameters. A time measure (time response) needs to be fit to a time-domain model using the same order of magnitude of calculations and uncertainties. While the presence of noise creates some inaccuracies going from models to measures, the same level of noise in measures can make convergence to a model almost impossible sometimes.

I have said in other parts of this book that engineers need to be able to iterate on designs, processes, experiments, etc. yet in this area of going between physical measures and analytic models, we are often deficient. To have reliable success, we must be able to iterate. To be able to iterate rapidly, we must be able to automate, and make these transitions easy to the human designer in all directions.

Something that is easy for the human may still involve tons of measurements and calculations by the computer, but if the computational steps are robust and reliable, and the calculations are short in human time constants, then this is not an issue. For example, transition from continuous time representation to discrete time representation involves using one or more imperfect methods. However the calculation, as done by a computer (say using Matlab's c2d function) is relatively easy for the designer.

We need to make this a round trip. We need to close the loop if we want to cross the divide between theory and practice.

## 3.5 An Outline of the Rest of the Chapter

The rest of this chapter will proceed as follows.

- Section 3.6 will review some ideas of discretization of measurements and system models. Because of our focus on practical systems, and those are often better parameterized by physical parameters, it is important to understand sampling and what it does to our models.
- Section 3.7 discusses what happens to physical parameters in discretization.
- Section 3.8 takes a brief look at discrete-time, time domain identification.
- Section 3.9 then moves on to step response measurements, their limitations, and how to get the most out of them.
- Section 3.10 is about how to segment a time response measurement so that it can be averaged for better results in step response methods. It borrows heavily from what is done in digital oscilloscopes.
- Section 3.11 discusses what parameters we can extract from step response measurements.
- Section 3.12 compliments that by showing which model parameters we can extract from step response methods.
- Section 3.13 discusses an idealized first order section, and what model parameters we can extract from step response measurements.
- Section 3.14 repeats this with a second order section.
- Section 3.15 shifts gears and discusses frequency response measurements.
- Section 3.16 discusses what options we have for frequency response measurements.
- Section 3.18 discusses two different frequency-response measurement configurations, and the tradeoffs between them.
- Section 3.19 goes into some detail on Fourier analysis.
- Section 3.20 explains some useful aspects of using Fast Fourier Transforms (FFTs) to analyze system data.
- Section 3.21 provides some analysis of the stepped-sine integral.
- Section 3.22 discusses implementing stepped-sine on an Field Programmable Gate Array (FPGA).
- Section 3.23, discusses some of the software that follows that integration.
- As the stepped-sine seems to require much more effort than an FFT, Section 3.24 discusses the tradeoffs between these two methods.

- Section 3.25 deviates from the math some to explain the importance of connected measurements.
- Similarly, Section 3.26 tries to make the case for building stepped-sine directly into digital controllers.
- Section 3.27 shows some simulation and measured results from built-in stepped sine measurements.
- Section 3.28 discusses the difficulty of extracting a parametric model from frequency response function measurements.
- Section 3.29 explains some methods for improving that.
- Section 3.30 explains the nasty effects of not accounting for delay in curve fits.

## 3.6 A Brief, Practical, and Incomplete Review of Discretization

As mentioned in the workshop introduction, we will assume that the control law will be implemented in a digital computer (or a computer for anyone born after 1980). By digital computer we may be discussing anything from a Field Programmable Gate Array (FPGA) chip connected to the physical system with Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs), to MicroControllers, to Digital Signal Processing (DSP) chips, to full processors, to Linux and Windows systems. What we are excluding is the implementation of control laws via analog circuitry or analog computers (the “other” computers). While there are some clear sampling advantages to analog implementation, and these still exist at very high frequencies – where it is hard to sample fast enough and actually do control calculations between the samples, these are increasingly the domain of a small niche. An understanding of analog circuit design is extremely helpful in understanding the interface circuit that connect sensors to the ADCs and DACs to driver electronics and actuators. We hope to touch on this at the end of the workshop, or in a future, extended workshop.

Realizing that most modern implementations are on digital computers, how do we discuss thinking in analog while implementing in digital? When is “sampling fast” enough and what insights can we gain from paying attention to how we discretize things?

The fact that controllers will be digital means that we should want a very physical, intuitive understanding of how sampling affects our perception of the physical signal (Figure 3.2), and how it limits

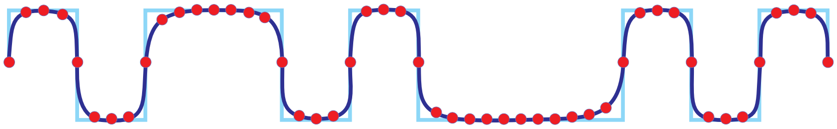


Figure 3.2: A repeat of Figure 2.18. A diagram of real-time sampling of a signal. The ideal signal (cyan) has sharp transitions, but passage through the physical system often has a low-pass filter (LPF) nature to it resulting in a rounded curve (blue). The sampled signal (red) typically (but not always) is produced at regular sample intervals and with a certain level of quantization in the conversion. One can say that the real goal of a sampling system is to recover or infer the behavior of the ideal signal, but this simple diagram points out that even the physical filtering must be taken into account in any attempt to do so.

what we can do with the controller. In particular, sampling adds delay to the system, delay results in negative phase, and negative phase reduces phase margin. Thus, the simple act of sampling the data limits our top end bandwidth before anything else can.

The second thing we must understand about sampling is that no one discretization model fully captures our ability to control a system digitally. The workhorse of most discretization in control is the Zero Order Hold (ZOH) Equivalent [15], but even the most cursory look at this method reveals that all physical intuition about any system more complex than a double integrator is lost using this method. We will discuss how using other discretization methods may preserve physicality with a small (and often negligible) loss of mathematical exactness in the model.

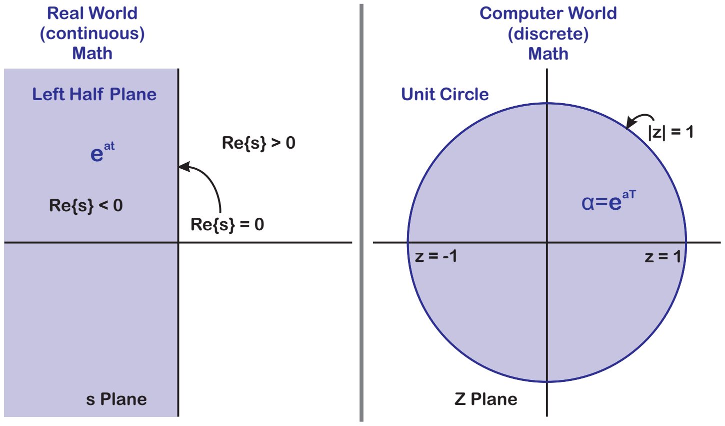


Figure 3.3: Continuous and discrete: the left half plane (s) versus the unit circle (z).

This brings up a somewhat important distinction. When most people are trained in their first control design work, they are taught system modeling and control design in continuous time. This is both mathematically easier and historically accurate, since analog control design and understanding preceded – and continues to lead – digital control design and understanding. The act of conversion of signals, models, and understanding is something that has been going on since the 1950s [51]. Generally, though, there are two schools of thought: that in which the system and controller are modeled in continuous time and the controller is discretized for implementation with caveats about too much sample delay and the other in which the physical model is discretized and then all the controller design is done in discrete time. As a rule of thumb, the latter has held sway since the groundbreaking work of [51], in part because the mathematics are exact in discrete time. However, much of the physical intuition is lost and this is why in practice, the former is almost certainly more common. The discretization of PID controllers falls into the former category, and so it is good to get an understanding of simple controller discretization methods and their effect on the controller behavior.

This section describes different methods of coming up with discrete versions of differential equations and transfer functions. There are tradeoffs in how this is done, and it seems that the discussion that I have always used from Franklin and Powell [15, 52] is worth having in a unified and coherent discussion.

At best, discretization methods map the left half of the $s$ plane into the unit circle of the $z$ plane as diagrammed in Figure 3.3. The interior of the unit circle can be thought of as the region where geometric series (the discrete version of exponentials) decay. There is no way to map a half plane into a small circle without some distortion, but different methods map the plane with different amounts of distortion.

### 3.6.1 Discretization Via Numerical Integration Equivalents

One of the easiest and most common ways to map from the continuous to discrete domain is through numerical integration, where a continuous function is approximated by some sum of discrete functions for which the integral is knowable simply by evaluating the function at the sample points.

In the text that follows, consider a function, $e(t)$. We wish to know the integral under that curve

$$u(t) = \int_{t_0}^{t_f} e(t)dt. \tag{3.4}$$

We approximate this by converting this into a sum,

$$u(t_0 + kT) = u(t_0) + \sum_{l=t_0+T}^{t=t_0+kT} g(e(lT), e((l-1)T), T). \tag{3.5}$$

In this case, the time axis has been broken into equal intervals of width, $t$.

Usually, we do two things to make this problem simpler: we reset the time axis so that $t_0 = 0$ and we choose a function, $g$ which makes this sum easy to compute. All the rules that will be discussed here are simply different ways of picking $g$. Each of these choices has consequences, but as a rule, if $t$ is small enough, any of them work. Also, for this discussion, we will only talk about functions, $g$, that rely on the sample points immediately bracketing the interval. It is possible to use points outside this interval, as is done in the stepped-sine (called swept sine in industry) calculation for the HP 3562A [53], but it's not useful for our current discussion.

### 3.6.2 Forward Rectangular Rule

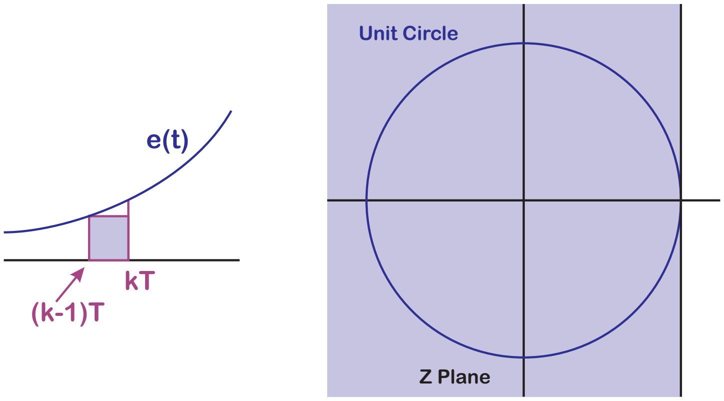


Figure 3.4: The forward rectangular rule (on the left) and how it maps the $s$ plane to the $z$ plane (right).

The simplest discretization rule is the forward rectangular rule, also known as Euler's method, shown in Figure 3.4. In this rule, the area under the integral of the curve is approximated by a rectangle. That is,

$$g(e(kT), e((k-1)T), T) = Te((k-1)T). \tag{3.6}$$

The idea is that if the steps are small enough, the error generated by these rectangles will be small relative to the value of the integral. In the case of the forward rule, we have

$$u(k) = u(k-1) + Te(k-1). \tag{3.7}$$

In other words, the forward rule takes the value of $e$ at time $(k-1)T$ and holds it steady until time $kT$. This rectangle is used to estimate the area under the curve. The $z$-transform of Equation 3.7 is

$$U(z) = z^{-1}U(z) + z^{-1}TE(z), \tag{3.8}$$

or

$$U(z)(1 - z^{-1}) = z^{-1}TE(z). \tag{3.9}$$

This means

$$\frac{U(z)}{E(z)} = \frac{z^{-1}T}{1 - z^{-1}}. \tag{3.10}$$

The Forward Rectangular Rule approximates $\frac{1}{s}$ with $\frac{z^{-1}T}{1-z^{-1}}$. That is

$$\frac{1}{s} \longleftarrow \frac{z^{-1}T}{1 - z^{-1}}, \tag{3.11}$$

or equivalently,

$$s \longleftarrow \frac{z-1}{T}. \tag{3.12}$$

We can see that this has mapped any differential element, $s$, into the $z$ plane by simply shifting it to the right by $1$ and scaling it by $T$. What this means is that the left half $s$ plane maps into the $z$ plane in a way where stable portions of the $s$ plane are outside the unit circle in the $z$ plane (unstable). This depends upon the original pole location in the $s$ plane and the size of $T$. The smaller $T$ is, the more likely the pole will fall within the unit circle, $||z|| = 1$.

### 3.6.3 Backward Rectangular Rule (BR) Equivalent

The next simplest discretization rule is the backward rectangular rule equivalent or simply the backwards rule (BR) equivalent , shown in Figure 3.5. As in the forward rule, the area under the integral of the curve is approximated by a rectangle. However, in this rule, the value of the current position on the curve is used rather than that of the previous position, i.e.

$$g(e(kT), e((k-1)T), T) = Te(kT). \tag{3.13}$$

$$u(k) = u(k-1) + Te(k). \tag{3.14}$$

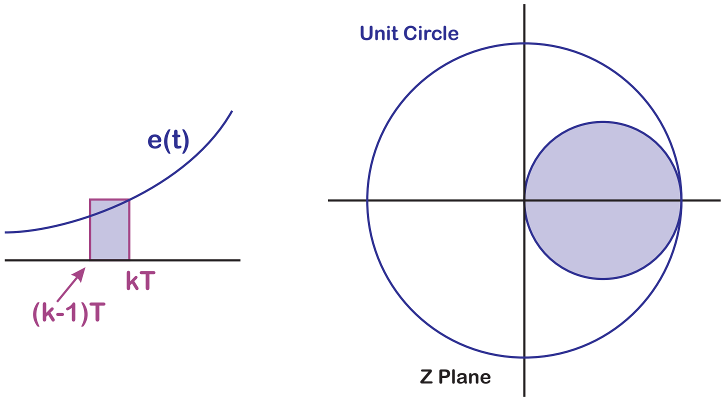


Figure 3.5: The backward rectangular rule (on the left) and how it maps the $s$ plane to the $z$ plane (right). Discretization using the backwards rectangular integration rule maps the left half plane into a small (conservative region of the unit circle. The the area where $s \rightarrow -\infty$ maps to an area around $z = 0$.

In other words, the backward rule takes the value of $e$ at time $kT$ and holds it steady backwards until time $(k-1)T$. This rectangle is used to estimate the area under the curve. The $z$-transform of Equation 3.14 is

$$U(z) = z^{-1}U(z) + TE(z), \tag{3.15}$$

or

$$U(z)(1 - z^{-1}) = TE(z). \tag{3.16}$$

This means

$$\frac{U(z)}{E(z)} = \frac{T}{1 - z^{-1}}. \tag{3.17}$$

The backward rectangular rule approximates $\frac{1}{s}$ with $\frac{T}{1-z^{-1}}$. That is

$$\frac{1}{s} \longleftarrow \frac{T}{1 - z^{-1}}, \tag{3.18}$$

or equivalently,

$$s \longleftarrow \frac{z-1}{Tz}. \tag{3.19}$$

This small change makes a big difference. Looking from Equation 3.17 to Equation 3.10, we see that Equation 3.10 has an extra sample delay of 1 time step. As Frankenstein's monster would say if he were a control engineer, "Delay bad!"

We can see from Equation 3.19 that this has mapped any differential element, $s$, into the $z$ plane by a bilinear transformation. This transformation is such that the left half $s$ plane is mapped into a circle centered at $z = \frac{1}{2}$ with radius $\frac{1}{2}$. Thus, the backward rule will not create instabilities in the $z$ plane when there were none in the $s$ plane. In fact, some unstable regions of the $s$ plane will be mapped within the unit circle and be stable. However, the ideal is to map the left half $s$ plane into the unit circle on the $z$ plane.

### 3.6.4 Trapezoidal Rule (TR) Equivalent

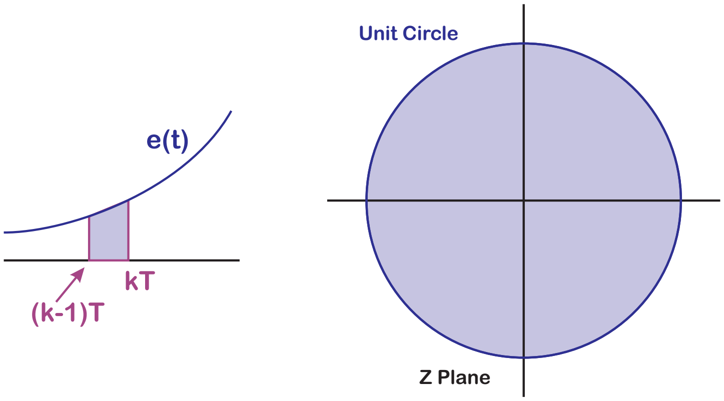


Figure 3.6: The trapezoidal rule (on the left) and how it maps the $s$ plane to the $z$ plane (right). Discretization using the trapezoidal integration rule maps the left half plane into the entire unit circle. The the area where $s \rightarrow -\infty$ maps to an area around $z = -1$.

The next simplest discretization rule is the trapezoidal rule (TR) equivalent , shown in Figure 3.6. Unlike the forward and backward rules, the area under the integral of the curve is approximated by a trapezoid. A line is drawn between the current and previous position, i.e.

$$g(e(kT), e((k-1)T), T) = T\left[\frac{e((k-1)T) + e(kT)}{2}\right] \tag{3.20}$$

$$u(k) = u(k-1) + \frac{T}{2}\left[e(k-1) + e(k)\right]. \tag{3.21}$$

In other words, the trapezoidal rule takes the average values of $e$ at times $(k-1)T$ and $kT$. This is equivalent to constructing a trapezoid in that space, or drawing a line between the two points. The $z$-transform of Equation 3.21 is

$$U(z) = z^{-1}U(z) + \frac{T}{2}(z^{-1}E(z) + E(z)), \tag{3.22}$$

or

$$U(z)(1-z^{-1}) = \frac{T}{2}E(z)(1+z^{-1}). \tag{3.23}$$

This means

$$\frac{U(z)}{E(z)} = \left(\frac{T}{2}\right)\left(\frac{1+z^{-1}}{1-z^{-1}}\right). \tag{3.24}$$

The trapezoidal rule approximates $\frac{1}{s}$ with $\left(\frac{T}{2}\right)\left(\frac{1+z^{-1}}{1-z^{-1}}\right)$. That is

$$\frac{1}{s} \longleftarrow \left(\frac{T}{2}\right)\left(\frac{1+z^{-1}}{1-z^{-1}}\right), \tag{3.25}$$

or equivalently,

$$s \longleftarrow \left(\frac{2}{T}\right)\left(\frac{z-1}{z+1}\right). \tag{3.26}$$

Again, this small change makes a big difference. We have now made a first order approximation to the curve. We can see from Equation 3.26 that this has mapped any differential element, $s$, into the $z$ plane by a bilinear transformation. This transformation is such that the left half $s$ plane is mapped entirely within the unit circle. This is not an ideal mapping, but it is possible to choose the frequency at which the map will be an exact match. This is called prewarping. The important thing to realize about this is that with this first order approximation, we can do a much more reasonable job of using our sampling bandwidth to approximate the curve. Likewise, using the bilinear transformation for mapping control designs from the $s$ plane to the $z$ plane should provide improved results.

## 3.6.5 Numerical Integration Equivalent Summary

| Method | $s$ Approximation | $z$ Approximation |
|---|---|---|
| **Forward Rectangular Rule** | $s \approx \frac{z-1}{T}$ | $z \approx 1 + Ts$ |
| **Backward Rectangular Rule** | $s \approx \frac{z-1}{Tz}$ | $z \approx \frac{1}{1-Ts}$ |
| **Trapezoidal Rule** | $s \approx \left(\frac{2}{T}\right)\frac{z-1}{z+1}$ | $z \approx \frac{1+\frac{T}{2}s}{1-\frac{T}{2}s}$ |

Table 3.1: **Summary of Discrete Integration Rules**

### 3.6.6 Matched Pole-Zero Equivalent

In the Matched Pole-Zero Equivalent, we essentially individually discretize each continuous time pole and zero via $\alpha = e^{aT_S}$. Consider the simple continuous time transfer function:

$$H(s) = K_c \frac{(s+b_{0,c})(s+b_{1,c})}{(s+a_{0,c})(s+a_{1,c})} \tag{3.27}$$

which we would like to transform into the discrete transfer function

$$H(z) = K_d \frac{(z-b_{0,d})(z-b_{1,d})}{(z-a_{0,d})(z-a_{1,d})}. \tag{3.28}$$

Each of the continuous time poles are mapped to discrete time poles via:

$$a_{i,d} = e^{a_{i,c}T_S}, \tag{3.29}$$

where once again, $T_S$ is our sample period. (Sometimes we use $T$, when it is clear from the context and it simplifies the reading.) Similarly, all finite zeros are mapped via

$$b_{i,d} = e^{b_{i,c}T_S}. \tag{3.30}$$

Generally, all zeros at $s = \infty$ are mapped to $z = -1$, which ends up being very similar to the trapezoidal rule. However, this method also allows one of the infinite continuous-time zeros to be mapped to $z = \infty$ to account for time delay [15]. A zero at $z = \infty$ corresponds to an extra pole at $z = 0$, and the $z^{-1}$ model for time delay is pretty standard. (Technically, it should be $q^{-1}$ since $z$ refers to the Z-transform, but everyone uses $z^{-1}$ and $q^{-1}$ interchangeably.) It is my opinion that we should not add extra time delay in our controller modeling, but we need to account for it in our plant modeling.

If the pole (or zero) is part of a complex pair, i.e. $a_{i,c} = -a_{R,i,c} + ja_{I,i,c}$, then

$$a_{i,d} = e^{-a_{R,i,c}T_S} e^{ja_{I,i,c}T_S} = e^{-\sigma_i} e^{\theta_i}, \tag{3.31}$$

where $-\sigma_i$ and $\theta_i$ represent the real and imaginary parts of the z-plane pole (or zero).

When the number of poles and zeros match, and they are all real and distinct, this is pretty straightforward. When they don't, we have to make some adjustments. Nevertheless, this method differs from the others in that we cannot work directly from a denominator polynomial or a numerator polynomial. We have to find all the roots. When the roots form a complex pair, we can adjust by discretizing the complex pair together as described in Section 6.16 for the Multinotch [54] and the Biquad State Space (BSS) [3, 4]. Because of that, we will repeat the derivation of coefficients for a biquad section here, even though it's ahead of the original use later in Section 6.16.

| | |
|---|---|
| $f_{N,i}$ | Center frequency of numerator (Hz) |
| $\omega_{N,i}$ | Center frequency of numerator (rad/s) |
| $Q_{N,i}$ | Quality factor of numerator |
| $\zeta_{N,i} = \frac{1}{Q_{N,i}}$ | Damping factor of numerator |
| $f_{D,i}$ | Center frequency of denominator (Hz) |
| $\omega_{D,i}$ | Center frequency of denominator (rad/s) |
| $Q_{D,i}$ | Quality factor of denominator |
| $\zeta_{D,i} = \frac{1}{Q_{D,i}}$ | Damping factor of denominator |

Table 3.2: **Physical coefficients used to specify a biquad section. Note that some of these are redundant, so that the choice of $\zeta$ versus $Q$ or $f$ versus $\omega$ is simply a user preference.**

Consider another continuous time second order filter:

$$B_i(s) = K_{i,c}\frac{s^2 + b_{i,1,c}s + b_{i,2,c}}{s^2 + a_{i,1,c}s + a_{2,c}} = K_c\frac{s^2 + 2\zeta_{N,i}\omega_{N,i} + \omega_{N,i}^2}{s^2 + 2\zeta_{D,i}\omega_{D,i} + \omega_{D,i}^2}, \tag{3.32}$$

with the resonance parameter explanations in Table 3.2. We want to transform this into a digital biquad

$$B_i(z) = \frac{b_{i,0}\left(1 + \tilde{b}_{i,1}z^{-1} + \tilde{b}_{i,2}z^{-2}\right)}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}}. \tag{3.33}$$

via matched pole-zero mapping. The particular choices for the names will become apparent if we get to Section 6.11.

Because the resulting digital filter is a digital biquad, there are no excess poles or zeros. Furthermore, using this form where we have factored the $b_{i,0}$ gain out of each numerator means that all the biquads will have a uniform structure. Taking our design from an analog response of a ratio of a second order numerator and denominator, we can discretize the poles and zeros using matched pole-zero mapping [15]. This allows us to parametrize each biquad section using very physical parameters, as shown in Table 3.2. The factored out gain, $b_{i,0}$, can be used as is, or can be altered so that, for example, the DC gain of the biquad section will be $1$.

Assuming a complex pair of poles (or zeros), mapping via $z = e^{sT_S}$, and recombining the results yields some straightforward formulas. For $a_{i,2}$ and $\tilde{b}_{i,2}$ we have

$$a_{i,2} = e^{-2\omega_{D,i}T_S\zeta_{D,i}} \tag{3.34}$$

$$\tilde{b}_{i,2} = e^{-2\omega_{N,i}T_S\zeta_{N,i}} \tag{3.35}$$

Whether the poles (or zeros) are a complex pair depends upon $\left|\zeta_{D,i}\right|$ ($\left|\zeta_{N,i}\right|$). For $\left|\zeta_{D,i}\right| < 1$ we have a complex pair of poles and so

$$a_{i,1} = -2e^{-\omega_{D,i} T_S \zeta_{D,i}} \cos\left(\omega_{D,i} T_S \sqrt{1-\zeta_{D,i}^2}\right). \tag{3.36}$$

If $\left|\zeta_{N,i}\right| < 1$ we have a complex pair of zeros and so

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i} T_S \zeta_{N,i}} \cos\left(\omega_{N,i} T_S \sqrt{1-\zeta_{N,i}^2}\right). \tag{3.37}$$

While these two cases represent cases when the desired filters have very sharp peaks or notches (for example to equalize a response with very sharp notches or peaks), there are other possibilities. For example setting $\left|\zeta_{D,i}\right| = 1$ ($\left|\zeta_{N,i}\right| = 1$) means that the poles (zeros) are real and equal, so $a_{i,1}$ ($\tilde{b}_{i,1}$) are given by:

$$a_{i,1} = -2e^{-\omega_{D,i} T_S \zeta_{D,i}} \tag{3.38}$$

and

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i} T_S \zeta_{N,i}}. \tag{3.39}$$

Finally, if $\left|\zeta_{D,i}\right| > 1$ ($\left|\zeta_{N,i}\right| > 1$) means that the poles (zeros) are real and distinct, so $a_{i,1}$ ($\tilde{b}_{i,1}$) are given by using the cosh relation:

$$a_{i,1} = -2e^{-\omega_{D,i} T_S \zeta_{D,i}} \cosh\left(\omega_{D,i} T_S \sqrt{\zeta_{D,i}^2 - 1}\right) \tag{3.40}$$

and

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i} T_S \zeta_{D,i}} \cosh\left(\omega_{N,i} T_S \sqrt{\zeta_{N,i}^2 - 1}\right). \tag{3.41}$$

The entire conversion routine, which turns the physical parameters of Table 3.2 into discrete filter coefficients can be implemented in a short MATLAB or Octave function.

When there is a pole-zero excess, then we must make smart decisions about where to place the extra zeros. That being said, this methodology works extremely well in the Multinotch (Section 6.11) and the Biquad State Space (Section 9.15) in part because they are structured around first and second order blocks.

### 3.6.7 The Zero-Order Hold Equivalent

The zero-order hold (ZOH) equivalent has become the default standard for most control systems discretization. It is often the gateway to the direct digital design methodology, in which the entire physical system model is discretized at once, and then control design applied only in discrete time, with nary a thought to that wiry old continuous time. One might ask why this method is so popular in theoretical work. It seems that there are a few probable reasons:

- The match between the continuous and discrete responses is exact at the sample points. Generally one hopes that things are well behaved in between the sample points.

- The ZOH equivalent builds in a single sample delay into the system model to account for all that sample and hold and other computer stuff. The single delay is for the entire model. This is in contrast to applying something like the trapezoidal rule (TR) equivalent 3.6.4 which makes no account for the sample and processing delay, or the backward rule (BR) equivalent 3.6.3 which adds extra delay to each conversion. When discretizing an entire model this might not matter much, but in practical systems, the individual blocks are often discretized individually and not taking these delays into account can have severe penalties.

- The ZOH equivalent is a one-time thing, and although it is difficult to do by hand for anything more than a second or third order system, it can be calculated numerically quite well in tools such as MATLAB . Thus, it is the default.

The ZOH equivalent maps the continuous-time poles to the discrete-time poles via $p_D = e^{p_C T_S}$. The location of the discrete-time zeros is much more complicated.

### 3.6.8 Discretization Summary

In the past 30 years, as the analysis tools have gotten better (i.e. as MATLAB has become more universal) discretization has become something most folks working in control take for granted. Just use c2d and be done. However, we will see as we get further along that while this might work when sample rates are high relative to the dynamics being shaped, in many practical situations we really should have an intuitive understanding of the different sampling models on our system understanding and on our controller implementation.

## 3.7 The Fate of Physical Parameters in Discretization

The fact that controllers will be digital means that we should want a very physical, intuitive understanding of how sampling affects our perception of the physical signal (Figure 3.2), and how it limits what we can do with the controller. In particular, sampling adds delay to the system, delay results in negative phase, and negative phase reduces phase margin. Thus, the simple act of sampling the data limits our top end bandwidth before anything else can.

The second thing we must understand about sampling is that no one discretization model fully captures our ability to control a system digitally. The workhorse of most discretization in control is the Zero Order Hold (ZOH) Equivalent [15], but even the most cursory look at this method reveals that all physical intuition about any system more complex than a double integrator is lost using this method. We will discuss how using other discretization methods may preserve physicality with a small (and often negligible) loss of mathematical exactness in the model.

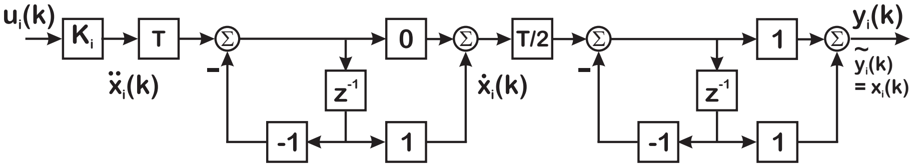


Figure 3.7: Discrete double integrator BLSS model (ZOH equivalent). In this drawing, we've chosen to make the index, i, as with a biquad stage, but we are explicitly labeling the different integration levels.

A simple example of what is wrong with our understanding of discretization is in discretizing the double integrator of Section 2.3.5 and Figure 2.11. In the continuous time drawing on the right, we can easily pick off the velocity term, which makes using both position and velocity measurements available for feedback. Let's remember that for a second order system, access to position and velocity for feedback is full state feedback, and this is the 800 pound gorilla of control theory – it puts the poles anywhere it wants. The situation gets much worse as soon as we discretize. If we use the standard Zero-Order Hold equivalent model and use a Bilinear State-Space form (BLSS) [5] shown in Figure 3.7, we can access the position, but the intermediate result does not do a good job of representing velocity, since the integrators are not interchangeable. It seems highly illogical to go to the trouble of generating a state-space realization on the basis that state-space gives us access to all the states and somehow lose access to velocity in one of the simplest state-space realizations available [5].

Consider the earlier example of the second order resonance, of Equations 2.18 and 2.19:

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}} = \frac{K\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}, \tag{3.42}$$

or to the more general analog biquad model:

$$\begin{aligned} \ddot{x} &= -a_1\dot{x} - a_2 x + u & (3.43)\\ y &= b_0\ddot{x} + b_1\dot{x} + b_2, & (3.44) \end{aligned}$$

with the transfer function description:

$$\begin{gathered} \left(s^2 + a_1 s + a_2\right) = U(s) \\ Y(s) = \left(b_0 s^2 + b_1 s + b_2\right), \end{gathered} \tag{3.45}$$

yielding

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}. \tag{3.46}$$

A discrete transfer function version of (3.46)

$$\frac{Y(z)}{U(z)} = \frac{b_{0,D}z^2 + b_{1,D}z + b_{2,D}}{z^2 + a_{1,D}z + a_{2,D}} = \frac{b_{0,D} + b_{1,D}z^{-1} + b_{2,D}z^{-2}}{1 + a_{1,D}z^{-1} + a_{2,D}z^{-2}}. \tag{3.47}$$

The question is what the meaning of the new coefficients in relation to the old ones, and this is entirely related to both the original parameters and the discretization method. Exact discretization methods obscure the coefficient meaning and couple states in a very non-intuitive way. Some other approximations, in which the original transfer function is broken into a cascade of second order sections (biquads) can preserve much physical intuition.

There are lots of discretization methods and even when one does the “exact” math, one doesn’t get a satisfying answer. In fact, the exact math can give an answer that is so convoluted as to obscure any hope of physical intuition and this is bad. The trapezoidal rule, also known as Tustin’s Rule or a bilinear equivalent, substitutes discrete time operators (based on the Z transform) for the continuous time operator (based on the Laplace transform). Using the Trapezoidal Rule, we make the substitution:

$$s \longleftarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) \tag{3.48}$$

and if we substitute for $s$ in (3.46) to get to (3.47) then we end up with the following mappings:

$$
\begin{array}{lcllcl}
\Delta & = & 1+a_1\frac{T}{2}+a_2\frac{T^2}{4} & & & \\
b_{0,D} & = & \frac{1}{\Delta}\left(b_0+b_1\frac{T}{2}+b_2\frac{T^2}{4}\right) & a_{0,D} & = & 1 \\
b_{1,D} & = & \frac{2}{\Delta}\left(b_2\frac{T^2}{4}-b_0\right) & a_{1,D} & = & \frac{2}{\Delta}\left(a_2\frac{T^2}{4}-1\right) \\
b_{2,D} & = & \frac{1}{\Delta}\left(b_0-b_1\frac{T}{2}+b_2\frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1-a_1\frac{T}{2}+a_2\frac{T^2}{4}\right)
\end{array}
\tag{3.49}
$$

For the simple spring-mass-damper system of (3.42), we end up with

$$
\begin{array}{lcllcl}
\Delta & = & 1+\frac{b}{m}\frac{T}{2}+\frac{k}{m}\frac{T^2}{4} & & & \\
b_{0,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{0,D} & = & 1 \\
b_{1,D} & = & \frac{2}{\Delta}\left(\frac{2}{m}\frac{T^2}{4}\right) & a_{1,D} & = & \frac{2}{\Delta}\left(\frac{k}{m}\frac{T^2}{4}-1\right) \\
b_{2,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1-\frac{b}{m}\frac{T}{2}+\frac{k}{m}\frac{T^2}{4}\right)
\end{array}
\tag{3.50}
$$

The point of the discussion is, if it looks complicated, it's supposed to be. The physical parameters get lost in the shuffle a lot and we need to fight to keep them in terms that are meaningful in our discrete time model. The $b$, $k$, and $m$ parameters are spread all over Equation 3.50. This also spells bad news for trying to extract physical parameters from time domain ID with discrete time models. The accuracy to which we need to identify the coefficients in Equation 3.47 in order to back out the physical coefficients using Equation 3.49 is tremendous. The problems get worse when the system order gets higher.

What I've come to realize is that breaking the problem down into blocks and discretizing the blocks makes a lot of sense in the sense that each block has it's own discretization error, but it also preserves the physical meaning of the original block (if you do it right, which still isn't trivial).

In even a simple problem such as our spring-mass-damper example, the physical parameters matter. Perhaps the damping coefficient, $b$, is changing as a shock absorber on a car wears out. Perhaps a mass, $m$, is changing as a device picks up a load. Were we to try to detect a changed mass from Equation 3.50, we would have to measure and back out changes from 5 parameters. Furthermore, $m$ shows up in a nonlinear way in all 5 terms. Contrast that with the elegant way in which $m$, $b$, or $k$ show up in the analog model of Section 2.3.8.

## 3.8 A Brief Look at Discrete-Time Time Domain Identification

In this section, we will briefly describe the basic idea behind time domain identification using linear, discrete time models. This type of identification dominates the academic literature and yet its use in high performance, physical dynamic systems can be limited. With this brief section we should be able to understand some of this dichotomy.

In the signal processing world, much of the work is dominated by Moving Average (a.k.a. all-zeros or FIR Filter) model. Say,

$$y(k) = b_0 u(k) + b_1 u(k-1) + \ldots + b_N u(k-N) + n(k), \tag{3.51}$$

where the $b_i$ are the model (filter) coefficients, $u_{k-i}$ are delayed values of the input, and $n(k)$ is some random noise input. This can be rephrase in a vector form:

$$y(k) = [b_0, b_1, b_2, \ldots, b_N] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \vdots \\ u_{k-N} \end{bmatrix} + n(k), \tag{3.52}$$

which is often used in system ID equations. The Z-transform of this (discrete time frequency domain) is:

$$Y(Z) = \left(b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}\right) U(z) + N(z). \tag{3.53}$$

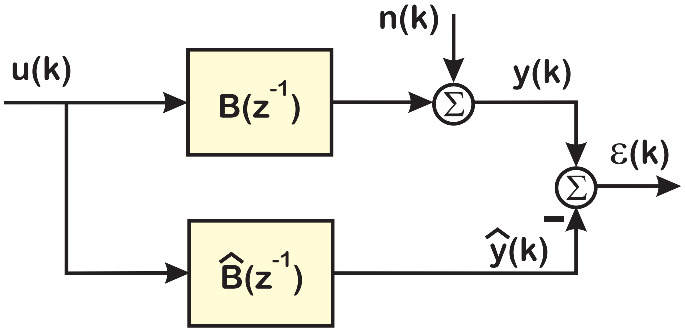


Figure 3.8: **A diagram of discrete time, time domain identification of a Moving Average (MA) or FIR model.**

If we do not know the model parameters, $b_i$, we can generate an estimate of the model with our best estimate of them:

$$\hat{y}(k) = \left[\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_N\right] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \vdots \\ u_{k-N} \end{bmatrix}, \tag{3.54}$$

where we assume that we know the past inputs and we don't add the unknown noise into our model. The error between the output of the actual model and the model estimate would be:

$$\varepsilon = y(k) - \hat{y}(k) = y(k) - \left[\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_N\right] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \vdots \\ u_{k-N} \end{bmatrix}. \tag{3.55}$$

This is diagrammed in Figure 3.8.

Estimating the model parameters, having the $\hat{b}_i$ converge to the $b_i$ over time generally involves forming the squared error, and doing a search, usually based on some gradient or gradient plus adjustment method. Since we do not know the actual parameters, we cannot form an error based on them, but only on the measured signals and the parameter estimates. If we let

$$\hat{B}_k = \left[\hat{b}_{0,k}, \hat{b}_{1,k}, \hat{b}_{2,k}, \ldots, \hat{b}_{N,k}\right] \text{ and} \tag{3.56}$$

$$U_k = \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \ldots \\ u_{k-N} \end{bmatrix}, \tag{3.57}$$

then

$$\varepsilon_k^2 = y_k^2 - 2\hat{B}_k U_k + U_k^T \hat{B}_k^T \hat{B}_k U_k. \tag{3.58}$$

To make sense of it, we take the expectation, and get:

$$E\left\{\varepsilon_k^2\right\} = E\left\{y_k^2\right\} - 2E\left\{\hat{B}_k U_k\right\} + E\left\{U_k^T \hat{B}_k^T \hat{B}_k U_k\right\}. \tag{3.59}$$

This results in a least-squares problem that can be solved recursively with a gradient or Newton or other method.

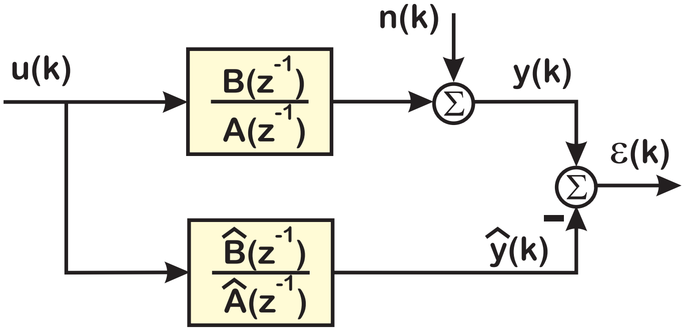


Figure 3.9: **A diagram of discrete time, time domain identification of an Auto-Regressive, Moving Average (ARMA) or IIR model.**

In control systems, we normally assume a model that has both poles and zeros, represented by an IIR (Infinite Impulse Response) filter model also known as an Auto-Regressive, Moving Average (ARMA) model. In this case, the equations are:

$$y(k) = b_0u(k) + b_1u(k-1) + \ldots + b_Nu(k-N) - a_1y(k-1) - a_2y_(k-2)\ldots - a_Ny(k-N) + n(k), \quad (3.60)$$

where the $b_i, a_i$ are the model (filter) coefficients, $y_{k-i}$ and $u_{k-i}$ are delayed values of the output and input, respectively, and $n(k)$ is some random noise input. This can be rephrase in a vector form:

$$y(k) = [b_0, b_1, b_2, \ldots, b_N, a_1, a_2, \ldots, a_N] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \vdots \\ u_{k-N} \\ -y_{k-1} \\ -y_{k-2} \\ \ldots \\ -y_{k-N} \end{bmatrix} + n(k), \quad (3.61)$$

which is often used in system ID equations. The Z-transform of this (discrete time frequency domain) is:

$$Y(Z) = \frac{\left(b_0 + b_1z^{-1} + \ldots + b_Nz^{-N}\right)}{\left(1 + a_1z^{-1} + \ldots + a_Nz^{-N}\right)}U(z) + N(z). \quad (3.62)$$

If we do not know the model parameters, $b_i, a_i$, we can generate an estimate of the model with our best estimate of them:

$$\hat{y}(k) = \left[\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_N, -\hat{a}_1, -\hat{a}_2, \ldots, -\hat{a}_N\right] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \ldots \\ u_{k-N} \\ -y_{k-1} \\ -y_{k-2} \\ \ldots \\ -y_{k-N} \end{bmatrix}, \tag{3.63}$$

The error between the output of the actual model and the model estimate would be:

$$\varepsilon = y(k) - \hat{y}(k) = y(k) - \left[\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_N, -\hat{a}_1, -\hat{a}_2, \ldots, -\hat{a}_N\right] \begin{bmatrix} u_k \\ u_{k-1} \\ u_{k-2} \\ \ldots \\ u_{k-N} \\ -y_{k-1} \\ -y_{k-2} \\ \ldots \\ -y_{k-N} \end{bmatrix}, \tag{3.64}$$

and we can similarly generate a least squares problem from this, diagrammed in Figure 3.9.

Somewhere, I have rederived the least squares problems in my notes several times over. However, I don't know where those notes are and am running out of time. The derivations that lead to various least squares problems that can be solved iteratively are found in many texts, e.g. [24, 47], so it can be found. Maybe by next year's notes. Right now, with the time I have left, I'm going to push to give some practical insight that may be missing from some textbooks.

### 3.8.1 Some Things to Note

1) In both formulations, there is an assumption that the model is essentially correct, that noise ‘ is additive, white, Gaussian noise (AWGN), and that the input is persistently exciting.

2) There is an assumption that the unknown parameters are changing slowly or not at all.

3) There is an underlying assumption that we can adapt parameters slowly – much more slowly than signal dynamics – but much faster than parameters can change.

4) Adaptation is done via some gradient, Newton, Newton-Raphson, etc. based on $E\left\{\varepsilon_k^2\right\}$. The simplest of these is least mean squares (LMS) [24, 55], where $\varepsilon_k$ is used as an estimate for the gradient of $E\left\{\varepsilon_k^2\right\}$. It is the slowest/simplest adaptive algorithm, and by far the most popular.

   - There is no convergence proof, but
   - it is the easiest to implement if the changes to the parameters are slow.
   - It is even the starting point for most neural network adaptive algorithms.

   The implementability was the key to it's success, and one of the two world changing things that Ted Hoff did in his career.

5) It's assumed that $\hat{B}$ & $\hat{A}$ are of high enough order to adequately model $A$ & $B$.

6) In textbooks and papers, lots of noise filters are added into the model to better account for shaped model noise.

7) Presumably, the model can be identified from test or operational data assuming persistent excitation (PE) , a term that indicates that there is enough rich signal content to adequately excite all the critical dynamics of the system. (It may be that achieving high SNR is a bigger problem that achieving PE.)

However,

a) Little structural information remains. $\hat{B}$, $\hat{A}$, $A$, & $B$ etc. are bland, polynomial form quantities, with little physical insight.

b) Because of the effects of discretization (Section 3.7), the physical parameter changes are spewed all over $A$ & $B$, and consequently even harder to extract from $\hat{B}$ & $\hat{A}$.

c) There is an inherent assumption that the sample rate is in the "Goldilocks" zone: not too slow and not too fast.

   - Too slow means little room in frequency for filtering and compensation. This is why there is a generic rule of thumb that sampling should be at 8–20× the highest relevant frequency in the system.

- Too fast and all the poles and zeros rush towards $z = 1$, making any ability to distinguish between them difficult at best. In single precision floating point or fixed point math, it means that the differences between large physical parameter ranges may be lost. Furthermore, the signal doesn't move much in any one time step, which means that the signal can be dominated by noise and or quantization.

d) Highly stimulative inputs often are not great for meeting control objectives. Meeting control objectives is often not great for getting a signal that stimulates a lot of dynamics.

e) If the physical parameters are smeared across $\hat{B}$ & $\hat{A}$, then small changes in $\hat{B}$ & $\hat{A}$ often represent large changes in physical parameters. Conversely, large changes in physical parameters may be so spread out across $\hat{B}$ & $\hat{A}$ that it may be hard to see them above the noise and quantization.

f) We have only shown the open-loop plant measurement. In many cases, the plant cannot be measured without wrapping it in a nominal feedback loop and this correlates the input noise and output noise, making the problem more difficult.

Discrete-time, time-domain identification produces a parametric model directly. Frequency response methods produce frequency response functions (FRFs) that must be curve fit to get a parametric model. We implement controllers with parametric models, but we often evaluate models/results with frequency responses.

Time domain ID often adds extra discrete-time, transfer-function type filter models, but never questions if the fundamental structure makes sense. Again, Section 3.7 should give pause.

When can discrete-time, time-domain methods work?

1) When you don't need physical understanding of the system. This is very rare for moving machinery II problems, but more common in AT and "big data" formulations.

2) When the sample rate is in the Goldilocks zone.

3) When you have great signal-to-noise ratios (SNR) and lots of bits of precision in your compensation.

4) When the dynamics are well damped and not high-Q.

5) When the underlying dynamic model is relatively low order.

6) When you can provide great input data.

### 3.8.2 When Discrete-Time, Time-Domain ID Goes Bad

If you've read to this point, you might believe that there are a lot of physical situations in which discrete-time,time-domain identification will have modest or bad results. That is, few of the parameters will be matched well and no physical intuition will be available. In this case, the designer who still needs to understand their physical system. In these cases, it is not an option to simply give up because the above textbook methods fail. Instead, we need to add in Step Response Methods (Section 3.9) and Frequency Domain Methods (Section 3.15).

## 3.9 Step Response Measurements

There is nothing like a stepped-sine for extracting high SNR Bode plots of the frequency response functions (FRFs) of complex physical systems (Section 3.15). However, lots of physical systems are not amenable to such measurements. For example:

- It is hard to do stepped-sine measurements on a functioning textile machine or batch chemical process. We cannot provide either sweepable sinusoids or random noise inputs of sufficient amplitude.
- For these, some variant of a step method using either operational inputs (inputs typical during normal operation) or some sort of special tuning inputs are used.
- What can be extracted from these is typically limited to parameters from one of our simple models.
- This means we need to assume one of these simple models, either from first principles or from measurements, and then use that model to extract parameters.
- In this tutorial, we will examine two of our simple models, the single order low pass with delay of Section 2.3.3 and the simple resonance with no zeros of Section 2.3.8 for step response methods.

Textbooks often show a system model, derive the transfer function, and then show how that model response will look in time. This section will work the other way, looking at a time response and seeing

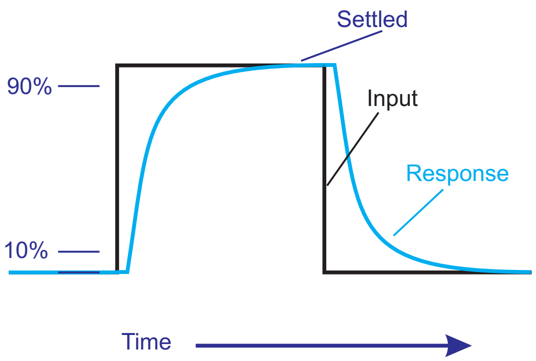


Figure 3.10: Zoomed in step response of a stable system. The behavior of this system is consistent with that of a first order system with transport delay.

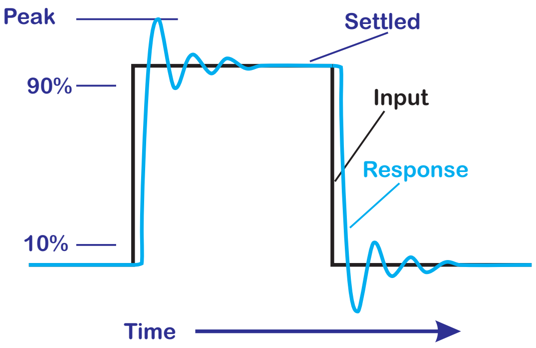


Figure 3.11: Zoomed in step response of a stable system. The behavior of this system is consistent with that of a second order system with low damping and transport delay.

what relevant time response parameters can be extracted. Specifically, we look at extracting step response parameters from the time response of the system to square wave inputs.

Textbook step response methods show a simple step and some extractable parameters, as shown in Figures 3.10 and 3.11 [56, 14, 57]. Step response measurements are useful because they can often be adapted from the normal operation of the device under test (DUT). Step changes in setpoints are typical in many systems. If one has access to the system input and output, then one can extract certain parameters from the step, among them:

- gain ($K$),
- time delay/startup time ($t_D$),
- rise time ($t_{rise}$ or $t_r$),

- settle time ($t_{settle}$), and (for second order systems with low damping)
- overshoot ($M_p$).

Under the assumption that the system is a first order low pass of the form of Equation 2.5, one can use the rise time and settle time to extract the gain, transport delay, and time constant of the system. Under the assumption that the system is a second order resonance of the form of Equation 2.19, one can use these parameters to extract such system parameters as natural frequency ($\omega_d$), damping factor ($\zeta_d$), and gain ($K$).

There are a few intuitive things that should be clear from this simple schematic:

- The system should be stable, otherwise the quantities are meaningless.
- The length of the step should be such that the response can settle.
- The height of the step must be large enough to clearly see the response through the noise, while not being so large as to saturate the Digital-to-Analog Converters (DACs) and actuators or the sensors and Analog-to-Digital Converters (ADCs).
- In generating the input signal, we have complete control over where things happen. This is the point to tag the data so that we can measure the times at which responses happen.

## 3.10 Signal Segmentation

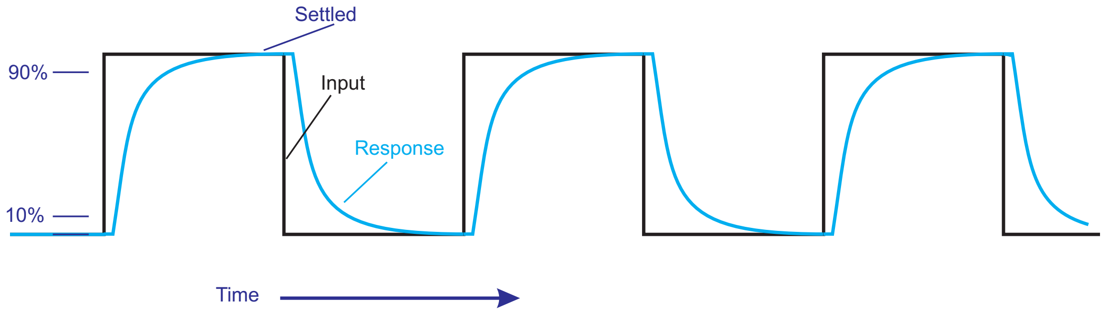


Figure 3.12: Square wave used to extract step response parameters and provide some averaging for a first order system.

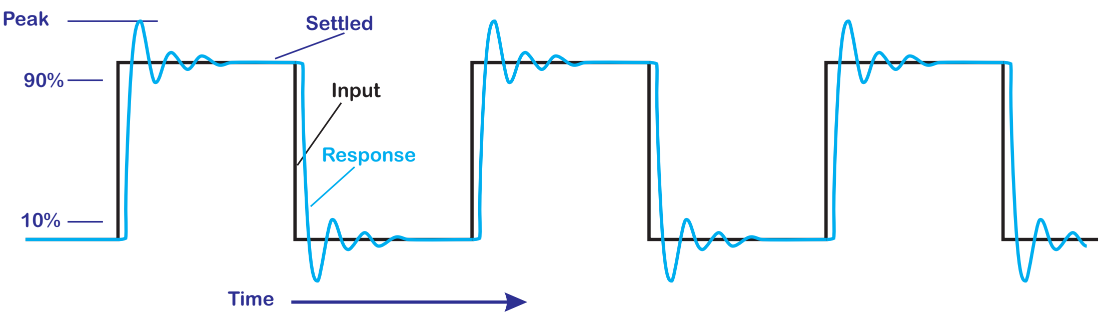


Figure 3.13: Square wave used to extract step response parameters and provide some averaging for a second order system with low damping.

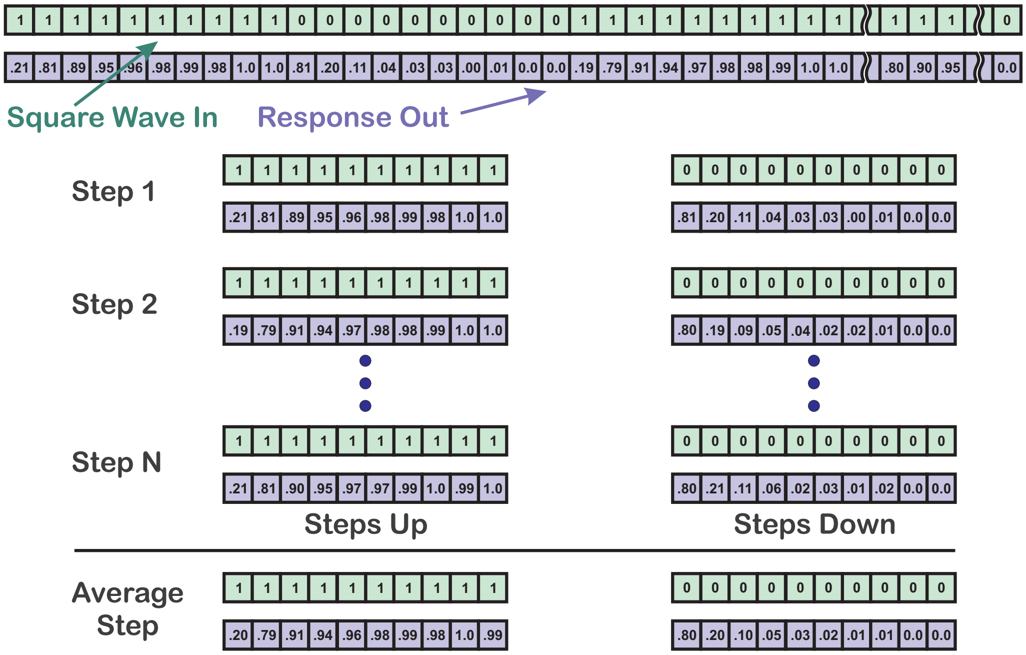
Since a step has infinite frequency at the instant of the step and zero frequency content afterwards, it can tell us a lot about the system's response. Textbooks discuss these parameters [56, 14, 57], but parameter extraction can only be managed with excellent signal-to-noise ratio (SNR). Real measurements have noise and so we need repeated steps, repeated responses, and averaging to improve our SNR. Now, in some cases, such as slow, chemical process control (CPC), thermal control, or pressure control systems, the time constants of the physical system are so slow compared to the sample rate of the measurement computer that there is plenty of data in a single step with which to average out noise effects. In some of these systems, the idea of needing to average out multiple cycles seems both unnecessary and slow.

For most other problems, a single step does not contain enough points to average out the data. The single step of Figures 3.10 and 3.11 are replaced by the square wave input and response of Figures 3.12 and 3.13. Acquiring this data only requires more computer memory than the step response, but making sense of it so that it can be averaged requires segmenting the data. While this makes intuitive sense, actually segmenting the data requires a bit more bookkeeping oriented computer programming than engineers are comfortable spending time with.

In order to average responses of a system to square wave inputs, we need to segment the data, so that each step portion and the response to that portion is treated as a cycle of a repeated signal. The segmentation is well understood when viewing averaging on a digital oscilloscopes [58, 59] where triggered signals are averaged together. Our intention is to mimic this behavior in our measurement software, but to have full access to all the data.

We start with two (or more) vectors of data, indexed by the sample time. The first is the system input, which we have defined as a square wave. The others are system responses to that square wave. We will stick with a single output measurement of the system here. We can see in the schematic drawings of Figures 3.12 and 3.13 that even in the noise free case, we are better off segmenting our data based on the input square wave signal.

Here, we will simplify the discussion by assuming that the square wave driving the system is generated by our control computer and injected into the system using a digital-to-analog-converter (DAC). We will also assume that the response is read into the computer using an analog-to-digital converter (ADC), so that both the stimulus and response share the same time base and sample rate. Furthermore, the designer has to determine if rising edge steps and falling edge steps will be treated in the same way, or if the physical system responds differently to these, requiring the data to be segmented differently. For simplicity here, we will assume that the response to a step up has the same shape as the response to a step down (albeit in the opposite direction), so that one could combine these to obtain a single set of step response parameters.



1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0

.21 .81 .89 .95 .96 .98 .99 .98 1.0 1.0 .81 .20 .11 .04 .03 .03 .00 .01 0.0 0.0 .19 .79 .91 .94 .97 .98 .98 .99 1.0 1.0 .80 .90 .95 0.0

Step 1

1 1 1 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 0 0 0

.21 .81 .89 .95 .96 .98 .99 .98 1.0 1.0 | .81 .20 .11 .04 .03 .03 .00 .01 0.0 0.0

Step 2

1 1 1 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 0 0 0

.19 .79 .91 .94 .97 .98 .98 .99 1.0 1.0 | .80 .19 .09 .05 .04 .02 .02 .01 0.0 0.0

Step N

1 1 1 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 0 0 0

.21 .81 .90 .95 .97 .97 .99 1.0 .99 1.0 | .80 .21 .11 .06 .02 .03 .01 .02 0.0 0.0

Average Step

1 1 1 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 0 0 0

.20 .79 .91 .94 .96 .98 .99 .98 1.0 .99 | .80 .20 .10 .05 .03 .02 .01 .01 0.0 0.0

Figure 3.14: Segmentation of a time measurement trace. At the top is a conceptual diagram of a square wave input data measurement. Since this is on the computer and can be cleanly recorded, we let the values be purely 0 and 1 without losing the point of the exercise. Below it is the response of what seems like a damped first order system. By indexing the data into time segments related to the input steps up (to 1) and steps down (to 0) we can also segment the output data. The output segments can then be averaged for a cleaner response from which to attempt to extract parameters.

The data segmentation is better understood by looking at Figure 3.14. Our goal is to take the long data vectors at the top and produce the repeated short data vectors that can be averaged together to minimize noise. We need to be aware of different time periods in the system. First of all, there is the sample period of our digital measurement system, $T_S = 1/f_S$, where $f_S$ is the measurement sample frequency. This has to be short enough, relative to the physical system time constants, to meet the requirements of the Nyquist sampling criterion. However, for any sort of practical system, we need $f_S$ to be 10 to 20 times as fast as the fastest dynamics we wish to identify or control. In practice, this is not much of a problem for most systems in the modern world. The explosion of incredible processing capabilities in low power and low cost packages such as the Raspberry Pi [60] and the Xilinx Zynq [61] means that real-time processing is available for even the cheapest application. The exceptions are generally high speed electronic or mechatronic systems. For most systems, even the smallest of today's processors and converters can easily sample 20–100 times as fast as the fastest dynamics of interest. We will then assume here, that the digital system can sample far, far faster than the system dynamics of interest and far, far faster than the frequency of the square wave being injected into the system.

The second time period of interest is the period of the square wave, $T_{SQ} = 1/f_{SQ}$, where $f_{SQ}$ is the frequency of the square wave input. Each period of the square wave generates a step up and a step down, and the responses to each of these is its own segment, so it is the half period of the square wave $T_{SQ}/2$ which is of greatest interest to us. We have assumed that $T_{SQ}/2 \gg T_S$. While it is not necessary, since we are generating the square wave with our digital system, we can simplify our lives by setting

$$\frac{T_{SQ}}{2} = MT_S. \tag{3.65}$$

That is, we make the square wave half period an integer multiple ($M$) of the sampling period, and make sure it is relatively long compared to that sample period, e.g. $M \geq 10$.

With this assumption, we know that we can use the edges of our square wave to accurately segment our measurement data. Furthermore, in our equations we can work with the sample indices, and factor $T_S$ back in when we need to convert data index back into actual time.

For a data vector that is $N$ samples long, assume we have found the first rising edge of the square wave at index, $c$. Then all other rising edges of the square wave should be found multiples of $2M$ away from that, so

$$Rising\_Edge(k) \text{ is at } c + 2 * M * k. \tag{3.66}$$

Likewise, falling edges will be found at the odd multiples of $M$ away from $c$, that is:

$$Falling\_Edge(k) \text{ is at } c + (2 * M + 1) * k. \tag{3.67}$$

This means that

$$Step\_Up(k) = \{data(i)\}\text{, where for } k = \{0,\ldots,N-1\} \\ c + 2*M*k \leq i < c + (2*M+1)*k, \tag{3.68}$$

and

$$Step\_Down(k) = \{data(i)\}\text{, where for } k = \{0,\ldots,N-1\} \\ c + (2*M+1)*k \leq i < c + (2*M+2)*k. \tag{3.69}$$

## 3.11 Extracting Step Response Parameters from Step Response Data

The ability to measure these quantities relies largely on knowing the parameters of the input steps. If we know those, then we can tell when the system response passed certain critical values. However, we do it, we should assume that if the system works, then the final value of the response will be system gain times the final value of the step (plus any offsets). We then draw our information from the step itself, including when the step starts and ends, what its initial value was and what its final value was.

With the data segmentation of Section 3.10 under our belts, we can now isolate on extracting step parameters from our segmented and averaged data. We will assume a few things in order to do this. There are often implicitly assumed in textbooks and by practicing engineers, but we will make an attempt to own up to them.

1) We will drive the system with a square wave to which we have complete access.

2) The system as driven is stable, so that the response to the square wave steps will damp out.

3) We are able to lengthen $T_{SQ}/2$ so that the response to each step has time to settle.

4) The data is segmented and the response to each step is averaged, as described in Section 3.10

5) One of the low order models of Section 2.3 can be assumed. In particular we will focus on the first order system in Equation 2.5 and the simple resonance with no zeros of Equation 2.19.

6) We will also assume that the responses to steps-up and steps-down can be handled together. This is not necessary, but useful in simplifying our description. We assume that the response data can be flipped and aligned so that they can be treated uniformly.

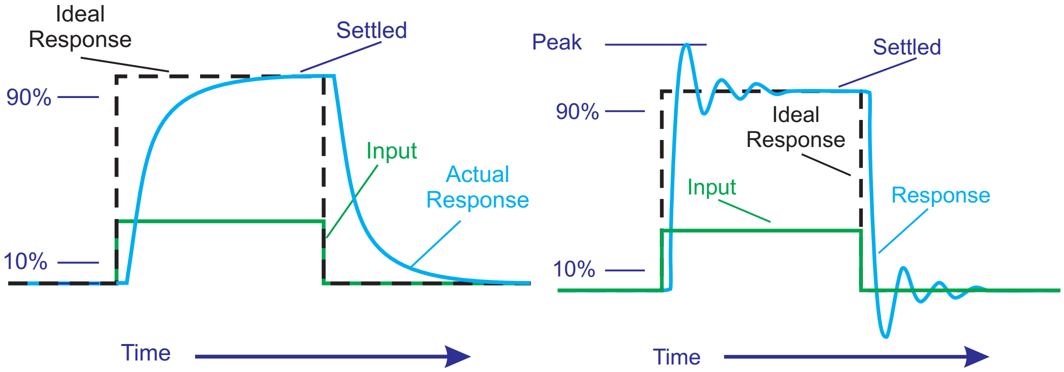


Figure 3.15: Zoomed in step response of two stable systems. The behavior of this system on the left is first order while the one on the right is consistent with that of a second order system with low damping and transport delay.

Let's consider the step responses of the two stable systems shown in Figure 3.15. Several time response parameters can be instantly measured from the step response and can be related to system parameters. Others require the assumption of a specific system model.

### 3.11.1 LTI Testing

One of the simple measures of linearity is that if one doubles the input, the output is doubled. Note that linear systems with offsets usually, technically are not called linear, but affine, since there is a portion of the response that stays constant with growing input and so the basic linearity test fails. However, most people can live productive lives without knowing or caring about that one, so we will stick to referring to those as linear with the "You know what we mean" proviso.

The first variation is to alter the height of the steps and then check the variability of the rise time, settle time, overshoot, etc. If the system were dominated by linear effects, these would not vary much. However, if the nonlinearities come into play, we will see a marked difference in these quantities, depending upon the step height. Linearity testing is fairly simple in this way, and should be a first step.

### 3.11.2 Gain

Gain involves the steady state response height over the step height. With segmented and averaged data, over the segment:

$$K = \frac{y_{ss} - y_{start}}{x_{ss} - x_{start}}. \tag{3.70}$$

For a step input, $x_{ss}$ is reached instantly with the step, so $x_{ss} - x_{start}$ is trivially the step height. To obtain $y_{ss}$ we need to take the end of the response. Since there may be noise on this response as well, we can average over the last bit of the averaged responses. For example, if we can determine that the system is settled in the last $20\%$ of the step time, then

$$y_{ss} = \frac{1}{0.2N} \sum_{k=0.8N}^{N} y(k), \tag{3.71}$$

where $N$ is the index of the last data in a segment.

### 3.11.3 Transport Delay or Startup Time

With a noise free system, transport delay is often discernible by inspection. One merely measures the time delay from the moment the input changes to the moment the output starts to respond and that is $\tau_D$. For real measurements, determining that a level is changing and not merely bumping due to noise is harder. The segment averaging helps. We can also assume that the response in the current segment was the steady state value of the final response in the previous segment, i.e.

$$y_{start}(segment_j) = y_{ss}(segment_{j-1}). \tag{3.72}$$

One simple threshold that shows up a lot is $10\%$ of the step response height. If one can measure the time from $y_{start}(segment)$ to $0.1 * (y_{ss} - y_{start}) + y_{start}$, then this is often considered a reasonable approximation for transport delay.

Of course, any threshold can be used so long as we can distinguish between system movement and noise. Increasing the number of averages in the segmentation step can significantly reduce the noise threshold.

### 3.11.4 Settle Time

Settle time can be defined as the time at which the response gets to and stays within a defined percentage of the steady state value. Again, staying with an averaged segment we define $t_{settle}$ as the first time that

$$t_{settle} = \min\{t\} \text{ s.t. } |y(t) - y_{ss}| \leq 0.1\,|y_{ss} - y_{start}|\,, \tag{3.73}$$

for 10% settle time or

$$t_{settle} = \min\{t\} \text{ s.t. } |y(t) - y_{ss}| \leq 0.01\,|y_{ss} - y_{start}|\,, \tag{3.74}$$

for 1% settle time.

The averaging of multiple segments helps reduce noise here, but since the value within the averaged segment is still changing as it is settling, it doesn't help much to average the last 20% of a response as we might do with establishing $y_{ss}$.

### 3.11.5 Overshoot

Overshoot is the peak level above the steady state response that the system achieves on a step up (or the peak level below steady state achieved on a step down). Overshoot is only defined when the response goes past the steady state value. It is a sign that a system is second order or higher and underdamped. Overshoot is defined in [56] as, $M_P$ where:

$$M_p = \left|\frac{y_{max} - y_{start}}{y_{ss} - y_{start}}\right|, \tag{3.75}$$

on a step up and

$$M_p = \left|\frac{y_{min} - y_{start}}{y_{ss} - y_{start}}\right|, \tag{3.76}$$

on a step down. We will limit ourselves to talking about the step up for now. So long as we can establish the steady state value and normalize by the height of the step response, we can easily measure overshoot if it exists.

Corresponding with overshoot is the peak time, $t_p$, is the time at which the maximum overshoot is reached.

### 3.11.6 Rise Time

Informally, rise time is the time between when the system starts responding and when it gets in the neighborhood of the steady state value. Because this is fairly non-specific, it is generally considered to be the time for the response to go from 10% of the steady state response to 90% of the steady state response. These values are typically used to avoid noise and stiction issues. Again, the multi-segment averaging can minimize the noise issues.

## 3.12 Extracting Model Parameters from Step Response Data

While gain, linearity, and transport delay are immediately available from the step response parameters, other model parameters need the assumption of a basic model to reduce the number of parameters. Only when we have done this, can we approximate model parameters from the step response parameters measured in Section 3.11. In this discussion, we will assume either a first or second order section, each of which will give rise to a different set of parameters.

## 3.13 The Mythical First Order Section

Repeating Equation 2.5 we have the transfer function of a simple first order section with transport delay:

$$\frac{X(s)}{F(s)} = \frac{Ka}{s+a}e^{-sT_D} \tag{3.77}$$

and this corresponds to a impulse response of:

$$\begin{array}{ll} h(t) = Kae^{-a(t-T_D)}, & \text{for } t - T_D \geq 0. \\ 0, & \text{otherwise.} \end{array} \tag{3.78}$$

If we have measured $K$ and $T_D$ as described earlier, we can adjust our data so that we are looking at the non delayed response,

$$\begin{array}{ll} h(t) = Kae^{-at}, & \text{for } t \geq 0. \\ 0, & \text{otherwise.} \end{array} \tag{3.79}$$

For the step response, we want to convolve the impulse response with a step,

$$\begin{array}{ll} 1(t-\tau) = 1, & \text{for } t-\tau \geq 0. \\ 0, & \text{otherwise.} \end{array} \tag{3.80}$$

It's useful to follow through with the calculation

$$1(t) * h(t) = \int_{-\infty}^{t} 1(t-\tau) K a e^{-a(\tau} d\tau, \text{ so} \tag{3.81}$$

$$\begin{aligned} 1(t) * h(t) &= Ka \int_{0}^{t} a e^{-a\tau} d\tau, & (3.82) \\ &= Ka \frac{e^{-a\tau}}{-a}\bigg|_{0}^{t} & (3.83) \\ &= K\left[1 - e^{at}\right] & (3.84) \end{aligned}$$

Since we have already extracted $K$ and $T_D$, the only parameter left to fit is $a$ which is the inverse of the system time constant. We can get $a$ from similar measurements used to get the rise time. The standard rise time is defined as:

$$t_r = t_{90} - t_{10}. \tag{3.85}$$

Given that we have determined $K$, measuring the time at which the response is at 10% and 90% of steady state means that if we assume an input step is applied at $t = 0$ and $u(t_{0+} - u(t_{0-}) = u_{step}$, then

$$y(t_{90}) - y(0) = 0.9Ku_{step} = K\left[1 - e^{-at_{90}}\right] u_{step} \tag{3.86}$$

and

$$y(t_{10}) - y(0) = 0.1Ku_{step} = K\left[1 - e^{-at_{10}}\right] u_{step} \tag{3.87}$$

we can back out two estimates of $a$, one for the 90% rise time and one for the 10% rise time:

$$\begin{aligned} 0.9 &= \left[1 - e^{-at_{90}}\right] & (3.88) \\ 0.1 &= e^{-at_{90}} & (3.89) \\ a &= \frac{-\ln(0.1)}{t_{90}} & (3.90) \end{aligned}$$

$$\begin{aligned} 0.1 &= \left[1 - e^{-at_{10}}\right] & (3.91) \\ 0.9 &= e^{-at_{10}} & (3.92) \\ a &= \frac{-\ln(0.9)}{t_{10}} & (3.93) \end{aligned}$$

If our measurements have been carefully made and the segments averaged, these two estimates should agree very well.

This relatively simple set of step response measurements and calculations allow us to extract the relevant system parameters from our first order model. A lot of systems involving thermodynamics (e.g. temperature control, HVAC) and chemical process control are adequately modeled this way. Furthermore, many of these systems do not easily lend themselves to frequency response function measurements, and so really this type of combined measurement and analysis is the best we can practically do.

The next section will move on to a simple second order model that describes some of our simpler electrical, mechanical, and combined (mechatronic) systems.

## 3.14 The Mythical Second Order Section

Most of the parameters that one computes from a step response are predicated on the system behaving like a second order section. The reason we do this is – as George Carlin would say – is because we can. A second order section has a very nicely defined response to a unit step. Because of that, parameters such as rise time, settling time, and overshoot are easy to compute analytically. Furthermore, these can be tied to parameters of the second order section.

We often have something with far more dynamics than such a second order section. Furthermore, the system might have some nonlinearity, might be discretized, might be noisy. With all that being said, there is still a lot that can be gained from the intuition of analyzing the response of such a section. Consider the classic second order linear section with transfer function:

$$\frac{X(s)}{F(s)} = H(s) = K\frac{\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2} \tag{3.94}$$

In this section, we will borrow heavily from the analysis of Franklin, Powell, and Emami [56], but we will come at the problem from a different perspective. In the book, the idea was to take the transfer function and relate model parameters to step response parameters. In our analysis, we will work the problem backwards, taking the step response parameters and try to extract model parameters. Furthermore, we will stick with the notation of Section 2.3, rather than that of [56]. Thus, $\omega_d$ will represent the undamped natural frequency of the denominator, and we will select $\omega_p$ to signify the damped natural frequency of the denominator. $\zeta_d$ will be the damping factor for the denominator.

### 3.14.1 Gain from Step Response

If $H(s)$ is stable, then its response to a unit step is given by the inverse Laplace transform of $H(s)/s$ [56]:

$$y(t) = K\left[1 - e^{-\sigma_d t}\left(\cos\omega_p t + \frac{\sigma_d}{\omega_d}\sin\omega_p t\right)\right], \tag{3.95}$$

where $\omega_p = \omega_d\sqrt{1-\zeta_d^2}$ and $\sigma_d = \zeta_d\omega_d$.

Looking at Figure 3.11 which is a zoomed in version of Figure 3.13, we can relate various quantities that define a step response to Equation 3.95. Even when our system isn't second order, continuous time, and linear, we can still use the intuition supplied by these quantities.

It is worth noting that the final value of $y(t)$ can be inferred from the final value theorem (FVT), i.e.

$$\begin{aligned}
\lim_{t\to\infty} y(t) &= \lim_{s\to 0} s\frac{H(s)}{s} && (3.96)\\
&= \lim_{s\to 0} H(s) && (3.97)\\
&= \lim_{s\to 0} \frac{K\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2} && (3.98)\\
&= \frac{K\omega_d^2}{\omega_d^2} && (3.99)\\
&= K && (3.100)\\
& && (3.101)
\end{aligned}$$

So, because this second order section happens to have a DC gain of K, the final value of the step response will be that step value. The point of dealing with the second order section is that for such an idealized system, these quantities all have well defined analytical expressions, related to the parameters of Equations 3.94 and 3.95.

For a middle level of damping, $\zeta_d = 0.5$

$$t_r \approx \frac{1.8}{\omega_p}. \tag{3.102}$$

Note the condition on $\zeta_d$ for this to hold. The rise time number is not a particularly good estimate.

### 3.14.2 Extracting Data from Ringing

If $\zeta_d$ is small enough, then the system will oscillate around $y_{ss}$ before settling down. The response, $y(t)$ will cross $y_{ss}$ and intervals of $T_p/2$ where

$$T_p = \frac{1}{f_p} \text{ and} \tag{3.103}$$

$$f_p = \frac{\omega_p}{2\pi} \text{ so that} \tag{3.104}$$

$$\omega_p = \frac{2\pi}{T_p}. \tag{3.105}$$

We can measure $T_p$ from the response, and then back out the damped natural frequency. Thus, from here, we can extract $\sigma_d = \zeta_d \omega_d$. The problem is that we need $\zeta_d$. If we had this, we could also extract $\omega_d$ from $\omega_p = \omega_d\sqrt{1-\zeta_d^2}$. Fortunately, for our model, if $\zeta_d$is small enough, we can extract it from the measurement of overshoot.

### 3.14.3 Extracting Data from Overshoot

For our measurement purposes, if we can measure the overshoot, $M_p$, we can back out $\zeta_d$. Looking at Equation 3.95, we can find $t_p$ and $M_p$ by looking for the points where $\dot{y}(t) = 0$.

$$\begin{aligned}
\dot{y}(t) &= K\left[\sigma_d e^{-\sigma_d t}\left(\cos\omega_p t + \frac{\sigma_d}{\omega_p}\sin\omega_p t\right)\right. \\
&\quad \left. -e^{-\sigma_d t}\left(-\omega_p \sin\omega_p t + \sigma_d \cos\omega_p t\right)\right] = 0, &(3.106)\\
&= e^{-\sigma_d t}\left(\frac{\sigma_d^2}{\omega_p}\sin\omega_p t + \omega_p \sin\omega_p t\right) = 0, &(3.107)
\end{aligned}$$

This occurs when $\sin\omega_p t = 0$, so

$$\omega_p t_p = \pi, \tag{3.108}$$

or

$$t_p = \frac{\pi}{\omega_p}. \tag{3.109}$$

Substituting (3.109) into (3.95), we get

$$\begin{aligned}
y(t_p) &= K(1+M_p) &(3.110)\\
&= K\left[1 - e^{-\sigma_d \pi/\omega_p}\left(\cos\pi + \frac{\sigma}{\omega_p}\sin\pi\right)\right] &(3.111)\\
&= K\left[1 + e^{-\sigma\pi/\omega_p}\right]. &(3.112)
\end{aligned}$$

So

$$M_p = e^{-\pi\zeta_d / \sqrt{1-\zeta_d^2}}. \tag{3.113}$$

From here

$$\ln M_p = -\frac{\pi\zeta_d}{\sqrt{1-\zeta_d^2}}, \tag{3.114}$$

so let

$$L = -\frac{\ln M_p}{\pi} = \frac{\zeta_d}{\sqrt{1-\zeta_d^2}}. \tag{3.115}$$

Now

$$L^2 = \frac{\zeta_d^2}{1-\zeta_d^2}, \tag{3.116}$$

which means

$$\zeta_d^2 = \frac{L^2}{1+L^2}. \tag{3.117}$$

Since our system is stable, $\zeta_d$ is the positive root, and thus:

$$\zeta_d = \sqrt{\frac{L^2}{1+L^2}} \tag{3.118}$$

where we compute $L$ from (3.115).

Of course, all of this depends upon there being overshoot. Looking at the problem from the other side, what values of $\zeta_d$ provide measurable levels of $M_p$? First of all, we are only considering values of $\zeta_d$such that $0 \leq \zeta_d < 1$. For $\zeta_d \geq 1$, the poles are real and so we get no oscillatory behavior. For $\zeta_d < 0$ the poles are unstable. When $\zeta_d$ is very close to $0$, $M_p$ is close to $1$ while as $\zeta_d$ gets close to $1$, $M_p$ drops to $0$. We can evaluate Equation 3.113 for different values of Blue$\zeta_d$:

| $\zeta_d$ | $M_p$ |
|---|---|
| 0.8 | 0.02 |
| 0.7 | 0.05 |
| 0.6 | 0.09 |
| 0.5 | 0.16 |

It is unlikely that overshoot below 5% will be detectable, so we really are talking about needing to have $\zeta_d \leq 0.7$ to make use of any of these techniques from measured data. For the damping factor above $0.7$ we are unlikely to find any discernible overshoot, and without that, we won't be detecting $y(t)$ crossing $y_{ss}$.

### 3.14.4 Extracting Data from Settling Time

If we define settling time as $t_s$ being the time from the application of the step until the response stays within a certain percentage of the final steady state value, then we can look at the envelope of the decaying exponential given by

$$v(t) = e^{-\sigma_d t} = e^{-\zeta_d \omega_d t} \tag{3.119}$$

The settling time for a given amount, $\delta$, is when $v(t) \leq \delta$. So for the 1% settling time,

$$e^{-\zeta_d \omega_d t_{s,1\%}} = 0.01, \tag{3.120}$$

or

$$\zeta_d \omega_d t_{s,1\%} = -\ln(0.01) = 4.6, \tag{3.121}$$

so

$$t_{s,1\%} = \frac{-\ln(0.01)}{\zeta_d \omega_d} \approx \frac{4.6}{\zeta_d \omega_d}. \tag{3.122}$$

Likewise,

$$t_{s,10\%} = \frac{-\ln(0.1)}{\zeta_d \omega_d} \approx \frac{2.3}{\zeta_d \omega_d}. \tag{3.123}$$

## 3.15 Frequency Response Measurements

The prior sections dealing with step response methods are useful for a wide variety of practical systems, but they also expose the great limitation of these methods: they only work on low order models, or on the low order portion of a more complex model. For any systems in which the behavior of higher order dynamics need to be identified, step response methods fall short. There is a great amount of literature in system identification using time domain methods [44, 49, 45]. These mostly focus on identifying discrete time system models based on time domain measurements. They have great utility in certain circles, but in order to be practical, one must be able to map the physical parameters to the identified discrete model parameters (and back) and guarantee that the inputs are rich enough in content to provide adequate signal-to-noise ratio (SNR) for the multi-parameter regression. The discussion of why this might be difficult follows the path outlined in Section 3.7. We will skip this set of measurements in this tutorial since they are often difficult to apply if one wishes to extract the physical parameters of models.

An alternative is to stimulate the system with some sort of excitation signal, calculate the frequency response function (FRF) from the input-output properties, and then fit a parametric model to that response. For a large set of systems, especially mechatronic systems with lightly damped dynamics, this is really the only way to avoid the limitations of the prior methods. This section will focus on the essentials of doing frequency response measurements.

### 3.15.1 Practical Limits on Frequency Response Methods

Frequency response methods are quite common in mechanical, electrical, and electro-mechanical or mechatronic systems. At the same time, it is rarely practical for slow process systems, such as thermal, pressure, biological, or chemical process control (CPC). There might be several reasons for this, but one major one is time constants. Frequency response methods looking at the system over some span of frequencies and when the dominant time constants of a system are in seconds or minutes, the integration time for the measurements described in the sections that follow make these tools a lot less practical.

In computing a frequency response, we want to evaluate the system response to signals over a broad frequency range, typically from a factor of 10 below to at least a factor of 10 above the key frequencies of the system. In order to compute the Fourier transform of any experimental system, we need to integrate over at least 1 and probably more periods of each frequency component. The Fast Fourier Transform (FFT) still has this integral underlying its computation, although certain assumptions are made to make the actual computation of the integral far easier for limited computer power. Still, the frequency width that one can resolve is dependent on the integration time: the longer the integral, the narrower the resolvable bandwidth.

What this means is that if a system has time constants that are on the order of a minute, then to get a well resolved frequency response requires integrals that are on the order of 10 minutes to have any chance of resolving the lowest frequency, significantly longer if we wish to have a more accurate integral. Even FFT calculations which tend to be faster are averaged on the order of 10–100 times to decrease the effects of noise. Thus, the time required to compute a lot of frequency domain measurements is so long as to make them impractical for many biological and chemical process control (CPC) problems.

Even if this were not the case, many of the measurements described below involve actively injecting a non-operational signals at the input of the system. With electrical, mechanical, and mechatronic systems, these input output measurements can be done at non-operational times and have no real effect on the final product. This means we can at test time shake the system with signals of our choosing in order to get rich set of output signals from which to calculate our responses. Once the system is in operation, we have to be far more judicious about what signals we inject and our identification functions are often done by passively monitoring operational input and output signals and working from those.

In the case of a large reactor that might be used in CPC, it is not clear that we can fill it with our own meaningless inputs (some chemicals) and make measurements off of some output based on that (other chemicals) in any practical way. (It's a lot of waste chemicals, which are a lot more of an issue than waste voltages or waste vibrations.) Our identification then must happen during normal operations, and injecting some variation to the product input would result in waste product at the output. This is usually seen as financially and environmentally unacceptable.

Finally, one can argue that frequency domain methods are limited by nonlinearities. This is partly, but not completely true. It is true that the one-to-one relationship between time domain and frequency domain models only exists when the model is a linear, time-invariant differential (or difference) equation. However, this does not mean that we cannot get useful insight from frequency domain analysis of systems that are – as Miracle Max from The Princess Bride would call them – mostly linear. Methods such as describing functions [13] lead to measurement techniques such as the swept-sine, describing function method [62, 63]. While these methods are not exact, and the frequency responses are amplitude dependent, they yield a large amount of usable insight into the system. Besides, if any nonlinearity would make frequency domain methods unusable, the first analog-to-digital (ADC) or digital-to-analog (DAC) converter in a system would eliminate FFTs forever. This clearly is not the case, so it is really a matter of what usable insight we can get from incomplete and always slightly flawed measurements.

### 3.15.2 Clearing Up Some Frequency Response Terminology

A bit of terminology is useful here. The terms transfer function (TF) and frequency response function (FRF) are often used interchangeably in the literature. Sometimes, the term Empirical Transfer Function Estimate (ETFE) is used [45]. For this tutorial, and for any of the author's writings, the term transfer function (TF) will be used only to describe an analytic function between input and output of some model, where the function is in the form of some frequency variable, typically $s$, $j\omega$, or $f$ for continuous time systems described by a linear-time-invariant differential equation and $z$ or $q$ for a discrete time system described by a linear-time-invariant difference equation. On the other hand, a frequency response function (FRF) is a set of ordered pairs of frequency variables (real numbers) and complex responses of the system evaluated at that frequency. A TF will look like some sort of rational function (fraction) of polynomials in the frequency variable and a FRF will be two or more columns of numbers. For most of this tutorial we will stick with frequency response functions in continuous-time, and specifically those obtained by a Fourier Transform, because they can be easily mapped back to Laplace Transforms, by replacing $j\omega$ by $s$ in the derived transfer functions. Mapping back to discrete-time $z$ transforms takes more work.

Converting from a TF to an FRF is fairly simple. One merely evaluates the TF at the desired frequency values. The reverse direction, converting FRFs to TFs is far more difficult. There has been much written about this, but the difficulty of this step on certain systems is the main reason why frequency response methods are less popular in some fields [64, 65, 66, 67]. What is clear to experienced engineers is that from an accurate FRF, one can extract many physical model parameters. The issue becomes doing this in an automated way.

The sections that follow will describe ways of generating that accurate frequency response function measurement.

### 3.15.3 A Note on Notation

There is a notational issue that must be dealt with in order to make the text and diagrams easier to follow. By convention, when we draw block diagrams, we are generally filling in the boxes with symbols for the frequency domain quantities e.g. Figure 3.16, in which $P$ is the physical system or plant model and by convention, the capital letter indicates this model to be a transfer function. IF we are to be mathematically precise, the signals into and out of this block should also be uppercase. Into a Transfer Function block, $P$, we inject the frequency domain input and noise ($U$ and $W$) and read an output that is a combination of the filtered input and sensor noise ($V$), i.e.

$$Z(s) = P(s)U(s) + P(s)W(s) + V(s). \tag{3.124}$$

However, the loop algebra works for any frequency variable, so long as we are self consistent, so once we've established what we need to establish about which frequency domain we are in, we often drop the frequency variable from the equations (as long as it's clear what our meaning should be). From that we get the simpler:

$$Z = PU + PW + V. \tag{3.125}$$

What I have found is that the readability of both the block diagrams and the equations becomes much clearer (assuming that it is obvious in which domain we are operating), if the frequency domain signals revert to lower case while the model blocks remain in upper case, i.e.

$$z = Pu + Pw + v. \tag{3.126}$$

Purists might argue that this is bad notation as we do not explicitly distinguish between the $z$ variable that is a time or space domain quantity and the $z$ variable that is a frequency domain quantity. However, if it is clear from the context what domain the variable occupies, then we are better off with a notation that is easy to parse. We will have plenty of opportunities to generate errors without having hard to parse notation.

## 3.16 Frequency Response Options

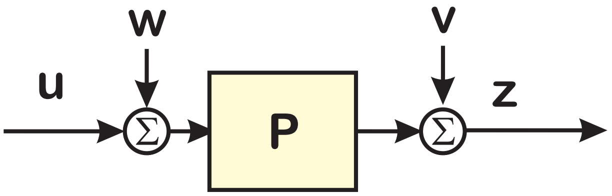


Figure 3.16: Basic “device under test (DUT)” view of frequency response function (FRF) measurement. The deterministic input to the physical system, $P$, is given by $u$, but this is by input noise, $w$, which is often called process noise in the literature. The pure output of the physical system is given by $y$, but we do not have access to this, only a noisy version, $z$, corrupted by measurement noise, $v$. Having access to only $u$ and $z$, our job is to extract a usable model of $P$.

From a test signal, a system or device under test, disturbed by input noise, $w$, and output noise, $v$, we would like to extract a model, in the case of Figure 3.16, a continuous-time transfer function, $P(s)$. Frequency response function (FRF) measurements transform the input and output into a frequency response. A further step (curve fitting) is needed to transform this into a parametric system model. At a fundamental level, FRF measurements allow for much more complex models than step response measurements, depending upon the richness of the input signal, and the accuracy of the measurement and post processing. However, with enough frequency content in the input signal, $u$, and enough signal-to-noise (SNR) in the output signal, $z$, the right processing can reveal an accurate model for a fairly complex system.

The number and complexity of these operations has led to the invention of pushbutton instruments, called Dynamic Signal Analyzers (DSAs), to handle them [53, 68, 69, 70, 71]. DSAs are essentially low frequency network analyzers, specifically adjusted for dynamic system measurements. The operation of DSAs makes it relatively straightforward to extract accurate FRFs of many systems, and sometimes straightforward to extract parametric data from those, but it is worth understanding the mathematical basis of the operations done by these devices [70, 72, 73, 74].

Looking at just the plant to measurement of Figure 3.16 we can measure:

$$Z(f) = P(f)U(f) + P(f)W(f) + V(f). \tag{3.127}$$

Here $Z(f)$ and $U((f)$ are the Fourier transforms of signals $z$ and $u$, respectively. $W(f)$ and $V(f)$ are the Fourier transforms of the noise terms, $w$ and $v$, and while they usually cannot be measured directly, their signal properties can be usually estimated.

Note that we are using $f$ as a generalized frequency variable. The loop algebra does not change whether the frequency domain is Fourier ($j\omega$), Laplace ($s$), Z ($z$ or $q$), or some other frequency representation. The frequency domain allows the convolution integrals/sums of systems interacting with a dynamic system to be turned into the product of their respective transforms.

In practice, it is easiest to compute Fourier Transform estimates from data. We will discuss that in greater detail in the subsections of Section 3.19. However, if the signal is corrupted with noise, then the transformed signal will be corrupted with the transform of the noise. Less experienced users are often tempted to extract $P(f)$ from

$$P(f) = \frac{Z(f)}{U(f)}, \tag{3.128}$$

however, this is susceptible to noise. That is:

$$\frac{Z(f)}{U(f)} = P(f) + P(f)\frac{W(f)}{U(f)} + \frac{V(f)}{U(f)}. \tag{3.129}$$

If we have excellent signal-to-noise ratios (SNR), i.e. if $U(f) \gg W(f)$ and $U(f) \gg V(f)$ in the frequencies, $f$, of interest, then Equation 3.129 certainly provides an easy way to extract the desired physical system FRF, $P(f)$ . However, while we can consider $U((f)$ to be noise free, $Z(f)$ will be corrupted by $V(f)$ resulting in a bias of the response, which show up in the last two terms of the right hand side of Equation 3.129.

Most experienced engineers and all dynamics analyzers instead do an operation based on the following observation. Since $w$, $v$, and $u$ are almost certainly uncorrelated, the relationship in Equation 3.133 will yield an unbiased response.

We start by computing the cross spectra and take the expectation [75]. That is we compute the spectra of the individual signals and multiply these spectra times either their own complex conjugate (for auto-spectra) or the complex conjugate of another signal (for cross-spectra). For measured frequency response functions, we are doing these computations one frequency at a time, so even though the frequency responses are each a pair of vectors, the computation of the cross and auto spectra are done on the scalar signals at any given frequency.

Expectation is a harder thing, since we are not going to compute an infinite number of averages. Instead, we pick a finite number of averages. In low noise environments, this number can be as low as 3. In higher noise environments the number can be much higher, even in the hundreds. There is a tradeoff between how much noise can be limited in Fourier calculation and how many averages we need to do. However, when we discuss "expectation" here for a physical measurement, then we really mean some finite number of averages.

It is important to note below that we average the cross and auto spectra, rather than computing the cross and auto spectra of averaged signal Fourier Transforms. This allows us to take advantage of signals being uncorrelated to drive the expected value of their cross spectra to $0$.

$$E\{Z(f)U^*(f)\} \quad = \quad P(f)E\{U(f)U^*(f)\}. \tag{3.130}$$

We assume that the noise, $w$ and $v$ is typically uncorrelated with $u$ (with the loop open).

Practically, the single-sided auto or cross spectrum is used [75], rather than the two-sided spectrum from theory. The two-sided spectrum is denoted here by $S_{ab}$ and the single sided spectrum is denoted by $G_{ab}$, where

$$G_{ab}(f) \quad = \quad \begin{cases} 2S_{ab}(f) & f > 0 \\ S_{ab}(f) & f = 0 \\ 0 & f < 0 \end{cases} . \tag{3.131}$$

or in terms of our variables:

$$G_{zu}(f) \quad = \quad \begin{cases} 2E\{Z(f)U^*(f)\} & f > 0 \\ E\{Z(f)U^*(f)\} & f = 0 \\ 0 & f < 0 \end{cases} . \tag{3.132}$$

which we can obtain the frequency response function, $P(f)$ , from

$$P(f) = \frac{G_{zu}(f)}{G_{uu}(f)} \tag{3.133}$$

where

$$G_{zz}(f) \quad = \quad \begin{cases} 2E\left\{Z(f)Z^*(f)\right\} & f > 0 \\ E\left\{Z(f)Z^*(f)\right\} & f = 0 \\ 0 & f < 0 \end{cases} \tag{3.134}$$

and

$$G_{uu}(f) \quad = \quad \begin{cases} 2E\left\{U(f)U^*(f)\right\} & f > 0 \\ E\left\{U(f)U^*(f)\right\} & f = 0 \\ 0 & f < 0 \end{cases} . \tag{3.135}$$

Note that this depends on expectation which means that we need to be careful about how we average our measured signals. This will be discussed in Section 3.23. In actual measurements, these transforms will be replaced by time-dependent estimates of the Fourier transforms [75]. Let's at least say at this point is that when we say these signals are independent, it means that the Expected Value of their cross correlation or cross spectra is zero. Approximating any sort expected value requires multiple measurements and averaging. In other words, we need to average data from more than one measurement to be able to claim some sort of independence. Furthermore, we need to do the averaging on the auto and cross spectra before we do the divisions of spectra. This allows the uncorrelated signals to have their cross spectra go to zero.

This result comes from the fact that if $P$ is linear then

$$\begin{aligned} G_{zu}(f) &= 2E\left\{Z(f)U^*(f)\right\}, & (3.136) \\ &= P(f)2E\left\{U(f)U^*(f)\right\}, & (3.137) \\ &= P(f)G_{uu}(f). & (3.138) \end{aligned}$$

and

$$\begin{aligned} G_{zz}(f) &= 2E\left\{Z(f)Z^*(f)\right\}, & (3.139) \\ &= P(f)P^*(f)2E\left\{U(f)U^*(f)\right\}, . & (3.140) \\ &= P(f)P^*(f)G_{uu}(f). & (3.141) \end{aligned}$$

## 3.17 The Coherence Function

The coherence function is given by

$$\gamma^2(f) = \frac{G_{zu}(f)G_{uz}(f)}{G_{uu}(f)G_{zz}(f)} \quad \text{where} \quad G_{zu}(f) = G^*_{uz}(f) \tag{3.142}$$

or

$$\gamma(f) = \sqrt{\frac{G_{zu}(f)G_{uz}(f)}{G_{uu}(f)G_{zz}(f)}} \text{ where } G_{zu}(f) = G_{uz}^*(f) \tag{3.143}$$

and gives an indication of how much of the output is generated from the input [75]. The coherence function is an excellent figure of merit, and is limited below by $0$ and above by $1$. If the output, $y$, is entirely caused by the input then $\gamma^2(f) = 1$. The presence of noise or nonlinearities will cause the coherence to be less than $1$. For any FRF measurement, $\gamma(f)$ tells us how much we can trust the measurement.

Consider Equations 3.138 and 3.141. From (3.138) we get that

$$G_{zu}(f)G_{uz}(f) = P(f)P^*(f)G_{uu}(f)G_{uu}^*(f) = P(f)P^*(f)G_{uu}^2(f), \tag{3.144}$$

and from (3.141) we get that

$$G_{zz}(f)G_{uu}(f) = P(f)P^*(f)G_{uu}^2(f), \tag{3.145}$$

so that calculated this way:

$$\gamma^2(f) = \frac{G_{zu}(f)G_{uz}(f)}{G_{uu}(f)G_{zz}(f)} = \frac{P(f)P^*(f)G_{uu}^2(f)}{P(f)P^*(f)G_{uu}^2(f)} = 1. \tag{3.146}$$

However, when the noises and signals are not uncorrelated, the numerator will generally be smaller than the denominator. This happens when we try to extract FRFs from signals inside a feedback loop, or when there are unaccounted for noises or nonlinearities in the system. To the extent that $\gamma(f) \approx 1$, we can say that the output is a linear, noise free, filtered representation of the input.

## 3.18 Closed-Loop Measurements: Two vs. Three Wire

It would be great if we could simply measure systems in open loop, as depicted in Figure 3.16. However, it turns out that a lot of interesting systems cannot be measured without the presence of a feedback controller, and so we often have to make measurements in the context of a closed-loop system, as shown in Figure 3.17. In this case, the user has to chose between two possible non-ideal measurements. The first, known as a three-wire measurement is one in which signal is injected at one of the input points ($r$, $n_1$, or $n_2$) and the open loop plant FRF ($P(f)$) is estimated using measurements of two signals within the loop $u$ and $z$. The issue here is that because of feedback, both $u$ and $z$ will have correlated noise in them. On the other hand, a two-wire measurement is a measurement of

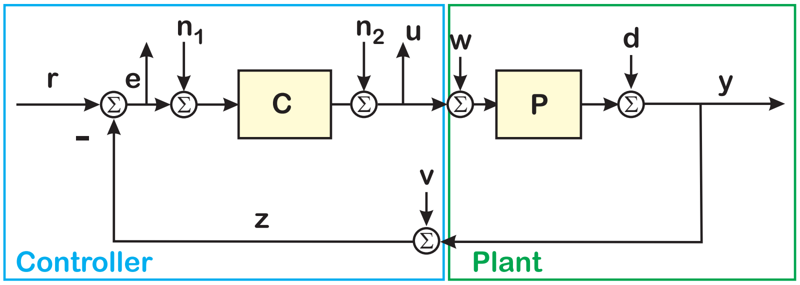


Figure 3.17: More complex view of FRF measurement inside of closed-loop system with noise added in. The process noise and measurement noise are signified by $w$ and $v$, respectively, while an unmeasured disturbance that affects the output is called $d$.

the closed-loop response and in this case the signal is injected as before, but the closed-loop FRF is extracted from measurements of say, $r$ and $z$ or $r$ and $e$. This closed-loop measurement is a ratio of a closed-loop signal to an external reference, where as the three-wire measurement involves the ratio of two closed-loop signals. The issue for the two wire measurement is that the FRF must be opened in order to calculate an open loop FRF. We could measure

$$T_{cl} = \frac{Y}{R} = \frac{Y}{N_1} = \frac{PC}{1+PC} \text{ and/or} \tag{3.147}$$

$$S_{cl} = \frac{E}{R} = \frac{E}{N_1} = \frac{1}{1+PC}. \tag{3.148}$$

Also, the compensator, $C$, can be measured from available signals $E$ and $U$, as:

$$C = \frac{U}{E}, \tag{3.149}$$

(or calculated based on a model of $C$), and the open loop FRF can be obtained from measurements of closed loop quantities by means of

$$PC = \frac{T_{cl}}{1-T_{cl}} \tag{3.150}$$

or

$$PC = \frac{1}{S_{cl}} - 1. \tag{3.151}$$

From an “opened” closed-loop measurement as in (3.150) and a measurement or model of $C$, we can get our plant response, $P$. Clearly, many combinations of measurements are possible and as long as there is sufficient signal to noise in the measurement, loop manipulations can be done to extract needed responses [76, 77].

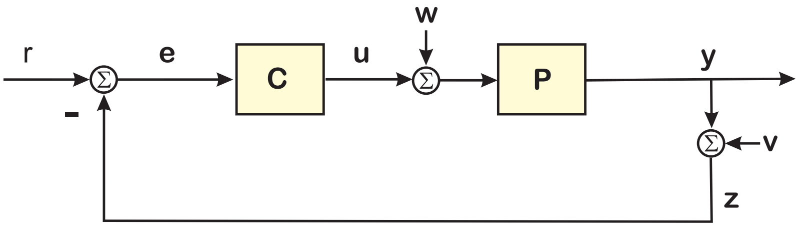


Figure 3.18: Structure for discussing closed-loop measurements. We have dropped the system annotations from Figure 3.17, as well as the disturbance, $d$. The process noise and measurement noise are signified by $w$ and $v$, respectively.

In discussing closed-loop measurements, we will simplify Figure 3.17 down to that of Figure 3.18. The measured output, $z$ is again defined by

$$z = Pu + Pw + v, \tag{3.152}$$

where $w$ is the process noise that affects the system response and $v$ is the measurement noise that affects the sensing of the signal. The physical system again is denoted by $P$, and the feedback controller is denoted by $C$. The feedback controller computes the input, $u$, as $Ce$ where $e = r - z$.

$$\begin{aligned} u &= Ce && (3.153)\\ u &= C(r-z) && (3.154)\\ u &= Cr - CPu - CPw - Cv && (3.155)\\ (1+CP)u &= Cr - CPw - Cv. && (3.156) \end{aligned}$$

If we stick with Single-Input, Single-Output (SISO) systems,

$$(1+PC)u = Cr - PCw - Cv \tag{3.157}$$

so that

$$u = \frac{C}{1+PC}r - \frac{PC}{1+PC}w - \frac{C}{1+PC}v. \tag{3.158}$$

From Equations 3.152 and 3.158, we can compute $z$:

$$z = \frac{PC}{1+PC}r - P\frac{PC}{1+PC}w - \frac{PC}{1+PC}v + Pw + v, \tag{3.159}$$

which leads to:

$$z = \frac{PC}{1+PC}r + P\left(1 - \frac{PC}{1+PC}\right)w + \left(1 - \frac{PC}{1+PC}\right)v, \tag{3.160}$$

and finally,

$$z = \frac{PC}{1+PC}r + \frac{P}{1+PC}w + \frac{1}{1+PC}v. \tag{3.161}$$

Now, we are still trying to measure the plant FRF, $P$. We have two choices:

- We can measure the closed loop response from Equations 3.147 or 3.148 and back out the open loop FRF, $PC$ from Equations 3.150 or 3.151. We then use the measurement of $C$ in Equation 3.149 to end up with $C$.
- Alternately, we can simply extract
$$P \approx \frac{G_{zu}}{G_{uu}}.$$

To see which of these is better, we need to compute the cross and auto spectra as we did in the open loop case. The math below really applies to generate the dual sided spectra, $S_{zr}$, $S_{rr}$, $S_{zu}$, and $S_{uu}$, but we will immediately go from there to the single sided spectra and so we will use the terms, $G_{zr}$, $G_{rr}$, $G_{zu}$, and $G_{uu}$. We need to compute these and the simplest one is $G_{zr}$:

$$zr^* = \left(\frac{PC}{1+PC}\right)rr^* + \left(\frac{P}{1+PC}\right)wr^* + \left(\frac{1}{1+PC}\right)vr^*. \tag{3.162}$$

Taking the expected value,

$$E\left\{zr^*\right\} = G_{zr} = \frac{PC}{1+PC}G_{rr}, \tag{3.163}$$

because the reference input is uncorrelated with the noise. Thus,

$$\frac{G_{zr}}{G_{rr}} = \frac{PC}{1+PC} = T_{cl}. \tag{3.164}$$

Notice that the effects of noise are negligible, assuming that we have done enough averaging so that the expected values start to converge. For FFT based measurements, the number of averages needed can be anywhere from $10$ to $100$, while for stepped-sine measurements, the SNR is so much higher that anywhere from $3$ to $10$ averages are often suitable.

$$zu^* = \left(\frac{PC}{1+PC}\right)ru^* + \left(\frac{P}{1+PC}\right)wu^* + \left(\frac{1}{1+PC}\right)vu^*, \tag{3.165}$$

$$zu^* = \frac{PC}{1+PC}r\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right], \tag{3.166}$$

$$+ \frac{P}{1+PC}w\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right], \tag{3.167}$$

$$+ \frac{1}{1+PC}v\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right]. \tag{3.168}$$

The common denominator ends up as

$$(1+PC)\,(1+PC)^* = \|1+PC\|^2\,, \tag{3.169}$$

so that

$$\begin{aligned} zu^* &= \frac{PCr\,[Cr^* - PCw^* - Cv^*]}{\|1+PC\|^2} \\ &+ \frac{Pw\,[Cr^* - PCw^* - Cv^*]}{\|1+PC\|^2} \\ &+ \frac{1v\,[Cr^* - PCw^* - Cv^*]}{\|1+PC\|^2}. \end{aligned} \tag{3.170}$$

When we take the expectation, the noises being uncorrelated from the reference means that:

$$E\,\{zu^*\} = G_{zu} \tag{3.171}$$

$$G_{zu} = \frac{P\,\|C\|^2\,G_{rr} - C\,\|P\|^2\,G_{ww} - C^* G_{vv}}{\|1+PC\|^2}. \tag{3.172}$$

Similarly, we can compute $G_{uu}$ via:

$$uu^* = \left(\frac{C}{1+PC}\right)uu^* - \left(\frac{PC}{1+PC}\right)wu^* - \left(\frac{C}{1+PC}\right)vu^*, \tag{3.173}$$

$$uu^* = \frac{C}{1+PC}r\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right], \tag{3.174}$$

$$- \frac{PC}{1+PC}w\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right], \tag{3.175}$$

$$- \frac{C}{1+PC}v\left[\left(\frac{C}{1+PC}\right)^* r^* - \left(\frac{PC}{1+PC}\right)^* w^* - \left(\frac{C}{1+PC}\right)^* v^*\right]. \tag{3.176}$$

Again making use of Equation 3.169, we have:

$$\begin{aligned} uu^* &= \frac{Cr\left[Cr^* - PCw^* - Cv^*\right]}{\|1+PC\|^2} \\ &+ \frac{PCw\left[Cr^* - PCw^* - Cv^*\right]}{\|1+PC\|^2} \\ &+ \frac{Cv\left[Cr^* - PCw^* - Cv^*\right]}{\|1+PC\|^2}. \end{aligned} \tag{3.177}$$

When we take the expectation, the noises being uncorrelated from the reference means that:

$$E\left\{uu^*\right\} = G_{uu} \tag{3.178}$$

$$G_{uu} = \frac{\|C\|^2 G_{rr} - C\|P\|^2 G_{ww} - \|C\|^2 G_{vv}}{\|1+PC\|^2}. \tag{3.179}$$

Finally, after all that, we can look at $G_{zu}/G_{uu}$:

$$\frac{G_{zu}}{G_{uu}} = \frac{P\|C\|^2 G_{rr} - C\|P\|^2 G_{ww} - C^* G_{vv}}{\|C\|^2 G_{rr} - C\|P\|^2 G_{ww} - \|C\|^2 G_{vv}}, \tag{3.180}$$

and we see the reason for all the work. Looking at Equation 3.180, we see that when $G_{rr}$ dominates $G_{ww}$ and $G_{vv}$ then we can easily extract $P$ from the three-wire measurement. On the other hand, in regions where the system components make the contributions of $G_{ww}$ and $G_{vv}$ cannot be ignored, then the estimate of $P$ will be biased.

While this would seem to settle the matter of using the two-wire measurements in place of the three-wire, there is an issue of the calculation of Equations 3.150 or 3.151. When the controller is working well, when the gain is high, then $T_{cl} \approx 1$ and $S_{cl}$ is very small. Equations 3.150 and 3.151., can be poorly conditioned in those regions. The engineering part comes in understanding the tradeoffs of each measurement and being willing to apply multiple measurements to improve our understanding.

## 3.19 Fourier Analysis

There are many many reference for general Fourier analysis. A classic starting point is Bracewell [78]. Other nice ways to populate one's bookshelf include Oppenheim & Shafer [79] and Press, Flannery, Teukolsky & Vettering [80]. Many of these are very good at explaining the math, but fall a bit short in relating this to real measurements. Bendat & Piersol [75] seems to be excellent for bridging this gap, but it is often hard to find the exact location where the appropriate incantations are revealed. Having run the gauntlet of trying to understand various spectra creation tools and how to compare results, I thought I'd write this stuff down before the knowledge got lost.

As far as making measurements of physical systems in order to extract a dynamic models, one has to actually compute some transform estimates based on input and output signals of some device-under-test (DUT). One of the great Emacs vs. VIM or Kirk vs. Picard debates among folks making FRF measurements is the choice between FFT based measurements and stepped-sine (also called sine-dwell by some academics and swept-sine in industry) measurements. In order to do this justice, and because there are no extra page charges in this tutorial, we will first go back and describe the underpinnings of Fourier Analysis, and how these two different measurements emerge. From here we will be able to see the tradeoffs of the two methods as the pertain to our goal of extracting good frequency-response function measurements (FRFs) and the physical parameters from our systems.

### 3.19.1 Fourier Transforms

The Fourier transform (FT) of a signal $x(t)$ is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt, \tag{3.181}$$

while the inverse Fourier Transform becomes

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df. \tag{3.182}$$

In some cases, the authors use $\omega$ in place of $f$. This doesn't affect the first integral (which is over the time variable, $t$)

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt, \tag{3.183}$$

but in integrating over $\omega$ in place of $f$ we need to factor out the $2\pi$, so the inverse Fourier Transform becomes

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(f) e^{-j\omega t} d\omega. \tag{3.184}$$

If only a finite data record of time length $t$ exists then the finite length Fourier Transform is

$$X(f,T) = \int_0^T x(t)e^{-j2\pi ft}dt. \tag{3.185}$$

We should note that for any practical measurement, only a finite data record of time length $t$ will ever exist, so to apply Fourier transforms in real life, we need to make use of Equation 3.185.

If that signal is sampled with a sampling period of $\Delta t$ then the sequence that results is

$$x_n = x(n\Delta t) \qquad\qquad n = 0, 1, 2, \ldots N-1 \tag{3.186}$$

and Equation 3.185 can be recast as the discrete Fourier Transform:

$$X(f,T) = \Delta t \sum_{n=0}^{N-1} x(n)e^{-j2\pi fn\Delta t}. \tag{3.187}$$

## 3.19.2 Fourier Series

A Fourier series is a method of representing a periodic signal by a sum of cosines and sines. Note that a Fourier Series is a related but different animal than a Fourier Transform since the latter generally is a broadband calculation, while the former really extracts the response for a single frequency at a time.

One way of representing the Fourier Series sum is described in Witte [81] (as well as many other sources) as:

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos 2\pi nf_0t + b_n \sin 2\pi nf_0t) \tag{3.188}$$

where

$$a_n = \frac{2}{T}\int_{\frac{-T}{2}}^{\frac{T}{2}} x(t)\cos(2\pi nf_0t)dt \tag{3.189}$$

$$b_n = \frac{2}{T}\int_{\frac{-T}{2}}^{\frac{T}{2}} x(t)\sin(2\pi nf_0t)dt \tag{3.190}$$

and where

$$f_0 = \text{the fundamental frequency of the periodic}$$

$$\text{signal in Hertz,} \tag{3.191}$$
$$T = \text{the period of the signal, and} \tag{3.192}$$
$$w_0 = 2\pi f_0 = \text{the frequency in radians/second.} \tag{3.193}$$

For a variety of reasons, it is often easier to use the complex sinusoid form of the Fourier series, which uses a complex exponential in place of sines and cosines:

$$x(t) = \sum_{n=1}^{\infty} c_n e^{j2\pi n f_0 t} \tag{3.194}$$

where

$$c_n = \frac{1}{T}\int_{\frac{-T}{2}}^{\frac{T}{2}} x(t)e^{-j2\pi n f_0 t}dt \tag{3.195}$$

and the two series are related by

$$c_n = \frac{(a_n - jb_n)}{2}. \tag{3.196}$$

Fourier series form the basis for several methods of analyzing signals. In particular, both stepped-sine (swept sine) methods and describing function methods make use of identifying only the first coefficient of the Fourier series, $c_1$. A good introduction to describing function methods can be found in Ogata [13]. Swept sine measurements will be discussed in Section 3.21.

## 3.20 Fast Fourier Transform (FFT) Based Analysis

Fast Fourier transforms (FFTs) are very fast to compute and thus are used by lots of instruments and by Matlab exclusively. While they lack certain niceties of Spectrum Analyzer methods, the speed of computation and the number of places that they show up make them very useful.

### 3.20.1 FFTs

By letting $f_k = \frac{k}{T} = \frac{k}{N\Delta t}$ in Equation 3.187 we get

$$X(k) = \frac{X(f_k)}{\Delta t} = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi nk}{N}}. \tag{3.197}$$

Let $W_N = e^{\frac{-j2\pi}{N}}$ and $\tilde{W}_N(u) = e^{\frac{-j2\pi u}{N}}$. Then Equation 3.197 can be written as

$$X_k = X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} = \sum_{n=0}^{N-1} x(n) \tilde{W}_N(kn). \tag{3.198}$$

This is what a standard FFT, including the one in MATLAB computes. Note that

$$X(f_k) = \Delta t X(k) \tag{3.199}$$

which returns the FFT to something closer to the physical units.

From this definition of the FFT, the inverse FFT is given by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi kn}{N}} = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}. \tag{3.200}$$

Note that the placement of the $\frac{1}{N}$ is arbitrary. However, it is significant in trying to return the FFT calculation to physical units. Alternate FFT definitions are available as:

$$\tilde{X}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n W_N^{kn} \Longleftrightarrow x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \tilde{X}(k) W_N^{-kn} \tag{3.201}$$

or

$$\hat{X}_k = \sum_{n=0}^{N-1} x_n W_N^{kn} \Longleftrightarrow x_n = \frac{1}{N} \sum_{k=0}^{N-1} \hat{X}(k) W_N^{-kn}. \tag{3.202}$$

This is generally a pain because we would like physical units when measuring stuff in the lab and the physical units do not have arbitrary scaling. One of the main reasons for writing this document is to alleviate some of the confusion that these arbitrary definitions which differ from tool to tool have caused me and the people that I have been interacting with.

### 3.20.2 Power Spectral and Cross Spectral Densities

Earlier, we discussed cross spectra, $S_{ab}(f)$, and auto spectra, $S_{aa}(f)$, and how they were used to compute frequency response functions that were more immune to noise. This section deals with the practical generation of these auto and cross spectra from spectra generated by Fourier Transforms (or FFTs) or Fourier Series calculations. When the cross or auto spectra is normalized by the bandwidth of the measurement, we get the spectral density. Because we are making discrete time measurements with a sample period, $\Delta t = T_S = 1/f_S$, our measurement bandwidth is from $-f_S/2$ to $f_S/2$. Thus, we end up normalizing by the sample frequency, $f_S$. A common term for the auto-spectral density is power spectral density (PSD). Cross spectral densities may be referred to as CSDs. Note that if we assume the same sample frequency for all of our FRF measurements, then we can use PSDs and CSDs in place of the spectra.

Now let's look at PSDs. To compute a PSD of a measurement we want

$$PSD(x) = \frac{X^*(f)X(f)}{B_e} \tag{3.203}$$

where $B_e$ is the Resolution Bandwidth of the filter used to compute the spectrum (or the Noise Equivalent Bandwidth which is technically not the same but very close to the Resolution Bandwidth) and where $X(f)$ is the Fourier Transform from Equation 3.181. This is the smallest change in frequencies that a given measurement can resolve.

In general $B_e$ is inversely proportional to the length of the time window over which a measurement is made , *i.e.*,

$$B_e = \frac{1}{T} \tag{3.204}$$

where $t$ is the length of the time record. For an FFT,

$$B_e = \frac{1}{T} = \frac{1}{N\Delta t} \tag{3.205}$$

where $N$ is the number of points in the FFT and $\Delta t$ is the sample period between points.

Note that for a FFT, the resolution bandwidth is fixed as all the integrations are done over a single period of time ($N\Delta t$, as in Equation 3.205. Band Selectable Fourier Analysis or Zoom-FFT [82, 83] can be used to maximize the resolution, but this is usually only known to experts. However, a spectrum analyzer computes a separate integral for each frequency. To eliminate errors due to a partial period integral, the integration should be done over an integer number of periods of the frequency in question. This means that except in special cases, the actual resolution bandwidth of the calculations at different frequencies will differ slightly.

MATLAB computes the FFT in Equation 3.198 and (with some details to be filled in later) then produces

$$P_{xx} = PSD(x) = \frac{X^*. * X}{N} \tag{3.206}$$

where $X^*$ is the complex conjugate of $x$ and $N$ is the number of points in the FFT and the $.*$ operation is the element by element multiply of two same-sized vectors in MATLAB . (Windowing and scaling are standard methods of improving the performance of FFTs by driving the time signal to $0$ at the beginning and end of the data run, but we will not discuss those here.)

**Note that these units are not physical.** From Bendat & Piersol[75], page 407 there is a procedure for computing a PSD from FFT based measurements. At any frequency, $f_k$, the PSD of a signal $x$ is given by:

$$\tilde{P}_{xx}(f_k) = \frac{X^*(f_k)X(f_k)}{N\Delta t} \tag{3.207}$$

where $X(f_k) = \Delta t X_k$. This means that

$$\tilde{P}_{xx}(f_k) = \frac{(\Delta t)^2 X_k^* X_k}{N\Delta t} \tag{3.208}$$

or

$$\tilde{P}_{xx}(f_k) = \frac{\Delta t X_k^* X_k}{N} \tag{3.209}$$

so

$$\tilde{P}_{xx} = \frac{\Delta t X^* . * X}{N} \tag{3.210}$$

and thus

$$\tilde{P}_{xx} = \Delta t P_{xx} \tag{3.211}$$

i.e. to go from MATLAB units to physical units, multiply the MATLAB PSD by $\Delta t$.

## 3.21 The Stepped-Sine Integral

The stepped-sine integral implements a special case of a lock-in amplifier, diagrammed in Figure 3.19. A lock-in amplifier mixes a signal with a sine and cosine at the desired frequency at which one wants to extract the response. The mixed components are then low pass filtered to produce an in-phase ($I_{LP}(t)$) and quadrature ($Q_{LP}(t)$). If the mixing and filtering are done properly, the resulting signals will be relatively flat representations of in-phase and quadrature portions of the signal at that frequency. The issue is that this low pass filtering takes a fair amount of time. It is not uncommon to specify filters with the time constants on the order of several hundreds of periods of the frequency to be measured.

The solution is to be far more careful about the integration, as described in [53]. In the case of stepped-sine (swept-sine), the input signal is generated by the instrument and therefore precisely

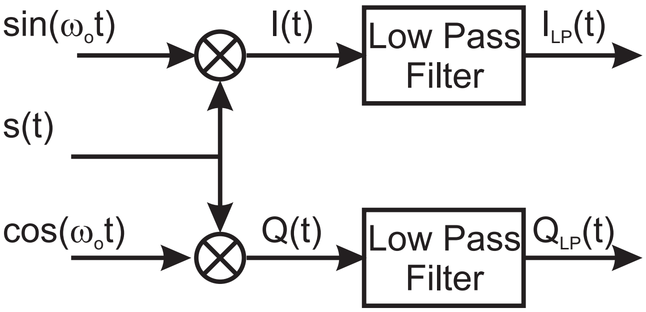


Figure 3.19: A lock-in amplifier mixes an in-phase and quadrature signal with the measurement signal and then uses a low pass filter to extract the response at the fundamental frequency.

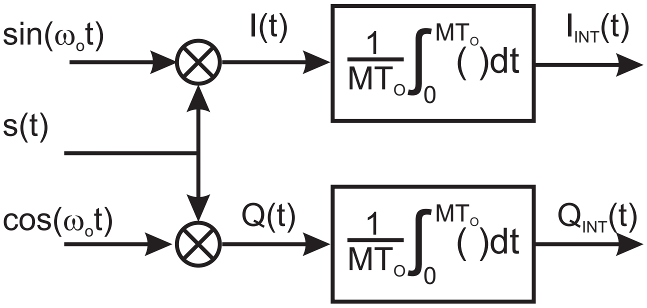

sin(ωot)
I(t)
$\frac{1}{MT_o}\int_0^{MT_o}(\ )dt$
IINT(t)
s(t)
cos(ωot)
Q(t)
$\frac{1}{MT_o}\int_0^{MT_o}(\ )dt$
QINT(t)

Figure 3.20: The stepped-sine demodulation replaces the low pass filter of the lock-in amplifier [84] with an integration over an integer number of periods of the sinusoid.

known. The response of the system should have a strong component of this stimulus signal. Moreover, since the input is precisely known, one can integrate over an integer number of periods of the input wave, as diagrammed in Figure 3.20. The mixed curve is approximated by a polynomial fit. In [53] a fifth order polynomial is fit through six points, three on either side of the region of integration. In part, this overcomes the limited sampling rate of the particular DSA, which is limited to 250 kHZ, leaving an effective measurement range of only up to 100 kHz. The author has presented several simplifications of the algorithm in [53] in the AC mode demodulator for Atomic Force Microscopes in [85, 86] and in the built-in stepped-sine measurements of [87]. In [85, 86] the integration was accomplished via a trapezoidal rule. However, the methodology of [87] allowed for a longer integration time and so that algorithm could use a backwards rectangular rule with little loss of fidelity. We will focus on the integration algorithm of the latter.

The solution in [87] involves breaking the computation up into two blocks, those that need to be done in real time and those that can be treated as pre and post processing. The real-time computations involve stimulating the loop at various points, extracting responses at other points, and computing the stepped-sine integral, will be described in Section 3.22. The pre and post processing, in which the measurement parameters are set and integrated response is tabulated and turned into magnitude and phase will be described in Section 3.23.

The stepped-sine method would have one of the loop stimulus signals set to a sinusoid at a desired frequency, $\omega_0 = 2\pi f_0 = \frac{2\pi}{T_0}$. The other stimulus inputs would be set to 0. A given output signal, $s(t)$, can be demodulated using a stepped-sine demodulator. We can use Fourier series to decompose the signal, $s(t)$, as

$$s(t) = A_0 + \sum_{k=1}^{\infty} \left( A_k \sin(k\omega_0 t) + B_k \cos(k\omega_0 t) \right). \tag{3.212}$$

We can expect that if the stimulus signal is single sinusoid, then $s(t)$ will have a strong first Fourier component:

$$\begin{aligned} s(t) &\approx A_1 \sin(\omega_0 t) + B_1 \cos(\omega_0 t) + n(t) & (3.213)\\ &= C_1 \sin(\omega_0 t + \phi_1) + n(t), & (3.214) \end{aligned}$$

where

$$C_1 = \sqrt{A_1^2 + B_1^2} \quad \text{and} \quad \phi_1 = arctan\frac{A_1}{B_1}. \tag{3.215}$$

Mixing with in-phase and quadrature signals as shown in Figure 3.20 yields

$$\int I(t)dt = \int s(t)\sin(\omega_0 t)dt \approx \int C_1 \sin(\omega_0 t + \phi_1)\sin(\omega_0 t)dt + \int n(t)\sin(\omega_0 t)dt \tag{3.216}$$

and

$$\int Q(t)dt = \int s(t)\cos(\omega_0 t)dt \approx \int C_1 \cos(\omega_0 t + \phi_1)\cos(\omega_0 t)dt + \int n(t)\cos(\omega_0 t)dt. \tag{3.217}$$

As mentioned above, in a stepped-sine demodulator, we will want to integrate over an integer, $M$, number of periods of the frequency that we wish to demodulate. Making the integrals definite and using well known trigonometric identities, yields:

$$\begin{aligned} &\frac{1}{MT_0}\int_0^{MT_0} I(t)dt & (3.218)\\ &= \frac{C_1}{2}\left( \cos\phi_1 \frac{1}{MT_0}\int_0^{MT_0} dt - \frac{1}{MT_0}\int_0^{MT_0} \cos(2\omega_0 t + \phi_1)dt + \frac{1}{MT_0}\int_0^{MT_0} n(t)\sin(\omega_0 t + \phi_1)dt \right) \end{aligned}$$

and

$$\begin{aligned} &\frac{1}{MT_0}\int_0^{MT_0} Q(t)dt & (3.219)\\ &= \frac{C_1}{2}\left( \sin\phi_1 \frac{1}{MT_0}\int_0^{MT_0} dt - \frac{1}{MT_0}\int_0^{MT_0} \sin(2\omega_0 t + \phi_1)dt + \frac{1}{MT_0}\int_0^{MT_0} n(t)\cos(\omega_0 t + \phi_1)dt \right). \end{aligned}$$

Equations 3.218 and 3.219 both have the properties that the second term on the right hand side goes to $0$ for all positive $M$. The third term goes to 0 for increasing $MT_0$ as long as $n(t)$ is uncorrelated with the mixing sinusoids.

Such precise control of the integration period is difficult in an analog circuit but straightforward in a digital operation. As $MT_0$ gets large the contribution of $n(t)$ goes to $0$, yielding the familiar relationships

$$I_{int} = \frac{1}{MT_0}\int_0^{MT_0} I(t)dt \approx \frac{C_1}{2}\cos(\phi_1) \tag{3.220}$$

and

$$Q_{int} = \frac{1}{MT_0}\int_0^{MT_0} Q(t)dt \approx \frac{C_1}{2}\sin(\phi_1). \tag{3.221}$$

By assembling the two integrals into one complex number we get the first Fourier component of $s(t)$ at $f_0$, $S\left(f_0\right)$. We can do this for any number of signals around the loop and any desired set of frequencies and using Equation 3.130, compute the complex FRF, $H(f)$, between those two measurement points.

There are several issues with standard methods of demodulation. The first is that imperfections in the integration approximation and noise in the signal require that $MT_0$ be large, relative to the period of the frequency at which demodulation is to take place, $T_0$, so $M$ must be large.

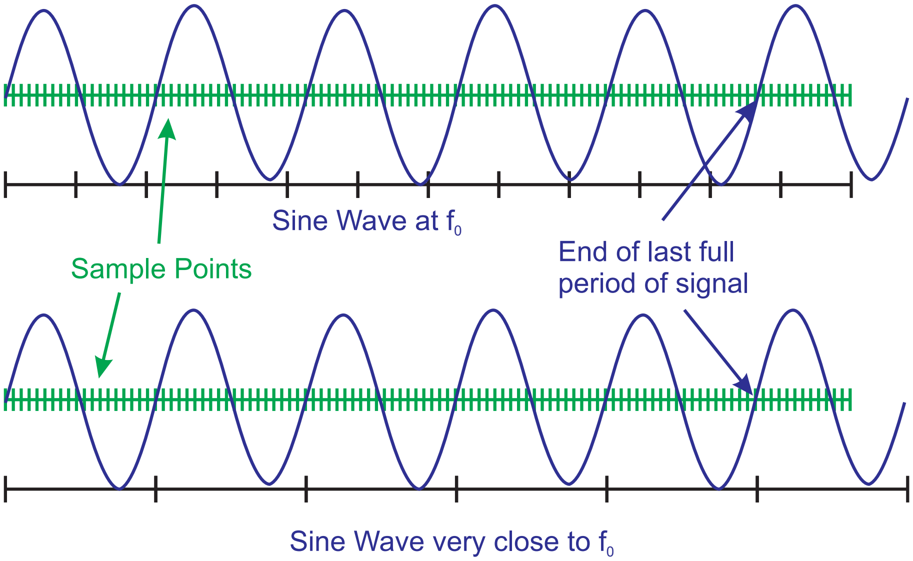


Figure 3.21: The top drawing shows a sine wave which doesn't end up on an integer number of sample points. Adjusting $f_0$ slightly allows an integer number of periods to line up with an integer number of samples, as shown in the bottom drawing.

The second is that with a digital controller, we have to be careful if we want to honor our desire to integrate over an integer number of periods of oscillation. We want

$$NT_S = MT_0, \tag{3.222}$$

where $N$ are are the number of samples in the integration, $T_S$ is the sample period, $M$ is the number of periods of oscillation, and $T_0$ is the period of oscillation. As illustrated in Figure 3.21 the data sample rate is rarely an integral multiple of the oscillation frequency, so it is difficult to make Equation 3.222 hold. Most digital systems are run at a fixed sample rate, $f_S = \frac{1}{T_S}$. The oscillation frequency, $f_0 = \frac{1}{T_0}$, comes from the frequencies at which we want to measure the FRF. That means $f_0$ will vary but $f_S$ will not. We can integrate over a fractional sample interval as described in [53] and [85]. An alternate solution is that for any desired $f_0$ and $M$, we can pick $N$ such that:

$$NT_S \leq MT_0 = N_{Real}T_S \leq (N+1)T_S. \tag{3.223}$$

We then round $N_{Real}$ to the nearest integer. We don't want to change $T_S$ or $M$, so we are left with adjusting $T_0$ so that

$$\tilde{N}T_S = M\tilde{T}_0, \tag{3.224}$$

Here $\tilde{N}$ is either $N$ or $N+1$ and $\tilde{T}_0$ is the adjusted period of oscillation which makes equality hold. The adjustments to that make equality hold can be kept small if $N$ and $M$ are made large. The computational hardware must have enough bits in the register that holds the intermediate integral approximation so as to allow for long integrals over many sample points. This is in contrast to the algorithm the author presented in [85] and [86] in which the oscillatory frequency was much closer to the sample frequency and there was a strong desire to minimize latency by keeping $M$ small.

The steps involved in generating a FRF with stepped-sines are:

- Select a set of oscillation frequencies, $\{f_0\}$, at which to compute the FRF.
- For each oscillation frequency, $f_0$, and desired number of oscillations, $M_0$, there will be a number of data samples, $N$, such that
$$NT_S \leq M_0T_0 \leq (N+1)T_S. \tag{3.225}$$
- Adjust each $T_0 = \frac{1}{f_0}$ so that equality holds for one side of Equation 3.225. For $M_0$ sufficiently large, this adjustment will be small.
- Set up a measurement period of $NT_S = M_0T_{0,adj}$.
- Set oscillator in the system controller to generate sinusoidal stimulus at frequency, $f_{0,adj} = \frac{1}{T_{0,adj}}$.

- At each sample time step, inject the sinusoidal stimulus signal into the chosen input in the control loop and measure the response at the chosen measurement points of the loop, and compute the next step in the Fourier integral.
- Store the partial integrals in memory.
- After $N$ samples, finalize the Fourier integrals by dividing by $MT_{0,adj} = NT_S$ to get $\frac{C_1}{2}\cos(\phi_1)$ and $\frac{C_1}{2}\sin(\phi_1)$ at each frequency, in real time.
- In post processing, use the integral values to compute the desired auto and cross spectra. If averaging is on, repeat the measurement and integral $N_{avg}$ times.
- In post processing, compute the desired FRFs between sets of these signals using the averaged cross and auto spectra.

The choice of measurement frequencies, the adjustments, and the setup of the integral can be done in a host computer. Generation of the stimulus, measurement of the signal responses, and calculation of the integral need to be done on the real-time system. (One could argue that one could simply store the measured data, pass it to the host computer, and do the calculations there. However, for even modest sample rates, the data transfer becomes huge and impractical. 10 periods of a 100 Hz signal sampled at 1 MHz results in $10^5$ points per channel to store transfer. That is a lot of memory for a real-time system.) Finally, Fourier coefficients are passed back to the host computer for post processing and another stimulus/integration run is set up.

## 3.22 Stepped-Sine Stimulus and Integration for FPGAs

Field-programmable gate arrays (FPGAs) offer parallel processing for simple operations, so if we can break our algorithm into simple operations, we can do much faster computing with FPGAs, allowing for control at significantly higher sample rates than are possible with conventional DSP chips. In our stepped-sine calculation, the key to approximating the integral in real time is to turn it into a convenient sum form, that is:

$$\frac{1}{MT_0}\int_0^{MT_0} I(t)dt = \frac{1}{NT_S}\int_0^{NT_s} I(t)dt \approx \frac{1}{NT_S}\sum_{k=0}^{N-1} I(kT_S)T_S = \frac{1}{N}\sum_{k=0}^{N-1} I(kT_S). \tag{3.226}$$

Likewise,

$$\frac{1}{MT_0}\int_0^{MT_0} Q(t)dt = \frac{1}{NT_S}\int_0^{NT_s} Q(t)dt \approx \frac{1}{NT_S}\sum_{k=0}^{N-1} Q(kT_S)T_S = \frac{1}{N}\sum_{k=0}^{N-1} Q(kT_S). \tag{3.227}$$

We are using a rectangular rule approximation, unlike the fifth order polynomial of [53] or the trapezoidal rule of [85], since our relatively high sample rate and large number of samples minimizes the integration error caused by the simple approximation. We still need to normalize the sums by $\frac{1}{N}$ and for a long integration with a fast sample frequency, $N$ can be huge. For example on a Xilinx FPGA using a DSP48E block [88], has multiplies of 25-bit by 18-bit numbers, where the numbers are in twos compliment form. If we have 25 bit data values multiplied times 18 bit sine/cosine values, the resulting product has 43 bits, of which the top 2 are redundant. We want to sum a lot of these values, so we need a fairly large accumulator. (The DSP48E has a 48 bit P (product) register.)

So, if we have up to 20 million sums, that's enough for 100 cycles of 10 Hz signals sampled at 2 MHz. For 20e6 sums, we need log2(20e6) = 24.25 bits or just under 25 bits. We have bits 43–48 free (6 bits) and then we need 25 bits of accumulation total, we will need a 67-bit register. Choosing a 68-bit accumulation register, we still need to multiply by 1/N, but doing a multiply on a 68-bit quantity would involve a slow process with multiple DSP48Es. However, there is a different way.

To do a 1/N average, we do the computation $N = L(2^K)$. The $1/2^K$ will be done by the right shift by K bits and the last part of the normalization will be done by a multiply by $1/L$. Now, $1 < L < 2$ so $0.5 < 1/L < 1$, and we can represent $1/L$ as s1.17 two's complement number where the number looks like 0.1XXXXXXXXXXXXXXXX, where the "X" bits can be either 0 or 1. Our host computer can calculate $K$ and $1/L$ for each frequency and download it to the real-time system, which can normalize a large sum by first right shifting by $K$ bits and then by multiplying the resulting value by our 18 bit representation of $1/L$. Our stepped-sine integral has been turned into a single multiply and accumulate at each time step, followed by the fairly easy $1/N$ normalization just described.

## 3.23 Software Pre and Post Processing

The choice and tweaking of measurement frequencies to tile into integer numbers of sample periods is all done in a host computer, as is the calculation of $1/L$ and $K$ for the normalization. Once the integrals are collected on the real-time system, cross and auto spectra are computed and the entire process is repeated for the requisite number of averages. Thus, we estimate Equation 3.130 with

$$E\{Z(f)U^{*}(f)\} \approx \frac{1}{N_{avg}} \sum_{1}^{N_{avg}} Z_i(f)U_i^{*}(f), \tag{3.228}$$

for all signals of interest. Typically, we need $N_{avg} \geq 3$ for the coherence to have any relevance, but the stepped-sine is clean enough that $N_{avg} \leq 10$ usually works. This is in sharp contrast to FFT methods, where $N_{avg}$ can often be in the hundreds to try to restore SNR. While this is being done, a new frequency is measured. Once all frequencies have been measured in this way, the FRFs can be extracted from the averaged cross and auto spectra.

## 3.24 FFT versus Stepped-Sine Tradeoffs

With all that previous discussion, we can now get back to the tradeoffs in FFT versus stepped-sine measurements.

Things to note about FFT measurements:

- The number of terms in an FFT based measurement is always a power of $2$. If the number of measurement points is not a power of $2$, then the data is either truncated or padded to make it fit.

- The FFT calculation comes from a time when computational power was limited, memory was expensive, and the high level programming language was FORTRAN [89]. As such, a lot of tricks are played to minimize the number of calculations, including every possible symmetry property of the frequency factors.

- The resolution bandwidth, $B_e$ is set by the integration time (as it is with stepped-sine). However, in this case
$$B_e = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{1}{NT_S} = \frac{f_S}{N}. \tag{3.229}$$
That is to say that to maintain the same frequency resolution, as the sample rate goes up, the number of samples must also go up. This is not surprising, except that the ratio is at certain quantized levels. We might increase $f_S$ by $10\%$ but need to double $N$ to be able to keep $B_e$ within a certain range.

- $f = 0$ i.e., DC, is always included in the calculation. That is to say if our frequency band of interest is between $f_a$ and $f_b$, we cannot simply integrate between those. If $f_a$ and $f_b$ are close together but still relatively high in frequency, this means that most of the frequency bins in our calculation are wasted. A seldom remembered technique known as Zoom-FFT can be used [82, 83], but often folks simply accept poor resolution bandwidth or huge $N$ to meet their needs.

- On the other hand, we can compute FFTs on any signal including square waves and their responses. This makes FFT methods more practical for FRF measurements on operational data. FFTs compute the entire spectrum from $-f_S/2$ to $f_S/2$. There might not be much signal content in some of those frequency bins, which is a real issue in getting accurate parameters. There is a lot of literature aimed at generating good input signals for exciting systems so that FFT based FRF measurements will be accurate [90, 91, 92].

Stepped-sine measurements are based on a computation of the first term in a Fourier Series expansion for a desired set of frequencies. Because of this:

- We can select the desired frequencies for evaluation. If we want to add a frequency, it is just appended to the list.
- We can narrow the resolution bandwidth, $B_e$ by simply extending the integration time, and we extend our integration time by adding an integer number of periods of the frequency of interest.
- We can filter and integrate to eliminate almost all the content outside of the frequency of measurement.
- However, we cannot generate stepped-sine measurements simply by using operational signals. The measurement requires controlled injection of a set of sinusoids and carefully controlled integration of the responses.

DSAs are capable of FFT based measurements and of stepped-sine (usually called swept-sine in industry) measurements. Figure 3.22 shows an example of why one might opt for the more complex swept-sine measurement over the theoretically more satisfying FFT based measurements.

Mechatronic systems are often characterized by system flexibility, which manifests itself as a large number of high Q resonances and anti-resonances in the physical system FRF. Such responses are typically hard to measure using time domain methods because the amount of signal concentrated near any particular feature is small. In other words, identifying large numbers of high Q features during normal operation is difficult if not impossible, unless normal operation continuously stimulates all of those features.

Likewise Fast Fourier Transform (FFT) based methods do not focus signal in any one feature area, instead relying on broadband excitation (pseudo-random, noise-like signals) or on a chirped sine

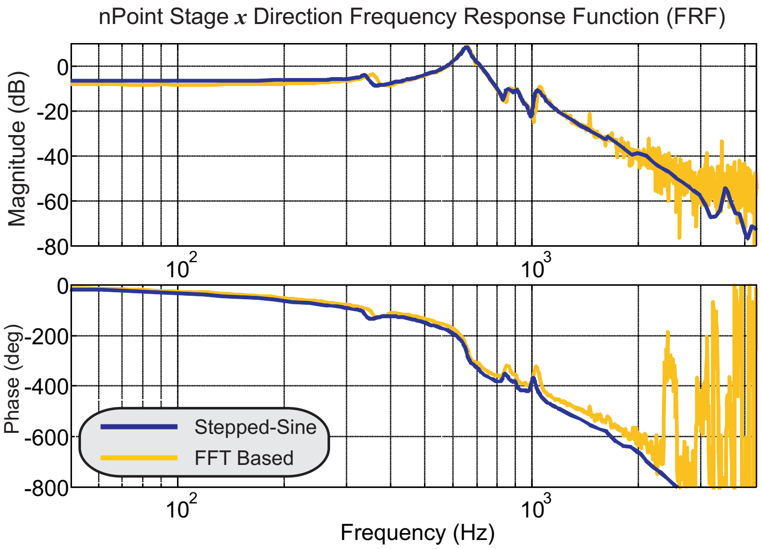


Figure 3.22: Comparison of stepped-sine and FFT based FRF measurements. (Courtesy: Jeff Butterworth).

signal. FFTs are computationally fast, but this speed comes at a price. Besides the lack of frequency isolation on the input, the frequency bins are fixed for a given sample rate and number of samples.

In contrast stepped-sine or sine dwell [53] uses a single sinusoid injected into the system. The input is continued until the system goes to steady state and then the output is measured. The LTI system response that sinusoid will be another sinusoid at a different magnitude and phase, but at the same frequency. By repeating measurements, as series of frequency points can be identified and these will delineate the frequency response function of the system. Because the stimulus is a single sinusoid, the algorithm uses coherent demodulation (mixing with a sine and a cosine wave of the same frequency and integrating over an integer number of periods of that wave) to extract the system response. The focus on one frequency at a time has a clear SNR advantage over broadband methods. While both methods will include components from the normal operation of the loop in the measured signal, FFT methods cannot isolate on a single stimulus frequency. The loop signals may show up in the stepped-sine measurements, but these will largely be not coherent with the stimulus and therefore suppressed by the coherent demodulation. Furthermore, the measurement degrades gracefully in the presence of nonlinearities [62, 63]. For this reason, stepped-sine responses typically produce much “cleaner” measurements, especially at higher frequencies where the mechatronic system response is low as clearly seen in the example of Figure 3.22. This has also been analyzed in the context of settling time and signal to noise ratios in earlier work [93].

The cost of doing this is a far more complicated software algorithm combined with significantly longer measurement times, as the measurements are repeated with each frequency step [93]. For this reason, stepped-sine measurements have largely been restricted to external and expensive instruments. As will be discussed in Section 3.26, this is a severe limitation in modern digital control systems. The contribution of this paper is to reformulate the algorithm so it is simple enough to implement in an FPGA. In doing so, it enables high precision FRF measurements to be added to any digital controller.

## 3.25 The Case for Connected Measurements

> "But the point of a measurement is lost, if you keep it a secret. Why didn't you record it so you could tell the world, eh?" – Dr. Strangelove, evangelizing about connected measurements

Consider the following scenario: An engineer is measuring the response to a laboratory system with one or more high quality tools, such as a digital oscilloscope or a dynamics analyzer. The engineer reads the response from the small, outdated screen of the instrument, pushes a few buttons and extracts some system properties. Perhaps they even capture a bitmap of the screen, but not the data. The data captured by this highly engineered instrument cannot be shared, cannot be used in CAD tools (such as MATLAB , Maple, or Labview), and if retained at all has some obscure filename such as m2016_57.csv. If John Oliver were an engineer, one of his "How is this still a thing?" segments would focus on this.

It is now close to 50 years since Hewlett-Packard introduced the Hewlett-Packard Instrument Bus (HP-IB) [94, 95] specifically to resolve what was recognized as an issue even in those early days. Starting in the 1970s, National Instruments built an entire business and platform around the idea of connected computer based instruments [95]. If one remained in the Labview [96] environment, one could connect one's measurements. However, this left many high quality boxed instruments out of the picture (unless one wished to connect them into Labview using a cryptic language known as Standard Commands for Programmable Instruments (SCPI) [97]. It seems that the programming effort needed to automate instruments using SCPI has left unconnected measurements as "still a thing". Furthermore, as Mike Borrello points out in his tutorial [98], instruments such as dynamic signal analyzers (DSAs) have not been evolved in recent years and so are stuck with ancient floppy disks and GP-IB connectors, but no Ethernet or USB connections. This section makes the case for overcoming these programming issues and connecting instruments anyway, even if the programming language is just this side of Cuneiform [99].

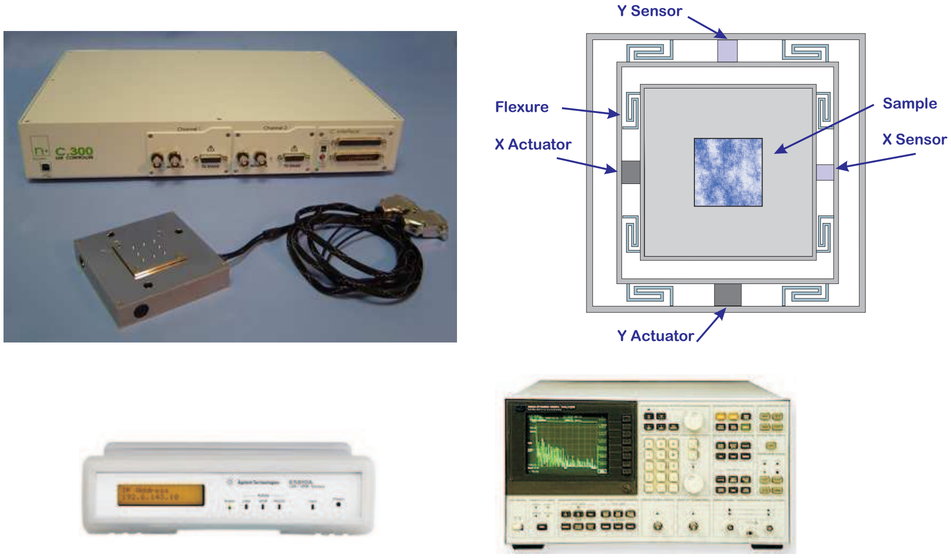


Figure 3.23: The pieces of an AFM X-Y measurement. At the top left is a picture of an nPoint nXY-100 stage and n-c300 controller. A conceptual and inaccurate drawing of the X-Y stage architecture is on the upper right. For control design, we wish to measure frequency response functions using our trusty HP-3562A [69, 70] dynamic signal analyzer (lower right). However, since modern computers do not have GP-IB ports, we need to find an alternate connection, such as the Agilent (now KeySight) E5810A GP-IB-LAN Bridge. All that is left is writing the software glue to tie these together.

nPoint makes several precision X-Y stages, such as the one shown in the upper left of Figure 3.23. On the right is a conceptual drawing of the X-Y stage from the author's imagination, rather than using information from nPoint. The n-c300 controller comes with a built-in PID loop designed for robustness and low speed accuracy. The controller some digital filters that can be adjusted if a researcher asks nicely. The controller provides analog IO ports, that is ports where an analog voltage can be injected in and others where analog voltages can be extracted out. However if one is channeling Jeff Goldblum's "Must go faster," (of both Jurassic Park and Independence Day) then one has to be able to transfer accurate frequency response measurements into design software, use these to extract model parameters, use those parameters to design a new loop shaping controller, and download those into the n-c300.

A connected measurement setup to accomplish just that is shown in Figure 3.24. The HP 3562A is used to stimulate the analog inputs and read the analog outputs of the controller module. The controller stimulates and measures the X-Y stage. Setting up the instrument is done in MATLAB using the Instrument Control Toolbox to talk to connected devices. The Agilent (now KeySight) GP-IB-LAN bridge allows the computer to connect via the LAN to HP-IB connected old instruments. This

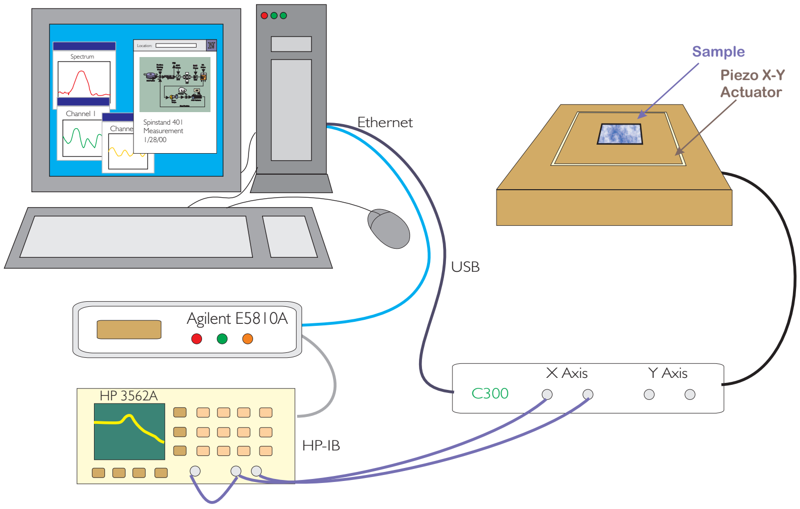


Figure 3.24: A connected measurement ties instruments and physical systems into CAD tools.

allows not only the measurement data to be captured, but also allows the instrument setup to be downloaded from a MATLAB m-file.

The only measurements that can be made via this setup is a closed-loop two-wire measurement of the type discussed in Section 3.18. In other words, for either axis ($x$ or $y$) we need to follow a type of “Lather, rinse, repeat” type process:

- Run DSA measurements on closed-loop system, controller.
- Use MATLAB to model existing controller or measure the existing controller by opening the loop on the system (disconnecting the wires from the controller to the X-Y stage).
- Open the loop using the equations from Section 3.18 to extract the FRF of $PC$.
- Divide out the FRF (measured or modeled) of $C$ to reveal $P$.
- Use $P$ to design a new controller $C$ using the structure of PID-plus-filters.
- Project new open and closed-loop responses ($PC_{new}$ and $\frac{PC_{new}}{1+PC_{new}}$) in MATLAB . This can be considered a frequency domain simulation.
- When a good projection is seen, dump new filter parameters from MATLAB to stage controller.
- Back to the top and measure again.

The key prerequisite steps, the busy work that most people avoid, are:

- Connect the DSA to MATLAB by writing all the SCPI commands into a MATLAB m-file. This involves looking through arcane old manuals, trying out programs with few debugging tools, and using the Instrument Control Toolbox and the LAN bridge to tunnel through to the GP-IB. It is tedious and frustrating.
- Write scripts in MATLAB (or one's preferred CAD tool) to mimic the loop operations of the DSA [76].
- Translate between DSA FRF format which has a header with start and stop frequencies and frequency spacing information and MATLAB format, which generates FRFs in magnitude and phase form. The latter must be converted to a complex response for manipulation, via

$$resp = mag. * e^{j\pi \frac{\varphi_{deg}}{180}}. \tag{3.230}$$

- For the former, the frequency spacing information from the DSA data header must be used to generate a frequency vector for use in MATLAB . That same frequency vector is used to generate frequency response in model so that Bode plots of measurement frequency response functions line up with MATLAB Bode plots.
- Mix and match measurement and model FRFs to get simulation of what FRFs we can measure with new system.

FRF measurements of the original closed-loop responses from both the $x$ and $y$ axes are shown in Figure 3.25. The closed-loop $x$ axis has a -3 dB point at 200 Hz, but has a resonance near 500 Hz. Similarly, the closed-loop $y$ axis has a -3 dB point at 10 Hz and a resonance peaking up around 600 Hz. Such a system cannot be pushed very fast, particularly in the $y$ axis. There is also nonlinear coupling as might be inferred from the conceptual drawing of Figure 3.23, which puts a limit on how high the inputs to the actuator can be. However, the reader should note the incredible noise rejection of the stepped-sine measurement of the HP 3562A.

The extracted plant FRFs are shown in Figure 3.26. From this, we are able to use the existing controller programming structure (PID for basic control and filters to knock down resonances) and iterate. The resulting projected open-loop responses both with old and new PID controllers is shown in Figure 3.27. We can also close the loop in FRF math and project the closed-loop as done in Figure 3.28. Note the improved bandwidth, better roll off, and lessened peaking. However, this is just a projection, based on combining extracted plant measurement with controller and filter model. Note

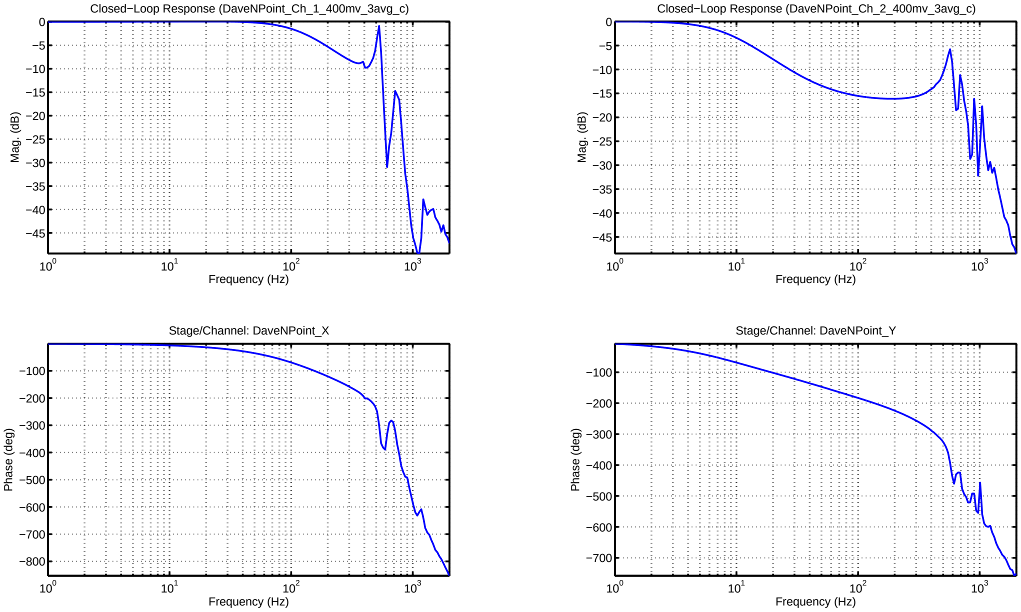


Figure 3.25: Closed-loop FRF measurements of X and Y stages with original controllers.

that the projected closed-loop with the new PIDs look slightly worse than those with the old PIDs. However, in the measured closed-loop responses, the new PID and filters work exceptionally well, as seen in Figure 3.29. The $x$ axis now has a -3 dB point at about 250 Hz, and no resonance to worry about. The $y$ axis has a -3 dB point at about 150 Hz, and no resonance to worry about. This means that scan signals beyond the bandwidth get low pass filtered in a graceful way.

Although often overlooked, projecting or simulating the frequency response can yield excellent controller designs. Tight integration between MATLAB , the dynamics analyzer, the physical system, and the controller make rapid iteration easy. All the tools have to pass data between themselves gracefully (the hardest part) and have to have similar data structures. As important, to do actual FRF manipula-

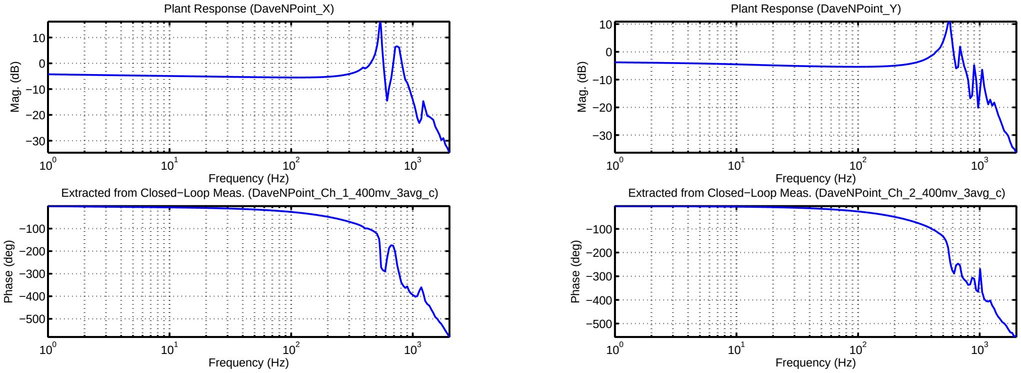


Figure 3.26: Extracted plant FRF measurements of X and Y stages.

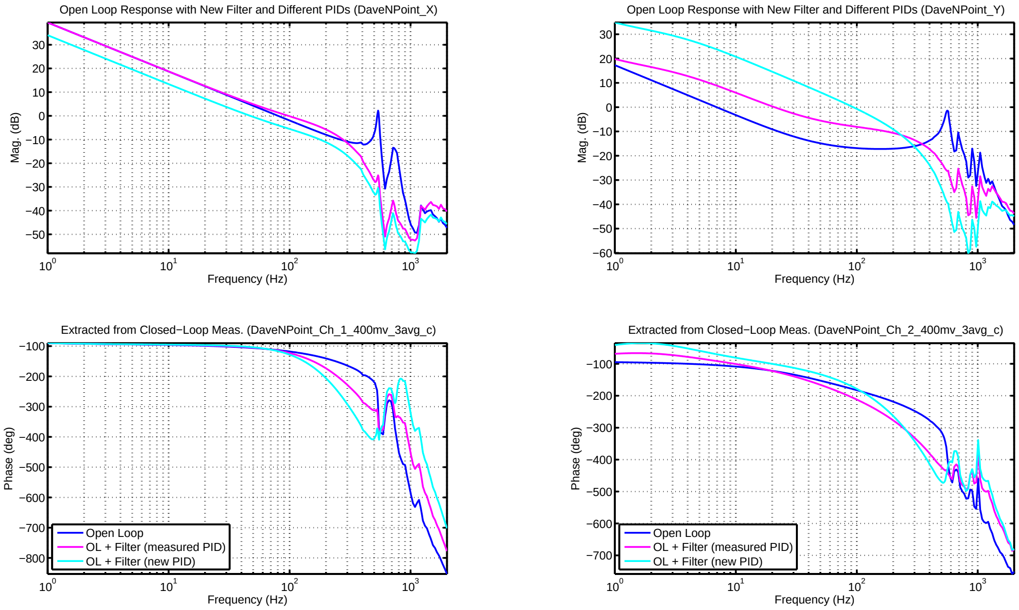


Figure 3.27: Extracted open loop FRFs and projected FRFs of new open loop using old PID and new filters and new PID and new filters for X and Y stages.

tions in MATLAB (or similar CAD tools) the frequency responses must be kept in a complex response form.

## 3.26 The Case for Built-In Stepped-Sine

This section will discuss the advantages of building a DSA right into the real-time digital controller. Figure 3.30 shows the FRF measurement in the context of an overall digital feedback and feedforward controller. In this case, the sheer complexity of the digital "patch-panel" argues against using external instruments and for having these measurements built into the digital controller [87]. We will discuss when that might be a good idea and what is involved.

Identifying the many component blocks in the system involves injecting and extracting signals at multiple locations, and the majority of access points are buried in the digital controller.

Using an instrument through analog test points involves not only creating those test points with circuitry, but also forcing many signals that were digital to be converted to analog signals before being measured with our external instrument. Perhaps more critical is the fact that much of the controller

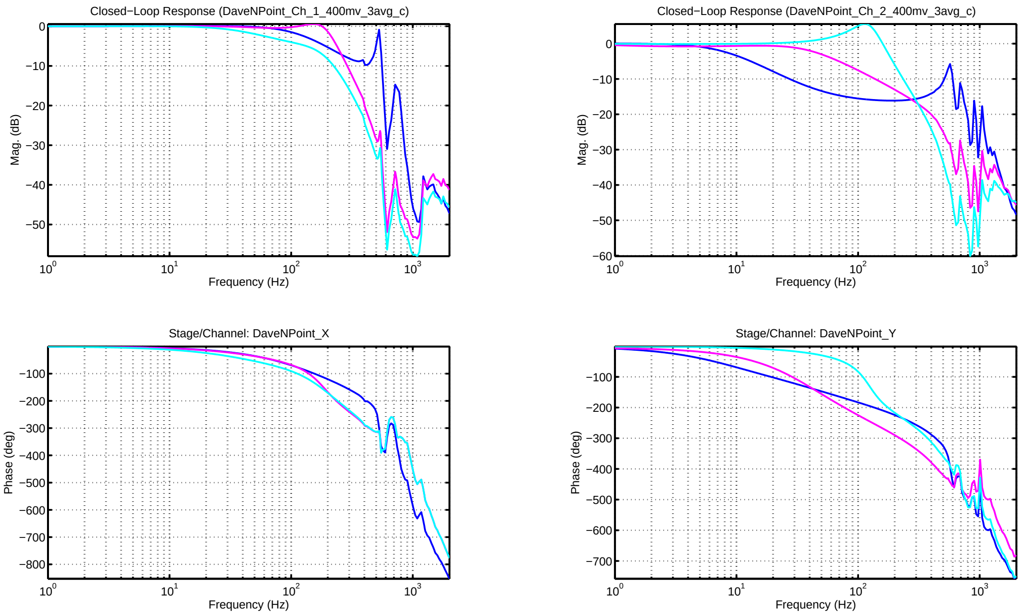


Figure 3.28: Measured closed-loop FRFs plotted against updated projected FRFs of new closed-loop using old PID and new filters and new PID and new filters for X and Y stages.

is inaccessible to the instrument. This can be remedied by modifying the instrument to have digital interfaces, as was done with the HP 3563A [70], but connecting such an instrument involved coming up with a digital bus protocol between the instrument and the control system [76, 19]. Given this level of work, one might as well build the instrument right inside the controller.

Returning to Figure 3.30, we see that not only do we want multiple digital access points, but reconfiguration of the controller would require modifying those access points. This reconfigurability is exactly what software allows us to do. In a single CPU or DSP based system, operations for computing the stepped-sine stimulus and integration would take away from the processing time available for real time control. This evaporates with the parallel processing capability of Field Programmable Gate Arrays (FPGAs). While difficult to program, FPGAs allow algorithms to multiplex in chip space instead of processing time. Very simply, this means that in an FPGA, we do not burden our controller by the addition of this measurement algorithm. This allows for measurements of systems with extremely high sample frequencies. In the examples of Section 3.27, the atomic force microscope (AFM) system in question was sampled at 2 MHz. The FPGA firmware would have allowed measurements on dynamic systems sampled at 40 MHz. In contrast, the HP 3562A and HP 3563A were external instruments with sample rates limited to 250 MHz [70].

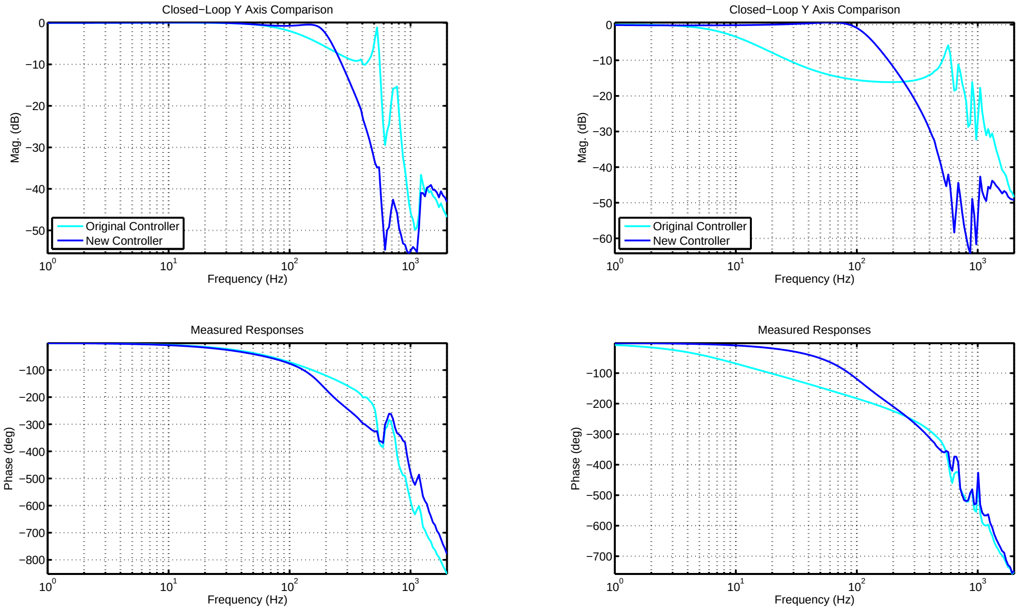


Figure 3.29: Closed-loop FRF measurements of original versus new responses.

## 3.27 Simulation and Measurement Results

The FPGA portion of the algorithm was simulated in ModelSim. Two results are shown in Figure 3.31 and 3.32. The sample rate for both of these is 2 MHz, and the simulated amplitude is 0.25 while the phase is at $-15^\circ$. The sines, at 2 kHz and 101 kHz, respectively, were corrupted by AGWN. The stepped-sine integral was run for 8 periods before faithfully returning the magnitude and phase. (These were extracted in the ModelSim testbench.)

Following the successful tests of the FPGA portion, it was integrated into a real-time digital controller while the software portion was implemented on a host computer. A closed-loop measurement of an nPoint N-XY30 stage is shown in Figure 3.33, while the extracted plant response is shown in Figure 3.34. Note that the NPXY30 has a much stiffer response than the NPXY100A of Figure 3.22. However, the main observation is the cleanliness of the FRF measurement.

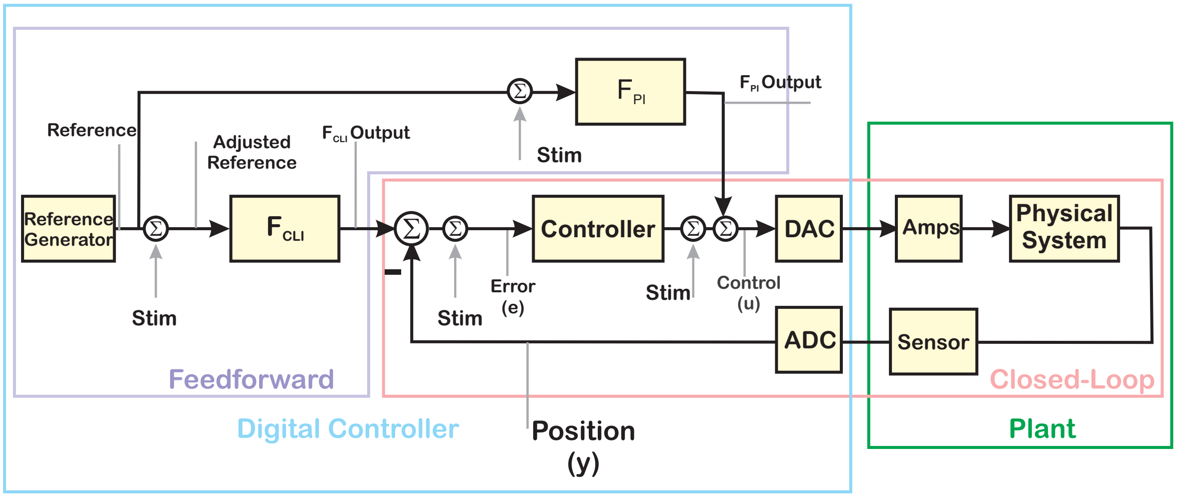


Figure 3.30: More complete measurement of system inside a digital control loop reveals the need for multiple stimulus and measurement points often not easily achieved with traditional instruments. Increased computing power means that far more of the conceptual system lies within the digital controller. The dramatic expansion in availability of sensors and low cost ADCs mean that single measurement or signal injection points in a measurement system are terribly outdated. By keeping the “digital patch panel” inside the digital controller, we can more easily assure a synchronized time base and compatible signal formats.

## 3.28 Extracting a Parametric Model

In the case of time domain system ID on a discrete time model, the identification itself is supposed to provide the transfer function (or state space canonical form) parameters. As we have tried to drive home far too much for an evening’s entertainment – but not nearly enough to make the point – the connection between these parameters and any physical parameters is often so tenuous that one might as well start looking for mystical energy fields. Nevertheless, if one uses the direct Z-domain approach of Ragazzini and Franklin [51, 15], this is sufficient for design, but maybe not for debugging and insight.

On the other hand, if we are using our step response methods. we can – under the assumption that we have picked our simple model correctly – extract some basic physical parameters out of the step response. These can be used both for controller design but also for intuition and understanding. The issue with them is that the number of parameters one can extract that way is limited, and it truly depends upon having an assumed model that describes reality pretty well. Still, sometimes that is the best you can do.

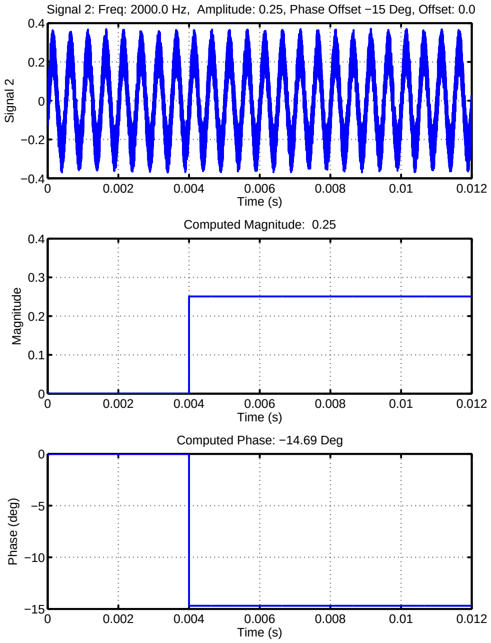


Figure 3.31: Bit accurate simulation of FPGA stepped-sine integral. $f_0 = 2000Hz$, Amplitude = $0.25$, and Phase Offset = $-15°$. Integration done for 8 periods of oscillation, after which result is extracted.

We have tried to make the case that for characterizing the system, frequency domain methods may be superior when they are available. This is not standard thinking on the topic, and I believe that the main reason for this is that when most folks in the academic/theoretical world discuss frequency domain methods, they are referring to FFT based methods with broadband excitation. In such discussions, Parseval's theorem [101] comes up as:

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \frac{1}{\pi}\int_{-\infty}^{\infty} |X(\omega)|^2 d\omega = \int_{-\infty}^{\infty} |X(2\pi f)|^2 df. \tag{3.231}$$

That is, the energy in the time domain is equal to the energy in the frequency domain and so no improvement in identification comes from moving to the frequency domain.

What this discussion leaves out is that in the frequency domain, we can get a better understanding of which parts of the system model to target with input. It makes no sense to provide input from "DC to daylight" (old analog engineer term), if the system response is 120 dB down above a frequency of 100 Hz. The frequency domain gives us a better understanding about how to target this input – assuming that we have such an opportunity – which is not always the case.

Another thing left out of this discussion is Resolution Bandwidth, described in Equations 3.203 and 3.204. Resolution Bandwidth tells us that there is no magic here. We can resolve no better than the reciprocal of our integration time (at best). Thinking about this and Parsevall's theorem, we can

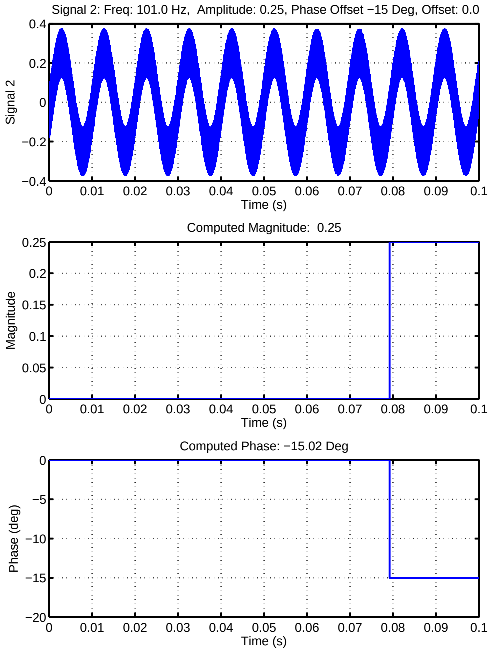


Figure 3.32: Bit accurate simulation of FPGA stepped-sine integral. $f_0 = 101Hz$, Amplitude = $0.25$, and Phase Offset = $-15°$. Integration done for 8 periods of oscillation, after which result is extracted.

realize that resolving frequencies with broadband excitation requires a lot of energy (large time integral to cover the broad frequency range). However, if we narrow the frequency to a very thin line then the amount of energy goes down both in time and frequency. Still, for a certain resolution, the Resolution Bandwidth tells us we need to integrate longer.

What does all of this mean? The resolving power of stepped-sine is in large part due to the long integration time. Of course, the coherent demodulation, the single sinusoid input, the gain adjustment all play a role, but eventually, we need to integrate for a while. Not nearly as long for the same resolution as broadband methods, but still.

When we generate a frequency response function (FRF), we still are not at a parametric description of the system. We need to do a curve fit [64, 65, 102]. For many of these instruments, the complex curve fitting method is used, but it runs into problems with the FRF is not pristine, or when there is extra negative phase, or when there are small measurement bumps in the FRF. A method that tries to reduce these effects by assuming certain filter block components and using the logarithmic magnitude response was proposed by Sidman et. al. back in 1991 and recently adapted for tuning of Atomic Force Microscopes [103, 19]. This shows promise but this method is limited to identifying stable, minimum phase responses.

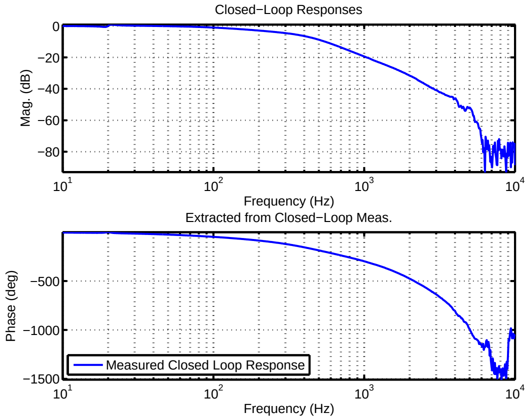


Figure 3.33: Built-in stepped-sine measurement of nPoint NPXY30 [100] x actuator in closed-loop.

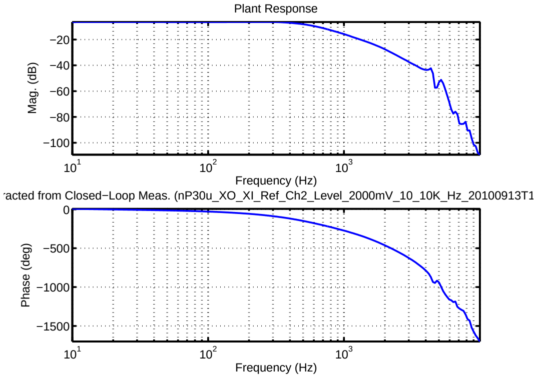


Figure 3.34: nPoint NPXY30 x axis plant response extracted from measurement of Figure 3.33. DSA.

## 3.29 Improved Curve Fitting for Mechatronic Systems

Working on optical drives [19, 31, 76], my desire was to move to MIMO models, which required model-based control, which required parametric models of the physical system. Extracting parametric models from the Frequency Response Functions (FRFs) produced by the HP 3563A Control System Analyzer (CSA) required curve fits, [104, 69, 102, 64, 65]. The CSA and DSA had curve fitting algorithms that worked well on measurements of analog circuits, but failed repeatedly on those of the drive mechanism. Instead of a second or fourth order model that physical intuition would have suggested, the models were of high order, and contained unstable poles and non-minimum phase zeros. A more mature version of myself would have worked to improve the measurements from the start, but it was sufficiently confusing for me at the time to realize that I could not use the existing tools to get parametric models from which to work. While I should have recognized that discrete-time FRFs needed to be sanity checked against continuous time FRFs, the discrete-time representation of high

Q systems had issues.

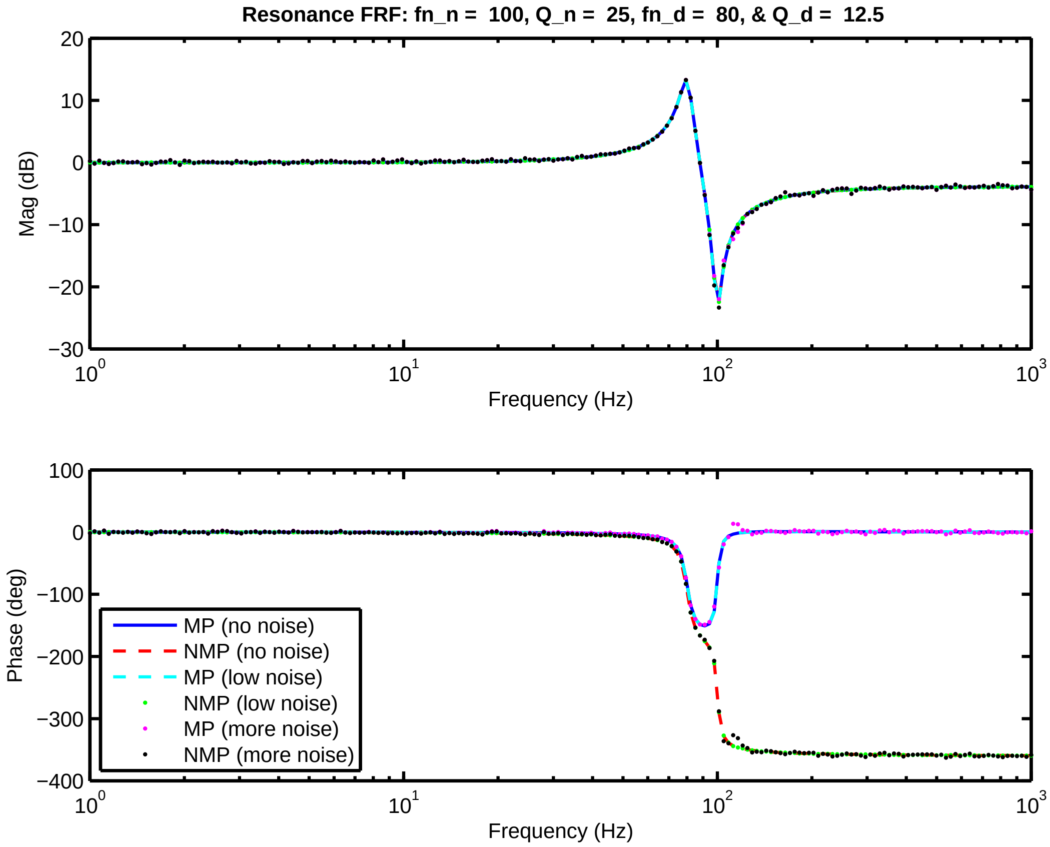


Figure 3.35: FRF of biquad filter with $f_{n,n} = 100$ Hz, $Q_n = 25$, $f_{n,d} = 80$ Hz, $Q_d = 12.5$. Additive white Gaussian noise with $\sigma = \{0, 0.002, 0.02\}$ is added to the real and imaginary responses.

Consider the example in Figure 3.35. This is a fairly simple second order section where the FRF has been corrupted by adding various levels of additive white Gaussian noise to the complex response. Two versions of the dynamics are plotted, one with a complex pair of minimum phase zeros and a second where the zeros have been flipped over the $j\omega$ axis to make them non-minimum phase. A simple version of the complex curve fit is applied to this system, in that the order of the complex fit is limited to be second order. For very small amounts of noise in the FRF measurement, the complex curve fit still works, as shown in Figure 3.36. However, increasing the noise level slightly causes the complex fit to badly miss the parameter locations, as shown in Figure 3.37. Furthermore, the fit has missed the sense of the NMP zeros as well.

One possible explanation is as follows: One of the problems with curve fitting results from small bumps in the FRF magnitude that are not accompanied by matching phase variations. Thinking about Bode's

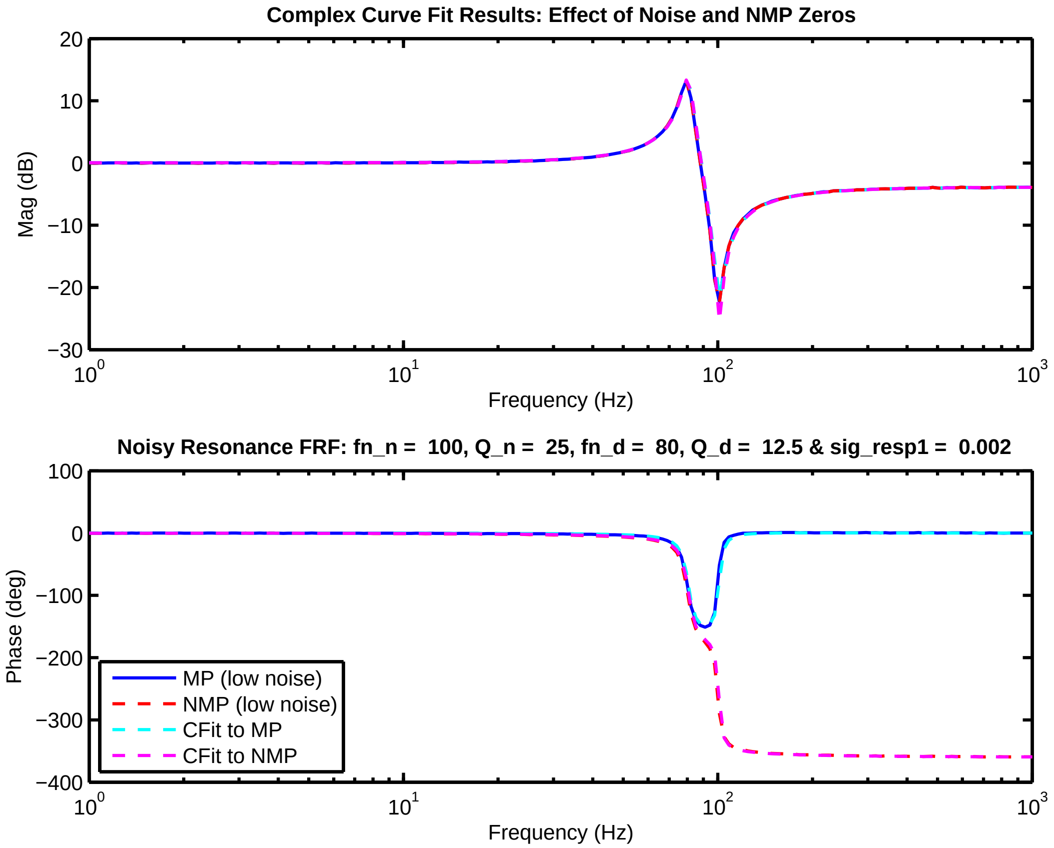


Figure 3.36: Complex curve fit applied to simple resonance/anti-resonance with FRF noise $\sigma = 0.002$. At this point, the curve fit still seems to work, and can match both minimum phase (MP) and non-minimum phase (NMP) responses. (Cyan overlays blue, magenta overlays red.)

gain-phase relationship [105], we realize that if the magnitude variation is not accompanied by phase variation, the only way for the curve fitter to explain it is by adding a pole-zero combination that result in a magnitude blip and a net 360 degree phase jump. Unfortunately, this is well suited to matching a high Q resonance with a high Q pair of NMP zeros. This kind of issue with the linear fit makes the normal curve fit method largely unusable. The examples here are very simple. For responses with many resonance, anti-resonance features, it only gets worse.

One major source of discrete-time non-minimum phase zeros is the unwitting fitting of poles and zeros to pure time delay, as discussed in [106]. One of the ways to approximate delay is with a Padé approximation, and even at low order this maps time delay to NMP zeros. Accounting for the delay directly means that the compensator is only trying to account for the part of the system it can do something about.

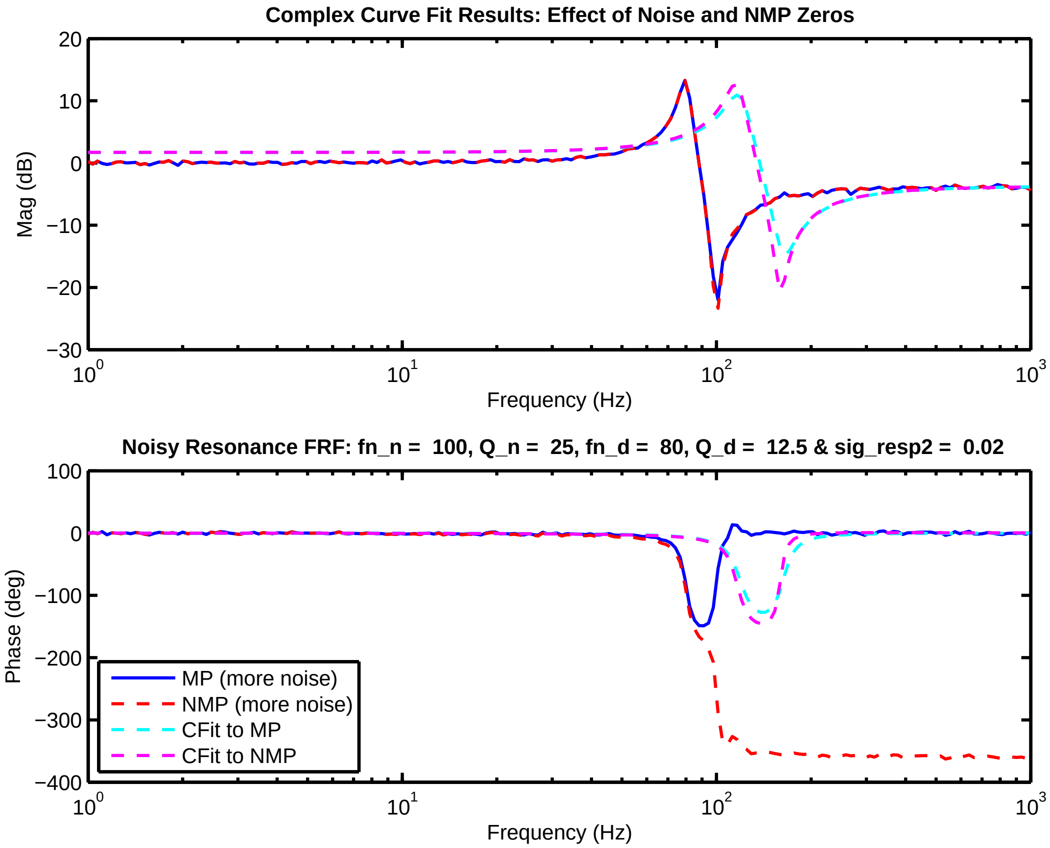


Figure 3.37: Complex curve fit applied to simple resonance/anti-resonance with FRF noise $\sigma = 0.02$. Even with this low level of noise, the complex curve fit has failed badly.

So, even if the FRF measurement is good, that is, even if it is done using sine-dwell and has high coherence as described in Section 3.26, we are a long way from a usable state-space model or a usable model for any sort of control design. At this point, the missing piece was the ability to extract reasonable models from good FRF measurements.

The situation in Figure 3.37 is telling, since even a simple biquad with fairly low levels of noise in the FRF measurement cause a pretty dramatic miss in the fit response. The inspiration for a solution came in an old paper by Sidman et. al. [66] in which they suggested two fixes: to work with the log magnitude response (which would force an assumption of a minimum phase system, but would de-emphasize noise in the amplitude) and to do the fit by cycling through a series of fixed dynamic models to see which produced the lowest residual error. I guessed that if I could fit low-order dynamics and remove them from the response – something I called successive dynamic removal – then a series of low-order fits would result in an eventual fit to the entire response. I was asked to hand off my notes that detailed this strategy [107] (as well as the Matlab scripts for automatic PID tuning [108, 109] and

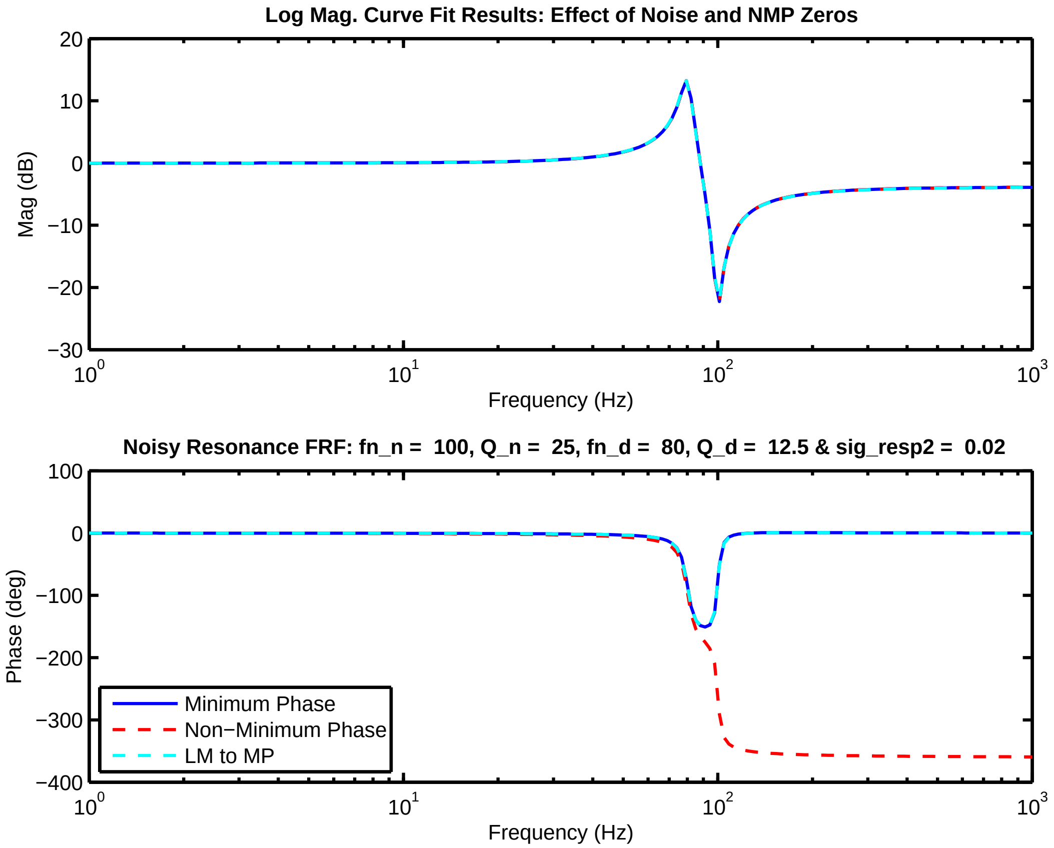


Figure 3.38: Least squares fit assuming a biquad filter structure and using only log magnitude (LM) measurement data. The fit done with the higher level of noise, $\sigma = 0.02$, still matches the no noise response extremely well.

multinotch parameterization [54, 33] to a new member of the Agilent Labs AFM team, Chris Moon. Using this approach, he found that if he used the multinotch of Section 6.11, the built-in stepped-sine of Section 3.26, Sidman et. al.'s log magnitude fit, and MATLAB 's lsqfit routine, he could use successive dynamic removal to eventually turn the open-loop response into an integrator. The filter that was fit to do this was a combination of PID controller and multinotch, with parameters automatically arrived at by the algorithm. He made the assumption that resonances and anti-resonances would be interlaced and added scripts to roughly approximate these peaks and troughs to give a starting point for fitting individual sections [110].

Applying this method to our earlier example results in the very accurate match of the minimum phase dynamics as shown in Figure 3.38, which shows none of the issues from the complex fit. However, the assumptions of the fit are that the system is minimum phase. To get around this, the fit can be adjusted by a method that checks the phase residuals in the area of an identified feature (resonance,

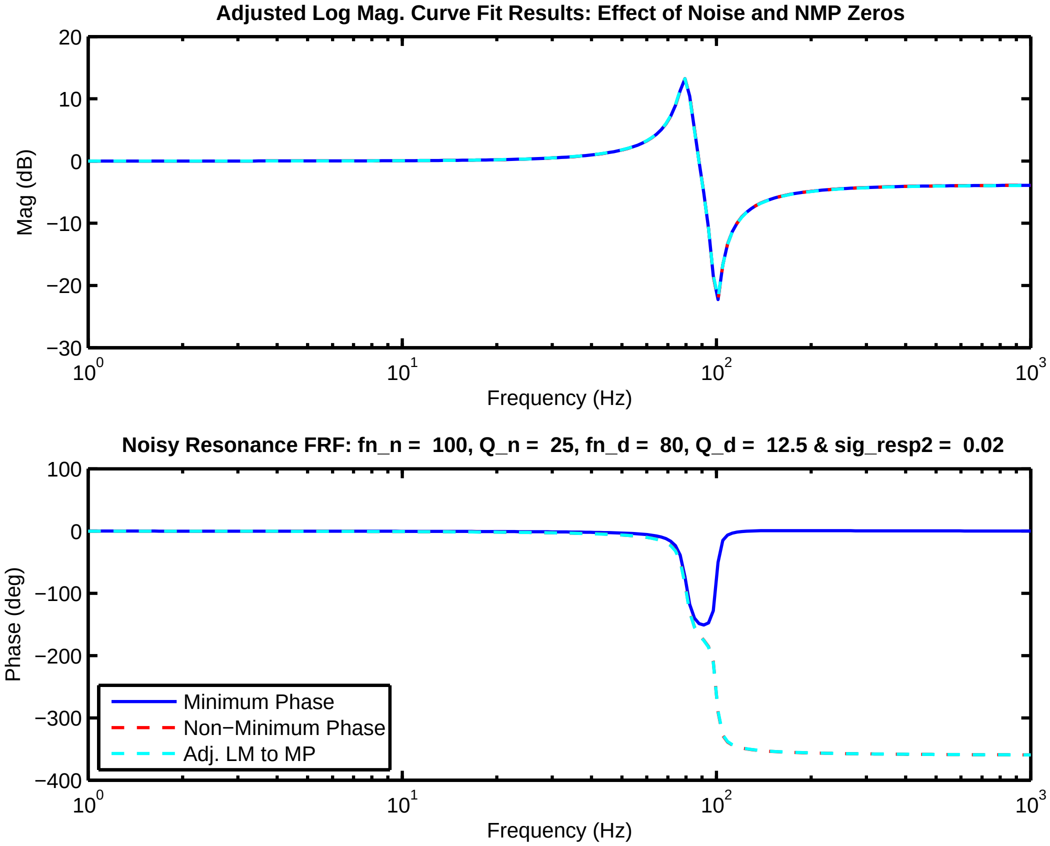


Figure 3.39: Adjusted least squares fit to the LM curve, where the fit from Figure 3.38 is subsequently checked for phase jumps near dynamic features.

anti-resonance, or pair) and then makes an adjustment and checks the phase residuals again. An example of this is shown in Figure 3.39 and this method will be discussed in [111].

## 3.30 The Effect of Delay on Curve Fits

Back in 2007, Jeff Butterworth, then a graduate student working with Lucy Pao was making measurements on an nPoint X-Y stage donated by Agilent Technologies to Professor Pao's group. (Technically, at that time it was on loan, but was donated later.) Jeff was measuring the x response and getting a 7th order discrete model, with 3 non-minimum phase (NMP) zeros. It turns out that at that time I was making similar measurements on a similar nPoint stage and getting a 4th order continuous model with some delay. The thing was that our frequency response functions looked almost identical. This set Jeff to thinking and within a couple of weeks, he came back with the elegant answer: attempting to model the delay in a discrete transfer function generated the non-minimum phase (NMP) zeros, and this was clear from any understanding of the Padé approximation. As soon as Jeff emailed me a couple of weeks later, it became obvious. It ended up in a couple of papers [112, 113].

As briefly noted in Section 2.5.2, time delay manifests itself as a pure negative phase in the frequency domain. Repeating Equation 2.31, for a delay, $T_D$, the Fourier Transform pair [26] is:

$$f(t - T_D) \supset F(s)e^{-j2\pi f T_D}, \tag{3.232}$$

which means once again that even if everything else is done perfectly, the time delay will eventually drive the open-loop phase below $-180^\circ$ at some frequency, $f = f_{lim}$.

It's easy enough to plot this in a Bode plot. The magnitude remains $1$ and the phase gets increasingly negative. That's great, but it doesn't give us a pole-zero type interpretation of things. For this, we need the Padé approximation [114], which allows us to model a function such as an exponential or a sinusoid as a transfer function.

In the case of $e^{-sT_D}$, we can use varying orders of transfer function approximations. Delay results in extra negative phase, and we can only get that in a transfer function by adding poles (but this affects the magnitude) or adding a section with magnitude $1$, but some non-minimum phase zeros that add negative phase. This is what happens with the Padé approximation. A first order Padé approximation would look like [14]

$$e^{-sT_D} \approx \frac{1 - \frac{T_D}{2}s}{1 + \frac{T_D}{2}s}. \tag{3.233}$$

Higher order approximations are the second and third order:

$$e^{-sT_D} \approx \frac{1 - \frac{T_D}{2}s + \frac{T_D^2}{12}s^2}{1 + \frac{T_D}{2}s + \frac{T_D^2}{12}s^2} \tag{3.234}$$

and

$$e^{-sT_D} \approx \frac{1 - \frac{T_D}{2}s + \frac{T_D^2}{10}s^2 - \frac{T_D^3}{120}s^3}{1 + \frac{T_D}{2}s + \frac{T_D^2}{10}s^2 + \frac{T_D^3}{120}s^3}, \tag{3.235}$$

respectively. Note that all of these have stable poles, have magnitude $1$, but have non-minimum phase zeros to produce the extra negative phase. However, thinking about a lot of control design techniques, those non-minimum phase zeros severely limit what the control design can do.

Why does this happen? Well, one of the things that made Bode plots useful to begin with was Bode's Gain/Phase Relationship, in which he pointed out that if a system was stable and minimum phase, then if one knew the magnitude of the response, one also knew the phase. Let's say we have a magnitude response but a lot more negative phase than can be accounted for by a stable, minimum phase model. The curve fit is trying to minimize the error between the model response and the measured response and the only way to do that with a rational function approximation is to throw in some non-minimum phase zeros. In other words, if you want a pole zero description of a system with extra delay, you will end up with some NMP zeros.

When we have delay, we can handle in in a combination of ways. We can ignore it if we believe it is of a small enough amount not to affect the system performance we need. We can use it as a direct limit on the performance of our system, using phase margin calculations that include delay to gate our overall system bandwidth. Alternately, we can model these as NMP zeros and try to use a design technique that attempts to compensate for this, however keeping in mind the limitations that Bode's Integral Theorem (Section 5.4) places on these. Finally, if we have some knowledge of the reference signal, extra sensors, or a portion of the disturbance that is repeatable, we can use Feedforward Control (Chapter 8) to help without offending Bode.

## 3.31 Chapter Summary

The performance of control systems relies on the designer having the ability to extract usable models from measurements of the physical system. Since it is only reasonable to think of measurements as being done at discrete time intervals, we must consider the effects of discretization as discussed in Section 3.6 and what happens to physical parameters under the effects of discretization (Section 3.7).

That being said, when they work, the discrete-time model, time domain identification methods outlined in Section 3.8 can result in a usable design model taken from time domain data. Furthermore, the methods go straight to a parametric model without having to detour through curve fits or other side measures of the step response methods.

One might very well ask why do step response methods (Section 3.9)? The most basic answer is that sometimes, that's all you've got. Sometimes, the discrete-time model, time domain identification doesn't give anything usable for nominal design. Sometimes, one has to wrap a nominal feedback controller around the plant to be able to drive the system with inputs that can be used for a more complete identification. Sometimes, the physical system only admits step inputs (such as a setpoint change). If you want to generate either confused looks or good belly laughs, suggest stepped-side methods to someone working in chemical process control.

Finally, the Frequency Domain Methods (Section 3.15) seem in many ways more complex, convoluted, and indirect than any of the others. The best of these – stepped sine in my opinion – requires special inputs and demodulation. The resulting FRFs still need to be curve fit in order to create a usable parametric model. Why would anyone in their right mind go through all this effort? The simple and inescapable answer is that when these methods work, they beat the crap out of anything else in the real world. Seriously, nothing comes close for complex mechatronic systems with high-Q resonances, so when this is what we are facing, the effort to build the infrastructure for Frequency Domain Methods is absolutely worth it.

We haven't discussed multisine methods that have been popular in the literature. The main issue that I have with them is that their seem to use multi-sines as inputs to the system, but still rely on FFT methods on the output. I am a big fan of the SNR improvement one gets from the coherent demodulation of the stepped sine, even if coding it is some work.

Some folks may model for the sake of modeling, but we control folks do it to generate good controllers. Good control engineers want to get the most out of the system that the physics will allow. You can't have any of this without models based on frequent and accurate measurements, and you won't make frequent and accurate measurements if they involve a lot of grunt work. Ergo, connect your measurement system, your physical system, your real-time system, and your CAD system together in a way that makes it trivial to pass measurements, models, and designs amongst these tools. Time spent on this aspect almost always is paid back by an order of magnitude or more return, in the speed and the quantity of measurements.

The problem with this is usually not the technology to do it; but the will of the engineer and their managers to put up with the busy work needed to make this happen. Engineers hate spending time away from their main area of contribution, fearing it will make them look like they are wasting time. Programmers hate the inelegant interfaces to instruments and the classless nature of real-time programming. Managers consider such projects out of the main line of contribution and not easily accounted for in Microsoft Project.

However, the low overhead, self consistent connection of time and frequency measurements with CAD programs such as Matlab, Octave, Maple, Mathematica, or Python, and with the real-time processing system open up whole new vistas for co-measurement and design. Consider the humble frequency response measurement. To do such a measurement, one must connect to analog or digital test points, set up the measurement, run the measurement (with appropriate levels of repetition for averaging and statistical confidence), and then transfer the data back to Matlab or its cousins. It is always amazing how many good engineers are willing to do all these steps manually and repeatedly over the course of months, rather than spending a couple of weeks to make it all happen with the push of a button (modulo the wiring, which we deal with in Section 3.26).

On the other hand, programming real-time digital controllers is hard enough without trying to add in sophisticated FRF generation code, and so much is done from time domain measurements or FFTs of time-domain measurements. Breaking down the barrier allows measurement decisions to be made based on what is best for modeling. It also allows measurements to be done repeatedly and quickly so that modeling is based on the best of many measurements rather than “that one time we actually got some lab data.” Spend the time to make the data path trivial and consistent, and everything in the control design gets better.

# Chapter 4

# Simple Controllers for Simple Models (or why so many controllers are PIDs)

## 4.1 In This Chapter

This chapter will discuss simple controllers: mostly leads, lags, and PIDs. More specifically, we will try to give some deeper insights into these ubiquitous structures and the physical system models to which they are most often applied. We will do this in a hope of being able to tune these simple controllers in a much more effectively than what is usually considered practical with these structures. We will see that:

- a generalized form of a PID controller can – in principle – control any second order LTI plant model which is not actually unstable and
- leads and lags, and lag-lead controllers can all be cast as a generalized or particular form of a PID.

It turns out that we can learn a lot about simple controllers by really understanding PID (proportional plus integral plus derivative) controllers. Perhaps the first question to ask is why are PIDs so common? One plausible reason is that engineers are very good at beating problems into a form (formally called redesigning) so that they are dominated by a second-order model. This ties back to those simple models introduced in Section 2.3. Once we have isolated the particular dominant simple model then we can often get a lot of information about the parameters from the simple step response methods described in Section 3.9.

This chapter will start with some observations about the use of PID controllers and some questions that arise from those. It may seem odd that something as “standard” as PID is not very standardized. There are so many different ways of parameterizing the coefficients of the proportional, integral, and derivative terms that three different vendors can sell PID controllers for the same application with seemingly no relationship between how they are parameterized and programmed. We will then work to cast PID controllers into one of a handful of standard forms so that engineers can both do more effective comparisons and also see the benefits and drawbacks of a particular form.

We will also relate PID controllers to a variety of second order filter structures, including notches, leads (single and double), lags, and lag-lead structures.

There will be a discussion of discretization methods. PID controllers are perhaps the most common example of a controller that is designed in continuous time and then specifically discretized (as opposed to starting with a discrete-time system model). Furthermore, PID controllers are unique in that they are almost universally discretized using a backwards rectangular rule equivalent described in Section 3.6.3.

We also go into anti-windup methods used in PI and PID controllers with an eye to understanding which methods are most appropriate in a given situation.

Finally, we will do something that is surprisingly uncommon: We will apply a standard form continuous-time model PID controller to each of the simple models introduced in Section 2.3 and analytically “close-the-loop” so that we can examine the best case behavior of these models under PID control. We look at each of them with proportional (P), proportional plus integral (PI), proportional plus derivative (PD), and full PID control. The results are at the same time intuitive and surprising, as we see very clearly why certain classes of models need certain types of control.

## 4.2 Chapter Ethos

One cannot shake a stick in an industrial control setting without hitting a proportional-integral-derivative (or Proportional plus Integral plus Derivative) (PID) controller. In fact, PIDs are so ubiquitous that feedback controllers are often simply referred to as PIDs even if they are far more sophisticated. PIDs offer the simplicity of a small number of tuning knobs, but often the tuning of PIDs is viewed either as too trivial to consider academically interesting, or too heuristic to apply even the most basic analysis.

It is ironic that if one refers to a controller as a controller, non-engineers will be confused, but if one calls it a PID, they will know what function the block accomplishes. How is it that so many industrial control engineers working on systems both mechatronic and otherwise can simply drop in a PID and get reasonable results. One can argue that they are simply naive or unthinking, but I believe that:

> If something keeps working over and over again in widely varying situations, it is probably not complete nonsense. There is probably a fundamental reason for this. It is worth the effort to understand that fundamental reason and how far it can be applied, i.e., what limits it.

In retrospect, this would seem obvious, yet it is often ignored in practice. Making a gross oversimplification, I will say that my academic friends generally ignore such a simple algorithm as being uninteresting for research, while my industrial friends generally never ask themselves why this thing keeps working. The questions of why these simple things work so well have stayed with me for a long time and I will try to give insight here.

One of the common observations about industrial control systems is that 95–98% of them are PID controllers. That is a lie, because almost all of these PID controllers have the D gain set at or close to $0$, so they are effectively PI controllers. Why is this? If one returns to our low order models of Section 2.3, we find that if one does not know anything, that is if one has very little knowledge of system parameters, one can still control most of these models using a PI controller with a relatively small I gain, and low overall system gain. That means that while there is zero steady state error to steps, the response is slow (low overall gain). This is a downside, but it generally guarantees the controller is not amplifying higher order dynamics. It turns out, most folks have bad models, so they get the system working with PI controllers, turn the knobs until it's right below ringing, and tell their bosses that it's working. The D term can get you into trouble by amplifying high frequency dynamics, and who needs that? It is only in the systems with relative order $2$, such as the double integrator model found in spacecraft movement or disk drive actuators, that requires some phase-lead, and hence a non-zero D.

While PID controllers are the "Brand X" of most control Ph.D. candidates' theses and spent the 1990s being derided by the denizens of fuzzy control, they remain today the most ubiquitous example of feedback controller design, by some measures accounting for 95–98% of all controllers in the field. Rather than dismissing this as an alternative and boring reality, we will examine the underlying implicit assumptions about modeling the physical system – and how those models derive from what can be measured (from Chapter 3), to motivate the generic and fundamental utility of PID controllers. With that context, we will show:

- Some simple control structures: lag, lead, double lead, lag-lead, and PID (Section 4.5).
- A unified framework for discussing PID controllers, which is helpful not only in generating a design, but also in understanding the underlying structures of off-the-shelf, commercial PID controllers. How do PID controllers relate to lead/lag controllers (Section 4.6)?
- A discussion for representing PID controllers in discrete time without losing the intuition of the continuous time framework. Put another way, what's up with Backwards Rectangular Rule discretization (Section 4.10)?
- How PID controllers can be expected to behave in closed-loop for various low order models (Section 4.11).
- Tuning PID controllers: from step response, from frequency response, from generalized Ziegler-Nichols (Section 4.13), from the relay method of Åström et. al. [115] (Section 4.14).
- If one is clever about working in the frequency domain and understands the parameterization of PID controllers, one can use a PID to affect some pretty reasonable loop shaping on systems that are second order or less (Section 4.15). It can even form the basis of loop shaping on higher order systems when combined with the correct filters (Chapter 5). Nevertheless, this starts with extracting the system model parameters, from one of the methods above or from step response or frequency response methods described in Chapter 3.
- Some PID code/pseudo-code examples (Section 4.16).
- A discussion of windup and anti-windup mitigation: why it's needed and what options exist (Section 4.18).

- We will go through some design examples in Section 4.17.

- A special case combination comes when PID controllers are used in conjunction with slow, low pass systems, such as heaters and pumps. In this case, rather than an analog signal output from a Digital to Analog Converter (DAC) or a digital connection, the PID output is scaled to go between $0$ and $1$ and then used to modulate a single digital signal line using pulse-width-modulation (PWM) (Section 4.19).

- A logical, but unexpected use of PID controllers is as an explanation for slow, biological feedback mechanisms. Our understanding of the principles behind and the behavior of a PID controller allow us to recognize PID-like behavior in biological (and other natural) models (Section 4.20).

- Why using “D” in PID often fails to improve performance and how to fix that. Where is “D” most often beneficial?

This chapter aims to walk between those AT (academic/theoretical) and the II (industrial/implementation) worlds, explaining some simple common frameworks for PID controllers, how they are affected by implementation issues (such as discretization), and what the consequences of turning the PID knobs are when the PIDs control a handful of simple physical systems. The hope is to give the reader both an intuitive and mathematically justified understanding of how to get the most from these devices in their daily work.

## 4.3 Chapter Introduction

Understanding feedback loops and how to apply them to physical systems really needs to start with a view of the physical system, its model, and with a view to how a feedback controller ties into the system. Figure 4.1 shows a schematic of a feedback loop using an analog controller, while Figure 4.2 shows a generic digital feedback system. In each of these, the stuff on the right does not change: the physical system behavior must be sensed and can only be affected through actuators. Converting the small signals that do computation into larger signals that can drive motors or open valves requires some actuation. On the other side, sensor signals often require conditioning before they can be used in a meaningful way. On the far left of each is the decision making, computation end of things. As engineers, we want to spend our time doing clever things in the left side of the drawing, but we really need to keep in mind that we need to understand the right side of the drawing and how to get between them to do a good job.

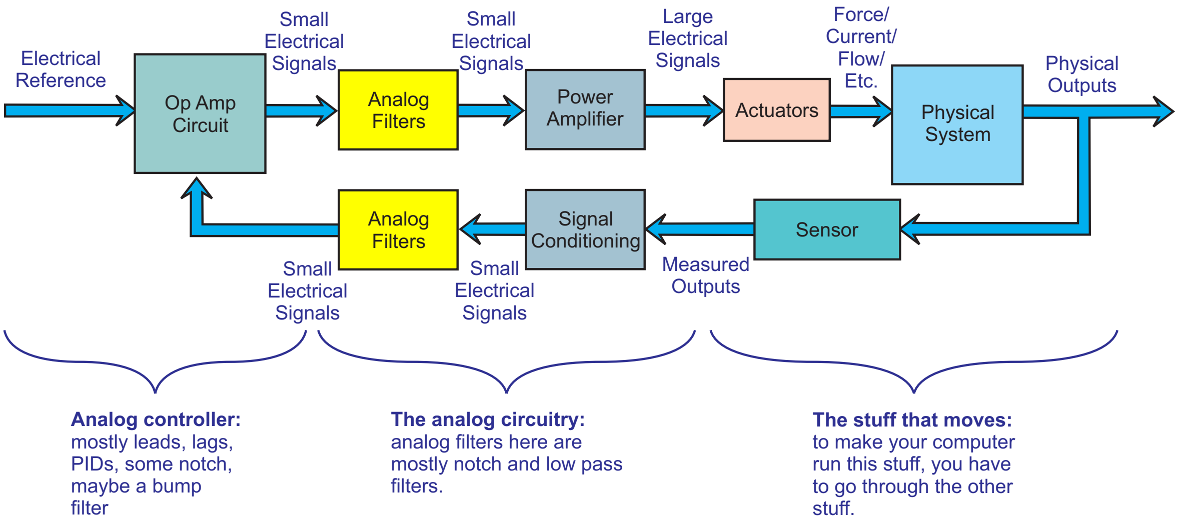

Electrical Reference
Op Amp Circuit
Small Electrical Signals
Analog Filters
Small Electrical Signals
Power Amplifier
Large Electrical Signals
Actuators
Force/ Current/ Flow/ Etc.
Physical System
Physical Outputs
Sensor
Measured Outputs
Signal Conditioning
Small Electrical Signals
Analog Filters
Small Electrical Signals
Analog controller: mostly leads, lags, PIDs, some notch, maybe a bump filter
The analog circuitry: analog filters here are mostly notch and low pass filters.
The stuff that moves: to make your computer run this stuff, you have to go through the other stuff.

Figure 4.1: **A generic overview of an analog control loop. This picture shows the component pieces in an analog feedback loop.**

The main difference in these two drawings is the discretization of signals for the digital side (Figure 4.2). Anyone with familiarity with digital control systems knows that we give up a lot of simplicity and clarity in going from the analog, continuous time world (Figure 4.1), to the digital computer world (Figure 4.2), but we accept this for the repeatability of behavior, simplicity of modification, and richness of data collection that comes with the use of digital computers. It is why NASA chose to use digital computers to guide astronauts to the moon [116, 117].

The reason that this matters is that we want to be aware of the effects of sampling on our control system. Discretization changes our math, but it is understood that when the sample rate is high enough, the effects of discretization can be safely ignored [15, 16]. PID controllers are often applied in such situations, that is, they are often applied in situations where the dynamics of the physical system are far slower than the speed at which the computer can look at and react to the system. In other words, many of these systems are so slow that even the most basic real-time computer such as the Raspberry Pi [60] and the Xilinx Zynq [61] has “high sample rate” compared to the dynamics of these systems. This is why so much of the literature on PID controllers focuses only on the continuous time world [115, 118, 119]. Even when the PIDs are considered as computer control components [120], the consequences of discretization are rarely discussed.

This chapter will discuss the common discretization method applied to PID controllers, give a framework that simplifies the translation between the analog and digital versions, and show a some of the effects that happen when our discretization is not fast enough to ignore the effects of sampling.

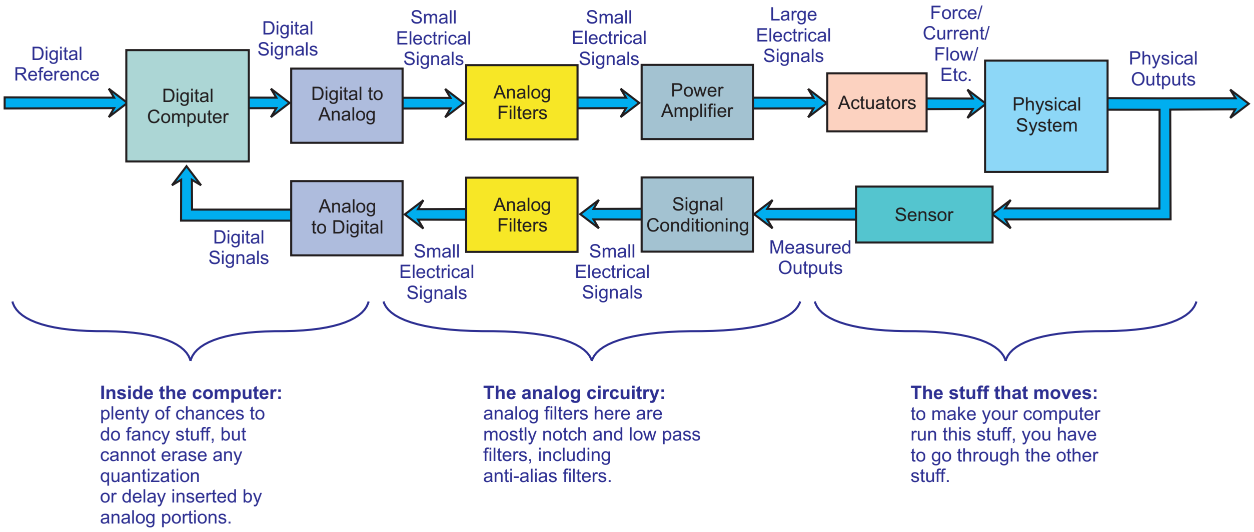

Digital Reference
Digital Computer
Digital Signals
Digital to Analog
Small Electrical Signals
Analog Filters
Small Electrical Signals
Power Amplifier
Large Electrical Signals
Actuators
Force/ Current/ Flow/ Etc.
Physical System
Physical Outputs
Sensor
Measured Outputs
Signal Conditioning
Small Electrical Signals
Analog Filters
Small Electrical Signals
Analog to Digital
Digital Signals
Inside the computer: plenty of chances to do fancy stuff, but cannot erase any quantization or delay inserted by analog portions.
The analog circuitry: analog filters here are mostly notch and low pass filters, including anti-alias filters.
The stuff that moves: to make your computer run this stuff, you have to go through the other stuff.

Figure 4.2: **A generic overview of digital control loop. This picture shows the component pieces in a digital feedback loop.**

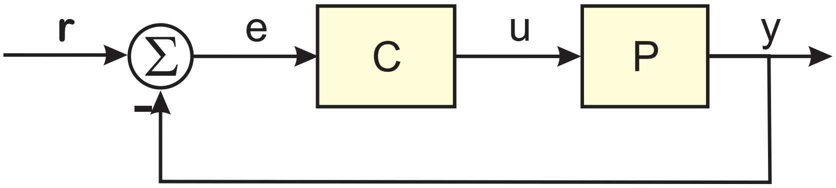


Figure 4.3: **A generic block diagram feedback loop.**

A generic feedback loop in which we ignore the effects of sampling and fold the effects of actuation, filtering, amplification, and signal conditioning into the plant – or assume that they are negligible – is shown in Figure 4.3. For the sake of simplifying basic analysis we stick with the diagram of Figure 4.3, which is feedback only. We will discuss integrating in feedforward control in Chapter 8.

Often, it is the case that the controller block, $C$, is implemented with a Proportional-Integral-Derivative (PID) controller. Increasingly, these controllers are digital, implemented on computers, as shown in Figure 4.1, but analyzed using the simplified thinking of Figure 4.3.

PID controllers are so ubiquitous that more generic controllers are often simply referred to as PIDs to people outside the field. One cannot examine most industrial control environments without tripping

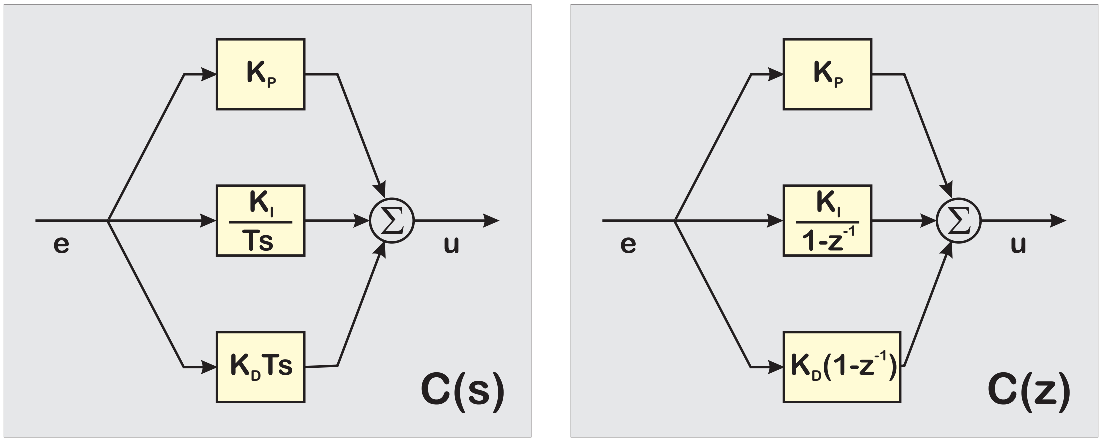


Figure 4.4: **A parallel form topology of a simple analog & digital PID controller. We will discuss this more around Figure 4.6.**

over simple controllers of all sorts of specifications broadly labeled as PIDs. A parallel form of a continuous time PID is depicted on the left side of Figure 4.4, with a similarly structured digital PID on the right side.

This chapter attempts to place PID controllers into the unified framework provided in [40]. We focus on the most common discretization method applied to PIDs and show how one of the most common forms of analog PID can be combined with the most common discretization to make for simple translation between analog and digital PID controllers.

It is also important to realize that PID controllers are almost always applied to low order models or at least to the low order behavior of more complex systems. We will give examples of several archetypal simple models in Section 2.3. This will set us up to analyze the closed-loop behavior of these models in Section 4.11. We will see that by knowing the basic model structure, we can predict which PID forms have a chance of working, and we can also explain why certain forms show up in different applications.

## 4.4 What is a PID and When is It Useful?

While most readers are likely to to know the answer to the question in the heading, it's best that we answer here, "What is a PID?" Very simply it is a (mostly) linear controller that involves three versions of the error signal:

- a signal proportional to the error,
- a signal proportional to the integral of the error, and
- a signal proportional to the derivative of the error.

These three components can be applied serially [119], but this may be a holdover from early implementation methodologies. Here we will (at least for now) stick with parallel realizations of PID controllers as diagrammed in Figure 4.4.

It should be obvious that there are many parameterizations that accomplish the above three actions, and we will try to give a useful and unifying framework in Section 4.6. Here we will preview Equations 4.30 and 4.36 here as they are simple and understandable. A simple continuous time PID controller is described by

$$u(t) = K_P e(t) + K_{IT} \int_0^t e(\tau) d\tau + K_{DT} \dot{e}(t), \tag{4.1}$$

in the time domain, where $e(t)$ error input to the controller and $u(t)$ is the controller In the frequency domain the forms for $C(s) = \frac{U(s)}{E(s)}$ is:

$$C(s) = K_P + \frac{K_{IT}}{s} + K_{DT} s. \tag{4.2}$$

PID controllers have enough degrees of freedom to control any of the simple systems in Section 2.3 of the Introduction chapter. In fact, one of the great things about a PID controller is that for simple control problems, we can turn on only the portions we need by proper use of the different gains.

Proportional control works well in some cases (e.g. when there is an integrator already in the plant model and the relative degree of the denominator polynomial to the numerator is 1, with all the poles and zeros in the left half plane), but with no integrator in the plant, it cannot achieve zero steady state tracking error to a step input. When we let the integrator gain be nonzero, we have a proportional-integral (or proportional plus integral) (PI) controller.

On the other hand, the classic double integrator (Section 2.3.5) rarely needs more integration, but can't be stabilized with only proportional control, so we set the integrator gain to $0$ and make the differentiator gain positive, giving us proportional-derivative (or proportional plus derivative) (PD) control. proportional plus derivative (PD) control. As we really don't want to build a pure differentiation circuit, we filter the derivative, resulting in the more rational lead filter.

The point is that by properly turning the PID "knobs," most of the desired closed-loop behaviors of physical systems described by simple, low order models (Section 2.3) can be achieved. This point becomes even stronger when we realize that in many, many practical systems, engineers work very hard to simplify the control problem until it is second order or less. In other words, PIDs can handle a lot of practical problems.

### 4.4.1 Recalling the Final Value Theorem

The final value theorem (FVT) relates the limit time value of some response to the zero-frequency, DC response of a transform. For control systems, it's main use is to tell us how many integrators we need in the forward path of a feedback loop in order to achieve $0$ steady state error to some level of input (step, ramp, quadratic, etc.). In continuous time, the theorem says if a function $f(t)$ is bounded for $t \in (0, \infty)$ and the $\lim_{t\to\infty} f(t)$ is bounded, then

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s), \tag{4.3}$$

where $F(s)$ is the Laplace transform of $f(t)$. In discrete time,

$$\lim_{k\to\infty} f(k) = \lim_{z\to 1}(z-1)F(z), \tag{4.4}$$

where $F(z)$ is the causal Z transform of $f(k)$.

So, why mention this theorem here? Because it is the reason why almost every unity feedback control loop attempts to have at least one integrator in either the plant or the controller. Working in the Laplace domain, consider the transfer function, $G(s) = P(s)C(s)$. For a unity feedback then we have:

$$E(s) = \frac{1}{1+G(s)}. \tag{4.5}$$

Now, to use the Final Value Theorem, we look at the Laplace transform of different inputs. We consider an impulse, a step, a ramp, and some higher order function of $t$. It is standard to have all but the impulse multiplied by a unit step, $1(t)$ which is $0$ for $t < 0$ and $1$ for $t >= 0$.

$$\begin{aligned}
\mathcal{L}\{\delta t\} &= 1, && (4.6)\\
\mathcal{L}\{1(t)\} &= \frac{1}{s}, && (4.7)\\
\mathcal{L}\{t1(t)\} &= \frac{1}{s^2}, \text{ and} && (4.8)\\
\mathcal{L}\{t^N 1(t)\} &= \frac{N!}{s^{N+1}}. && (4.9)\\
& && (4.10)
\end{aligned}$$

Now, to use the FVT, we want the final value of $e(t) * r(t)$, where $r(t)$ is the reference input to the closed-loop system, we have

$$\lim_{t\to\infty} e(t) * r(t) = \lim_{s\to 0} sE(s)R(s) = s\left(\frac{1}{1+G(s)}\right)R(s). \qquad (4.11)$$

Let's say $r(t)$ is a step of height $K$. Then

$$\begin{aligned}
\lim_{s\to 0} sE(s)R(s) &= \lim_{s\to 0} s\left(\frac{K}{s}\right)\left(\frac{1}{1+G(s)}\right) && (4.12)\\
&= K\lim_{s\to 0}\left(\frac{1}{1+G(s)}\right). && (4.13)
\end{aligned}$$

To use the FVT we have to assume that $G(s)$ is stable, although we allow poles on the $j\omega$ axis. Now, if $G(s)$ is strictly stable, that is if it has no poles on the $j\omega$ axis, then $G(0)$ exists, so the final value of the error is nonzero, but finite, and given by:

$$\lim_{s\to 0} sE(s)R(s) = \frac{K}{1+G(0)}. \qquad (4.14)$$

Note that the error gets smaller with higher DC gain of the system, so as $G(0)$ gets larger the steady state error gets lower. Adding an integrator makes the DC value infinite, so if $G(s)$ contains an integrator, then we can separate it as $G(s) = \frac{1}{s}\tilde{G}(s)$. This means that

$$\begin{aligned}
\lim_{s\to 0} sE(s)R(s) &= \lim_{s\to 0} K\left(\frac{s}{s+\tilde{G}(s)}\right) && (4.15)\\
&= 0. && (4.16)
\end{aligned}$$

Right here, we have the reason why integrators show up in so many control loops: With an integrator in the loop, we can get zero steady state error to a step input. For many simple models, a PI or a PID controller is enough to provide that integrator term and thus, gets the system to track a step.

How do we extrapolate this? Let:

$$G(s) = \frac{1}{s^M}\tilde{G}(s). \tag{4.17}$$

Then

$$\begin{aligned}
\lim_{s\rightarrow 0} sE(s)R(s) &= \lim_{s\rightarrow 0} s\left(\frac{K}{s^N}\right)\left(\frac{s^M}{s^M+\tilde{G}(s)}\right), && (4.18)\\
&= K\lim_{s\rightarrow 0}\left(s^{M-N+1}\right)\left(\frac{s^M}{s^M+\tilde{G}(s)}\right), && (4.19)\\
&= \infty,\ \text{for } M-N+1<0, && (4.20)\\
&= \frac{K}{\tilde{G}(0)},\ \text{for } M-N+1=0\ \&\ M\neq 0, && (4.21)\\
&= \frac{K}{1+G(0)},\ \text{for } M-N+1=0\&M=0,\ \text{and} && (4.22)\\
&= 0,\ \text{for } M-N+1>0. && (4.23)
\end{aligned}$$

Put another way, the number of integrators in the forward path of the unity feedback loop determines the order of the signal that the loop can track with zero steady state error. The higher the order of the input, the more integrators needed to achieve zero steady state error. For discrete time systems, the $\frac{1}{s}$ integrator is replaced by a $\frac{1}{1-z^{-1}}$ discrete integrator, and thus the open loop Z-domain model needs discrete integrators to achieve zero steady state error.

In a discrete-time representation, the final value theorem also applies, albeit with discrete integrators.

Repeat FVT for discrete time

## 4.5 Lags, Leads, Lag-Leads, Double Leads, and the Like

In Chapter 2 we introduced the bilinear filter as a physical system model, but as with Shimmer (Floor Wax/Dessert Topping – you all will want to Google that), the same model can also serve as a controller circuit. Repeating Equation 2.6:

$$C(s) = \frac{U(s)}{E(s)} = K\left(\frac{a_{1c}}{b_{1c}}\right)\left(\frac{s+b_{1c}}{s+a_{1c}}\right) \tag{4.24}$$

Back in the days of control via analog circuits, such a circuit was used to stabilize systems by providing band limited differentiation. It is also the form that an analog lag filter takes. The key difference is that for a lead, $b_{1c} < a_{1c}$ and for a lag $b_{1c} > a_{1c}$. One prominent form of a lag is to set $a_{1c} = 0$, making the circuit an integrator with a zero, which is completely equivalent to a PI controller.

Also common, especially when we are starting with a double integrator (Section 2.3.5) plant that also may have some high frequency resonances, is a double lead circuit, i.e.

$$C(s) = \frac{U(s)}{E(s)} = K\left(\frac{a_{1c}a_{2c}}{b_{1c}b_{2c}}\right)\left(\frac{s+b_{1c}}{s+a_{1c}}\right)\left(\frac{s+b_{2c}}{s+a_{2c}}\right) \tag{4.25}$$

where a double lead requires $b_{1c}$ & $b_{2c} < a_{1c}$ & $a_{2c}$.

We will see in a feedforward example in Chapter 8 that when the closed-loop design produces a nice low-pass-filter behavior, we can extend the input-output bandwidth by using a double lead as the closed-loop input feedforward filter ($F_{CLI}$) (Section 8.6).

We can implement Equation 4.25 as a biquad (short for biquadratic filter) by simplifying it as:

$$C(s) = K\left(\frac{a_{1c}a_{2c}}{b_{1c}b_{2c}}\right)\left(\frac{s^2+(b_{1c}+b_{2c})s+b_{1c}b_{2c}}{s^2+(a_{1c}+a_{2c})s+a_{1c}a_{2c}}\right), \tag{4.26}$$

or finally

$$C(s) = b_0\left(\frac{s^2+\tilde{b}_1 s+\tilde{b}_2}{s^2+a_1 s+a_2}\right). \tag{4.27}$$

Here

$$\begin{aligned} a_1 &= a_{1c}+a_{2c} & a_2 &= a_{1c}a_{2c}, \\ \tilde{b}_1 &= b_{1c}+b_{2c}, & \tilde{b}_2 &= b_{1c}b_{2c}, \text{ and } b_0 = K\left(\frac{a_{1c}a_{2c}}{b_{1c}b_{2c}}\right). \end{aligned}$$

In our example, we started with real pole and zero locations for the controller, but we do not need to restrict ourselves that way. This author's prior work [108] as well as the independent work of Messner et. al. [121, 122, 123] show great advantages in implementing this biquad with complex roots. The latter work restricts itself to the continuous time domain, while the former includes discretization.

We discuss these here because most PID controllers can be thought of as second order filters – with an important caveat for idealized continuous time PIDs. Furthermore, if one turns off parts of the PID, one returns either a lead controller (for $K_I = 0$) or a lag controller (for $K_D = 0$).

## 4.6 PID Control: A Unified Framework

Five basic versions of analog PID control equations show up in the control literature [20, 118, 122, 123, 119, 57, 124, 15, 16, 119, 16, 120, 14] and in commercial PID controllers. In the time domain representation those forms are:

$$u(t) = K\left(e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau + T_D\dot{e}(t)\right), \tag{4.28}$$

$$u(t) = K_P e(t) + \frac{K_I}{T_I}\int_0^t e(\tau)d\tau + K_D T_D\dot{e}(t), \tag{4.29}$$

$$u(t) = K_P e(t) + K_{IT}\int_0^t e(\tau)d\tau + K_{DT}\dot{e}(t), \tag{4.30}$$

$$u(t) = K_P e(t) + \frac{K_I}{T_I}\int_0^t e(\tau)d\tau + K_D T_D\dot{x}_1(t), \tag{4.31}$$

$$u(t) = K_P e(t) + K_{IT}\int_0^t e(\tau)d\tau + K_{DT}\dot{x}_2(t), \tag{4.32}$$

where $e(t)$ error input to the controller, $u(t)$ is the controller output, and

$$\dot{x}_1 = \dot{e} - \frac{a_1}{T_D}x_1 \quad \text{and} \quad \dot{x}_2 = \dot{e} - a_1 x_2. \tag{4.33}$$

In the frequency domain the five forms for $C(s) = \frac{U(s)}{E(s)}$ are:

$$C(s) = K\left(1 + \frac{1}{T_I s} + T_D s\right), \tag{4.34}$$

$$C(s) = K_P + \frac{K_I}{T_I s} + K_D T_D s, \tag{4.35}$$

$$C(s) = K_P + \frac{K_{IT}}{s} + K_{DT}s, \tag{4.36}$$

$$C(s) = K_P + \frac{K_I}{T_I s} + K_D \frac{T_D s}{T_D s + a_1}, \tag{4.37}$$

$$C(s) = K_P + \frac{K_{IT}}{s} + K_{DT} \frac{s}{s + a_1}. \tag{4.38}$$

The first form [119], in Equations 4.28 and 4.34 are often associated with chemical process control (CPC), which involve slower processes often modeled as first or second order systems, or first order plus time delay (FOPTD). These are alternately known as first order plus dead time (FOPDT) . In these cases, the time constants are so slow that any modern processing system can easily sample 20–100× faster than the highest frequency in the physical system. Thus, the integration time $T_I$ and the differentiation time $T_D$ are used as gain constants but it is bad nomenclature. These terms should be related to the actual time over which one wants to integrate and differentiate, rather than as generic tunable knobs.

For ease of explanation, we will keep to the frequency domain forms. In 4.37 and 4.38 we have chosen the derivative filter gain so that – in combination with the derivative – it has a high frequency gain of $1$. We could also have chosen to a filter with DC gain of $1$. Some books also apply a low pass filter around the entire controller of Equations 4.35 and 4.36 [118]. The four forms are chosen by picking two options:

- explicit time specification and
- differentiator filtering.

Explicit time specification simply refers to whether the $T_I$ and $T_D$ terms are present, or whether they are absorbed into $K_I$ and $K_D$, respectively. It is perfectly legitimate to have

$$K_{IT} = \frac{K_I}{T_I} \quad \text{and} \quad K_{DT} = K_D T_D, \tag{4.39}$$

where $K_{IT}$ and $K_{DT}$ can be considered “implicit time” versions of the integral and differential gains. Alternately, the designer can easily go from explicit to implicit time simply by setting $T_D = T_I = 1$. However, leaving the $T_I$ and $T_D$ terms in the equation give the designer some flexibility and also allow these terms to drop out when the discrete-time PID is generated. In particular, for the backward rule equivalent of an ideal PID controller with the sample period, $T = T_I = T_D$, the time terms drop out of the equation, making it appear much simpler.

The second option is differentiator filtering. We know that any practical analog differentiator will eventually roll off. That is, real world circuits and devices are low pass after some frequency and so a pure differentiator stage must attenuate at high frequency in this time-space continuum. It should make sense to explicitly include this in the controller design, but this is not common. Perhaps designers are expecting the plant dynamics and/or circuits to provide low pass behavior. Still, one might wonder why use of a low-pass derivative filter is not a standard practice. The authors' best guess is that the most common implementation of a PID controller is a backward rule discrete equivalent approximation. As we will see in Section 4.10.1, this equivalent puts in its own low pass filter on the differentiator. The typical over-conservatism of the backwards rule equivalent saves the casual designer the trouble of thinking about low pass filter design and will tend to behave well, especially at low frequencies. It is natural to think that we need only filter the differentiator, since that is the only non-proper term, but we will discuss doing this versus putting a low pass filter on the entire PID in Section 4.7.

Understanding these four basic forms are useful to a user that has purchased a system that includes a PID controller *e.g.* the controller of a motion control system. Invariably, the user trying to model these systems will find that one of these forms has been used without it being documented in the product literature. Likewise, technical papers on PID controllers will often default to one of these forms without any discussion about the particular choice Because of this, it is pretty common to see PID gain ranges that vary all over the place, even for the same basic controller. These 4 forms are summarized in Table 4.1.

| **Form** | **Time Dom. Equation** | **Freq. Dom. Equation** |
|---|---|---|
| Time Constant, No Filtering | (4.28) | (4.34) |
| Explicit Time, No Filtering | (4.29) | (4.35) |
| Implicit Time, No Filtering | (4.30) | (4.36) |
| Explicit Time, Deriv. Filtering | (4.31) | (4.37) |
| Implicit Time, Deriv. Filtering | (4.32) | (4.38) |

Table 4.1: **A summary of the four basic forms of analog PID control with references to their associated time domain and frequency domain equations.**

It should be obvious that we can put these separate terms into one transfer function. What may not be obvious is how this will look once it is combined. Sections 4.9 –4.10 discuss this.

Equations (4.35)–(4.38) can all be related to second order sections and these can be used for loop shaping. We will focus on (4.35) and (4.37) since setting $T_I = T_D = 1$ gets these to (4.36) and (4.38). Thus, by setting the parameters of the PID, we can set the parameters of the notch. Note that the parameterization will change quite a bit depending upon if we use (4.35) or (4.37), and this is because the filter in (4.37) is acting only on the differentiator.

In this chapter, we will not lose too much generality by assuming that our fundamental starting point is a PID controller in the form shown on the left side of Figure 4.4. Tuning the controller often starts with the proportional feedback term, $K_P$, and then integral action from $K_I$ is added in. The derivative term, $K_D$, finds far less usage in practice, although this seems to be short sighted if one understands the tradeoffs between noise amplification and signal lead, and applies the proper filtering.

## 4.7 Derivative Filtering Versus Whole PID Filtering

Looking at any of the unfiltered analog PID formulas in Section 4.6, e.g. Equations 4.28, 4.34, 4.29, 4.35, 4.30, and 4.36, we see that they are not proper. They have more zeros than poles if $K_D = 0$. This is almost universal in PID formulas and yet it seems that nobody worries about it, at least from a “properness” point of view. Filtering of the derivative or the whole PID does get a page in [118], mostly as a way of limiting noise amplification in the controller. For the most part, though, the texts seem largely unconcerned. I think that there are two basic reasons for this. The historical one is that older analog PIDs would have been implemented in circuits or pneumatics which had low pass characteristics. In effect, any PID implementation would add in low pass which would make the effective model proper, even if the controller equations were not. The more modern reason is that – as will be discussed later – the most typical discretization of PID controllers is via a backwards rectangular rule, and this implements the differentiator with a pole at $z = 0$.

However, given higher bandwidth analog circuits and/or the possibility of using a trapezoidal rule discretization, it is worthwhile to get a clear mental picture of the use of a low pass filter in the PID to make it proper. We will only discuss a single pole analog low pass,

$$L(s) = \frac{a}{s+a}, \tag{4.40}$$

and compare applying this to the entire PID versus only applying it to the derivative section (D term, the part that really needs it).

First, some intuition. The PI portion of the PID is in effect in the lower frequency range of the controller – from DC up to some zero location. Most typically, the D portion is active at the higher frequency range of the PID. That means that whether we apply our single pole $L(s)$ on the entire PID or just on the D term, the effective behavior of the controller will be largely the same. This is definitely not true if we forget and apply a higher order low pass to the entire PID as described in [118]. Mentally (and emotionally, perhaps ecumenically) it really makes the most sense to separate the single pole low pass filter that makes the PID proper from the other low pass filters that should be considered as part of a loop shaping scheme. When one has a more complex controller (e.g. PID plus filters) then the extra LPF needs to be viewed as part of the higher order controller design.

Applying $L(s)$to the entire PID from Equation 4.35

$$\begin{aligned} C_{F1}(s) &= \left(K_P + \frac{K_I}{T_I s} + K_D T_D s\right)\left(\frac{a}{s+a}\right), && (4.41)\\ &= \left(\frac{K_D T_D T_I s^2 K_P T_I s + K_I}{T_I s}\right)\left(\frac{a}{s+a}\right), && (4.42)\\ &= K_D T_D \left(\frac{a}{s+a}\right)\left(\frac{s^2 + \frac{K_P}{K_D T_D} s + \frac{K_I}{K_D T_D T_I}}{s}\right). && (4.43) \end{aligned}$$

Applying $L(s)$to the D term as done in Equation 4.37

$$\begin{aligned} C_{F2}(s) &= K_P + \frac{K_I}{T_I s} + K_D \frac{T_D a s}{T_D s + a}, && (4.44)\\ &= \frac{K_P T_I s(s+a) + K_I(s+a) + K_D T_D T_I a s^2}{T_I s(s+a)}, && (4.45)\\ &= \frac{s^2(K_P T_I + K_D T_D T_I a) + s(K_P T_I a + K_I) + K_I a}{T_I s(s+a)}, && (4.46)\\ &= \left(\frac{K_P + K_D T_D a}{s+a}\right)\left(\frac{s^2 + \frac{(K_P T_I a + K_I)}{T_I(K_P + K_D T_D a)} s + \frac{K_I a}{T_I(K_P + K_D T_D a)}}{s}\right), && (4.47)\\ &= (K_P/a + K_D T_D)\left(\frac{a}{s+a}\right)\left(\frac{s^2 + \frac{K_P + \frac{1}{a} K_I/T_I}{K_P/a + K_D T_D} s + \frac{K_I/T_I}{K_P/a + K_D T_D}}{s}\right). && (4.48) \end{aligned}$$

We can see that although they have similar forms and accomplish the same things, the whole PID filter version of Equation 4.43 seems considerably simpler than the derivative only form of Equation 4.48. If we use the implicit time versions where $K_{I,i} = K_I/T_I$ and $K_{D,i} = K_D T_D$, then

$$\begin{aligned} C_{F1}(s) &= K_{D,i}\left(\frac{a}{s+a}\right)\left(\frac{s^2 + \frac{K_P}{K_{D,i}} s + \frac{K_{I,i}}{K_{D,i}}}{s}\right) \text{ and} && (4.49)\\ C_{F2}(s) &= (K_P/a + K_{D,i})\left(\frac{a}{s+a}\right)\left(\frac{s^2 + \frac{K_P + \frac{1}{a} K_{I,i}}{K_P/a + K_{D,i}} s + \frac{K_{I,i}}{K_P/a + K_{D,i}}}{s}\right). && (4.50) \end{aligned}$$

So long as the frequency of the low pass, i.e. $a = 2\pi f_a \Longleftrightarrow f_a = \frac{a}{2\pi}$, is reasonably well above the zero of the PI section, thew two forms will behave similarly and $C_{F1}(s)$ is more intuitive.

Either way we filter, we can relate it to the lag and lead filters of Section 4.5. First, we start with a PI controller as a lag filter:

$$PI(s) = K_{P1}\left(\frac{s+b_I}{s}\right). \tag{4.51}$$

The PI filter does not change with different versions of derivative filtering. On the other hand, the PD filter, which to be practical, must be implemented as a lead filter, can take two forms. The first is consistent with whole PID filtering:

$$PD_1(s) = K_{P3}a\left(\frac{s+b_1}{s+a}\right). \tag{4.52}$$

The other form with derivative only filtering looks like

$$\begin{aligned}
PD_2(s) &= K_{P2} + \frac{K_{D2}sa}{s+a}, && (4.53)\\
&= \frac{K_{P2}(s+a) + K_{D2}sa}{s+a}, && (4.54)\\
&= \frac{(K_{P2}+K_{D2}a)s + K_{P2}a}{s+a}, && (4.55)\\
PD_2(s) &= (K_{P2}+K_{D2}a)\left(\frac{s+\frac{K_{P2}a}{K_{P2}+K_{D2}a}}{s+a}\right). && (4.56)
\end{aligned}$$

Combining these into respective filtered PIDs, for the whole PID filtering form, we get:

$$\begin{aligned}
PI(s)PD_1(s) &= K_{P1}\left(\frac{s+b_I}{s}\right)K_{P3}a\left(\frac{s+b_1}{s+a}\right), && (4.57)\\
&= K_{P1}K_{P3}\left(\frac{a}{s+a}\right)\left(\frac{(s+b_I)(s+b_1)}{s}\right), && (4.58)\\
PI(s)PD_1(s) &= K_{P1}K_{P3}\left(\frac{a}{s+a}\right)\left(\frac{s^2+(b_I+b_1)s+b_Ib_1}{s}\right). && (4.59)
\end{aligned}$$

Likewise, for the derivative filtering form, we get:

$$\begin{aligned}
PI(s)PD_2(s) &= K_{P1}\left(\frac{s+b_I}{s}\right)(K_{P2}+K_{D2}a)\left(\frac{s+\frac{K_{P2}a}{K_{P2}+K_{D2}a}}{s+a}\right), && (4.60)\\
&= \frac{K_{P1}(K_{P2}+K_{D2}a)}{s}\left(\frac{(s+b_I)\left(s+\frac{K_{P2}a}{K_{P2}+K_{D2}a}\right)}{s+a}\right), && (4.61)
\end{aligned}$$

$$PI(s)PD_2(s) = \frac{K_{P1}(K_{P2}+K_{D2}a)}{s}\left(\frac{s^2+\left(b_I+\frac{K_{P2}a}{K_{P2}+K_{D2}a}\right)s+\frac{b_IK_{P2}a}{K_{P2}+K_{D2}a}}{s+a}\right). \qquad (4.62)$$

We can equate coefficients from Equations 4.26 or 4.27 to either the form in Equation 4.59 or 4.59 if we want, but the main part of this section is to get some intuition for the equivalences between different forms.

## 4.8 PID Regions

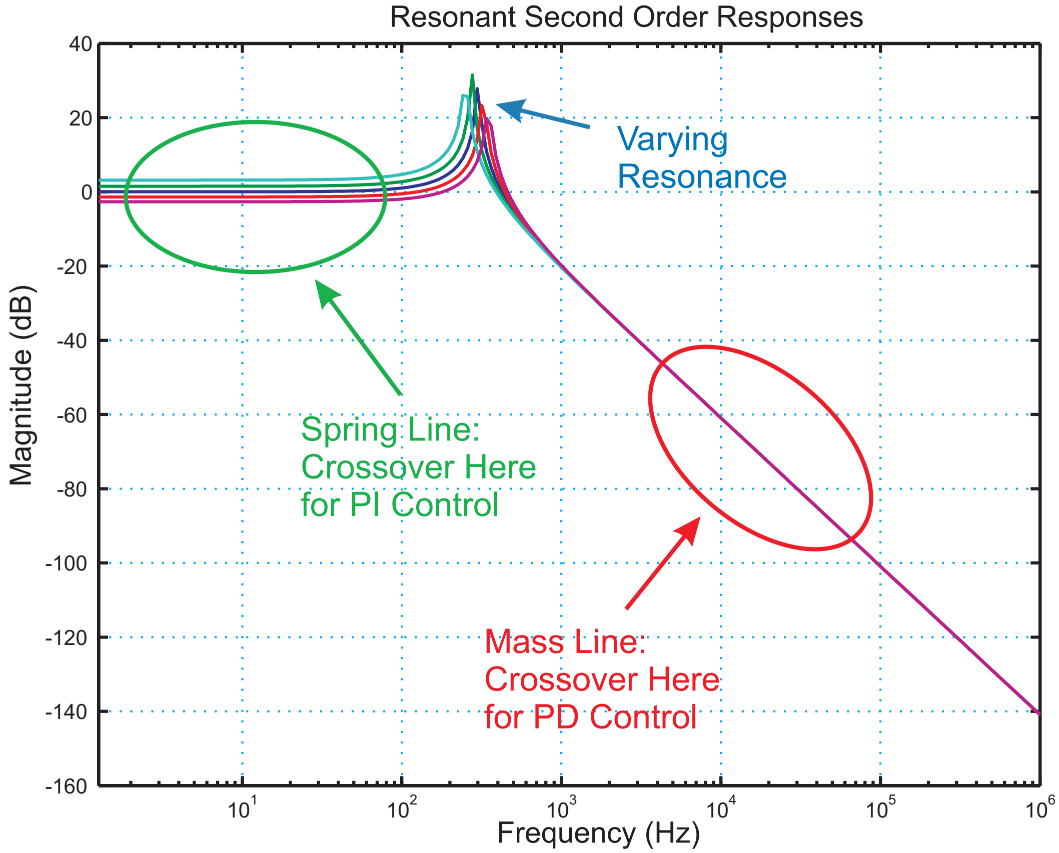


Figure 4.5: **The regions of PID control for a second order, resonant plant. Such a Bode plot often characterizes mechatronic systems, flexible structures, or electronic circuits.**

In many practical uses, one can consider a PID controller to be operating on a second order or lower plant, such as the one diagrammed in Figure 4.5. In many cases, the control action is taken and removed well below the resonance, and in this case the proportional plus integral (PI) part of the controller is used, to make the open loop response look like an integrator near gain crossover, and then remove the phase effects before the $-180^\circ$ effect of the resonance. In these cases, the derivative term is seldom used. Similarly, when the resonance is well below the crossover region, the system can be controlled as if it were a double integrator, and in this case the proportional plus derivative (PD) action is used. In these cases, there may still be a motivation to use integral action at low frequency and thus a PID with derivative filtering resembles separate lead and lag controllers. In neither case is precise knowledge of the resonance needed. It is only when the crossover region is relatively close to the resonance that more complex methods, such as those described in [108, 122, 123] become important. The formulas that follow concentrate on the latter, more complex case, while emphasizing the effects of digital implementation as described in [108]. In the case of first order time delay models, such as those found in process control applications [20], full PID control is used, where the PI portion provides low frequency gain and the PD portion adds some lead to compensate for the phase due to the delay, the the requirements of matching a high Q resonance are not present.

## 4.9 Unfiltered Analog PID and Second Order Sections

Starting with (4.35), let's set $T_D = T_I = T$ and put everything over a common denominator:

$$C(s) = K_P + \frac{K_I}{Ts} + K_D T s, \tag{4.63}$$

$$\begin{aligned} C(s) &= \frac{1}{Ts}\left[K_P T s + K_I + K_D (Ts)^2\right], && (4.64)\\ &= \frac{K_D}{Ts}\left[(Ts)^2 + \frac{K_P}{K_D} T s + \frac{K_I}{K_D}\right], && (4.65)\\ &= \frac{K_D T}{s}\left[s^2 + \frac{K_P}{K_D}\frac{s}{T} + \frac{K_I}{K_D T^2}\right], && (4.66)\\ &= \frac{K_D T s^2 + K_P s + \frac{K_I}{T}}{s}. && (4.67) \end{aligned}$$

While (4.66) allows us to solve for the numerator parameters as a second order section, (4.67) is a standard numerator/denominator form that we might use in Matlab. This form of the PID is not proper, but this is typically mitigated by the way PID controllers are implemented.

Practically speaking, nothing gets realized this way for one of two reasons:

1) Any real differentiation circuit eventually flattens out, so there is some low pass filter, even if it's not acknowledged in the design.

2) Discretizing the PID using a backwards rectangular rule fixes this and makes the discrete-time transfer system proper. It forces a low pass digital filter with a pole at $z = 0$.

The numerator of (4.66) also has the form of a second order section i.e.,

$$N(s) = \frac{K}{\omega_n^2}\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right), \tag{4.68}$$

so we should be able to set

$$\frac{K_D T}{s}\left[s^2 + \frac{K_P}{K_D}\frac{s}{T} + \frac{K_I}{K_D T^2}\right] = \frac{K}{\omega_n^2 s}\left[s^2 + 2\zeta\omega_n s + \omega_n^2\right]. \tag{4.69}$$

If we were simply to try to match $N(s)$ in (4.68) then we might choose to match the DC gain to some prespecified value, $K$. However, the form that we want our PID to match has infinite DC gain, so we need to pick a frequency and gain that we wish to match and then evaluate the right side of (4.69). If:

$$\frac{N(s)}{s} = \frac{K}{s\omega_n^2}\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right), \tag{4.70}$$

then

$$\frac{N(j\omega_0)}{j\omega_0} = \frac{K}{j\omega_0\omega_n^2}\left(\omega_n^2 - \omega_0^2 + j\frac{\omega_n\omega_0}{Q}\right), \tag{4.71}$$

where $Q = \frac{1}{2\zeta}$. If we pick our desired gain, $K_0$, at a certain frequency, $\omega_0 = 2\pi f_0$, then we get

$$K_0 = \left|\frac{N(j\omega_0)}{j\omega_0}\right| = \frac{K}{\omega_0\omega_n^2}\sqrt{(\omega_n^2 - \omega_0^2)^2 + \left(\frac{\omega_n\omega_0}{Q}\right)^2}. \tag{4.72}$$

This can be solved for $K$ via

$$K = \frac{K_0\omega_0\omega_n^2}{\sqrt{(\omega_n^2 - \omega_0^2)^2 + 4\zeta^2\omega_n^2\omega_0^2}} = \frac{K_0\omega_0\omega_n^2}{\sqrt{(\omega_n^2 - \omega_0^2)^2 + \left(\frac{\omega_n\omega_0}{Q}\right)^2}}. \tag{4.73}$$

Using (4.73) to pick $K$ allows us to equate terms in (4.69) gives us:

$$K_D T = \frac{K}{\omega_n^2}, \tag{4.74}$$

$$\omega_n^2 = \frac{K_I}{K_D T^2} \quad \text{and} \tag{4.75}$$

$$2\zeta\omega_n = \frac{\omega_n}{Q} = \frac{K_P}{K_D T}. \tag{4.76}$$

We can solve for $\omega_n$, $\zeta$, and $Q$: Using (4.73) to pick $K$ allows us to equate terms in (4.69) allows us to solve for $\omega_n$, $\zeta$, and $Q$:

$$\omega_n = \frac{1}{T}\sqrt{\frac{K_I}{K_D}},\ \zeta = \frac{K_P}{2\sqrt{K_I K_D}},\ \text{and} \tag{4.77}$$

$$Q = \frac{1}{2\zeta} = \frac{\sqrt{K_I K_D}}{K_P}. \tag{4.78}$$

However, for design, we might want to specify $\omega_n$ and $\zeta$ or $Q$ and then re-derive the PID gains as a function of those parameters. This should give us a better way of picking $K_P$, $K_I$, and $K_D$, if we know which center frequency and damping we want the controller numerator to achieve.

Let's set $\omega_n$. We also know $T$, which is our integration and differentiation time, but will also be our sample time. Finally, we set $Q$ (which is equivalent to setting $\zeta$). From (4.69) we have

$$\frac{K_I}{K_D} = (\omega_n T)^2 \text{ and } \frac{K_P}{K_D} = \frac{\omega_n T}{Q} = 2\zeta\omega_n T. \tag{4.79}$$

If we now let $K_D$ be our overall controller gain, scaling $K_D$ means scaling $K_P$ and $K_I$ in the same proportions to maintain the desired shape of the compensator. In summary pick $K$ from (4.73) to set the controller gain. Then

$$K_D = \frac{K}{T\omega_n^2},\ K_I = KT, \text{ and } K_P = \frac{K}{Q\omega_n}. \tag{4.80}$$

It is worth noting that this particular analog PID controller is not a real physically realizable device since it is impossible to design a true analog differentiator that works over all frequencies. More likely is the case that there actually is some high pass filter on the analog differentiator, but that this is at a frequency that is high enough to be ignored from the perspective of the control design. However, it does have implications for the implementation of a discrete PID as will be discussed in Section 4.10.

In particular, if we use a backwards rectangular rule discrete equivalent, then we get a physically realizable controller because the backwards rule places an extra pole at $z = 0$ in the differentiator section, making it realizable. Furthermore, the discrete PID coefficients of that form will be very closely related to the continuous PID coefficients of Equation 4.63.

# 4.10 Discrete PID

More complex approximations to discretization will produce more accurate results, but since they are often linked to the use of more points, the latency of the method goes up. For the three discrete integration rules discretization methods used with PID controllers Table 3.1 is helpful. While the Trapezoidal Rule is by far the most accurate of these methods, the issue of a pure differentiator in the PID equation (unfiltered) creates a problem, and thus it is most common for the Backwards Rectangular Rule to be used.

An ideal PID without differentiator filtering (from (4.35)) can be discretized using a backward rectangular rule but not the trapezoidal rule. The usually conservative backward rule equivalent allows the use of an unfiltered derivative PID design, since the continuous derivative maps to a zero at $z = 1$ and a pole at $z = 0$. We can't apply the trapezoidal rule to the unfiltered differentiator of (4.35), because the $z + 1$ term in the denominator results in a pole at $z = -1$, which will be an internal oscillatory pole in the compensator. So, while the closed-loop system might be stable, we wouldn't have internal stability.

The filtered differentiator of (4.37) can be implemented using either backward rectangular or trapezoidal rule as will be shown later.

## 4.10.1 Backward Rectangular Discrete PID

Applying the backward rectangular rule to (4.35) yields

$$C(z) = K_P + \frac{K_I T z}{T_I(z-1)} + K_D T_D \frac{z-1}{Tz}, \tag{4.81}$$

and setting $T = T_I = T_D$ we get

$$C(z) = K_P + K_I \frac{z}{z-1} + K_D \frac{z-1}{z}. \tag{4.82}$$

In terms of $z^{-1}$ this is

$$C(z) = K_P + K_I \frac{1}{1-z^{-1}} + K_D(1-z^{-1}). \tag{4.83}$$

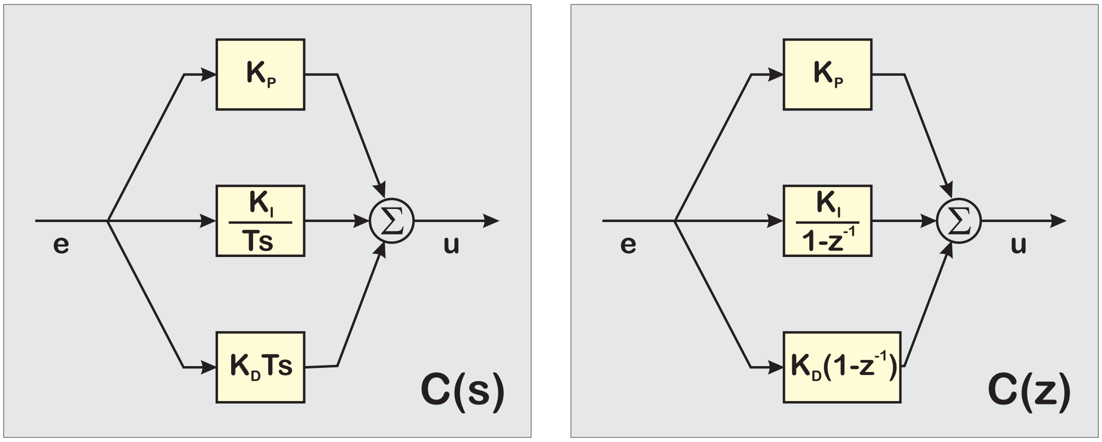


Figure 4.6: **The structural similarity between the explicit time continuous PID with $T_D = T_I = T$ and the discretization of that PID using the Backwards Rectangular Rule and a sample period of $T$.**

Note that (4.82) and (4.83) have discrete PID gains that are trivially related to the analog PID gains through the sample period, $T$. The similarity to Equation 4.29 is striking, as seen in Figure 4.6. Equation (4.83) is useful for generating the time domain difference equation in 3 separate units, proportional, integral, and derivative. It is the equation from which we would program this controller, as it would make it easy to add an anti-windup piece to just the integral portion despite being harder to analyze in the $z$ domain. We can rewrite (4.82), though, as:

$$C(z) = \frac{K_P(z-1)(z) + K_I z^2 + K_D(z-1)^2}{z(z-1)}, \tag{4.84}$$

$$C(z) = \frac{b_0 z^2 - b_1 z + b_2}{z^2 - z}, \text{ where} \tag{4.85}$$

$$b_0 = K_P + K_I + K_D, \tag{4.86}$$

$$b_1 = 2K_D + K_P, \text{ and } b_2 = K_D. \tag{4.87}$$

Using (4.85), we can examine the discrete-time properties of the linear model of this PID in Matlab.

### 4.10.2 Notes on Backwards Rule Discrete PID

Using the backwards rule discrete equivalent on the continuous time PID of Equation 4.63, that is, unfiltered, explicit time PID, with $T = T_D = T_I$, we end up with the discrete PID of Equations 4.83 and

4.84.

Now, while the backwards rule is overly conservative, for fast sample rates, it faithfully reproduces the behavior of the continuous system, so in many, many PID applications, where the sample rate, $f_S = 1/T$ is much faster than the dynamics we wish to control, we can implement the controller using Equation 4.84, but do our analysis using Equation 4.63. This is what we will do in Section 4.11.

## 4.11 Closed-Loop Responses

In this section, we will take some of the simple models of Section 2.3 and apply an analog PID controller to them, closing the loop with unity gain. We will extract the closed-loop transfer function symbolically from these simple systems, which will allow us to see how, when the PID gains are adjusted, we can predict their effect on the overall system response.

Looking at Equation 4.63, we will define $K_{IT} = \frac{K_I}{T_I} = \frac{K_I}{T}$ and $K_{DT} = K_D T_D = K_D T$. The analog PID controller becomes

$$\begin{aligned} C_{PID}(s) &= K_P + \frac{K_{IT}}{s} + K_{DT}s & (4.88) \\ &= \frac{K_{DT}}{s}\left(s^2 + \frac{K_P}{K_{DT}}s + \frac{K_{IT}}{K_{DT}}\right). & (4.89) \end{aligned}$$

We don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than or equal to $0$. If $K_{DT}$ is $0$ we have proportional plus integral (PI) control:

$$C_{PI}(s) = K_P + \frac{K_{IT}}{s} = \frac{K_P\left(s + \frac{K_{IT}}{K_P}\right)}{s}. \tag{4.90}$$

If $K_{IT}$ is $0$ we have proportional plus derivative (PD) control:

$$C_{PD}(s) = K_P + K_{DT}s = K_{DT}\left(s + \frac{K_P}{K_{DT}}\right). \tag{4.91}$$

Finally, if both $K_{IT}$ and $K_{DT}$ are $0$ we have proportional (P) control:

$$C_P(s) = K_P. \tag{4.92}$$

There are situations in which pure integral (I) control or pure derivative (D) control are used, but these are much more rare.

### 4.11.1 Closed-Loop PID on an Integrator

In this section, we will apply various PIDs to the integrator of Section 2.3.1, given in Equation 2.1

$$\frac{X(s)}{F(s)} = H(s) = \frac{K}{s} \tag{4.93}$$

#### Proportional Control

Using proportional control we have

$$\begin{aligned} T(s) &= \frac{PC}{1+PC} && (4.94)\\ &= \frac{\frac{K_P K}{s}}{1+\frac{K_P K}{s}} && (4.95)\\ T(s) &= \frac{K_P K}{s+K_P K} && (4.96)\end{aligned}$$

This is a nice, simple result. Proportional feedback on a pure integrator produces an ideal first order low pass filter that can have arbitrary bandwidth set by choosing the correct $K_P$. Note in the Bode plot of Figure 4.7, this idealized system has $90^\circ$ phase margin and infinite gain margin.

Perhaps the easiest open-loop system to control is an analog integrator, $K/s$. In the absence of delay, it has infinite gain margin and $90^\circ$ phase margin (PM). As plotted in Figure 4.7, we see that the sensitivity function, $S$, has no peaking and the complimentary sensitivity function, $T$, is an ideal low-pass filter, also with no peaking. Even the addition of a control filter is simple. With the desire for high phase margin any filter action would be removed long before gain crossover. The most likely

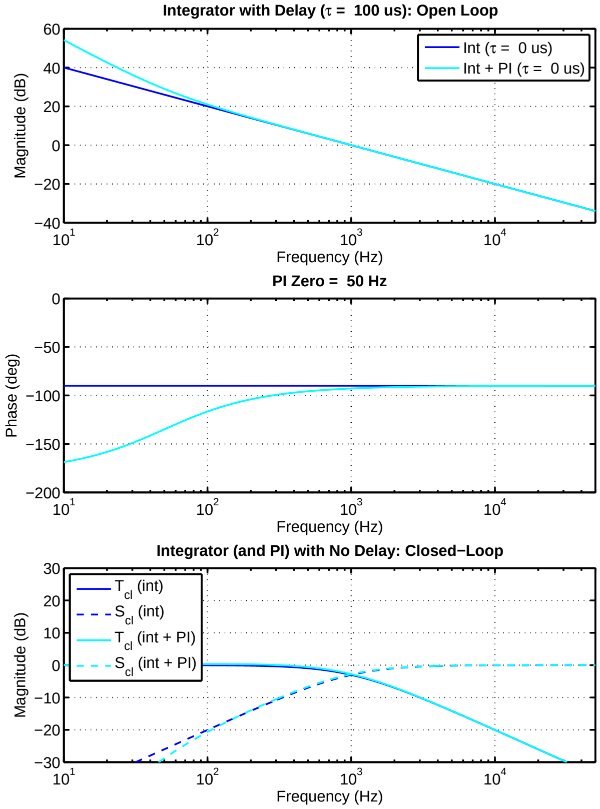


Figure 4.7: Plot of open-loop integrator + PI control, without time delay. Essentially, if the PI action is far below crossover, the effects on stability can be ignored, and they usually are. Note the well behaved closed-loop sensitivity and complimentary sensitivity, owing to the 90° phase margin.

controller is a lag filter where the pole may or may not be an integrator (PI control). Such a system is usually stable even when discretized and saturated subject to any extra delay in the discretization [125]. This explains the seeming lack of analysis done in phase-locked loop (PLL) work. Even PID control of second-order systems can be viewed as an attempt to close the loop on an open-loop integrator [108]. However that example also illustrates many of the difficulties involved in “turning the open loop into an integrator”.

### PI Control

Using proportional plus integral (PI) control we get

$$T(s) = \frac{PC}{1+PC} \tag{4.97}$$

$$= \frac{\frac{K_P K}{s}\left(s + \frac{K_{IT}}{K_P}\right)}{1 + \frac{K_P K}{s}\left(s + \frac{K_{IT}}{K_P}\right)} \tag{4.98}$$

$$T(s) = \frac{K_P K\left(s + \frac{K_{IT}}{K_P}\right)}{s^2 + K_P K s + K K_{IT}} \tag{4.99}$$

We don't lose much generality to assume that $K$, $K_P$, and $K_{IT}$ are all greater than $0$. If $K_{IT}$ is $0$ we are back to the case in Section 4.11.1.

The closed-loop zero is at $s = -\frac{K_{IT}}{K_P}$. As $K_{IT}$ increases relative to $K_P$ the derivative action that usually adds stability shows up at higher frequency. For the poles we use the quadratic formula on the denominator:

$$z_{1,2} = \frac{-KK_P \pm \sqrt{(KK_P)^2 - 4KK_{IT}}}{2} \tag{4.100}$$

Since $K$ and $K_P$ are both positive, the closed-loop poles are stable. They can become oscillatory unless the term under the radical stays $\geq 0$. This is guaranteed if $KK_P^2 \geq 4K_{IT}$. If $K_{IT} = 0$ then we have two poles at $s = -KK_P$ and at $s = 0$ but the latter is canceled out by the zero at $s = 0$.

In summary, PI control on an integrator will result in a stable closed-loop, but if the integrator gain is too high relative to the proportional gain, that closed loop response will exhibit ringing. The plots in Figure 4.7 confirm that so long as $K_{IT}$ is low enough, the effects of the integrator are gone far below open loop crossover and the system has close to $90°$ phase margin and infinite gain margin.

We have not spent much time on systems with time delay, but it is easy enough to analyze for such a simple physical system. Our happy results of Figure 4.7 get disturbed when we add some time delay, as shown in Figure 4.8. It is worth the effort to look at simple delay calculations for a system whose open loop is an integrator.

From this, pure delay, one can add the negative phase effects of delay as:

$$D(j\omega) = e^{-j\omega T_D} \text{ with angle } \angle D(j\omega) = -\omega T_D. \tag{4.101}$$

With phase margin, PM, in degrees, our open-loop phase looks like:

$$-\omega T_D + \angle\frac{K}{s} \geq (-180 + PM)\frac{\pi}{180} \text{ or} \tag{4.102}$$

$$\omega T_D = 2\pi f T_D \leq (90 - PM)\frac{\pi}{180}, \text{ so} \tag{4.103}$$

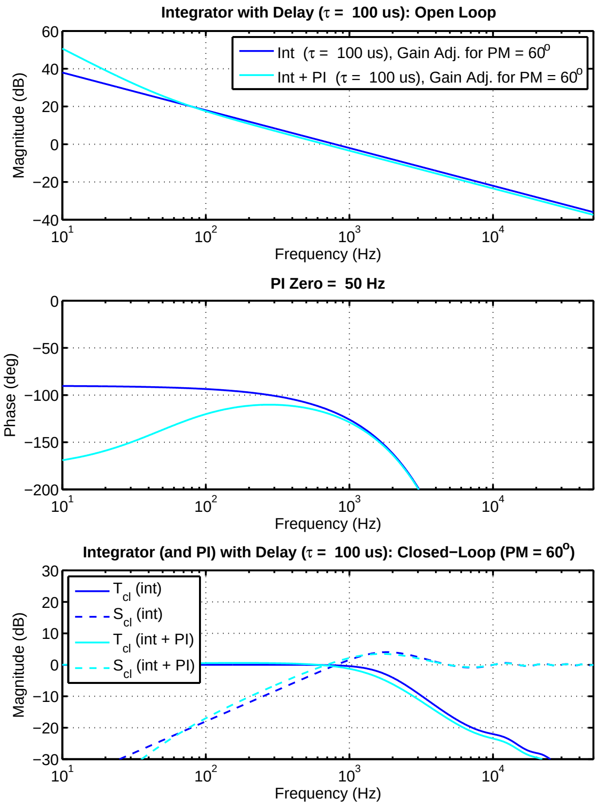


Figure 4.8: Plot of open-loop integrator + PI control, with time delay. The effects of pure time delay limit the bandwidth that can be achieved with 60° phase margin and add some peaking in the sensitivity function.

e

$$f \leq \frac{90 - PM}{360 T_D}. \tag{4.104}$$

In particular, for a desired phase margin, Equation 4.104 gives the highest crossover frequency for the open-loop gain plot of the integrator, using the delay of $T_D$. For the conservative phase margin of 60° we have the simple formula of

$$f \leq \frac{1}{12 T_D}. \tag{4.105}$$

In an ideal world, where the data conversions, computation, and communication are instantaneous, if we use the average sampling delay of $T_D = \frac{T_S}{2}$, and the conservative phase margin of 60° we have the simple formula of

$$f \leq \frac{1}{6 T_S} = \frac{f_S}{6}. \tag{4.106}$$

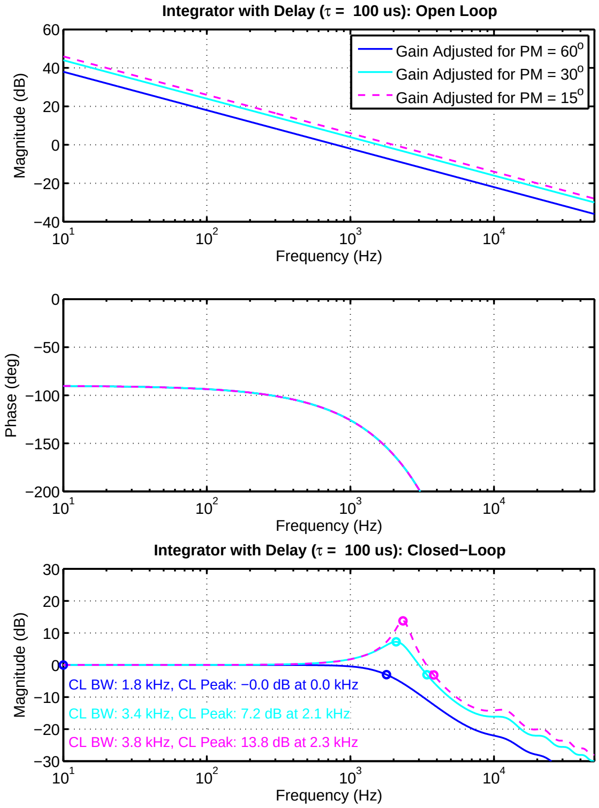


Figure 4.9: Plot of open-loop integrator, with fixed time delay. This plot shows the consequence of adjusting the open-loop gain crossover to achieve phase margins of 15°, 30°, and 60°. Note the closed-loop peaking that results from pushing the open-loop gain crossover to the point of such low phase margin.

Why spend much time on such a trivial plant? First it is a ubiquitous plant ([40, 30]). Furthermore it gives us a representation of the best case plant that we can control and what limits it. No matter what else we do in our controller, it will be hard to improve on the open-loop crossover frequency limit in Equation 4.105. We can see that even with such a simple open loop, with a given $T_D$, if we choose bandwidth over phase margin, we are subject to the closed-loop peaking shown in Figure 4.9. It seems unlikely that any more complicated open-loop system will do any better. Thus, if we pick the conservative phase margin of 60°, we then get an open-loop crossover limit based on that, and this limits our closed-loop bandwidth, as shown in Figure 4.10.

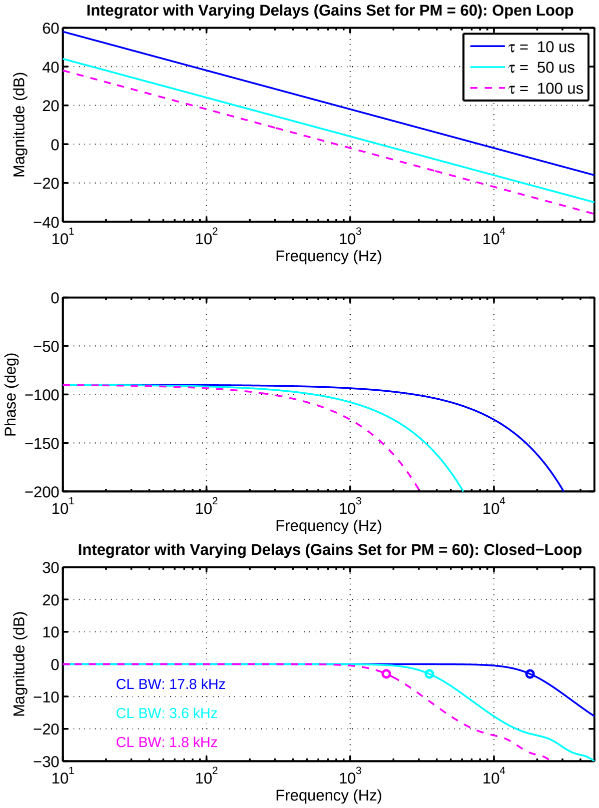


Figure 4.10: Plot of open-loop integrator, with varying time delay. Gain is adjusted for maximum open-loop crossover that achieves 60° phase margin. The seemingly obvious result shown is that for an open-loop response that looks like an integrator with time delay. It is the time delay that limits the closed-loop bandwidth for a desired phase margin.

### PD Control

Using proportional plus derivative (PD) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than 0. If $K_{DT}$ is 0 we are back to the case in Section 4.11.1.

$$T(s) = \frac{PC}{1+PC} \tag{4.107}$$

$$= \frac{\frac{K_{DT}K}{s}\left(s+\frac{K_P}{K_{DT}}\right)}{1+\frac{K_{DT}K}{s}\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.108}$$

$$= \frac{K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)}{s+K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.109}$$

$$= \frac{K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)}{(1+K_{DT}K)s+KK_P} \tag{4.110}$$

$$T(s) = \frac{\frac{K_{DT}K}{1+K_{DT}K}\left(s+\frac{K_P}{K_{DT}}\right)}{s+\frac{KK_P}{1+K_{DT}K}} \tag{4.111}$$

$$\tag{4.112}$$

Now, this is first order, but has both a stable pole and a minimum phase zero. This means that the closed-loop response will behave as a lead or a lag depending upon the relative sizes of $K$, $K_P$, and $K_{DT}$. We know in fact that a pure differentiator is not realizable in the real world so in fact there must be some roll off in our true system that we are modeling here. However, even this result indicates that we may not want to design a system in which we have not designed the closed-loop roll off. It is not clear that a PD controller does much good on a pure integrator.

### PID Control

Using proportional plus integral plus derivative (PID) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ or $K_{IT}$ are $0$ we are back to one of our prior cases.

$$T(s) = \frac{PC}{1+PC} \tag{4.113}$$

$$= \frac{\frac{K_{DT}K}{s^2}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{1+\frac{K_{DT}K}{s^2}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} \tag{4.114}$$

$$= \frac{K_{DT}K\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^2(1+K_{DT}K)+KK_Ps+KK_{IT}} \tag{4.115}$$

$$T(s) = \frac{\frac{K_{DT}K}{1+K_{DT}K}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^2+\frac{KK_P}{1+K_{DT}K}s+\frac{KK_{IT}}{1+K_{DT}K}} \tag{4.116}$$

So, with a single pole differential in the plant, the derivative term (i.e. $K_{DT} \neq 0$) ends up making a net $0$ pole-zero excess difference in the closed loop. This means that the closed-loop response does not roll off. For simple integrator plants, $P$ or $PI$ control seem to be the answer.

### 4.11.2 Closed Loop PID on a First Order Low Pass

In this section, we will apply various PIDs to the integrator of Section 2.3.3, given in Equation 2.5

$$\frac{X(s)}{F(s)} = \frac{Ka}{s+a}e^{-sT_D}. \tag{4.117}$$

We will simplify this by setting $T_D = 0$, knowing full well that any significant value of $T_D$, relative to the time constants of the system, will make things a lot harder. Thus we have,

$$\frac{X(s)}{F(s)} = \frac{Ka}{s+a}. \tag{4.118}$$

#### Proportional Control

Using proportional control we get

$$\begin{aligned} T(s) &= \frac{PC}{1+PC} && (4.119)\\ &= \frac{\frac{K_PKa}{s+a}}{1+\frac{K_PKa}{s+a}} && (4.120)\\ &= \frac{K_PKa}{s+(a+K_PKa)} && (4.121)\\ T(s) &= \frac{K_PKa}{s+(1+K_PK)a} && (4.122)\end{aligned}$$

This is a nice, simple result. Proportional feedback on a first order low pass filter produces a first order low pass filter that can have arbitrary bandwidth set by choosing the correct $K_P$. However, the presence of $a$ means that the DC ($s = 0$) gain is always less than $1$.

#### PI Control

Using proportional plus integral (PI) control we get

$$T(s) = \frac{PC}{1+PC} \tag{4.123}$$

$$= \frac{\frac{K_P K a}{s(s+a)}\left(s + \frac{K_{IT}}{K_P}\right)}{1 + \frac{K_P K a}{s(s+a)}\left(s + \frac{K_{IT}}{K_P}\right)} \tag{4.124}$$

$$T(s) = \frac{K_P K a\left(s + \frac{K_{IT}}{K_P}\right)}{s(s+a) + K_P K a\left(s + \frac{K_{IT}}{K_P}\right)} \tag{4.125}$$

$$T(s) = \frac{K_P K a\left(s + \frac{K_{IT}}{K_P}\right)}{s^2 + (1 + K_P K)as + K K_{IT} a} \tag{4.126}$$

We don't lose much generality to assume that $K$, $K_P$, and $K_{IT}$ are all greater than $0$. Furthermore, for a stable low pass filter, $a \geq 0$. If $K_{IT}$ is $0$ we are back to the case in Section 4.11.2. For $a = 0$, we are back to Section 4.11.1.

The closed-loop zero is at $s = -\frac{K_{IT}}{K_P}$. As $K_{IT}$ increases relative to $K_P$ the derivative action that usually adds stability shows up at higher frequency. For the poles we use the quadratic formula on the denominator:

$$z_{1,2} = \frac{-(1 + K K_P)a \pm \sqrt{(1 + K K_P)^2 a^2 - 4 K K_{IT} a}}{2} \tag{4.127}$$

Since $a$, $K$ and $K_P$ are all positive, the closed-loop poles are stable. They can become oscillatory unless the term under the radical stays $\geq 0$. This is guaranteed if $K a K_P^2 \geq 4 K_{IT}$. If $K_{IT} = 0$ then we have two poles, one at $s = -(1 + K K_P)a$ and at $s = 0$ but the latter is canceled out by the zero at $s = 0$.

In summary, PI control on an first order low pass will result in a stable closed-loop, but if the integrator gain is too high relative to the proportional gain, that closed loop response will exhibit ringing.

### PD Control

Using proportional plus derivative (PD) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ is $0$ we are back to the case in Section 4.11.2. For $a = 0$, we are back to Section 4.11.1.

$$T(s) = \frac{PC}{1 + PC} \tag{4.128}$$

$$= \frac{\frac{K_{DT}Ka}{s+a}\left(s+\frac{K_P}{K_{DT}}\right)}{1+\frac{K_{DT}Ka}{s+a}\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.129}$$

$$= \frac{K_{DT}Ka\left(s+\frac{K_P}{K_{DT}}\right)}{s+a+K_{DT}Ka\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.130}$$

$$= \frac{K_{DT}Ka\left(s+\frac{K_P}{K_{DT}}\right)}{(1+K_{DT}Ka)s+KK_Pa+a} \tag{4.131}$$

$$T(s) = \frac{\frac{K_{DT}Ka}{1+K_{DT}Ka}\left(s+\frac{K_P}{K_{DT}}\right)}{s+\frac{(KK_P+1)a}{1+K_{DT}Ka}} \tag{4.132}$$

Now, this is first order, but has both a stable pole and a minimum phase zero. This means that the closed-loop response will behave as a lead or a lag depending upon the relative sizes of $a$, $K$, $K_P$, and $K_{DT}$. Again, we know that a pure differentiator is not realizable in the real world so there must be some roll off in our true system that we are modeling here. Again, this result indicates that we may not want to design a system in which we have not designed the closed-loop roll off. It is not clear that a PD controller does much good on a first order low pass filter.

### PID Control

Using proportional plus integral plus derivative (PID) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ or $K_{IT}$ are $0$ we are back to one of our prior cases.

$$T(s) = \frac{PC}{1+PC} \tag{4.133}$$

$$= \frac{\frac{K_{DT}Ka}{s(s+a)}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{1+\frac{K_{DT}Ka}{s(s+a)}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} \tag{4.134}$$

$$= \frac{K_{DT}Ka\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s(s+a)+K_{DT}Ka\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} \tag{4.135}$$

$$= \frac{K_{DT}Ka\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^2(1+K_{DT}Ka)+s(a+K_PKa)+K_{IT}Ka} \tag{4.136}$$

$$T(s) = \frac{\frac{K_{DT}Ka}{1+K_{DT}Ka}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^2+a\frac{1+K_PK}{1+K_{DT}Ka}s+\frac{K_{IT}Ka}{1+K_{DT}Ka}} \tag{4.137}$$

So, with a single pole differential in the plant, the derivative term (i.e. $K_{DT} \neq 0$) ends up making a net $0$ pole-zero excess difference in the closed loop. As with $PD$ control, this means that the closed-loop response does not roll off. For simple integrator plants, $P$ or $PI$ control seem to be the answer.

### 4.11.3 Closed Loop PID on a Double Integrator

In this section, we will apply various PIDs to the double integrator of Section 2.3.6, given in Equation 2.16:

$$\frac{X(s)}{F(s)} = \frac{K}{s^2} e^{-sT_D} \tag{4.138}$$

We will simplify this by setting $T_D = 0$ to get:

$$\frac{X(s)}{F(s)} = \frac{K}{s^2} \tag{4.139}$$

#### Proportional Control

Using proportional control we have

$$\begin{aligned} T(s) &= \frac{PC}{1+PC} && (4.140) \\ &= \frac{\frac{K_P K}{s^2}}{1+\frac{K_P K}{s^2}} && (4.141) \\ T(s) &= \frac{K_P K}{s^2 + K_P K} && (4.142) \end{aligned}$$

Proportional feedback on a pure double integrator an oscillatory closed-loop system with poles on the $j\omega$ axis. We need some lead (i.e. differentiation) to make this system stable, which doesn't bode well for the idea of using PI control on the double integrator.

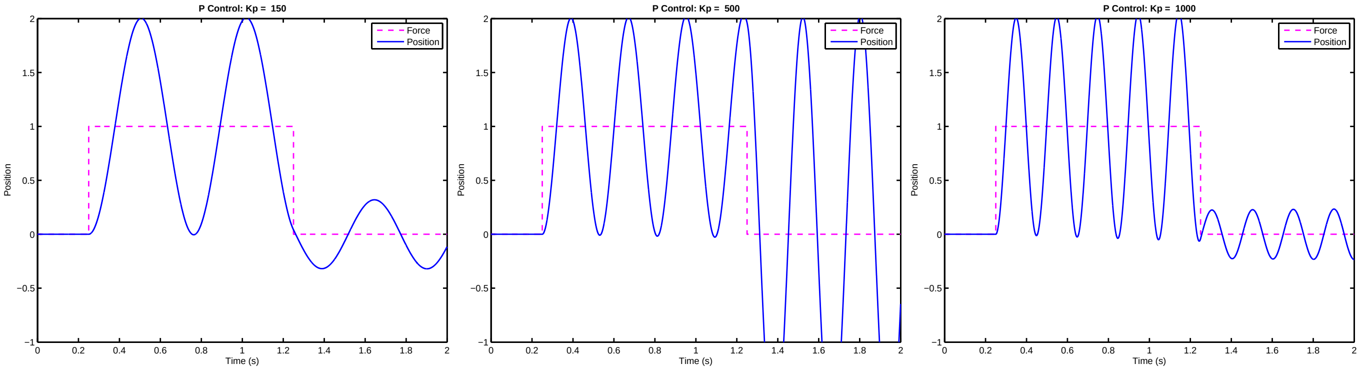


Figure 4.11: Plot of proportional control on double integrator simulation, with $K_P = 150$ (left), $K_P = 500$ (center), and $K_P = 1000$ (right). As expected, it doesn't work and results in an oscillation as predicted by Equation 4.142.

### PI Control

Using proportional plus integral (PI) control we get

$$
\begin{aligned}
T(s) &= \frac{PC}{1+PC} && (4.143)\\
&= \frac{\frac{K_P K}{s^2}\left(s+\frac{K_{IT}}{K_P}\right)}{1+\frac{K_P K}{s^2}\left(s+\frac{K_{IT}}{K_P}\right)} && (4.144)\\
T(s) &= \frac{K_P K\left(s+\frac{K_{IT}}{K_P}\right)}{s^3+K_P K s+K K_{IT}} && (4.145)
\end{aligned}
$$

As predicted, the lack of lead doomed this controller to failure as well. The denominator roots are not stable, since there is a missing coefficient, and Routh's Stability Criterion [57, 56] states that all the coefficients of the polynomial must be present and of the same sign.

### PD Control

Using proportional plus derivative (PD) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ is $0$ we are back to the case in Section 4.11.3.

$$
T(s) = \frac{PC}{1+PC} \qquad (4.146)
$$

$$= \frac{\frac{K_{DT}K}{s^2}\left(s+\frac{K_P}{K_{DT}}\right)}{1+\frac{K_{DT}K}{s^2}\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.147}$$

$$= \frac{K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)}{s^2+K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.148}$$

$$T(s) = \frac{K_{DT}K\left(s+\frac{K_P}{K_{DT}}\right)}{s^2+K_{DT}Ks+K_PK} \tag{4.149}$$

The closed-loop zero is at $s = -\frac{K_P}{K_{DT}}$. As $K_P$ increases relative to $K_{DT}$ the derivative action that usually adds stability shows up at higher frequency. For the poles we use the quadratic formula on the denominator:

$$z_{1,2} = \frac{-K_{DT}K \pm \sqrt{(K_{DT}K)^2 - 4K_PK}}{2} \tag{4.150}$$

With $K$, $K_{DT}$, and $K_P$ all greater than $0$ this is always stable, as one would expect from applying a lead to a double integrator. (In this case PD control can be considered a lead circuit with no flattening in the controller model Bode plot.) The poles are real so long as

$$K_{DT}^2K \geq 4K_P, \tag{4.151}$$

so too much $K_P$ relative to $K_{DT}$ and $K$ can cause stable, but ringing behavior in the closed-loop response.

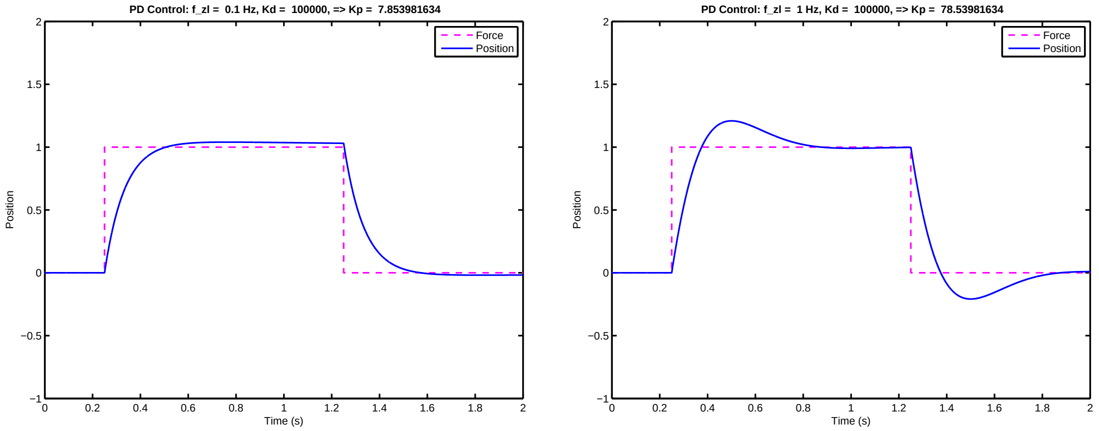


Figure 4.12: Plot of proportional + derivative control on double integrator simulation, with $K_D = 100,000$ and $K_P = 7.8531981634$ (left) and $K_P = 78.531981634$ (right). As expected, it works as predicted by Equation 4.149. Increasing $K_P$ relative to $K_{DT}$ produces a faster response but ringing.

### PID Control

Using proportional plus integral plus derivative (PID) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ or $K_{IT}$ are $0$ we are back to one of our prior cases.

$$\begin{aligned}
T(s) &= \frac{PC}{1+PC} && (4.152)\\
&= \frac{\frac{K_{DT}K}{s^3}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{1+\frac{K_{DT}K}{s^3}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} && (4.153)\\
&= \frac{K_{DT}K\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^3+K_{DT}K\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} && (4.154)\\
&= \frac{K_{DT}K\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^3+K_{DT}Ks^2+KK_Ps+KK_{IT}} && (4.155)
\end{aligned}$$

In order to check stability, we can use the Routh-Hurwitz Stability Criterion [56, 57], on the denominator polynomial. The trivial checks are that all coefficients are present and of the same sign, which we get from our assumption that $K$, $K_{DT}$, $K_P$, and $K_{IT}$ are greater than $0$. The second step is o construct the Routh array, which ends up reducing to the following. If

$$P(s) = s^3 + a_1 s^2 + a_2 s + a_3, \qquad (4.156)$$

then the Routh Array requires that

$$\frac{a_1 a_2 - a_3}{a_1} > 0. \qquad (4.157)$$

We have already checked that $a_1 > 0$, so this becomes:

$$a_1 a_2 > a_3. \qquad (4.158)$$

In our case, this means,

$$K_{DT}K_PK^2 > KK_{IT}, \qquad (4.159)$$

or

$$K_{DT}K_PK > K_{IT}. \qquad (4.160)$$

Put into words, the level of integral action is limited by the amount of proportional and derivative action. Integral action is helpful with steady state response, but since the system already starts out as a double integrator, this isn't a major issue. Derivative action is still needed to stabilize this system.

### 4.11.4 Closed Loop PID on a Simple Resonance

In this section, we will apply various PIDs to the simple resonance with no zeros of Section 2.3.8, given in Equation 2.19:

$$\frac{X(s)}{F(s)} = P(s) = K\frac{\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2} \tag{4.161}$$

Note that we are using the resonance parameters to define the plant, since these seem far more physically instructive than simply a set of generic coefficients. When we apply the full PID controller, we can also parameterize that as an anti-resonance (in the numerator).

#### Proportional Control

Using proportional control we have

$$\begin{aligned}
T(s) &= \frac{PC}{1+PC} && (4.162)\\
&= \frac{\frac{K_PK\omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}}{1+\frac{K_PK\omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}} && (4.163)\\
&= \frac{K_PK\omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2+K_PK\omega_d^2} && (4.164)\\
T(s) &= \frac{K_PK\omega_d^2}{s^2+2\zeta_d\omega_d s+(1+K_PK)\omega_d^2} && (4.165)
\end{aligned}$$

So, if the original resonance was stable, proportional feedback keeps the closed-loop system stable. In fact, one can look at the denominator of Equation 4.165, and see that we have a new resonance with undamped natural frequency,

$$\tilde{\omega}_d = \omega_d\sqrt{1+K_PK}. \tag{4.166}$$

Since the coefficient of $s$ in the denominator has not changed, we have

$$\begin{aligned}
2\zeta_d\omega_d &= 2\left(\frac{\zeta_d}{\sqrt{1+K_PK}}\right)\left(\omega_d\sqrt{1+K_PK}\right) && (4.167)\\
&= 2\tilde{\zeta}_d\tilde{\omega}_d, \text{ where} && (4.168)\\
\tilde{\zeta}_d &= \frac{\zeta_d}{\sqrt{1+K_PK}}. && (4.169)
\end{aligned}$$

Proportional feedback on a simple resonance produces another simple resonance in closed-loop, with higher resonant frequency and lower damping. This is a case when simply turning the $K_P$ knob can cause problems.

### PI Control

Using proportional plus integral (PI) control we get

$$T(s) = \frac{PC}{1+PC} \tag{4.170}$$

$$= \frac{\frac{K_P K \omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\left(s+\frac{K_{IT}}{K_P}\right)}{1+\frac{K_P K \omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\left(s+\frac{K_{IT}}{K_P}\right)} \tag{4.171}$$

$$= \frac{K_P K\left(s+\frac{K_{IT}}{K_P}\right)}{s(s^2+2\zeta_d\omega_d s+\omega_d^2)+K_P\left(s+\frac{K_{IT}}{K_P}\right)K\omega_d^2} \tag{4.172}$$

$$= \frac{K_P K\left(s+\frac{K_{IT}}{K_P}\right)}{s^3+2\zeta_d\omega_d s^2+\omega_d^2(1+K_P K)s+K_{IT}K\omega_d^2} \tag{4.173}$$

This closed-loop has a pole-zero excess of $2$ which is generally a danger sign in such a low order system. Again, the simplest check here is to use Routh-Hurwitz. We need all the coefficients to be present and of the same sign (in this case $> 0$). If we wish to use Equation 4.157, we have $a_1 = 2\zeta_d\omega_d$, so as long as we have non-zero damping in the original system and an actual resonant frequency, then we can divide this out and go to Equation 4.158.

$$a_1 a_2 > a_3. \tag{4.174}$$

This becomes

$$2\zeta_d\omega_d\omega_d^2(1+K_P K) > K_{IT}K\omega_d^2, \tag{4.175}$$

which can be reduced to:

$$2\zeta_d\omega_d(1+K_P K) > K_{IT}K. \tag{4.176}$$

Now, generally, this means that the amount of integral gain, $K_{IT}$, allowed is limited by the damping, resonant frequency, and proportional gain. Once again, we have to limit integral gain relative to other factors to not destabilize the closed-loop system. In fact, this case seems to show up a lot in practice. Engineers apply PID controllers to mechatronic systems with simple resonances in them, and generally set $K_{DT} = 0$. They they realize that while they want to use nonzero $K_{IT}$ so as to have $0$ steady state error to a step, they must limit the value of $K_{IT}$ or the system starts ringing and eventually becomes unstable. It is reassuring to see that we can predict what we see in practice from a little bit of algebra. In other words lag or integral control works so long as it is taken out well below the resonance. How far below depends upon the sharpness ($Q$ factor) of the resonance where $Q = \frac{1}{2\zeta_D}$.

## PD Control

Using proportional plus derivative (PD) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ is $0$ we are back to the case in Section 4.11.4.

$$T(s) = \frac{PC}{1+PC} \tag{4.177}$$

$$= \frac{\frac{K_{DT}K\omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\left(s+\frac{K_P}{K_{DT}}\right)}{1+\frac{K_{DT}K\omega_d^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.178}$$

$$= \frac{K_{DT}K\omega_d^2\left(s+\frac{K_P}{K_{DT}}\right)}{s^2+2\zeta_d\omega_d s+\omega_d^2+K_{DT}K\omega_d^2\left(s+\frac{K_P}{K_{DT}}\right)} \tag{4.179}$$

$$T(s) = \frac{K_{DT}K\omega_d^2\left(s+\frac{K_P}{K_{DT}}\right)}{s^2+(2\zeta_d\omega_d+K_{DT}K\omega_d^2)s+K_PK\omega_d^2} \tag{4.180}$$

The closed-loop zero is at $s = -\frac{K_P}{K_{DT}}$. As $K_P$ increases relative to $K_{DT}$ the derivative action that usually adds stability shows up at higher frequency. For the poles we use the quadratic formula on the denominator:

$$z_{1,2} = \frac{-(2\zeta_d\omega_d+K_{DT}K\omega_d^2) \pm \sqrt{(2\zeta_d\omega_d+K_{DT}K\omega_d^2)^2-4K_PK\omega_d^2}}{2} \tag{4.181}$$

With $K$, $K_{DT}$, and $K_P$ all greater than $0$ this is always stable, as one would expect from applying a lead to a simple stable resonance. (In this case PD control can be considered a lead circuit with no flattening in the controller model Bode plot.) The poles are real so long as

$$(2\zeta_d\omega_d + K_{DT}K\omega_d^2)^2 \geq 4K_PK\omega_d^2, \tag{4.182}$$

which can be reduced to

$$(2\zeta_d + K_{DT}K\omega_d)^2 \geq 4K_PK. \tag{4.183}$$

Again, too much $K_P$ relative to $K_{DT}$, $K$, and $\omega_d)^2$ can cause stable, but ringing behavior in the closed-loop response. In frequency, if $K_P$ is high relative to $K_{DT}$, the compensator lead is at too high a frequency to damp ringing.

### PID Control

Using proportional plus integral plus derivative (PID) control we don't lose much generality to assume that $K$, $K_P$, $K_{IT}$, and $K_{DT}$ are all greater than $0$. If $K_{DT}$ or $K_{IT}$ are $0$ we are back to one of our prior cases.

$$\begin{align}
T(s) &= \frac{PC}{1+PC} \tag{4.184}\\
&= \frac{\frac{K_{DT}K\omega_d^2}{s\left(s^2+2\zeta_d\omega_d s+\omega_d^2\right)}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{1+\frac{K_{DT}K\omega_d^2}{s\left(s^2+2\zeta_d\omega_d s+\omega_d^2\right)}\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)} \tag{4.185}\\
&= \frac{K_{DT}K\omega_d^2\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s\left(s^2+2\zeta_d\omega_d s+\omega_d^2\right)+K_{DT}Ks^2+KK_Ps+KK_{IT}} \tag{4.186}\\
&= \frac{K_{DT}K\omega_d^2\left(s^2+\frac{K_P}{K_{DT}}s+\frac{K_{IT}}{K_{DT}}\right)}{s^3+(2\zeta_d\omega_d+K_{DT}K)s^2+(\omega_d^2++KK_P)s+KK_{IT}} \tag{4.187}\\
&= \frac{K_{DT}K\omega_d^2\left(s^2+2\zeta_n\omega_n s+\omega_n^2\right)}{s\left(s^2+2\zeta_d\omega_d s+\omega_d^2\right)+K_{DT}K\omega_d^2\left(s^2+2\zeta_n\omega_n s+\omega_n^2\right)} \tag{4.188}\\
T(s) &= \frac{K_{DT}K\omega_d^2\left(s^2+2\zeta_n\omega_n s+\omega_n^2\right)}{s^3+(2\zeta_d\omega_d+K_{DT}K\omega_d^2)s^2+(\omega_d^2+K_{DT}K\omega_d^2 2\zeta_n\omega_n)s+K_{DT}K\omega_d^2\omega_n^2} \tag{4.189}
\end{align}$$

In Equation 4.188 we have made the following substitutions:

$$\omega_n^2 = \frac{K_{IT}}{K_{DT}} \text{ and } 2\zeta_n\omega_n = \frac{K_P}{K_{DT}} \tag{4.190}$$

Equation 4.185 becomes:

$$T(s) \quad = \quad \frac{\frac{K_{DT}K\omega_d^2}{s}\left(\frac{s^2+2\zeta_n\omega_n s+\omega_n^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\right)}{1+\frac{K_{DT}K\omega_d^2}{s}\left(\frac{s^2+2\zeta_n\omega_n s+\omega_n^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\right)} \tag{4.191}$$

One very simplistic view of this is that if we perfectly match the plant denominator response with the controller numerator response, i.e. if we explicitly set

$$\omega_n^2 = \omega_d^2 \text{ and } \zeta_n = \zeta_d, \tag{4.192}$$

then Equation 4.191 reduces to:

$$T(s) \quad = \quad \frac{K_{DT}K\omega_d^2}{s+K_{DT}K\omega_d^2}. \tag{4.193}$$

This is a sweet result. It tells us that if we could perfectly identify the plant resonance, we can use our PID to cancel it out, leaving the closed-loop behavior to be that of a first order low pass filter with unity DC gain and a bandwidth set by adjusting $K_{DT}$.

More generally, if we do not have that perfect cancellation condition, we can check stability from applying the Routh-Hurwitz Criterion to the denominator of Equation 4.189. Repeating Equation 4.156 here

$$P(s) = s^3 + a_1 s^2 + a_2 s + a_3, \tag{4.194}$$

then the Routh Array requires that

$$\frac{a_1 a_2 - a_3}{a_1} > 0. \tag{4.195}$$

From Equation 4.189,

$$\begin{aligned}
a_1 &= 2\zeta_d\omega_d + K_{DT}K\omega_d^2, && (4.196)\\
&= \omega_d(2\zeta_d + K_{DT}K\omega_d), && (4.197)\\
a_2 &= \omega_d^2 + K_{DT}K\omega_d^2 2\zeta_n\omega_n, && (4.198)\\
&= \omega_d^2(1 + K_{DT}K2\zeta_n\omega_n), \text{ and} && (4.199)\\
a_3 &= K_{DT}K\omega_d^2\omega_n^2. && (4.200)
\end{aligned}$$

We need each of these to be $> 0$, and given that $K_{DT} > 0$, we guarantee that $a_1 > 0$ so that we can use Equation 4.158,

$$a_1 a_2 > a_3. \tag{4.201}$$

This results in

$$\begin{aligned}&(2\zeta_d\omega_d + K_{DT}K\omega_d^2)\omega_d^2(1 + K_{DT}K\omega_d^2 2\zeta_n\omega_n)\\ &\quad > K_{DT}K\omega_d^2\omega_n^2,\end{aligned} \tag{4.202}$$

which can be reduced to

$$(2\zeta_d\omega_d + K_{DT}K\omega_d^2)(1 + K_{DT}K2\zeta_n\omega_n) > K_{DT}K\omega_n^2. \tag{4.203}$$

Now, this seems interesting, but what's a few more pages of algebra between friends?

$$\begin{aligned}&(2\zeta_d\omega_d + K_{DT}K\omega_d^2)K_{DT}K2\zeta_n\omega_n +\\ &\quad (2\zeta_d\omega_d + K_{DT}K\omega_d^2) - K_{DT}K\omega_n^2 > 0,\end{aligned} \tag{4.204}$$

or

$$\begin{aligned}&K_{DT}K(\omega_d^2 - \omega_n^2) + 2\zeta_d\omega_d(1 + K_{DT}K\omega_d^2)K_{DT}K2\zeta_n\omega_n\\ &\quad + (K_{DT}K)^2\omega_d^2)2\zeta_n\omega_n > 0\end{aligned} \tag{4.205}$$

Now, the only way that the left hand side of Equation 4.205 can be $\leq 0$ is if $\omega_n > \omega_d$ and in fact, because of all the other terms, it really has to be significantly higher, $\omega_n \gg \omega_d$. If $\omega_d \geq \omega_n$ this is always stable.

One more special case that is very useful should be discussed. Let's say that we set $\omega_n = \omega_d$ and $\zeta_n \geq \zeta_d$. Say $\zeta_n = \zeta_d + \zeta_r$ where the $r$ subscript indicates something like "robustify". The idea behind this is that we might think that we can match the frequency of the resonance perfectly, but we want a bit of width to our "notch" in case we miss slightly. Equation 4.188 becomes:

$$\begin{aligned}T(s) &= \frac{K_{DT}K\omega_n^2\left(s^2 + 2\zeta_n\omega_n s + \omega_n^2\right)}{s\left(s^2 + 2(\zeta_n - \zeta_r)\omega_d s + \omega_n^2\right) + K_{DT}K\omega_n^2\left(s^2 + 2\zeta_n\omega_n s + \omega_n^2\right)} \qquad (4.206)\\ &= \frac{K_{DT}K\omega_n^2\left(s^2 + 2\zeta_n\omega_n s + \omega_n^2\right)}{s\left(s^2 + 2\zeta_n\omega_n s + \omega_n^2\right) + K_{DT}K\omega_n^2\left(s^2 + 2\zeta_n\omega_n s + \omega_n^2\right) - s^2\zeta_r\omega_n} \qquad (4.207)\\ &= \frac{K_{DT}K\omega_n^2}{s + K_{DT}K\omega_n^2 - \frac{s^2\zeta_r\omega_n}{s^2+2\zeta_n\omega_n s+\omega_n^2}} \qquad (4.208)\\ & \qquad (4.209)\end{aligned}$$

Now, the result in Equation 4.208 shows that we have a first order closed loop, plus some parasitic nonsense in the denominator. When $s$ is small, it has little effect, but as $s \longrightarrow \infty$ we get a term that goes to $-\zeta_r\omega_n$, which boosts the gain slightly by lowering the corner frequency a bit. At $s = -j\omega_n$ the extra denominator term becomes:

$$-\left.\frac{s^2\zeta_r\omega_n}{s^2 + 2\zeta_n\omega_n s + \omega_n^2}\right|_{s=j\omega_n} = \frac{\zeta_r\omega_n^3}{2j\zeta_n\omega_n^2} = \frac{-j\omega_n}{2} \tag{4.210}$$

Alternately, we can look at the term:

$$\left.\frac{s^2 + 2(\zeta_d + \zeta_r)\omega_d s + \omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}\right|_{s=j\omega_d} \tag{4.211}$$

$$= \frac{2(\zeta_d + \zeta_r)j\omega_d^2}{2\zeta_d j\omega_d^2} \tag{4.212}$$

$$= \frac{(\zeta_d + \zeta_r)}{\zeta_d} \tag{4.213}$$

$$= 1 + \frac{\zeta_r}{\zeta_d} \tag{4.214}$$

which means that Equation 4.191 becomes (in the region of $\omega_d$)

$$T(s) \approx \frac{K_{DT}K\omega_d^2\left(1 + \frac{\zeta_r}{\zeta_d}\right)}{s + K_{DT}K\omega_d^2\left(1 + \frac{\zeta_r}{\zeta_d}\right)} \tag{4.215}$$

## 4.12 General Thoughts on Closed-Loop Analysis for Second Order Models

Generally speaking, we have seen that a pole-zero excess in the closed loop model of $1$ is ideal, and usually implies stability when using positive PID coefficients.

A pole-zero excess of $0$ means that we must rely on the unmodeled low pass nature of the physical plant to achieve roll off of the closed loop response.

A pole-zero excess of two or more in the low-order closed loop is a danger sign, and usually means possible instability, depending upon the parameters.

As things become more complicated, we must rely on some knowledge of the frequency spacing in order to have intuition.

- $K_P$ (proportional action) applies across all frequencies and shifts the open loop gain without affecting the open loop phase.
- $K_I$ (integral action) is best restricted to low frequencies with low being defined by the physical system. The more one wants to push $K_I$ the better the system model has to be.
- $K_D$ (derivative action) is best used at high frequencies (but not too high) with high being defined by the physical system. The more one wants to push $K_D$ the better the system model has to be and the less noise and phase due to sampling one can allow. $K_D$ working with $K_P$ can be considered lead action and is normally applied around open loop crossover. $K_D$ offers improved stability but too much of it amplifies noise in areas beyond the closed-loop bandwidth.

## 4.13 Intuitive and Manual Tuning

When your system is of sufficiently low order so as to be well modeled by one of the models in Section 2.3, then a fairly simple PID tuning method can be used that is essentially a systematic way of turning the PID coefficient knobs. There are a variety of such schemes but probably the best well-known are the Ziegler-Nichols tuning rules, named for its inventors in the 1940's [126, 127]. These begin with the basic assumption that your plant is well-behaved with an (open-loop) response that is similar the the process curve shown in Fig. 4.13. Such models can be well-approximated by a simple first order low-pass filter with lag, as in Equation 2.5, and it was with this model in mind the Ziegler and Nichols designed their rules[1].

The Ziegler-Nichols tuning rules use a form of the PID controller involving explicit time specification and without differentiator filtering, namely

$$C(s) = K_P \left( 1 + \frac{1}{T_I s} + T_D s \right). \tag{4.216}$$

While slightly different than Equations 4.35–4.38, the gains are easily translated between the forms. There are actually two different Ziegler-Nichols tuning rules, each relying on simple physical measurements of the plant. This was especially important when digital computers did not exist and thus

[1] This section was co-authored with Sean Andersson[12].

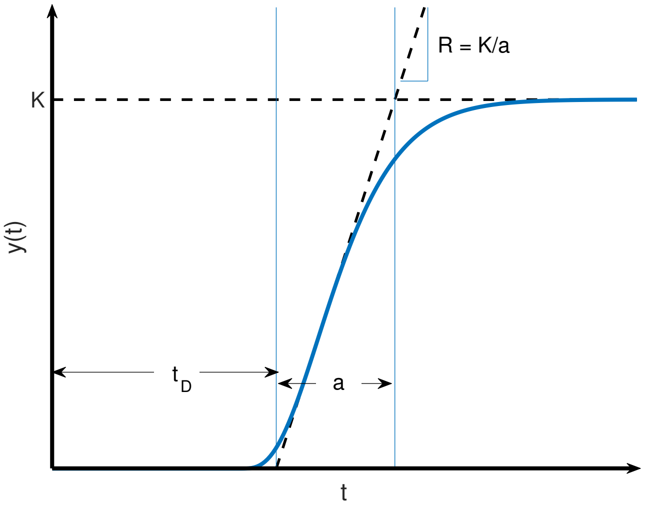


Figure 4.13: Process reaction curve used by Ziegler and Nichols.

data processing methods of tuning were not available. Due to their simplicity, the rules remain useful even today.

The first set of rules uses measurements from the open-loop step response of Figure 4.13. The goal is to produce a closed-loop response with a transient that decays by one-quarter of its value in one period of oscillation, corresponding to a damping of $\zeta = 0.21$. While perhaps a bit on the low side, Ziegler and Nichols felt this was a good trade-off between response, overshoot, and disturbance robustness. The resulting set of tuning values are given in Table 4.2. The parameters $R$ (reaction rate), $t_D$ (delay time), and $a$ (filter pole) that define the gains are all measured from the open loop step response.

The second set of rules uses the closed loop system under a simple proportional controller. To determine the desired controller parameters, first the proportional gain is increased until the output exhibits sustained oscillations; this is called the ultimate gain, $K_u$ and the period of that oscillation the ultimate period, $P_u$. The controller parameters designed by the ultimate sensitivity method of Ziegler and Nichols are then calculated from these measurements according to the formulas in Table 4.2. Generally, this method suggests gains that are smaller than those from the step response approach.

In many cases, the Ziegler-Nichols rules provide an acceptable closed-loop response, at least as a starting point for further tuning. Except when they don't. Depending on your plant, measuring the required parameters may not be practical. For example, if your process is very slow then ramping up

|  | Step response |  |  | Ultimate sensitivity |  |  |
|---|---|---|---|---|---|---|
|  | $K_P$ | $T_I$ | $T_D$ | $K_P$ | $T_I$ | $T_D$ |
| P | $\frac{1}{Rt_D}$ | - | - | $0.5K_u$ | - | - |
| PI | $\frac{0.9}{Rt_D}$ | $\frac{t_D}{0.3}$ | - | $0.45K_u$ | $\frac{P_u}{1.2}$ | - |
| PID | $\frac{1.2}{Rt_D}$ | $2t_D$ | $0.5t_D$ | $1.6K_u$ | $0.5P_u$ | $0.125P_u$ |

Table 4.2: Ziegler-Nichols tuning rules

the proportional gain until you see marginal stability may take inordinately long. And, of course, if your system is not well-modeled by that first-order low pass filter with delay, then the rules may not work well at all. Still, their simplicity and utility to many plants of interest make the Ziegler-Nichols tuning rules an important tool kit for any practicing engineer.

As an example, consider the system responses shown in Fig. 4.14. No process model is given because the point of using these tuning rules is that no actual model is needed. Shown instead are the open loop step response (the left plot) and the closed loop response under pure proportional control (the right plot). From the first plot we estimate that the delay is approximately 2 seconds while the process rate is approximately 0.5 units/sec. From the second plot we measure an ultimate period of approximately 7 seconds, found when the proportional gain was 1.56. Choosing to apply a PI controller, we find the Ziegler-Nichols gains to be

$$K_P = 0.919, \quad T_i = 6.67 \tag{4.217}$$

when using the step response rules and

$$K_P = 0.702, \quad T_i = 5.83 \tag{4.218}$$

when using the ultimate gain method. The corresponding step responses are shown in Figure 4.15. Note that the ultimate gain method yields a lower overshoot but is slightly more sluggish.

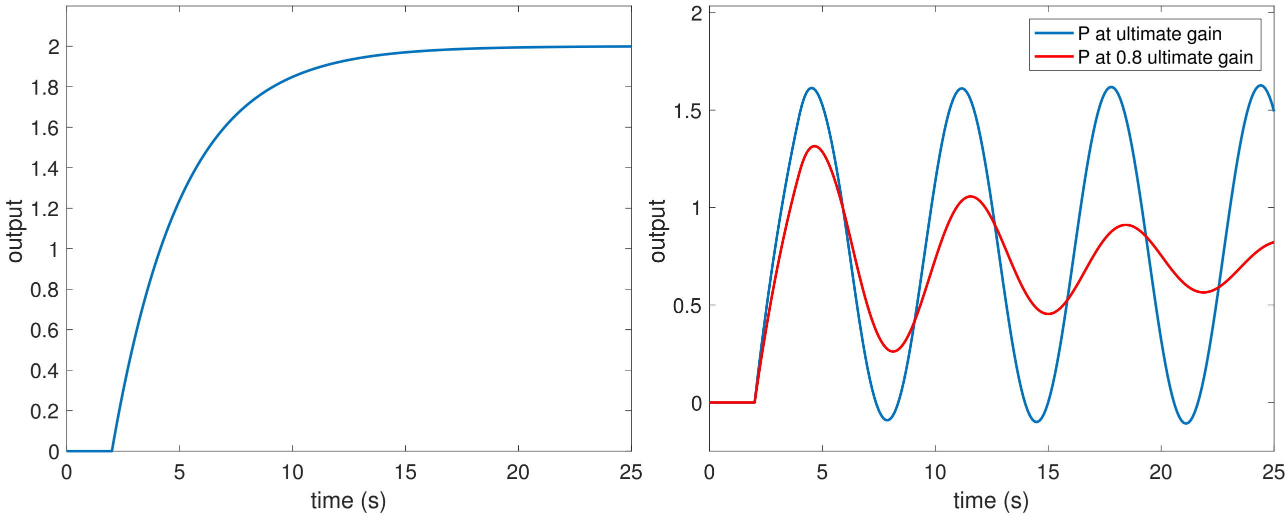


Figure 4.14: Measuring the parameters for the Ziegler-Nichols tuning rules. (left) Open response of the system showing a delay of approximately two seconds and a process rate of approximately 0.5 units/sec. (right) Closed-loop response under pure P control at both the ultimate gain (of $K_u = 1.56$) and at 80% of the ultimate for reference. The ultimate period is approximately 7 seconds.

## 4.14 Relay Tuning of PID Controllers

Relay tuning of PID controllers is a method proposed by Åström et. al. [115, 118] for low order plants which are not resonant. The basic idea is to replace the PID controller in the loop with a relay action, which will prompt a limit cycle in the closed-loop behavior that allows the designer to characterize a few of the plant parameters. From this the PID parameters can be picked.

**Will try to add more when possible.**

## 4.15 Loop Shaping

While Ziegler-Nichols is very useful at getting a basic, low performance controller working, it really has no hope of generating excellent closed-loop response in the absence of a more accurate system model. Furthermore, it can have serious issues when the system has high Q dynamics close to the system bandwidth. When one has the luxury or the need of a frequency response function (FRF)

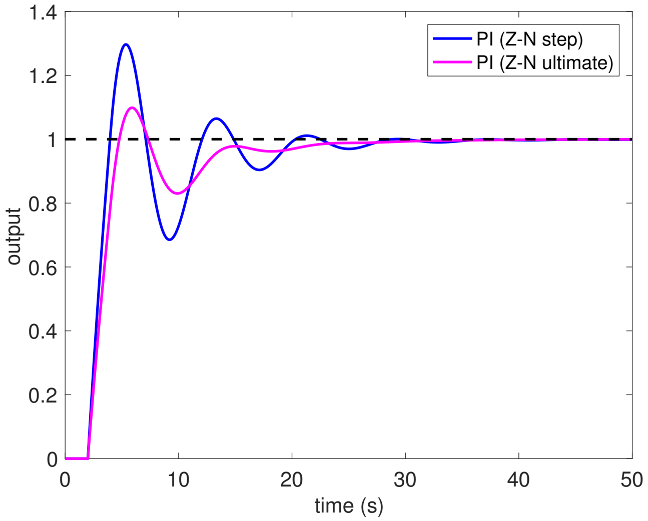


Figure 4.15: Step response of the system in Figure 4.14 under the two different tuning rules.

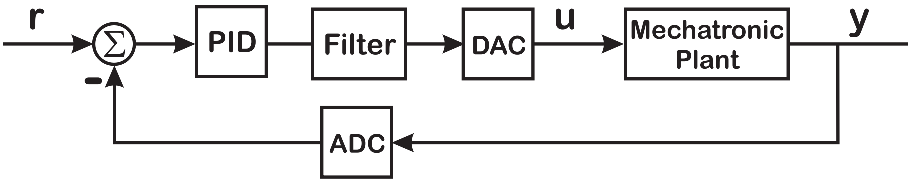


Figure 4.16: **A mechatronic system often has more resonances and anti-resonances than can be handled by a PID alone. In that case, a PID plus some filtering is often used.**

measurement, then it is reasonable to use the filter form of the PID to generate a PID that adjusts the open-loop shape so as to yield a desirable closed-loop response.

When the high frequency behavior of the physical system is prominent, as it might be in a mechatronic system with rigid body behavior and high frequency resonances and anti-resonances, then simple PID tuning rules must give way to loop shaping and extra filters. This requires a lot of pieces. In the examples that follow, it will become obvious that the higher performance uses of PID for loop shaping require accurate models, and this is almost always tied to identification of the physical system.

The companion tutorial for this one, [11], describes practical methods for making system measurements for control, focusing on step response and frequency response function (FRF) methods. The one by Mike Borrello [98] is more specific on the use of dynamic signal analyzers (DSAs). Generally speaking, step response methods are associated with Ziegler-Nichols tuning or conservative PI control, while loop shaping points the use of FRF methods. The first author has discussed self tuning PIDs for atomic force microscopes in Section 4.17.3 and in [108], as well as a more recent discussion in [40, 19]. It also requires possible use of complex responses for PIDs, as discussed there. Some related material involves the use of second order complex phase leads, which may or may not be implemented as PIDs [123, 121]. It is useful to note that while the discussions in [40, 108, 19] discuss discrete implementations, the discussions in [123, 121] are restricted to continuous time.

## 4.16 Examples of PID Code

It is useful to have some example code to understand how one might implement a PID controller on a processor.

Table 4.3 shows a simple example of PID code. Note the simplicity of this code segment, although a few things need to be pointed out:

- This formulation is using a backwards rule equivalent of an unfiltered analog PID with explicit time and $T = T_I = T_D$. This means that the $K_P$, $K_I$, and $K_D$ are the same analog design parameters.
- We have not added in any integrator anti-windup to this simple code example, nor have we added in any hooks for direct feedforward from the reference input to the PID output.
- We have not added in any presaturation that limits the output of the PID to some known limit range.
- We haven't shown the top of the segment, which would have to take into account how to have the prior values in the subroutine. Likewise, we haven't discussed where the gains get passed in. This is important because for most PIDs (as well as most filter subroutines) the gains don't change every iteration, if at all during the operation of the system. Remembering that most functions/subroutines by default blank any data not passed in on the parameter list, we have a few options for keeping around old values from previous steps.
  - Use global variables. This is a very old programming practice, which is usually discouraged. Global variables are space efficient and fast, but since any routine can access them

(not just the routines who are supposed to), they can cause a lot of massive bugs, and are best used sparingly.

- Retain the history of variables that need to be persistent outside of the routine, passing them to the routine in the parameter list. The efficiency of such a choice depends largely on the programming language. In a language where one passes data to and from routines by value (e.g. in MATLAB functions) then this potentially means passing a lot of data back and forth. In one where one can pass by reference (e.g. in C/C++/C#, etc.) then only a reference is passed, but since these are scalar quantities, there isn't too much savings when passing a reference to a float compared to passing the float. This changes when we are passing a pointer to a structure or class, as opposed to a scalar value.
- Use persistent/static variables in the routine. These variables keep their values between calls. Doing this usually requires some sort of initialization flag, so that we know the first time we are entering the routine, to initialize the variables. This can work when there is only one instance of a particular routine, say when there is only one PID control block. It becomes a mess when there are multiple loops trying to use the same piece of code, as each would somehow have to keep track of their own persistent variables. This leads to the logical conclusion of . . .
- Use object oriented programming (OOP), because – to paraphrase Neil Diamond – that's what it's there for. We do this by creating a PID class, which has its own member data (a.k.a. persistent data) and member methods (a.k.a. routines). Each instance of the class has its own persistent data space, so we can store and access parameters, old values of data, etc. This requires more memory and can be a tiny bit slower than one of the raw methods above, but memory is a lot cheaper than it was when C was first developed and modern compilers mean that the extra abstractions of C++ cost very little time.

```
//--------------------------------------------------------------------
// One step of a PID controller.
// We assume here that we have three gains, K_P, K_I, and
// K_D, plus an overall gain, K.
// While this is an extra term, it allows us to shape the
// PID with the first 3 terms, and then scale the overall gain
// up and down.
//--------------------------------------------------------------------
// Okay, now we save the previous values.
//--------------------------------------------------------------------
e_km1 = e_k;                    // Save the previous error
i_km1 = i_k;                    // Save the previous integral term
//--------------------------------------------------------------------
// Compute current integral and differential terms.
//--------------------------------------------------------------------
i_k = i_km1 + e_k;              // Integrator
d_k = e_k - e_km1;              // Differentiator
//--------------------------------------------------------------------
// Compute individual control branches.
//--------------------------------------------------------------------
u_kp = K_p*e_k;                 // Proportional term
u_ki = K_i*i_k;                 // Integral term
u_kd = K_d*d_k;                 // Differential term
//--------------------------------------------------------------------
// Final output has an overall gain attached.
//--------------------------------------------------------------------
u_k  = K*(u_kp + u_ki + u_kd);  // Overall control output.
```

Table 4.3: Basic PID Code Snippet in C++

# 4.17 Examples of PID Control

## 4.17.1 PID Controller Response Shapes

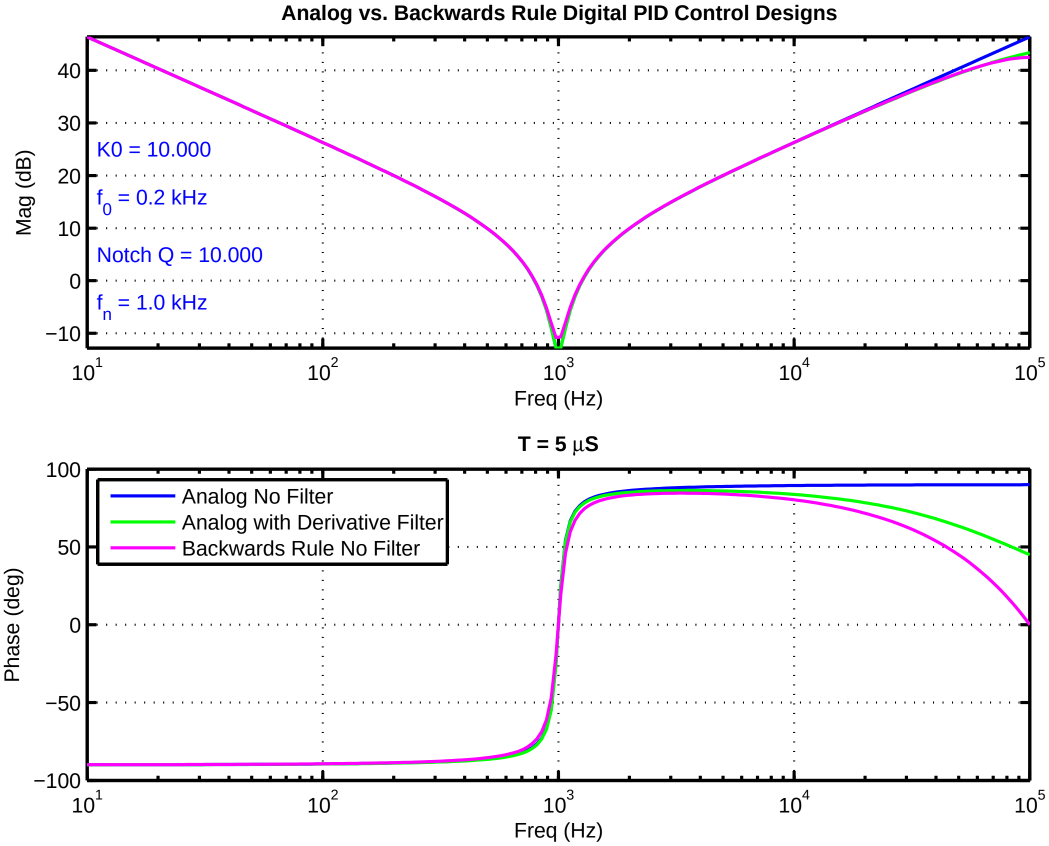


Figure 4.17: **Continuous and discrete PID controller response derived from notch filter model. The parameters of the notch filter model are $K_0 = 10$ at $f_0 = 200$ Hz, $f_n = 1$ kHz, and $Q = \frac{1}{2\zeta} = 10$. The integration time, $T_I$, differentiation time, $T_D$, and eventual sampling time $T_S$ are all set equal to $T = 5\mu S$. In the case of derivative filtering, the low pass frequency is chosen as $50$ kHz. The backwards rule version is based on the unfiltered continuous design.**

This section shows how the one can use filter parameters to design a PID controller, and what the frequency response functions of different versions of this controller looks like. The parameters of the notch filter model are $f_n = 1$ kHz and $Q = \frac{1}{2\zeta} = 10$. The parameters $K_0 = 10$ at $f_0 = 200$ Hz, mean that the parameters are adjusted so that the filter has a gain of $10$ at $200$ Hz. The integration time, $T_I$, differentiation time, $T_D$, and eventual sampling time $T_S$ are all set equal to $T = 5\mu S$. In the case of design based upon a filtered differentiator, the corner frequency is chosen to be at $50$ kHz.

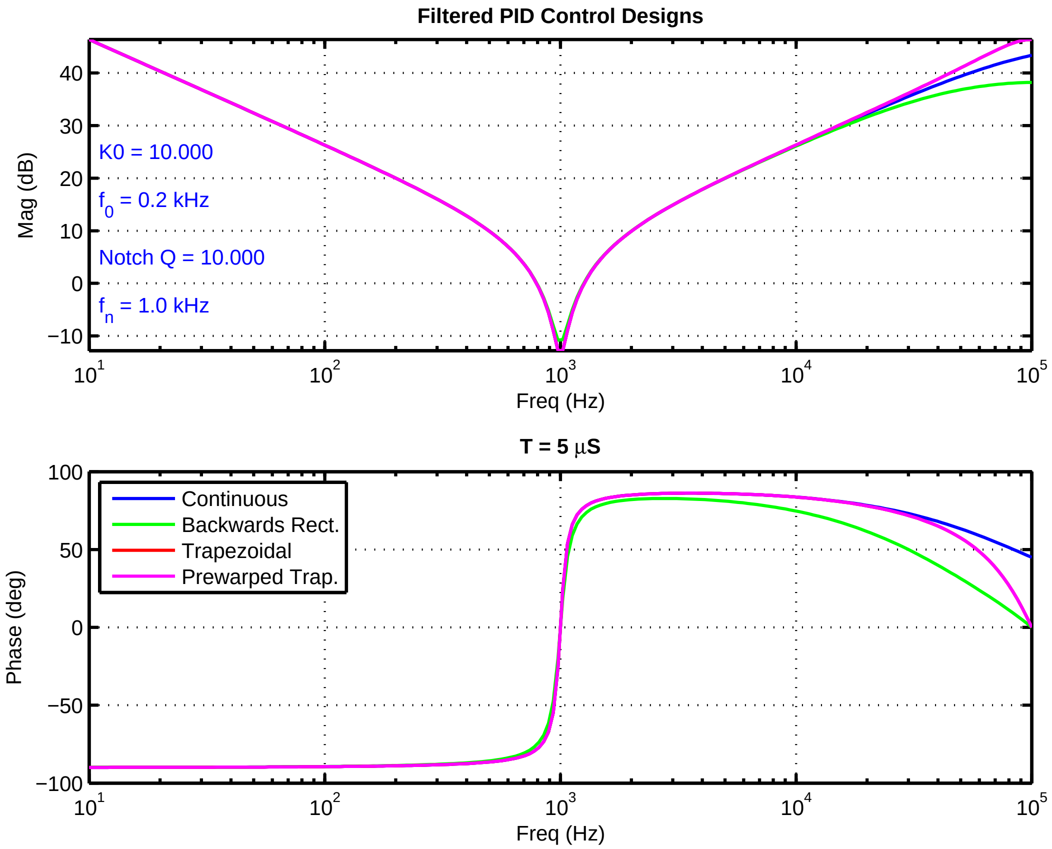


Figure 4.18: **PID controller response derived from notch filter model. This version has a low pass filter on the derivative. Note that the prewarped trapezoidal and the trapezoidal rule are virtually identical, as the prewarp frequency is at $1$ kHz, far below the $200$ kHz sample frequency.**

We can see from Figure 4.17 that the net effect of the backwards rule is to place a low pass filter on the differentiator portion. (The pole at $z = 0$.) The main issue is that we do not have design freedom with choice of this filter, but for many the simplicity of a simple translation between analog PID parameters and digital PID parameters is worth this. The comparison in Figure 4.18 shows that with the trapezoidal rule equivalent of a filtered analog design, we can minimize the phase penalty. This is most useful when dynamics at higher frequency require equalization, such as with a multinotch [54] and we do not want to be limited by our PID implementation.

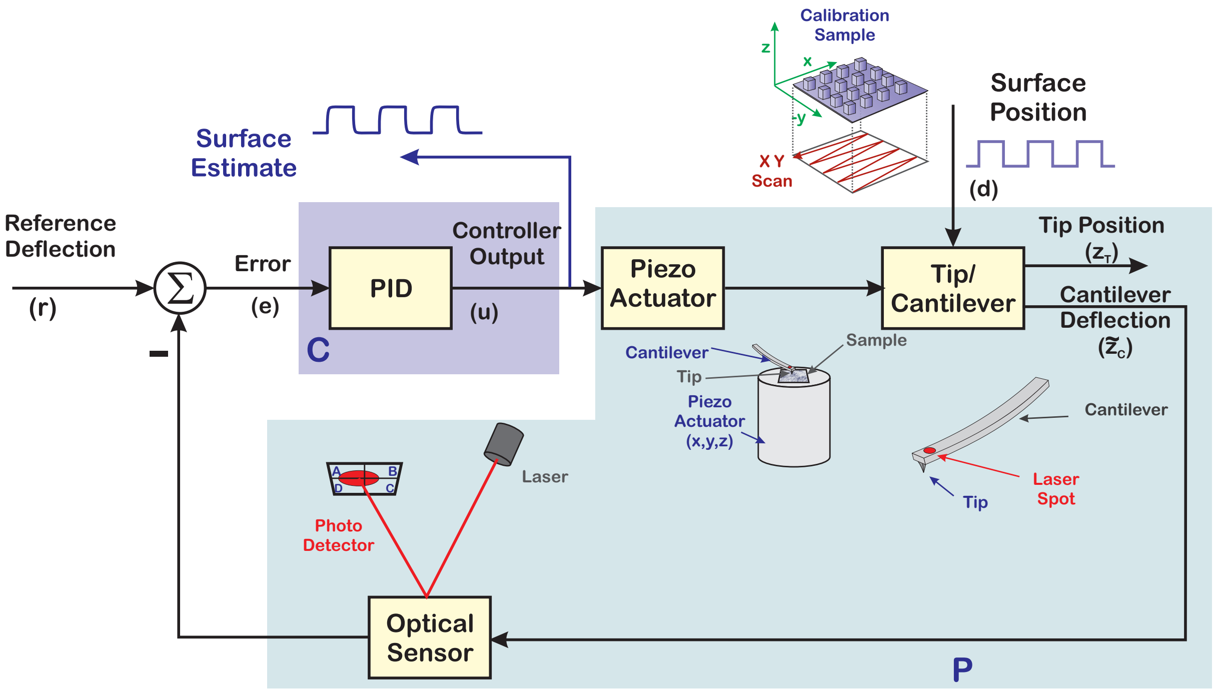


Figure 4.19: An AFM Control Block Diagram. The diagram shows a scanned sample design, where the tip and cantilever are fixed and the sample is moved under the tip by the piezo actuator. In this mode, the controller attempts to maintain a constant level of deflection which corresponds to a constant level of contact force. The quantity to be measured, the surface profile, comes in as an unknown disturbance to the control loop. The deflection of the cantilever is sensed with optical detection.

### 4.17.2 Atomic Force Microscopes and PI Control

The following example is largely taken from a tutorial on the control of atomic force microscopes (AFMs) [27]. An example set of frequency response curves for the $Z$ axis of an idealized piezo tube actuator is shown in Figure 4.21. The piezo tube resonances shown here are around 1 kHz, which is in the typical range of 500 Hz to 20 kHz. Some experimental systems have resonances above 40 kHz [129, 130].

In Figure 4.21 a series of five models of the piezo-cantilever system are plotted with the resonant frequency varying between 900 Hz and 1.1 kHz, and the quality ($Q$) factors varying between 10 and 30. The uncertainty results both from variation across multiple actuators and variation of the same actuator with varying signal amplitudes and environmental conditions. At higher frequency one sees a nominal 300 kHz cantilever resonance with a nominal $Q$ of 100. Again, there is variation across different cantilevers in the same batch and different conditions for the same cantilever. All of this

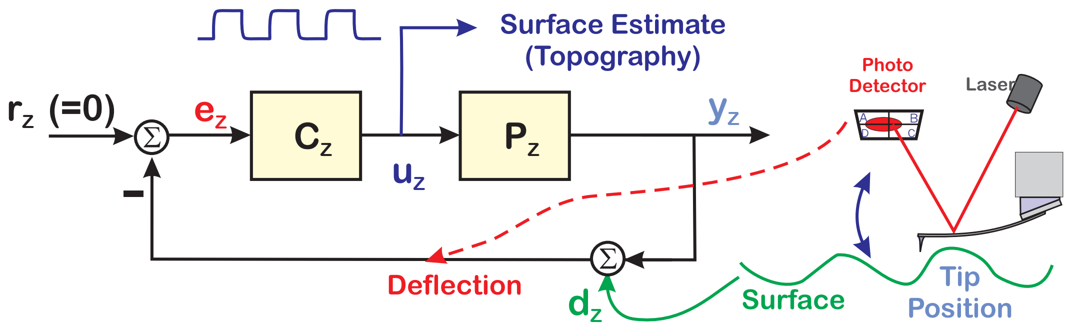


Figure 4.20: Most commercial AFM controllers focus on low frequency control of the $z$ axis, and the controller output is considered a scale factor away from the topography signal.

illustrates the need to do identification, whether as a preliminary calibration step or in an on-line form.

Note that these plots are idealized in that they neglect any extra dynamics – including non-minimum phase zeros – typically present in the actuator and cantilever. Furthermore, any dynamics of the electrical circuitry, such as low pass effects of the power amplifiers used to drive the piezos are neglected. Finally, these plots show no effects of transport or computational delay. However, even when using such an idealized model, the significant limitations and issues with AFM control are evident.

The effects of this structure on the feedback system can be immediately seen. If a feedback controller is to include a 300 kHz resonance in the model, then a typical rule of thumb sample rate of 10-20 times the highest frequency of the dynamics of interest would imply a $3 - 6$ MHz sample rate for the control system. Obviously, such a high sample rate puts severe constraints on the signal processing system, not just in accomplishing the needed processing between samples, but also in minimizing the latency of the computations, signal conditioning, and data conversion.

On the other end of the spectrum are control systems that will restrict bandwidth to be safely below the Z-piezo actuator's resonance. For a 1 kHz resonance, this implies a sample rate of no less than 10 kHz. Thus, a typical sample rate for control on an industrial AFM is in the 50–100 kHz range [131], although newer controllers sample considerably faster – up to 500 kHz in the case of [132].

Because the piezo actuator is modeled as a second-order resonance, the lack of integrators in the forward path necessitates the use of integral action for zero steady-state error to any steps in the surface height. The addition of a second integrator via PII control can provide zero steady-state error to surface slopes, which are common in many samples. Such controllers are necessarily low

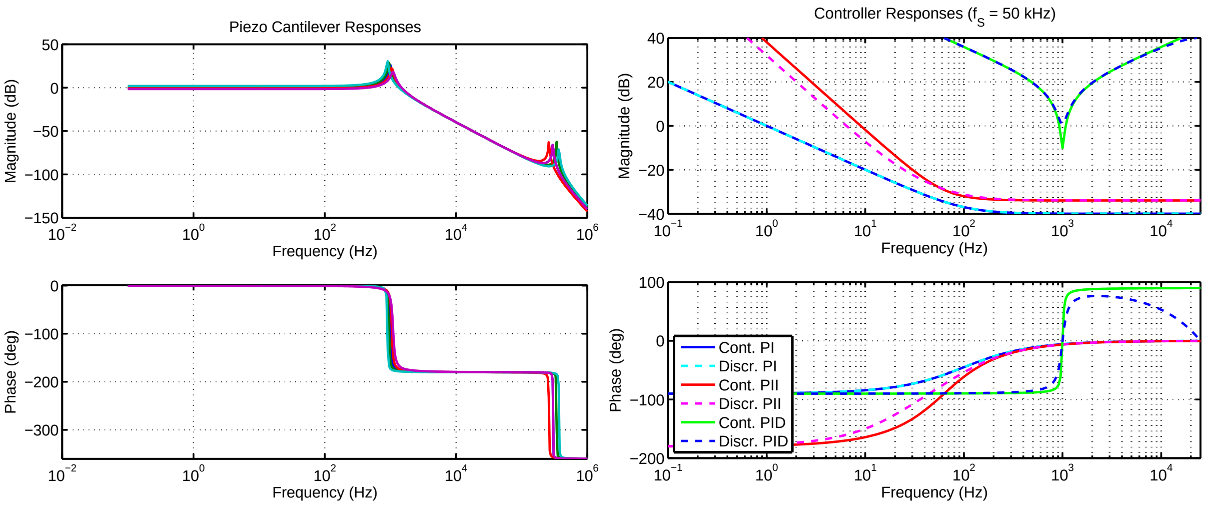


Figure 4.21: On the left is a set of “generic” AFM plants. This shows the combination of the Z-piezo actuator and a 300 kHz cantilever. Note that hysteresis, creep, and nonlinearity in the piezo [128] makes the exact modeling of a given actuator difficult, and thereby hampers the control. The cantilever properties also vary considerably within a batch. On the right are a collection of possible PI and PII controllers that might be applied to these plants. Note that without compensating for the high frequency resonance, we are limited on raising the bandwidth, but even doing that depends on having some phase lead (derivative control). The PID controller FRFs accomplish just that. Note however, that the high frequency boost may present issues with the cantilever resonance at 300 kHz.

bandwidth, since the lack of phase lead means that the gain must be rolled off below the resonance of the actuator.

A look at the configuration of Figure 4.19 shows that the fundamental feature of the $Z$ axis control loop is that the control system only sees the deflection (error), not the surface. As such it is an output error loop, without direct access to any reference signal. This limits any attempt at feedforward in the Z direction to methods that use some prior Z measurement (such as the previous scan line). It also limits the bandwidth of any state-space controller that one may use, since the estimator error can go away no faster than the error in the overall control loop [133]. Furthermore, as one sees from the typical example shown in Figure 4.21, there is considerable variation in the response of the actuator at low frequency and the cantilever at high frequency. This uncertainty means that either the control system has to be very robust or adaptive.

The typical industrial AFM control loop, whether done in contact or dynamic mode, is a low frequency

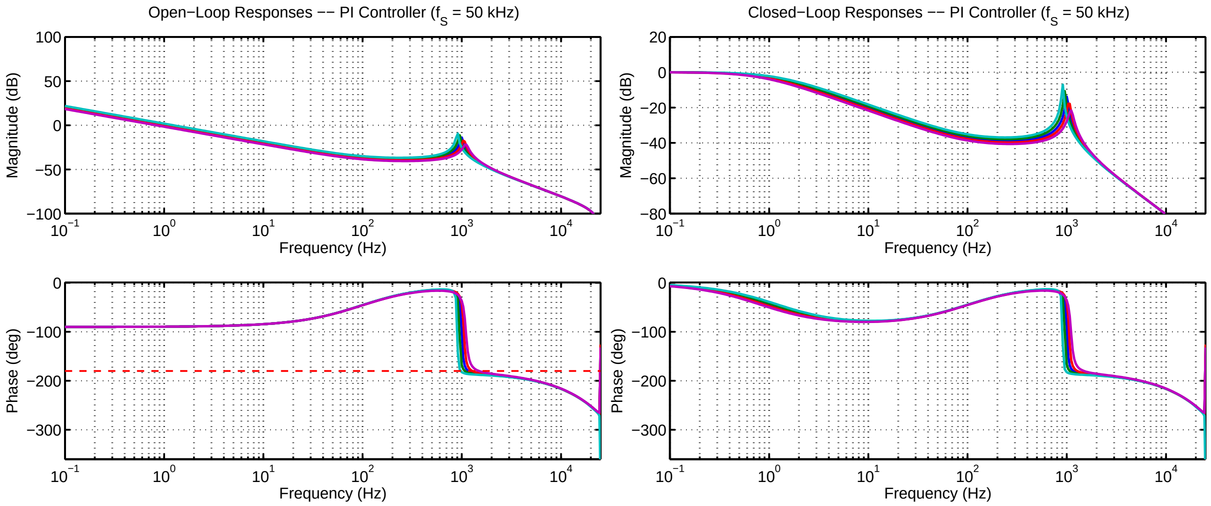


Figure 4.22: On the left: open-loop response for piezo/cantilevers of the left side of Figure 4.21 with digital PI controller from the right side of Figure 4.21. On the right: the closed-loop responses.

PI or PII loop. A general form of an analog controller that admits PI, PD, PID, PII, and even PIID is:

$$C(s) = \left(K_p + \frac{K_{IT}}{s} + \frac{K_{IIT}}{s^2} + K_{DT}s\right)E(s) \tag{4.219}$$

where $E(s)$ is the Laplace transform of the error signal $e(t)$. For a P, PI, PII, or PID controller, one or more of the $K_{DT}$, $K_{IT}$, or $K_{IIT}$ gains are set to zero. Note that as written the derivative term, $K_{DT}s$, is not practically implementable, but this is often rectified by having some low pass filter added to it. For digital implementation, the backward rectangular integration rule is most often used for PID controllers since this allows for direct translation from (4.219) [133, 134].

It is tempting to try to increase the bandwidth of the system by adding phase lead, such as with a PID controller. However, the use of this is limited by the uncertainty in the modeling of the piezo actuator. Furthermore, boosting the bandwidth with a PID requires lower noise in the optical measurement of deflection, otherwise this noise will be amplified by the effects of the derivative term.

For the models on the left of Figure 4.21, a pair of controllers (PI and PII ($K_{DT} = 0$) and PID ($K_{DT} > 0$) were synthesized as shown on the right side of Figure 4.21. The system was sampled at 50 kHz, and no attempt was made to add any extra computational or transport delay. Thus, the open-loop plots of the left side of Figures 4.22 and 4.23, representing the application of the PI and PII controller respectively, should be considered an idealized case. What is clear in these plots is that the open-loop crossover frequency must be substantially below the nominal resonant frequency for there to be any gain margin. Furthermore, the low frequency gain is quite limited in the case of the PI controller.

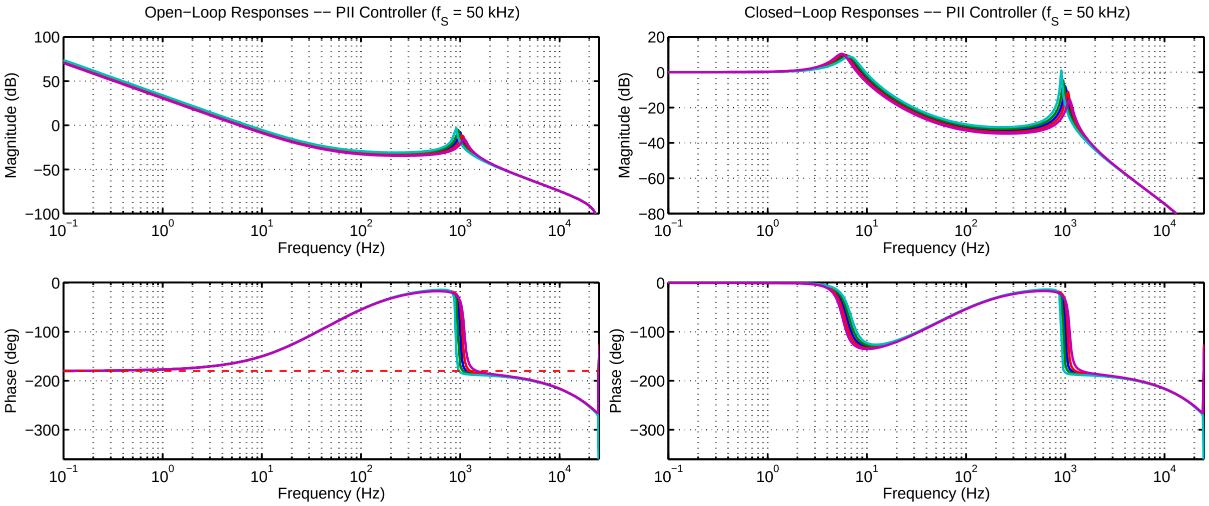


Figure 4.23: On the left: open-loop response for piezo/cantilevers of the left side of Figure 4.21 with digital PII controller from the right side of Figure 4.21. On the right: the closed-loop responses.

The PII controller has more gain at low frequency, at the expense of decreased phase margin. The effects of these choices become clear in the closed-loop plots of the right sides of Figures 4.22 and 4.23, where the PI controller has significantly less bandwidth, but also less ringing than the PII controller. The difficulty in finding a single robust controller for these varying plants which provides both reasonable bandwidth and acceptable gain and phase margins illustrates why there is so much hand tuning of AFM control loops by the end users of the instruments.

The PID controller seems to usually fare a little bit better, as we see from Figure 4.24. Note that the lack of any response from the 300 kHz cantilever resonance indicates that this was notched separately, in this case with a digital biquad filter [54]. The open loop plot on the left shows a response that is very reminiscent of an integrator except in the area of the piezo motor resonance peak. We can see that if the mismatch is small enough, then the open loop phase never crosses $-180^{\circ}$ until after well beyond the 1 kHz piezo motor resonance. However, when the mismatch between the single PID controller and the physical system model is great enough (green and cyan curves), then we have terrible phase margin and possible instability. In most industrial environments, a robust controller that will not go unstable with a small model mismatch wins out over a better performing controller that can go unstable if the parameters are mismatched, and this probably accounts for much of the hesitance to use anything other than $K_{DT} = 0$ unless the system is well modeled by a double integrator. The solution for the robustness and performance issue is to be able to self tune the system model and adjust the controller parameters. Some discussion of how to do this is in [19], and an example is shown in Section 4.17.3 and in [108]. As mentioned earlier, step response methods are associated

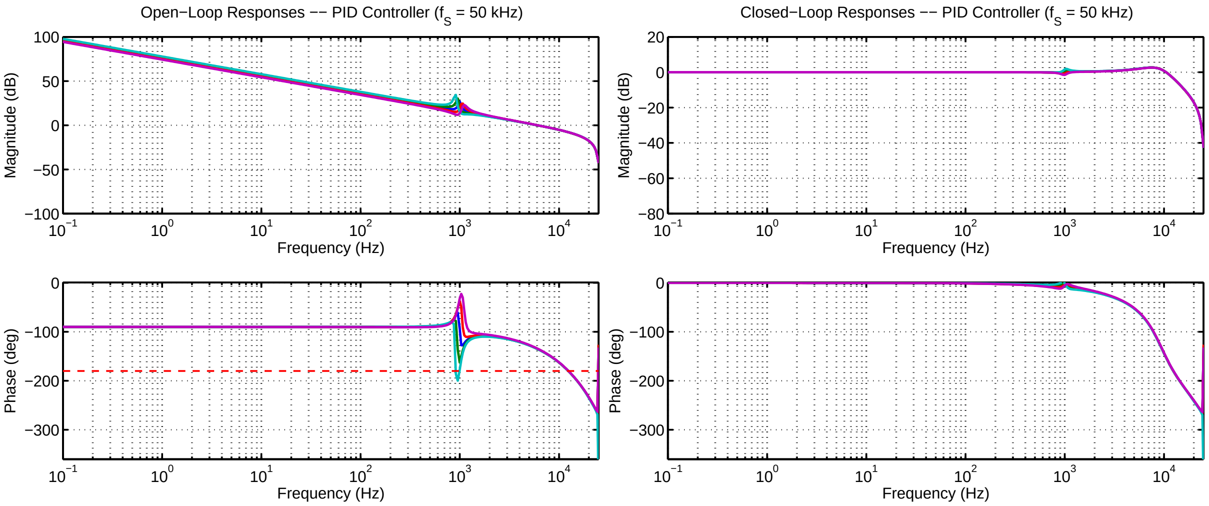


Figure 4.24: On the left: open-loop response for piezo/cantilevers of the left side of Figure 4.21 with digital PID controller from the right side of Figure 4.21. On the right: the closed-loop responses.

with conservative PI control,the example of Figure 4.24 points the use of FRF methods.

Because tube scanners often lack sensors, much of the feedback control work is done only in the Z direction, leaving compensation of the X-Y directions to be done using open loop methods [128], [135]-[137]. Because scanning is most often a raster scan, with a fast axis (X) and a slow axis (Y), the compensation is often applied only to the fast axis.

Note also that the 50 kHz sample rate is only reasonable for actuators with their significant dynamics below about 5 kHz. For smaller actuators - such as those being proposed in higher bandwidth experiments - the control has to be done either with faster sampling or an analog controller [130], [138]-[140].

These issues are fundamental to the control of an AFM. The desire for a single robust, low-order controller is thwarted by the uncertainty in the system. The solution involves either an improved model and/or a higher-order robust controller. Because tube scanners often lack X-Y sensors, much of the original advanced feedback control work was done in the Z direction [141, 142], while feedforward controllers were developed for the X-Y motions [128, 136, 137]. The advent of sensored X-Y stages has led to feedback control methods being developed for X-Y motions as well [143, 144]. Combined feedforward and feedback controllers have also been investigated for both the Z and X motions [145, 146, 147].

In demonstrations of advanced control for nanopositioning, researchers have made careful models of a specific AFM under controlled conditions and then have been able to achieve significantly higher closed-loop bandwidths. While robust control methods may provide practical controllers in the presence of model uncertainty, development of adaptive control methods for AFMs remains an open area that may provide enhanced performance. Further discussion of the control problem from a multi-axis point of view is provided in [148].

### 4.17.3 Loop Shaping on an AFM Actuator Using PID

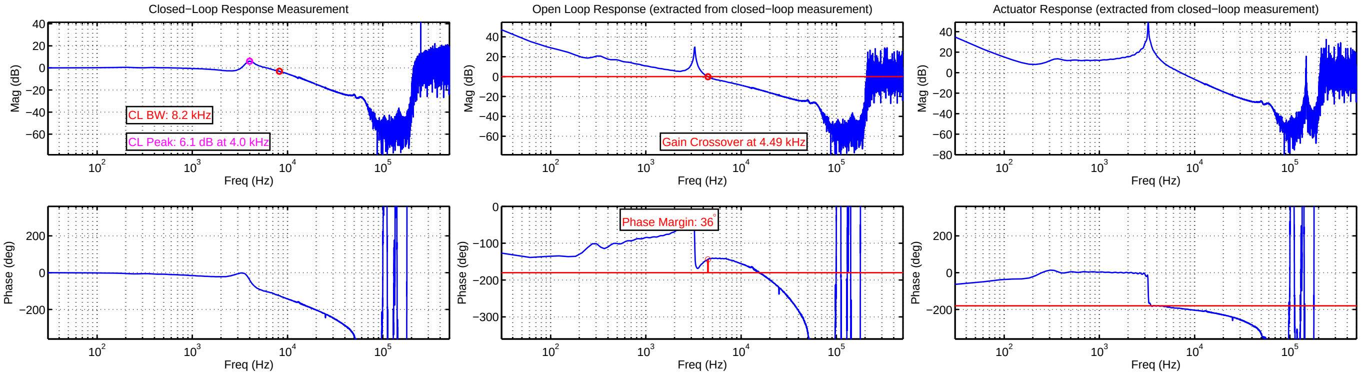


Figure 4.25: A closed-loop frequency response function (FRF) measurement of the original system is on the left. This is followed by opening the FRF loop in the center and then factoring out the plant model FRF on the right, to reveal the plant FRF.

In this section, we give an example of constructing a PID for a mechatronic system measurement. Specifically, we repeat one of the authors work from [108] in which an experimental MEMS actuator for AFMs was identified using FFT based frequency response functions (FRF) [11]. These three steps are shown in Figure 4.25. From the closed-loop FRF (left plot), the loop was opened (center plot), and the controller FRF was divided out to reveal the plant FRF (right plot).

Figure 4.26 shows how, in the range that a reasonable measurement was possible, the frequency response of the plant was fit to a simple resonance with no zeros (left plot). This was then used to design a PID acting as a notch of the motor resonance. The PID was implemented using a discrete backwards-rule equivalent. A biquad filter was used to notch out the plant resonance above 125 kHz. The resulting open loop frequency response looked much like that of an integrator (center plot), and so the closed-loop bandwidth was extended by a factor of 4 while the closed-loop peaking was dropped significantly (right plot).

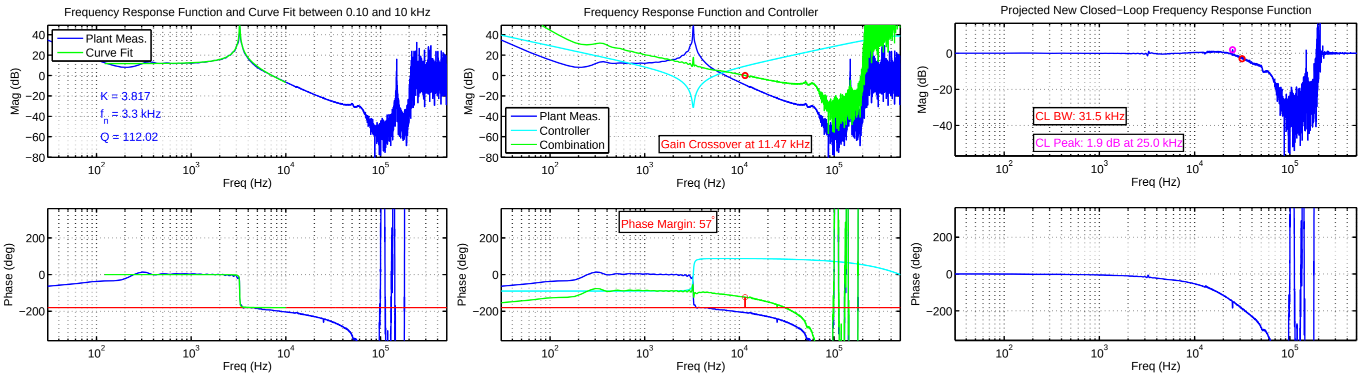


Figure 4.26: On the left is a fit to the main motor resonance, revealing the resonance parameters. In the center, the PID is designed to create a notch for that resonance, resulting in an open loop response that is very much like an integrator. In the right, the new closed-loop FRF is constructed revealing much improved response and bandwidth.

In this example, only one resonance, that of the main motor resonance was shaped. That is a good start, but a more complete answer would address both the main motor resonance and any higher order resonances. We will see some of these in Section 5.8, but before we can do that, we need to describe the controller elements that can successfully compensate for multiple, high frequency, high-Q system dynamics. We will discuss these in Chapter 6.

## 4.18 Integrators, Saturation, and Wind-Up

There are always limits on a signal amplitude or on an amount of control saturation. The latter is often reflected as a model of a physical system limit, where a power amplifier can only drive a finite amount of current, a valve can only open so far, or a joint can only move through a finite range.

These saturation limits are not dealt with explicitly in many control design methodologies, but they present a design challenge for PID controllers. This is probably because the integral action is separated out in a way that can make for isolated analysis and remedies.

The integrator, which is there to help with steady state error, gives phase lag to the system. It is usually the only part of a controller that is not absolutely stable. This is usually fine without saturation, as the loop dynamics stabilize it. In saturation, the integrator keeps accumulating an error that the

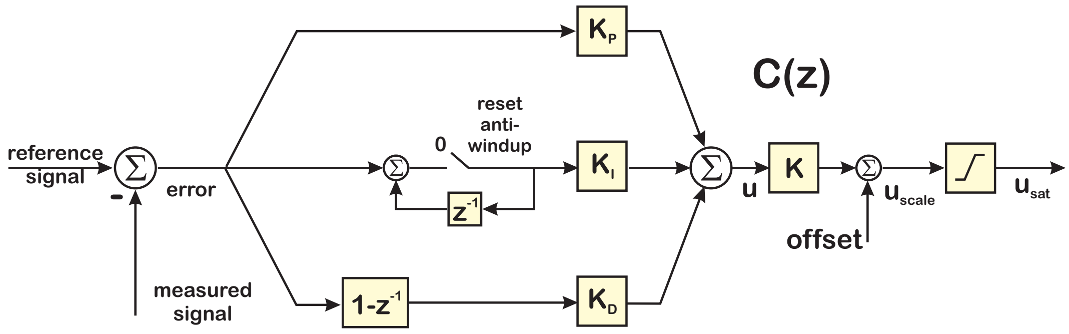


Figure 4.27: **An example of integrator anti-windup in a digital PID controller. This version is called a reset anti-windup. When $u_{sat} \neq u_{scale}$, the integrator value is set to 0 until the system comes out of saturation.**

controller can't fix, at least rapidly. Thus, the integrator sums up "old error" so that even when the actual error is small, the "remembered error" is still inside the controller which thinks that it must be fixed. The error is no longer present, but the integrator has "wound up", resulting in the controller generating errors.

Saturation can happen when the control signal is operating near the saturation limit in regulation, when a noise spike gives a temporary large error signal, or when there is a large change in setpoint. On a large step response, windup can result in a lot of overshoot as the integrator is trying to correct for errors it remembers that aren't really there anymore. Several almost heuristic methods are available for dealing with windup.

- Ignore it. Maybe it's not too big a deal. Intentionally or not, this is often the solution that folks end up with. One of the main issues here is that for systems that are close to the instability boundary, operating in closed-loop requires a certain amount of feedback gains. For certain large errors, the system can get into an unstable region. This is one of those situations where examining the root locus gives a lot of insight, as we can see the gain levels that cause the closed-loop poles become unstable (if they do).

- Reset the integrator. When we hit saturation or have a huge error signal, set integrator to 0 (Figure 4.27). This seems logical for large steps up from zero input, but when the integrator value was nonzero before saturation, instantaneously resetting the integrator will result in a jump in the controller output. Most of the time, jumps in the controller output are frowned upon.

- Back calculation. The basic idea here is to feed back a signal that eventually has the input and

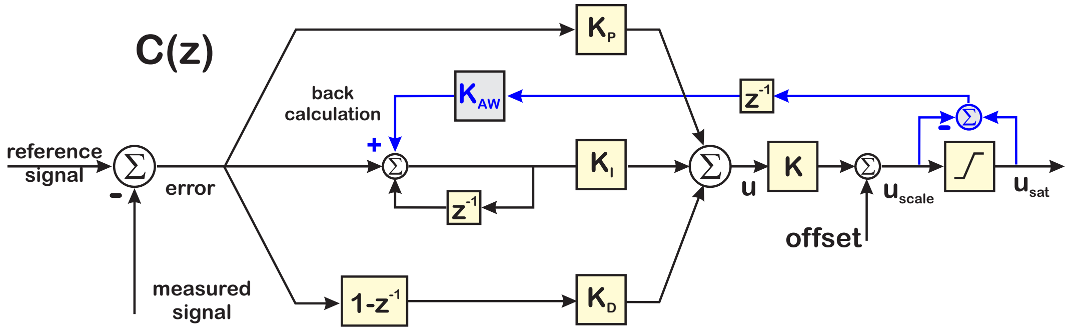


Figure 4.28: **An example of integrator anti-windup in a digital PID controller. This version is called a back-calculation anti-windup. When $u_{sat} \neq u_{scale}$, the negative difference is fed back into the integrator, eventually driving the difference between $u_{sat}$ and $u_{scale}$ to $0$. When the saturation is inactive, that feedback term has a $0$ value.**

the output of the saturation block match. The scheme in Figure 4.28. In this case, the difference between $u_{sat}$ and $u_{scale}$ is fed back through a new gain, $K_{AW}$. This secondary feedback can eventually zero out the difference between the unsaturated and saturated signals.

Note something that only becomes obvious as soon as one tries to write code: we cannot instantly feed back $K_{AW} \cdot (u_{sat}(k) - u_{scale}(k))$ to the integrator as this would try to instantaneously change $u(k)$, $u_{scale}(k)$, and $u_{sat}(k)$. Instead, we use those values as inputs to the next calculation.

Theoretically can null input to integrator so that it is in a good place when control comes out of saturation. Very popular for academic and continuous time systems(Figure 4.29), since it can be implemented without if/then statements [115]. It appears that this is most often used when the saturation is a result of operating near the limit, rather than a large input change. The ability to implement it in continuous time PIDs may have made this the most common anti-windup scheme. In every description that this we have come across using this scheme, the base controller is a PI not a PID.

- Conditional integration or “clamping”. (Figure 4.30) Here the presence of saturation causes the input to the integrator to be zeroed, meaning that the integrator holds at or “clamps” at its current value. This is a safe choice, especially when the system briefly saturates but does not stay there long. An additional integrator bleed off term that allows the integrator to decay while it’s in saturation is sometimes added, as it helps get the integrator to play less of a role in saturation, especially when the loop is saturated for a long time.

  The integrator value is also held when the error is huge. If integrator and error have opposite signs, allow error input to integrator. Very popular with discrete time systems, since if/then isn’t a big deal. Very intuitively pleasing. Furthermore, it handles large reference input changes

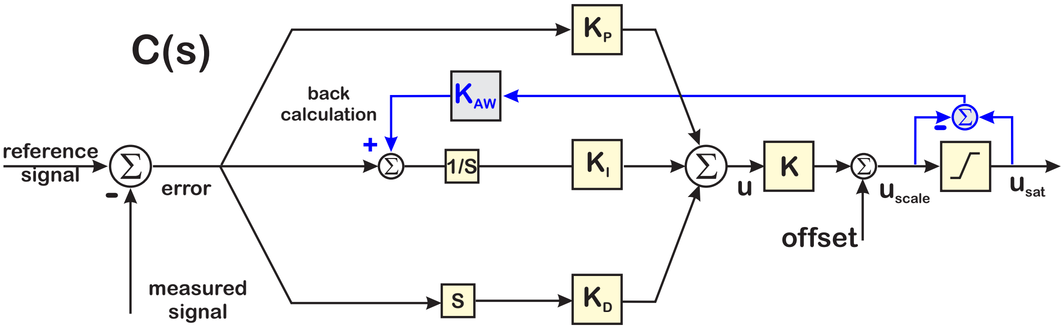


Figure 4.29: **An example of integrator anti-windup in an analog PID controller. This version is called a back-calculation anti-windup. When $u_{sat} \neq u_{scale}$, the negative difference is fed back into the integrator, eventually driving the difference between $u_{sat}$ and $u_{scale}$ to $0$. When the saturation is inactive, that feedback term has a $0$ value.**

consistently as well.

## 4.19 Slow Applications and PWM

While it is tempting to channel our inner Ricky-Bobby and always want to go fast, there are many, many applications for simple controllers/PIDs that are fairly slow by any reasonable computer speed standard. Controlling the “goop” of chemical process control [28, 20], thermal, and pressure control problems all have time constants that are 1 second or slower.

In these applications, the processors are generally much faster than the dynamics of the system (for a more complete description of computing, see Chapter 10). Because of this, and also because such systems often require sturdier cables and signal lines for their harsher environments, it is wasteful to have a multi-bit Digital to Analog (DAC) converter driving a signal line. Instead, these systems often have controller inputs that expect a pulse-width-modulated (PWM) signal. The controller signal is modulated onto binary (0-1) levels with the value determining how much of the signal is at 0 and how much is at 1 (Figure 10.21). The device receives this signal and does its own PWM to analog drive signal conversion. This operation inherently assumes that the modulation is so fast compared to the device that the only the low-pass, averaged value of the PWM signal will be seen by the device, which will give it a reasonable value. We will discuss this more in Section 10.10.

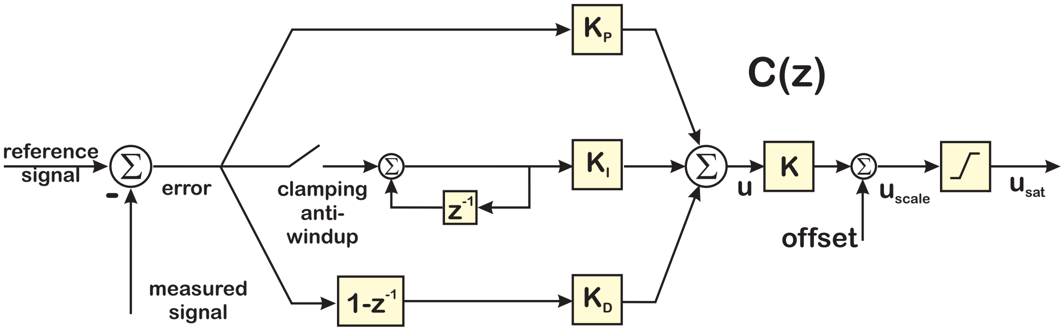


Figure 4.30: **An example of integrator anti-windup in a digital PID controller. This version is called a clamping or conditional integration anti-windup. When $u_{sat} \neq u_{scale}$, the input to the integrator opens up, meaning that the integrator holds its current position until the system comes out of saturation.**

## 4.20 PIDs as an Explanation

There is one more use of PID controllers that comes to mind: as a model for explaining biological control loops. An amazing example is in the paper by Mustafa Khammash and Hana El-Samad [149]. The typical model for “Plasma Calcium Homeostasis in Mammals” uses proportional feedback. The authors essentially say at that point, “But we are control engineers! We know that you cannot get zero steady-state error to a step without some integrator in there.” They then show that PI control explains the observed response and based on that insight, go about finding the biological source of the integral term. It is an under appreciated moment in showing how having control intuition can keep one from assuming bad models.

## 4.21 Conclusions

The popularity and ubiquity of PID controllers in practice begs for some common understanding from the users. This chapter has provided some common frameworks for understanding the various parameterizations of PIDs, so that the engineer who builds them or finds them in a controller purchased from a manufacturer has an idea how to understand the behavior and limits of their controller. We have spent a bit of time in discussing the discretization of PID controllers, as this is a key (and often neglected) part of understanding their behavior.

We have also gone into some depth on how PID controllers can control most of the second order or lower simple system models that often show up in control problems. With a properly PID controller, we can even shape the closed-loop response at will for these problems. If it were all that simple, we would be done here. Unfortunately (or fortunately) real control problems often have higher frequency dynamics. While many control engineers close the loop by staying well below any resonances, this limits the performance of these systems. If we have a little bit of Ricky-Bobby in us and we "wanna go fast," then we need to extend the methods. Chapter 5 starts the discussion with a generalization of ideas on loop shaping.

# Chapter 5

# Practical Loop Design, Or Why Most Open Loops Should Be an Integrator, and How to Get There

## 5.1 In This Chapter

In Chapter 4, we discussed simple controllers for simple models. The idea was that with a first or second order linear model, we can almost always get by with a decent PID controller, and since many, many physical control problems are reduced to first or second order models, we are there. (I think that the British would add something about Jack being a donut or Bob being your uncle, but still.) In fact, in Section 4.17.3, we showed that one could use the PID as an equalizing filter to compensate for a motor resonance, resulting in an open loop response that looked for all the world like that of an integrator (up to a certain frequency when negative phase due to delay kicked in). Doing so allowed us to close the loop and quadruple the original closed-loop bandwidth of the system.

Two questions kind of leap up at us from here:

1) Where did this idea of turning the open-loop into an integrator come from? Did it spring from the ocean fresh, like Botticelli's Venus [150], or was some other line of reasoning involved?

2) What happens when our simple models are not enough, say when the system has significant extra dynamics?

We will start in Section 5.2 with the answer to the first of these, by talking about how I first came to understand that most ubiquitous of human built control systems, with a plant that was always an integrator.

Section 5.3 discusses applying insight from the PLLs to consider turning the open-loop response, i.e. $P(s)C(s)$, into one that matches the response of an integrator. We argue that in lieu of any other pressing design guidance, pushing for $P(s)C(s) \approx K/s$ at least until we are limited by the negative phase due to time-delay – is a pretty reasonable design goal.

In Section 5.4, we visit Bode's integral theorem as taught to the controls community by Gunter Stein in his famous Bode lecture, **Respect the Unstable** [151, 1]. **This discussion continues with some visualization of Stein's dirt digging analogy and the effects of sampling in Section 5.5. We then return to one of the main limitations – time delay – and its effects on what we can do with loop shaping in Section 5.6. Section 5.7 discusses how we can iteratively tune our performance, once we have made that open-loop response look like an integrator.**

**Section 5.8 gets into how we can do loop shaping on systems with multiple resonances, providing examples from a high speed AFM. Of course, this kind of loop shaping depends strongly on having a precise measure of the frequency response of the plant. Section 5.9 discusses how we can still think about loop shaping when our only measurements are step responses. Finally, the options that remain when all we have is operational data are discussed in Section 5.10.**

The simplest way to understand loop shaping is to think of it as a series of filters warping the open-loop frequency response into something that makes the closed-loop response less sucky (technical term). For this reason, understanding filter blocks, how they shape the loop, and what their tradeoffs are is an important skill in servo design. We also need to understand:

a) that most of these will be discrete, either a discrete equivalent or a direct digital design that will only be really understood by thinking of it's continuous counterpart and

b) the phase consequences (intended or not) of the filters used.

However, it happens we are adding a lot of extra variables (knobs, degrees of freedom) to the controller, so why put in all this stuff? Realistically, to be both useful and feasible we should:

- Have a requirement to control faster. If it's not an actual requirement, it's not clear why one has to control faster, why one has to add all the extra complexity.
- Have a reliable model of the higher frequency dynamics from measurements on the system itself. In Chapter 3 we discussed ways to make these measurements more automated, more built in to the controller, thereby minimizing the per measurement costs.
- Have the right elements to add into the loop to shape the design without loosing all track of what is going on or trashing the numerical stability of the controller (Chapter 6).
- Have a design methodology tying the requirements, measurements, models, and compensation elements together in an understandable way.

## 5.2 Phase-Locked Loops: So Much Feedback, Such Simple Analysis

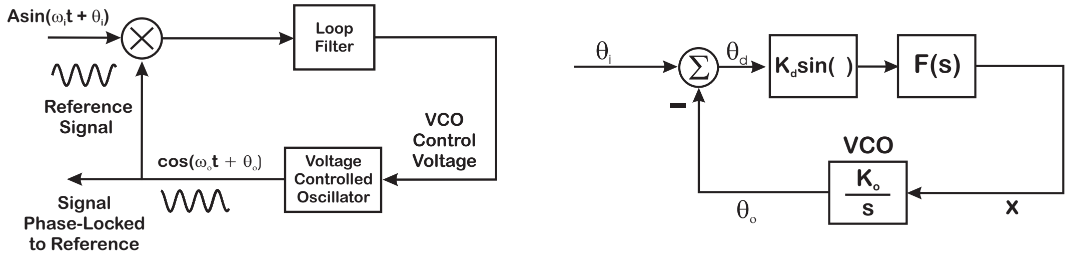


Figure 5.1: **A simple analog PLL and its baseband model. On the left is the basic loop model, where a sinusoidal signal of known frequency but unknown phase enters the system, and a voltage controlled oscillator locks to that input (reference) signal. On the right is the simplified nonlinear model of the baseband system, that is the one arrived at by looking at only how the phases of the oscillatory signals, and not the signals themselves, behave. One final approximation, of linearizing the sinusoidal phase detector, allows the phase behavior to be analyzed as a simple analog feedback loop. The VCO is the "plant" and it is modeled as a simple integrator.**

Early in my career, a fellow engineer at Ford Aerospace named Dan Witmer walked into my cubicle and asked me how I would do nonlinear analysis for a phase-locked loop. "What's a phase-locked loop?" Once he patiently explained it to me, I blindly stated that I would simply use Lyapunov redesign [152, 153]. While nonlinear analysis of phase-locked loops was an interesting subject, it was not the main learning point of these devices. I have since then claimed that PLLs are the most ubiquitous feedback loops built by humans [30], showing up in all of our smart phones, digital watches, and every other computational device we have, and yet the feedback analysis done in textbooks is only of the simplest type [154, 155, 156]. Discussions with PLL experts at several companies also showed that while they knew intricacies of the circuits and the envelope behavior of the phase detector, they generally used only very simple linear feedback analysis in their designs. They paid surprisingly little attention to the stability of the PLL.

It took a while to understand that since most PLLs were first or second order, and for most of these one could show that even the most basic rules of filter design led to closed-loop responses for the phase-space where the parameters that made the linear model stable also made the nonlinear model stable [152, 125]. (In this case phase space refers to the modulation domain or the baseband or envelope behavior of the PLL.) These loops were simple and stable, because the "plant" to be controlled was always an integrator, as shown in the lower drawing of Figure 5.1, and the loop filter was either a gain or a first-order lag, which resulted in stable closed-loop behavior of the system. The Lyapunov analysis showed that in these cases the parameters that made the second-order linear model stable also made the second-order nonlinear model stable. For the circuit designers creating PLLs, they knew from experience that even the simplest, dumbest controller (a.k.a. loop filter) would produce a stable response and so they gave it no mind. Higher order PLLs failed this, and thus were harder to analyze [157].

If our open-loop system was adequately modeled by an integrator, then feedback control of that open-loop system would become trivial (Section 4.11.1). It is easy to show that the first and second-order analog PLLs are always stable, even with the sinusoidal phase-detector nonlinearity [152]. Furthermore, with the right discretization, even the classical discrete-time PLL was stable [125]. What about harmonics? Didn't the designers need to throw in some filters to get rid of signal at the carrier frequency and its harmonics? Why were these not in the textbooks? The answer came from HP/Agilent PLL expert, Rick Karlquist, who laughed and said (more or less), "Well, no RF engineer worth their salt wouldn't know to put in those filters! It goes without saying. That's why it doesn't have to be put in the textbooks." In this moment I realized that any PLL that was not second order was likely to be beaten into a form that looked second order. All those analog RF filters were there, but they were never considered part of the analysis. A lot of work was done to make the system look like an integrator, and then control was done from there. What kind of control was done? Both kinds, lag filter and PI control.[1] The designers were left to work out the more taxing PLL design problems of having a good phase detector and an oscillator with minimal phase noise

And there it was. The most common human-built control system in existence was usually designed with little or no stability analysis beyond what one would do with a PI controller controlling an integrator. Sure, there were those fancy-dancy third order PLLs (used in deep space communications to track through the Doppler Shift of the phase), but most loops were a lot simpler in their linear analysis.

Plus, beyond anything the PLL folks did, there were a lot of advantages to doing proportional, or PI control on an integrator, as mentioned in Section 4.11.1. My inner Homer Simpson was screaming, “Stupid open-loop. If it was only an integrator, then I could do simple loop design, too!” If I could turn the open loop into an integrator, then:

- I could make the rest of the controller pretty simple. In fact, if we separated the problem into (a) turn OL into an integrator and then (b) adjust controller, the (b) portion is really simple and I can look very cool talking about automated scripts, etc.
- The system will have $0$ steady state error to a step input.
- The system will have infinite gain margin and $90^\circ$ phase margin, at least until time delay and noise start to have major effects.

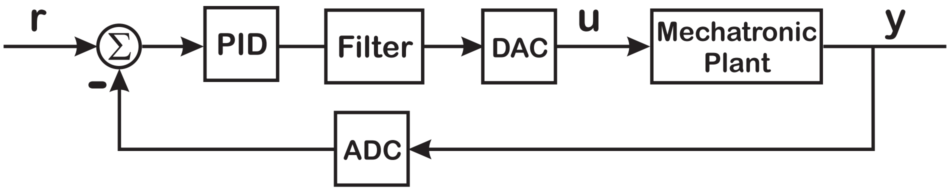


Figure 5.2: **A practical digital control loop for a mechatronic system. The digital controller is often implemented as a PID like controller in series with filtering to lower the effect of high frequency resonances. For now, we are focusing on small mechatronic control systems – of a size and cost that then cannot be adjusted by a human engineer for each device.**

But there was the second problem, that occurs even though engineers had a habit of beating most control problems into something that looks second order. Those PLL circuit jocks will keep putting in filters to remove dynamics and harmonics until they are left with a simple second order PLL (that only has 5 notch sub-circuits). What happens when this fails? What happens when there were dynamics

[1] A tip of the hat to *The Blues Brothers* film.

that we can't ignore, or when we decide to include all those notch filters into our actual design? This is the case for mechatronic control systems, especially when we try to push their bandwidth.

In the movie trailer, this chapter could be labeled, "when simple models go bad." Specifically, we will discuss system models with higher order dynamics, and what this means for control design. In many frameworks, the first resonant mode signifies the frequency at which all control effort should stop. The commonly used PI controllers generally stop at $\frac{1}{4}$ the first resonant frequency. For other systems, such as chemical process control, the performance limiting negative phase is dominated by delays in the system.

A mechatronic system is defined as one in which mechanics and electronics are tightly coupled. Generally, the mechatronic systems that present difficulties are ones that have resonances and anti-resonances with low damping factors (i.e., high Q). I will simply call these $high$-Q systems. The mechatronic systems that I dealt with that presented problems were small-scale mechatronic systems that need to be manufactured in large numbers, such as disk drives, optical disks, scanning probe microscopes, inexpensive robotics, scientific and electronic instruments, consumer products, and the like. This scale of system cannot be individually tuned in the way that a group of engineers might tune the control system for a multi-million dollar fighter aircraft, nuclear reactor, or cargo ship. The controllers either have to calibrate themselves or be robust enough to operate without repeated calibration.

A lot of practical control loops are really controlled with PI controllers, and the open loop gain crossover is set at about $\frac{1}{4}$ the frequency of the first resonance. Traditionally, disk drive control loops would do a lot with the double integrator model and maybe one resonance, but would give up well before all the flexure/sway/arm/gimbal resonances associated with the arm. At this point, some broad notch would be used to damp the magnitude of these and the conservativeness of that notch meant that there was no way that the phase could be accurate enough to make that portion of open loop an integrator. The simple breaking point for all of these was the difficulty in getting high fidelity measurements and models of these high frequency, high Q dynamics. The signal to noise in the FFT frequency domain and the discrete-time model, time domain identification was far too low a these frequencies to even consider this. The FFT based measurements in Figures 4.25 and 4.26 should be convincing of that. The stepped-sine methods produce much cleaner measurements, but if there is plant to plant variation and the stepped-sine is not built into the controller itself, it may be unworkable to have to attach expensive external instrumentation to each small device. Even with the use of stepped sine, we need to do a curve fit to get back to a parametric model and this has its own issues, as discussed in Chapter 3, Section 3.29.

So, what to practical designers of mechatronic systems such as these do? They divide and conquer. Maybe they divide and survive, but they divide. Figure 5.2, shows the breakdown of a mechatronic control loop. The PID handles the simple second-order-model part, and there is a filter block to take care of all those annoying extra dynamics. Of course, we need a couple of things to do this.

- A way to make measurements of all those extra dynamics on top of the simple second order ones. This is the reason for the emphasis in Chapter 3.
- A way to extract reliable parametric models from those measurements. Again: Chapter 3.
- A way to handle the low frequency/baseband/first-or-second-order behavior (Chapter 4).
- Filter sections to shape the loop once the PID has done all it can. We will go into depth in these in Chapter 6. For now, we will assume that the filter components existence that will allow us to add digital filter sections to equalize out dynamics of the open loop until we hit limits imposed by delay and noise.
- A methodology for selecting the gain and bandwidth of that "integrator" open-loop. (This chapter.)
- An understanding of the tradeoffs of loop shaping explained to us by Gunter Stein's talk [151] on Bode's Integral Theorem and Stein's Dirt Digging (this chapter).
- A way of characterizing noise and its effect around the loop, as well as insight on how to keep those noise sources from entering the loop (Chapter 7).

It is worth digressing here to make a note about working with CAD tools in the frequency domain. Most of us use the bode() function in MATLAB to generate Bode plots. MATLAB's bode() routine returns the convenient for plotting magnitude and phase (in degrees). However, if we stop there, we miss a chance to make greater use of the returned data. If we convert this magnitude and phase back into a complex number, we can then manipulate the frequency response function along with others.

- We can combine it with measured FRFs to make a composite response. If we have measured a closed-loop response, $T = \frac{PC}{1+PC}$ we can open the loop via $PC = \frac{T}{1-T}$. If we have a model of $C$, we can generate an FRF for $C$ and extract $P$. We can then fit a model to $P$ for control design. We can generate an FRF for $C_{new}$ and evaluate it both in open loop, $OL_{new} = PC_{new}$ and closed-loop $T_{new} = \frac{PC_{new}}{1+PC_{new}}$.
- This allows us to sanity check designs before any implementation is done.

- It is worth noting that we usually have a cleaner controller model, $C$, than we have of the plant, $P$.
- On the sanity check, the original $C_{model}$ can be compared against the measured $C$. This allows us to asses both the quality of our model and any delays that are occurring between measurement points.
- All of these can be in fairly automated scripts.
- However, none of it can handle unless the frequency axes of the measurements and the models are the same. Thus, it is often best to use the frequency axis generated by the instrument/built-in FRF measurement.

As with many other parts of this workshop, we are not after some optimization of a mathematical abstraction, but a way to set design targets so that we arrive at something that it close to some theoretically excellent result, but at the same time intuitively understandable. The latter part gets us to that sacred ground of being able to debug our systems in the lab or the real world. So, we've got that going for us, which is nice.

## 5.3 Making $PC$ an Integrator

Back in Chapter 4, in Section 4.11.1, we saw that we could learn a lot about what we could achieve with a simple PID controller simply by applying one to a pure integrator, or an integrator plus delay. We saw there that once we have a pure integrator,

$$\frac{X(s)}{F(s)} = H(s) = \frac{K}{s} \tag{5.1}$$

proportional feedback produces an ideal first order low pass filter that can have arbitrary bandwidth set by choosing the correct $K_P$.

$$T(s) = \frac{K_P K}{s + K_P K} \tag{5.2}$$

Note in the Bode plot of Figure 4.7, this idealized system has $90^\circ$ phase margin and infinite gain margin. Of course, time delay limits this, leading to loss of phase margin, which is associated with peaking of the closed-loop response. Still, this ideal open loop response gives us one intuitive model to tune towards: If we can use our filters and loop shaping to transform the open loop into an integrator, then the closed-loop response should resemble an ideal low-pass filter. What are the tradeoffs in this type of a design goal?

For one thing, we can do this type of controller tuning using with remeasured frequency response functions (FRFs) and projected controller FRFs (generated from the new controller model). With this, we can project both the open and closed-loop responses in MATLAB , giving us a estimate of how the system will behave once implemented. We can find the maximum bandwidth subject to constraints of gain margin, phase margin, and closed-loop peaking via relatively simple iterations.

We can also see where our push for more bandwidth results in some margin violation. Perhaps one of the biggest advantages, though, is that a clean closed-loop response can be compartmentalized in a hierarchy. We can employ a divide-and-conquer approach to complexity, in which the inner loops are tuned to some clean closed-loop behavior and that is how that piece is presented to the rest of the system.

Finally, clean open and closed-loop models are far easier to use in combination with feedforward methods, as we will discuss in Chapter 8. In Chapter 6, we talk about using filters to equalize the open-loop response. In particular, we described assigning a digital biquad to each resonant/anti-resonant pair in the identified plant model. By tuning these we can equalize the input-output response of the open loop towards that open-loop integrator behavior. This makes sense to do manually, but generating an automated way of handling this involves some work. It can be done though [103, 19].

Suffice it to say, the loop shaping that one does depends a lot upon the system one has. If one can only make step response measurements or work from operational data only, then loop shaping based on precise measurements of frequency response functions (FRFs), curve fitting, etc. are off the table. Much of this chapter deals where that is not the case; where frequency domain methods are available and we can iterate rapidly. We might think of this as a heavily instrumented frequency response based loop shaping. This often assumes:

1) The system is amenable to injecting test data i.e. non-operational data.

2) The designer can access sufficient test points to characterize the system (Figure 3.30).

3) Either SNR is high enough for FFT based measurements to produce good curve fits of the needed dynamics or a stepped-sine has been implemented.

4) You like, trust, and respect your curve fitter. (This is a big if.)

5) You have everything connected (Section 3.25).

6) The system dynamics are not changing, or if they are changing, they are changing on a fairly long time constant so that one can make a measurement, do all this modeling and design and apply it to largely the same system that was modeled.

If all of these things hold, then one has a chance to do relatively clean loop shaping. The steps are roughly:

a) Get a frequency response of the key loop components. If the system is instrumented to measure $P$ in isolation, great. If not, then perhaps a 3-wire measurement of $P$ or a closed-loop FRF measurement of $T$ is needed (Section 3.18).

b) If we have a closed-loop measurement of $T$, we can unwrap it to get $PC$ via waveform math (applying the same math operations to each ¡x,y¿ pair in the waveform):

$$PC_{meas} = \frac{T}{1-T}.$$

At this point, we can do waveform math to divide out $C$ and get $P$, either using a $C$ waveform generated from a model, or a measured $C$ frequency response function.

c) If $C$ can be measured on the device, do so, if only to validate the generated FRF of $C$ from the design model.

d) If we have both closed-loop and 3-wire measurements, compare

$$\hat{P}_{cl} = \left(\frac{T}{1-T}\right)\left(\frac{1}{C}\right) \; and \; \hat{P}_{3wire}.$$

$\hat{P}_{cl}$ is usually better at high frequency, while the $\hat{P}_{3wire}$ is often better at low frequency, where $T \approx 1$, making $1-T$ close to $0$.

e) Do design on extracted $P$, either modeling it directly (Section 4.17.3) or tuning an inverse model to make the open loop look like an integrator (up to a certain frequency) [19, 103]. We do this by:

- Finding the residual dynamics of the system that have not yet been equalized.
- Assigning a filter section (usually a biquad) to compensate for that particular dynamic. Structures for this are described in Chapter 6.
- Tuning that filter section.

- Iterating on what is left of the new residual dynamics.

f) When the open loop resembles an integrator plus delay, adjust the open loop gain to maximize the open loop gain crossover subject to gain and phase margin constraints. In the absence of anything else, a gain margin constraint of $2$ and a phase margin constraint of $60^\circ$ are pretty safe. That being said, if the open loop truly looks like an integrator plus delay, the gain margin will be infinite.

- We can also examine the projected closed-loop response by doing waveform math to re-close the loop:

$$T_{pr} = \frac{P_{meas}C_{new}}{1 + P_{meas}C_{new}}.$$

This can be compared to the original measurement and to any closed-loop constraints.

g) If the results of the prior step seem reasonable, download the newly synthesized controller to digital controller, and repeat measurements.

To make this at all reasonable, everything has to be hooked together. This involves a lot of programming and wiring, but is well worth it. The ability to iterate is one of the fundamental tools of engineering, and when we make it hard to do so – or when we fail to put in the work to make it easy to do so – we are cutting ourselves off from one of our most basic methodologies. As a rule, control engineers have been lax in putting in the work to connect their various lab systems. This has held us back quite a bit.

What else do we need to consider?

- Latency, latency, latency! When everything else is done, the first main limitation is latency (Section 4.11.1), since as Yogi Bera says, “It’s hard to make predictions, especially about the future.” An appendix to this limitation is uncertainty or jitter in that latency. In many problems today, the digital systems are so fast relative to the time constants of the devices being controlled that latency can be (and often is) ignored. However, for any problem where bandwidth will be pushed to the achievable limit, that limit is imposed by latency. Furthermore, one cannot simply have a fast DSP chip and ignore the latency of tenth-order Butterworth filters on the inputs to the ADCs and the outputs of the DACs. Instead, the entire loop has to be considered in the latency calculation.
- The other main limitation is noise. We might count other disturbances into this, but these can often be detected with a separate sensor. Injection of broadband noise into the loop must be

minimized at the source so that we can minimize moving around Gunter Stein's dirt subject to Bode's integral theorem. Put extra effort into detection schemes (Sections 7.13, 7.19).

Up until now, we've described loop shaping as something that we can do at will, provided we have the tools and are not hampered by delay. However, it is worth noting that loop shaping has a conservative property embodied in Bode's Integral Theorem. Basically, it is a mathematical way of stating that sooner or later (in frequency) we pay for every good deed (namely disturbance/noise attenuation) in a feedback loop (specifically by amplifying noise somewhere else in the frequency domain). Thus, as we shape the loop, and specifically as we push bandwidth to get better tracking, we will get closed-loop peaking somewhere else, and this peaking – if it shows up where there is significant noise or disturbances – can really mess up our system's performance.

## 5.4 Bode's Theorem on Sensitivity Functions

There is is an old theorem by Bode[158] which deals with what he calls regeneration. It turns out that this theorem has some very interesting applications to control systems. This has only recently come to light as a tool for evaluating control systems[159]. However it is the starting point for design methodologies such as QFT[160]. There is even a discrete time version of this theorem[161] that gives some insight on how this theorem is affected by sample rates.

We need to understand this because our loop shaping will affect the system sensitivity and noise amplification. This section will talk about system sensitivity, and about Bode's Integral Theorem and what it tells us about how far we can push a system. It also gives another reason why low pass filters on sensors tend to be useful. Our amplification of sensitivity and noise tends to happen at higher frequencies. By limiting the noise before it ever enters the loop, the extra amplification has a lot less to amplify. Of course, low pass filters also affect the phase of the signals, and if they are in the loop can give negative phase to our loop dynamics. If the filter bandwidth is low enough, those phase effects start to affect our loop phase margins. It's a tradeoff.

Chapter 7 will flip that around and show us how we can use that same understanding to track down noise sources, rate their effects on our system, and then learn how to minimize them before they ever enter the loop.

### 5.4.1 Sensitivity Functions

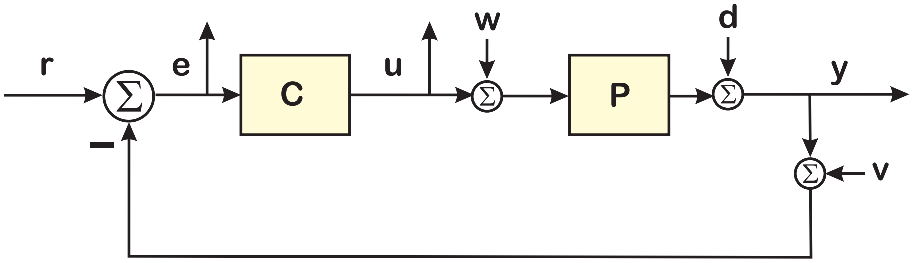


Figure 5.3: Block diagram of closed-loop system.

The block diagram for the following discussion is in Figure 5.3. The closed loop transfer function from $u_1$ to $y_2$ is given by the standard form:

$$T = \frac{y}{u} = \frac{PC}{1+PC}. \tag{5.3}$$

The sensitivity function is also known as the disturbance rejection function. Designated $S$, it is given by:

$$S = \frac{e}{r} = \frac{1}{1+PC} = \frac{y}{d} = -\frac{e}{d}. \tag{5.4}$$

Note that

$$S + T = \frac{1}{1+PC} + \frac{PC}{1+PC} \equiv 1, \tag{5.5}$$

hence $T$ is commonly called the complementary sensitivity function. Note that $S = H_{yd}$ (= the transfer function from $d$ to $y$).

The sensitivity function is important because it shows how disturbances, $d$, go through the system and show up at the output, $y$, or at the error signal $e$. For a unity feedback system

$$S \triangleq H_{yd} = -H_{ed} = H_{eu}. \tag{5.6}$$

Thus, the transfer function from disturbance, $d$, to the output, $y$ is the same as the transfer function from the input $u_1$, to the error (PES), $e_1$, and the transfer function from disturbance, $d$, to error (PES), $e_1$. In other words it is very good gage of how noise will be filtered through the system.

### 5.4.2 Bode's Integral Theorem

The following statements are all different versions of the same idea:

- “Sooner or later, you must pay for every good deed.” (Eli Wallach in the The Magnificent Seven) (Time Domain)
- “No good deed ever goes unpunished.” (The 285th Ferengi Rule of Acquisition) (Time Domain)
- Bode’s Integral Theorem (Frequency Domain)

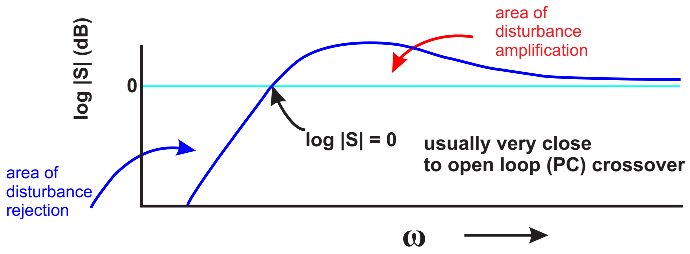


Figure 5.4: Sensitivity function.

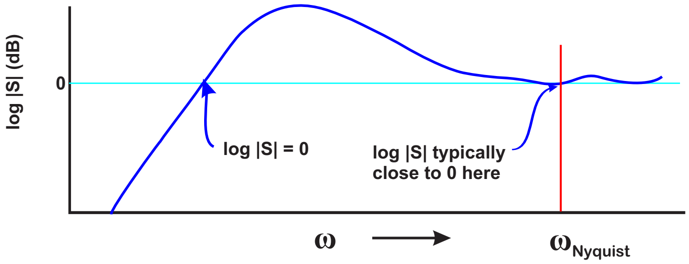


Figure 5.5: Sensitivity function in discrete time.

While the mathematics used to prove both versions of Bode’s theorem can be fairly complicated, the result is fairly simple and extremely powerful. We will leave the proofs to those papers[158, 159, 161] and talk simply about the interpretation. Looking at Figure 5.4 it says simply that:

$$\begin{array}{c}\text{the area of}\\ \text{disturbance}\\ \text{amplification}\end{array} = \begin{array}{c}\text{the area of}\\ \text{disturbance}\\ \text{rejection}\end{array} + \begin{array}{c}\text{a non-}\\ \text{negative}\\ \text{constant.}\end{array} \tag{5.7}$$

Looking at Figure 5.5, we see that for discrete time the main difference is that the Nyquist frequency limits the space we have to work with. In both cases, if we want to attenuate disturbances at one frequency, we must amplify them at another. There is no way to get around this.

**Theorem 1** **(Bode’s Integral Theorem for Continuous Time, Open Loop Stable Systems)** For a sta-

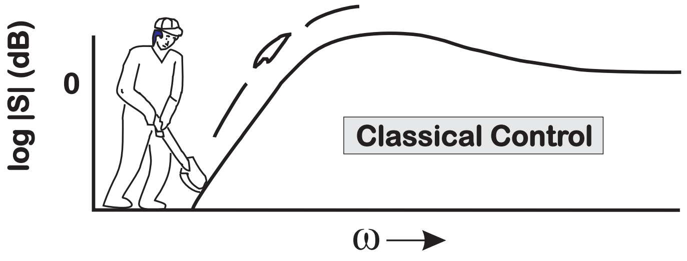


Figure 5.6: Stein's depiction of classical control.

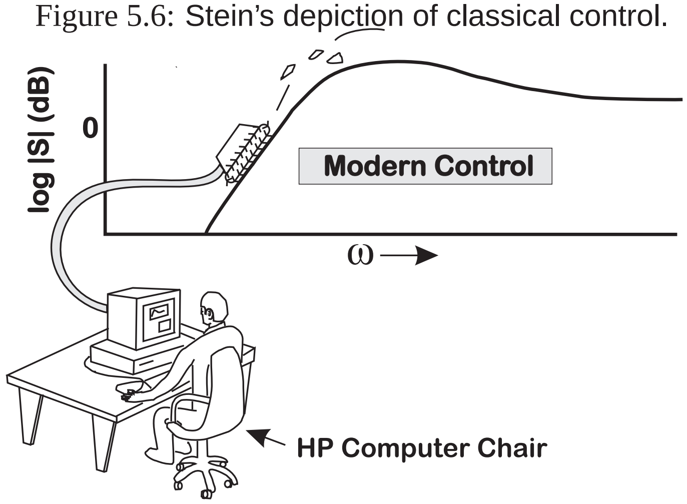


Figure 5.7: Stein's depiction of modern control.

ble, rational $P$ and $C$ with $P(s)C(s) = O(s^2)$ (i.e. they fall off as $\frac{1}{s^2}$)

$$\int_0^{\infty} \log |S(\omega)| d\omega = 0.$$

Consequences: "Sooner or later, you must answer for every good deed." (Eli Wallach in the The Magnificent Seven)

Translation: If you make the system less sensitive to noise at some frequencies, you then make the system more sensitive at other frequencies.

Typical control designs attempt to spread the increased sensitivity (noise amplification) over the high frequencies where noise and/or disturbances may be less of an issue.

A wonderful treatment of this theorem and the importance of it was given as the Bode Lecture at the 1989 IEEE Conference on Decision and Control (Tampa, FL)[151]. The talk, by then Honeywell Researcher and MIT Professor, Gunter Stein, was entitled "Respect the Unstable." Unfortunately, no papers accompanied Bode Lectures at that time, although there is a video of the talk on YouTube [151]. The paper based on Gunter's legendary talk would not be published until 2003 [1]. Stein used this theorem to show how tightly control engineers are dancing when we deal with unstable systems. Stein described the net effect of control systems design as trying to get a certain amount of disturbance rejection at some frequency span while trying to thinly spread the amplification over a large frequency span. Stein referred to this as shoveling dirt. An attempt to recreate his drawing is in Figure 5.6. The guy shoveling dirt is moving around the disturbance amplification. He is doing classical control. He can move the dirt around, but the dirt does not go away. Even with our modern, sophisticated control tools, in Figure 5.7, the dirt is still there.

Now, if the plant or compensator are not stable — , *i.e.*, if $P$ and/or $C$ have finite number of unstable poles — then the result generalizes to

$$\int_0^\infty \log |S| d\omega = 2\pi \sum_{k=1}^{K} Re(p_k)$$

(a positive number) where $K$ is the number of unstable poles of $C$ and $P$ and $p_k$ are those poles. Thus, any unstable poles in the system only make life worse in that more of the noise would have to be amplified.

Note that the integration is done on a linear scale even though these drawings may imply a logarithmic frequency scaling.

### 5.4.3 Bode's Integral Theorem for Discrete Time

The paper on Bode's Integral Theorem for discrete time systems[161] uses a slightly different notation than that used above.

**Theorem 2** **(Bode's Integral Theorem for Discrete-Time Systems:)** For all closed-loop stable, discrete-time feedback systems, the sensitivity function has to satisfy the following integral constraint:

$$\frac{1}{\pi}\int_0^\pi \ln \left| S\left(e^{j\phi}\right)\right| d\phi = \sum_{i=1}^{m} \ln |\beta_i|$$

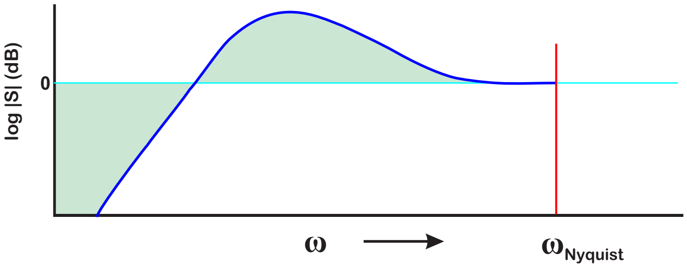


Figure 5.8: Bode's Theorem in Discrete Time

where $\beta_i$ are the open-loop unstable poles of the system, $m$ is the total number of unstable poles, and $\phi = \omega h$ where $h$ is the sample period and $\omega$ is the frequency in radians/sec.

There are some implications of this theorem dealing with discrete time systems. Basically, they say the following:

a) With $h$ as the sample period, the ideal upper limit of the frequency spectrum is $\frac{\pi}{h}$, the Nyquist frequency. Mohtadi assumes for this discrete time theorem that there are no frequencies in the loop above the Nyquist frequency[161]. This would imply that $PC = 0$ for $\omega > \frac{\pi}{h} = \omega_N$ and $S = \frac{1}{1+PC} = 1$, which is in general false for a physical system. However, typical digital control systems do assume that $PC$ is small at or above the Nyquist frequency. Thus, even though the exact assumptions of the theorem will not hold for most physical systems, it is reasonable to assume that some insight can be gained from this theorem.

   Note: This interpretation does leave open the door for multirate control. With the actuator signal going out at a higher rate than the input sensor, it might be possible to do something (good or bad) at frequencies above the Nyquist rate, $\omega_N$, of the sensor.

b) Since we can only manipulate frequencies up to $\omega_N = \frac{\pi}{h}$, and since $|S| \approx 1$ above that frequency, the theorem says that if for some frequency $|S| < 1$ then at some other frequency $|S| > 1$. Unlike the continuous time result, though, there is not infinite bandwidth to spread this over. Thus, the $|S| > 1$ all happens below the Nyquist frequency (and therefore in a finite frequency range).

c) Loop transfer recovery (LTR), as shown in a famous paper by Doyle and Stein[162], cannot be done. LTR attempts to asymptotically match the LQR result that says that LQR provides $|S| < 1$ for all frequencies, in part because LQR uses full state feedback. LTR tries to the same for frequencies up to some point. The $|S| > 1$ part is dumped over the infinite frequency band above that point. The Nyquist limit eliminates the possibility of doing this.

### 5.4.4 What does it mean?

Looking at Figure 5.8, the discrete time version simply states that (analogous to the continuous time theorem):

$$\begin{array}{c}\text{the area of}\\\text{disturbance}\\\text{amplification}\end{array} = \begin{array}{c}\text{the area of}\\\text{disturbance}\\\text{rejection}\end{array} + \begin{array}{c}\text{a non-}\\\text{negative}\\\text{constant,}\end{array} \tag{5.8}$$

and this all this must happen before the Nyquist frequency. The reason why this becomes important is that by working to reject noise at one frequency, we will dump noise amplification at another frequency, but now that the Nyquist frequency establishes a limit, we may end up putting noise amplification at frequencies that we care about.

### 5.4.5 Effect of Sample Rate

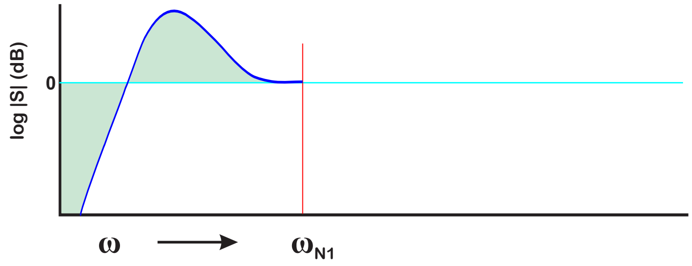


Figure 5.9: Sensitivity function at nominal sample rate, $\omega_{N_1}$.

If the control system is merely a process of shoveling the "disturbance amplification dirt" around, then what does the Nyquist rate signify? It can be thought of as a retaining wall which prevents the "dirt" from going out beyond the Nyquist frequency. Thus, the freedom to spread the dirt around is limited by the Nyquist "retaining wall."

Graphically it is quite easy to see what the theorem implies with sample rate. Looking at Figure 5.9, say we have a certain amount of rejection, $|S| < 1$, for a compensator sampled at $\omega_{N_1}$. This implies a

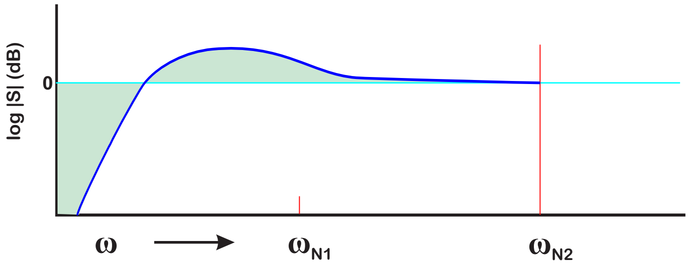


Figure 5.10: Effects of doubling the sample rate ($\omega_{N_2} = 2\omega_{N_1}$). The filtering option.

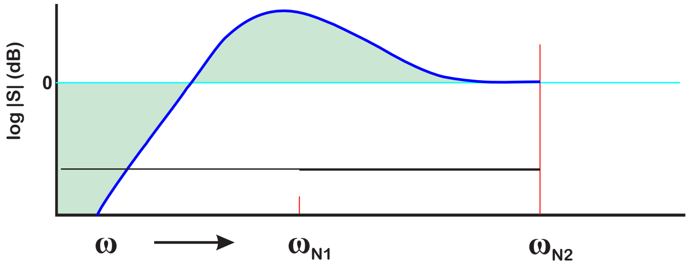


Figure 5.11: Effects of doubling the sample rate ($\omega_{N_2} = 2\omega_{N_1}$). The higher bandwidth option.

certain area of $|S| > 1$. This has to be done before the Nyquist rate retaining wall.

Now, we double the sample rate. With the extra "space," we can either do some extra filtering but keep the bandwidth of the closed-loop system constant (Figure 5.10) or demand more rejection at low frequency and higher bandwidth (Figure 5.11). Note that the effect of using the extra bandwidth to filter is essentially to spread the amplification, $|S| > 1$, over a broader frequency band. This shrinks the height of any amplification hills (Figure 5.10). By pushing the closed-loop bandwidth (Figure 5.11), better performance at low frequency may result in much worse performance at high frequency.

There are two reasons why Bode's Integral Theorem is important in a discussion of a disk drive's position error signal (PES) . First of all, it gives us a very good gage on what we can and cannot do with disturbance rejection and noise in a control system. Amazingly it comes from such an old and simple result that is generally not well known. This result tells us that whenever we improve with the noise rejection at one frequency we pay for it at another. If we are smart and put the noise amplification at places where there is only a small amount of noise, then we do well. If not, we may inadvertently be boosting much of the noise that we are trying to eliminate.

The second reason will become apparent in the next section. It turns out that when we measure PES from a closed loop system, we should actually open the loop and look at PES. The exact same effects that are the point of the above theorem affect our measurement of PES. We will see that when we measure PES that is flat in closed-loop, opening the loop (mathematically in MATLAB or on a spectrum analyzer) will give us a PES spectrum that looks considerably different from the ones we are accustomed to.

## 5.5 Once More with the Dirt

In Section 5.4, we described Gunter Stein's famous explanation of Bode's integral theorem [158] in terms of dirt digging [1]. Moreover, the work of Mohtadi [161] can be interpreted as the Nyquist rate being a retaining wall that limits where we can hide the dirt.

Why return to this now? Because how we design our filters to equalize our responses matters in the sense of noise amplification. Bode's Integral Theorem tells us that sooner or later, we must pay for every good deed. The more dirt we have shoveled with our controller to equalize the response, the more we pay for it in noise amplification, usually near the open-loop crossover frequency or at higher frequency.

This turns into a particular decision point for sharp resonances in mechatronic systems. In the disk drive problems, the flexible modes had all been pushed into a high frequency zone, but at that point, they were hard to distinguish from each other and therefore hard to compensate individually. Instead, drive designers would have low pass filters or broad notches that accounted for the envelope of expected resonances. Such broad filters could not help but eat into the phase margin and it was this phase margin that was often the limiting factor in control bandwidth.

Similar problems were faced in the AFM world, but if the individual resonances and anti-resonances could be identified and equalized out, then high bandwidth could be achieved [19, 103]. One of the fundamental keys was to have a tight notch equalizing any resonance, so that we only shoveled the dirt needed. This cannot happen without precise models, and precise models cannot happen without precise measurements. This is the reason for the focus on frequency response methods in Section 3.16. The Rosetta Stone measurement made by Jeff Butterworth (Figure 3.22, repeated below), makes very clear that not just any FRF measurement will do if one wants to capture high frequency resonances; one wants stepped-sine (Section 3.26).

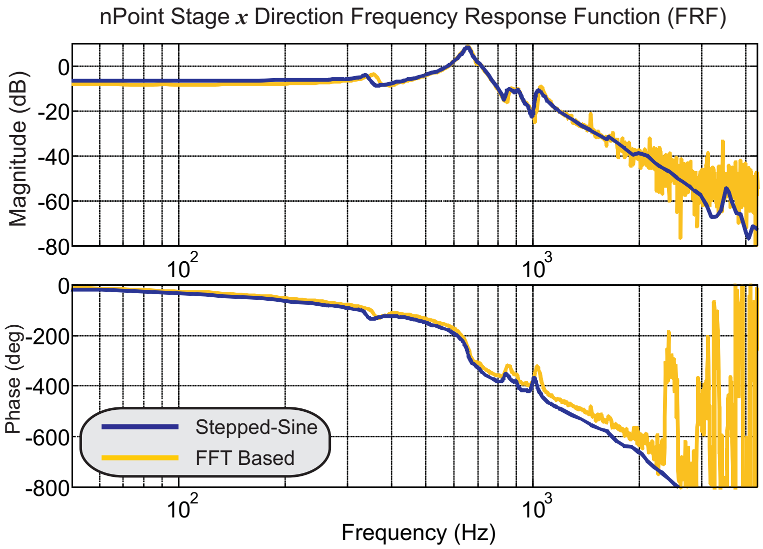


Figure 5.12: Comparison of stepped-sine and FFT based FRF measurements. (Courtesy: Jeff Butterworth).

## 5.6 The Effects of Time Delay on Loop Shaping

Time delay is a subject that might very well find itself in Chapter 3 on **System Models and Characterizing Them with Measurements** modeling section. Certainly, this is where one might first look to find some characterization. However, the characterization of delay often does not motivate us to learn more. It is when we try to build a controller on some system, and the benefits are dramatically limited by the effects of time delay that we are motivated.

For that reason, we will return to the effects of pure time delay on a closed-loop system – brought up previously in Chapter 4 in Section 4.11.1 on **closed-loop PID on an integrator**. In that section, the simplicity of the integrator allowed us to directly consider the effects of the phase-lag due to delay on the resulting closed-loop response. While a main theme of this chapter is to advocate to shape the open-loop response to resemble an integrator (thereby allowing the intuition of Section 4.11.1), there is more that we can say about time delay in general, and how it limits our ability to push the loop shapes we might want.

As noted above, if we have no significant time delay, then by successfully tuning each heater gain and time constant (and in our case we have active heating and passive cooling time constants and an offset temperature), we can have a “perfect” PI control as described in Section 4.11.1. It is time delay (also known as dead time or transport delay) that ruins all of our fun. In different problems, what gives us the relative time delay is different. For example, in a lot of high-speed mechatronic systems, the real delay is in data conversion and computation, while the mechanics themselves are quite fast. In chemical process control, bioprocess, thermal, and pressure control systems, the time constants of the physical process are absurdly slow compared to the computation and so it is the physical transport lag (the time it takes a signal to propagate through fluids, etc.) that give us the delay. We have to be aware that it exists, and it limits us. Even for our best-case scenario of having the open-loop transfer function as an integrator, the addition of time delay limits our achievable responses [19].

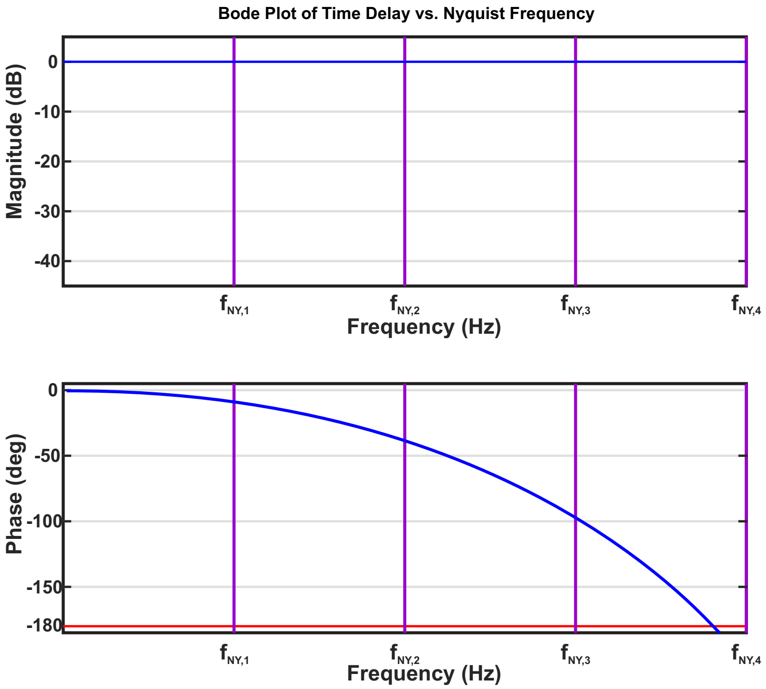


Figure 5.13: Frequency response of pure time delay versus increasing Nyquist frequencies. Increasing the sample frequency, and therefore the Nyquist frequency, does nothing to the time delay, but leaves more "frequency room" in which to add phase lead.

Looked at from a frequency response view, time delay (latency) adds negative phase related to the length of the delay. For a delay of $\Delta$ seconds, we can plot the frequency response of $e^{-s\Delta}$ as shown in Figure 5.13. We can very broadly classify the components of delay in a control as being due to:

- physical properties of the system,
- sensor/actuator effects,
- conversion delays, and
- computational and sample rate delays.

For our purposes here, we will group the sensor and actuator delays in with the physical ones. At the very least we thing of these as an analog grouping, which would require some redesign of the physical system to improve. Conversion delays were discussed at some length in Chapter 10 on computation for real-time control systems.

Computational delays may be minimized to some respect by faster sampling coupled with streamlining processing to minimize latency. For this we are in the domain of hard-real-time programming discussed in Chapter 10. Avoiding computationally expensive operations, e.g. replacing division with fixed divisors into multiplication by a precomputed inverse and/or using look up tables (LUTs) and interpolation for the evaluation of transcendental functions, make us better able to complete our controller calculations in a single sample period. Using pre-calculation [15, 54] can further minimize the time between when a input sample is received and the output signal is returned. None of these computational/conversion fixes have any effect on the physical delay, so without any wholesale plant/sensor/actuator redesign, we will come to some minimum time delay, $\Delta$s, with its Laplace transform of $e^{-s\Delta}$.

### 5.6.1 Time Delay and the Padé Approximation

The problem with modeling a time delay of $\Delta$ seconds is that its Laplace transform is $e^{-s\Delta}$, and this is not in the form of a rational transfer function. This really makes it hard to do typical control systems analysis. Most often, when we try to model this in a transfer function or state space, we often end up using a Padé approximant or Padé approximation [106], which generally introduces one or more NMP zeros. This can be seen as follows. Padd́ approximants are most accurate for relatively small values of $s\Delta$ in the $e^{-s\Delta}$ expression.

There are many forms of this, but for here I'll stick with three different first order versions. If we have $e^x$, we can approximate it as:

$$
\begin{aligned}
e^x &= 1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+, && (5.9)\\
e^{-x} &= 1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+, && (5.10)\\
e^x &\approx \frac{1}{1-x}, && (5.11)\\
e^x &\approx \frac{1+\frac{x}{2}}{1-\frac{x}{2}}. && (5.12)
\end{aligned}
$$

This means that we can approximate $e^{-s\Delta}$ with:

$$
\begin{aligned}
e^{-s\Delta} &\approx 1 - s\Delta, && (5.13)\\
e^{-s\Delta} &\approx \frac{1}{1+s\Delta} = \frac{\frac{1}{\Delta}}{\frac{1}{\Delta}+s}, && (5.14)\\
e^{-s\Delta} &\approx \frac{1-\frac{s\Delta}{2}}{1+\frac{s\Delta}{2}} = \frac{\frac{2}{\Delta}-s}{\frac{2}{\Delta}+s}. && (5.15)\\
& && (5.16)
\end{aligned}
$$

This approximation is diagrammed in Figure 5.14, where we see the stable pole and non-minimum phase (NMP) zero reflected across the $j\omega$-axis.

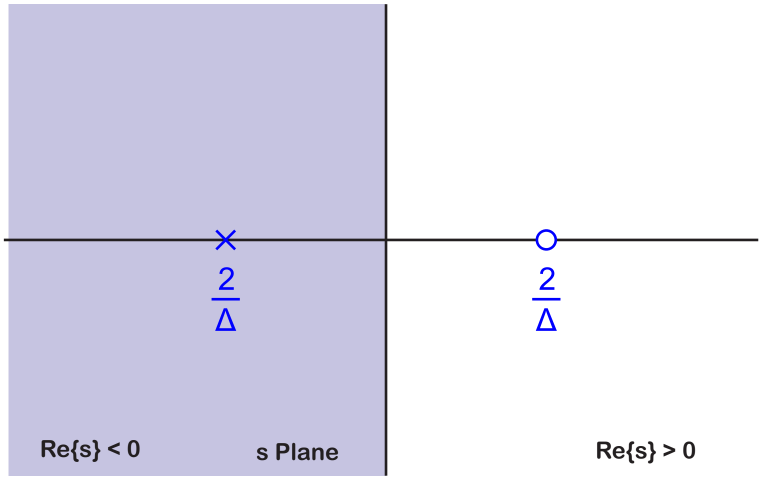


Figure 5.14: The first order Padé approximant for the delay of $\Delta$ seconds ends up with a stable pole and non-minimum phase (NMP) zero reflected across the $j\omega$-axis. The higher the value of the delay, $\Delta$, the closer these get to the $j\omega$-axis, making compensation harder.

Note that the approximations get worse as $s\Delta$ gets larger. We can improve the approximations by using higher order polynomials in the numerator and denominator, but there's a tradeoff between accuracy and usability. For many control problems it's really a choice between the approximation in (5.14) and the more accurate approximation of (5.15). The improved accuracy of (5.15) comes with one major downside: a non-minimum phase zero (i.e. the root of the numerator is in the right half of the $s - plane$). This means that most "equalization/compensation" methods that assume one can invert the response (or close to it) are not available. This is not just a mathematical abstraction: to be able to perfectly invert time delay means achieving what Cher sang about, turning back time, or equivalently exactly predicting the future. In the latter case, Yogi Berra has already pointed out that this is a hard problem.

So, there are two paths here: One is to use the analog approximation up to a point to guide an analog PID design and then discretize that analog PID design. The other is to go through the discretization of this approximation to see how far we can push it. Both are instructive, but eventually, we will probably design using the first choice. For these, and for a situation where the negative phase of the time delay does not dominate the negative phase of the first order system, we will be able to add a bit of lead to the controller to compensate a bit.

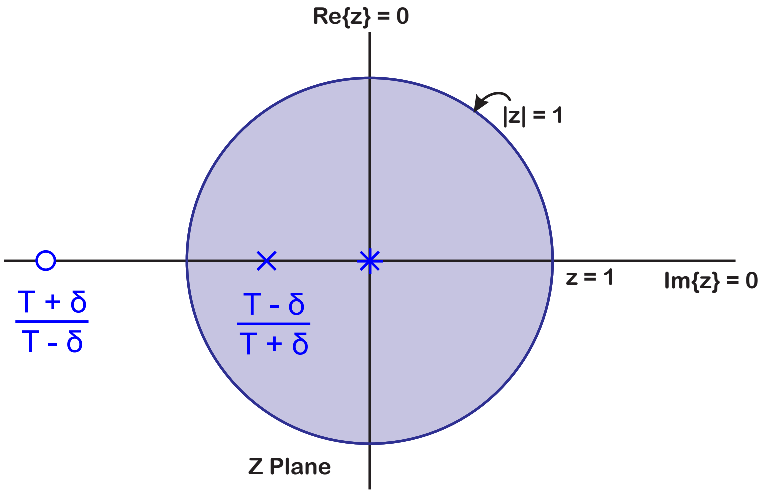

Re{z} = 0
|z| = 1
$\frac{T + \delta}{T - \delta}$
$\frac{T - \delta}{T + \delta}$
z = 1
Im{z} = 0
Z Plane

Figure 5.15: The first order Padé approximant for the delay of $\Delta$ seconds modeled in discrete time. Using Equations 5.17 and 5.18, we have $M$ unmatched poles at $z = 0$ and one reciprocal pair of a stable pole and NMP zero. This means at least $M$ closed loop poles will head to $|z| = \infty$, which is – to be obvious – outside the unit circle. Even the reciprocal pair will result in an extra closed-loop pole going outside the unit circle. As $\delta \longrightarrow T$ the zero will head towards $z = -\infty$ while the pole will approach $z = 0$.

For digital modeling of time delay, it is useful to frame the problem as follows. Say $\Delta > T$.

$$e^{-s\Delta} = e^{-s(MT+\delta)}, \tag{5.17}$$

where $\Delta = MT + \delta$ and $0 \leq \delta < T$. Put this way,

$$e^{-sMT} = e^{(-sT)M} = z^{-M}, \tag{5.18}$$

so we have $M$ poles at $z = 0$. If we sample faster, $M$ gets larger but $\Delta$ doesn't change. $\Delta$ is a physical constant, related to the delay in the physical system. $M$ is a consequence of the relation between that physical constant and the sample period. As we attempt to model $\Delta$, sampling faster makes $T$ smaller, meaning that $M$ has to get larger to keep pace. This means more poles at $z = 0$ as the sample rate goes up. Our intuition indicates that sampling faster should help, but more poles at $z = 0$

means that we have more zeros at $z = \infty$, as diagrammed in Figure 5.15. That's not too big an issue until we close the loop and our knowledge of root locus tells us that the open-loop poles go to the open-loop zeros as the feedback gain gets higher. Thus, the extra zeros at $z = \infty$ put a real limit on how far we can push the feedback. This actually models what happens in the physical world. Time delay does limit how hard we can push the feedback system.

What good is it then to sample faster? We can see this in Figure 5.13, where the Nyquist frequency, the frequency above which a digital (computer based) compensator can do no more, moves further to the right as we sample faster. This, in turn, leaves more "frequency room" in which to add more phase lead. The higher the sample frequency, the higher the frequency at which we roll off the lead. Put another way, more samples faster allows us to do a better job of predicting what will come next.

What about the last bit, the $e^{-s\delta}$ portion of the delay? We have already limited it to being less than our sample period, $T$. We will look at a couple of different discretizations of the two different useful Padé models of delay, limiting ourselves to the fractional portion of the the delay, $0 \leq \delta < T$. For $\delta = 0$ we have exactly $M$ poles at $z = 0$ and for $\delta = 1$ we have exactly $M + 1$. It makes sense then that the best time delay approximation would approach these as $\delta$ gets close to one end of the span or the other. If we use the approximation of (5.14) and discretize with a backwards rule, we get:

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{BR} \frac{\frac{T}{T+\delta}z}{z - \frac{\delta}{T+\delta}}, \text{ for } 0 \leq \delta < T. \tag{5.19}$$

In this approximation, using the backwards rectangular rule, we see that

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{BR} \frac{\frac{1}{2}z}{z - \frac{1}{2}}, \text{ as } \delta \longrightarrow T, \text{ and} \tag{5.20}$$

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{BR} \frac{1}{z-1}, \text{ as } \delta \longrightarrow 0. \tag{5.21}$$

If we discretize with a trapezoidal rule, we get:

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{TR} \frac{\frac{T}{T+2\delta}(z+1)}{z - \frac{2\delta-T}{2\delta+T}}, \text{ for } 0 \leq \delta < T. \tag{5.22}$$

In this approximation, using the trapezoidal rule, we see that

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{TR} \frac{1}{3}\left(\frac{z+1}{z-\frac{1}{3}}\right), \text{ as } \delta \longrightarrow T, \text{ and} \tag{5.23}$$

$$e^{-s\delta} \approx \frac{1}{1+s\delta} \xrightarrow{TR} \frac{z+1}{z+1} = 1, \text{ as } \delta \longrightarrow 0. \tag{5.24}$$

If we use the approximation of (5.15) and discretize with a backwards rule, we get:

$$e^{-s\delta} \approx \frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \xrightarrow{BR} \left(\frac{2T - \delta}{2T + \delta}\right)\left(\frac{z + \frac{\delta}{2T+\delta}}{z - \frac{\delta}{2T+\delta}}\right), \text{ for } 0 \leq \delta < T. \tag{5.25}$$

In this approximation, using the backwards rectangular rule, we see that

$$e^{-s\delta} \approx \frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \xrightarrow{BR} \frac{1}{3}\left(\frac{z + 1}{z - \frac{1}{3}}\right), \text{ as } \delta \longrightarrow T, \text{ and} \tag{5.26}$$

$$e^{-s\delta} \approx \frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \xrightarrow{BR} 1\left(\frac{z}{z}\right) = 1, \text{ as } \delta \longrightarrow 0. \tag{5.27}$$

If we discretize with a trapezoidal rule, we get:

$$e^{-s\delta} \approx \frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \xrightarrow{TR} \frac{(T - \delta)z + T + \delta}{(T + \delta)z + T - \delta}, \text{ for } 0 \leq \delta < T. \tag{5.28}$$

In this approximation, using the trapezoidal rule, we see that

$$e^{-s\delta} \approx \frac{\frac{2}{\delta} - s}{\frac{2}{\delta} + s} \xrightarrow{TR} \frac{1}{z} = z^{-1}, \text{ as } \delta \longrightarrow T, \text{ and} \tag{5.29}$$

$$e^{-s\delta} \approx \frac{\frac{2}{\delta} - s}{\frac{2}{\delta} + s} \xrightarrow{TR} \frac{Tz + T}{Tz + T} = 1, \text{ as } \delta \longrightarrow 0. \tag{5.30}$$

It is the bilinear Padé approximation, discretized with the trapezoidal rule, that results in the most logical digital approximation of $e^{-s\delta}$.

$$e^{-s\delta} \approx \frac{1 - \frac{s\delta}{2}}{1 + \frac{s\delta}{2}} = \frac{\frac{2}{\delta} - s}{\frac{2}{\delta} + s} \xrightarrow{TR} \left(\frac{T - \delta}{T + \delta}\right)\left(\frac{z + \frac{T+\delta}{T-\delta}}{z + \frac{T-\delta}{T+\delta}}\right). \text{ for } 0 \leq \delta < T. \tag{5.31}$$

This is the one that we should use as it matches the endpoints precisely. This will help our modeling when we choose to model delay in discrete time. The unmatched poles at $z = 0$ result in NMP zeros at $z = \infty$. In summary, for discrete modeling,

$$e^{-s\Delta} = e^{-s(MT+\delta)} = z^{-M}e^{-s\delta}, \tag{5.32}$$

and discretize $e^{-s\delta}$ using Equation 5.31 for $0 \leq \delta < T$.

**To sum up, time delay imposes limits on what we can do with our controller. As we base controller design on manipulation of plant models, the most usable analytical models for time delay show us that it will add one or more NMP zeros to the system, in both continuous and discrete time. Fast sampling cannot remove the effects of physical system delay, but it can buy us frequency space in which to add phase lead and mitigate some portion of the delay at lower frequencies.**

## 5.7 Iteratively Tuning the Integrator Response

Once we have shaped the open-loop response to look like an integrator, we still want to adjust the gain of our controller. If we are making that adjustment to maximize performance one could argue that an excellent way to do this is to adjust the controller gain to maximize bandwidth, subject to constraints on:

- gain margin,
- phase margin, and
- closed-loop peaking.

## 5.8 Loop Shaping on Systems with Multiple Resonances

In Chapter 4 on **Simple Controllers for Simple Models (or why so many controllers are PIDs)**, we spent some effort to show how – when the plant model was second order or less – a PID could be tuned to make the open loop look like an integrator. More generally, when the system has dynamics beyond second order, we will try to add filtering into the controller to equalize those out. The specific forms of filters recommended will be discussed in Chapter 6 on **Resonances, Anti-Resonances, Filtering, and Equalization**. Starting in Section 6.14, it will introduces the multinotch filter structure [54, 33]. What we need to emphasize here is that the equalizing filter we choose will need to provide a series of notches, resonances, leads, and lags in order to neutralize the effects of the dynamic features of the plant. It will need to do this accurately enough to equalize the phase as well, and this relies not only on a high fidelity system model, but on a computational structure that is insensitive to small parameter changes.

### 5.8.1 Example: Loop Shaping on an AFM with Multiple Resonances

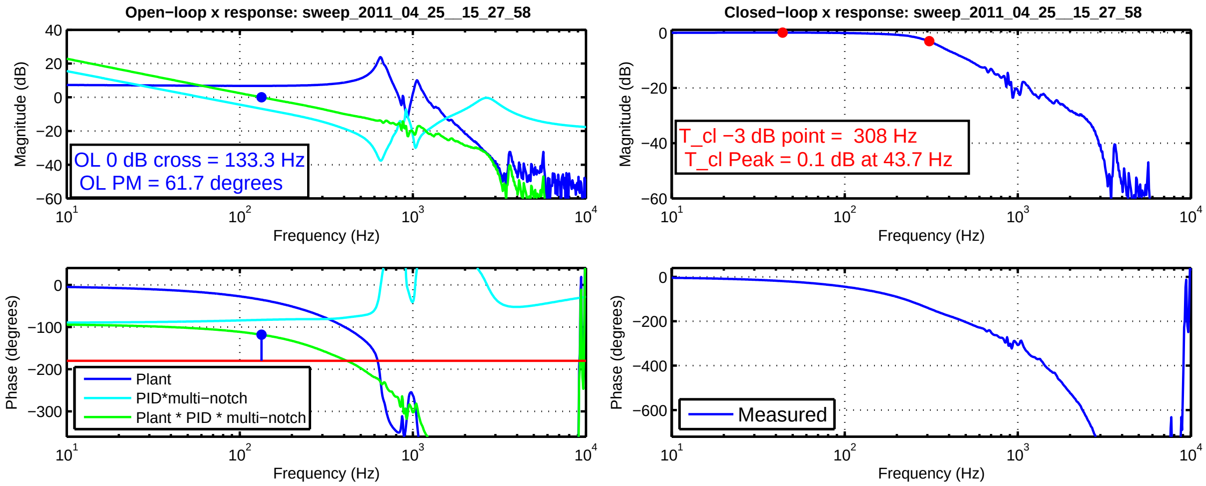


Figure 5.16: On the left: Measured plant, compensator (PID+multinotch), and generated open-loop frequency response for AFM x-axis actuator. The PID and multinotch [54, 33] are automatically tuned to generate an open-loop response that looks like an integrator over the frequency range of interest, allowing the open-loop crossover to be set subject to phase-margin constraints. On the right: The measured closed-loop response from this design. Note the nice, low-pass look of the closed-loop response and the minimal peaking. The closed-loop bandwidth can be adjusted from open-loop constraints or closed-loop peaking constraints.

As the dynamics get more complicated, it becomes much harder to use a PID and a single filter to shape the loop. An example was presented in [19], in which the author used their invention of the low latency, high numerical fidelity multinotch [54, 33] along with the PID tuning [108, 110] to self tune the combined PID and multinotch responses.

As discussed earlier, control of an integrator can easily be optimized by hand in closed-form, but on real systems which have to be shaped to resemble an integrator generally require evaluation by computer. We can use the integrator guidance to build software that searches real measured responses (and projected designs on those measured responses) to find our limits. For a mechatronic system with many high Q resonances and anti-resonances, the combination of PID and multinotch [54, 33], guided by the tuning described in [110], yields results such as those shown in Figure 5.16. Note on the left side of Figure 5.16 that the open loop has indeed been shaped into the form of an integrator. The limiting factor for bandwidth is the phase margin requirement of $60^\circ$. This does result in the closed-loop response of the right side of Figure 5.16 which has 308 Hz bandwidth and virtually no peaking.

Beating the open-loop response into an integrator is certainly not a unique concept, but a brilliant example of this is described in [163] for adjusting the dynamics of the room sized NASA Vertical Motion Simulator.

## 5.9 When All We Have is Step Response

We have concentrated on systems for which we can run high precision frequency responses, generate curve fits, and do loop shaping. However, many problems simply cannot take this kind of identification. For a good laugh, go down to your local chemical plant and suggest a stepped sine on their chemical reactors.

Instead, we are often left with step response, and as we have discussed in Chapter 3, this can only give us a handful of parameters, and these only if we have guessed the correct model from our understanding of the process. In this case, the tuning process looks something like:

a) Assume one of the simple models of Section 2.3.

b) Track the input and output of the system.

c) Trigger a step response measurement with each step.

d) Scale, align, and average the step input data and use that scaling and alignment to adjust the output data.

e) Extract parameters from averaged step response.

f) Apply a simple controller of Chapter 4.

g) Look at the operational behavior of the system under closed-loop. Specifically, a well behaved system in closed-loop would have a closed-loop step response that to first order seems like a well damped second order system.

## 5.10 When All We Have is Operational Data

Can’t drive input signal.

Can still do FFT on input/output data, but it has been argued that these methods do not yield much more than time-domain methods because there is no improved signal richness in FFT methods (Section 3.24). That being said, even with FFT methods, frequency domain techniques often yield a better physical understanding of behavior than discrete-time time domain methods.

May have to use discrete time model, time domain ID.

## 5.11 Chapter Summary and Context

This chapter has been all about loop shaping: what measurements we need in order to attempt it and what shape we want to have once we can shape the loop. We have argued that – barring any other constraint – the best shape for the open-loop response is that of an integrator. We also showed how one might adjust the gain, subject to margin and peaking constraints, once we have achieved that.

# Chapter 6

# Resonances, Anti-Resonances, Filtering, and Equalization

## 6.1 In This Chapter

Chapter 4 started us along the path of what loop shaping could be done with a simple PID controller and hinted that we would use filters to shape dynamics above the rigid body behavior. Chapter 5 pushed the idea that the open-loop frequency response should look like an integrator up to the point in which the phase lag caused by time delay became significant. In this chapter we will talk about the filters themselves, the forms that they take and notes on how to implement and debug them. We will also discuss some of the most useful filters for feedback control systems.

## 6.2 Chapter Ethos

Almost any linear, time-invariant model can be cast as a combination of polynomial filters in the frequency domain.

- Most, but not all analog filters are modeled as an infinite impulse response (IIR) filter.

- Some old finite impulse response (FIR) type filters are implemented using an analogy tapped delay line, where distance down the line indicates delay.
- Digital filtering – or the more general term digital signal processing (DSP) – made FIR filters practical (and ubiquitous, except in feedback systems).
- Digital IIR filters frighten many signal processing/machines learning types. Given that many, many of their problems are insensitive to latency, they have often ignored digital IIR filters for the slow safety of digital FIR implementations. (The chip makers are not blind to this and consequently there are many more built in tools and structures for FIR than for IIR filters.)
- There are plenty of tools to help you design a filter (pick the order and coefficients) given that you decide you need one, such as:
  - Matlab's Signal Processing Toolbox and DSP System Toolbox,
  - Octave's Signal Package, and
  - Python SciPy package.

The reader knows all that. What we can help with here is to give intuition about when and how to use specific filter styles in a feedback system. We can describe most filter behavior with a low-order (usually second-order) model. Higher orders are used to add more features or behaviors, or to increase a particular feature or behavior. We can start by describing most behaviors using analog second-order filters. This will help in understanding the transfer functions and effects of the analog circuitry that connects any digital controller to the real world. Moreover, it provides a more intuitive platform for understanding the filter behavior. We can then map these to either a circuit implementation or to a digital equivalent.

Assuming that the filters are part of the controller, the single time step delay needed to discretize most physical models (e.g. with a Zero-Order Hold (ZOH) Equivalent) is not needed here. Generally, for digital controllers, we should understand the effects of different discretization methods, so that when we do employ one of the design toolboxes/packages, we have an idea what choices it made for us. Moreover, we should know the consequences of filter uses inside (and in the external inputs to) a feedback loop. Mostly, as the DSP and circuit folks often ignore phase and/or delay, we cannot. Finally, we will explain a couple of specialty filters, including the outlier removal (a.k.a. median) filter and the Cascaded Integrator Comb (CIC) filter [164] that are only made possible by analog thinking and digital implementation.

With Chapter 5 as prelude, we will discuss filters in the context of loop design and loop shaping. Filters also serve the task of signal conditioning, of noise attenuation, but we will discussed in Chapter 7. In using filters for loop shaping, it is helpful to consider a few filter “Lego ” blocks. We will do that in Section 6.6.

At that point, we should talk about filter implementation. We will discuss the two basic linear filter shapes, Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). As with most of this material, we will be speaking of digital implementation of filter shapes. The most straightforward of these and what we would generate to analyze in MATLAB is the polynomial form digital filter, by which we mean digital filters as transfer functions in $z$, $z^{-1}$, or $q^{-1}$, where the numerator (for FIR and IIR filters) and the denominator (for IIR filters) are polynomials. We will discuss these in Section 6.3. In this section, we will also bring up the subject of precomputation, a subject that most folks learn in their digital control class and promptly forget afterwards. We will also provide some generic FIR and IIR C code blocks in Section 6.4, along with some tips on programming and debugging.

Polynomial filters are easy to analyze in MATLAB and compact, but for anything higher than a second order filter, the designer looses the physical understanding of what the coefficients mean. Furthermore, they can have severe numerical issues, particularly when they are filtering high frequency, high Q dynamics, and when the operations are in fixed point. A fix to this is proposed in Section 6.11. Finally, we will discuss the mostly beneficial, sometimes nefarious situation of high sample rates, specifically those that are several orders of magnitude higher than the dynamics being shaped/filtered. This has led to a slight modification of the multinotch called $\Delta$ coefficients (Section 6.19). Some large $\Delta$ small $\delta$ confusion in a paper review led me to take a look at the classic $\delta$ parameterization (Section 6.24), and has generated some interesting comparisons in recent years [165, 166].

Finally, we will close this chapter with some common sense filters-in-control-loops dos and don’ts (Section 6.25). Filters can also be used as prefilters on reference inputs to do things like prevent overshoot. This type of filter can be put under the umbrella of the close-loop-input feedforward block in Chapter 8. If we can get to it, repetitive feedforward controllers (which often look like particular forms of FIR filters) and and auxiliary sensor controllers (which also look like filters) will be discussed there.

## 6.3 Basic Digital Filter Ideas

Just as the radio frequency (RF) engineers do with phase-locked loops (PLLs), many scientists and engineers add independent filters to the signal path of their system. For an analog circuit designer, these filters are typically combinations of first or second-order op amp circuit blocks dropped into the signal path to “clean things up”. While the designer will note how the filter improves the signals they see on the scope, few tie any phase effects into things they might see in the overall loop response. When filtering is done on the digital side, it can range anywhere from an average of the last N samples to a cascade of M first-order digital filters to an $n^{th}$ order discrete filter based on an analog Butterworth filter prototype [167, 168]. Depending upon the point in the signal path, the phase effects of these filters may or may not be significant, but it is often the case that this is not considered from an overall system view.

FIR filters are rarely used to shape loop dynamics, because the number of delays (taps) needed to produce the same response as an IIR filter is much larger, and this impacts latency. That being said, programmers and scientists often make the mistake of adding in functions that average the last $N$ ADC samples or stringing together $M$ first-order low-pass filters (which they easily understand), rather than generating a more effective $L^{th}$ order filter, where $L < M$.

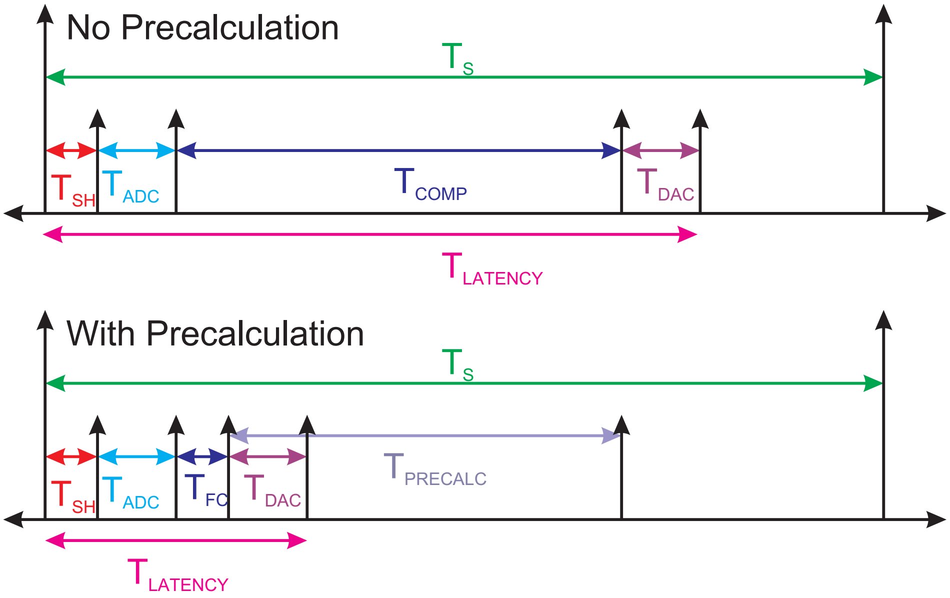


Figure 6.1: Input and output timing in a digital control system. The top drawing is without precalculation; the bottom drawing is with. Note that precalculation can be started as soon as the output has been sent to the DAC and therefore is in parallel with the DAC conversion time. The computation time, $T_{COMP}$, of the top diagram is now split into $T_{PRECALC} + T_{FC}$ where $T_{PRECALC}$ is the computation time needed for the precalculation and $T_{FC}$ is the time needed for the final calculation after the input sample. Modulo some small programming overhead, the split time should equal the total computation time. Here $T_{SH}$, $T_{ADC}$, and $T_{DAC}$ represent the sample and hold, ADC conversion, and DAC conversion times, respectively.

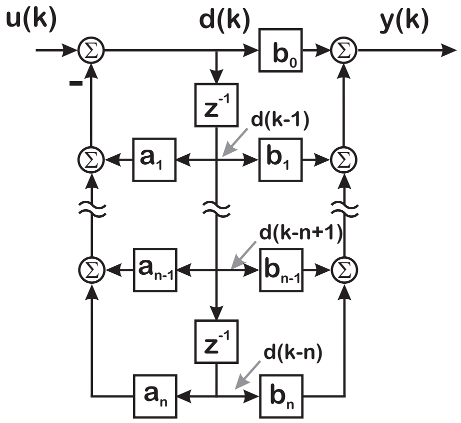


Figure 6.2: An n<sup>th</sup> order polynomial filter in Direct Form II configuration [167].

For feedback systems, we are critically concerned with phase, and filters affect the phase not only by the shape of the response, but by the delay inherent in the processing. Figure 6.1 shows a timing diagram for two mathematically equivalent filter implementations: the normal way without precalculation and the lower latency method with precalculation.

The advent of high speed floating point on DSP chips has made it possible to implement high-order polynomial filters of the form shown in Figure 6.2 in real-time. This allowed the BMW system of [76] to take controller designs from MATLAB and implement them with little modification in real-time DSP on the TMS-320C30.

Single-Input, Single-Output (SISO) digital filters, as shown in Figure 6.2, can be represented as transfer functions in the $Z$ transform operator, $z$:

$$\frac{Y(z)}{U(z)} = \frac{b_0 z^n + b_1 z^{n-1} + b_2 z^{n-2} + \ldots + b_n}{z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_n} \qquad (6.1)$$

or equivalently in the unit delay operator, $z^{-1}$ which lends itself readily to real-time implementation in assembly [169] or a high level language:

$$\frac{Y(z^{-1})}{U(z^{-1})} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}} \qquad (6.2)$$

The transfer function in(6.2) is not unique but has an advantage in that the coefficient of the current output term, $y(k)$, is $1$, so the filter implementation is:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) \\ &\quad + b_0 u(k) + b_1 u(k-1) + \ldots + b_n u(k-n). \end{aligned} \qquad (6.3)$$

Looking at (6.3), we see that $y(k)$ depends mostly on previous inputs and outputs. The only current value needed is $u(k)$ and this is only multiplied by $b_0$. So we can break this up into [15]:

$$y(k) = b_0 u(k) + \text{prec}(k), \text{ where} \tag{6.4}$$

$$\begin{aligned} \text{prec}(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) \\ &\quad + b_1 u(k-1) + \ldots + b_n u(k-n), \end{aligned} \tag{6.5}$$

and $\text{prec}(k)$ depends only on previous values of $y(k)$ and $u(k)$. This means that $\text{prec}(k)$ can be computed for step $k$ immediately after the filter has produced the output for time index, $k-1$ [16]. When the input at time step $k$, $u(k)$, comes into the filter, it needs merely be multiplied by $b_0$ and added to $\text{prec}(k)$ to produce the filter output. Thus, the delay between the input of $u(k)$ and the output of $y(k)$ is small and independent of the filter length. The precalculation for the next step can then be computed.

## 6.4 Programming Our SISO Digital Filter

Most textbooks will go into a ton of theory here before showing you how to actually code the filter. This isn't a textbook, so I do not feel bound by this convention. It's worth remembering that there are reasons for doing various things, but I'm skipping over a lot of them to get you to a practical result.

One does not need to know how we got the various filter coefficients have been determined to program them efficiently. Linear, digital filters, are full of operations that multiply and add, which is why a Digital Signal Processor (DSP) chip is often defined as a processor that has a built in Multiply and Add or Multiply and ACcumulate (MAC) instruction. Generally, for an FIR filter, we think of two vectors of numbers. here,

$$y(k) = b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) + b_3 u(k-3) + b_4 u(k-4) + \ldots + b_m u(k-m) \tag{6.6}$$

we can rewrite this as

$$y(k) = \left[ \begin{array}{ccccc} b_0 & b_1 & b_2 & \cdots & b_m \end{array} \right] \left[ \begin{array}{c} u(k) \\ u(k-1) \\ u(k-2) \\ \vdots \\ u(k-m) \end{array} \right] \tag{6.7}$$

The first vector has the $b_j$ coefficients. The second vector has the current and past inputs. We are essentially forming the output, $y(k)$ at each time step by taking the dot product of these two vectors.

The coefficient vector does not change at each time step, but our vector of current and past inputs does move back once in time with each step.

We can do this via brute force, or we can show some elegance. What constitutes each of these options depends upon which kind of processing we are using. For example, in a CPU or DSP, one can loop through the filter index to do the multiplication one step at a time. The simplest conceptual way to do this is shown in Table 6.1. Note that if I knew that I only had very small filters, say 4-5 taps, I might use this kind of code since the number of wasted operations is still pretty small. Another case where this might be useful is implementation on an FPGA, where the filter delays might be in the fabric, not in a memory block. However, this code is pretty wasteful otherwise, as there is an entire loop dedicated simply to shifting data backwards in an array before the new data value is entered.

As with the PID code displayed in Section 4.16 a something needs to be pointed out about the code segments that follow, namely that we haven't shown the top of the segment, which would have to take into account how to have the prior values in the subroutine. Likewise, we haven't discussed where the gains get passed in. This is important because for most filter subroutines the gains don't change every iteration, if at all during the operation of the system. Remembering that most functions/subroutines by default blank any data not passed in on the parameter list, we have a few options for keeping around old values from previous steps. As in Section 4.16, the options are:

- Use global variables. This is a very old programming practice, which is usually discouraged. Global variables are space efficient and fast, but since any routine can access them (not just the routines who are supposed to), they can cause a lot of massive bugs, and are best used sparingly.
- Retain the history of variables that need to be persistent outside of the routine, passing them to the routine in the parameter list. The efficiency of such a choice depends largely on the programming language. In a language where one passes data to and from routines by value (e.g. in MATLAB functions) then this potentially means passing a lot of data back and forth. In one where one can pass by reference (e.g. in C/C++/C#, etc.) then only a reference is passed, but since these are scalar quantities, there isn't too much savings when passing a reference to a float compared to passing the float. This changes when we are passing a pointer to a structure or class, as opposed to a scalar value.
- Use persistent/static variables in the routine. These variables keep their values between calls. Doing this usually requires some sort of initialization flag, so that we know the first time we are entering the routine, to initialize the variables. This can work when there is only one instance of a particular routine, say when there is only one PID control block. It becomes a mess when

there are multiple loops trying to use the same piece of code, as each would somehow have to keep track of their own persistent variables. This leads to the logical conclusion of . . .

- Use object oriented programming (OOP), because – again paraphrasing Neil Diamond (and you've had a couple of chapters in which to Google this reference) – that's what it's there for. We do this by creating a filter class, which has its own member data (a.k.a. persistent data) and member methods (a.k.a. routines). Each instance of the class has its own persistent data space, so we can store and access parameters, old values of data, etc. This requires more memory and can be a tiny bit slower than one of the raw methods above, but memory is a lot cheaper than it was when C was first developed and modern compilers mean that the extra abstractions of C++ cost very little time.

```
//------------------------------------------------------------------------
// One step of an FIR filter.  The accumulation is done in a
// quantity called sum2.  The order of the filter is loaded into
// the variable N.  There is no computational elegance here, so
// the code wastes some opportunities.
//------------------------------------------------------------------------
N = order;
for (k = 0; k < N; k++){
  delays[N+1-k] = delays[N-k];   // delays(j) = delays(j-1)
}

delays[0] = xIn;                 // Load the current input
sum2 = 0.0;                      // Blank the sum

for (k = 0; k <= N; k++){        // sum2 += b[k]*delays(k)
  sum2 = sum2 + coef[k]*delays[k];
}

yOut = sum2;                     // Store the output
```

Table 6.1: Brute Force FIR Filter Code Snippet in C++

The same filter can be implemented in the mode shown in Table 6.2. In this case, we have gotten rid of the giant shift block by using an index into the *delay* array, which is shifted at each step. Thus, our past values of input do not move; rather the last value is replaced with the current input once it has served its purpose in the multiply. Note that I also changed the indexing into the *delay* array, so that it now moves forward as the indexing of the coefficients moves backwards. This is because the last

```
//---------------------------------------------------------------
// One step of an FIR filter.  The accumulation is done in a
// quantity called sum2.  The order of the filter is loaded into
// the variable N.  Note that the indexing goes forward in the
// delay vector and backwards in the coefficient vector. The input,
// xIn, is loaded into the delay vector last, and multiplied by
// the first entry in the coefficient vector, coef[0], which
// corresponds to b[0].
//---------------------------------------------------------------
sum2 = 0.0;                        // Blank the sum
N = order;
for (k = 0; k < N; k++){
                                   // sum2 += b[N-k]*delays(n-(N-k))
  sum2 = sum2 + coef[N-k]*delays[del_ind];
  del_ind = (del_ind + 1) % N;     // Increment index mod N
}
delays[del_ind] =  xIn;            // Store current state
sum2 = sum2 + coef[0]*delays[del_ind];
del_ind = (del_ind + 1) % N;       // Increment index mod N
yOut = sum2;                       // Store the output
```

Table 6.2: More Efficient FIR Filter Code Snippet in C++

multiply is by $b_0$, which has to use the most current input. Only after all the other multiplies have been done does this input replace the oldest of the delay values.

Now, one thing that should be discussed are subroutines/functions (or methods in an object oriented world) and persistence of the data. Typically, a filter routine is invoked and passed the most recent input, producing the newest output. This means that the routine must have a memory of coefficient values and prior data values. They must be persistent or static, so that they are around even after the routine exits. Barring that, they could be global variables, but this solution has is a really bad programming practice in general, and limits the scalability of the program. However, for real-time code, a lot of minimalist programming methods are used to save space and execution time. At the other end of the spectrum, the filter coefficients and previous values can be passed to the routine each time it is called. However, this is a lot of overhead in each filter call. For filters that are not in a real-time environment, object oriented code really shines here because each filter class can have individual instances of the class for each needed filter, and those instances can hold all the static memory of each filter.

Using what we have learned in this FIR filter, we can now efficiently implement an IIR filter of Equation 6.3 as shown in Table 6.3. We now have a coefficient vector that holds $2N + 1$ coefficients, where $N$ is the order of the filter. We make our indexing efficient by computing both the denominator sum and the numerator sum in the same loop. And we have done a trick with our delays, creating an internal set of signals that are analogous to internal states of the system. More on why this is so good in later sections, but this little loop does all the filtering for our IIR filter. Of course, we know how to program these now, but we haven't said anything about what the coefficient values should be. (There should be a drum roll here.)

```
//-----------------------------------------------------------------------
// One step of an IIR filter.  The accumulation is done in  two
// separate sums, sum1 and sum2.  The indexing is made more efficient
// by indexing the same delay element for two multiplications at
// each step.  Thus, the numerator and denominator coefficients
// are paired together with the exception of the b0 coefficient
// that is multiplied outside of the loop.
// Because this is an IIR filter, we assume coefficients of
// numerator and denominator are set by the highest order of
// either (we can have zero coefficients to get around this)
// and so we have 2N+1 coefficients and N+1 delay elements for
// an Nth order filter.  However, by first using delay elements
// from prior steps, we can actually only keep N delay elements.
// As with the FIR, the order of the filter is loaded into
// the variable N.  Note that the indexing goes forward in the
// delay vector and backwards in the coefficient vector. The input,
// xIn, is loaded into the delay vector last, and multiplied by
// the first entry in the coefficient vector, coef[0], which
// corresponds to b[0].
//-----------------------------------------------------------------------
sum1 = 0.0;                              // Blank the denominator sum
sum2 = 0.0;                              // Blank the numerator sum
N = order;                               // Better for looping

for (k = 0; k < N; k++){
                                         // sum1 -= a[N-k]*delays(n-(N-k))
  sum1 = sum1 - coef[2*(N-k)]*delays[del_ind];
                                         // sum2 += b[N-k]*delays(n-(N-k))
  sum2 = sum2 + coef[2*(N-k) - 1]*delays[del_ind];
  del_ind = (del_ind + 1) % N;           // Increment index mod N
}
delays[del_ind] = sum1 + xIn;            // Store current state
sum2 = sum2 + coef[0]*delays[del_ind];   //
del_ind = (del_ind + 1) % N;             // Increment index mod N
Out = sum2;                              // Store the output
```

Table 6.3: IIR Filter Code Snippet in C++

### 6.4.1 Filter Programming Tips

The simplest filter on the planet lent it's name to a great HBO series, The Wire. A wire filter is exactly what you think it should be: the output should exactly match the input. This seems trivial enough until one tries to make a digital filter subroutine out of it. Most likely it will fail, and when it's a wire, i.e. $b_0 = 1$, everything else $(a_i, b_i) = 0$, it's not because of numerical issues, or some poor choice of coefficients. Thus, the wire should be the first set of filter coefficients used in any of your filter routines. Until you can see the output mirror the input, the rest of your code isn't right.

The next filters raise the complexity by setting $a_1 = \frac{1}{2}, a_0 = 1$ for an IIR filter test and $b_0 = b_1 = \frac{1}{2}$ for an FIR test. These simple tricks and nonsense will save you hours of debugging time on more complex filters.

In signal processing, it is reasonable to batch filter all the data in many applications. After all, time delay doesn't matter. Unless the filtered data is being streamed to another output right away, batch filtering is okay. However, in batch filtering we enter the subroutine, filter all the data and exit. The variables don't go out of context; they aren't zeroed out. On the other hand, real-time filtering (used in control) essentially enters the routine, does one time step of the processing, and exits. This means that the data has to be static in memory between calls to the subroutine. Different languages have different words for this (static in the C/C++/C# world, persistent in MATLAB ), but it is important to have these.

## 6.5 Generating Filter Coefficients

This is the question that all parents anticipate and get nervous about: when your child comes up to you and says, “Mom/Dad, where do filter coefficients come from?” “(Cough) Well, um, you see, when a numerator and a denominator love each other very much . . . ”

Given that we have shown how to write simple subroutines that perform IIR or FIR filtering efficiently, what is left is the generation of those coefficients, i.e. filter design. The sections that follow will give some general filter shapes and what they do, also describing if they can be implemented in analog and/or digital forms, or if they can only be implemented using digital means. We won’t go too deeply into this, since there are many, many excellent books, papers, and tutorials about any specific, complex filter shape. Instead, in this section and the ones that follow, we are after intuition for what certain filters will do and how to implement them without doing something foolish (as that happens as well). Our goal here is to have an intuitive understanding before ceding all control to the machines.

Whether we are using filters to limit the noise coming into a loop, as part of the loop compensator, or to post process the measured or estimated signals from a loop, there are a handful of characteristics that they can exhibit, depending upon the structure and the coefficients. It is also worth having this intuitive explanation when one is discussing a topic like filtering with coworkers who often do not have an engineering background.

Most of the filtering we do (including the IIR and FIR code above) is linear filtering. FIR filters perform a weighted average of some number of the past inputs. FIR averagers do this with the weights all being equal. The effect of any filter can be considered to either smear or sharpen a signal response, in a particular frequency range, based on the coefficients. If all the coefficients of an FIR have the same sign, the filter can only smear the signal. For them to sharpen any signal, there must be some sign changes somewhere, which represent digital approximations to derivative action. IIR filters perform a weighted average of some number of past inputs and past outputs. Because of the action of poles, the sharpening available in an IIR filter has much more “bang for the coefficient” than FIR filters.

FIR filters have really hit their stride in the digital world. Often, this means that folks who work with digital FIR filters have lost track of the physical signals on which they operate. Furthermore, the lack of an equivalent analog circuit often means that there isn’t an accompanying analog prototype to consider. In our experience, it is best to retain some knowledge of the actions of different parts of the filter. This might mean having a cascaded analog prototype for initial design. As we will likely implement these filters digitally, we should pay attention to the effects of discretization on these subsections. However, we should be aware that there is a large set of problems for which the dynamics are so well damped, the latency constraints so minimal, and the need to extract physical signals so unnecessary, that some high order polynomial form FIR or IIR filter works just fine. We do not ignore these problems, but believe very strongly that we should be certain we are in one of those problems before abandoning the physical intuition.

## 6.6 Basic Filter Types & Understanding

If we consider a feedback controller as having a PID portion for low frequency behavior and a set of filters for high frequency behavior, we can view these filters as equalizers that allow a complex system model to be simplified to one that can be controlled with a PID. That complicated filter generally can be considered to be composed of smaller blocks whose behavior we can often understand with first or second order models. FIR filters are rarer in feedback systems because for the same loop shaping they require many, many more taps. We will discuss mostly IIR filters, with a few exceptions.

### 6.6.1 First Order Digital Low Pass

Consider the first order, low pass, digital filter, in the frequency domain form:

$$F(z) = \frac{1-\alpha}{1-\alpha z^{-1}} \tag{6.8}$$

This has a pole (a root of the denominator polynomial) at $z = \alpha$ and has a DC gain (gain to a constant input) of $1$. The time domain version of this (what we would actually program in a computer to filter signals) looks like

$$\begin{aligned} y(k) - \alpha y(k-1) &= (1-\alpha)u(k) && (6.9) \\ y(k) &= \alpha y(k-1) + (1-\alpha)u(k). && (6.10) \end{aligned}$$

We have our digital filter. The response is shown diagrammatically in Figure 6.3. Actually, the diagram in Figure 6.3 is what our low pass filter strives to be. In real life, there are tradeoffs. We don't have to go through tons of analysis on these tradeoffs, but it is good to have an idea about them so that we

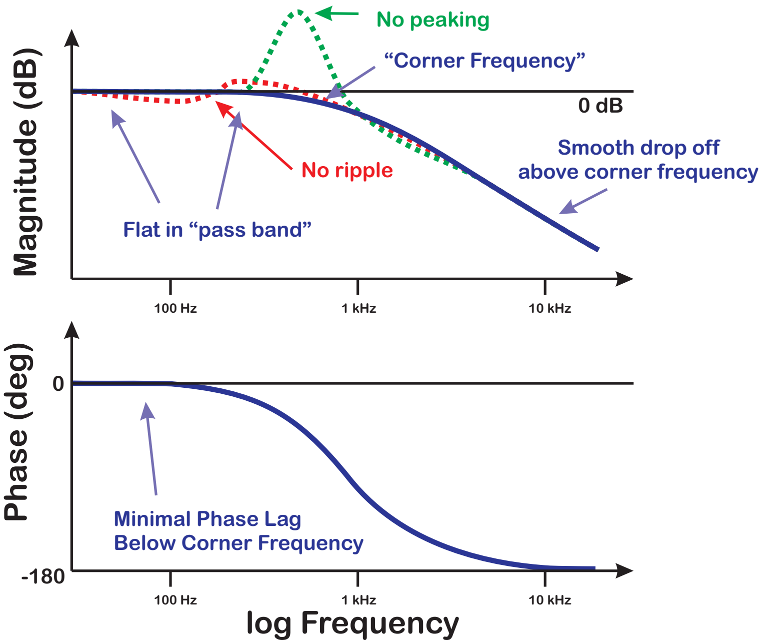


Figure 6.3: A diagram of an ideal low pass filter response. Frequencies in the pass band go through unchanged, while frequencies in the stop band are attenuated. The filter bandwidth is typically determined by the so called corner frequency, at which the log of the magnitude of the response turns downward.

can make smart design choices. A couple of things about the diagram: we see that the filter response that we most care about is the magnitude, that is we compute the magnitude of the complex filter response at each frequency and plot it. Also, we plot the magnitude in decibels (dB), which for a filter response is $20 \log |F(z)|$. Note that $20 \log 1 = 0$, so the filter gain of $1$ at low frequency shows up as $0dB$ in our plot. Our first order filter will be more rounded than the filter in our diagram, but the general idea is that low pass filters have responses that look like this.

How do we pick $\alpha$? Well, let's consider an analog low pass filter with the frequency domain form

$$F(s) = \frac{a}{s+a}. \tag{6.11}$$

This filter might be implemented with electrical circuits, mechanics, pneumatics, whatever, but would have this same model. This filter has a pole at $s = -a$. (In case you are wondering, the analog filter is described by differential equations in time and Laplace Transforms in frequency. The digital filter is described by difference equations in time and Z Transforms in frequency. What we're doing is drawing some connections between continuous and discrete time and frequency stuff.)

For this filter, we can generate what we diagrammed in Figure 6.3. This actually tells us how this filter

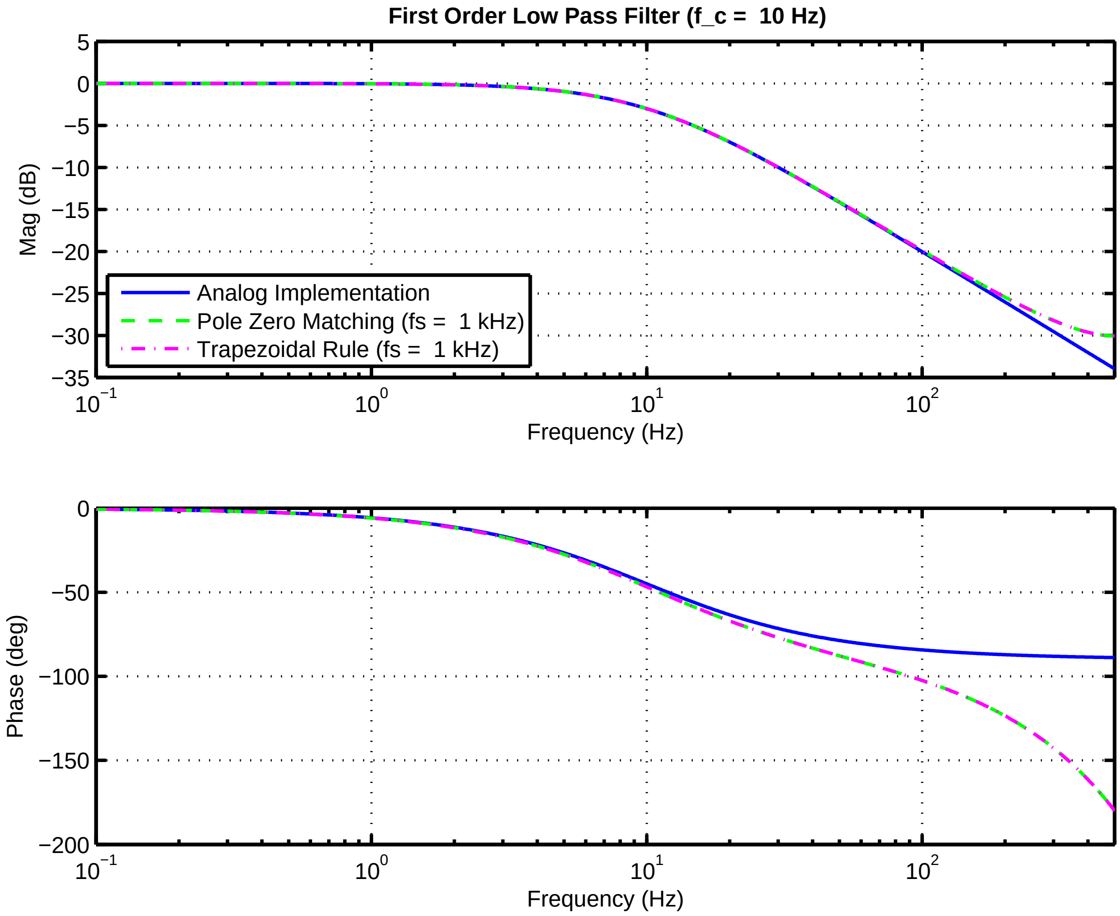


Figure 6.4: Three versions of a first order low pass filter with corner frequency at 10 Hz. The blue curve shows the continuous time (analog) version of the filter that one might build with circuits. The green and magenta curves are digital equivalents which could be implemented on a computer. Add backwards rule version.

would respond to sinusoids of various frequencies. For the analog filter of Equation 6.11 we get the blue plot of Figure 6.4. This is called a Bode plot, in honor of a famous Bell Labs engineer, Henryk Bode, and it breaks apart the magnitude (top curve) and phase (bottom curve) responses. Furthermore, not only is the magnitude is plotted in dB, but the frequency axis is also logarithmic. Anyone who has looked at audio equipment specifications has seen curves such as these. They indicate that signals at frequencies well below the corner frequency will pass through the filter unaltered, while signals at frequencies above the corner frequency will be attenuated. Hence, this is a “low pass” filter as low frequencies will pass.

The lower “phase” plot is a lot less familiar. Basically, it shows how signals of different frequencies are delayed by different amounts. Low frequency signals experience almost no delay, while high frequency signals experience up to 90 degrees of delay.

We can also see that the green and magenta curves which represent two digital filter approximations of the analog filter do not match all the way. In particular, we see that there is “extra” negative phase, which is entirely due to the digital approximation. This distortion can be reduced by raising the sample frequency (lowering the time between successive samples), but it never completely goes away.

A way to understand the frequency at which the low pass filter acts is to use a special version of the frequency, where $s = j\omega = j2\pi f$. Our pole at $s = -a$ means that in terms of $f$ we have a pole when $f = \frac{-ja}{2\pi}$. Usually, we drop the $-j$, but we know that if we want a low pass filter that lets things pass at frequencies below $f_c$ and attenuates them at frequencies above $f_c$, then we set $a = 2\pi f_c$.

Now, you have my to trust me here, but we map poles in continuous time to poles in discrete time with the relationship:

$$z = e^{-aT_S} \tag{6.12}$$

where $T_S$ is the sample period (the time between successive samples) of our digital filter. Since our digital filter of (6.8) has a pole at $z = \alpha$ we set

$$\alpha = e^{-aT_S}. \tag{6.13}$$

Equation 6.8 now becomes

$$F(z) = \frac{1 - e^{-aT_S}}{1 - e^{-aT_S} z^{-1}} = \frac{b_0 z}{z + a_1}, \tag{6.14}$$

where $b_0 = 1 - e^{-\alpha T_S}$ and $a_1 = -e^{-\alpha T_S}$. This filter corresponds to the green curve in Figure 6.4.

Another way go from (6.11) to (6.8) is to know from our study of differential equations and Laplace Transforms (pretend that you’ve had these already) that the $s$ in Equation 6.11 represents differentiation in the time domain (it does). Since we are going to the discrete domain, one thing we would want to do is to approximate differentiation. There are a lot of ways to do this, some more dubious than others, but let’s choose one that is halfway decent most of the time, the trapezoidal rule (TR) equivalent or Tustin’s rule [15]:

$$s \longrightarrow \frac{T}{2}\frac{z-1}{z+1}. \tag{6.15}$$

(The simplest explanation of this rule is that it is what you use when you want to approximate an integration period by a linear fit to the two endpoints. The area that approximates the slice being integrated is a trapezoid (hence the name), but rather than using this relationship to integrate we are using the inverse of the relationship to differentiate.)

Now, we start with (6.11) and substitute in Equation 6.15.

$$F_T(z) \quad = \quad \frac{a}{\frac{T}{2}\frac{z-1}{z+1} + a} = \frac{\frac{a2}{T}}{\frac{z-1}{z+1} + \frac{aT}{2}}. \tag{6.16}$$

$$F_T(z) = \frac{\frac{\frac{a2}{T}}{1+\frac{aT}{2}}(z+1)}{z - \frac{1-\frac{aT}{2}}{1+\frac{aT}{2}}} = \frac{b_0 z + b_1}{z + a_1}, \tag{6.17}$$

where $b_0 = b_1 = \frac{\frac{a2}{T}}{1+\frac{aT}{2}}$ and $a_1 = -\frac{1-\frac{aT}{2}}{1+\frac{aT}{2}}$. If we want to check the DC gain (response to a constant signal), we set $z = 1$ (trust me here, this is the digital version of DC)

$$F_T(1) \quad = \quad \frac{\frac{\frac{a2}{T}}{1+\frac{aT}{2}}(2)}{1 - \frac{1-\frac{aT}{2}}{1+\frac{aT}{2}}} = \frac{aT}{2\frac{aT}{2}} = 1 \tag{6.18}$$

This filter corresponds to the magenta curve in Figure 6.4.

Hmm. What just happened here? I used two different methods to go from the analog filter to the digital filter. The DC gains are the same (1), but the filters aren't the same. Well, it turns out that we can approximate $e^{-aT}$ by

$$e^{-aT} \approx \frac{1-\frac{aT}{2}}{1+\frac{aT}{2}}. \tag{6.19}$$

This means that the denominators actually are pretty close. In fact, as $T \longrightarrow 0$ this approximation becomes increasingly accurate. If $T$ is our sample period, $T_S$ then we see how fast sampling makes all our discrete models more accurate.

The numerator is a different story. The two methods yield slightly different digital numerators. For now, let me just say that the differences involve different digital design philosophies. The results in the Bode plot look the same for these first order filters, but will diverge for higher order filters (filters defined by more complicated polynomials).

The morals of this exercise:

1) We can design/understand physical properties and behavior of filters using analog concepts.
2) We can map these analog designs to digital (computer) designs by digital approximations to derivatives/integrals over a time step or by using pole/zero mapping methods (the $\alpha = e^{-aT}$ thing).
3) There is generally a trade off between mathematical exactitude of the approximation and retaining any physical intuition.
4) Good design involves understanding the analog (physical world) and the digital (computer world).
5) We can do things that produce exact digital results, but most physical intuition is lost.

Note that a lot of the IIR digital filters one sees in programs are these simple first order filters. They work, but not that well, especially when we are trying to knock down noise and interference. Later sections will give examples of higher order filters that give better bang for the buck. Mostly, these can be composed of combinations of second order filters. From these, we can get low pass (lets low frequency signals pass), high pass (lets high frequency signals pass and keeps out bias), band pass (lets frequencies in a specified band go through), and notch (takes out signals at a single frequency out). First, we'll go the other way and describe a very common FIR filter, the averaging filter.

## 6.7 Averaging Filter

It's often the case that signals are simply averaged. We take a running average of say the last 7 samples and produce a filtered output. We can describe this as an FIR filter. A diagram for this is in Figure 6.5. Say

$$y(k) = \frac{1}{N}\left[u(k) + u(k-1) + \ldots + u(k-N+1)\right]. \tag{6.20}$$

This is an N-tap averager. It averages the last N values. We can look at the discrete frequency domain polynomial by applying $z^{-1}$ for each time delay so that

$$F(z) \quad = \quad \frac{1}{N}\left[1 + z^{-1} + z^{-2} + \ldots + z^{-N+1}\right] = \frac{1-z^{-N}}{1-z^{-1}} \tag{6.21}$$

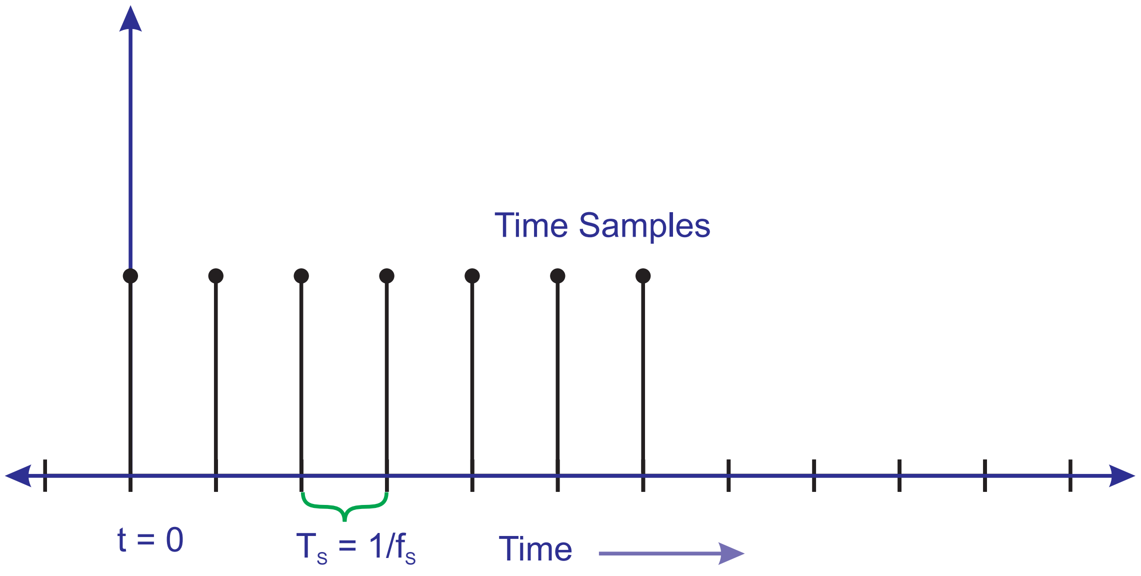


Figure 6.5: An N-sample averaging filter. N samples are summed, and then the result is divided by N.

$$= \frac{z^{N-1}}{Nz^{N-1}}\left[1 + z^{-1} + z^{-2} + \ldots + z^{-N+1}\right] \tag{6.22}$$

$$F(z) = \frac{1}{N}\left(\frac{z^{N-1} + z^{N-2} + \ldots z^{1} + 1}{z^{N-1}}\right) \tag{6.23}$$

So, an N-tap averager is an FIR filter of order N-1. It is called an all zeros filter because it can be defined as in Equation 6.21 as a polynomial in $z^{-1}$. However, it really is a filter that is a rational function of polynomials in $z$, where the numerator defines the behavior of the filter, but the denominator is N-1 roots at $z = 0$. As for the roots of the numerator? Well it turns out that a $N-1^{th}$ order polynomial with all coefficients equal to $1$, has $N-1$ complex roots of radius $1$. That is, if we draw a circle of radius $1$ on the complex $z$ plane, all the roots of this polynomial will be on that circle, and all the roots of the denominator will be at the center of the circle. (Include a diagram.) (This circle is significant enough in digital filter work to be known as the “unit circle” and where the roots of the numerator and denominator are – relative to the circle – tell us a lot about the filter.)

Another interpretation comes from the far right side of Equation 6.21, where we see that this N tap FIR filter can be implemented as an integrator that subtracts off a delayed version of the input. Thus, an averager is the same as an IIR filter that is an integrator, but that subtracts off the input from $N$ samples in the past. In other words, this IIR filter performs an FIR operation. This is known as a Cascaded Integrator Comb (CIC) filter [164]. It can be used to dramatically lower the number of

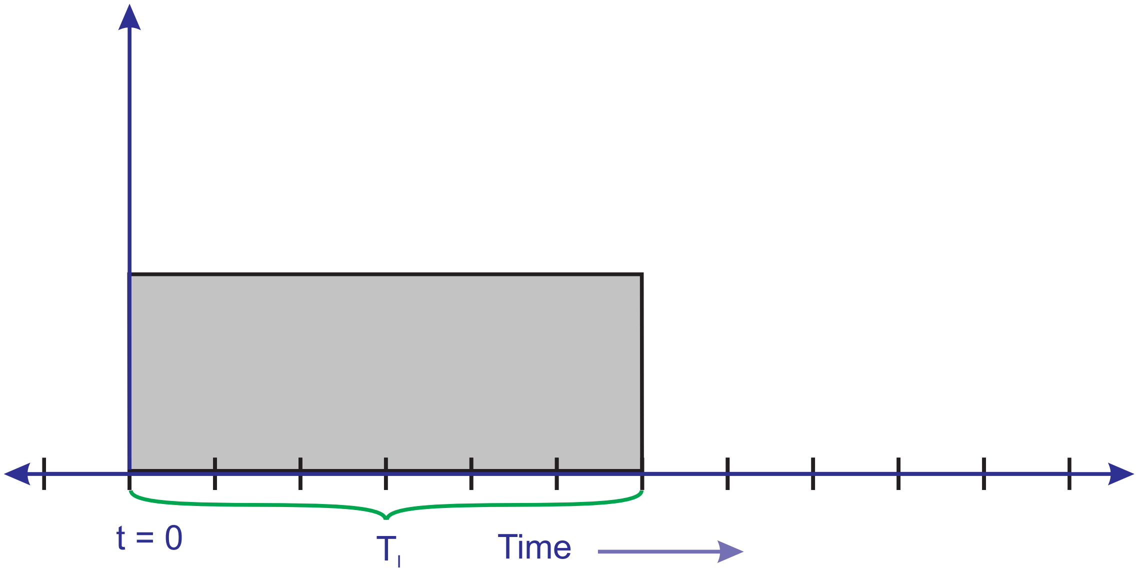


Figure 6.6: An analog finite integrator filter, also called integrate and dump. The integration takes place over a length of time, $T_I$, and the result is usually divided by that to get an average value.

computations needed to average $N$ samples (see the integrator in the low latency, high fidelity AFM demodulator of Section 7.19), but – and this is really important – being an integrator one has to be very careful that even small differences in subtracting off an input can lead to an integrated error that overflows the registers very quickly.

The averaging filter is almost always associated with digital filters. The analog equivalent would be something called an integrator working over finite time:

$$y(t) = \frac{1}{T_I} \int_0^{T_I} u(t)dt. \tag{6.24}$$

This is diagrammed in Figure 6.6. If we set $T_I = T_S$ we see part of the Zero Order Hold Equivalent.

The frequency domain version of this (you just have to trust me until you take the classes) looks like:

$$Y(s) = \left(\frac{1}{s} - \frac{1}{s}e^{-sT_I}\right)U(s) \text{ or equivalently} \tag{6.25}$$

$$\frac{Y(s)}{U(s)} = \frac{1 - e^{-sT_I}}{s}. \tag{6.26}$$

The weirdness is the $e^{-sT_I}$, which isn't a rational function of polynomials. There are ways to approximate this with rational polynomials (called a Padé approximation, which has its own issues). My point was not that we would ever use the finite time analog integrator, but that there really is an analog version of this common digital filter.

### 6.7.1 Mini Summary

Okay, there was a lot of stuff about IIR and FIR filters here. How do I know which one to use?

**FIR Filter:**
- Always stable.
- Can have linear phase (technical but sometimes important).
- Easy to multiplex (useful for filter banks).
- Computationally easy.
- BUT . . . lots of taps for a given filtering action.

**IIR Filter:**
- A lot less phase per tap, but more complicated.
- Can be unstable if we don't pick coefficients the right way.
- BUT few taps for filtering.
- USE this when delay matters.

## 6.8 Two Simple Methods to Remove Outliers

For a variety of reasons, sensor data may be corrupted by really large noise values. This is hard to remove with linear filtering, which essentially is a weighted average of past inputs (FIR) or past inputs and outputs (IIR). Large outliers skew the results of such filters. At some point, one has to decide if those large deviations are so far out of the model that we should throw them out. For example:

- A twos complement ADC or DAC will – when crossing an analog 0 value, go from all digital all 1s to digital all 0s (or the other way) and cause a large noise spike in the digital circuitry. Even an offset binary format can go from almost all 1s to all 0s with a MSB (most significant bit) 1 when crossing analog 0. While it may not be enough to flip bits that transient can radiate to other analog signal lines.

- A signal may overflow a fixed-point number, causing it to go from its maximum possible positive value to its minimum possible negative value (or vice-versa). This can be prevented with proper embedded software or firmware design, but many systems exhibit this.
- Noise on the digital supply rails may cause some signal to go from a normal value to its maximum or minimum digital value and back.
- In systems with electronics near high voltage components of the system, arcing in the high voltage sections, or large signal noise in actuators, motors, etc. can manifest itself in the digital section as “pops” and dropouts.

None of these digital “jumps” or “pops” are part of our normal system theory. They are not linear, they are not happening on a predictable schedule, and they provide no useful information to our measurements and our algorithms. Removing them can only minimally touch the data processing inequality [170] because their appearance in our signal cannot be considered part of the same Markov process. The question then is how to remove such useless, parasitic signals if we cannot use linear filtering.

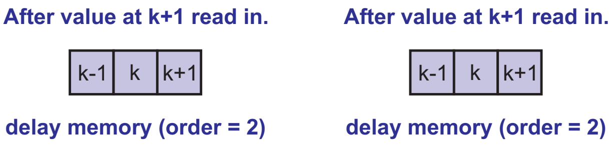


Figure 6.7: Simple filter memory and indexing for outlier removal.

If we consider a run of data and think about the last $N+1$ samples, including the current one. For simplicity, we make $N$ even. If we look at the point in the center, there are $N/2$ samples on either side. In the examples of Figure 6.7, $N$ is either 2 or 4, meaning that the center point is has either one or two points on either side.

- A median filter computes the median (center value in a ranked order of the values in the buffer) and replaces the sample at time index $k$ with that median of the $N+1$ points in the buffer. The median of the group usually will remove the outlier in this nonlinear way, but when we do not have an outlier it can lower the quality of the data compared to linear filtering. (A mean is an unbiased estimate, while a median is not.)

- An outlier removal filter estimates the value at time index $k$ using the other $N$ points. If $N+1 = 3$, then the point at index $k$ is the average of the values at $k-1$ and $k+1$, a linear estimate. If the measured value at time $k$ is outside some predetermined bounds from the estimate, we replace the measurement by the estimate. For larger values of $N$, more sophisticated estimators can be used, but the basic idea is the same.

This section should be expanded, but we can make a couple of points right away:

- The longer filters provide more checking, but incur more delay. Our DSP friends don't care, but we in feedback should.
- A median filter is always on, always replacing. Requires a sort ($O(N^2)$) to get median. An outlier removal filter only replaces a value that is way out of bounds. Requires a simple fit ($O(N)$) and a few comparisons (essentially a case statement).

## 6.9 Some Useful Filters

This section will give some classic filters and explain how to put them into computer code.

| Filter | Computer Version |
|---|---|
| Integrate and dump | FIR Averager |
| First Order Analog Low Pass | First Order Digital Low Pass |
| Second Order Analog Low Pass | Second Order Digital Low Pass |
| Biquad Notch | Biquad Notch |
| Second Order Analog High Pass | Second Order Digital High Pass |

Table 6.4: Some useful filters.

The first order filters were already presented. For this section, I will focus on the second order responses that are useful as filters, and in particular how to generate useful second order digital filters from the analog values.

$$F(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}, \tag{6.27}$$

$$\begin{aligned} F(s) &= b_0 \tilde{F}(s) \\ &= b_0 \left( \frac{s^2 + \tilde{b}_1 s + \tilde{b}_2}{s^2 + a_1 s + a_2} \right). \end{aligned} \tag{6.28}$$

Alternately, we can define these coefficients in terms of things that mean something to dynamic response. If we consider the second order polynomials of the numerator and denominator to have possibly complex roots, then we can define the parameters that would make things oscillate. Considering only the denominator polynomial,

$$D(s) = s^2 + 2\zeta\omega s + \omega^2 \tag{6.29}$$

we have the roots at

$$s = \frac{-2\zeta\omega \pm \sqrt{4\zeta^2\omega^2 - 4\omega^2}}{2} \tag{6.30}$$

$$= \frac{-2\zeta\omega \pm 2\omega\sqrt{\zeta^2 - 1}}{2} \tag{6.31}$$

$$= -\zeta\omega \pm \omega\sqrt{\zeta^2 - 1} \tag{6.32}$$

If $\zeta > 1$ then the roots are real and distinct. If $\zeta < 1$ then the roots are a complex pair. If $\zeta = 1$ then the roots are real and equal. One of the major advantages of using a second order filter over a pair of first order filters strung together is that we can use the complex pair to get much, much sharper responses than we could from a real pair.

| | |
|---|---|
| $f_N$ | Center frequency of numerator (Hz) |
| $\omega_N = 2\pi f_N$ | Center frequency of numerator (rad/s) |
| $Q_N$ | Quality factor of numerator |
| $\zeta_N = \frac{1}{2Q_N}$ | Damping factor of numerator |
| $f_D$ | Center frequency of denominator (Hz) |
| $\omega_D = 2\pi f_D$ | Center frequency of denominator (rad/s) |
| $Q_D$ | Quality factor of denominator |
| $\zeta_D = \frac{1}{2Q_D}$ | Damping factor of denominator |
| $K_F$ | Filter gain |

Table 6.5: Parameters that define a second order filter.

Okay, so if we have numerator and denominator, we need to do the bookkeeping and label our quantities with $N$ and $D$ subscripts so that we can keep them straight:

$$F(s) = K_F \tilde{F}(s) = K_F \frac{s^2 + 2\zeta_N\omega_N s + \omega_N^2}{s^2 + 2\zeta_D\omega_D s + \omega_D^2} \tag{6.33}$$

The numerator and denominator are second order and thus we easily find the roots via the quadratic equation. By adjusting $K_F$ we can set the gain at a single point. For example, it is common to set the gain to a steady input, the DC gain, to $1$. For this we set

$$K_F = \frac{\omega_D^2}{\omega_N^2}, \tag{6.34}$$

which means that when the input signal is constant/steady/unchanging ($s = 0$) then we get $F(s) = 1$.

### 6.9.1 Second Order Low Pass Filter

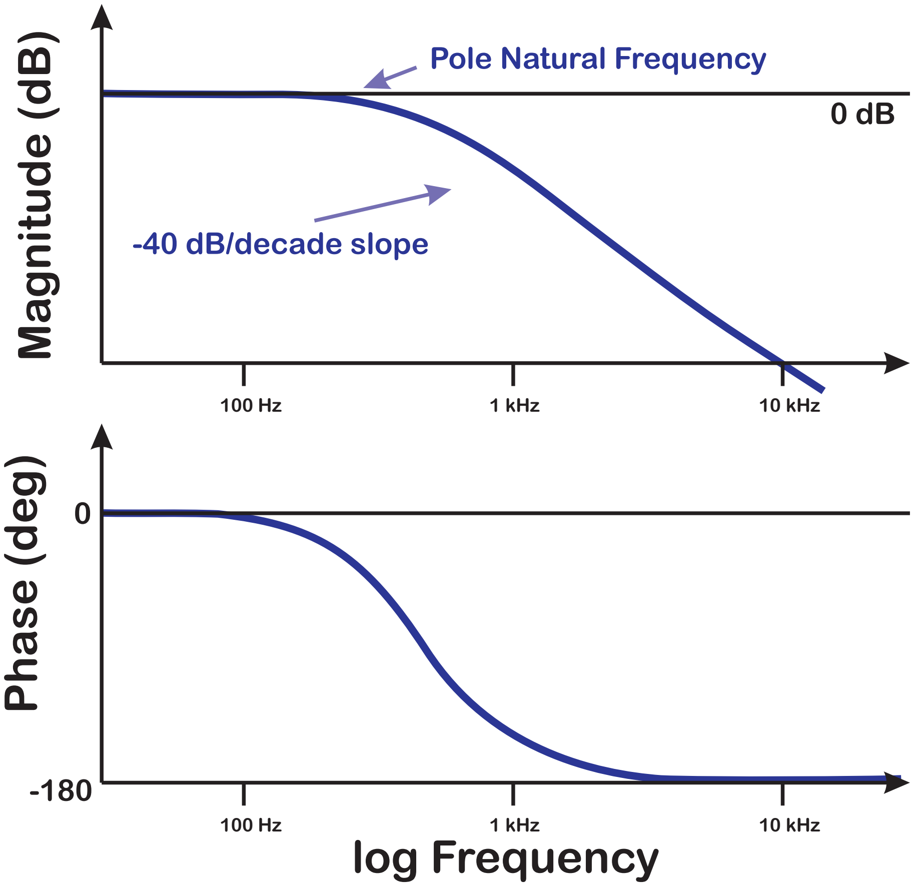


Figure 6.8: Schematic response of a low pass filter implemented with a simple second order section.

Low pass filters can and do show up as both analog and digital filters. We generally think of low pass filters as being constructed with analog circuitry, but most physical systems in the universe exhibit low pass behavior. Thus, the low pass filter is a great metaphor for understanding the behavior of most real world systems. Eventually, most physical things stop responding to higher frequency inputs. As Professor Bob Grey of Stanford used to say, "Pure white noise is an abstraction. It means that there is a possibility of things moving infinitely fast over infinitesimal intervals, which requires the release of infinite energy and makes the universe blow up." In the same way, infinite bandwidth is also an abstraction and so everything either blows up or has finite bandwidth.

If we take a special case of Equation 6.27 and set $b_0 = b_1 = 0$ and $b_2 = \omega_D^2$. This filter is in the form of a simple second order resonance,

$$F(s) = \frac{\omega_D^2}{s^2 + 2\zeta_D\omega_D s + \omega_D^2} \tag{6.35}$$

and is shown schematically in Figure 6.8. Equation 6.35 can describe a lot of physical phenomena, but in order to describe a good filter, we want to adjust $\zeta_D$ so that it neither provides too much damping (resulting in a larger negative phase earlier and a less distinct corner in magnitude) nor too little damping (resulting in ringing). Generally, this means that $0.5 \leq \zeta_D \leq 1$. With $\zeta_D = 1$ we arrive at critical damping, where the roots of the denominator are real and equal. However, this is often a bit overdamped from the perspective of filtering and we can afford to drop the damping to $\zeta_D = \frac{\sqrt{2}}{2} \approx 0.707$. Even damping factors of $0.6$ result in very little peaking of the filter response. The "corner frequency" is set by $f_D$ where $\omega_D = 2\pi f_D$. We need $\omega_D$ for the equations, but it is easier to think in terms of Hertz (cycles/second) rather than radians/second, so I like to start with $f_D$ and compute the rest.

This is an effective low pass filter. After the corner frequency the gain drops off at $-40dB/decade$, or in non-logarithmic terms, for every factor of 10 in frequency that one goes out past the corner frequency, the signal is attenuated by a factor of 100 (which corresponds to -40 dB). The other thing to note is that this filter is far more efficient in filtering than a pair of first order low pass filters, provided that we constrain $\zeta_D$ as above. The best a cascade of two first order filters can do is to have the poles be real and matched (so the same as our $\zeta_D = 1$ critically damped case. However, by allowing for complex conjugate roots, we get a sharper filter response with minimal peaking. The key to this is intelligently picking $\zeta_D$.

If we go out even further in frequency, the attenuation is truly impressive. However, we have taken a phase hit to do this and at high frequency our phase is $-180°$. We can create a filter with programmed attenuation, but where the phase returns to $0$ by having a numerator with two zeros as well, and this is described in Section 6.9.3. However, before we go there, it's worth taking a look at the dual of our two pole filter, which is the two zero filter.

### 6.9.2 Second Order High Pass Filter

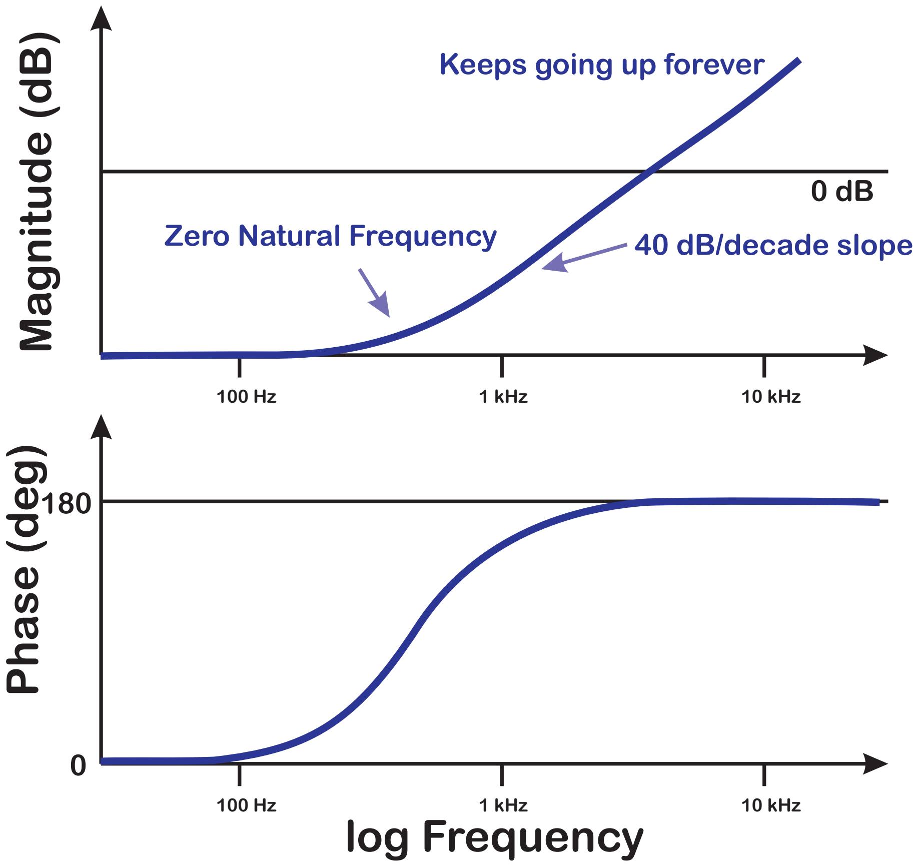


Figure 6.9: Schematic response of a high pass filter implemented with a simple second order section and all zeros. This is not physically realizable.

If we take a special case of Equation 6.27 and set $a_0 = a_1 = 0$ and $a_2 = \omega_N^2$. This filter a weird kind of filter, which could never be implemented in the real world

$$F(s) = \frac{s^2 + 2\zeta_N \omega_N s + \omega_N^2}{\omega_N^2} \tag{6.36}$$

and is shown schematically in Figure 6.9.

Equation 6.36 describes this “all zeros analog filter” which isn’t really something we could build because it has pure differentiation in it with no frequency limits. We see that the gain curve in Figure 6.9 keeps rising with frequency, so this corresponds to having infinite gain at infinite frequency, which is generally bad unless you want another Big Bang. Still, this shows us conceptual form, that we could, at least for some low frequencies, increase the gain and give positive phase (which is really akin to predicting the future). In the physical world, though, we need to roll this off at some frequency. The other detail to remember is that we can generate all zeros filters in the digital domain. This is what FIR filters are. Remember, though, that FIR filters actually do have poles, they are just at $z = 0$.

As with the low pass, we would want to adjust $\zeta_N$ so that it neither provides too much damping (resulting in a larger positive phase earlier and a less distinct corner in magnitude) nor too little damping (resulting in ringing). Generally, this means that $0.5 \leq \zeta_N \leq 1$. The math is pretty much analogous to the filter in Section 6.9.1, except that it doesn't describe something physical.

### 6.9.3 Biquad Low Pass Filter

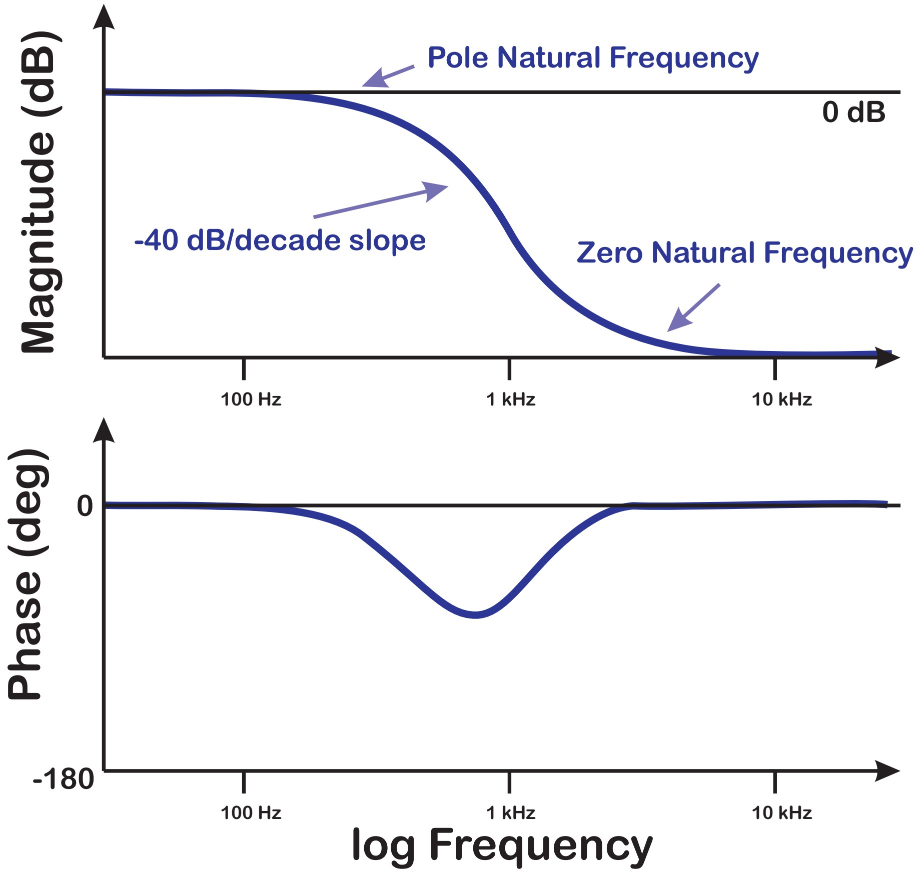


Figure 6.10: Schematic response of a low pass filter implemented with a biquad.

For a biquad low pass filter, we want $f_D < f_N$ or from Equation 6.33, $\omega_D < \omega_N$. Typically, we value flat response with no peaking in a low pass filter, so $\zeta_D$ and $\zeta_N$ are generally greater than $0.5$ or $0.7$. Because the filtering effect goes away when we are beyond $f_N$ the amount of filtering we get is determined by the separation between $f_D$ and $f_N$. Of course, the lower $f_D$ the less bandwidth in the filter.

### 6.9.4 Biquad High Pass Filter

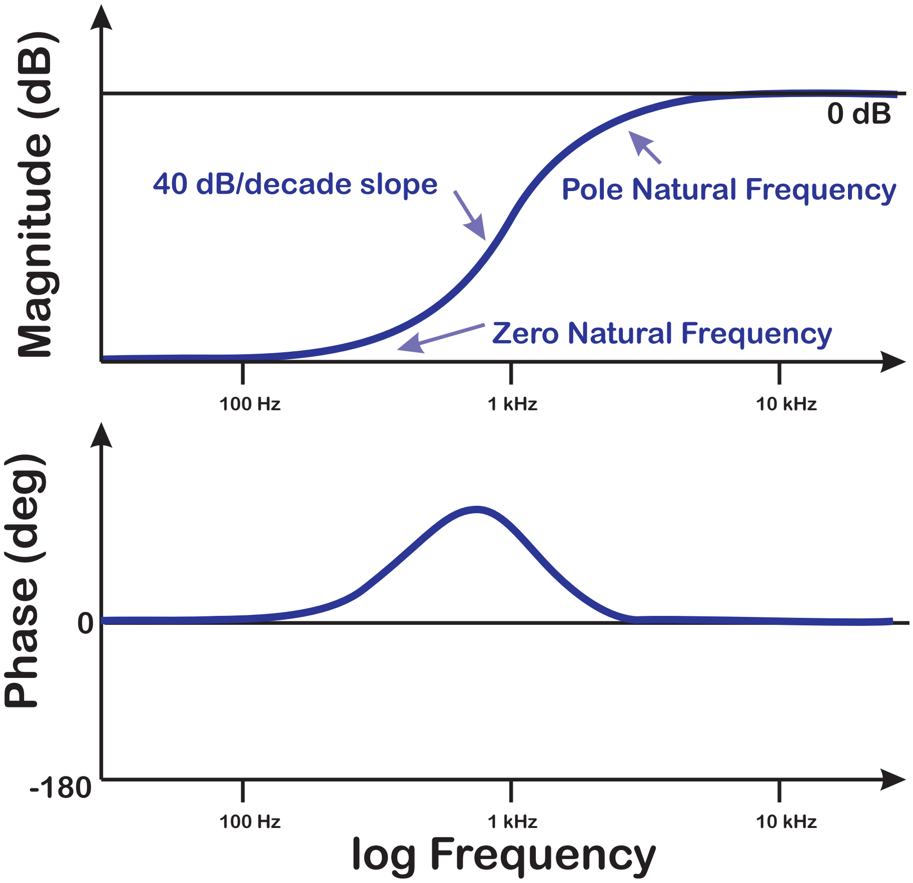


Figure 6.11: Schematic response of a high pass filter implemented with a biquad.

The biquad high pass filter is the opposite of the biquad low pass filter. In this case, $f_N < f_D$ or from Equation 6.33, $\omega_N < \omega_D$. Typically, we value flat response with no peaking in a high pass filter, so $\zeta_D$ and $\zeta_N$ are generally greater than $0.5$ or $0.7$. Because the filtering effect goes away when we are beyond $f_D$ the amount of filtering we get is determined by the separation between $f_D$ and $f_N$. Of course, the higher $f_D$ the higher the frequency at which the filter's pass band applies.

### 6.9.5 Two Biquad Band Pass Filter

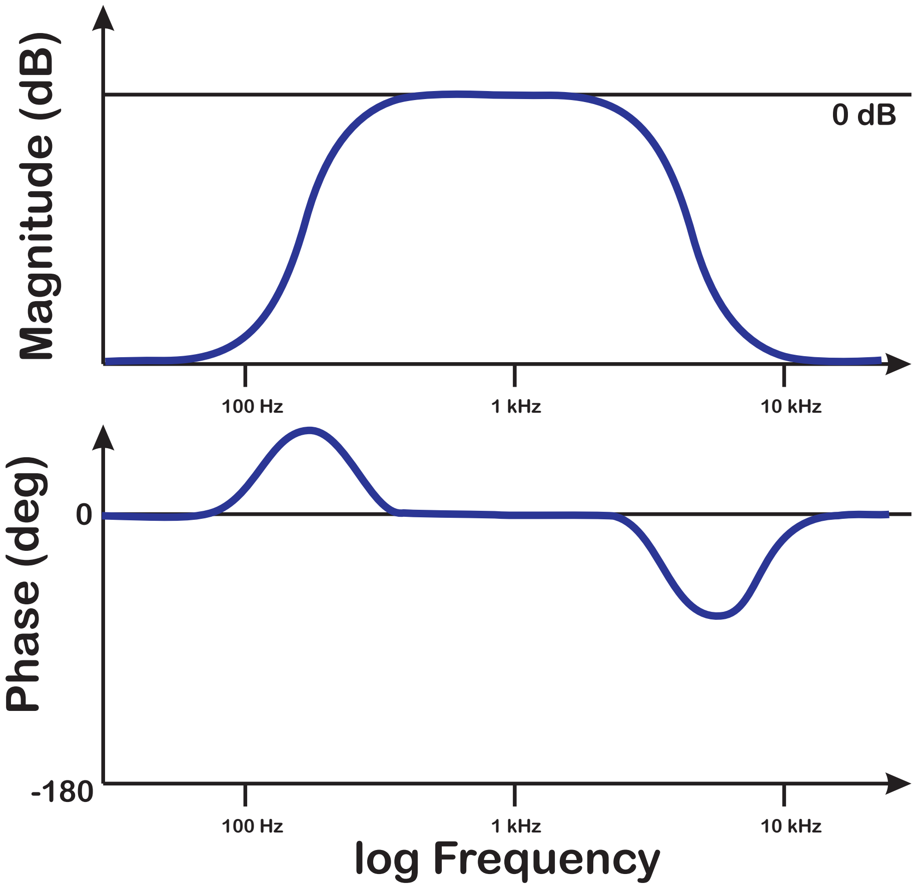


Figure 6.12: Schematic response of a band pass filter implemented with a pair of biquads, one low pass and one high pass.

The two biquad bandpass filter is implemented with a high pass filter cascaded with a low pass filter.

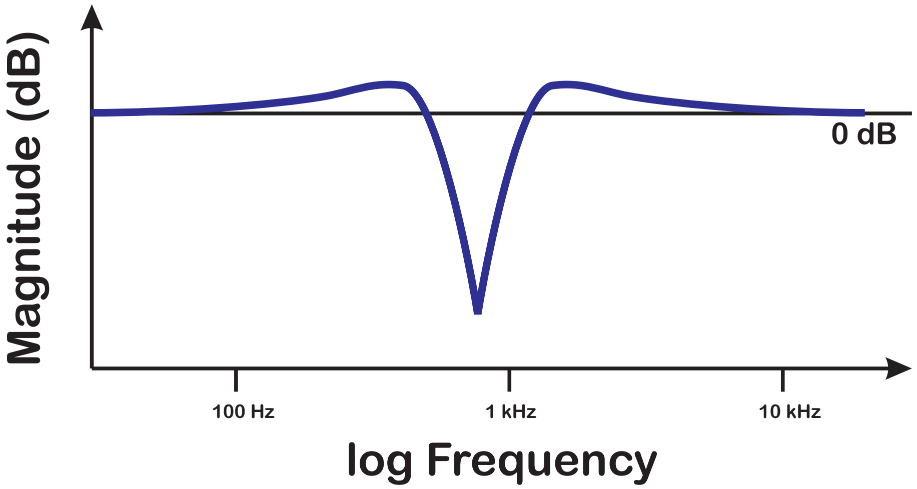


Figure 6.13: Schematic response of a notch filter implemented with a single biquad. The resonant frequencies of numerator and denominator are the same, but the numerator has a much smaller damping factor, resulting in a strong dip at the notch frequency. Is there a reason to include phase?

### 6.9.6 Biquad Notch Filter

There are times when we want to kill a single frequency of signal without affecting much around it. This is where a notch filter becomes useful. This implementation uses the same frequencies in numerator and denominator $f_D = f_N$, but $\zeta_N << \zeta_D$. This results in a response similar to Figure 6.13. By raising both $\zeta_N$ and $\zeta_D$ in proportion, we can get the same filtering effect but over a much narrower span. This is useful when the desired notch frequency is known very well. If it moves, or if there is any uncertainty, the notch must be made wider, which results in larger effects in nearby frequencies.

### 6.9.7 Biquad Peak Filter

More rare than the notch are the times when we specifically want to highly amplify one frequency, but they exist. This implementation also uses the same frequencies in numerator and denominator $f_D = f_N$, but $\zeta_N >> \zeta_D$. This results in a response similar to Figure 6.14. By raising both $\zeta_N$ and $\zeta_D$ in proportion, we can get the same filtering effect but over a much narrower span. This is useful when the desired peak frequency is known very well. If it moves, or if there is any uncertainty, the notch must be made wider, which results in larger effects in nearby frequencies.

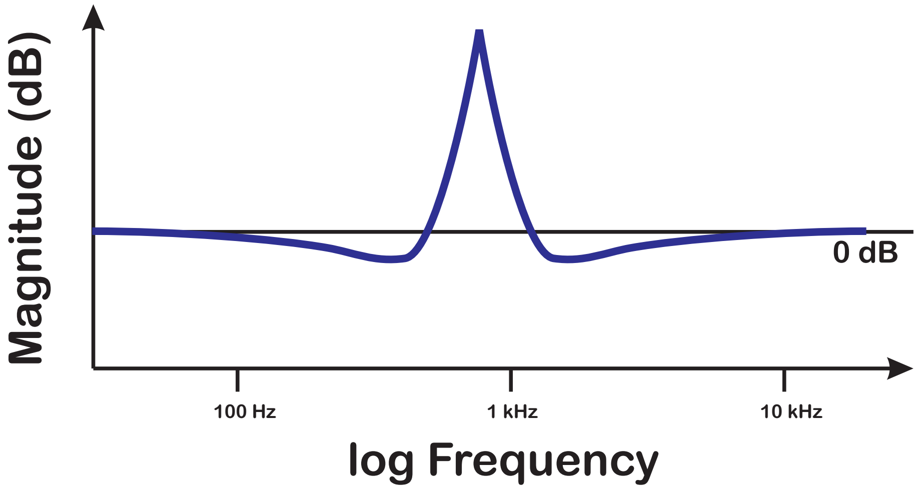


Figure 6.14: Schematic response of a peak filter implemented with a single biquad. The resonant frequencies of numerator and denominator are the same, but the denominator has a much smaller damping factor, resulting in a strong rise at the peak frequency. Is there a reason to include phase?

## 6.10 Filter Summary

A simple but effective filter design procedure can now be summarized here:

- Pick out a physical response that we want to adjust.
- Pick the correct filter form.
- Pick the analog (physical world) coefficients.
- Convert to digital (computer) form. (Section 3.6)
- Drop into filter blocks.
- Bazinga!

Understanding these basic shapes (and a few others), we are now in a position to drop them into a controller to reshape portions of the open loop frequency response. Of course, our standard polynomial design methods want us to multiply them all out, which will cause us to lose a physical understanding of what is going on. Instead, of that Brand X, I'm going to suggest we learn about The Multinotch as a vehicle for housing these pieces.

## 6.11 The Multinotch

The Multinotch was created specifically for loop shaping on lightly damped, high frequency dynamics in a system that could not tolerate much time delay. The original control system was in programmable logic (FPGA) and the math was fixed point. The idea was to have both the positive numerical stability properties of biquad chains without giving up on the ability to do precalculation. The resulting system had a lot of other positive attributes. The filter blocks were compact, with roughly the same number of free parameters as polynomial form or canonical realizations. The representational accuracy remained high in converting from continuous to discrete models and from floating point to fixed point – although I had to put in a bunch of tweaks to take care of some extreme cases. Under duress, I had to find a state space form that could be used on lightly damped mechatronic systems, and so I converted The Multinotch to state space with the Biquad State Space (BSS) form (Section 9.15). That had its own benefits that I couldn't have anticipated. The bookkeeping in these forms is tedious, but they have a lot of advantages that seem intuitively obvious the first time one runs through them. The next few sections will go through some of the papers published on this, hopefully pulling the work together in an understandable way.

Control of lightly damped mechatronic systems is often accomplished in practice with a PID-like controller in series with a filter to limit the effects of high frequency resonances as diagrammed in Figure 6.15. The high frequency filtering is often limited by an inability to precisely match multiple lightly damped resonances with a digital filter, and by the extra computational delay of the such filters. The multinotch is a filter topology that addresses these issues, allowing for precise matches to many lightly damped resonances and anti-resonances, while maintaining a small and fixed computational delay.

Two issues that must be addressed when using digital filtering in a feedback loop are numerical issues and computational latency. Numerical issues typically come from the implementation of algorithms and filters in finite word length arithmetic. In particular, high order polynomial filters lose both physical intuition and numerical sensitivity because of how physical parameters get compressed into coefficients.

The phase lag due to latency in a feedback loop erodes stability and performance characteristics. For a causal filter, the latency is in part determined by the filter length. In particular, for a symmetric tap finite impulse response (FIR) filter with $N$ taps and a sample period of $T_S$, the average latency through the filter will be $\frac{N}{2}T_S$ if $N$ is even, and $\frac{N-1}{2}T_S$ if $N$ is odd. Infinite Impulse Response (IIR) filters are

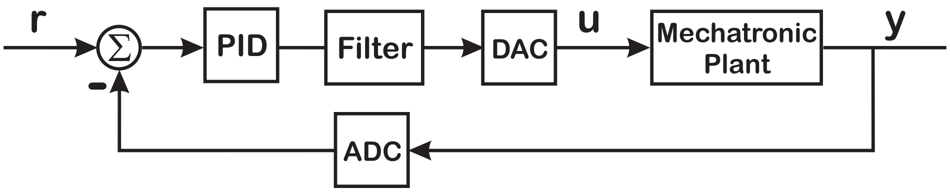


Figure 6.15: A practical digital control loop for a mechatronic system. The digital controller is often implemented as a PID like controller in series with filtering to lower the effect of high frequency resonances.

usually favored over FIR filters in feedback loops, as they can achieve similar bandwidth shaping with considerably smaller average delay (find reference).

The second source of latency is simply the time required to compute the filter equations between the time that a new sample comes into the filter and the filter produces its output. This delay is generally - but not necessarily - less than one sample period, but it is complicated by the fact that it can change with the number of taps in a filter. That is, a second order filter obviously takes fewer computation steps than a tenth order filter. This variable latency can cause unexpected problems with the control loop. A standard technique to minimize this variable latency is to compute everything that does not depend upon the most recent sample ahead of time in a precalculation [16], diagrammed in Figure 6.1. Once the most recent sample arrives, the last few calculations are performed and the filter output is produced. This has the benefit of not only minimizing the computational latency, but also of making it independent of filter length.

However, doing a precalculation usually involves using a form of the digital filter that can have numerical issues with finite word length. The structure of the multinotch allows it to minimize the computational latency using precalculation, while still preserving the numerical properties needed for finite word length arithmetic.

This helps mitigate a third common, but hard to quantify source of latency, which is overly conservative filter design because of the designers lack of confidence in the ability of the implemented digital filter to perform as desired (cite some disk drive, mechatronic papers). This lack of confidence means that rather than implementing a sharp notch filter, designers opt for more conservative low pass filters to more indiscriminately damp out any dynamics at higher frequency. The filters are “low Q”, that is overly broad compared to the actual dynamics, and therefore more robust to parameter changes caused by finite wordlength effects. By the same token, broadening the width of the filter means that the negative phase effects reach far beyond the dynamics they were meant to cancel, especially affecting the lower frequency dynamics where the nominal control action is being performed. These filters are commonly employed in mechatronic systems, but the phase lag due to such filters puts a severe limit on the achievable closed-loop bandwidth. The use of precisely matched notch and resonant filters to equalize a dynamic response can be used to achieve significantly higher bandwidth [19].

## 6.12 Digital Filter Equations and Biquads

The problem with polynomial filters is that the elements that generate discrete filter coefficients can become extremely sensitive, particularly when they relate to high Q elements near the unit circle. Essentially, as the coefficients of the simple poles and zeros or complex pairs get convolved together to form the polynomial coefficients, small physical parameter changes get distributed across many polynomial coefficients. Not only is physical intuition completely lost, but relatively large changes in a physical value may be only a few bits of any one polynomial coefficient value. This is particularly true with fixed point arithmetic used in many DSP and FPGA (Field Programmable Gate Array) implementations. Thus, what we would like to do is implement the filter of (6.2) as a series of second order filters, known as biquads, but still maintain the ability to do final calculations in the form of (6.4) [169].

## 6.13 Biquads

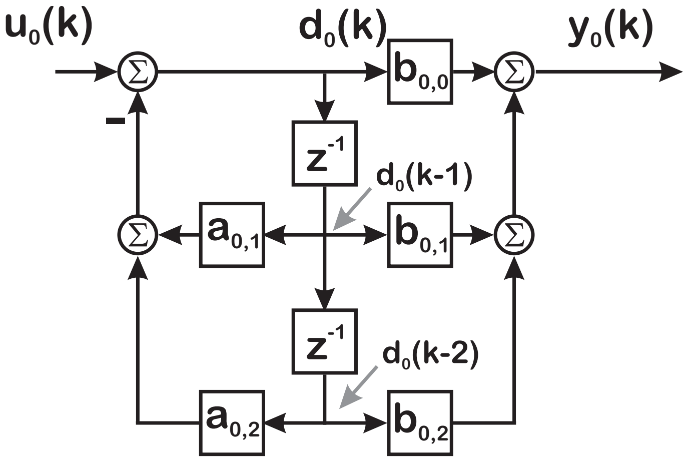


Figure 6.16: A second order polynomial filter in Direct Form II, known as a digital biquad filter.

Biquads are second order polynomial filter sections that typically are numerically well behaved. For example, if we implement a tenth order filter using five biquads, coefficient quantization issues that we might get with a tenth order polynomial filter are localized to each biquad and therefore limited. The latter might be something that we would do if we had a floating point DSP chip, but would not be advisable for fixed point calculations such as the ones that we do on the FPGA. Furthermore, assigning the poles and zeros of the filter so that the biquad poles and zeros are close to each other in the $z$ plane implies that the numerator and denominator will tend to compensate for each other so that at frequencies far from those of the biquad, the section has a minimal effect on the response. So, the biquad of Figure 6.16 would look like:

$$\frac{Y_0(z)}{U_0(z)} = N_0(z) = \frac{b_{0,0} + b_{0,1}z^{-1} + b_{0,2}z^{-2}}{1 + a_{0,1}z^{-1} + a_{0,2}z^{-2}} \tag{6.37}$$

which gets implemented in the time domain as:

$$\begin{aligned} y_0(k) &= -a_{0,1}y_0(k-1) - a_{0,2}y_0(k-2) + b_{0,0}u_0(k) \\ &\quad +b_{0,1}u_0(k-1) + b_{0,2}u_0(k-2) \end{aligned} \tag{6.38}$$

It turns out that it is easier to implement this using the delay format [169] which resembles a controller canonical form [171] in control or a direct form II IIR filter [167, 172, 173]:

$$d_0(k) = -a_{0,1}d_0(k-1) - a_{0,2}d_0(k-2) + u_0(k) \tag{6.39}$$

$$y_0(k) = b_{0,0}d_0(k) + b_{0,1}d_0(k-1) + b_{0,2}d_0(k-2) \tag{6.40}$$

Biquads are nice because the growth in values can be limited by the short nature of the filter. Thus, finite word length problems are minimized as the sums from the numerator and denominator can balance each other out for a well designed filter [169].

As with (6.3), only one part of the filter actually depends upon the current input, $u(k)$, and that is the calculation in (6.38). Likewise, in (6.39) only one part of the sum is due to $u(k)$, and there is only one multiply involving $d(k)$ in (6.40). Everything else can be calculated ahead of time. So, we can generate the precalculation for this form of the biquad as:

$$d_0(k) = \text{prec}_{0,1}(k) + u_0(k) \text{ and} \tag{6.41}$$

$$y_0(k) = b_{0,0}d_0(k) + \text{prec}_{0,2}(k) \text{ where} \tag{6.42}$$

$$\text{prec}_{0,1}(k) = -a_{0,1}d_0(k-1) - a_{0,2}d_0(k-2) \text{ and} \tag{6.43}$$

$$\text{prec}_{0,2}(k) = b_{0,1}d_0(k-1) + b_{0,2}d_0(k-2). \tag{6.44}$$

As before, we see that $\text{prec}_{0,1}(k)$ and $\text{prec}_{0,2}(k)$ depend only on prior values of the filter input and output and can be computed ahead of time.

## 6.14 Higher Order Filters as a Series of Biquads

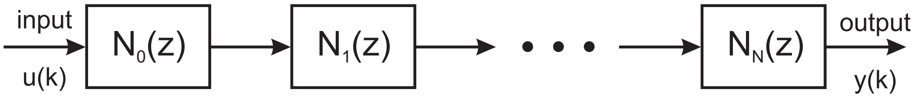


Figure 6.17: Series connection of multiple filters. In our case, these filters are each a second order digital transfer function (biquad).

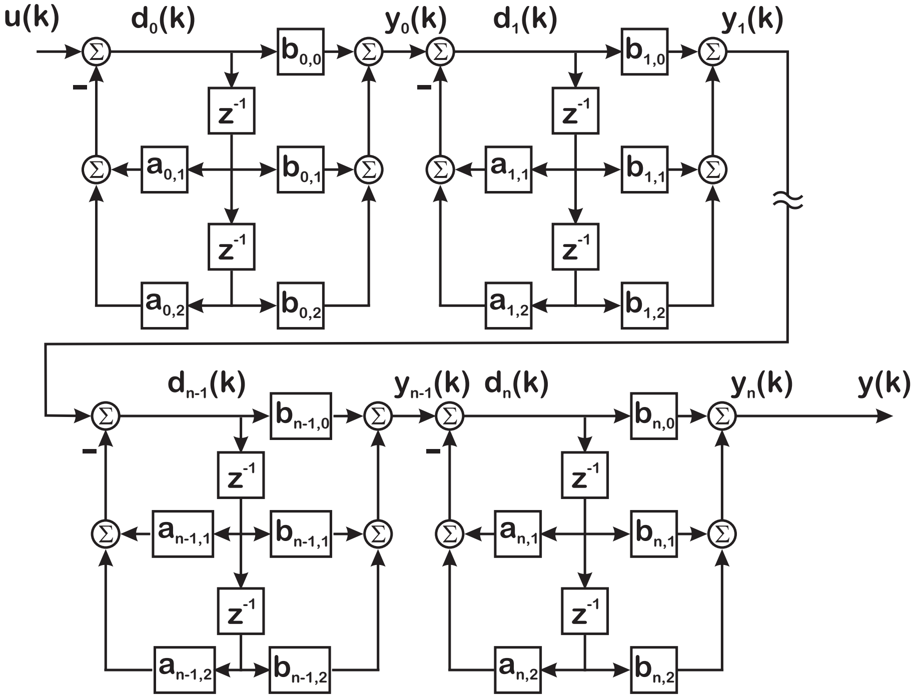


Figure 6.18: An expanded realization view of the serial biquad chain from Figure 6.17.

If we wish to stack individual blocks of biquads together to filter out multiple resonances and/or anti-resonances, we can simply do a serial cascade of blocks as shown in Figure 6.17, which could be represented internally as shown in Figure 6.18. The problem here is that in this form it is not possible to do precalculation of anything but the first block, because the downstream blocks all depend upon the output of the previous blocks. Consider the example of the two biquad case (Figure 6.19), where in we augment Equations 6.37–6.40 with:

$$\frac{Y_1(z)}{U_1(z)} = N_1(z) = \frac{b_{1,0} + b_{1,1}z^{-1} + b_{1,2}z^{-2}}{1 + a_{1,1}z^{-1} + a_{1,2}z^{-2}} \tag{6.45}$$

which gets implemented in the time domain as:

$$y_1(k) \quad = \quad -a_{1,1}y_1(k-1) - a_{1,2}y_1(k-2) + b_{1,0}u_1(k)$$

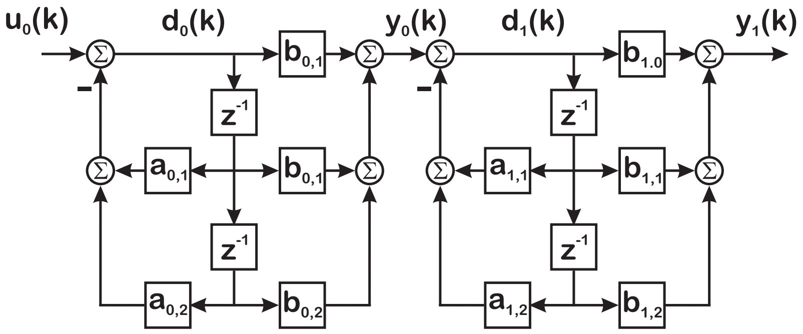


Figure 6.19: A two biquad cascade.

$$+b_{1,1}u_1(k-1)+b_{1,2}u_1(k-2) \tag{6.46}$$

The delay forms are:

$$d_1(k) = -a_{1,1}d_1(k-1) - a_{1,2}d_1(k-2) + u_1(k) \tag{6.47}$$

$$y_1(k) = b_{1,0}d_1(k) + b_{1,1}d_1(k-1) + b_{1,2}d_1(k-2) \tag{6.48}$$

with precalcs implemented as:

$$d_1(k) = \text{prec}_{1,1}(k) + u_1(k) \text{ and} \tag{6.49}$$

$$y_1(k) = b_{1,0}d_1(k) + \text{prec}_{1,2}(k) \text{ where} \tag{6.50}$$

$$\begin{aligned}
\text{prec}_{1,1}(k) &= -a_{1,1}d_1(k-1) - a_{1,2}d_1(k-2) \text{ and} && (6.51)\\
\text{prec}_{1,2}(k) &= b_{1,1}d_1(k-1) + b_{1,2}d_1(k-2). && (6.52)
\end{aligned}$$

Furthermore, they are stitched in series by letting the overall input go into biquad 0, the overall output come from biquad 1 and using the output of biquad 0 as the input to biquad 1. That is

$$u_0(k) = u(k), y(k) = y_1(k), \text{ and } u_1(k) = y_0(k). \tag{6.53}$$

We see that $y_1(k)$ depends on $b_{1,0}d_1(k)$, which depends on $u_1(k) = y_0(k)$, which depends on $b_{0,0}d_0(k)$, which depends upon $u_0(k)$. This is obvious if we try to chain these together:

$$\begin{aligned}
d_1(k) &= prec_{1,1}(k) + u_1(k) = prec_{1,1}(k) + y_0(k) && (6.54)\\
&= prec_{1,1}(k) + b_{0,0}d_0(k) + prec_{0,2}(k) && (6.55)\\
&= prec_{1,1}(k) + b_{0,0}\left[prec_{0,1}(k) + u_0(k)\right] + prec_{0,2}(k) && (6.56)\\
d_1(k) &= prec_{1,1}(k) + prec_{0,2}(k) + b_{0,0}\left[prec_{0,1}(k)\right] + b_{0,0}u_0(k) && (6.57)
\end{aligned}$$

The output, $y_1(k)$ is obtained via:

$$\begin{aligned} y_1(k) &= b_{1,0}d_1(k) + prec_{1,2}(k) && (6.58)\\ &= prec_{1,2}(k) + b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k) + b_{0,0}prec_{0,1}(k) + b_{0,0}u_0(k)\right] && (6.59)\\ y_1(k) &= prec_{1,2}(k) + b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right] + b_{1,0}b_{0,0}prec_{0,1}(k) + b_{1,0}b_{0,0}u_0(k) && (6.60)\end{aligned}$$

We can add in a third biquad:

$$\begin{aligned} d_2(k) &= prec_{2,1}(k) + u_2(k) = prec_{2,1}(k) + y_1(k) && (6.61)\\ &= prec_{2,1}(k) + prec_{1,2}(k) + b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right]\\ &\quad +b_{1,0}b_{0,0}\left[prec_{0,1}(k)\right] + b_{1,0}b_{0,0}u(k) && (6.62)\end{aligned}$$

and likewise we get $y_2(k)$ via

$$\begin{aligned} y_2(k) &= b_{2,0}d_2(k) + prec_{2,2}(k) && (6.63)\\ y_2(k) &= prec_{2,2}(k) + b_{2,0}\left[prec_{2,1}(k) + prec_{1,2}(k)\right]\\ &\quad +b_{2,0}b_{1,0}\left[prec_{1,1}(k) + prec_{0,2}(k)\right] && (6.64)\\ &\quad +b_{2,0}b_{1,0}b_{0,0}prec_{0,1}(k) + b_{2,0}b_{1,0}b_{0,0}u_0(k).\end{aligned}$$

Several things start to emerge about this recursion. First, there is a definite pattern. Next, we see that we can still precompute our original precalc sections, but each one is scaled by some concatenation of $b_{i,0}$ terms to update the most recent $d_i(k)$ and $y_i(k)$ values. This means that the more biquad sections we add, the more multiplications are required to generate the current $d_i$ and $y_i$ terms once the $u(k)$ input is available. The further down the chain we go, the more multiplies need to be computed once the current For the $i^{th}$ biquad, counting from $0$, $d_i(k)$ requires $i$ multiplies and $y_i(k)$ requires $i+1$ multiples (after $u(k)$ is available). This means that the computational latency grows with filter length.

## 6.15 An Improved Structure

A look at Equations 6.62 and 6.64 reveals that the problem lies with the $b_{i,0}$ terms. If these terms were only $1$, then our problem would be solved. Remembering third grade mathematics, we note that multiplication is associative, and thus we can factor out the $b_{i,0}$ terms from our filters, ending up with an equivalent cascade of biquads:

$$\begin{aligned} \frac{Y(z)}{U(z)} &= N_n(z)N_{n-1}(z)\cdots N_1(z)N_0(z) && (6.65)\\ &= b_{n,0}\cdots b_{1,0}b_{0,0}\tilde{N}_n(z)\cdots\tilde{N}_1(z)\tilde{N}_0(z) && (6.66)\end{aligned}$$

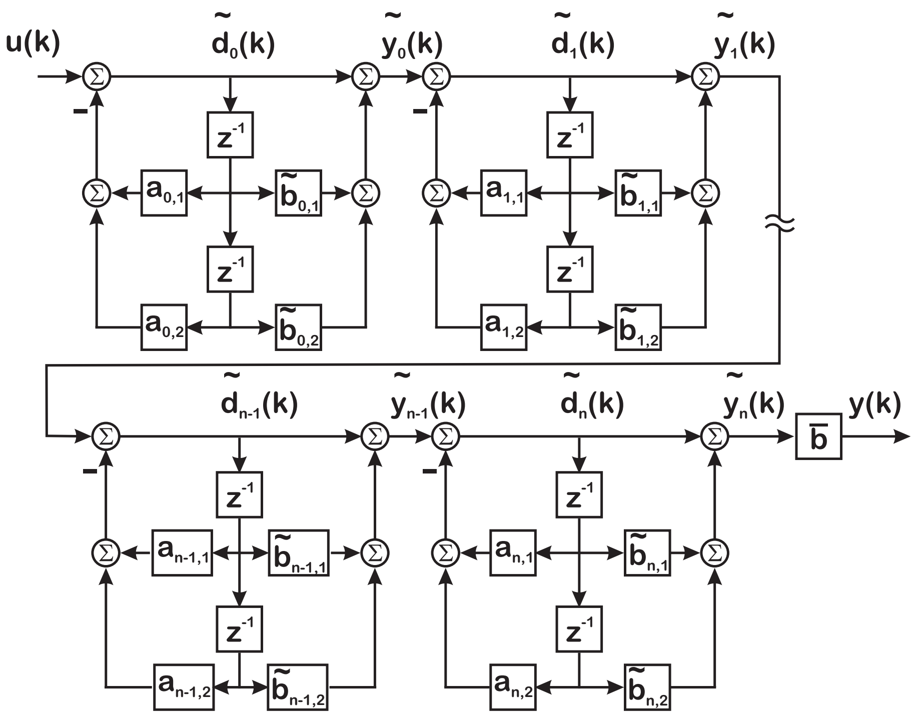


Figure 6.20: The updated biquad cascade, with factored out $b_0$ terms.

where

$$\tilde{N}_i(z) = \frac{1 + \tilde{b}_{i,1}z^{-1} + \tilde{b}_{i,2}z^{-2}}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}} \tag{6.67}$$

$$\tilde{b}_{i,1} = \frac{b_{i,1}}{b_{i,0}}, \text{ and } \tilde{b}_{i,2} = \frac{b_{i,2}}{b_{i,0}}. \tag{6.68}$$

The direct feedthrough gains are concatenated together as:

$$\bar{b} = b_{n,0}b_{n-1,0}\cdots b_{1,0}b_{0,0}. \tag{6.69}$$

This structure, shown in Figure 6.20 has the advantage that:

- It retains the biquad form with the same poles and zeros as the original filter of Figure 6.18.
- $\tilde{d}_i(k)$ and $\tilde{y}_i(k)$ can be computed from summing precalculated terms with the current input, $u(k)$.
- Once $u(k)$ is available, the computation of $y(k)$ involves two additions of precalculated terms and one multiplication by $\bar{b}$. This means that the computational latency between the measurement input and the filter output is very small and independent of filter length.

- The coefficients of individual biquad sections are computed independently, retaining most of the physical intuition in the filter, even after discretization.
- Picking complex pole-zero pairs that are close to each other generally limits the signal growth in any biquad block, which is helpful for fixed point math.
- Such a structure can be implemented in such a way that different biquad sections can be turned on or off. In this case only the aggregate direct feedthrough gain, $\overline{b}$ needs to be adjusted when a biquad block is activated or deactivated in a real time controller.

Looking at the structure, we see that:

$$\tilde{d}_0(k) = D_{P,0}(k-1) + u(k), \tag{6.70}$$

$$\tilde{d}_i(k) = \sum_{j=0}^{i} D_{P,j}(k-1) + \sum_{j=0}^{i-1} F_{P,j}(k-1) + u(k), \tag{6.71}$$

for $i \geq 1$, and

$$\tilde{y}_i(k) = \sum_{j=0}^{i} D_{P,j}(k-1) + \sum_{j=0}^{i} F_{P,j}(k-1) + u(k), \tag{6.72}$$

for $i \geq 0$, where the delay precalculations are

$$D_{P,j}(k-1) = -a_{j,1}\tilde{d}_j(k-1) - a_{j,2}\tilde{d}_j(k-2) \tag{6.73}$$

and the output precalculations are

$$F_{P,j}(k-1) = \tilde{b}_{j,1}\tilde{d}_j(k-1) + \tilde{b}_{j,2}\tilde{d}_j(k-2). \tag{6.74}$$

Note that as the signal moves through biquad stages, sums get added to the precalculation. However, because these are all scaled the same, the sums that have already been computed can be reused. The final output, $y(k)$ is obtained from the last biquad output using $\overline{b}$ from Equation 6.69 as

$$y(k) = \overline{b}\tilde{y}_n(k). \tag{6.75}$$

## 6.16 Multinotch Filter Coefficients

Because this filter is a digital biquad, there are no excess poles or zeros. Furthermore, using this form where we have factored the $b_{i,0}$ gain out of each numerator means that all the biquads will have

| | |
|---|---|
| $f_{N,i}$ | Center frequency of numerator (Hz) |
| $\omega_{N,i}$ | Center frequency of numerator (rad/s) |
| $Q_{N,i}$ | Quality factor of numerator |
| $\zeta_{N,i} = \frac{1}{Q_{N,i}}$ | Damping factor of numerator |
| $f_{D,i}$ | Center frequency of denominator (Hz) |
| $\omega_{D,i}$ | Center frequency of denominator (rad/s) |
| $Q_{D,i}$ | Quality factor of denominator |
| $\zeta_{D,i} = \frac{1}{Q_{D,i}}$ | Damping factor of denominator |

Table 6.6: Physical coefficients used to specify a biquad section. Note that some of these are redundant, so that the choice of $\zeta$ versus $Q$ or $f$ versus $\omega$ is simply a user preference.

a uniform structure. Taking our design from an analog response of a ratio of a second order numerator and denominator, we can discretize the poles and zeros using matched pole-zero mapping [15]. This allows us to parametrize each biquad section using very physical parameters, as shown in Table 6.6. The factored out gain, $b_{i,0}$, can be used as is, or can be altered so that, for example, the DC gain of the biquad section will be $1$.

Assuming a complex pair of poles (or zeros), mapping via $z = e^{sT_S}$, and recombining the results yields some straightforward formulas. For $a_{i,2}$ and $\tilde{b}_{i,2}$ we have

$$a_{i,2} = e^{-2\omega_{D,i}T_S\zeta_{D,i}} \tag{6.76}$$

$$\tilde{b}_{i,2} = e^{-2\omega_{N,i}T_S\zeta_{N,i}} \tag{6.77}$$

Whether the poles (or zeros) are a complex pair depends upon $\left|\zeta_{D,i}\right|$ ($\left|\zeta_{N,i}\right|$). For $\left|\zeta_{D,i}\right| < 1$ we have a complex pair of poles and so

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}} \cos\left(\omega_{D,i}T_S\sqrt{1-\zeta_{D,i}^2}\right). \tag{6.78}$$

If $\left|\zeta_{N,i}\right| < 1$ we have a complex pair of zeros and so

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{N,i}} \cos\left(\omega_{N,i}T_S\sqrt{1-\zeta_{N,i}^2}\right). \tag{6.79}$$

While these two cases represent cases when the desired filters have very sharp peaks or notches (for example to equalize a response with very sharp notches or peaks), there are other possibilities. For example setting $\left|\zeta_{D,i}\right| = 1$ ($\left|\zeta_{N,i}\right| = 1$) means that the poles (zeros) are real and equal, so $a_{i,1}$ ($\tilde{b}_{i,1}$) are given by:

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}} \tag{6.80}$$

and

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{N,i}}. \tag{6.81}$$

Finally, if $\left|\zeta_{D,i}\right| > 1$ ($\left|\zeta_{N,i}\right| > 1$) means that the poles (zeros) are real and distinct, so $a_{i,1}$ ($\tilde{b}_{i,1}$) are given by using the cosh relation:

$$a_{i,1} = -2e^{-\omega_{D,i}T_S\zeta_{D,i}}\cosh\left(\omega_{D,i}T_S\sqrt{\zeta_{D,i}^2 - 1}\right) \tag{6.82}$$

and

$$\tilde{b}_{i,1} = -2e^{-\omega_{N,i}T_S\zeta_{D,i}}\cosh\left(\omega_{N,i}T_S\sqrt{\zeta_{N,i}^2 - 1}\right). \tag{6.83}$$

The entire conversion routine, which turns the physical parameters of Table 6.6 into discrete filter coefficients can be implemented in a short MATLAB or Octave function.

## 6.17 Multinotch Examples

| **Example 1** | | | | |
|---|---|---|---|---|
| **Biquad #** | $f_{N,n}$ **(Hz)** | $Q_n$ | $f_{N,d}$ **(Hz)** | $Q_d$ |
| 1 | 200 | 10 | 400 | 10 |
| 2 | 1000 | 5 | 2000 | 5 |
| **Example 2** | | | | |
| **Biquad #** | $f_{N,n}$ **(Hz)** | $Q_n$ | $f_{N,d}$ **(Hz)** | $Q_d$ |
| 1 | 200 | 10 | 400 | 10 |
| 2 | 1000 | 5 | 2000 | 5 |
| 3 | 10,000 | 10 | 20,000 | 10 |
| 4 | 8000 | 10 | 4000 | 10 |

Table 6.7: Filter parameters for both examples.

In order to compare filters, a set of filter parameters was chosen in the form of sets of analog biquad parameters, such as those in Table 6.7. These parameters were then translated into an analog polynomial filter, which was discretized via Matlab's c2d function. The coefficients of the digital filter were then scaled up by a quantization factor, say $2^{16} - 1$ for an s2.16 quantization. The scaled up coefficients were then fixed (fractional portion removed) and scaled down by the same quantization factor. Thus, floating point numbers were made to represent fixed point coefficients. Frequency responses were computed for the analog, digital polynomial, and various quantized digital polynomial filters.

To compare against the multinotch, the same analog biquad parameters were converted into individual digital biquads as described in Section 6.16. Since the biquads act serially on data, the frequency responses of each digital biquad was multiplied with those that were previously computed to form the overall response of the filter. In the case where quantization was applied, the individual biquad parameters were scaled up by the quantization factor, fixed, and scaled back down. As the direct feedthrough gain is applied afterwards, this was separately scaled up, fixed, and scaled down to compute the individual quantized biquad complex responses. Again, these responses were multiplied to that of the previous sections to give an overall biquad frequency response.

The filter parameters were set according to Table 6.7. Both examples use a sample rate of 100 kHz for discretization. The results for the first example are in Figure 6.21. In this simple fourth order filter, the effects of quantization are only evident for the polynomial digital filter quantized at s2.16, although we can see that the discretized polynomial filter never fully matches the analog response.

In the second example, there are far more biquads and they have higher Q values. The results on the left of Figure 6.22 show that the polynomial digital filter fails to match the analog response even without quantization. With quantization, only the highest number of bits, s2.30, matches the unquantized digital polynomial filter. On the other hand, the multinotch on the right side of Figure 6.22 match the analog response, even with the coarsest s2.16 quantization.

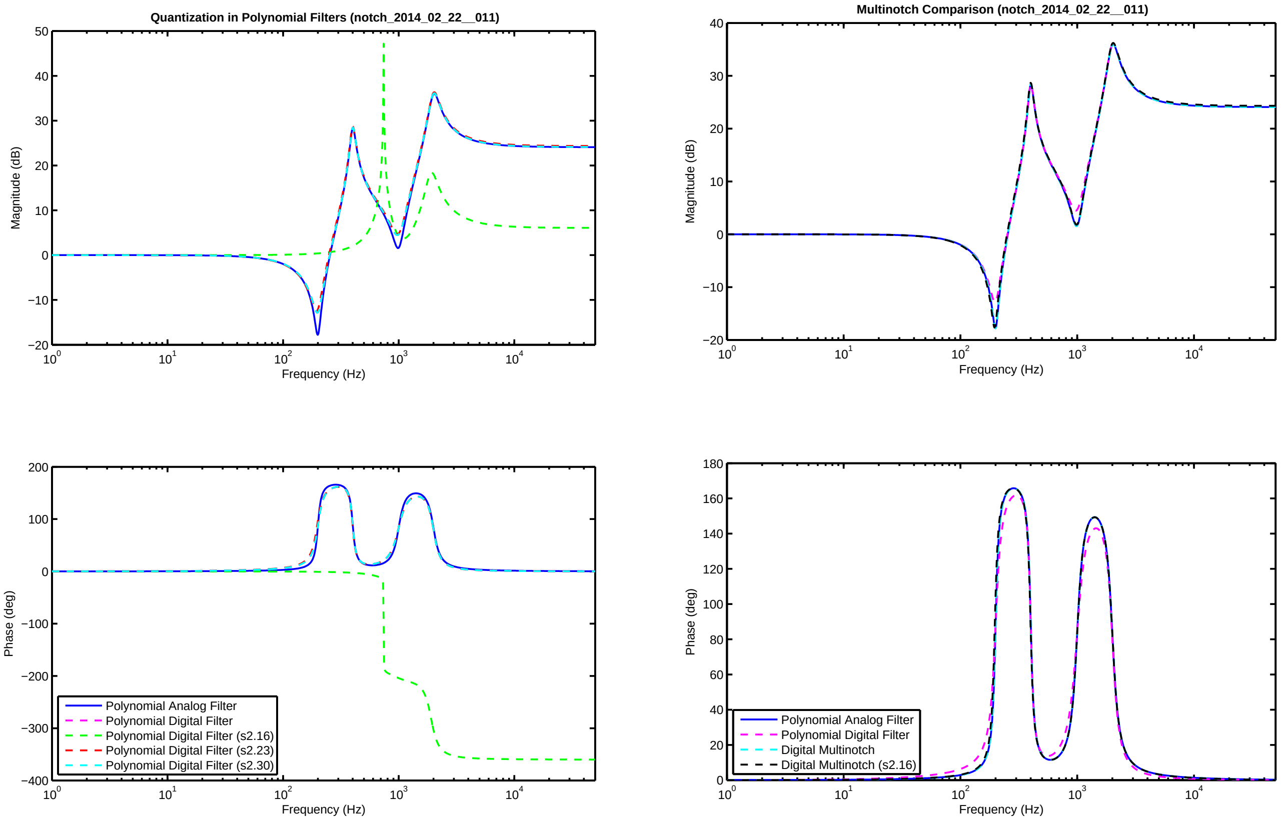


Figure 6.21: On the left: Effects of quantization on polynomial filter in the first example of Table 6.7. Quantization in the polynomial filters shows severe effects for the s2.16 and s2.23 cases. For this case, the s2.30 quantization is able to represent the unquantized digital polynomial filter, but this still does not match the analog response. The effect is only pronounced for the s2.16 quantization. On the right: A comparison of the quantized multinotch to unquantized filters for the first example of Table 6.7. It is clear that even at the extreme case of s2.16 coefficients, the multinotch still produces an accurate representation of the analog response. The curves are practically identical.

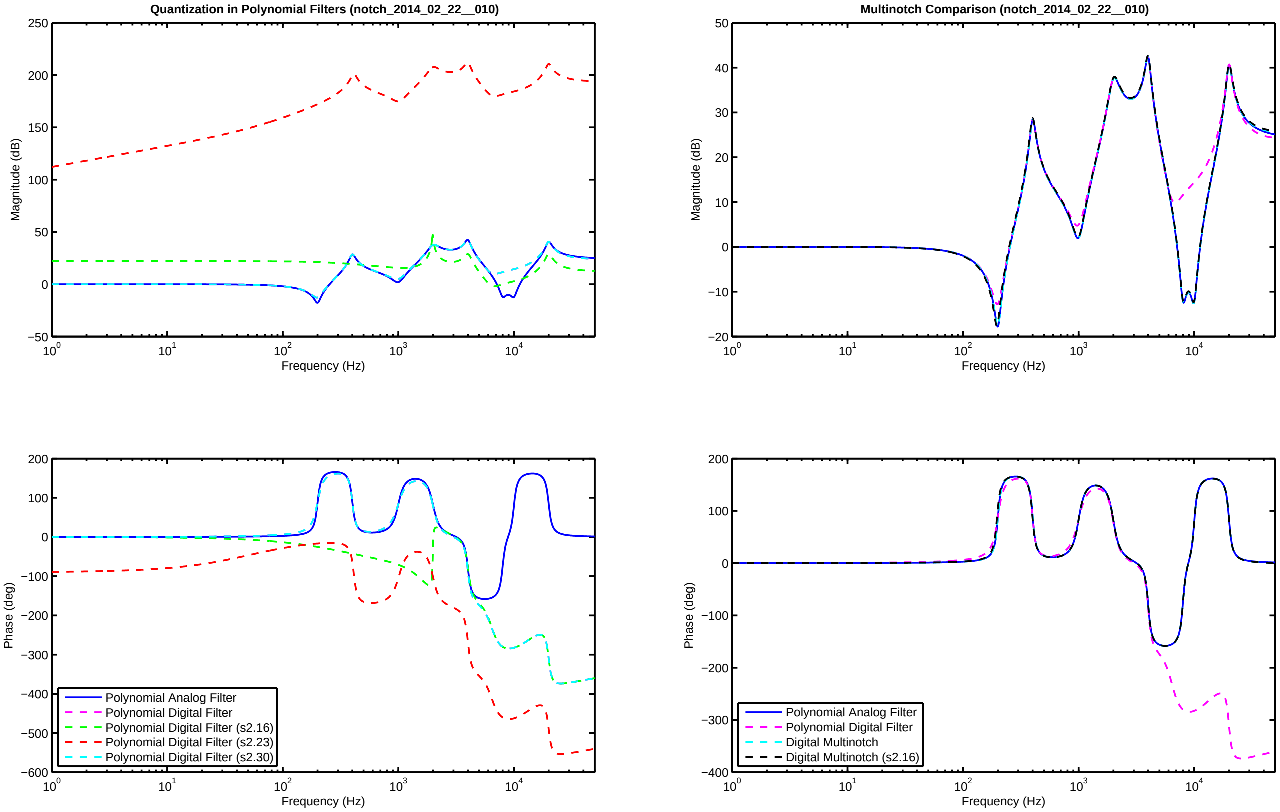


Figure 6.22: On the left: Quantization in the polynomial filters of the second example of Table 6.7 shows severe effects for the s2.16 and s2.23 cases. For this example, the s2.30 quantization is able to represent the unquantized digital polynomial filter, but this still does not match the analog response. On the right: A comparison of the quantized multinotch to unquantized filters in the second example of Table 6.7. Note that even without quantization, the discrete polynomial filter no longer matches the analog response. The multinotch, with even the coarsest quantization considered in these tests, s2.16, still matches the analog response.

## 6.18 Effects of a Relatively Small $T_S$

A significant problem can arise when $\omega_{D,i}T_S$ or $\omega_{N,i}T_S$ get very small. Equations 6.76 and 6.77 imply

$$\lim_{\omega_{D,i}T_S \to 0} a_{i,2} = 1 \quad \text{and} \quad \lim_{\omega_{N,i}T_S \to 0} \tilde{b}_{i,2} = 1. \tag{6.84}$$

Similarly, since $\cos(0) = \cosh(0) = 1$, we can see from Equations 6.78, 6.80, and 6.82 (and 6.79, 6.81, and 6.83) that

$$\lim_{\omega_{D,i}T_S \to 0} a_{i,1} = -2, \quad \text{and} \quad \lim_{\omega_{N,i}T_S \to 0} \tilde{b}_{i,1} = -2. \tag{6.85}$$

| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | $k$ |
|---|---|---|---|---|
| 10 kHz | **Float** | 9.972435e-001 ± 6.273633e-002 | 9.902495e-001 ± 6.181130e-002 | 1 |
| 10 kHz | **s2.16** | 9.972381e-001 ± 6.280493e-002 | 9.902495e-001 ± 6.173522e-002 | 1 |
| 10 kHz | **s2.23** | 9.972435e-001 ± 6.273632e-002 | 9.902495e-001 ± 6.181097e-002 | 1 |
| 10 kHz | Δ **s2.16** | 9.972436e-001 ± 6.273490e-002 | 9.902496e-001 ± 6.181050e-002 | 1 |
| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | $k$ |
| 100 kHz | **Float** | 9.999017e-001 ± 6.282160e-003 | 9.991955e-001 ± 6.228970e-003 | 1 |
| 100 kHz | **s2.16** | 9.999008e-001 ± 5.523423e-003 | 9.991913e-001 ± 6.717367e-003 | 1 |
| 100 kHz | **s2.23** | 9.999017e-001 ± 6.280814e-003 | 9.991955e-001 ± 6.229847e-003 | 1 |
| 100 kHz | Δ **s2.16** | 9.999017e-001 ± 6.282048e-003 | 9.991955e-001 ± 6.228704e-003 | 1 |
| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | $k$ |
| 1 MHz | **Float** | 9.999919e-001 ± 6.282645e-004 | 9.999213e-001 ± 6.233415e-004 | 1 |
| 1 MHz | **s2.16** | 0.999999999999995, 0.999969481956212 | 0.999999999999971, 0.999832150759166 | 1 |
| 1 MHz | **s2.23** | 9.999919e-001 ± 6.904864e-004 | 9.999213e-001 ± 5.928139e-004 | 1 |
| 1 MHz | Δ **s2.16** | 9.999919e-001 ± 6.283486e-004 | 9.999213e-001 ± 6.234487e-004 | 1 |

Table 6.8: Filter poles and zeros under quantization at different sample frequencies. Notch design parameters: $f_N = 100$, $Q_N = 40$, $f_D = 100$, $Q_D = 4$.

This means that in the limit, both the numerator and denominator approach $P(z) = z^2 - 2z + 1$, which has two roots at $z = 1$. The effect of increased sample rate relative to a given feature frequency is to push that feature closer and closer to $z = 1$ on the $z$ plane. While it is difficult to see on a z-plane plot, Tables 6.8 and 6.9 show two examples of the effect of sample rate on the quantized coefficients, by translating those back into poles and zeros. The details of how the quantized values are obtained are in Section 6.17, but for now we see that Tables 6.8 and 6.9 demonstrate that features in continuous time wind up as extremely small perturbations around $z = 1$ in discrete time. We can see from both tables that for the lower sample rate, the poles and zeros are relatively consistent despite quantization. As the sample rate goes higher, the poles and zeros corresponding to quantized values vary slightly, but these result in significant performance differences. Only the $\Delta$ Coefficients, to be introduced in Section 6.19, result in poles and zeros near the ones from the unquantized filters.

| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | $k$ |
|---|---|---|---|---|
| 10 kHz | **Float** | 9.676381e-001 ± 5.270507e-002 | 9.335457e-001 ± 1.019989e-001 | 1 |
| 10 kHz | **s2.16** | 9.676356e-001 ± 5.274999e-002 | 9.335393e-001 ± 1.020527e-001 | 1 |
| 10 kHz | **s2.23** | 9.676381e-001 ± 5.270573e-002 | 9.335457e-001 ± 1.019985e-001 | 1 |
| 10 kHz | **Δ s2.16** | 9.676385e-001 ± 5.271557e-002 | 9.335460e-001 ± 1.020009e-001 | 1 |
| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | $k$ |
| 100 kHz | **Float** | 9.968486e-001 ± 5.424303e-003 | 9.936777e-001 ± 1.081442e-002 | 1 |
| 100 kHz | **s2.16** | 9.968414e-001 ± 5.983327e-003 | 9.936751e-001 ± 1.061067e-002 | 1 |
| 100 kHz | **s2.23** | 9.968485e-001 ± 5.433827e-003 | 9.936776e-001 ± 1.081786e-002 | 1 |
| 100 kHz | **Δ s2.16** | 9.968486e-001 ± 5.422943e-003 | 9.936777e-001 ± 1.081251e-002 | 1 |
| $f_S$ | Quantization | $z_{1,2}$ | $p_{1,2}$ | k |
| 1 MHz | **Float** | 9.996857e-001 ± 5.439689e-004 | 9.993713e-001 ± 1.087596e-003 | 1 |
| 1 MHz | **s2.16** | 1.000000000000061, 0.999359121080277 | 1.000000000000042, 0.998733501182532 | 1 |
| 1 MHz | **s2.23** | 9.996857e-001 ± 5.087695e-004 | 9.993712e-001 ± 1.128530e-003 | 1 |
| 1 MHz | **Δ s2.16** | 9.996857e-001 ± 5.509819e-004 | 9.993713e-001 ± 1.088241e-003 | 1 |

Table 6.9: Filter poles and zeros under quantization at different sample frequencies. Lead design parameters: $f_N = 100$, $Q_N = 1$, $f_D = 200$, $Q_D = 1$.

We will see in Section 6.17 that these minor variations due to fixed point math give significant performance issues, as demonstrated by generating frequency responses of the different filters.

## 6.19 Δ Coefficients

Fixing this problem in fixed point math might be attempted with adding extra bits to the fractional portion. We will see in the examples of Section 6.20 that this has limited positive effect, unless the number of added bits gets significant. If we add enough bits to achieve numerical accuracy, we risk making our multiplicands too wide to fit into the hardware multiplier blocks of the desired real-time processor or FPGA. For example, Xilinx FPGA multiplier blocks in Virtex Generations 4–6, and in all Xilinx FPGAs in Generation 7, multiply a pair of twos-compliment numbers that are 25 bits by 18 bits [88]. Earlier generations, such as the Xilinx Spartan line, featured multiplies of two 18 bit twos compliment numbers [174], while the more expensive Virtex line had 25 × 18-bit twos compliment multiplies [175]. The point is that in FPGAs and fixed point DSPs, it is necessary to implement multiply and accumulate (MAC) operations with relatively narrow fixed point numbers. It is certainly possible to extend the width of multiplies using extra multiplier blocks in an FPGA or extra code in a DSP, but at the cost of extra delay [176]. Minimizing delay, especially in feedback systems, should be a high priority. (also include Altera info)

Another option is to convert all the signals in the system away from a $Z$ Transform and to a $\delta$ operator form [177, 178, 179, 180]. In [181], the filter is also broken into biquad sections and these are transformed using the $\delta$ operator. However, this method adds some complexity to the filter operation. The method proposed below simply restores accuracy to the discrete coefficients without changing the basic math operations. Some further comparison will be done in Section 6.21. The method here borrows the basic inspiration from $\delta$ operator methods but note that we are not concerning ourselves with the $z$ terms per se being clustered around $z = 1$, but the roots of the biquad polynomials. In other words, only the coefficients and not the signal space are modified. As we saw in Section 6.18, the $a_{i,1}$ and $\tilde{b}_{i,1}$ terms go to $-2$ while the $a_{i,2}$ and $\tilde{b}_{i,2}$ terms go to $1$. With this understanding, we can define:

$$\begin{align}
a_{i,1} &= -2 + a_{i,1\Delta} \quad \text{so} \quad a_{i,1\Delta} = a_{i,1} + 2, \tag{6.86}\\
a_{i,2} &= 1 + a_{i,2\Delta} \quad \text{so} \quad a_{i,2\Delta} = a_{i,1} - 1, \tag{6.87}\\
\tilde{b}_{i,1} &= -2 + \tilde{b}_{i,1\Delta} \quad \text{so} \quad \tilde{b}_{i,1\Delta} = \tilde{b}_{i,1} + 2, \text{ and} \tag{6.88}\\
\tilde{b}_{i,2} &= 1 + \tilde{b}_{i,2\Delta} \quad \text{so} \quad \tilde{b}_{i,2\Delta} = \tilde{b}_{i,1} - 1. \tag{6.89}
\end{align}$$

Now, $a_{i,1\Delta}$, $a_{i,2\Delta}$, $\tilde{b}_{i,1\Delta}$, and $\tilde{b}_{i,2\Delta}$ are small numbers. The smaller $\omega_{D,i}T_S$ gets, the smaller $a_{i,1\Delta}$ and $a_{i,2\Delta}$ get. Likewise the smaller $\omega_{N,i}T_S$ gets, the smaller $\tilde{b}_{i,1\Delta}$ and $\tilde{b}_{i,2\Delta}$ get. However, we can split up the signal multiplications

$$a_{i,1}d_i(k-1) = -2d_i(k-1) + a_{i,1\Delta}d_i(k-1), \tag{6.90}$$

$$\tilde{b}_{i,1}d_i(k-1) = -2d_i(k-1) + \tilde{b}_{i,1\Delta}d_i(k-1), \tag{6.91}$$

$$a_{i,2}d_i(k-2) = d_i(k-2) + a_{i,2\Delta}d_i(k-2), \tag{6.92}$$

and

$$\tilde{b}_{i,2}d_i(k-2) = d_i(k-2) + \tilde{b}_{i,2\Delta}d_i(k-2). \tag{6.93}$$

In each of these, the first multiplication on the right is either a trivial multiply by $2$, accomplished with a shift to the left by one bit, or it is a more trivial multiply by $1$, accomplished by doing nothing. We can now concentrate on making the second multiply more accurate. In real numbers,

$$a_{i,1\Delta}d_i(k-1) = (2^E a_{i,1\Delta})d_i(k-1)2^{-E}. \tag{6.94}$$

If we scale up $a_{i,1\Delta}$ by a number to maximize the number of significant digits in the fixed point representation, our multiplication will have the maximum accuracy. We can scale the product down by that same number for adding into the precalc sum. Likewise, we can do the same thing for the other $\Delta$ Coefficients:

$$\tilde{b}_{i,1\Delta}d_i(k-1) = (2^E \tilde{b}_{i,1\Delta})d_i(k-1)2^{-E}, \tag{6.95}$$

$$a_{i,2\Delta}d_i(k-2) = (2^E a_{i,2\Delta})d_i(k-2)2^{-E}, \tag{6.96}$$

and

$$\tilde{b}_{i,2\Delta}d_i(k-2) = (2^E \tilde{b}_{i,2\Delta})d_i(k-2)2^{-E}. \tag{6.97}$$

### 6.19.1 Computing Scaling

How do we compute the scaling factor, $2^{-E}$? Consider a coefficient, $c_i$:

$$\log_2 |c_i| = x_i \text{ means } 2^{x_i} = |c_i|. \tag{6.98}$$

Let's say we want to do multiplies with a coefficient that has a magnitude between 1 and 2. In this case we want

$$1 \leq 2^{-E_i}|c_i| < 2 \text{ or} \tag{6.99}$$

$$0 \leq \log_2 2^{-E_i}|c_i| < 1 \text{ which means} \tag{6.100}$$

$$0 \leq x_i - E_i < 1. \tag{6.101}$$

$E_i$ represents the integer part of $\log_2 |c_i|$ so,

$$\text{floor}(\log_2 |c_i|) \text{ will give us } E_i. \tag{6.102}$$

If we divide by $2^{E_i}$, it is equivalent to multiplying by $2^{-E_i}$. What is the effect of this operation? If $E_i$ is positive, then we are shrinking the magnitude of the coefficient by a power of 2. If $E_i$ is negative, then we are raising the magnitude of the coefficient by a power of 2.

Now, for each of the floating point versions of $a_{i,1\Delta}$, $a_{i,2\Delta}$, $\tilde{b}_{i,1\Delta}$, and $\tilde{b}_{i,2\Delta}$ we could have a separate value of $E_i$. However, experience has shown that if the multinotch poles and zeros are grouped so that biquads have poles and zeros that are as close as possible to each other, a single value of $2^{-E_i}$ can be used for each biquad. If we pick

$$E_i = \max(E_{i,a1}, E_{i,a2}, E_{i,b1}, E_{i,b2}), \tag{6.103}$$

that is we pick the maximum of the negative exponents for the $\Delta$ Coefficients, then we will be multiplying by the minimum $2^{-E_i}$.

### 6.19.2 Implementing $\Delta$ Coefficients

Implementing $\Delta$ Coefficients is mostly a matter of restructuring the filter calculations (or precalculations) slightly. We start by rewriting (6.73)

$$D_{P,i}(k-1) \quad = \quad -a_{i,1}\tilde{d}_i(k-1) - a_{i,2}\tilde{d}_i(k-2) \tag{6.104}$$

$$\begin{aligned} &= (2 - a_{i,1\Delta})\tilde{d}_i(k-1) \\ &\quad -(1 + a_{i,2\Delta})\tilde{d}_i(k-2) \qquad (6.105) \\ &= 2\tilde{d}_i(k-1) - \tilde{d}_i(k-2) - a_{i,1\Delta}\tilde{d}_i(k-1) \\ &\quad -a_{i,2\Delta}\tilde{d}_i(k-2) \qquad (6.106) \\ &= D_{PW,i}((k-1) + D_{PF,i}((k-1) \qquad (6.107) \end{aligned}$$

where

$$D_{PW,i}(k-1) = 2\tilde{d}_i(k-1) - \tilde{d}_i(k-2) \text{ and} \qquad (6.108)$$

$$D_{PF,i}(k-1) = -a_{i,1\Delta}\tilde{d}_i(k-1) - a_{i,2\Delta}\tilde{d}_i(k-2). \qquad (6.109)$$

The fractional precalc can be done in two steps as:

$$\begin{aligned} D_{PFL,i}(k-1) &= -(2^{-E_i}a_{i,1\Delta})\tilde{d}_i(k-1) \\ &\quad -(2^{-E_i}a_{i,2\Delta})\tilde{d}_i(k-2). \qquad (6.110) \\ D_{PF,i}(k-1) &= 2^{E_i}D_{PFL,i}(k-1). \qquad (6.111) \end{aligned}$$

The coefficients in (6.110) are computed from the floating point numbers, $2^{-E_i}a_{i,1\Delta}$, and $2^{-E_i}a_{i,2\Delta}$, before being converted to fixed point. Once the multiplication has been done with high precision, the product is shifted back in (6.111) for addition with $D_{PW,i}(k-1)$. If the scaled down product is insignificant compared to $D_{PW,i}(k-1)$. However, the high precision multiplication means that if the product is significant, it is also accurate.

We repeat the process with the output precalcs. Rewriting (6.74)

$$\begin{aligned} F_{P,i}(k-1) &= \tilde{b}_{i,1}\tilde{d}_i(k-1) + \tilde{b}_{i,2}\tilde{d}_i(k-2) \qquad (6.112) \\ &= (2 - \tilde{b}_{i,1\Delta})\tilde{d}_i(k-1) \\ &\quad -(1 + \tilde{b}_{i,2\Delta})\tilde{d}_i(k-2) \qquad (6.113) \\ &= 2\tilde{d}_i(k-1) - \tilde{d}_i(k-2) \\ &\quad -\tilde{b}_{i,1\Delta}\tilde{d}_i(k-1) - \tilde{b}_{i,2\Delta}\tilde{d}_i(k-2) \qquad (6.114) \\ &= F_{PW,i}((k-1) + F_{PF,i}((k-1) \qquad (6.115) \end{aligned}$$

where

$$F_{PW,i}(k-1) = 2\tilde{d}_i(k-1) - \tilde{d}_i(k-2) \text{ and} \qquad (6.116)$$

$$F_{PF,i}(k-1) = -\tilde{b}_{i,1\Delta}\tilde{d}_i(k-1) - \tilde{b}_{i,2\Delta}\tilde{d}_i(k-2). \qquad (6.117)$$

The fractional precalc can be done in two steps as:

$$\begin{aligned} F_{PFL,i}(k-1) &= -(2^{-E_i}\tilde{b}_{i,1\Delta})\tilde{d}_i(k-1) \\ &\quad -(2^{-E_i}\tilde{b}_{i,2\Delta})\tilde{d}_i(k-2). \qquad (6.118) \\ F_{PF,i}(k-1) &= 2^{E_i}F_{PFL,i}(k-1). \qquad (6.119) \end{aligned}$$

The equations above illustrate one of the beauties of the $\Delta$ Coefficient approach. While there are a few extra additions and right shifts of multiplied values in the precalculation portion of the filter, there are no extra multiplies. Instead many of the existing multiplies have been made far more accurate. Additions and shifts are extremely easy operations in digital hardware, and therefore the added computational burden of this extra accuracy in very small.

## 6.20 $\Delta$ Coefficient Examples

| Example 1 | | | | |
|---|---|---|---|---|
| **Biquad #** | $f_{N,n}$ **(Hz)** | $Q_n$ | $f_{N,d}$ **(Hz)** | $Q_d$ |
| 1 | 100 | 40 | 100 | 4 |
| **Example 2** | | | | |
| **Biquad #** | $f_{N,n}$ **(Hz)** | $Q_n$ | $f_{N,d}$ **(Hz)** | $Q_d$ |
| 1 | 100 | 1 | 200 | 1 |

Table 6.10: Filter parameters for both examples.

In order to compare filters, a set of filter parameters was chosen in the form of sets of analog biquad parameters, such as those in Table 6.10. Unlike the examples in Section 6.17, only a single biquad was needed to demonstrate the desired effects. For a given set of biquad parameters, the sample frequency was varied between 10 kHz, 100 kHz, and 1 MHz. In the first example from Table 6.10, a high Q notch filter was chosen, centered at 100 Hz. In the second example, a lead-lag filter was implemented where the lead natural frequency was set to 100 Hz and the lag natural frequency was set to 200 Hz. The poles and zeros from these examples have already been presented in Tables 6.8 and 6.9 Section 6.18.

The filter design parameters are specified by Table 6.10. These parameters were then translated into digital filter in the form of a single biquad, essentially a single notch in the same form as the filters in Figure 6.16. The coefficients of the digital filter were then scaled up by a quantization factor, say $2^{16} - 1$ for an s2.16 quantization. The scaled up coefficients were then fixed (fractional portion removed) and scaled down by the same quantization factor. Thus, floating point numbers were made to represent fixed point coefficients. Frequency responses were computed for the filters with floating point coefficients and with quantized coefficients. The same original filter specifications were used to show the variation with sample frequency. Finally, these were compared to a quantized biquad implemented with $\Delta$ Coefficients. The three sample rates of $f_S = 10$ kHz, 100 kHz, and 1 MHz were adequate to demonstrate the result.

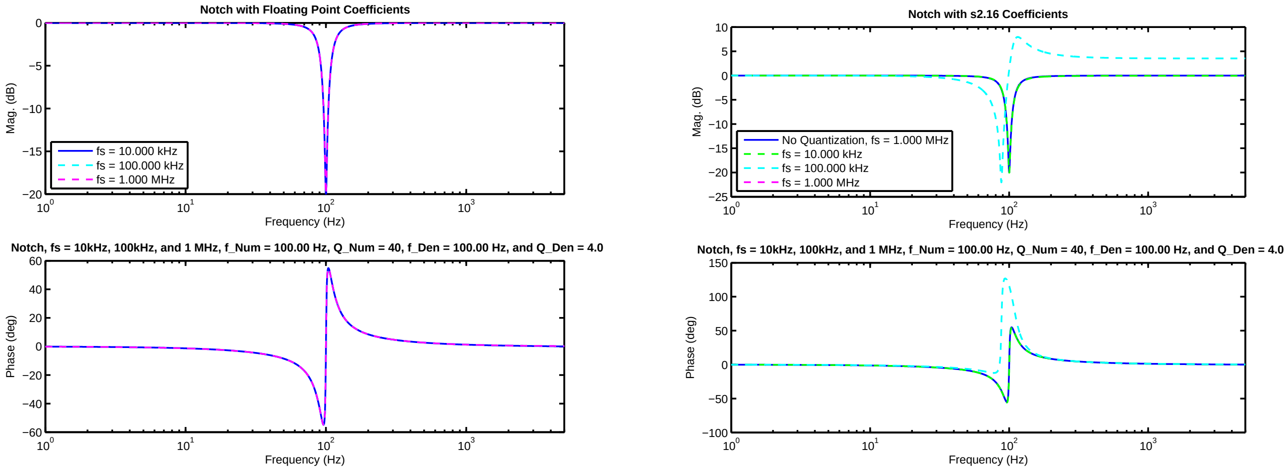


Figure 6.23: Notch with $f_n$ and $f_d$ at $100$ Hz, $Q_n = 40$, $Q_d = 4$. On the left: no quantization. On the right: quantized to s2.16. Sample frequencies are $f_S = 10$ kHz, $100$ kHz, and $1$ MHz. With no quantization, there is effectively no difference.

With floating point coefficients, we can see from the left sides of Figures 6.23 and 6.25 that the filters are essentially unaffected by the change in sample frequency. Quantizing the filter coefficients to a s2.16 format, as shown on the right side of Figures 6.23 and 6.25. These show that while the quantized filter is accurate at $f_S = 10$ kHz, it becomes inaccurate at $f_S = 100$ kHz. With $f_S = 1$ MHz, the plot is off scale. The situation improves some by adding more bits to a s2.23 format, as shown on the left of Figures 6.24 and 6.26, however none of these are at all accurate. In fact, both figures demonstrate one of the dangers of quantizing such parameter values, the nonlinear degradation. In both examples the filter response at $f_S = 1$ MHz is more accurate than that at $f_S = 100$ kHz. The success of the $\Delta$ Coefficients is demonstrated on the right of Figures 6.24 and 6.26. Using only s2.16 $\Delta$ Coefficients, we have essentially restored the filter accuracy for all three sample frequencies.

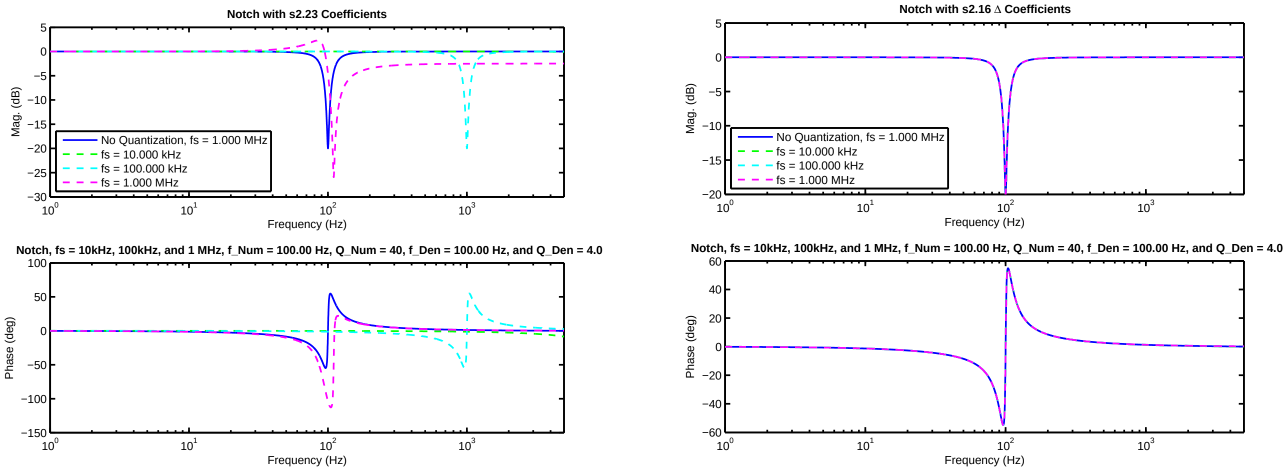


Figure 6.24: Notch with $f_n$ and $f_d$ at 100 Hz, $Q_n = 40$, $Q_d = 4$. On the left: quantized to s2.23. On the right: using Δ Coefficients, quantized to s2.16. Sample frequencies are $f_S = 10$ kHz, 100 kHz, and 1 MHz.

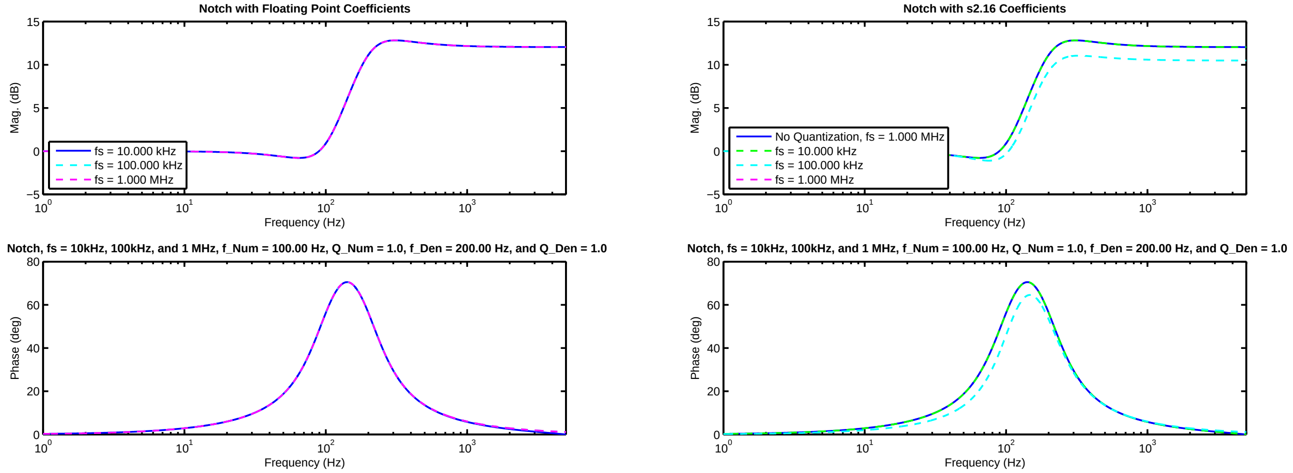


Figure 6.25: Lead with $f_n$ at 100 Hz and $f_d$ at 200 Hz, $Q_n = 1$, $Q_d = 1$. On the left: no quantization. On the right: quantized to s2.16. Sample frequencies are $f_S = 10$ kHz, 100 kHz, and 1 MHz. With no quantization, there is effectively no difference.

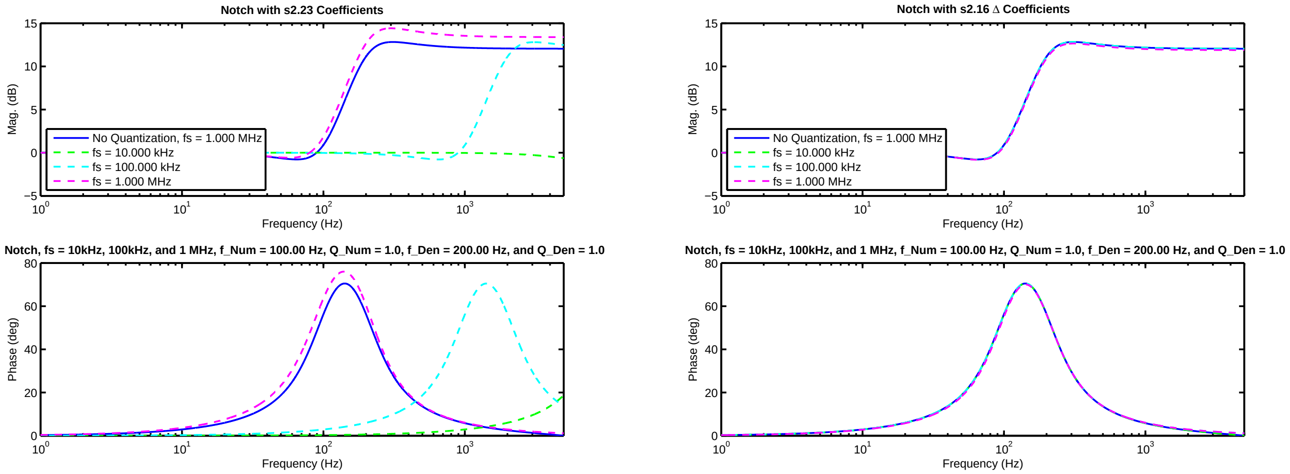


Figure 6.26: Lead with $f_n$ at 100 Hz and $f_d$ at 200 Hz, $Q_n = 1$, $Q_d = 1$. On the left: quantized to s2.23. On the right: using $\Delta$ Coefficients, quantized to s2.16. $f_S = 10$ kHz, 100 kHz, and 1 MHz.

## 6.21 $\Delta$ Coefficients Versus $\delta$ Parameterization and Floating Point

The two most natural comparisons to make with $\Delta$ Coefficients are to the $\delta$ parameterization [177, 178, 179, 180] and to floating point operations.

In contrast with the $\Delta$ Coefficients which maintain the delay form ($z^{-1}$) of the discrete filter while improving the coefficient accuracy, the $\delta$ parameterization [177, 178, 179, 180] pushes the discrete parameters closer to the the continuous time parameters and the filter calculation are transformed from step form to a differential form. Most similar to the method here is [181] which breaks the filter into biquads as well and then applies the $\delta$ parameterization to each biquads. The assumption when using this is that the filter has been somehow discretized already and then is reparameterized using a forward rectangular rule integration approximation. In the $\Delta$ Coefficient formulation, each biquad is discretized individually and the matched pole-zero method provides excellent agreement to continuous frequency responses for biquads. Further comparisons will be made in [165]. The $\Delta$ Coefficients approximate floating point coefficients in a computationally inexpensive way. The filter computations are done using the delay form, which is slightly less complicated than the differential form of $\delta$ parameterization, where the $\delta^{-1}$ block of [181] is an inner loop digital integrator. Finally, the $\delta$ parameterization is useful primarily when the sample rate is high relative to the dynamics being filtered, when $\omega_0 T_S = 2\pi f_0 / f_S$ gets very small. The forward rectangular rule becomes much more inaccurate for slow sample rates. Mechatronic systems are known for having a wide spread in dynamics, and so the $\delta$ parameterization could have some issues not present in the $\Delta$ coefficients [165].

In minimal latency control of a mechatronic system, the original inspiration for the multinotch [54], floating point computations may take too long. A native mode 25-bit × 18-bit floating point multiply and add in a Xilinx DSP48E will have a latency of 4 clock cycles [88]. Single precision floating point multiplies take 8 clock cycles, but the additions must be made separately and take 11 clock cycles [176]. (Altera?)

## 6.22 Multinotch Summary

The multinotch represents a new way of structuring digital filters so as combine physical intuition, numerical stability, and low, fixed latency into a single architecture [54, 33, 110, 182]. It lends itself extremely well to real time implementation, and is especially effective in fixed point environments, such as FPGA or DSP code. Furthermore, the regular, recursive structure of the filter allows it to be programmed easily. It combines for the first time two extremely desirable features in a real time filter: small, fixed computational latency and high numerical fidelity.

However, even the multinotch can degrade severely when the sample frequency is several orders of magnitude higher than the frequencies of filter features. In order to correct this while still maintaining fixed point math, suitable for high speed, real time implementation on an FPGA or a DSP, the $\Delta$ Coefficient parameterization has been presented. This restores the performance of the multinotch, while adding only a few extra precalculations and no extra bits. The $\Delta$ Coefficients extend the range and accuracy of the multinotch without affecting the physical intuition of the analog parameters in the digital filter implementation, which is extremely helpful in debugging physical systems.

On it's own, the multinotch is extremely useful in the control of low latency, mechatronic control systems (and other systems with high-Q resonances and anti-resonances. However, it also can be used to generate a state space structure that preserves both these numerical properties and physical intuition [3]. In particular, the discretization method described earlier allows direct comparison of analog and digital states [4]. We believe that this Biquad State Space (BSS) structure will be very useful in applying modern control methods to lightly damped systems [183, 184].

## 6.23 Filters without Direct Feedthrough

The multinotch [54, 33] was originally created specifically for loop shaping on a high speed AFM with lots of high frequency, high Q dynamics. The needed filter was going to have direct feedthrough in order to have minimum latency. However, there are cases where one simply needs something like a low pass filter, either for low pass or to include in a system state space model such as the Biquad State Space (BSS) (Section 9.15). One of my recent efforts is to put that component in place [166]. Continuous time low pass filters are an example of filters with no direct feedthrough (Section 9.27). Depending upon the discretization choices made, the discrete-time low pass may or may not have direct feed through. We will focus on low pass filters here, and in the state space structures in Section 9.27.

## 6.24 The $\delta$ Parameterization

$\delta$ Parameterization: when it helps, when it hurts

## 6.25 Filters for Loop Shaping: Do's and Don'ts

This chapter has presented a lot of building blocks for extra components in a digital controller. Unlike some sort "global model" state space design, we can work on the individual pieces independently, but at some point we need to sanity check how they work in the whole.

There is nothing that stops us from taking these individually designed components and combining them back into a polynomial form filter for analysis in MATLAB . Generating Bode plots of the whole controller and the open loop, combining these with the plant FRF and mathematically "closing the loop" are all pieces of the checkout. From the same combined model, one can generate the discrete root locus. The thing that stops most engineers is the tedium of transferring between models. That's why, (as we advocated in Section 3.25) building the software pipes to transfer models and measurements between different design/analysis stages is critical. When checking different forms of a model/design take an afternoon, people avoid it. When it only takes a few keystokes or button pushes, it gets done 15 times a day. Taking the time to build the tools allows us to iterate rapidly, and this is at the core of engineering.

## 6.26 Chapter Summary and Context

# Chapter 7

# Signal Detection, Sensors, Sample Rates, and Noise (Oh My)

## 7.1 In This Chapter

In this chapter we will focus on characterizing the noise that comes into feedback loops with more detail than is typically discussed. Of issue is identifying the components of noises that enter the loop and their effect on different measurement points around the loop. The fundamental method for this is called PES Pareto, which – under certain assumptions – allows us to build up the strata of noises at different measurement points in the frequency domain. Because Bode's integral theorem teaches us that we can only use our loop filtering to shape where noise is amplified and attenuated in the frequency domain, we then discuss how to limit the effects of noise sources before they get into the feedback loop. Of particular interest are the phase effects of anti-alias filters.

As many sensor signals are modulated, we also discuss the subject of signal demodulation. It turns out that coherent demodulation can not only limit the noise entering the loop through a particular sensor, but can also considerably cut the time delay incurred via non-coherent demodulation methods.

None of this creates a good feedback control algorithm. Instead, the methods in this chapter help us keep from messing up good feedback control algorithms due to inattention to noise and delay.

## 7.2 Motivation: Why Talk About Signals, Sensors, and Noise?

We have tried to emphasize the idea that when all other design methods have been done as well as possible, what remains as the final limits of performance – even for a system that can be fairly accurately modeled as being linear and time-invariant – are time delay and noise. Time delay eventually wipes out phase margin and noise at some point just gets passed by the closed-loop through to some critical signal limiting accuracy. Bode's Integral Theorem shows us that we cannot escape the effects of noise that has entered the loop; we can only diminish our sensitivity to it at some set of frequencies with the consequence of increasing our sensitivity to noise at a different set. While Chapter 6 described filtering from the context of how we shape our loop response to achieve bandwidth and performance goals, that same loop shaping is applied to noise and disturbance signals entering the loop with one glaring exception: the entry points for noise inputs are scattered all along the loop and so each of these noises has its own filter to traverse before showing up in one of our other measured signals.

This chapter has two main themes, and they generally follow the outlines of the tutorials in [2, 185]. The first portion will explain the methodology of noise analysis known as PES Pareto [34, 35, 36, 37]. This method aims to tackle the propagation of noise from different sources around the feedback loop. A combination of a few theoretical constructs and a measurement methodology, PES Pareto allows the engineer to construct the strata of the noise spectra from different sources around the loop and with that in hand, to model the effects of changing one of those input noise sources. The second theme is that of improving the signal detection used in feedback loops as a means of limiting noise before it enters the loop. The other consequence of learning about modern demodulation methods is that they usually allow a considerable decrease in the time needed to detect a change in a signal. In other words, modern demodulation can limit latency in a measured signal, thereby keeping that latency out of our loop.

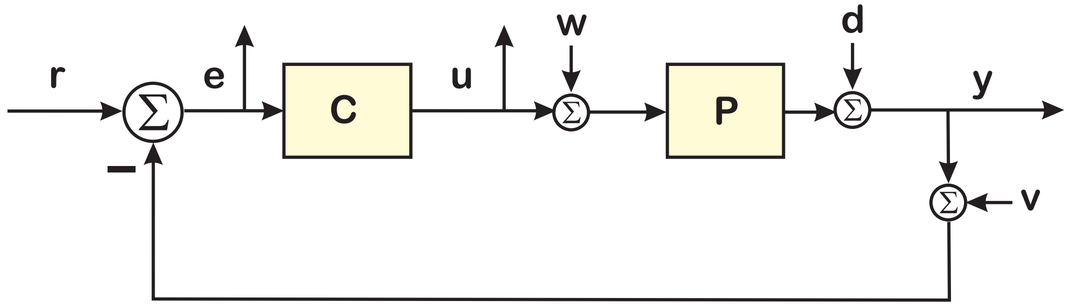


Figure 7.1: **A classic linear feedback loop, including "both kinds" of noise, process and sensor, for our state-space formulation.**

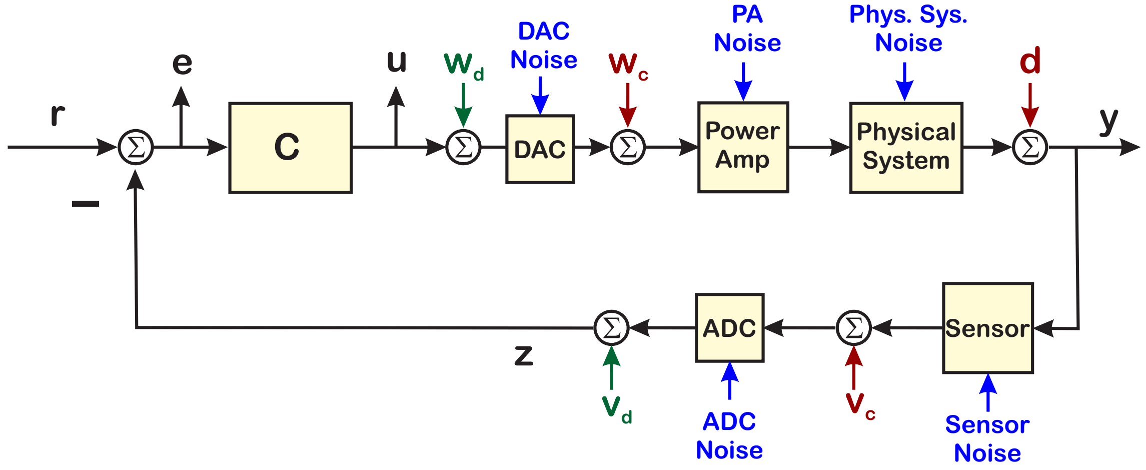


Figure 7.2: **Unbundling the $C$ and $P$ to reveal the components that generate the noise. Furthermore, we normally think of process and measurement noise as either continuous or discrete, but this has implications for where these noises show up in the loop. The process ($w$) and sensor ($v$) noises are really abstractions of different noises from the loop components. The effects of these true blue noises shows up in the measured loop signals.**

Earlier we stated that when everything else has been done correctly, when the system is in its linear range, when the compensator is using an accurate model of the physical system, when the design equalizes out the unacceptable dynamics, the limiting factors will be time delay and noise. While that all makes sense, there was little stated about how to actually characterize and deal with these issues. The first fallacy about noise is the idea that it can be treated as a single entity. We should really be talking about noises in the system. The issues with noises are now to measure them, how to trace them back to their sources, how to quantify their effects on our control loops, and how to minimize the effects of those noises on our loops. Our goal is not some esoteric wholly unusable mathematical construct. Instead, we want a practical methodology that gives us insight and directs us where to put our effort. For that we will introduce a method called PES Pareto (named for its historical roots).

“But wait!” you say. “All those block diagrams used to justify state-space methods e.g. Figure 7.1 clearly show noise, both kinds: process and sensor, as the Blues Brothers would say. Well, yes, but the idea that noise enters into the loop at two distinct spots is a nice mathematical abstraction that has made several optimization methods practical, but it tells us very little about the noise itself. It is, however, “actionable” and “tractable”, in that if we really knew the noises as if they entered at those two spots, we could apply our optimization methods, e.g. $H_2, H_\infty$, or something else. What we need is a way to measure different noises, to get to them as a source, and to see how those noise sources affect our system at different points of interest for our design.

If we expand our view of the block diagram to that of Figure 7.2 we gave opened up some of the groupings of the $C$ and $P$ blocks to show some underlying structure. Ignoring for a moment the individual block noises, we see something else: that we have to choose whether to consider sensor and/or process noise to be digital (discrete time) or analog (continuous time). That choice moves the noise to either the computer side of the ADC and DAC ($w_d$ and $v_d$) or to the physical system side ($w_c$ and $v_c$). It is almost universal that engineers will pick both of these noises to be in the same domain, but one thing that often gets forgotten is that we want the continuous time and discrete time noises to have the same frequency domain representation, which implies integrating the continuous time noise over the sample interval. It is also worth being fully aware that disturbances into the system, in this case the red $d$ term, are treated somewhat different from noise in that engineers usually assume that we know something about the characteristics of $d$, that it might have predictable or even repeatable components, and that it might be sensed using an auxiliary sensor. Returning to the noises, for the sake of having any of our modeling tools work, we will pick either the digital representation of noise, $w_d, v_d$ or the analog representation of noise, $w_c, v_c$.

However, before we get there, we have to try to understand the noise sources that we wish to amalgamate into our $w, v$ terms. These are the individual block noises in Figure 7.2, denoted in true blue. We would like a way to isolate each of these as a noise source, that is as an input to the loop prior to being filtered by the loop dynamics. If we can do this, we would then like to see what the effect of each of these is on our loop measurement points, e.g. how does ADC noise show up at the error signal, $e$, the control signal, $u$, or the system output, $y$.

Let me say right here that this involves a lot of assumptions and approximation to make the characterization problem at all tractable. We have to assume that we can isolate components of the loop and that their behavior and noise properties will be the same as when they are connected. We have to assume that the system is “mostly linear” so that we can filter the noise properties back and forth. Only one type of noise gracefully passes through linear filters in a way that we can characterize it, and that is additive, white, Gaussian noise (AWGN), so we have to assume that our measured noises can be beat into a form to call them AWGN. We have to assume some time invariance in our systems, so that a measurement made when the loop is open is still meaningful an hour later when the loop is closed. However, these assumptions are really pretty mild compared to the abstractions and assumptions needed to make almost any theoretical optimization problem tractable, so we will proceed where we can. We will find that if we are willing to make those assumptions, we are rewarded with some really insightful and practically useful characterization. This allows us to isolate and attack certain noises at the source (before they enter the loop).

We will have to make a special exception when handling ADC and DAC quantization, which is often modeled as additive, white, uniform, noise (AWUN) using the Widrow model [38]. It turns out that only AWGN is guaranteed to remain AWGN when passed through a linear filter, but we will see below and in Section 7.6.3 that this doesn't present a real problem to PES Pareto.

Once again, we draw inspiration and guidance from Bode's integral theorem , because that gives us a mathematical limit on the idealizes "best we can do" and provides guidance on how we must shape our control loops so as not to amplify noises unnecessarily. We will focus on signals that we can measure in the lab, while occasionally having to solve the problem in the other direction when accurate measurements cannot be made. We will follow the guidance of Sherlock Holmes [186], who taught us that, "Once you have removed all that is impossible, whatever remains, no matter how improbable, must be the truth."

The estimates we can get are sometimes rough, but they are more complete than many other measurements that are practical to implement. We rely on an assumption that the system is "mostly linear" so that we can apply frequency domain methods. We make frequency response function measurements to get usable system models and use those models to do loop algebra on the cross and auto spectra of the signals.

In previous chapters we discussed using filters for loop shaping (Chapter 5) and how to build those filter components in a numerically robust and minimal latency way (Chapter 6). We also discussed the implications of Bode's Integral Theorem and Stein's Dirt Digging [1]: that noise – once it enters the loop – will not be eliminated by the loop (at least not through any linear filter means), but merely amplified or attenuated with a "conservation of dirt" (noise amplification) principle underlying the whole thing.

This chapter then focuses on noise; not the esoteric noise of cosmic background radiation or the quantum sources of shot noise (although these are really cool things to learn about), but on how to measure noise in a feedback loop, back it out as an external input to the loop, and see its effect on the the closed-loop error and closed-loop output. This Bode's Theorem inspired methodology is called

PES Pareto. This is introduced in Section 7.3 and we delve into it in Section 7.4. In PES Pareto, we were able to measure much of the noise in the loop by opening the loop at various points and doing a process of elimination. However, ADC and DAC quantization provided such a low level of noise that we had to try something different.

Once we have tracked different noise components to their respective sources and found out how much each of the components affects the loop error and output, we can attack the noise at the inputs by filtering the noise before it enters the loop (for baseband signals) or by demodulation methods that provide significant filtering (for modulated signals). In this we are not violating Bode's Integral Theorem, merely reading the fine print. Input filters are often tied to off-the-shelf sensors. That may be convenient packaging, but as with many things in control systems, that filtering might not be designed with the idea of using it in a feedback loop and can cause problems.

Another aspect of input signals is that they start out analog and are shaped by analog input filters. Perhaps the best known of these to control engineers is the anti-alias filter, a low pass filter placed on the input signal to ensure that nothing above the Nyquist rate enters the control loop. There are multiple issues with this (Section 7.12).

Returning to control sensor signals are modulated onto a carrier, we will discuss the tradeoffs between non-coherent demodulation and coherent demodulation with a couple of examples. This is standard fare for communication engineers, but is not something is largely ignored by control texts.. Nevertheless, poor demodulation can allow significant noise and nonlinearities into the loop, and so it behooves us – makes our control life significantly easier – if we can understand coherent demodulation algorithms that can provide significant nonlinear and linear filtering. We will see how spending the effort to properly demodulate these signals can dramatically cut the apparent sensor noise and effective signal latency seen by the control loop. We will do this in Sections 7.13, 7.21, and 7.19.

## 7.3 Noise Filtering in Feedback Introduction

Now, I know what you're thinking: Didn't we just leave the filtering party back in Chapters 6 and 5? Well, that was the "filters for loop shaping" discussion, not the "filters for noise minimization" discussion. I'm glad we got that cleared up. Noise is a term used to describe anything from shot noise to biases due to cables or friction. Noise can be easy to model (e.g additive white Gaussian noise) and not describe what we are observing or may be perfectly descriptive and hard to handle analytically. As with all of the above, it places a limit on what information we can cleanly extract from a system, either in our attempts to model the system or to properly control it.

Less common is the understanding of the effects of noise through the feedback loop, or how to back noise measurements out to their sources. In the legendary “Respect the Unstable” Bode Lecture of 1989 [1] Gunter Stein educated us to the idea that loops do not eliminate noise, they merely move it around, as he brilliantly illustrated with a dirt digging problem, reconstructed from memory on the right side of Figure 7.3.

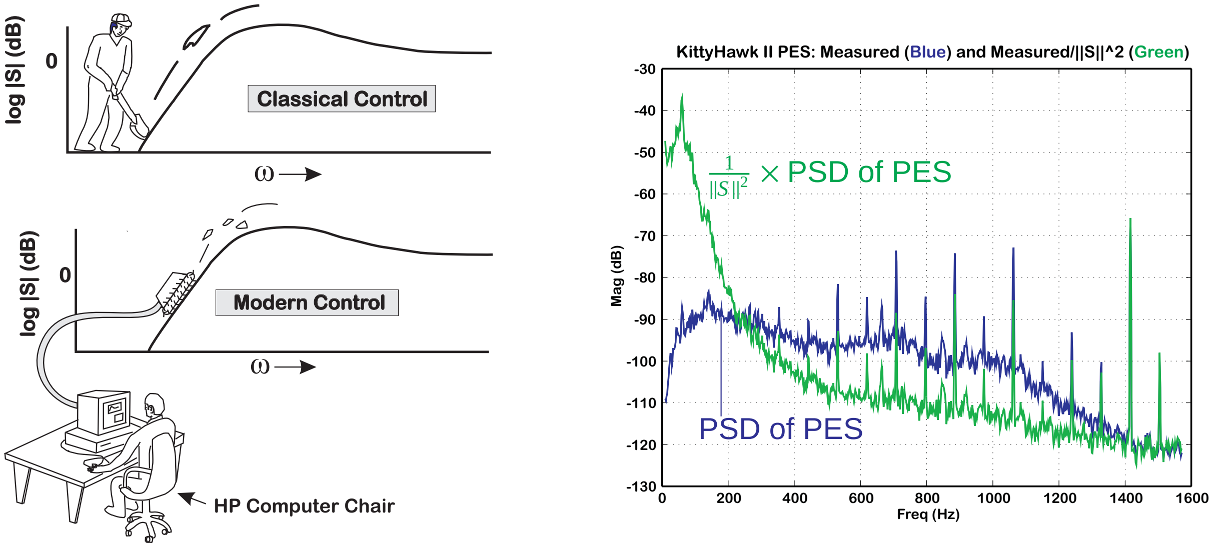


Figure 7.3: On the left, Gunter Stein’s dirt digging analogy, recreated from memory circa 1994. On the right, KittyHawk 1.3” disk drive: PSD of PES, and PSD of PES filtered by $\frac{1}{\|S\|^2}$.

If one uses that as a starting point and works backwards, one can establish what the “input noise” must have looked like, as shown in Figure 7.3. This became the basis for the PES Pareto methodology of analyzing the effects of noise on a system [34, 35, 36, 37].

One of the issues with passing noise through a loop is having any of it be tractable. It turns out (and later versions of this document may actually have the reference) that if we can model the noise as additive, white, Gaussian noise (AWGN) then the power spectral density (PSD) of that noise can be analyzed as it passes through a filter. We saw this in our measurements of frequency response functions in Chapter 3, Section 3.16. Thus, this assumption of having AWGn and being able to measure the power spectral density (PSD) of the noise allows us to filter the noise forward and backward around the loop to (a) track it to its source and (b) find its effect on the loop error.

Even then, when we are trying to make practical measurements, we must default to some idealizations. The Widrow model [38] of quantization (Section 7.6.3), assumes quantization error can be modeled as additive white, uniform noise (AWUN) on the interval, $[-q/2, q/2]$, where $q$ is a single quantization unit (1 bit). To analyze noise through a linear filter requires an assumption of additive, Gaussian, white noise (AWGN). That is, when we pass AWGN through a filter, it is still AWGN, but with a possibly different mean and variance. We have no such guarantees for AWUN, but what matters in the frequency domain is that the autocorrelation of both AWGN and AWUN are delta functions at $\tau = 0$, which means that they both have uniform power spectral densities (PSDs) out to the Nyquist frequency. For our analysis, that is enough.

We step away from analytical purity and mathematical exactitude to be able to gain understanding. The Widrow model is used to analyze the effects of ADC and DAC quantization, but it is not the only measure. A popular one with circuit designers is called the Effective Number of Bits (ENOB). Several of these measures are discussed in Section 7.6.3, not because they are useful for PES Pareto, but because they are useful when someone using PES Pareto has to discuss quantization noise with a circuit designer.

In order to get a handle on how noise at a sensor, or due to quantization at an ADC, or quantization at a DAC, or a power amplifier, or an input signal affects the overall loop response we need to make an approximation and realize that it has its limitation. Additive, white, Gaussian noise can pass through a linear filter and the result is that the noise power spectral density is shaped by the magnitude squared of the filter response. In other words, we can use this approximation to look at how noise from different inputs filters through a loop. Perhaps as importantly, we can back out the noise contributions of different inputs. This allows us to focus our effort on the key contributors. In the PES Pareto method, it was enough to start with the Widrow quantization model, we can take the mean and variance from that, and spread that variance across the relevant frequency bandwidth to create a Power Spectral Density (PSD) that when integrated is consistent with Parseval's theorem [26]. We can then use that calculated PSD for the rest of our noise analysis. It is not exact, but pretty effective.

The question becomes, what to do about it? If Bode's integral theorem tells us that we cannot get rid of noise amplification, only move it around, and the discrete time version of it tells us that we have a finite range over which to work, are we forever stuck with noise? Can any filter help us? The answer is that Bode's Theorem applies to noise amplification in the loop. If we can minimize the noise before it enters the loop, at the source, then we are fine. This is where filters can play a key role. However, many of the signals that we rely upon for servo information are modulated on some carrier. Traditionally, when Silicon was expensive, simple methods of demodulation to extract the signal were used. We will discuss the problems with these methods and give a couple of examples of how intelligent use of coherent demodulation can minimize the noise being demodulated. In the end, the same filters we use for loop shaping also shape our noise spectra.

Before we can do these things, we need some method of tracking noise through a feedback loop. It turns out that if we are willing to use linear analysis and assume additive, white, Gaussian noise (AWGN), we are able to say quite a lot because the power spectral density (PSD) of such noise can be added to the PSD of other noises and they can be understood when filtered through loop elements. Few other noise “measures” can be managed this way, and so even though circuit designers might prefer the effective number of bits (ENOB) as a measure of ADC and DAC quality, their Widrow model PSDs are more informative.

## 7.4 An Introduction to PES Pareto

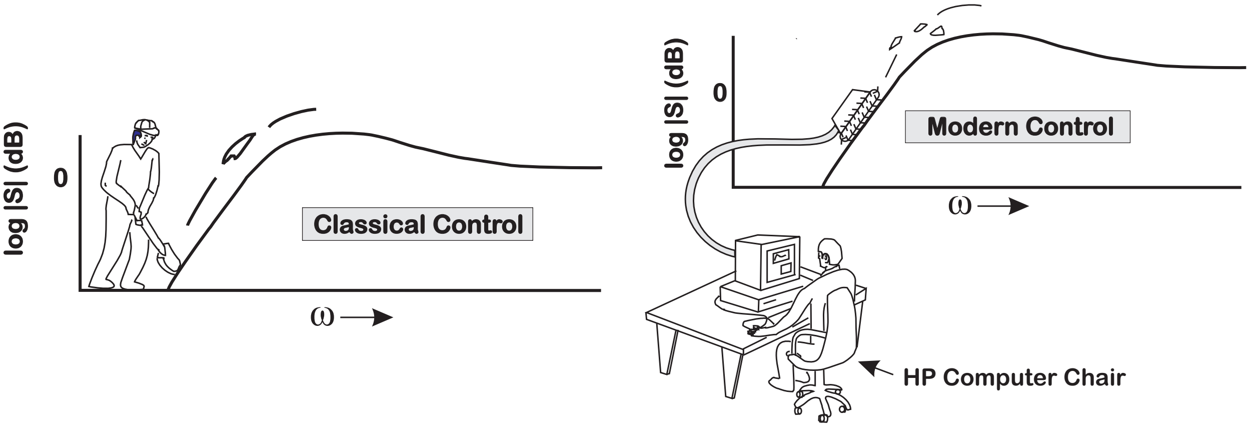


Figure 7.4: Gunter Stein’s dirt digging analogy, recreated from memory circa 1994.

The PES Pareto method arose out of trying to quantify the fundamental limits of position accuracy for hard disk drives (HDD) [34, 35, 36, 37]. The work, first internal to Hewlett-Packard, was published after HP exited the disk drive business. In the years that followed, it became apparent from the disk drive control sessions at the American Control Conferences in the early 2000s, that many of the disk drive manufacturers of the day had adopted this method for their work. In the years since then, the number of disk drive companies has reduced to three and the disk drive control sessions at ACC are only a fond memory. It is not clear if PES Pareto is still used in the remaining industry, but it seems

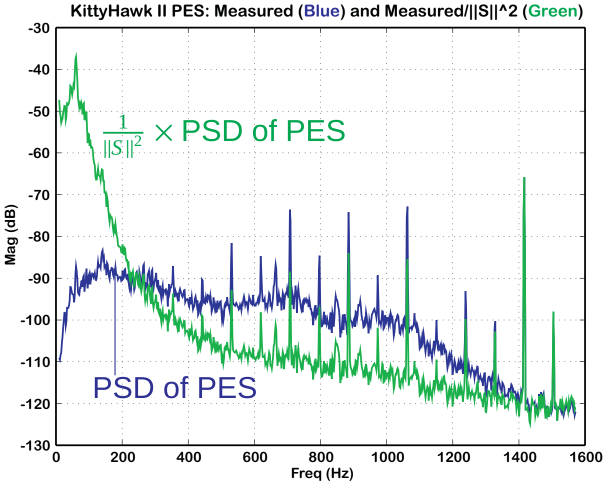


Figure 7.5: HP KittyHawk 1.3" disk drive: PSD of PES, and PSD of PES filtered by $\frac{1}{\|S\|^2}$.

like a good time to teach a new generation of control engineers how to apply it to their control loops.

It is worth spending a moment to consider why the PES Pareto methodology was so quickly accepted by the disk drive industry while barely penetrating control practice anywhere else. Servo engineers in the disk drive industry had several common practices that made an intuitive understanding of PES Pareto easy for them, once it had been explained:

- They were used to making extensive, but not unified, measurements on physical systems. In fact, they had a variety of relatively expensive electronic test instruments in the lab and were well versed in making measurements in both time and frequency domain. What was far less common was a unified view of how to practically combine those measurements.
- They were used to working on difficult to control problems with severe limits on processing power, sample frequency, cost, and model completeness. This meant that they were receptive to the notion that there was no one "magic bullet" measurement method that would apply to all the different components around the feedback loop, and were ready to find a reasonable way to piece these together. Put another way, they had no issues with the "mixed metaphor" approach that physical system limits mandate and that PES Pareto embraces.
- They were acutely aware of noise and uncertainty limiting what they could do in their control designs, working against what they called a "noise budget." However, they were were flying

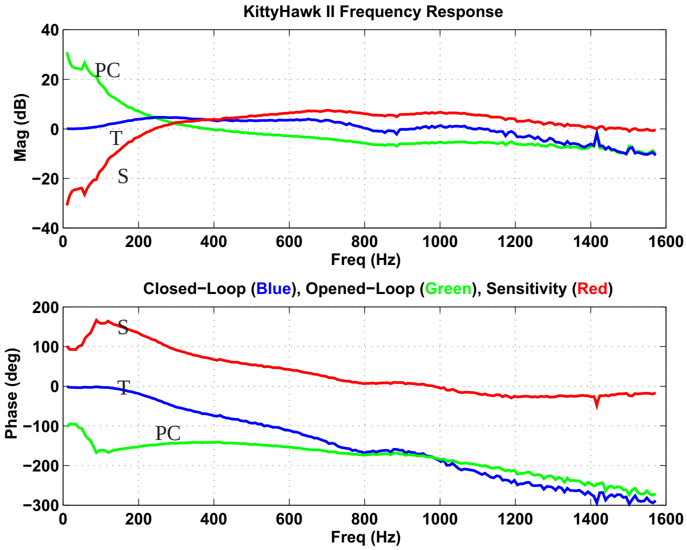


Figure 7.6: Frequency response of a KittyHawk II.

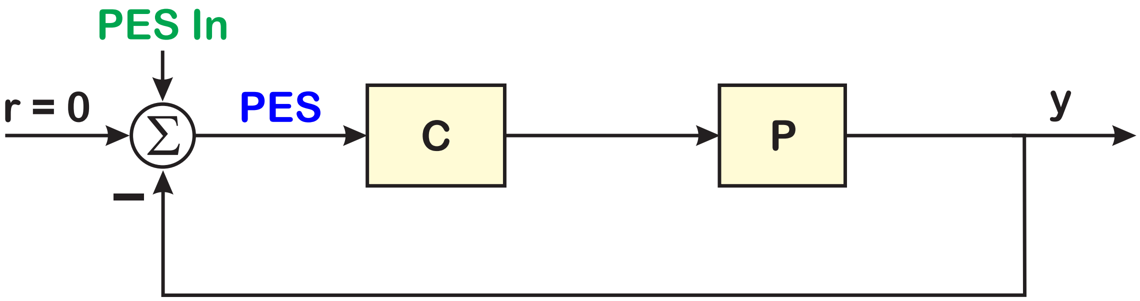


Figure 7.7: Block diagram of original KittyHawk measurement that led to PES Pareto.

blind as to where the noise was actually originating and how much any one source consumed of their noise budget.

It took this author years to realize that this kind of knowhow was not prevalent in the academic controls research community [19] and that the lack of these very utilitarian practices had likely prevented PES Pareto from filtering up to them and then out to their students. For this reason, this tutorial will teach far more about the pedantic intricacies of measurement between domains to give the reader the intuitive familiarity with what has to happen to assemble the measurements and models needed to get a much better picture of noise throughout the loop. It is hoped that by doing this, PES Pareto can become useful to a new generation of control systems engineers, far beyond what remains of the disk drive industry.

This paper will focus on understanding of the effects of noise through the feedback loop, or how to back noise measurements out to their sources. In the legendary "Respect the Unstable" Bode Lecture of 1989 [151, 1] Gunter Stein educated us to the idea that loops do not eliminate noise, they merely move it around, as he brilliantly illustrated with a dirt digging problem, reconstructed from memory in Figure 7.4.

The original measurement that was spawned by the persistent memory of Stein's talk (nine years before the paper [1] was published) was a disk drive problem measuring the Position Error Signal (PES) in HP's KittyHawk 1.3" disk drive in the early 1990s. That measurement is diagrammed in Figure 7.7 and the data displayed in Figure 7.5. The blue Position Error Signal (PES) was considered flat in the frequency domain by the servo engineers at HP's Disk Memory Division (DMD) in the early 1990s. Certainly, the blue measurement of PES in Figure 7.5 looks relatively flat across most frequencies. It was Stein's pile of closed-loop dirt that led to the realization that the blue curve was a closed-loop quantity, that had been filtered by the servo loop. What would that noise look like if it were a signal being injected into the loop, if it were the PES In signal?

The "Aha!" moment was realizing that the filtered, closed-loop PES signal could be quantified in frequency using its Power Spectral Density (PSD) and that PSD looked like what would happen if PES In had been filtered by the magnitude squared of the sensitivity function. If one had a model or measurement of that sensitivity function that covered the same frequency bins as the measurement of PES, one could do some math. Denoting the PSD of PES as $\Phi_{EE}$ and the PSD of PES In as $\Phi_{EE}$, if

$$\Phi_{EE}(f) = \left\|\frac{1}{1+PC}\right\|^2 \Phi_{II}(f), \tag{7.1}$$

then

$$\Phi_{II}(f) = \left\|\frac{1+PC}{1}\right\|^2 \Phi_{EE}(f). \tag{7.2}$$

The resulting green curve in Figure 7.5 and was out of norm with what the engineers had been used to that it took a while to accept as correct. If the noise looked flat after being filtered by the sensitivity function (which has a lot of rejection at low frequency), then the unfiltered curve should have high levels at low frequency. Realizing this launched the PES Pareto methodology.

Even in such a simple diagram as Figure 7.7, it becomes a fundamental set of questions. If we are track following ($r = 0$), what does the noise in PES look like as the input PES In. We get that in Figure

7.5, and in the case of a disk drive, a lot of the broadband noise can be modeled to enter at that point, be in noise in the generation of the PES signal (called Position Sensing Noise, PSN) or air flow over the disk drive heads buffeting the physical position around. However, even in a disk drive, there are far more noise injection points around the loop, as diagrammed in Figure 7.1. If $\Phi_{EE}(f)$ is flat, or no matter what shape it takes, what do the input PSDs of $\Phi_{WW}(f)$ and $\Phi_{VV}(f)$ look like? Furthermore, how do we quantify how much of $\Phi_{EE}(f)$ is due to one versus the other? It is clear that without any other knowledge, it is hard to separate out the effects of $\Phi_{WW}(f)$ versus $\Phi_{VV}(f)$.

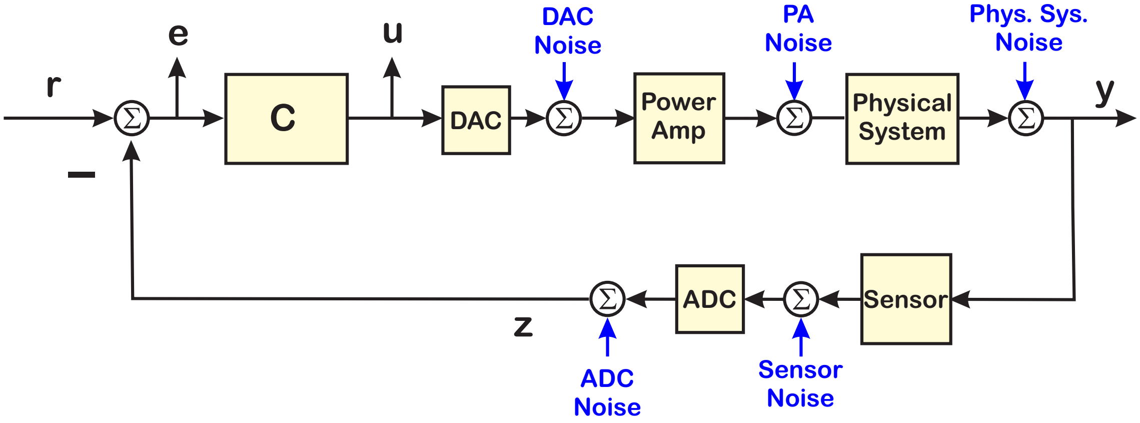


Figure 7.8: Simplification for analysis of Figure 7.2. Closed-loop system with each block having its own noise source as an additive output noise.

In a more complex model where each block can be modeled to have its own noise source (Figure 7.2), how do we:

- Isolate and measure some noise source at some downstream output or measurement point?
- Back up through whatever effective filter there is to get to the particular noise source as an input?
- Push that source (and others) forward through the closed-loop system to see the effects of that noise source on the rest of the loop?

The second two questions are answered by setting up the math:

- We need Power Spectral Densities (PSDs) in a measurement frequency range with consistent frequency bins.

- We need measurements and/or models of all the blocks in such a way that we can match the frequency bins.
- We need to understand the relationship between physical PSDs, the integral across the frequency band, and (via Parseval's Theorem [26]) the noise variance in time, $\sigma^2$.

The first question involves a lot of hands-on cleverness and some fudging, but it is worth it.

- We see right away that in order to isolate some noise sources, we need to open the loop.
- In other cases, we cannot make the measurements without the system being in closed loop.
- Some noises are arrived at when we channel Sherlock Holmes and eliminate all the others as a potential source [186].

The model of Figure 7.2 can become intractable for our analysis, unless we can simplify it to that of Figure 7.8, in which we have taken the block noises and modeled them as individual output noises of the blocks. Furthermore, we have made a reasonable assumption that we can consider the digital noises (apart from quantization) to be negligible, at least in the sense that they can be mitigated via numerical stability methods. In that respect, they are part of any reasonable controller design. Figure 7.8 will serve as a prototype for the type of block diagram that best fits into the PES Pareto environment.

Once we have these noise "sources" and their effect on the loop, we can model the effects of changing one of those sources. This measurement based modeling tells us where to put our system design and control effort. The achievable bandwidth of even the best control algorithm is eventually limited by time delay and noise on a sensor that the loop amplifies at high frequency [19]. In such cases, attention to the sensors, to the critical and dominant noise sources, and perhaps to how to demodulate signals [185], often enables a much higher performance increase than the 10% improvement achieved through 20× the math in the controller.

Finally, it is worth understanding why we focus on broadband noise. In the case of a hard disk drive, the error signal, called the Position Error Signal (PES), can be decomposed in the frequency domain into four components:

**External Shock and Vibrations** are heavily influenced by the drive's operating environment. It has been shown that accelerometer feedforward control can considerably reduce the effect on PES [187, 188].

**Synchronous or Repeatable Excitations** are due to the rotation of the spindle and therefore synchronous with it or one of the spindle orders. While synchronous excitations may be large, standard practice in the disk drive industry includes using feedforward cancelers to reduce the effects of synchronous excitations [189, 190]. More importantly in the hard disk industry, the move of drives from the office into the living room meant that the audio noise caused by the ball bearing spindles were disturbing enough to people viewing their recorded movies that the ball bearing spindles quickly were replaced with fluid bearing spindles [31]. In recent years, feedforward has seen a dramatic increase in popularity with its use in nanopositioning control loops such as atomic force microscopes [191, 192, 193] as well as macro-positioning control such as wind turbines [194].

**Non-synchronous or Non-repeatable Excitations** include sharp spectral peaks due to spindle bearing cage orders and structural resonances (which are less sharp but still narrow band). There was work suggesting that disturbances due to resonances or cage orders could be considerably reduced by the use of damped disk substrates and fluid bearing spindles [195, 196, 197]. As mentioned above, this happened for other reasons. In applications beyond rotating machinery, these are also minimized via physical redesign. Alternately, loop gain can be raised over a narrow portion of the pass band with a bump filter, or dropped outside the passband). In these cases, if the band is narrow enough, the area of noise amplification can be kept relatively small.

**Broadband or Baseline Noise** is what remains when all of the narrow band components have been removed. Of the four categories, baseline noise has received the least attention in the literature. Therefore, broadband noise became the focus of PES Pareto. Broadband noise was not viewed as something that could be managed via repetitive or other feedforward control methods. By its very nature, it could not be minimized with narrowband filters. When combined with Stein's revelation about Bode's Integral Theorem, understanding the effects broadband noise gained much higher priority.

Consider that if signals including noise are filtered by closed-loop (CL) dynamics to get to PES (the error signal), then inverse filtering by closed-loop filter dynamics should give us a "reference" or "noise" input. How do we filter noise? It is better to ask what kind of noise can be analyzed through a filter? The answer is additive, white, Gaussian noise (AWGN). AWGN has the property that auto and cross spectra can be analyzed when passed through a linear filter [75]. This means that if we can generate frequency responses for our closed loop dynamics, generate magnitude squared filters, and invert those, we can back out noise sources. We need to keep in mind that each source injection point has it's own back filter from the measurement point, and each measurement location has it's own forward filter from any injection point.

This became the basis for the PES Pareto methodology of analyzing the effects of noise on a system

[34, 35, 36, 37]. While that work was focused on the control of hard and optical disk drives, this tutorial will attempt to give participants a view to applying this method to all manner of control problems.

The structure of this half of the chapter is as follows. Section 7.5 will review how Bode's Integral Theorem and its discrete-time counterpart motivate the study of noise passing through a feedback loop and outline the methods for doing this. Section 7.6 will establish a common mathematical framework that will motivate and underpin all of our measurements and the rest of our analysis. Section 7.7 will provide an old but real example system, illustrates the methods. Section 7.8 will discuss the "Simple Tricks and Nonsense" [10] needed to get actual noise measurements out of a physical system, using the two examples from Section 7.7. Section 7.9 will illustrate backing closed-loop noise measurements back to their open-loop sources, and then propagating them forward to the loop error signal so that they can be ranked by their effects on that error. Section 7.10 will present some examples of how these new measures can be used to evaluate the effects of a particular noise source increasing or being minimized. Finally, Section 7.12 will discuss methods of minimizing the effect of noise sources before they enter the feedback loop [185].

## 7.5 Bode's Theorem and Noise Shaping

Bode's integral theorem [158] deals with what Bode calls regeneration, and dates back to the 1940s. In the years since Stein's Bode Lecture [151], it has gained prominence in the controls community, as this theorem has significant implications for and applications to control systems.

The sensitivity function, $S$, is also known as the disturbance rejection function because it shows how disturbances, $d$, go through the system and show up at the output, $y$, or at the error signal $e$.

$$S = \frac{e}{r} = \frac{1}{1+PC} = \frac{y}{d} = -\frac{e}{d}. \tag{7.3}$$

While the mathematics used to prove both versions of Bode's Theorem can be fairly complicated, the result is fairly simple and extremely powerful. We will leave the proofs to the references [158], [161] and talk simply about the interpretation. Looking at Figure 7.9 it says simply that:

$$\begin{array}{c}\text{the area of}\\ \text{disturbance}\\ \text{amplification}\end{array} = \begin{array}{c}\text{the area of}\\ \text{disturbance}\\ \text{rejection}\end{array} + \begin{array}{l}\text{a non-}\\ \text{negative}\\ \text{constant.}\end{array} \tag{7.4}$$

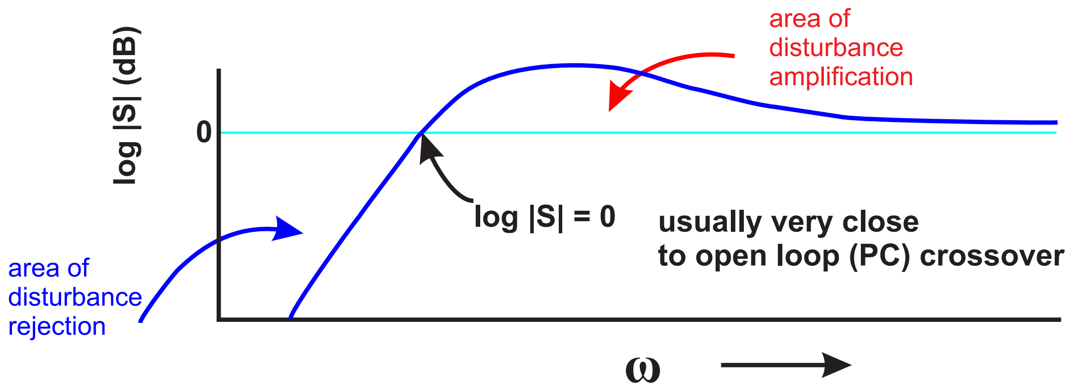


Figure 7.9: Sensitivity function drawing.

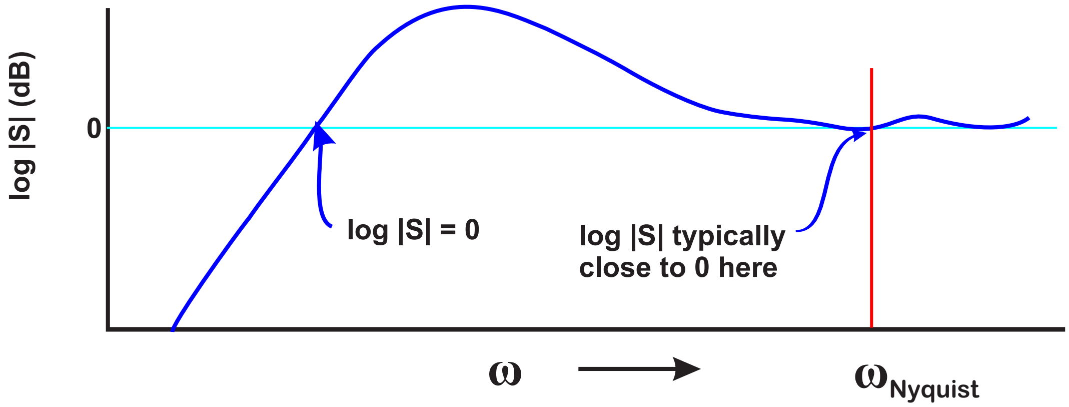


Figure 7.10: Drawing of sensitivity function in discrete time.

Mathematically, this is stated as

$$\int_0^\infty \log |S| d\omega = c, \tag{7.5}$$

where $c$ is some positive constant dependent only on the open loop unstable poles and non-minimum phase zeros.

Consequences: “Sooner or later, you must answer for every good deed.” (Eli Wallach in the The Magnificent Seven)

Translation: If you make the system less sensitive to noise at some frequencies, you then make the system more sensitive at other frequencies.

Typical control designs attempt to spread the increased sensitivity (noise amplification) over the high frequencies where noise and/or disturbances may be less of an issue. The image of this was provided in the Bode Lecture at the 1989 IEEE Conference on Decision and Control (Tampa, FL)[151]. The talk, by then Honeywell Researcher and MIT Professor, Gunter Stein, was entitled "Respect the Unstable." Stein described the net effect of control systems design as trying to get a certain amount of disturbance rejection at some frequency span while trying to thinly spread the amplification over a large frequency span. Stein's drawing had a guy shoveling disturbance amplification "dirt" as in Figure 7.4. The dirt can be moved, but not eliminated. Furthermore, the discrete-time version of Bode's Integral Theorem [161] has some implications for discrete time systems [35], however they are essentially those of the continuous time theorem with the Nyquist rate forming a retaining wall for the disturbance amplification dirt 7.10.

There are two reasons why Bode's Integral Theorem is important in a discussion of a feedback loop's error signal. First of all, it gives a very good gage on what can and cannot be done with disturbance rejection and noise in a control system. An intelligently designed control system puts noise amplification in places where there is little noise. A poorly designed system results in significant noise amplification.

The second reason will become apparent in Section 7.8 on making measurements. It turns out that when the error signal is measured from a closed loop system, the loop should actually be opened up to look at the error's contributing sources. The exact same effects that are the point of the above theorem affect a measurement of error in closed-loop. Before we can do any of this, we need to establish a common mathematical framework that will motivate and underpin all of our measurements and the rest of our analysis. Basically, we need the machinery to do what was discussed above. This will be in Section 7.6.

## 7.6 Noise Analysis and PSDs

The PES Pareto methodology requires measuring, averaging, and isolating the spectra of signals at different points in a feedback loop and then filtering them in different ways to get to input noises and output noises. It goes without saying that all of the operations need to be done on the same frequency axis. Whether a spectrum is measured using a spectrum analyzer, or generated from a time trace, the frequency bins (width, count, and location) have to be the same. The models of the different loop components that “filter” the spectra must also have the same frequency axis so that the filtering can be done bin-by-bin. This requires some rethinking of our measurements. Frequency response functions are often measured on a logarithmic frequency scale since control engineers are used to plotting frequency response functions using logarithmic frequency. Spectrum analyzers typically generate spectra on a linear frequency axis, so our frequency response functions (even if we are generating them from an analytic model) really need to be generated on this same linear frequency axis. Calculating spectra from time measurements – usually via FFT calculations – also are usually done on a linear frequency scale. Generally, the most reasonable way to do this is to generate an analytic model of the different loop components from our measurements. That analytic model can then be evaluated at the frequency points of the spectrum measurements. This brings in all the difficulties of extracting models from measurements [19, 87], but is a needed step.

### 7.6.1 Useful PSDs from Measurements

Our goal in this section is to describe how to measure Power Spectral Densities, either directly from a spectrum analyzer, or from one or more time domain measurements that have been transformed via a Discrete Fourier Transform (DFT) or Fast Fourier Transform (FFT). As mentioned above, we need these spectra measurements to have the same number and distribution of bins in the frequency domain. The easiest way to get a spectrum measurement of a signal is with a spectrum analyzer, an instrument that does this computation automatically . Spectrum analyzers are nice because they essentially package all the necessary computations. The modern ones all compute FFTs [198, 199], while some of the older ones computed a spectrum by sweeping an narrow band Gaussian filter across the frequency range and simply using the output of that filter as the spectral value at that frequency point [200].

The issue with any spectrum analyzer is that they have a fixed sample frequency and typically have a fixed number of frequency bins at which they evaluate the spectrum. It is also often difficult to extract from their documentation exactly what scaling is used in their FFT calculations, as the equations for computing FFTs are scaled differently in different devices. Let’s get back to basics so we can have a unified understanding of all these measurements.

The Fourier Transform of a signal $x(t)$ is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi f t} dt, \tag{7.6}$$

while the inverse Fourier Transform becomes

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi f t} df. \tag{7.7}$$

In some cases, the authors use $\omega$ in place of $f$. This doesn't affect the first integral (which is over the time variable, $t$)

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt, \tag{7.8}$$

but in integrating over $\omega$ in place of $f$ we need to factor out the $2\pi$, so the inverse Fourier Transform becomes

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(f)e^{-j\omega t} d\omega. \tag{7.9}$$

If only a finite data record of time length $T$ exists then the finite length Fourier Transform is

$$X(f, T) = \int_{0}^{T} x(t)e^{-j2\pi f t} dt. \tag{7.10}$$

We should note that for any practical measurement, only a finite data record of time length $T$ will ever exist, so to apply Fourier Transforms in real life, we need to make use of Equation 7.10.

If that signal is sampled with a sampling period of $\Delta t$ then the sequence that results is

$$x_n = x(n\Delta t) \qquad\qquad n = 0, 1, 2, \ldots N - 1 \tag{7.11}$$

and Equation 7.10 can be recast as the discrete Fourier Transform:

$$X(f, T) = \Delta t \sum_{n=0}^{N-1} x(n)e^{-j2\pi f n\Delta t}. \tag{7.12}$$

By letting $f_k = \frac{k}{T} = \frac{k}{N\Delta t}$ in Equation 7.12 we get

$$X(k) = \frac{X(f_k)}{\Delta t} = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi nk}{N}}. \tag{7.13}$$

Let $W_N = e^{\frac{-j2\pi}{N}}$ and $\tilde{W}_N(u) = e^{\frac{-j2\pi u}{N}}$. Then Equation 7.13 can be written as

$$X_k = X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = \sum_{n=0}^{N-1} x(n)\tilde{W}_N(kn). \tag{7.14}$$

This is what a standard FFT, including the one in Matlab computes. Note that

$$X(f_k) = \Delta t X(k) \tag{7.15}$$

which returns the FFT to something closer to the physical units. From this definition of the FFT, the inverse FFT is given by

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{\frac{j2\pi kn}{N}} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-kn}. \tag{7.16}$$

Note that the placement of the $\frac{1}{N}$ is arbitrary. However, it is significant in trying to return the FFT calculation to physical units. Alternate FFT definitions are available as:

$$\tilde{X}_k = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x_n W_N^{kn} \Longleftrightarrow x_n = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} \tilde{X}(k)W_N^{-kn} \tag{7.17}$$

or

$$\hat{X}_k = \sum_{n=0}^{N-1} x_n W_N^{kn} \Longleftrightarrow x_n = \frac{1}{N}\sum_{k=0}^{N-1} \hat{X}(k)W_N^{-kn}. \tag{7.18}$$

This is generally a pain because we want physical units when measuring signals in the lab and the physical units do not have arbitrary scaling.

In order to make any of this analysis self-consistent, we need to have all the FFTs computed with the same scaling. This means that we need to know what any spectrum analyzer or digital oscilloscope is using as its primary equation, and this often involves digging through the middle pages of the manuals. Beyond that, we need to consider the sample period ($\Delta T = T_S = 1/f_S$) used in each measurement, and the number of points in each measurement as these two factors define the frequency bins available in an FFT calculation.

So, whether by spectrum analyzer, by digital oscilloscope, or by using our digital control system to capture a block of data, we understand that we have a linear spectrum of the data. Considering the spectrum analyzer as more of a corner case moving forward, we might use a digital scope or a time capture feature of our digital control system to capture a long stretch of sampled data and use a Digital Fourier Transform (DFT) or Fast Fourier Transform (FFT) to compute the linear spectrum of that signal. No matter how they are computed linear noise spectra do not pass through filters in an analytical way, so we need to generate the complex conjugate of this spectra, multiply it times the original at each frequency point, and then normalize it to get a power spectral density. This gives us a Power Spectral Density (PSD). Section 7.6.2 will discuss some details about this calculation.

### 7.6.2 Power Spectral and Cross Spectral Densities

This section deals with the practical generation of auto and cross spectra from spectra generated by Fourier Transforms (or FFTs) or Fourier Series calculations. When the cross or auto spectrum is normalized by the bandwidth of the measurement, we get a spectral density. Because we are making discrete time measurements with a sample period, $\Delta t = T_S = 1/f_S$, our measurement bandwidth is from $-f_S/2$ to $f_S/2$. Thus, we end up normalizing by the sample frequency, $f_S$. A common term for the auto-spectral density is power spectral density (PSD). Cross spectral densities may be referred to as CSDs. Note that if we assume the same sample frequency for all of our FRF measurements, then we can use the FRFs to filter the spectra in each frequency bin.

For PES Pareto, where we are backing noises out to the point where they are independent inputs, we can consider the CSDs of these noises to be $0$. Thus, we really care about the Power Spectral Density (PSD).

Let's consider that we have made a measurement sampled in time at intervals of $\Delta t = T_S$, where there are at least $N$ points in the measurement and $N$ is a power of 2. From this time measurement, we could compute the FFT on the range from $-\frac{f_S}{2}$ to $\frac{f_S}{2}$ using Equation 7.13. We scale this to physical units via Equation 7.15. To produce the PSD from $X(f)$, we compute the complex conjugate $X^*(f)$. This is a fairly straightforward computation. At that point we compute the element by element product of the two complex vectors:

$$PSD(x) = \frac{X^*(f)X(f)}{B_e} \tag{7.19}$$

where $B_e$ is the Resolution Bandwidth of the filter used to compute the spectrum (or the Noise Equivalent Bandwidth which is technically not the same but very close to the Resolution Bandwidth) and where $X(f)$ is the Fourier Transform from Equation 7.6. This is the smallest change in frequencies that a given measurement can resolve.

In general $B_e$ is inversely proportional to the length of the time window over which a measurement is made , *i.e.*,

$$B_e = \frac{1}{T} \tag{7.20}$$

where $T$ is the length of the time record. For an FFT,

$$B_e = \frac{1}{T} = \frac{1}{N\Delta t} \tag{7.21}$$

where $N$ is the number of points in the FFT and $\Delta t$ is the sample period between points.

Note that for an FFT, the resolution bandwidth is fixed as all the integrations are done over a single period of time ($N\Delta t$, as in Equation 7.21. Band Selectable Fourier Analysis or Zoom-FFT [82, 83] can be used to maximize the resolution, but this is usually only known to experts. In a brute force spectrum calculation, we could computes a separate integral for each frequency. To eliminate errors due to a partial period integral, the integration should be done over an integer number of periods of the frequency in question. This would that except in special cases, the actual resolution bandwidth of the calculations at different frequencies will differ slightly, but the spectral leakage would be largely eliminated.

MATLAB computes the FFT in Equation 7.14 and (with some details to be filled in later) then produces

$$P_{xx} = PSD(x) = \frac{X^{*}. * X}{N} \tag{7.22}$$

where $X^{*}$ is the complex conjugate of $X$ and $N$ is the number of points in the FFT and the $.*$ operation is the element by element multiply of two same-sized vectors in MATLAB . (Windowing and scaling are standard methods of improving the performance of FFTs by driving the time signal to $0$ at the beginning and end of the data run, but we will not discuss those here.)

**Note that these units are not physical.** From Bendat & Piersol[75], page 407 there is a procedure for computing a PSD from FFT based measurements. At any frequency, $f_k$, the PSD of a signal $x$ is given by:

$$\tilde{P}_{xx}(f_k) = \frac{X^{*}(f_k)X(f_k)}{N\Delta t} \tag{7.23}$$

where $X(f_k) = \Delta t X_k$. This means that

$$\tilde{P}_{xx}(f_k) = \frac{(\Delta t)^2 X_k^{*} X_k}{N\Delta t} \tag{7.24}$$

or

$$\tilde{P}_{xx}(f_k) = \frac{\Delta t X_k^{*} X_k}{N} \tag{7.25}$$

so

$$\tilde{P}_{xx} = \frac{\Delta t X^{*}. * X}{N} \tag{7.26}$$

and thus

$$\tilde{P}_{xx} = \Delta t P_{xx} \tag{7.27}$$

i.e. to go from MATLAB units to physical units, multiply the MATLAB PSD by $\Delta t$. Now, the MATLAB PSD function has been deprecated, and now they favor a Welch algorithm that does the physical scaling found in Equation 7.27. Alternately, one can use MATLAB 's periodogram function. So, for a time domain measurement in the vector, d, of length $N_{meas} \leq N = NFFT = 2^{M}$, we can:

- Compute from MATLAB 's FFT routine:

$$
\begin{aligned}
d.FFT &= fft(d, NFFT); && (7.28)\\
d.FFT_{phys} &= T_S * d.FFT; && (7.29)\\
d.PSD_{FFT} &= \frac{(d.FFT)^{*}. * d.FFT}{NFFT}; && (7.30)\\
d.PSD_{FFT,T_S} &= T_S * d.PSD_{FFT}; && (7.31)
\end{aligned}
$$

- Compute using MATLAB 's Welch routine:

$$[P_{xx,w}, f_w] = pwelch(d, NFFT, [], NFFT, fs); \quad (7.32)$$

- Compute using MATLAB 's periodogram routine:

$$d.PSD_{Per} = periodogram(d, [], NFFT); \quad (7.33)$$

These three methods give the same PSD and give a PSD that is scaled to be equivalent to the Fourier transform of an analog signal. We believe that for most purposes, the first method has the advantages that it is relatively straightforward and completely transparent. Furthermore, the routines are easily mimicked in computer languages besides MATLAB or Python, making the math far more portable.

The FFT calculations above all compute an FFT from $-\frac{f_S}{2}$ to $\frac{f_S}{2}$ but since we really can only make use of the positive frequency axis, we want to use the FFT frequency bins from $0$ to $\frac{f_S}{2}$. The Fourier Transform of real function is Hermitian [26], which means that the real part is even and the imaginary part is odd, but when we multiply the FFT by its complex conjugate, we get a real and even function on the frequency axis. We can then take double the values of the PSD for the positive frequency bins, leave the bin at DC the same, and discard the portion with the negative frequency bins.

Note that it is typical to pick the next power of $2$ above $N_{meas}$ to compute the FFT. However, we may have a situation where we have a lot of data i.e. $N_{meas}$ is far larger than any reasonable FFT we might wish to make. This means that we have the option to do some averaging. Averaging is a good thing in noise analysis because – to put it simply – one cannot get an expected value from a single vector of data, and one cannot approximate an expected value without a bit of averaging. If one makes a long data measurement, and one assumes that the noise process is ergodic, that is that the time averages are the same as the ensemble averages [75], then we can break a long measurement into sections to be FFTed and those FFTs can be averaged. One of the tricks one could do with the old MATLAB PSD algorithm was perform overlap processing: a long run of data could be broken in to multiple segments for FFTs, but those segments could overlap giving the illusion of more points while still averaging, as diagrammed in Figure 7.11.

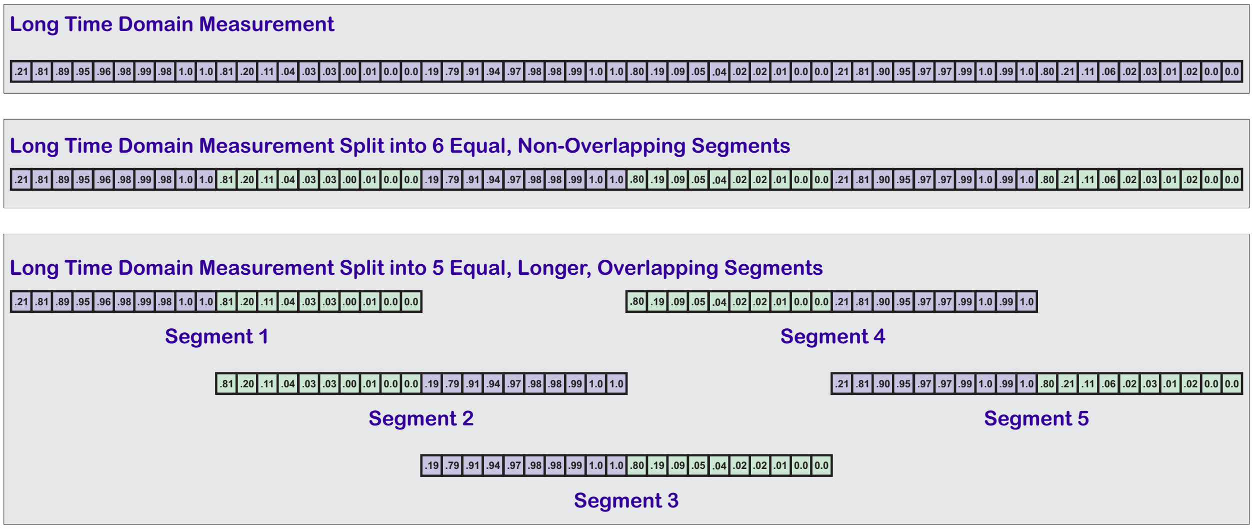


Figure 7.11: Diagram of overlap processing.

### 7.6.3 Quantization Noise: The Widrow Model and Others

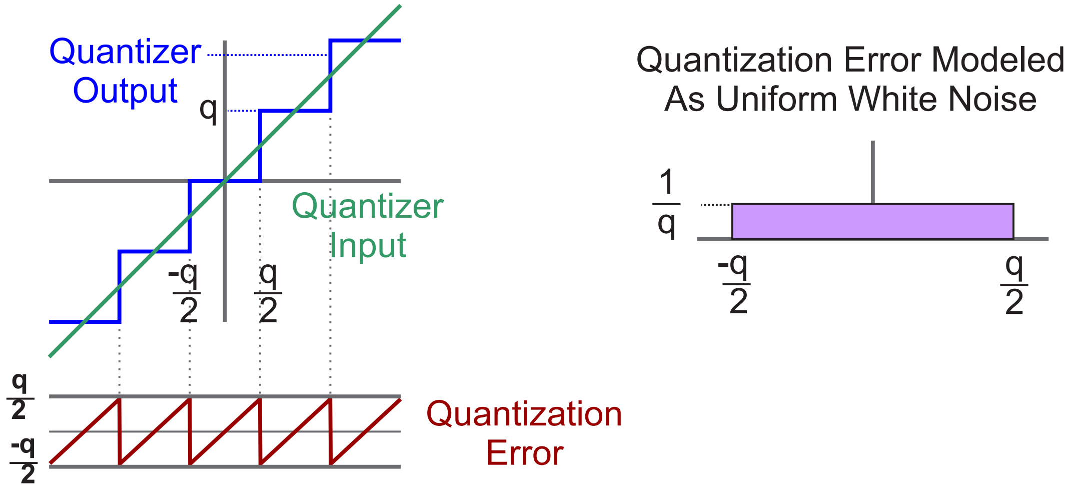


Figure 7.12: Diagram of quantization and Widrow model.

There are other time domain measurements that yield only a single number, such as the variance due to quantization in the Widrow model [38]. This variance must be spread across the frequency band in some logical way, so the authors chose to normalize it by the frequency bandwidth. This is consistent with the texts [201, 202] on quantization devices and with will be described briefly here. Note that [202] is erroneous in that the normalization by $T_S = 1/f_S$ has been omitted.

The Widrow model [38] of quantization is based on an analog of the Nyquist sampling theorem [203]. A conceptual quantizer is shown on the left side of Figure 7.12. Quantization is not a random process, but a deterministic, nonlinear operation. This makes it hard to do anything with passing the math through a filter. Widrow's insight was that while quantization was deterministic and nonlinear, if the signal excited enough of the scale of the quantizer, and the quantization bins were fine enough, that the probability of the quantized signal falling anywhere in the quantization bin could be modeled as a uniform density white noise on the interval $[-\frac{q}{2}, \frac{q}{2}]$, where $q$ is the size of a minimum quantization interval. This is displayed on the right side of Figure 7.12. With this model, computing the mean and the variance of the quantization "noise" reveals that the mean and variance are:

$$\mu_q = 0 \text{ and } \sigma_q^2 = \frac{q^2}{12}. \tag{7.34}$$

This number for variance is used in texts all over the world [15]. That is all well and good, but how do we translate this variance into a PSD that we can pass through filters?

The leap of faith here is to treat that Additive Uniform White Noise as an Additive Gaussian White Noise, with zero mean and variance of $\frac{q^2}{12}$. As both are white, they have an autocorrelation that is only nonzero at an offset of $0$. At an offset of $0$, the autocorrelations are a Delta functions with a hight equal to the variance. We know from Parseval's Theorem [26, 204] that the variance in time is equal to the integral of the PSD over frequency, in this case, from $-\frac{f_S}{2}$ to $\frac{f_S}{2}$, so we can as a first approximation, give the quantization noise a uniform PSD on the interval $[-\frac{f_S}{2}, \frac{f_S}{2}]$ with magnitude $\frac{q^2}{12 f_S}$. In summary, we set the noise PSD due to quantization as:

$$\Phi_{QQ}(f) = \frac{q^2}{12 f_S} = \frac{q^2 T_S}{12}, \text{ using } f \in [-\frac{f_S}{2}, \frac{f_S}{2}] \tag{7.35}$$

If we are only using the interval from $[0, \frac{f_S}{2}]$, then the PSD magnitude is $\frac{q^2}{6 f_S}$.

$$\Phi_{QQ}(f) = \frac{q^2}{6 f_S} = \frac{q^2 T_S}{6}, \text{ using } f \in [0, \frac{f_S}{2}] \tag{7.36}$$

This is a correction to the classic circuit text [202] where the noise PSD is set to $\frac{q^2}{12}$ from $f \in [0, \frac{f_S}{2}]$. The key to understand is that the integral of the PSD over the frequency range, must equal the total variance of $\frac{q^2}{12}$.

The swap between a uniform and Gaussian additive white noise is not perfect, but it does allow us to deal with quantization noise in the PES Pareto method, i.e. to pass it through digital filters. We will discuss how to apply these ideas to measurements of ADC and DAC "noise" in Section 7.8.4. This model serves us as a theoretical minimum for the quantization noise, if all of the noise in the ADC or DAC noise and can be used to generate a PSD as described above.

Other measures of ADC and DAC noise depend upon the access to the system. If one can open up the system and drive the ADC with either a single tone, or with an open circuit, one can establish (for the ADC) an input noise level. Applying an open circuit to the ADC means that all that is coming in is noise. However, this doesn't exercise the full range of the ADC. An alternate input is to drive the ADC with a single sinusoid that excites the full range of the ADC. The issue then is that the tone dominates the response, so it needs to be removed digitally. This can be done in the time domain with an adaptive noise canceler [24], which adapts coefficients of a generated sine and cosine to cancel the magnitude and phase of the input signal. If done after an FFT, we need to look at the frequency bins that contain the tone and set it equal to some average of side bins outside the skirt of the tone. If the tone is removed in the time domain, we can also remove the DC level by averaging the entire data run to establish the sample mean. For an N-sample measurement,

$$\bar{x}_{k,N} = \frac{1}{N}\sum_{i=0}^{N-1} x_{k-i}, \tag{7.37}$$

and we proceed as above.

We can also use this to back out a measure of how many bits we get out of the ADC. Consider if we have an N-bit ADC that spans a particular full voltage scale, FS. Then

$$\frac{FS}{q} = 2^N \text{ or } q = \frac{FS}{2^N} \text{ or } FS = q2^N. \tag{7.38}$$

Now using (7.34) we can substitute:

$$\sigma^2_{meas} = \frac{q^2_{eff}}{12}, \text{ so} \tag{7.39}$$

$$q_{eff} = \sigma_{meas}\sqrt{12} = \frac{FS}{2^{N_{eff}}}. \tag{7.40}$$

This means that

$$2^{N_{eff}} = \frac{FS}{\sigma_{meas}\sqrt{12}}, \text{ so} \tag{7.41}$$

$$N_{eff} = \log_2\left(\frac{FS}{\sigma_{meas}\sqrt{12}}\right) \text{bits.} \tag{7.42}$$

Equation 7.42 is not useful for PES Pareto per se, but does give an answer to the question of how many bits we are getting out of our ADC. This is an almost religious bone of contention with analog circuit designers and mixed signal device engineers, who have a handful of measurements that they like to use (none of them useful for PES Pareto). However, since anyone applying the above methods will likely hear from a member of those two groups that the measure is wrong, it is good to understand what they use and how they related. There are essentially three popular measures:

**Effective Resolution** is a measure that is easy to compute and defined as:

$$\text{Effective Resolution} = \log_2\left(\frac{FS}{\sigma_{meas}}\right) \text{bits.} \tag{7.43}$$

While this is easy enough to compute, as we should know the full scale and can measure $\sigma_{meas}$, it uses a peak-to-peak signal $FS$ and compares to an Root Mean Square (RMS) signal, $\sigma_{meas}$.

**Signal-to-noise ratio (SNR)** is a measure commonly understood by engineers as some ratio of signal power to noise power. Both of these are RMS signals. For the signal, the assumption is to use a sinusoid to represent the signal. The RMS amplitude of a sinusoid of amplitude, $A$ is $\frac{A}{\sqrt{2}}$, but a sinusoid that spans the entire ADC range would mean $FS = 2A$ so $A = FS/2$

$$\text{SNR} = \frac{FS_{RMS}}{\sigma_{noise}} = \frac{FS}{2\sqrt{2}\sigma_{noise}}. \tag{7.44}$$

What happens is that many analog engineers will use the $\sigma_{noise}$ from the Widrow model, so

$$\begin{aligned} \text{SNR}_q &= \frac{FS}{2\sqrt{2}\left(\frac{q}{\sqrt{12}}\right)} = \frac{FS\, 2\sqrt{3}}{2\sqrt{2}q} = \frac{FS}{q}\sqrt{3/2}, & (7.45)\\ &= \frac{q2^N}{q}\sqrt{3/2} = 2^N\sqrt{3/2}. & (7.46) \end{aligned}$$

Furthermore, because engineers like SNR in decibels, they compute:

$$\begin{aligned} \text{SNR}_{q,dB} &= 20\log_{10}\left(2^N\sqrt{3/2}\right) & (7.47)\\ \text{SNR}_{q,dB} &= N20\log_{10}(2) + 20\log_{10}\sqrt{3/2}. & (7.48) \end{aligned}$$

Now, $20\log_{10}(2) \approx 6.02$ and $20\log_{10}(\sqrt{3/2}) = 10\log_{10}(3/2) \approx 1.76$ which gives the equation often seen in the device literature:

$$\text{SNR}_{q,dB} \approx 6.02N + 1.76. \tag{7.49}$$

From there, one could back out $N$ with malice a forethought as:

$$N = \frac{\text{SNR}_{q,dB} - 20\log_{10}\sqrt{3/2}}{20\log_{10}(2)} \approx \frac{\text{SNR}_{q,dB} - 1.76}{6.02}. \tag{7.50}$$

What the device literature usually neglects is that (7.49) only holds when we attribute all of the noise in the device to quantization and use the Widrow model. What happens when there are other noises and distortions? For that, the device engineers use a term called SINAD for SIgnal to Noise And Distortion ratio. The most useful definition seems to be

$$\begin{aligned} \text{SINAD} &= \frac{\sqrt{P_{signal}}}{\sigma_{tot,n}}, \text{ where} && (7.51) \\ \sigma_{tot,n} &= \sqrt{\sigma_q^2 + \sigma_{n1}^2 + \ldots + \sigma_{nm}^2}. && (7.52) \end{aligned}$$

Thus we take the ratio of our pure signal to all the noises and distortions (assuming that they are independent) that we can find or measure. Using SINAD in place of SNR and the total noise in place of quantization noise in Equations 7.48 and 7.50, they arrive at what they call **Effective Number of Bits (ENOB)**:

$$\begin{aligned} \text{SINAD}_{dB} &= ENOB \cdot 20\log_{10}(2) + 20\log_{10}\sqrt{3/2}. && (7.53) \\ ENOB &= \frac{\text{SINAD}_{dB} - 20\log_{10}\sqrt{3/2}}{20\log_{10}(2)}, \text{ or} && (7.54) \\ ENOB &\approx \frac{\text{SINAD}_{dB} - 1.76}{6.02} \text{ bits.} && (7.55) \end{aligned}$$

The issue with ENOB, which is considered the gold standard by many device engineers is that the $20\log_{10}(\sqrt{3/2}) \approx 1.76$ is based on all the noise being due to quantization and then they count quantization again in the SINAD calculation. This double counting seems to be based more on what could easily be measured from traditional instruments rather than reworking the math. Because of that, there is also a division by $20\log_{10}(2) \approx 6.02$ instead of simply computing $\log_2(\cdot)$ in place of $\log_{10}(\cdot)$.

A "harder to compute on ancient instruments but trivial in MATLAB " correction for this might be Effective Bits (EB). Staring with the SINAD definition of Equation 7.51 and realizing that it is simply the SNR of Equation 7.44

$$\text{SINAD} = \text{SNR}_{tot} = \frac{FS_{RMS}}{\sigma_{tot,n}} = \frac{FS}{2\sqrt{2}\sigma_{tot,n}}, \quad (7.56)$$

where $\sigma_{tot,n}$ is either measured or computed using (7.52). Since $FS = q2^N$ (7.38) we have

$$\text{SINAD} = \text{SNR}_{tot} = \frac{q2^N}{2\sqrt{2}\sigma_{tot,n}}. \quad (7.57)$$

We now replace $N$ by $EN$, analogously to (7.53), to get

$$\begin{aligned} 2^{EN} &= \frac{(\text{SINAD})2\sqrt{2}\sigma_{tot,n}}{q} && (7.58) \\ EN &= \log_2\left(\frac{(\text{SINAD})2\sqrt{2}\sigma_{tot,n}}{q}\right) \text{ bits.} && (7.59) \end{aligned}$$

We can, of course, simplify this with

$$EN = \log_2(\text{SINAD}) + 1.5 + \log_2(\sigma_{tot,n}) - \log_2(q). \tag{7.60}$$

To put it in the less logical but more familiar framework of decibels, we would have:

$$\begin{aligned} SINAD_{dB} &= EN20\log_{10}(2) + 20\log_{10}(q) - \\ & \quad 20\log_{10}(2\sqrt{2}) - 20\log_{10}(\sigma_{tot,n}). \end{aligned} \tag{7.61}$$

From here

$$EN = \frac{SINAD_{dB} - 20\log_{10}(q) + 20\log_{10}(2\sqrt{2}\sigma_{tot,n})}{20\log_{10} 2}, \tag{7.62}$$

or using the approximations from before and $20\log_{10}(2\sqrt{2}) \approx 9.03$

$$EN = \frac{SINAD_{dB} - 20\log_{10}(q) + 9.03 + 20\log_{10}(\sigma_{tot,n})}{6.02}. \tag{7.63}$$

This is an awful lot to go through for measures that do not help PES Pareto, but they do help in understanding the terms used by circuit and device engineers, especially when we need to explain to them that their measures do not help us move quantization through a block diagram.

### 7.6.4 Using PSDs in PES Pareto

With all these tricks, one ends up being opportunistic, generating as many different spectra from as many different locations as possible, with the loop open whenever possible, so as to isolate different noise sources. Because PSDs can be added, they can be subtracted from each other and so by a process of elimination, we arrive at isolated noise sources. However, there are measurements that can only be made when the system is in closed-loop. The measurement of PSN is one such case. In this case, we must use subtraction of the PSD from the sources that we could isolate to leave us with the PSD of the remaining source.

As we try to convert any measurement into a PSD, the following is useful to keep in mind:

- We will be working on the waveform. That is, we will have a frequency axis from some lower frequency to some higher frequency. That higher frequency will be no more than the Nyquist frequency.
- In order to do waveform math on multiple waveforms (scalar math one frequency bin at a time), all waveforms must share the same frequency axis.

- If we have an instrument, or instrumentation functionality that can measure the PSD directly, we can use that. If the instrument can measure the FFT of the signal, then we must multiply it, frequency bin by frequency bin, times its complex conjugate. If we simply have a single time domain variance number, then this must be distributed across the frequency spectrum as described in discussion of quantization in Section 7.6.3.
- PSDs generated by independent processes can be added. Thus, if we have different, independent noise sources generating the PSDs at a particular measurement point, then the results of those sources can be considered added at the measurement point. We have a strata of noise contributors.
- By model or measurement, generate frequency response functions (FRFs) for the loop components on the same frequency axis as our PSD measurements. Generically call these $H$ for this discussion.
- Multiply $H$ times its complex conjugate to get $||H||^2$.
- Multiply $||H||^2$ times the power spectrum or PSD to get effect of the loop on noise. Note that $||H||^2$ must be the appropriate units for filtering the PSDs.
- The resulting output is another power spectrum or PSD.
- We can use superposition to built up contributions from many sources.
- We need to do some "loop unwrapping" to extract proper input noise levels for model.

The chief restriction of manipulating PSDs is that we will have to limit ourselves to linear models of the system. However, by doing so we are able to actually add and subtract PSDs. In order to do so, we formally should require some knowledge that the noise sources are independent. It turns out that there is no way to verify this for all sources, but it is very likely true. While any measured signal in the loop is correlated to several noise sources, each source arises from an independent physical phenomenon. Furthermore, without allowing for superposition of noise measurements, it would be next to impossible to analyze the noise of a measured system. Thus, it is a starting point we must choose.

## 7.7 Using the HDD Example Guide Us

When we introduced the PES Pareto methodology in the 1990s, we were working in the context of the disk drive industry and had plenty of measurements from actual disk drives. In the years since then, there have been few chances to measure more disk drives. That being said, the measurements from

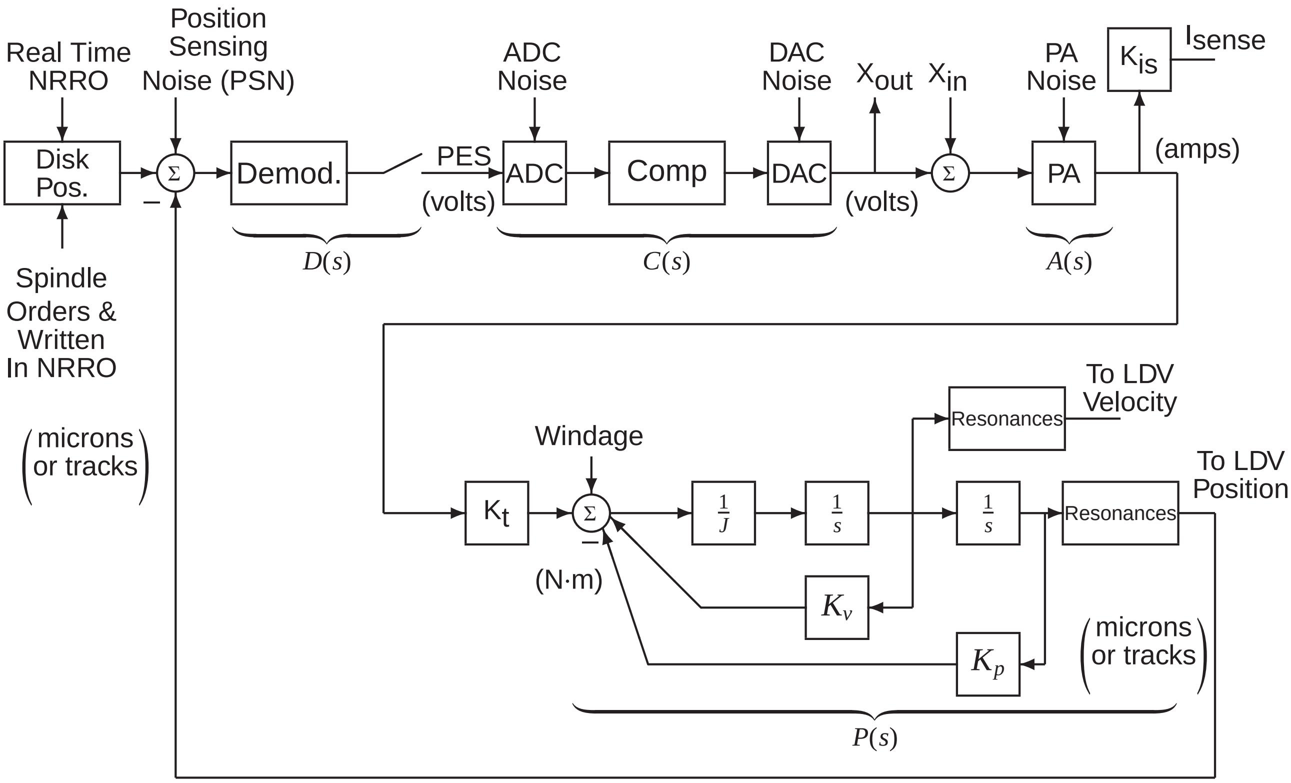


Figure 7.13: Generalized view of track following model in an HDD. Each block can be considered both a source of noise and affected by noise. PSN refers to Position Sensing Noise, that is the inaccuracies caused in trying to extract position information from alternating polarities of magnetic domains.

that original work are still quite instructive. We will reference the block diagram in Figure 7.13 to help us "walk around the loop." We will have an augmented discussion of the measurements made on that system, showing how different PSD measurements were extracted for different blocks in Section 7.8. We and how we put it all together to get noise strata in Section 7.9 and then Section 7.10 will show how to use our extracted sources and models to extrapolate how changes in a noise source would affect a particular measurement point.

Terril Hurst, Dick Henze (our manager at the time), and I introduced PES Pareto to HP's Disk Memory Division (DMD) in 1995. In 1996 HP exited the hard disk business, so Terril and I got permission to publish the work, which we first did at the 1997 American Control Conference in Albuquerque [35, 36, 37]. Publishing three papers from the same authors in one disk drive control session was pretty much unheard of, but then CMU ME Professor William (Bill) Messner (now at Tufts) was adamant that the material should be heard together and rearranged the session to make it happen.

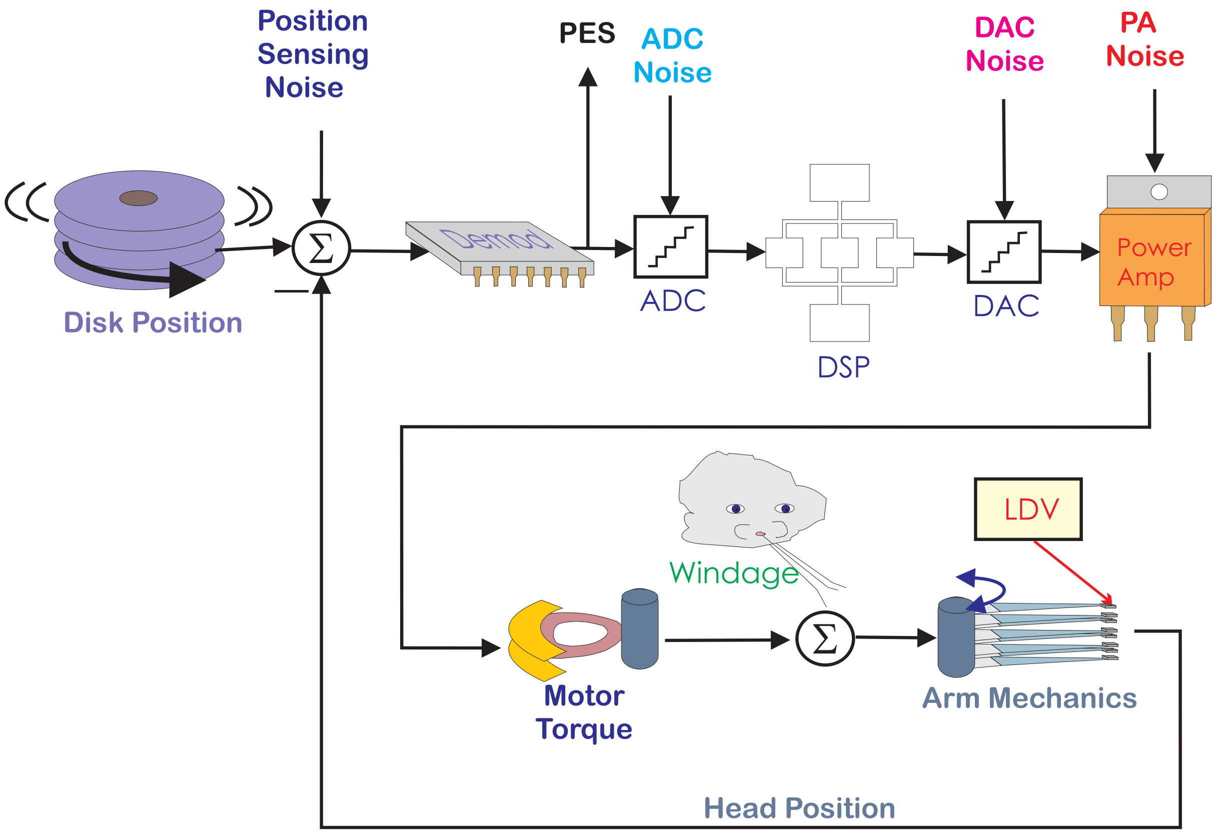


Figure 7.14: Generalized view of track following model for executives.

Together, those papers described the PES Pareto method [35], how to make measurements on a disk drive to feed the method [36], and the results of applying the method to some HP disk drives of the time [37] (KittyHawk II (1.3" diameter, 40 MB), and Lynx II (3.5" diameter, 1 GB). They describe a method of breaking down the Position Error Signal (PES) of a magnetic disk drive to its contributing components. Once these components are identified, they can be ranked in terms of their overall effect on PES and thus the most critical ones can be worked on first. In order to do a practical analysis of the contributors to PES, the fundamental question that must be answered is: What can be measured? While this may seem whimsical at first, it should be noted that in any real system, we will not have access to all the measurement points that we desire. Furthermore, although many different analysis tools might theoretically be available, they are useless to us if they cannot make use of the actual laboratory measurements available to us.

In order to guide our measurements and our modeling, it is useful to have a map of the system. The block diagram in Figure 7.13 will serve as the map for our tour of noises in the system. Starting at the left of this diagram, the reference position that the actuator arm must follow is the position of the magnetic track written on a disk, turning on a spindle. Only the position error – the difference between the reference track position and the readback head position – is sensed by the readback head, and this error signal is sent to the demodulator. The demodulator outputs a set of numbers at the system sample rate, and these are combined electronically to form PES. This PES signal is then converted to a digital format via an analog to digital converter (ADC), filtered by the compensator and then sent back out to the power amplifier via a digital to analog converter (DAC). The power amp converts the desired voltage into a current to drive the voice coil actuator (with torque constant $K_t$). The actuator itself has rigid body behavior as well as resonances. Through this, the head position is set. The position error is then sensed by the head. Absolute head position is not generally known from what is read off of the disk surface, but can be obtained in the laboratory by shining a laser spot from a Laser Doppler Vibrometer

(LDV)[1] off of the side of the head. While this nominally measures velocities, the result can be accurately integrated in time (for the frequencies we are concerned with) to obtain position.

We engineers like block diagrams that explain things in detail to us. This certainly was the case with Figure 7.13. Something happened that changed my view of how to present this material. Joel Birnbaum, then Director of HP Labs and Hewlett-Packard's CTO, was touring our lab with his entourage. Now, Joel is a brilliant technical contributor, but not everyone in the entourage would remember any of their technology. Thus, I made a non-engineer, executive version of Figure 7.13, and that was Figure 7.14. I was kind of embarrassed to draw Mr. Wind to represent air flow. However, during the lab tour, the most amazing thing happened: they loved it. This might feel cheap, but as anyone familiar with Tom Wolfe's seminal book, The Right Stuff [205], knows: "No bucks, no Buck Rogers." You have to get the folks controlling the money to think they understand what you are doing. Figure 7.13 would not have done that but Figure 7.14 sure as heck did.

There are several measurement points that can be accessed around the loop: $X_{out}$, $I_{sense}$, PES, and head velocity (and position) via the LDV. In general test signals can be injected into the loop only at $X_{in}$.

There are several likely noise input points on a disk drive. First of all, there are the noises associated with the moving disk and the readback process. These all enter the loop at the same point, but have different root causes. The noise due to the motion of the disk attached to a ball bearing spindle creates both Repeatable Run Out (RRO) (typically at orders of the spindle rotational frequency) and Non-Repeatable Run Out (NRRO). One of the interesting properties of servowritten disks is that one pass of the NRRO is usually locked into the servo position information when it is written. Thus, this written in NRRO is repeated at every revolution of the disk. The other noise source that enters at this point is the noise from the readback process of position information, called Position Sensing Noise (PSN). This noise can

[1] In this case, made by Polytec.

be due to the magnetic domains on the disk, the behavior of the magnetic readback head, the interaction of these two, or the action of the demodulator. (We lump demodulator noise into PSN for our current analysis.) Downstream in the loop, there are potential noise sources at the ADC and DAC (due to quantization), noise at the power amp, and finally windage. Windage is caused by the air flow generated as the disk spins. This air flows over, under, around, and into the actuator arms and the readback head, disturbing the head position. Given all these potential noise sources, there is a fundamental need to identify which of these – if any – are the most significant contributors to PES. With this information, the effort to reduce the noise in PES can be concentrated on the critical few.

It is worth noting that we purposely ignore external shock and vibration in this analysis for two reasons. First of all, external shock and vibration is heavily influenced by the drive's operating environment while the above noises are a function primarily of the drive. The second is that prior work in this area[187] gives us some confidence that we already have a reasonable engineering solution to many types of external shock and vibration. Thus, this work will focus on internal noises.

The tools available to us are a set of laboratory instruments that can make both time and frequency domain measurements. In particular, Digital Storage Oscilloscopes (DSO) can record time domain data as can certain spectrum analyzers. The spectrum analyzers are most useful, though, for measuring linear spectra, power spectra, power spectral densities (PSDs), and frequency response functions of systems. Spectrum analyzers typically only measure spectra as an independent quantity, but a class of these called network analyzers measures an output spectrum over an input spectrum. In particular, the network analyzers that we use are the HP 3563A Control Systems Analyzer and the HP 3567A Multi-Channel Analyzer. The latter instrument has the advantage of allowing more than two signals from the system to be measured at once.

For analysis, we used the standard set of matrix based tools, in particular MATLAB and **Simulink**. The tools one might use today would be the same, or one could imagine using Octave or some of the Python tools. What was novel at the time and still a lot more rare than it should be is that we made our measurements with the conscious thought of transferring them in to MATLAB for analysis [206].

## 7.8 Measurements for PES Pareto

The ideas of Section 7.5 fundamentally depend upon having measurements of noises at different parts of the loop to manipulate back and forth. Some noises can be measured simply by breaking the loop at some point and making a measurement downstream of the block in question. In other cases, the measurements are only available when the loop is closed in feedback. For the most part, the method is assuming that we can apply linear analysis through our feedback loop, especially when the signals are small (as we expect them to be for noise analysis). This breaks down when we want to relate the effects of quantization from analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) to these linear methods.

It should be noted that this section contains the most ad-hoc techniques in the tutorial. The measurement test points we want are not always available. Some measurements can only be made when the loop is broken, while others are only feasible in closed-loop. A certain amount of faith is needed that the noises we measure when we isolate a component are representative of those noises when the loop is operational.

### 7.8.1 Measurements in Open and Closed Loop

In order to do a practical analysis of the contributors to error, the fundamental question that must be answered is: What can be measured? In any real system, we will not have access to all the measurement points that we desire. Furthermore, although many different analysis tools might theoretically be available, they are useless to us if they cannot make use of the actual laboratory measurements available to us.

In order to guide our measurements and our modeling, it is useful to have a map of the system. In the original hard disk drive example, the Figure 7.13 served as the map for our tour of noises in the system. Starting at the left of this diagram, the reference position that the actuator arm must follow is the position of the magnetic track written on a disk, turning on a spindle. Only the position error – the difference between the reference track position and the readback head position – is sensed by the readback head, and this error signal is sent to the demodulator. The demodulator outputs a set of numbers at the system sample rate, and these are combined electronically to form PES. This PES signal is then converted to a digital format via an analog to digital converter (ADC), filtered by the compensator and then sent back out to the power amplifier via a digital to analog converter (DAC). The power amp converts the desired voltage into a current to drive the voice coil actuator (with torque constant $K_t$). The actuator itself has rigid body behavior as well as resonances. Through this, the head position is set. The position error is then sensed by the head. Absolute head position is not generally known from what is read off of the disk surface, but can be obtained in the laboratory by shining a laser spot from a Laser Doppler Vibrometer (LDV) off of the side of the head. While this nominally measures velocities, the result can be accurately integrated in time (for the frequencies we are concerned with) to obtain position.

We had the following useful models that we could measure from our physical system. Our block groupings in Figure 7.13, are the plant, $P(s)$:

$$P(s) = \frac{1}{K_t}\left(\frac{LDV}{I_{sense}}\right)_{\text{3WM}} \approx \frac{1}{K_t A(s)}\left(\frac{LDV}{X_{in}}\right)_{\text{OfCLM}}, \tag{7.64}$$

the power amplifier, $A(s)$:

$$A(s) : \text{from block model}, \tag{7.65}$$

the compensator, $C(s)$:

$$C(s) = \left(\frac{X_{out}}{PES}\right)_{\text{meas}} \approx \text{Analytic Model}, \tag{7.66}$$

and the servo position generator or demodulator, $D(s)$:

$$D(s) = \frac{\left(\frac{NPES}{I_{sense}}\right)_{\text{meas}}}{\left(\frac{LDV}{I_{sense}}\right)_{\text{meas}}}. \tag{7.67}$$

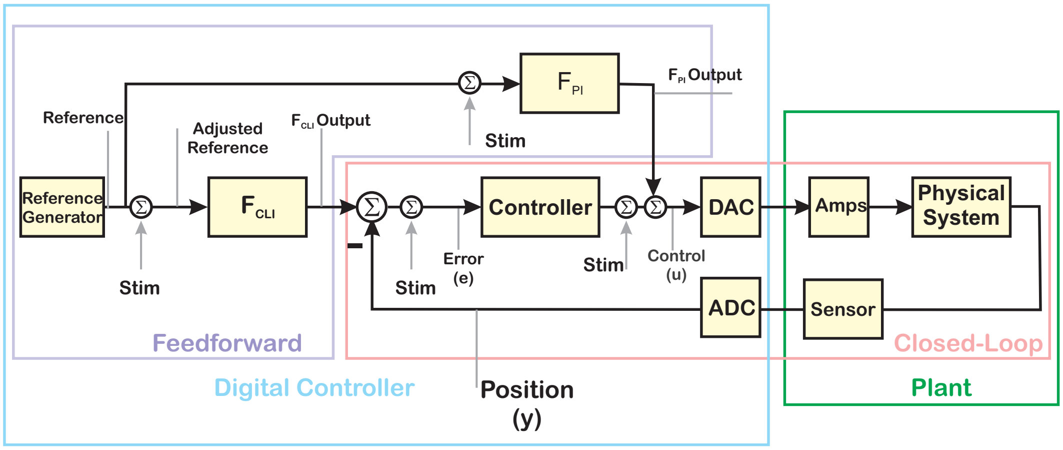


Figure 7.15: Generic feedforward-feedback loop with measurement points. Note that if measurement IP is built into the digital controller, then the number of test injection and measurement points gets much larger. Furthermore, the measurements are made at frequencies compatible with the digital control system.

In traditional measurement systems using external instruments, such as this disk drive example, there were a limited number of measurement points that could be accessed around the loop: $X_{out}$, $I_{sense}$, PES, and head velocity (and position) via the LDV. In general test signals could only be injected into the loop at $X_{in}$. However, in the two decades since the original PES Pareto work, the nature of instrumentation and measurements has changed quite a bit. As digital controllers have gotten more ubiquitous, it has made less and less sense to connect analog instruments to the loop, that is, instruments that required their own ADCs and DACs. Many of the classic instruments from the original PES Pareto work [34, 35, 36, 37] are no longer manufactured. Even for these instruments, many of the digital control loops run at sample rates an order of magnitude higher than these instruments could allow.

Instead, it has become more logical to build in measurement tools right into the digital controller [87]. Diagrams such as the one in Figure 7.15 should become the norm rather than the exception. In particular, such built-in measurements allow the measurement algorithms to use the same data conversion path as the control loop. Furthermore, the number of digital measurement points (both for signal injection and measurement) are dramatically increased and the data for measurement does not need a format conversion from the data for running the loop.

The classic laboratory instruments for these kinds of measurements include spectrum analyzers (discussed earlier), network analyzers (which calculate the ratio of input and output spectra), and digital storage oscilloscopes (DSO). Any of these can be recreated within a modern digital controller and should be for the reasons mentioned above. What are known as dynamic signal analyzers (DSA) used for measuring frequency response functions (FRFs) of control systems, are generally low frequency versions of network analyzers. Rather that focusing on the specific instruments of the original work, it seems more useful to discuss the types of measurements that we want for the methodology.

Returning to our disk drive example, there are several likely noise sources in the loop. First of all, there are the noises associated with the moving disk and the readback process. These all enter the loop at the same point, but have different root causes. The noise due to the motion of the disk attached to a ball bearing spindle creates both Repeatable Run Out (RRO) (typically at orders of the spindle rotational frequency) and Non-Repeatable Run Out (NRRO). One of the interesting properties of servowritten disks is that one pass of the NRRO is usually locked into the servo position information when it is written. Thus, this written in NRRO is repeated at every revolution of the disk. The other noise source that enters at this point is the noise from the readback process of position information, called Position Sensing Noise (PSN). This noise can be due to the magnetic domains on the disk, the behavior of the magnetic readback head, the interaction of these two, or the action of the demodulator. (We lump demodulator noise into PSN for our current analysis.) Downstream in the loop, there are potential noise sources at the ADC and DAC (due to quantization), noise at the power amp, and finally windage. Windage is caused by the air flow generated as the disk spins. This air flows over, under, around, and into the actuator arms and the readback head, disturbing the head position. Given all these potential noise sources, there is a fundamental need to identify which of these – if any – are the most significant contributors to PES.

With this information, the effort to reduce the noise in PES can be concentrated on the critical few.

It is worth noting that we purposely ignore external shock and vibration in this analysis for two reasons. First of all, external shock and vibration is heavily influenced by the drive's operating environment while the above noises are a function primarily of the drive. The second is that prior work in this area [187, 188] gives us some confidence that we already have a reasonable engineering solution to many types of external shock and vibration. Thus, PES Pareto focuses on internal or self-generated noises.

### 7.8.2 Measurements/Modeling of Power Amplifier Noise

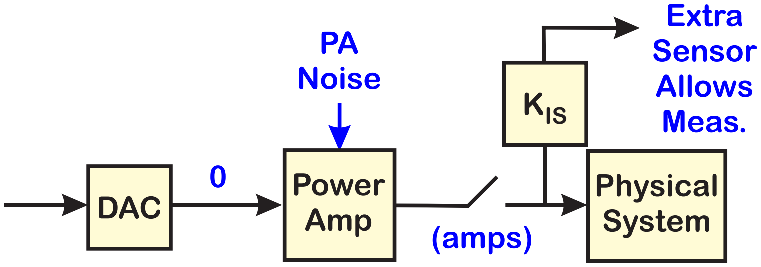


Figure 7.16: Generating different conditions to measure power amplifier noise.

Referring back to Figure 7.8, power amplifiers are usually needed to convert the low voltage signals out of DACs into currents large enough to drive an actuator. This is pretty universal, unless that amplification is done in the actuator itself. They tend to have a low-pass nature, but their bandwidth is usually considered beyond that of the actuator, plant, or closed-loop system. The nature of their amplification allows for the generation of noise, and so we can often break out these components as individual blocks to be analyzed for their noise contribution. When a model is needed for filtering noise, it can often be generated in MATLAB to mimic the data sheet frequency response function (FRF), but at the frequencies that we want for our PSD measurements. Alternately, with enough control over the digital controller (as diagrammed in Figure 7.15), one can stimulate the DAC to generate an FRF measurement, provided we have a way to measure the current being produced by the power amplifier. In Figure 7.16 there is a sense resister often used for monitoring the current being produced by the power amp to drive the actuator. In the disk drive, such a signal has utility as a secondary measurement of what is going into the actuator. Note that we still need to convert the voltage (converted from the sensed current using a reference resistor).

To isolate a PSD of the noise due to the power amplifier we need some sort of differential measurement.

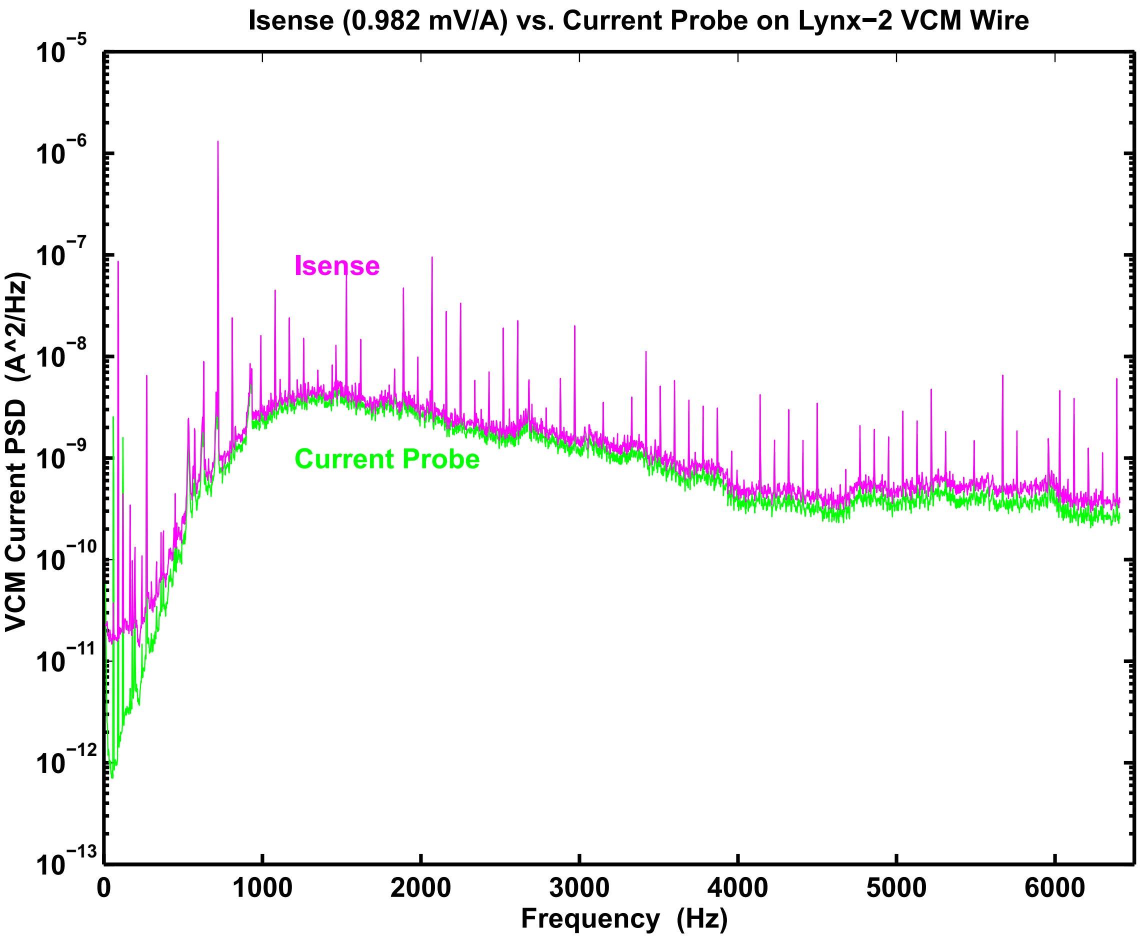


Figure 7.17: Calibrating the current sense measurement against a highly accurate voltage probe.

Since the amplifier can be run typically in open loop, we can set the DAC output to $0$ to generate a PSD of the PA and ambient actuator input noise (Figure 7.16). In this case, we can drive the DAC with an all zeros code. We would normally be concerned with DAC quantization, but an all zeros code should produce a calibrated $0$ output at the DAC voltage. (The signal is not moving, so there is no quantization to speak of.) In the second measurement, we physically open the circuit from the power amp to the rest of the actuator. The current sense measurement now only has the ambient actuator input noise. If we take the expected values of the PSDs from both measurements (and this means averaging multiple measurements) we can assume independence and subtract the two PSDs. This means taking more than one measurement of these signals to get the statistical independence to kick in.

Figure 7.17 shows a calibration of the current sense measurement. The close match between the current sense, $I_{sense}$ in the plot, and that from a Current Probe, indicates that with a bit of smoothing the $I_{sense}$ is a reliable measurement of the current being sent to the actuator. We then measure directly at $I_{sense}$ and since that is the source point as well, the back filter to the source is simply $1$ for all frequencies of

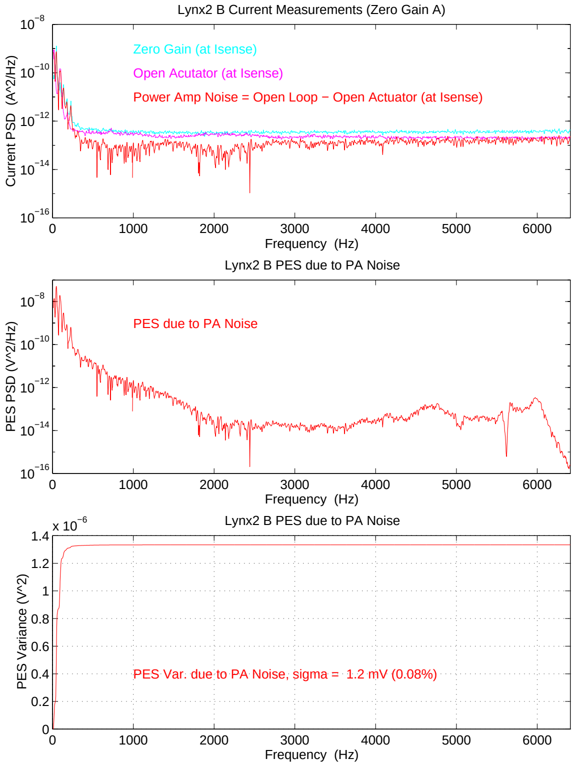


Figure 7.18: Measurement of Power Amplifier Noise on Lynx 2 Disk Drive. PSDs are generated from open loop and open actuator configurations, allowing the difference to be considered the power amplifier noise.

interest. We difference the PSDs in the top plot of Figure 7.18. In the middle plot of Figure 7.18, we forward filter to PES through the magnitude squared filter:

$$\left\| \frac{K_t P(s) D(s)}{1 + K_t P(s) D(s) C(s) A(s)} \right\|^2 \tag{7.68}$$

to get the contribution to PES, where $P(s)$, $A(s)$, $C(s)$, and $D(s)$ are defined in Equations 7.64 – 7.67. In the lower plot of Figure 7.18, this PSD is integrated to get the variance (in the frequency domain) due to the power amplifier.

### 7.8.3 Measurements/Modeling of Plant Disturbance

Separating out plant disturbances from noise in the position sensing has a particular challenge when the same sensor is used in the measurement for the error signal as is used to try to quantify plant

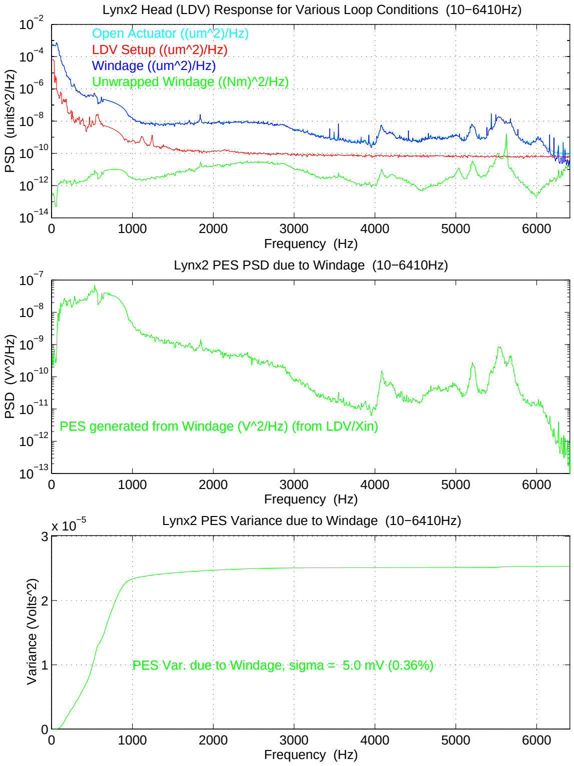


Figure 7.19: Measurement of the disturbance due to air flow on the drive head (windage) on the drive example via an LDV. Backfiltering the LDV measurement through the inverse plant model gave the windage as an open-loop source, which could then be filtered forward through the model to see its effect on PES.

disturbances. This almost always calls for an external sensor, one that is of high enough quality to give a signal that can be trusted on its own. In the case of the HDD, the issues with the loop sensor were that it had a limited bandwidth (due to the sectored servo patterns used in HDDs) and the generation of the error signal itself was part of the experiment. There was no option to up the sample rate and filter, and the error signal (PES) was only really available when the drive was in closed loop. Furthermore, it was an error only sensor, not giving us the actual position of the drive head. This is a major limitation, since two significant noise suspects were the buffeting of the drive head by the air flow generated by the spinning disks and the actual generation of the error signal from the servo position dibits. An instrument such as an laser interferometer [207] or a laser Doppler vibrometer (LDV) could provide an actual position measurement (against a calibrated distance set to 0) if an appropriate spot on the drive head could be found for reflecting the laser spot.

In the disk drive example, a Polytec LDV provided highly accurate position measurements for any frequency above 10 Hz. We could minimize the feedback loop gain in the controller and measure the head position (not the error) by shining the laser spot on the side of the drive head. We could then back filter to the noise source (modeled at the plant input) by back filtering through:

$$\left\| \frac{1}{P(s)} \right\|^2 . \tag{7.69}$$

With this windage "input noise", we now filtered forward through

$$\left\| \frac{P(s)D(s)}{1 + K_t P(s)D(s)C(s)A(s)} \right\|^2 \tag{7.70}$$

to get the effect of windage on PES.

This is an example of a more general problem: there will always need to be a way to separate the disturbance generated movement from the error generated in measuring position. After all, we cannot perform any reasonable tradeoffs in a Kalman filter design [208] if we have no idea what the noise powers of $w$ and $v$ are. There are other noises in the loop, but at some point, we need to be able to isolate these as well.

### 7.8.4 Measurements/Modeling of ADC and DAC Noise

In Section 7.6.3 we discussed some basic modeling for quantization noise. In this section, we will apply those ideas to generating noise as a source for our Pareto measurements. Recalling Section 7.6.3, we had arrived at a quantization PSD (7.35) of $\Phi_{QQ}(f) = \frac{q^2}{12 f_S} = \frac{q^2 T_S}{12}$. At a first cut, we can use this to define the quantization noise being added in at the output of our ADC or DAC.

However, there are methods of making measurements on ADCs. Usually, the engineer injects either a simple tone or $0$ at the analog input to the ADC. The signal is sampled and quantized and the spectrum of the measurement is FFTed. In the case of the tone, this signal will dominate any integral that is done, so it is removed, either by excising the frequency bin and replacing an average of neighboring bins, or by applying an adaptive noise canceling approach [24] to match and remove the tone before the FFT is computed. In either case, the sample mean of the data is removed.

Now, why inject a tone at all if we are going to remove it? The tone exercises the full range of the ADC, which was one of the assumptions of the Widrow model discussed in Section 7.6.3. As such, the noise properties of the ADC with and without a tone might be different, so it is good to have both measurements. Secondly, there are several possible noise measures used with respect to signals, and one of them is dBc or Decibels with respect to a carrier. We need a tone, a carrier, to compute this.

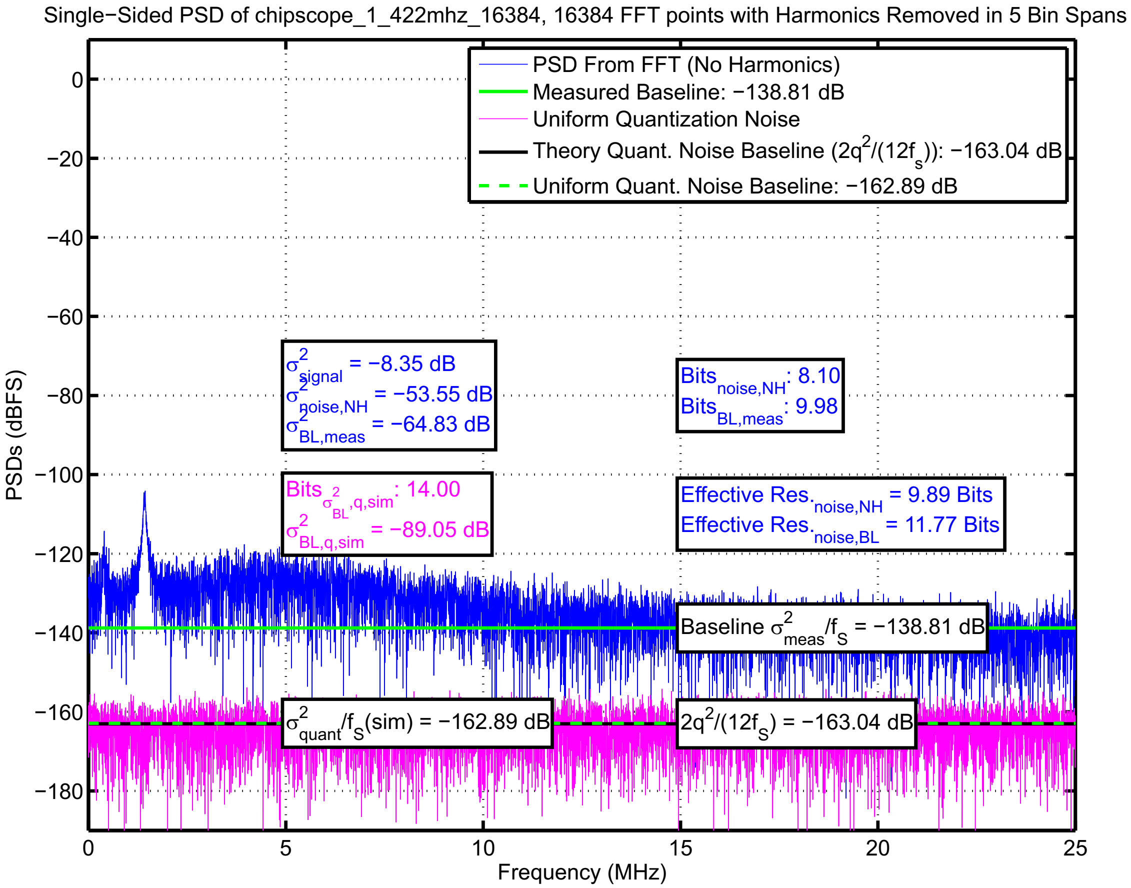


Figure 7.20: Decomposition ADC noise with a removed tone.

Removing the tone itself has its own issues. The best method is probably the adaptive noise canceling approach, since that removes the matched signal only and not other signals at that frequency. Removing a tone once an FFT has been done has issues of spectral leakage, that is that the FFT bin itself bleeds into others. To combat that long data runs being FFTed are often windowed – in this context meaning that they are multiplied by a window that drives the end points of the data record to 0. (This is another reason why overlap processing is a useful trick, allowing us to lengthen the record to a point that the window doesn't affect the data points we want.)

Assuming we have made a measurement (with a tone that was removed or without a tone) and computed the FFT, we then can compute the PSD and make it one-sided using the methods described in Section 7.6.2. How do we related this back to our flat quantization PSD model? We fit that data to a flat PSD across the frequency band (usually looking at the high frequency tail to establish the level), and then set that level to $\frac{q_{eff}^2}{12 f_S}$. We can now back out our effective quantization level, $q_{eff}$.

An example is shown in Figures 7.20 and and 7.21. Figure 7.20 shows a measurement of a 14-bit ADC

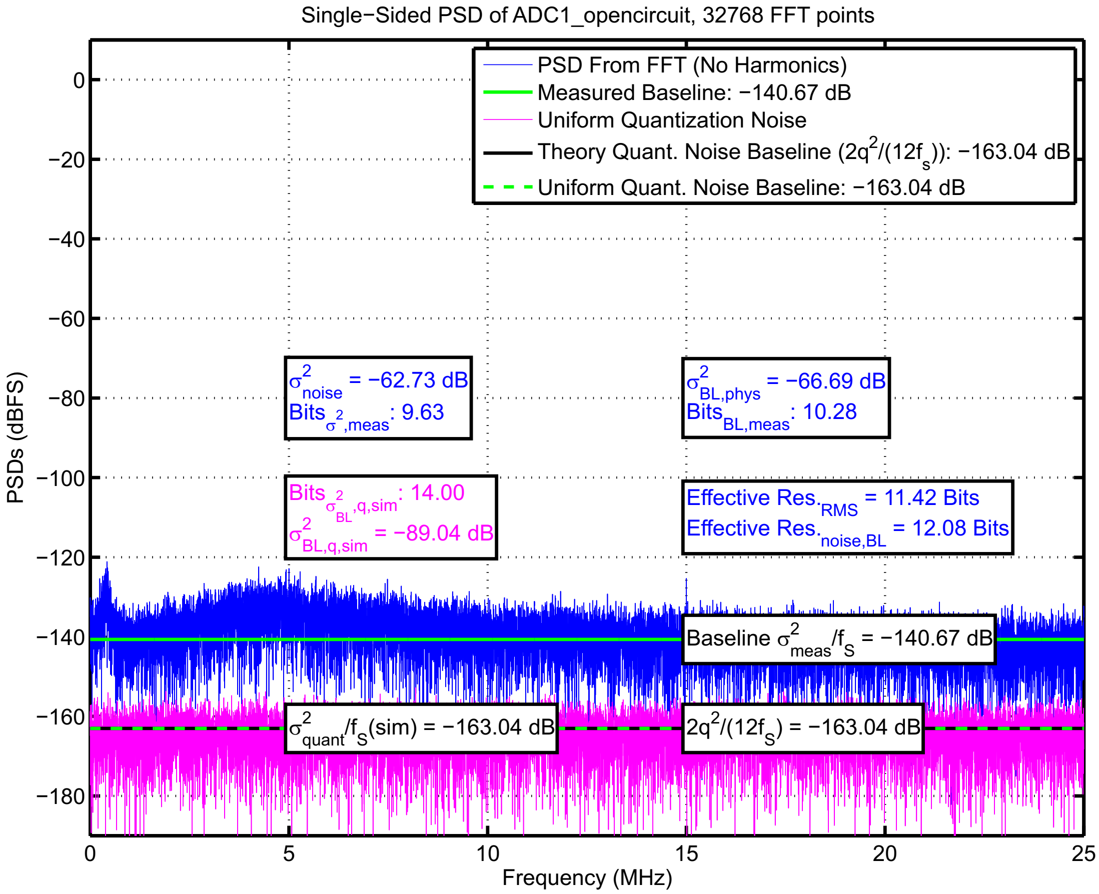


Figure 7.21: Decomposition ADC noise with an open circuit.

sampled at 50 MHz, into which a 1.422 MHz tone was injected. A 16,384 point FFT was performed and the frequency bins containing the tone and its harmonics were flattened by replacing their signals with the averages of nearby bins. To keep the spectral leakage low, the data was windowed with a 16,384 point Hanning window to drive the end points to 0. If we start with by normalizing the full scale range, FS to $\pm 1$, we can then calculate the effective value of $q$ and using the math in Section 7.6.3, compute the PSD for the theoretical quantization noise. This is the magenta curve, and the computed PSD is plotted along with the pure theoretical curve. Another curve, taken from averaging the high frequency end of the computed PSD is also drawn, as a way of verifying that the curve represented the original theoretical value. We then could take the measured signal (with the tone removed) and calculate the average of the high end of the PSD, allowing us to draw the green curve back, to reveal an effective level of quantization PSD. Again, using the calculations from Equation 7.42 in Section 7.6.3, we back out an $N_{eff}$. Figure 7.21 repeats the same calculation, but with an open circuit input to the ADC. There is no tone to remove, but it is helpful to see that the resulting calculations give us similar numbers for the effective bits.

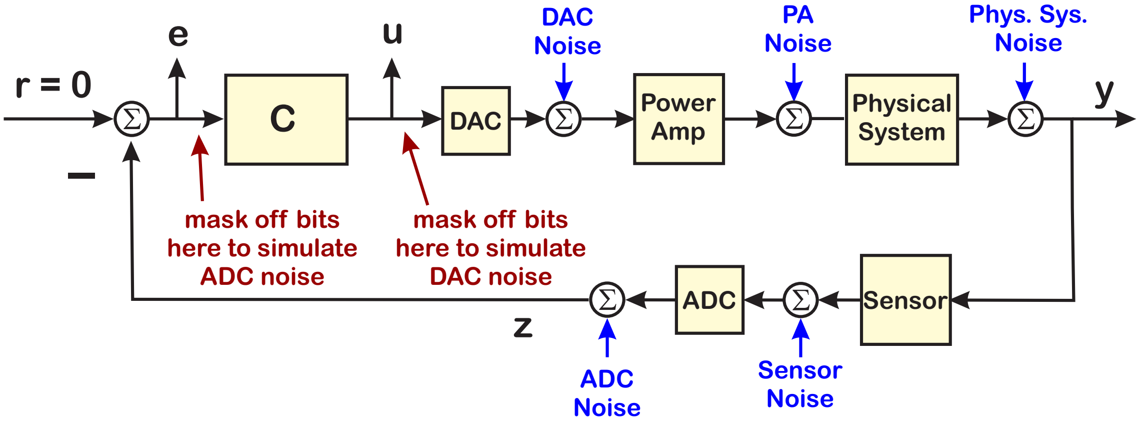


Figure 7.22: Simulating ADC and DAC quantization by masking off bits at different signal points. On their own, the masked off bits are still one of many noise sources, but when they are differenced from each other, the unchanging noise PSDs drop out, revealing the effects due to the quantization.

This can be done with an ADC if we have a reliable analog signal source of higher resolution than the ADC and if we have the ADC connected to a test system. The first characteristic is to make sure we are not including the noise in the signal source, and the second characteristic is needed just to get the discretized signal into a place where it can be processed. It becomes more difficult for a DAC, because while we may be able to generate a high fidelity signal to send out to the DAC, we now need to digitize the analog voltage produced and get that back into some analysis engine, without that discretization becoming the dominant or even a significant part of that measurement. For either device, we often do not have the ability to evaluate the components separately. In this case we can employ a scheme diagrammed in Figure 7.22.

When we cannot separate out the ADC from the loop, we use this scheme of masking off bits in the error measurement (PES) (assuming the reference signal, $r = 0$). By artificially masking off these bits, we can generate a PSD of the different levels of PES. For the hard disk example, the resulting plots are shown in Figure 7.23. Note that this PSD was made with frequency bins out only to 2 kHz, which did not match the other measurements, so the frequency bins were zero padded with more bins at higher frequency to produce the top plot of Figure 7.24. The differencing of those measurements is shown in the middle plot of Figure 7.24. We find that differencing these small PSD levels made it difficult to have any sort of non-visual fit to the data. Instead, the effective level of quantization noise was back calculated from PES measurements by assuming a uniform input noise distribution. This was then forward filtered to PES via:

$$\left\| \frac{K_t P(s) D(s) C(s) A(s)}{1 + K_t P(s) D(s) C(s) A(s)} \right\|^2 = \left\| \frac{1}{T_{cl}} \right\|^2 . \tag{7.71}$$

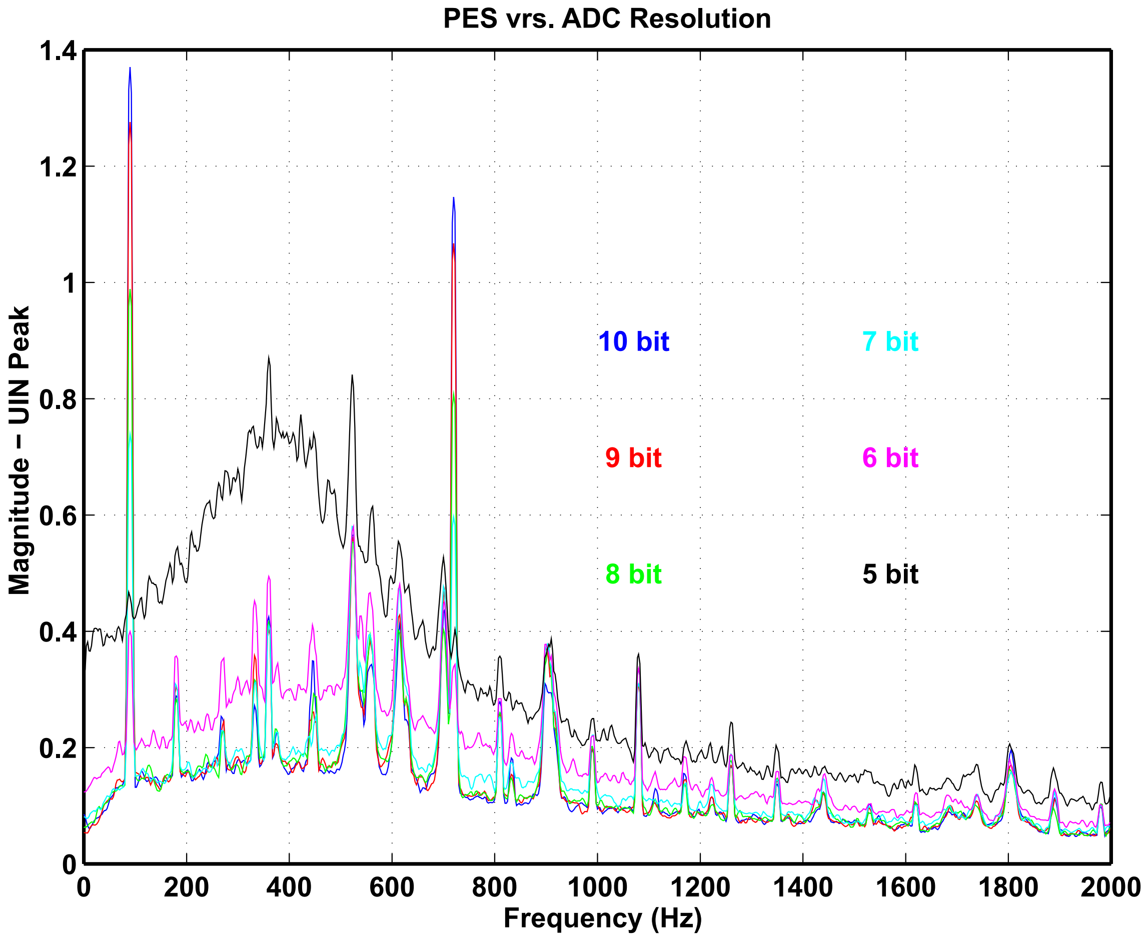


Figure 7.23: Measurement of ADC quantization by masking off bits at the error signal.

Eventually, the scaling of quantization that matched the curves in the lower plot of Figure 7.24 was:

$$q = \frac{1.25\ V}{512\ counts} * \frac{1}{20}. \tag{7.72}$$

We employ a similar measurement scheme for quantization noise from the DAC. By masking off bits being sent to the DAC, we can generate a PSD of the different levels of PES. For the hard disk example, the resulting plots are shown in Figure 7.25. Again, this PSD was made with frequency bins out only to 2 kHz, which did not match the other measurements, so the frequency bins were zero padded with more bins at higher frequency to produce the top plot of Figure 7.26. The differencing of those measurements is shown in the middle plot of Figure 7.26. Once again, we find that differencing these small PSD levels made it difficult to have any sort of non-visual fit to the data. Instead, the effective level of quantization noise was back calculated from PES measurements by assuming a uniform input noise distribution. This

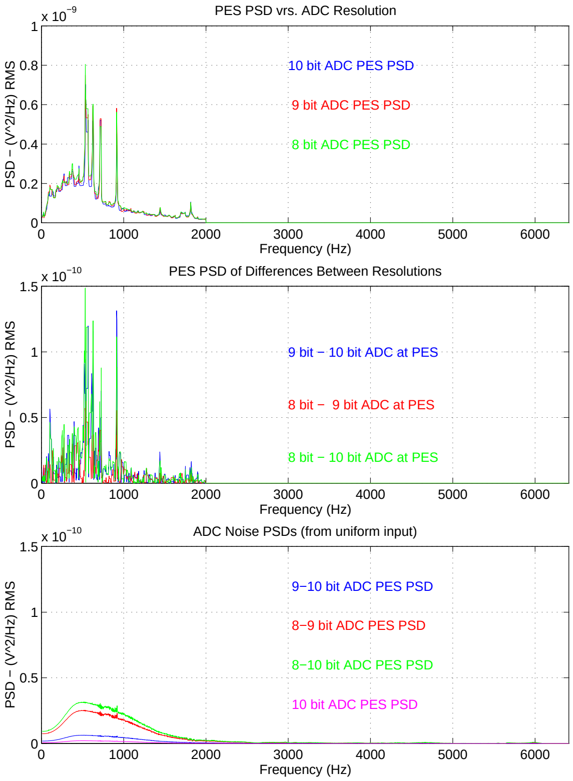


Figure 7.24: Measurement of ADC noise in closed-loop by differencing measurements of the type in Figure 7.23. Rather than try a numeric fit of the differences in PES, a scaled uniform noise is filtered through to the PES to match the difference signals.

was then forward filtered to PES via:

$$\left\|\frac{K_tP(s)D(s)A(s)}{1+K_tP(s)D(s)C(s)A(s)}\right\|^2 = \left\|\frac{C(s)}{T_{cl}}\right\|^2. \tag{7.73}$$

Eventually, the scaling of quantization that matched the curves in the lower plot of Figure 7.26 was:

$$q = \frac{0.3125\ V}{512\ counts} * \frac{1}{9}. \tag{7.74}$$

### 7.8.5 Channeling Sherlock Holmes

At some point, we have isolated all the contributors to the error signal (PES in our disk drive example) that we can and we can put them together. We can show them independently to show the most important ones, or stack them cumulatively to show how much of the PES PSD we have accounted for, as shown

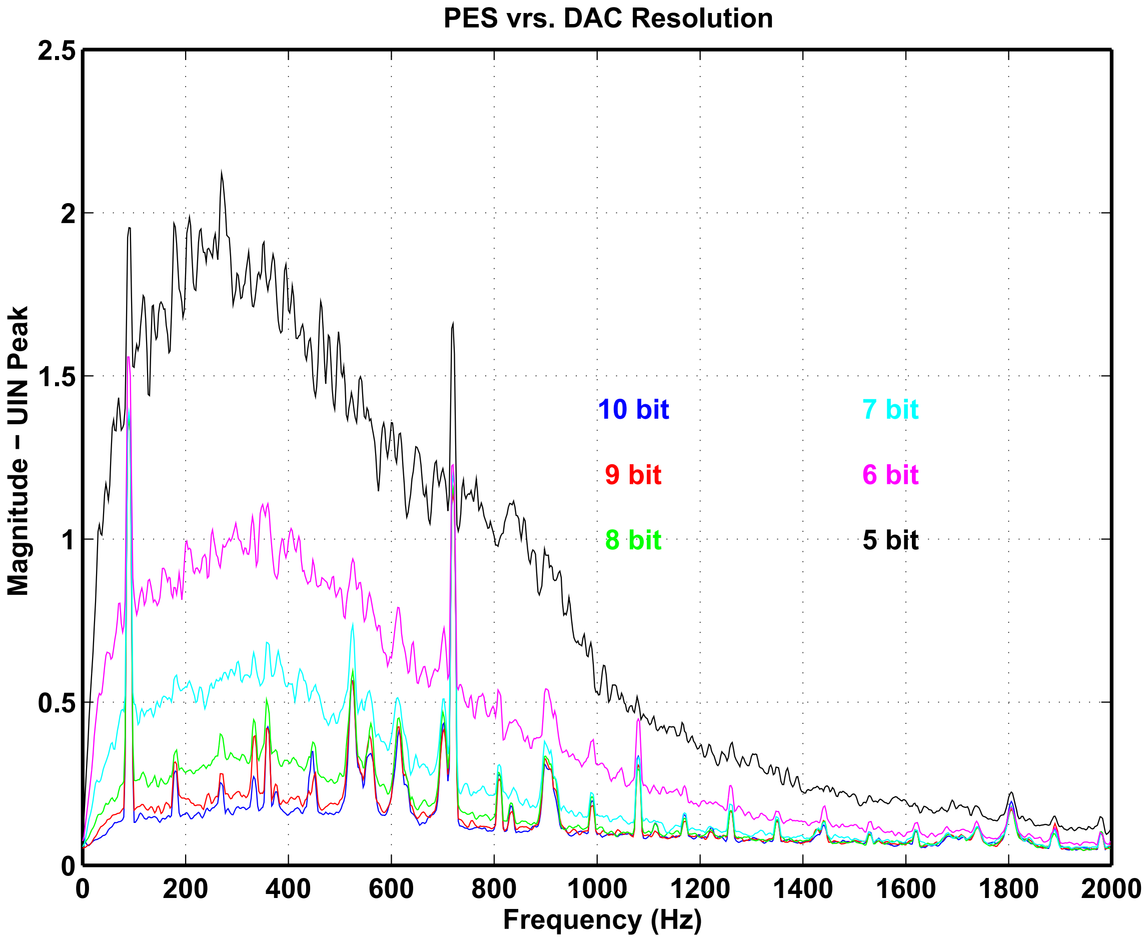


Figure 7.25: Measurement of DAC quantization by masking off bits at the controller output signal.

in Figure 7.27. There is still a large amount of noise that is unaccounted for. If we plot the PSD of PES - the Cumulative PSD of all the noises we have accounted for, we get the plot of Figure 7.28. A feel for sensitivity functions, especially those plotted on a linear frequency scale reveals that this looks a lot like:

$$k * \|D(s)S_{cl}(s)\|^2 \tag{7.75}$$

At this point in the original effort [34, 35, 36, 37], we turned to the wisdom of Sherlock Holmes, who although fictional [186], was an ancestor of the equally fictional Mr. Spock. "An ancestor of mine maintained that if you eliminate the impossible, whatever remains, however improbable, must be the solution. [209]" To make this unaccounted for noise an input to the error signal (which is where any noise in position sensing would appear in this diagram), we filter the PES PSD and unaccounted for PSD by

$$\left\|\frac{1}{D(s)S_{cl}(s)}\right\|^2 \tag{7.76}$$

to get to a possible input Position Sensing Noise (PSN) PSD. This is shown for our example in Figure 7.29.

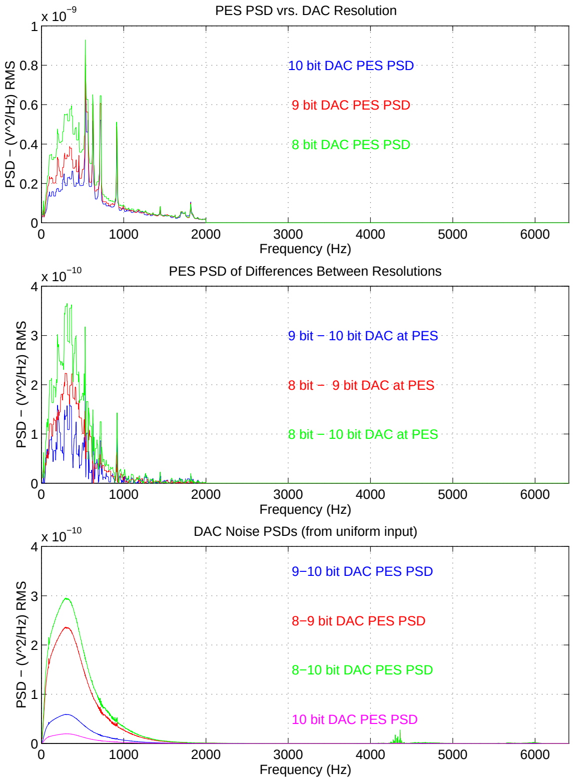


Figure 7.26: Measurement of DAC noise in closed-loop by differencing measurements of the type in Figure 7.25. Rather than try a numeric fit of the differences in PES, a scaled uniform noise is filtered through to the PES to match the difference signals.

Note that there is a broadband, essentially white noise component to "what's left". There is also a large hump at low frequency. As windage is accounted for already, the most likely source of this large hump is the actual non-repeatable motion of the disk on the rotating spindle (RT-NRRO). Likewise the broadband flat noise cannot be from the power amplifier, ADC, or DAC (since these have been eliminated) and therefore it follows that this is Position Sensing Noise. If this PSD is fed forward to PES and then integrated in frequency to yield the PES baseline variance due to PSN, this number, $\sigma_{PSN} = 0.03\mu m$, closely matches the prediction of the ANOVA analysis[36].

In our measurements, we have been able to find "simple tricks and nonsense [10]" to isolate noise PSDs from quite a few noise sources. We have brought in extra sensors and measurement points to give representative measures of other noise sources. In the disk drive example, windage (Section 7.8.3) was a large component. Still, we get to the total PSD of the broadband noise at the error signal, remove the PSDs that we have isolated, and we have something left over. This is the unaccounted for noise, but if we have confidence in our structural model, and if we have found a way to the other noises, then what is left, no matter how improbable, must be the sensing noise PSD.

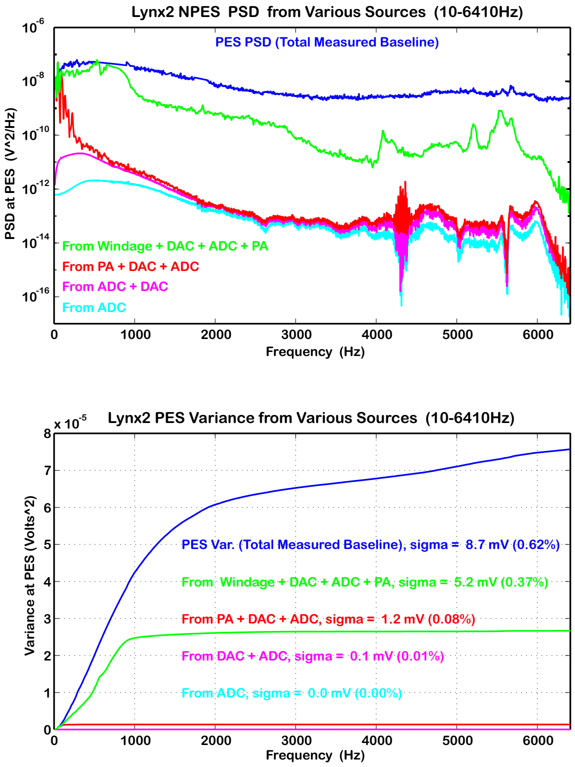


Figure 7.27: Decomposition of baseline noise sources in a hard disk. Cumulatively plotted to account for all of PES. The top plot shows the PSDs. The bottom plot has them integrated across frequency.

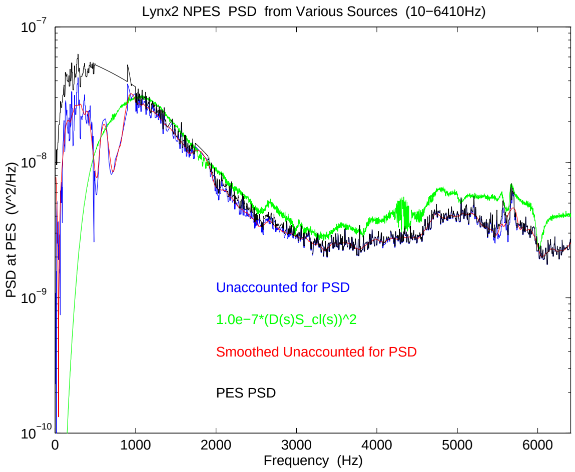


Figure 7.28: Unaccounted for PES PSD noise.

## 7.9 Noise Sources and Noise Strata

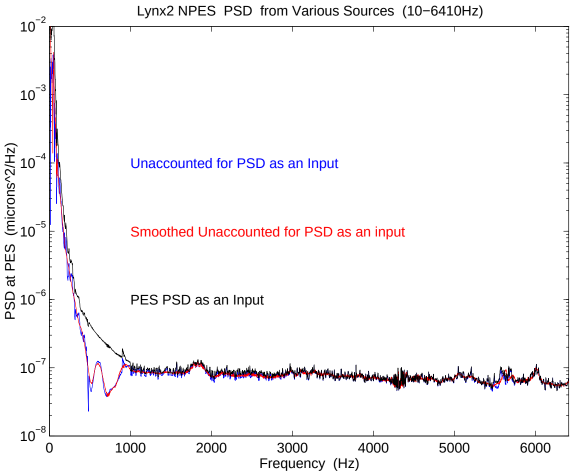


Figure 7.29: Unaccounted for PES PSD noise as an input.

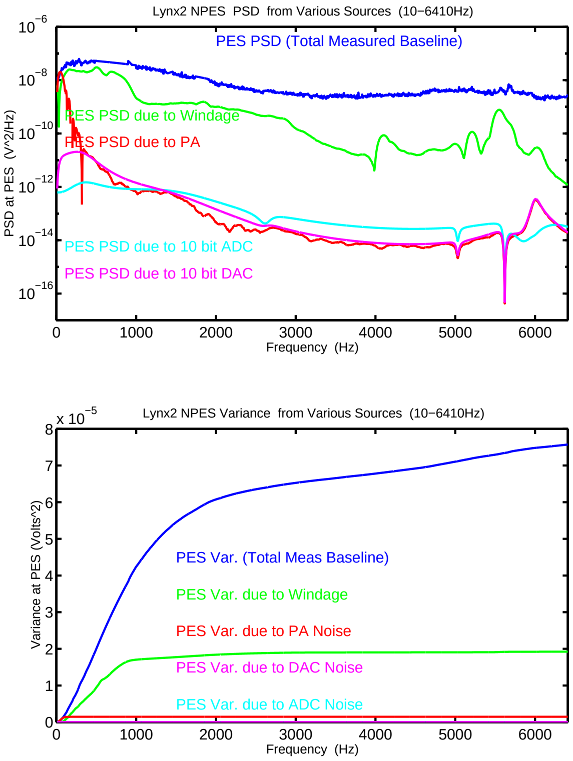


Figure 7.30: Decomposition of baseline noise sources in a hard disk. Independently plotted to show relative importance. The top plot shows the PSDs. The bottom plot has them integrated across frequency.

> A theory is a good theory if it satisfies two requirements: It must accurately describe a large class of observations on the basis of a model that contains only a few arbitrary elements,

> and it must make definite predictions about the results of future observations. — Stephen Hawking in "A Brief History of Time[7]"

> Science, therefore, for all the reasons above, is not what it appears to be. It is not objective and impartial, since every observation it makes of nature is impregnated with theory. Nature is so complex and so random that it can only be approached with a systematic tool that presupposes certain facts about it. Without such a pattern it would be impossible to find an answer to questions even as simple as 'What am I looking for?' James Burke in "The Day the Universe Changed[210]"

If we have a good model, and we have made all these measurements of different noises, we are now in a position to put them all together at the error measurement. We use our best models to filter the noises from their backed out sources to their effect on the error (with appropriate unit conversions). There are two reasonable ways to think of this. One can compare them independently to see which noise sources are the dominant ones that we should worry about. This is shown in Figure 7.30.

Once the potential contributors to the error signal are identified, they can be ranked in terms of their overall effect on error and thus the most critical ones can be worked on first. In our HDD example, Figure 7.30 shows these independent noise sources and what is telling is not only how little the ADC and DAC quantization contribute to the overall error in this system, but how much of it is due to two sources, the disturbance of windage buffeting the drive head and the actual sensing of the position error. The former led to Terril Hurst working on redesigning the air flow inside of hard disks before HP exited the business in 1996. This latter fact, that close to 2/3 of the baseband noise in the PES was due to sensing errors led to the work on improving the demodulation circuits for HDDs [211, 212, 213]. While improving the noise in the actual magnetic domains from which position was derived might have taken considerable effort, providing smarter electronics to cleanly process these signals was an inexpensive and dramatic improvement. The benefits of understanding demodulation are discussed in the tutorial of [185].

To be clear, as quantization is nonlinear, the fundamental assumptions of the Widrow model assumes that quantization is fine enough to model the error as noise. These results argued against worrying whether to add extra bits to the next generation of of ADCs and DACs. On other systems, they may be the dominant noise sources.

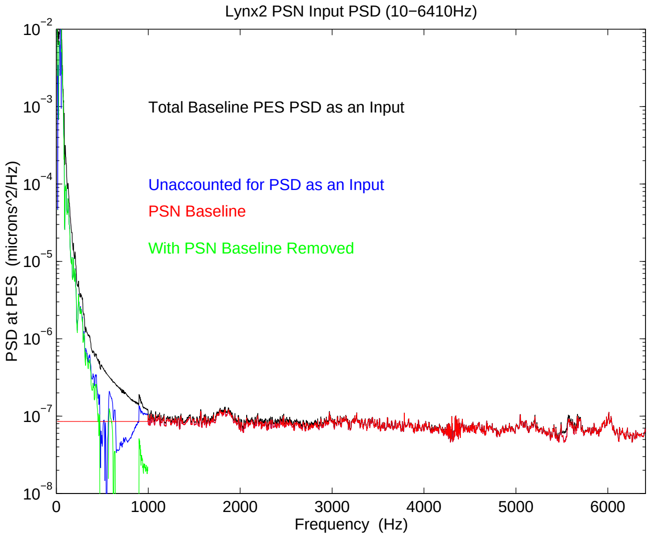


Figure 7.31: Baseline PSN

## 7.10 Using Pareto Models for Extrapolation

The quality of the results allow the effort to be put on the sources that most significantly limit the servo loop performance. Beyond analyzing the key sources of noise, present in a current system, the PES Pareto method, because it gave us noises as open-loop inputs and gave us a method tho add them independently at the error signal, allowed us to model the amplification or attenuation of any of these noise sources. Whether we could make a measurement (say of increasing the air flow by spinning the disk at a higher rotational speed) or simply increasing or decreasing a noise in a particular band of its input or across all frequencies, we could see what that change would have on our control loop.

For windage (or more generally the disturbance into the plant) we were able to change the level and measure the new result with the external sensor (the LDV). This is shown in the results of Figure 7.33. On top is the measurement at the LDV. The middle plot gives the effects of the different windage levels on PES. The lower plot integrates these to show the variance. Those variances are summarized in Figure 7.34. In the case of the Position Sensing Noise (PSN) we could scale that signal up and down, as shown in Figure 7.35. The plots show the potential benefit of dramatically reducing the PSN, which motivated the work on advanced demodulation.

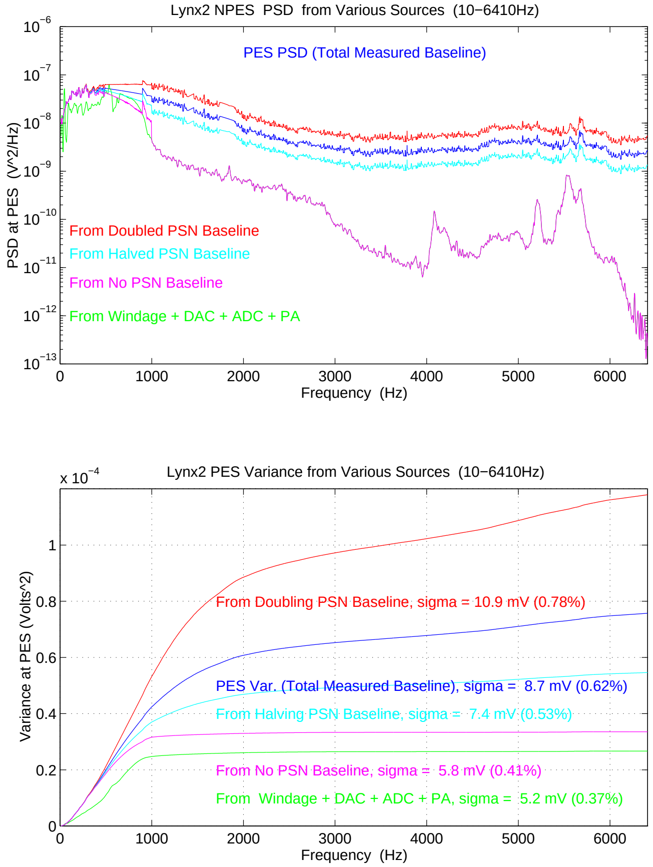


Figure 7.32: Effect of Changing Baseline PSN (5400 rpm)

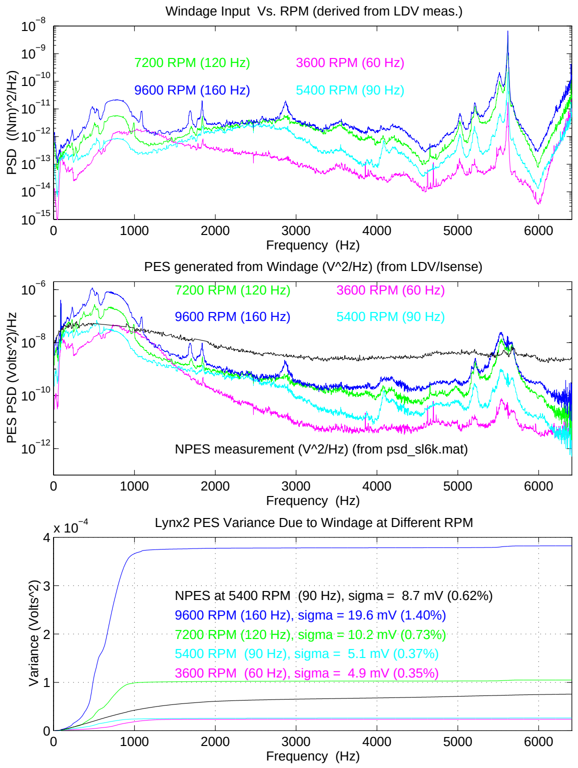


Figure 7.33: Effects of changing spindle RPM. On top is the measurement at the LDV. The middle plot gives the effects of the different windage levels on PES. The lower plot integrates these to show the variance.

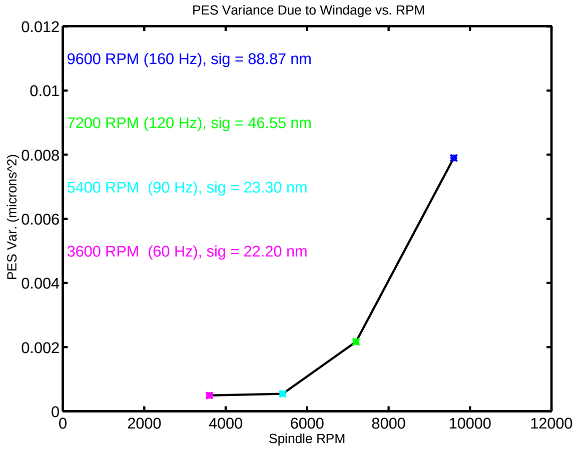


Figure 7.34: PES Variance Due to Windage Versus RPM

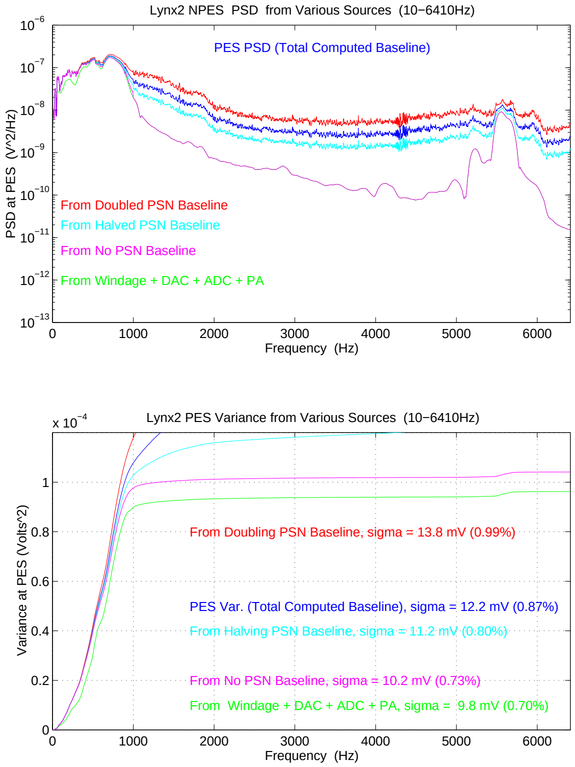


Figure 7.35: Effects of changing Baseline Position Sensing Noise (PSN) with the spindle at 7200 rpm. The top plot gives the effects of the different PSN levels on PES. The lower plot integrates these to show the variance.

## 7.11 PES Pareto Summary

For the purposes of feedback control, once we have done everything else right, we are limited [19] by latency around the loop and by the noise that Bode's Integral Theorem tells us we cannot completely eliminate [1]. Looked at another way, the noise we see at different measurement points can be interpreted as a filtered amalgam of different noise sources that have independently entered the loop at different points to be filtered through to our measurement point. Once the noise enters, Stein's "dirt digging" analogy makes it easy to see that eliminating sensitivity in one region of the frequency space must increase it elsewhere.

It follows then that a key to understanding the limits of performance of our control system then is to understand the noise environment in which we operate. The PES Pareto methodology allows us to isolate different noises as inputs to the system and then to combine them at specific measurement points – such as the error signal – so that we can rank the main contributors. While the method depends upon a few assumptions of linear analysis and the ability to measure and isolate PSDs of different noise sources, it is inherently practical and flexible, making use of whatever measurements are available. As such, the method is applicable to a wide variety of control systems that lend themselves to frequency domain analysis. The ability to scale these independent noise sources, and see the effects of this at different points around the loop, allow us to localize the key noise sources that should be the focus of engineering design effort. We hope that this is helpful.

The hardest and most ad-hoc part of PES Pareto is getting measurements to isolate the different noise inputs. However, much of this can be seen as a consequence of not building the measurements into out digital controller from the start, requiring physical loop breaking and auxiliary sensors and/or instrumentation to extract a quantity. At that point, an assumption of time-invariance needs to be made for us to believe that the measurement we made two weeks ago on the testbench still applies to the operational closed-loop system today. In the time of inexpensive ADCs and DACs, and parallel processing via multi-core CPUs, Systems on a Chip (SoC), and FPGAs, there is really little excuse not to build PES Pareto (as well as the system identification measurements of Chapter 3) into our digital controller. When we build these measurements into the digital controller, we can better ensure data consistency (even in the measurement frequency axes). Furthermore, since the patch panel is digital and the switches programmatic, a process that originally took months to generate can be done with the touch of a button or automatically from some supervisory loop in the digital controller. This again brings to mind some variant of Figure 7.15.

Given our need to have good models to use advanced methods, and given the need for those models to be derived from measurements, it should not be surprising if the measurement and modeling code inside of a well designed digital controller would be dramatically more substantial than the running code itself. AFter all, we have shown many examples when a relatively simple, but well tuned to a physical model, controller can perform almost as well as far more sophisticated algorithms that are not parameterized with measurement based data.

## 7.12 Minimizing Noise Before It Enters the Loop

One of the take aways from Stein's explanation of Bode's Integral Theorem [1] and the discussion above is that since attempting to suppress noise at one frequency range will amplify it in another range, it makes sense to try to minimize the noise before it enters the loop.

One way to manage this is through greater use of feedforward control when possible. If most of the reference signal following can be accomplished with a feedforward component [191, 192, 193] (which does not amplify any sensor noise), then the feedback loop can be optimized to minimize the effects of disturbance. Likewise, if external disturbances can be sensed [187, 188], this also changes the constraints of the feedback system design. We will touch on these methods in Chapter 8.

Without the use of auxiliary sensors or feedforward and repetitive control methods, we are really talking about filtering the main sensor signals of the loop (if they are baseband signals) or improving the demodulation of these signals, if the sensor signals are in-fact modulated onto another signal. Modulated sensor signals are far more commonplace than we think, for basic reasons of making the sensor signal more immune to DC offsets and other biases. This is even the case in the pulses fired by neurons in the brain. Thus, the latter part of this chapter will give an introduction to the use of demodulation in feedback loops and how a little bit of the math that control engineers generally know can go a long way in cutting noise and latency. By being careful about the demodulation methods used can dramatically lower the noise that is allowed into the feedback loop [185]. That discussion will start with Section 7.13.

That leaves us with the noise and latency injected by our baseband sensor signals and whatever filters we apply to them. Repetitive signals can often be identified and canceled using methods that end up being classified as feedforward (because they are outside of the normal feedback loop and when tuned have minimal negative phase in the canceling signal).

In a perfect world, we would be able to sample fast enough so that our anti-alias filter would provide minimum phase lag in the frequencies in which we want to control the system. For all other situations, it is a tradeoff between whether we can handle the filtering needs digitally or whether we need to add some well designed analog circuitry outside of our converts operating regions. The phase you save may be your own.

### 7.12.1 Thoughts About Anti-Alias Filters

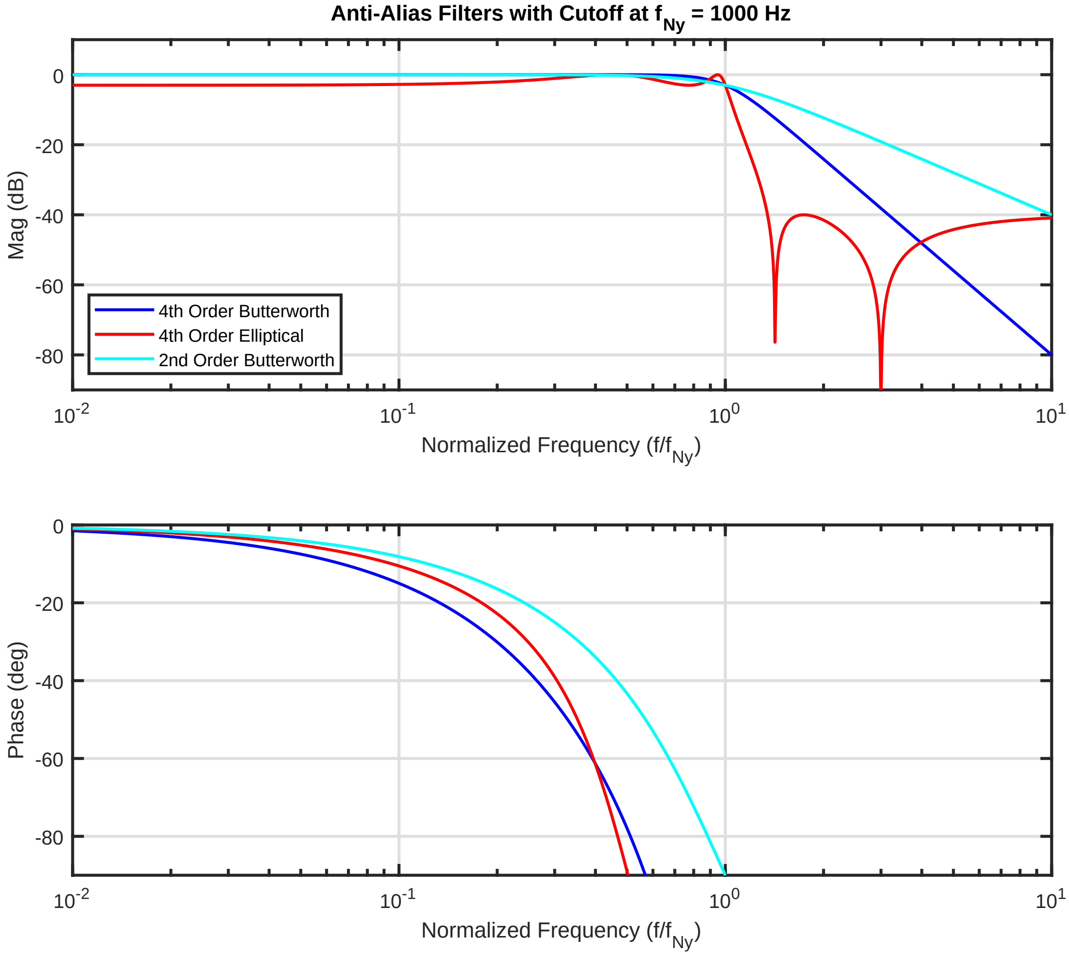


Figure 7.36: Frequency responses of various anti-alias filters. All filters have a DC gain of 1, with the passband ending at the Nyquist Frequency ($f_{Ny}$ = 1kHz here). Under an assumption that the Nyquist frequency is 5 or 10× the open loop crossover frequency, we can examine the filter phase response, which can significantly degrade the phase margin of the system, as documented in Table 7.1.

A perfect example for understanding the role of filters is in the use of anti-alias filters. Virtually every book on digital control will mention that designers should use an anti-alias filter to limit signals at frequencies above the Nyquist frequency into the feedback loop. That is, they recommend an analog filter applied to signals before the ADC that cuts off signals above the Nyquist rate.

| **Filter** | **Phase at $f_{Ny}/10$** | **Phase at $f_{Ny}/5$** | **Attenuation at $10 f_{Ny}$** | **Peak Gain Beyond Roll Off** |
|---|---|---|---|---|
| 4th Order Butterworth | $-15.0276°$ | $-30.1223°$ | $-80.1201$ dB | NA |
| 4th Order Elliptical | $-10.5523°$ | $-22.8086°$ | $-40.8932$ dB | $-40$ dB |
| 2nd Order Butterworth | $-8.1486°$ | $-16.4211°$ | $-40.0605$ dB | NA |

Table 7.1: Phase penalty of representative anti-alias filters. The corner frequency is chosen to be at the Nyquist frequency, half the sample frequency, $f_{Ny} = f_S/2$. Comparisons are made with respect to the Nyquist frequency, as it is considered the limit of intentional digital control action. The two Butterworth filters are flat in the passband, but they incur a larger phase penalty relative to the elliptical filter for stopband gain attenuation they provide. On the other hand, the elliptical filter has up to 3 dB magnitude distortion in the passband. Outside the passband, beyond the Nyquist frequency, the Butterworth filters drop off monotonically, while the Elliptical filter has lobes with the peak gain at approximately $-40$ dB at frequencies up to $100 f_{Ny}$ (beyond the range of the plot). Ultimately, the choice of anti-alias filter structure should not be separated from the available sample rate options, nor the robustness of the system to gain and phase distortions.

We are told that we should use anti-alias filters to remove high frequency signals from showing up as impostors in the frequencies below the Nyquist frequency. This makes sense, but one has to consider the phase effects of such filters. Ideally, we would have a filter remove all signals above $\frac{f_S}{2}$. Such an ideal rectangular filter would have a *sinc* function response in the time domain, and have infinite extent both in positive and negative time. This is probably hard to implement. Anything else produces phase effects and the closer the attenuation is to $\frac{f_S}{2}$, the stronger the negative phase effects will be on signals below $\frac{f_S}{2}$. The question then becomes: how much faster than the system dynamics is $\frac{f_S}{2}$?

We come back to this idea that for practical uses, one does not simply apply any old analog filter (just as one simply does not walk into Mordor). Instead, there is a tradeoff between attenuation and phase effects. This clearly gets easier if we can sample at a higher rate than the dynamics we care about. If we are barely meeting the Nyquist Criterion then it is very likely that any sort of anti-alias filter that has a corner frequency close to $\frac{f_S}{2}$ will have a serious phase effect on our loop. On the other hand, if we are sampling 20 times as fast as the highest dynamic we care to handle digitally, then a reasonable analog low pass filter above then Nyquist frequency probably does not impact our control design.

Here we run into a different issue: defining what is a reasonable low pass filter. A Butterworth filter has very smooth magnitude and phase response, but they phase effects kick in very early compared to the corner frequency. There are other analog filter designs (e.g. elliptical) that have ripple in the pass band of the filter, but that have a much smaller phase effect at low frequency. Again, it's a design tradeoff and a control engineer that wants their design to work in the real world does not leave these decisions to a circuit designer on their own. It is important to know enough, to be conversant enough in their art to be able to guide their choices.

Finally, the amount of anti-alias low pass filter attenuation that we need might not be so much if we understand our signal environment. If we know that we have a general level of falling off dynamics and noise, but that there are a handful of disturbance signals at high frequency, or harmonics of some fundamental signal that is being generated as digital square wave that gets filtered by the system, then we can attack these specific disturbances, either by eliminating the original source; via direct digital synthesis (DDS) of high frequency signals, or coherent demodulation of readback signals (Section 7.13) or attenuating the specific signals with targeted notch filters (Section 7.12.2). By removing these narrow band disturbances individually, we not only beat the problem into a much simpler form, but we also allow a much milder anti-alias filter that does not affect our loop phase nearly as much. It's the circle of life.

What is left out of these discussions is any talk of the phase effects of anti-alias filters, as demonstrated in Figure 7.36 and Table 7.1. One might desire maximum flatness in the pass band (frequencies below Nyquist) and a sharp drop after Nyquist, but filters with such sharp shapes are either not causal or have substantial effects on the system phase even a decade below the Nyquist frequency. The 4th Order Butterworth filter does not achieve 40 dB of attenuation until the frequency is at 3× the Nyquist Frequency. If all critical frequencies of the feedback loop are below $f_{Ny}/10$ then the phase penalty is only about 15 degrees, but in systems that can only sample at 10× the critical frequencies, this phase penalty doubles to 30 degrees. A 2nd Order Butterworth halves this phase effect, but does not get to 40 dB of attenuation until a full decade above the Nyquist Frequency. One can apply a 4th Order Elliptical filter to get better stop band attenuation and less phase effects, but this comes at the cost of inaccuracies in the magnitude of the pass band. In this case, the design limit was 3 dB, but that is still a gain variation of $\sqrt{2} \approx 1.4142$. Many control systems cannot tolerate a gain uncertainty of 40% in the passband.

At this point, we are affecting the basic performance of the system. If one chooses an anti-alias filter with higher bandwidth to combat this problem, signals above the Nyquist rate are allowed into the feedback loop. A similar thing happens if one chooses a less aggressive anti-alias filter. Moves made to decrease the phase effects result in more aliased signals. Eliminating more aliased signals severely affects the phase margin.

The discussion of anti-alias filters cannot reasonably proceed without a discussion of the converter circuits, the ADCs and to a lesser extent the DACs. At this point, we are not discussing the quantization, but the speed and the latency of the converters. Control engineers need to be aware that of the many types of ADCs, the fastest get their speed by pipelining the digital conversion in a "bucket brigade" of processing. For signal processing applications this means very little, but for a control engineer, the idea that my ADC samples at 1 MHz but has a pipeline delay of 20 sample periods is unacceptable. From a latency perspective, the engineer has paid extra money to get a 50 kHz ADC. There are ADCs which convert in a single cycle, such as Flash ADCs and Successive Approximation Register (SAR) ADCs. With those, the engineer knows what they are getting.

Conceptually, from a minimum phase and latency perspective, the control engineer would do well to specify ADCs that could sample at 10–20 times the desired Nyquist frequency. (This is obviously not always possible, given the speed requirements or the limitations on sampling imposed externally such as in a sectored servo hard disk drive.) If the engineer can specify such a high sample rate relative to what they need, then an anti-alias filter that provides reasonable rejection at the device Nyquist frequency would still impose 10–15° phase lag at the working Nyquist frequency. However, since the control engineer is very likely to not be the person specifying the mixed signal electronics, they need to come to the meeting fully aware of what they need and why. Imagine spending a year to robustly get 10% more bandwidth out of a system only to throw it away because of not making the case for a slightly more expensive ADC.

### 7.12.2 Analog Notches

Good place for some notch filter stuff from my notes.

As mentioned earlier, notch filters are an essential component of removing signal harmonics from signal paths. Often, these are left out of the documentation, but a look at the schematics of any mixed signal board interacting with a system's ADCs and DACs will reveal them. Analog notch filters are used to remove signals that we really just don't want to deal with in our digital system. Analog notch filters are most efficiently made using op-amps, and are often second order. The control engineer would do well to become familiar with a few of these devices, just to have the conversation with the circuit designer.

Sallen and Key, Twin-T Notch, basic notch.

Part of the attraction of a notch filter, compared to simply using a low pass filter, is that the phase effects can be quite low and localized. The two numerator zeros produce a $+180^\circ$ phase bump. Since the filter has to have at least the same number of poles, the two poles produce a $-180^\circ$ phase bump. If the two are close together, the phase effects are minimized away from the notch frequency. Thus, there is a positive incentive for making the notch narrow, for making it very high Q. The problem with this can be seen by paraphrasing Eli Wallach's Tuco in The Good, the Bad, and the Ugly: Whomever misses with a high Q notch, he'd better miss very well." That is, as with other high Q things described in this workshop, we had better get the frequency right, or we create a magnitude/phase/signal dipole that can really mess up the system. In the case of a filter external to the loop, this simply means that we have severe signal attenuation in a place where there is no harmonic or disturbance signal and the harmonic or disturbance we wanted to remove is not there.

## 7.13 An Introduction to Demodulation for Use in Feedback Loops

Understanding the utility and methodologies of using modulated signals in feedback loops is not common, and is usually limited to a particular type of modulation in a particular application. Seen primarily as belonging to communication theory, this subject gets little attention. Often only the simplest embodiments are presented, ignoring the potential noise, delay, and nonlinearities that these methods can introduce in the desired signal. While the simplest methods have the advantage of ease of implementation with simple analog circuits, they are decades behind the times. New digital (and analog) methods make practical a whole slew of coherent methods. Even then, these methods are often only understood from a communications perspective, where the lack of a feedback loop make the role of noise, delay, and nonlinearities in the sensor signal far less significant. To get the most out of our control loops, it is worth fundamentally understanding demodulation.

Still, whether by design or necessity, many of the signals that are used to mark position, velocity, and/or acceleration are modulated onto some carrier. The position signals in hard disk drives are in a pattern of alternating polarity magnetic domains. Motor control is often achieved via shaft encoders with alternating patterns around the circumference. Laser interferometers work by detecting the phase between a reference signal reflected off of a mirror at a fixed distance and a measurement signal reflected off of a mirror on a moving object.

Nature also makes use of modulation in the firing of neurons [214, 215] which "fire" not with a DC level, but with the presence or absence of a string of pulses. A higher level of firing corresponds to a higher frequency of the pulses, not a greater amplitude. As these pulses are all non-negative, demodulation of the neuron signal can be done simply with averaging. Why would nature choose to use a modulated signal? Modulating a signal makes it less susceptible to offsets and baseline noise. Baseband (DC) signals often can't travel far, encode biases and offsets, and are susceptible to drift.

Often, a modulated signal is the only way to encode a position or velocity measurement with sufficient signal-to-noise ratio (SNR) for feedback control. Sometimes, we are looking for the effects of the system on the modulated signals. Sine-dwell (also known as swept-sine in industry) measurements of dynamic systems rely on the system's response to a particular sinusoid at a particular frequency. For atomic force microscope (AFM) measurements of soft biological samples [216, 217, 27], it's more advantageous to tap the probe tip across the surface using an AC drive signal, so that the surface experiences compression forces but minimal shear. In these problems, extracting the surface's affect on the cantilever oscillation is the key to characterizing the surface. The faster and more accurately an AFM can do this, the more effective the measurement [85, 86, 218, 219].

No matter what the original motivation, we often need to demodulate these signals to actually use them in a feedback loop. How we do this demodulation depends not only on the modulation method, but on the technology available to do the math. The advent of improved digital electronics has allowed for a far more exotic set of modulation/demodulation schemes. For use in feedback control loops, we must further consider the latency of the demodulation computation and the achievable SNR from a method. This paper will attempt to give a cohesive overview of the different demodulation methods as they are applied in feedback loops.

For the purposes of feedback control, once we have done everything else right, we are limited [19] by latency around the loop and by the noise that Bode's Integral Theorem tells us we cannot completely eliminate [1]. A practical linear analysis of noise through a loop can be accomplished using a PES Pareto methodology [2], but the final takeaway from that method is that one should pay attention to eliminating noise before it enters the loop. While the simplest, non-coherent, "slow and noisy" demodulation methods may suffice for a large set of important problems, there are problems for which doing a little bit more math – in the right way – can dramatically improve the signal to noise and dramatically lower the latency of the extracted signal. In other words, we can minimize some of the factors that limit the performance of our feedback loop. Those factors are hard to limit from control design only.

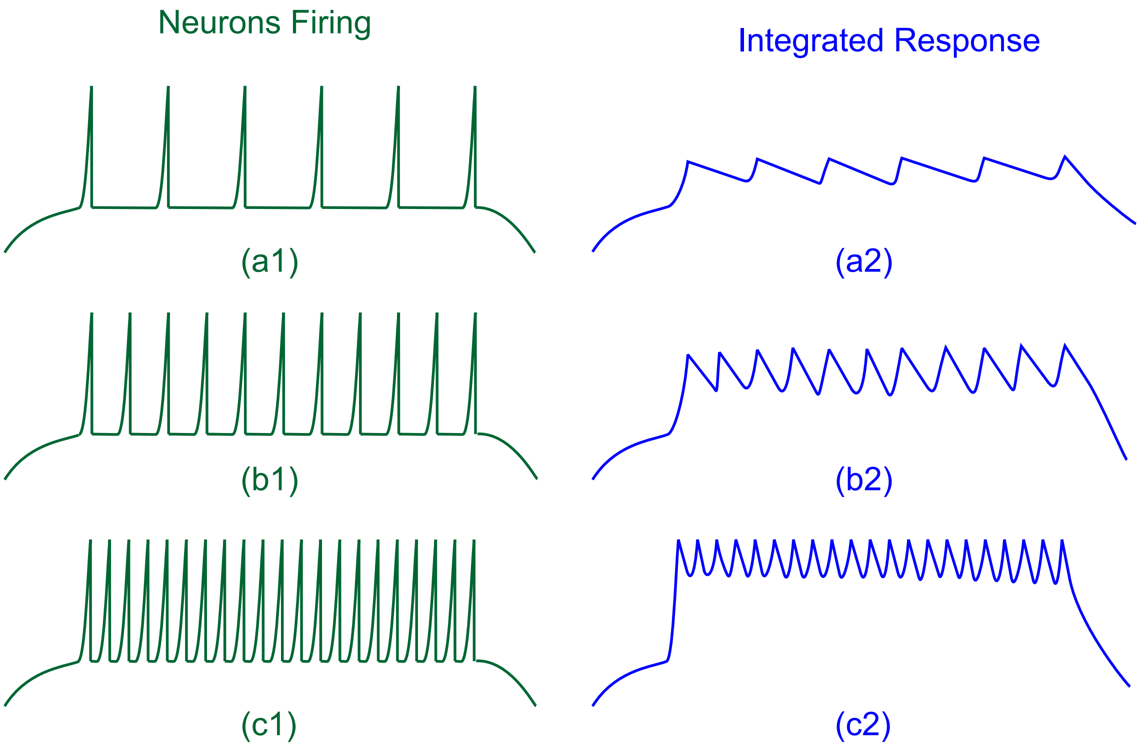


Figure 7.37: Drawings of pulse modulation of the type used by neurons firing. On the left, we see that an increase in the input to a neuron causes an increase in the frequency (not the magnitude) of the pulse train. On the right, we see the simplest demodulation, a kind of averaging, that can be done on the pulse train to extract a lower frequency value.

## 7.14 Pulse Modulations

One family of modulation schemes involves modulating pulses of fixed height, either by their position (Pulse Position Modulation, PPM) or by their duty cycle (Pulse Width Modulation, PWM) . One of the simplest forms of modulation can be found in nature, where neurons "fire" with a series of positive voltage pulses. The absence of the pulses can be viewed as not firing, but once the pulses do start firing, an increase in signal is denoted not by a higher amplitude, but a higher frequency of the positive pulses [214, 215]. This is sketched in Figure 7.37. Clearly, the lack of pulses denotes a baseline for the received signal, but the level above a baseline must be obtained through some finite length averaging of the pulse train, which can be seen on the right side drawings, symbolizing integration of their counterparts to their left. The higher the pulse frequency, the higher the baseline of the integrated result.

It is worth noting that averaging only works when the signal is single sided and that the speed of obtaining a usable output is limited by the length of the average taken.

Outside of nature, pulse-position-modulation (PPM), pulse-frequency-modulation (PFM), and pulse-width-modulation (PWM) use essentially the same ideas. While PFM is the closest analog to neurons, PWM is perhaps the most ubiquitous, showing up in many forms of low level, slow control. While it shows up in communication systems [30] as NRZ data, for the purpose of control, it is usually used as a method to encode a control input or output signal into a binary, [0,1] signal.

Pulse Position Modulation (PPM) is usually a matter of whether a pulse is there or not. The meaning of the signals is – in large part – related to whether the output signal is meant to be a baseband (low frequency) value or whether it is a driving a clock or oscillator signal. In the former case – and if the pulses are all single signed (e.g. all positive) then a low pass filter (LPF) will average the signals and give a usable low frequency output. If the thing being driven is a clock or oscillator, then the leading edges can go through a phase-detector (along with the clock leading edges) to generate a phase error (Section 7.18). Pulse Frequency Modulation (PFE) is similar to PPM, but the value of the modulation is encoded in the density of the pulses. When driving a clock, there is usually a minimum pulse rate to allow the clock (a phase tracking loop based on a PLL) to maintain synchronization.

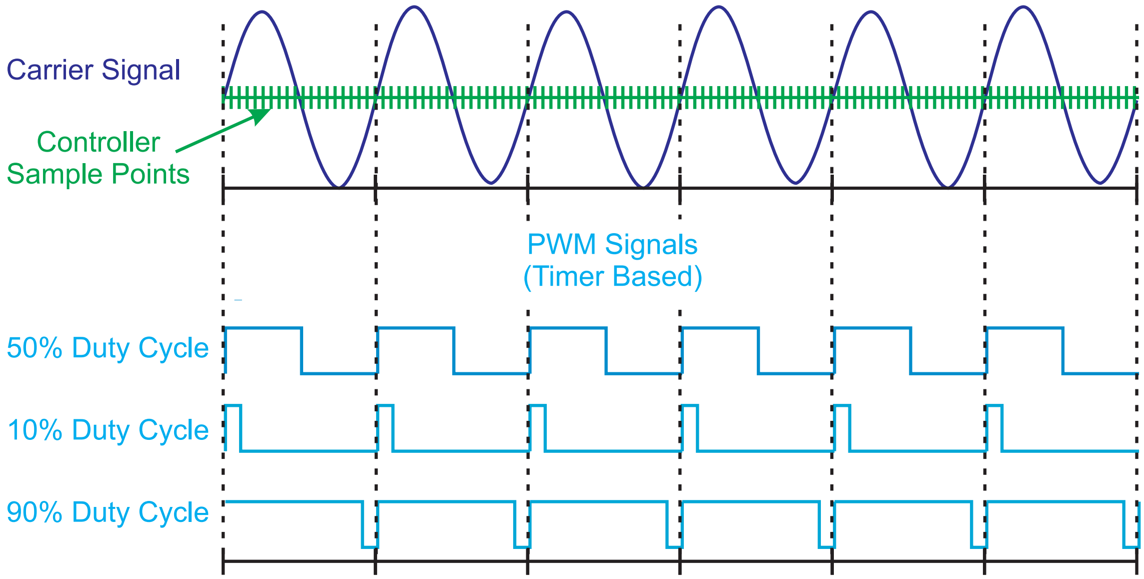


Figure 7.38: Classic Pulse Width Modulation (PWM). The numerical input modulates the duty cycle of the pulses with respect to a fixed carrier frequency.

Pulse Width Modulation (PWM) (Figure 7.38) is commonly used in control systems as a cheap substitute for a digital-to-analog converter (DAC). The control signal that has values ranging from 0 to 1 is modulated into the width of a pulse stream with 0 being represented by a 0% duty cycle and 1 being represented by a 100% duty cycle. PWM allows the modulated signal to be transmitted as binary data on a digital signal line, greatly increasing the immunity to low level random noise. The digital signal – being received at the plant – is simply low pass filtered (LPF) to produce an averaged output that once again spans the [0,1] range. This signal can be converted into a voltage and amplified to drive an actuator.

Pulse Width Modulation (PWM) need not have a nonzero offset, although this enables the most trivial form of demodulation: averaging. For our purposes, averaging and low pass filtering are used interchangeably in this context. PWM is a critical method to understand in part because of its ubiquity in controlling slow processes, such as voltage references, pressure, and temperature. In these contexts, the control law is most likely a PID or PI controller producing an "analog" (multi-bit) value. These values could be sent to the plant via a Digital-to-Analog Converter (DAC), and transmitted via an analog signal line, but this is more expensive than the PWM function and the analog signal is more susceptible to noise than a binary signal that is pulse-width-modulated. The PWM signal, transmitted on a binary signal line, can have small amounts of noise cleaned up by simply adding a relay centered at the voltage equivalent of 1/2 to estimate if the received signal was at logical 0 or logical 1. Once inside the receiving device, an average of the signal recovers the original multibit value (between 0 and 1). This can then be appropriately scaled to drive whichever device is being controlled.

PWM relies on the assumption that the actual signal value changes far more slowly than the modulation. This means that a slowly changing "analog" value can be encoded into the PWM and the averaging of the 0-1 binary input will produce a value on the [0,1] range that represents the original multi-bit signal. While demodulating the signal can be accomplished simply with LPF/averaging, modulating the slowly changing signal into PWM involves producing a binary signal that is changing far faster than the "multibit" signal. This would require a microprocessor to spit out signals at a far faster rate than the control loop signaling. For example, a processor sampling analog data at say 0.5 Hz might need to modulate this signal two orders of magnitude higher, say at 100 Hz, so that the averaged/LPF signal was relatively smooth when looked at with a 0.5 Hz sample rate.

For this reason, many microcontroller chips include a bit of PWM logic that accomplishes the modulation without the processor needing to "bit bang" – a term used to describe when a processor spends a significant fraction of its processing time merely flipping bits at a high rate. The function can also be accomplished using programmable logic (PL) such as an FPGA (Field Programmable Gate Array).

## 7.15 Basic Modulation of Sine Waves

Another family of modulations used in control systems can be explained as variants on modulating a sinusoidal carrier. Consider the carrier signal, $c(t)$,

$$c(t) = A_0 \sin(\omega_0 t + \theta_0), \tag{7.77}$$

where $A_0$ is the fixed amplitude of the carrier, $\omega_0 = 2\pi f_0$ is the carrier frequency, and $\theta_0$ is the carrier phase. Often $\theta_0$ is defined as $0$ for simplicity. For **amplitude modulation (AM)**, we modify the carrier, $c(t)$, by changing the amplitude [220] i.e.

$$s(t) = (A_m(t) \cdot A_0) \sin(\omega_0 t + \theta_0). \tag{7.78}$$

**Phase modulation (PM)** [221] is shown in

$$s(t) = A_0 \sin(\omega_0 t + (\theta_0 + \theta_m(t))), \tag{7.79}$$

while **frequency modulation (FM)** [222] is shown in

$$s(t) = A_0 \sin((\omega_0 + \omega_m(t))t + \theta_0), \tag{7.80}$$

The development of sophisticated electronics has allowed for more complex modulation schemes, such ad the amplitude-phase modulation schemes (e.g. QAM) that enabled faster modems in the days before high speed Internet connections [223].

## 7.16 Non-Coherent AM Demodulation

It is worth making a distinction here between AM and PM/FM signals. AM signals, like the common use of Pulse Modulated Signals, can be demodulated – if one is not too precise about values or timing – using non-coherent methods. That is, a carrier or clock need not be used. There is a cost to using non-coherent methods (no clock synchronization), but the simplicity of the circuits often make up for it. Phase and frequency modulation require the use of a precise mixing signal (a signal normally at the carrier frequency) which requires more complex electronics.

Sticking with non-coherent methods, we see one of the most common forms in the drawings of Figure 7.39. On top we see a diode bridge circuit which is one way to build a full wave rectifier using passive

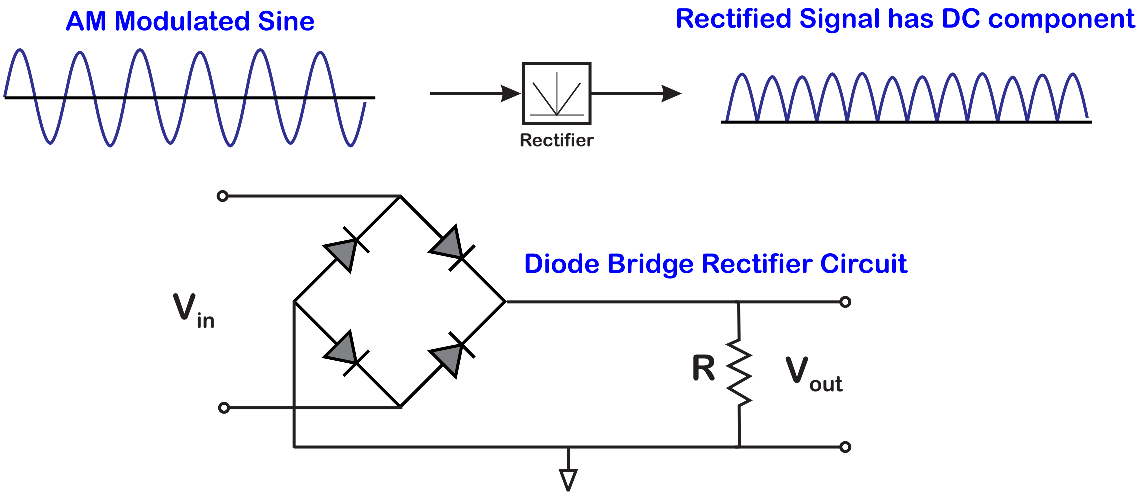

AM Modulated Sine
Rectifier
Rectified Signal has DC component
Diode Bridge Rectifier Circuit
$V_{in}$
R
$V_{out}$

Figure 7.39: Non-coherent demodulation of an AM signal with a rectifier made from a passive diode bridge. The amount of ripple in the output can be limited by adding a low pass filter on the output. It can be something as simple as adding a capacitor across $V_{out}$.

circuits. The output of the bridge is tied to a resistor so that if the input is an amplitude modulated (AM) sine wave (upper left), the output (upper right) is rectified. That rectified signal can now be averaged using a low pass filter (LPF) to show extract the amplitude modulation.

The issue is that many sensor signals are modulated on a carrier, and those signals need to be demodulated off that carrier in order to be useful to the rest of the controller. The classical demodulator uses a bridge circuit as a rectifier, as diagrammed in Figure 7.39. The circuit has the advantage that it is passive (all diodes and passive elements) and that it works at all frequencies (until the parasitic capacitances of the circuit kick in which is probably far beyond the frequencies that we will care about here). The circuit of Figure 7.39 is often followed by a low pass filter which serves as an analog averager. In Figure 7.39, we see that $V_{out}$ has a DC component (what we want for our control loop) and a component at 2× the carrier frequency. That is the ideal behavior. We put a low pass filter on the system such that most of the $2f_C$ signal goes away and we get a smooth output. How smooth that output needs to be determines the lower cutoff of that low pass filter, but the lower the cutoff, the more it can impact our loop signals.

This has served well for years but it has several limitations. If we consider an input signal as being modulated on a carrier, $f_C$, then passing it through even a perfect rectifier will produce a desired component of the signal around DC, but also at every even harmonic, $2f_C$, $4f_C$ .... Furthermore, any broadband noise that might have had 0 DC content now shows up in the demodulated signal, because while it may have been zero mean at the input of the bridge circuit, the rectification means that the zero mean noise is now biased noise. Furthermore, as we have a tighter requirement on the accuracy of our filtered output, we have to filter for longer. Very realistically, that translates into the number of signal cycles needed before we detect a change in the signal level. This can very realistically be tens or hundreds of cycles, and so by using one of these circuits, we have potentially added a large chunk of latency to our control system.

There is an inherent assumption that the modulation is at frequencies far lower than the carrier frequency, $f_C$. The averaged signal will exhibit different levels of ripple (signal at $2f_C$ getting past the LPF) depending upon the LPF itself, but generally we see that the averaged signal can return the modulated amplitude. One implementation of this is known as an analog RMS-to-DC circuit[224]. In this reference the example of a 36 ms settling time of the AD736 [224] is 3,168 periods of the 88 kHz signal used in [85]. A 36 ms settling time sets the Nyquist frequency at 0.5*(1/36e-3) = 13.89 Hz. From this a reasonable closed-loop bandwidth limit would be 1/10 of the Nyquist frequency or about 1.4 Hz, which severely limits achievable bandwidth from a fairly high speed signal.

For many low speed, low to moderate precision control problems, this type of signal demodulation is sufficient. The disadvantages of such a simple scheme are that it allows through broadband noise, nonlinearities, and harmonics of the carrier frequency. Adding more LPF to minimize these effects will also lower the effective bandwidth of the demodulated signal.

## 7.17 Basic IQ Demodulation: Lock-In Amplifiers

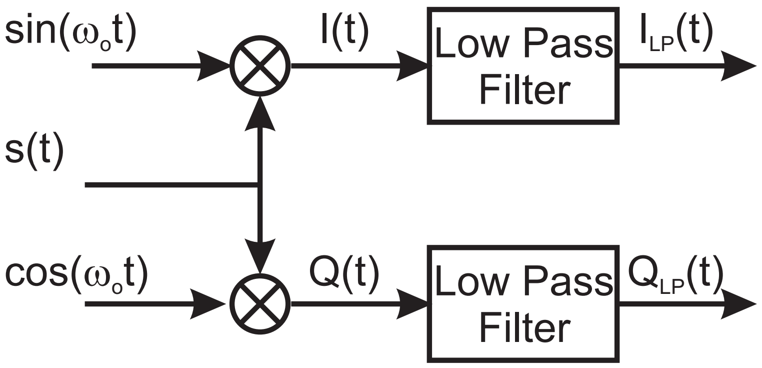


Figure 7.40: Operation of a lock-in amplifier.

The next important demodulation component to understand is the in-phase/quadrature (IQ) demodulator. These are commonly used in communication systems and in precision instrumentation. Among the simplest of these instruments to understand is the lock-in amplifier (Figure 7.40). Lock-in-amplifiers (LIAs) and coherent demodulation have typically been used in a variety of communication and measurement systems [75, 53, 70, 84, 87]. The difference is that those systems did not close the feedback loop and thus were not affected by latency. Furthermore, many LIAs require post-integration low pass filtering to minimize the effects of harmonics [225, 226].

Lock-in amplifiers are common measurement instruments which produce an in-phase and quadrature signal to mix with the incoming modulated sinusoid. In Figure 7.40, the input signal is mixed with a sine and a cosine at frequency $\omega_0 = 2\pi f_0$. By convention one signal, $I(t)$, is called the in-phase signal and the other, $Q(t)$, is called quadrature. The mixing signals will cause any component of $s(t)$ at $\omega_0$ to produce a signal at baseband (no carrier) and one at $2\omega_0$. The low pass filters are supposed to remove this 2X frequency harmonic, as well as anything else at high frequency. Generally any signal not at $\omega_0$ should average out in the low pass filters.

The integrated in-phase ($I(t)$) and quadrature ($Q(t)$) branches now have signals from which the magnitude and phase can be extracted via a standard rectangular to polar coordinate transformation. That is, if

$$s(t) = A\sin(\omega_0 t + \theta), \text{ then} \tag{7.81}$$

$$A = \sqrt{I(t)^2 + Q(t)^2} \text{ and } \theta = \arctan\frac{I(t)}{Q(t)}. \tag{7.82}$$

Again, there are inherent assumptions about $A$ and $\theta$ having frequency contents far below $\omega_0$. Lock-in amplifiers are very useful laboratory instruments for examining signals when delay doesn't matter much, but when we chose to use IQ demodulation in feedback loops, we are far more concerned about delay. A second issue is doing the computations of Equation 7.82. Square root and arctangent functions, are relatively expensive for real-time computations. Even the famous CORDIC algorithms developed in the 1950s for a US Air Force computer and used in the original HP-35 pocket calculator [227, 228] typically took 1 computation cycle per bit of accuracy, meaning that 32 bits of accuracy took up to a whopping 32 computer clocks. While this seems relatively small by current standards, it is not when the computation is being pushed by extremely high sample rates, as found in nano-mechatronics. For such systems, both the number of cycles and the variability of the number of cycles needed for any particular computation, are a particular hindrance to high bandwidths. We will see later, that there are ways of mitigating this. First, we need to introduce one more foundational piece, the phase-locked loop (PLL).

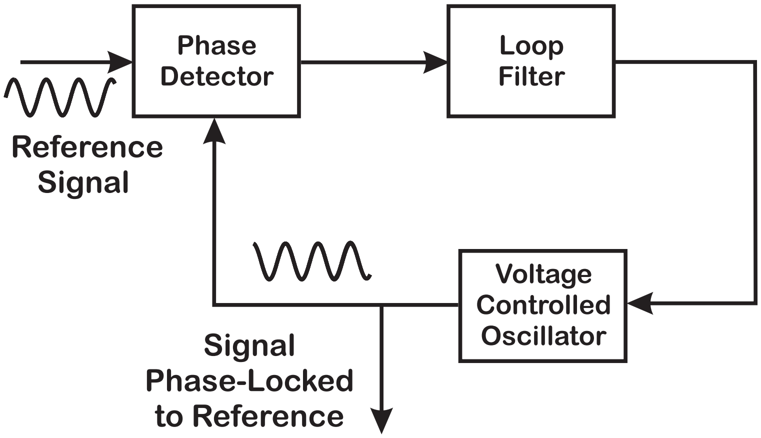

Phase Detector
Loop Filter
Reference Signal
Voltage Controlled Oscillator
Signal Phase-Locked to Reference

Figure 7.41: A general PLL block diagram. Each PLL has a phase detector, an oscillator, a loop filter, and operates in feedback.

## 7.18 Basic Coherent Demodulation: Phase-Lock Methods

One last critical component in this modulation/demodulation universe is that of phase-lock methods. The most basic component is the phase-locked loop (PLL) (Figure 7.41), which will be described below, but these methods are far more general and thus deserve their own mental subset. This author has argued that with PLLs in every computer, smart watch, television, radio, and generally any piece of digital electronics, the PLL is the most ubiquitous feedback loop designed by humans [30]. PLLs are unique amongst most feedback systems in that they include two intentionally inserted nonlinearities: the voltage or numerically controlled oscillator (VCO/NCO) and the phase detector. The former generates an oscillator frequency in response to an input voltage level or number and the latter extracts that phase and/or frequency from combining a pair of oscillating signals [156, 229, 230, 155, 231, 232].

In its most mathematically pure form, a PLL involves the same mixing (multiplying of two sinusoids) described in Section 7.17 on IQ demodulators. The difference here is that the latter are open-loop devices while PLLs are feedback mechanisms. While the non-coherent demodulation methods described in Section 7.17 are only useful for extracting the amplitude of a modulated signal. A PLL – by aligning the internal oscillator with the fundamental oscillation of the incoming signal – allows the phase (and sometimes frequency) of the incoming signal to be determined. This then opens up a world of other demodulation methods [233, 234, 152, 125].

First introduced in the 1930s [235], a classical analog phase-locked loop (Figure 7.42) takes a reference sinusoid and mixes (multiplies) another sinusoid that is conceptually in quadrature (90° out of phase)

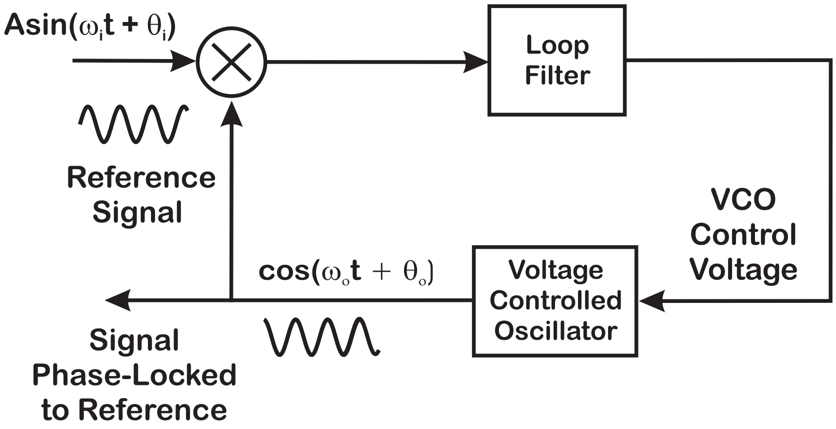


Figure 7.42: A classical mixing (analog) phase-locked loop.

with it. Via the use of trigonometric identities and low pass filtering (LPF), the loop creates an error signal in the baseband which is a sector 1-3 nonlinearity. This allows the loop to be closed and the difference signal – corresponding to the phase – to be driven to $0$.

Because the low frequency (baseband) behavior of the multiplied sinusoids is a sine – a quadrant 1 and 3 nonlinearity (Figure 7.46), the phase detector output signal can be used as an error signal to drive the frequency and phase differences between the reference sinusoid and the internal sinusoid to some constant value or 0 (depending on the system type). The internal sinusoid then represents a filtered or smoothed version of the reference sinusoid.

For multibit analog input signals, one group of digital PLLs (DPLLs) approximates the analog loop in the same sense way that digital controller approximate analog ones: the modulated carrier signals are digitized using an ADC, acted upon by a set of digital filters, fed into a numerically controlled oscillator (NCO), and that signal can be converted back to analog if needed using a DAC. However, this limits the carrier signals to only those that can be effectively digitized by an ADC. PLL circuit designers, being a clever bunch, have found ways to modify these methods so as not to require a full ADC conversion. For these binary digital signals, Walsh functions replace sinusoids. Special phase detectors work on edges of clock signals, or even on simply [0,1] bits coming in from a communication link. In these cases, the analysis moves from the comfortably analytical methods that use trigonometric identities on sinusoids to almost purely heuristic methods based on an intuitive understanding of the phase detector behavior both in its baseband signals (the demodulated ones) and in the residual high frequency signals. Although uncomfortable for those of us used to having analytical descriptions, these circuits are ubiquitous in low power digital electronics and therefore should not be ignored.

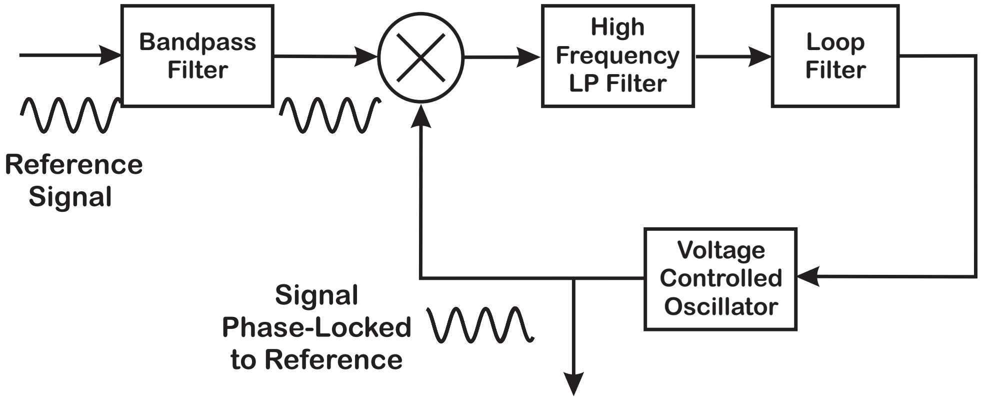


Figure 7.43: A practical version of the classic mixing phase-locked loop: note the addition of a bandpass filter preceding the loop to limit input noise and a high frequency low pass filter within the loop to attenuate the 2X frequency component with minimal impact on the loop dynamics.

Typical block diagrams of PLLs in the literature resemble Figure 7.42, however practical loops often more closely resemble Figure 7.43, in which a high frequency low pass filter is used to attenuate the double frequency term and a bandpass filter is used to limit the bandwidth of input signals to the loop. A general sinusoidal signal at the reference input of a PLL as shown in Figure 7.43 can be written as:

$$v_i = R_1(t) = A\sin(\omega_i t + \theta_i). \tag{7.83}$$

Without loss of generality, we can assume that the output signal from the voltage-controlled oscillator (VCO) into the mixer is given by

$$v_o = VCO_{out}(t) = \cos(\omega_o t + \theta_o). \tag{7.84}$$

The output of the mixer in Figure 7.43 is then given by

$$v_d = Mixer_{out}(t) = AK_m \sin(\omega_i t + \theta_i)\cos(\omega_o t + \theta_o), \tag{7.85}$$

where $K_m$ is the gain of the mixer.

Typically, analysis of such a PLL is done by taking several simplifying steps. Using the familiar trigonometric identity in terms of the PLL:

$$\begin{aligned} 2\sin(\omega_i t + \theta_i)\cos(\omega_o t + \theta_o) = \\ \sin((\omega_i + \omega_o)t + \theta_i + \theta_o) + \sin((\omega_i - \omega_o)t + \theta_i - \theta_o) \end{aligned} \tag{7.86}$$

and then making two fundamental assumptions leads to the commonly used model of the analog PLL. Let $\theta_d = \theta_i - \theta_o$. Then these assumptions are:

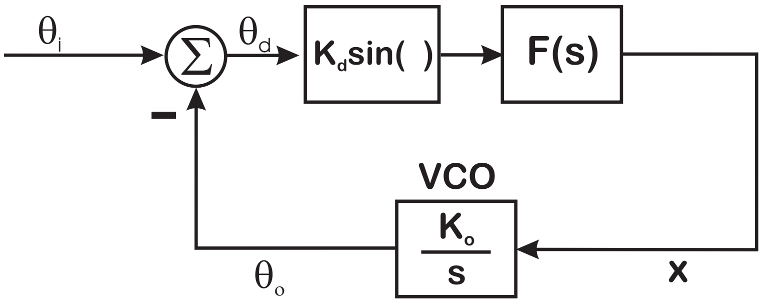


Figure 7.44: Conceptual block diagram of PLL with sine detector. This is a transition stage in the analysis of the classical mixing loop. This model represents the effect of the multiplying detector once the high frequency component has been attenuated.

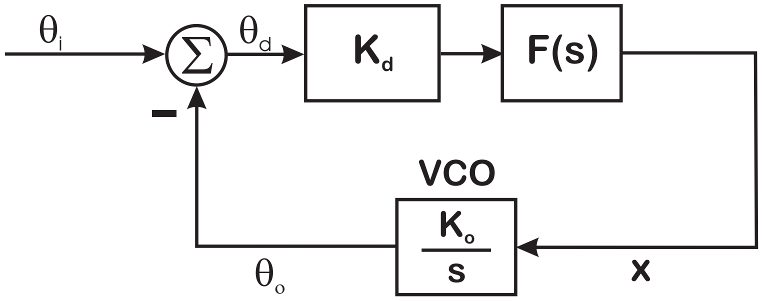


Figure 7.45: Conceptual block diagram of linear PLL. This is derived from the sine detector loop by assuming that the phase error is small and thus $\sin(\theta_d) \approx \theta_d$. This is the model with which most analyses of phase-locked loops are done.

1. The first term in (7.86) is attenuated by the high frequency low pass filter in Figure 7.43 and by the low pass nature of the PLL itself.

2. $\omega_i \approx \omega_o$, so that the difference can be incorporated into $\theta_d$. This means that the VCO can be modeled as an integrator.

Making these assumptions leads to the PLL model shown in Figure 7.44. The problem is that this is still a nonlinear system, and as such is in general difficult to analyze. The typical methods of analysis include:

1) Linearizion: For $\theta_d$ small and slowly varying

$$\sin\theta_d \approx \theta_d, \; \cos\theta_d \approx 1, \text{ and } \dot{\theta_d}^2 \approx 0.$$

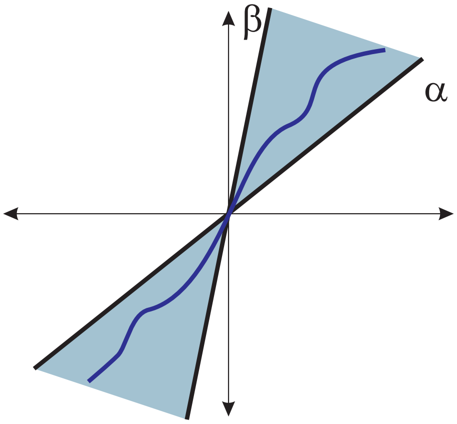


Figure 7.46: A quadrant 1-3 sector nonlinearity assumed for almost all PLL phase-detectors.

While this is useful for studying loops that are near lock, it does not help for analyzing the loop when $\theta_d$ is large.

2) Phase plane portraits [156, 230]. This method is a classical graphical method of analyzing the behavior of low order nonlinear systems about a singular point. The disadvantage to this is that phase plane portraits can only completely describe first and second order systems. The saving grace here is that by far the vast majority of PLLs are first or second order.

3) Simulation. Note that explicit simulation of the entire PLL is relatively rare. Because the problem is stiff, it is more typical to simulate the response of the components (phase detector, filter, VCO) in signal space and then simulate the entire loop only in phase space.

The linearized model is used for most analysis and measurements of PLLs. We can still analyze the the sinusoidal phase detector model shown in Figure 7.44, it has been possible to apply the technique of Lyapunov Redesign [153] to phase-locked loops [152, 30]. This can even be applied when the phase detector is digital, but the rest of the loop is analog, known as a classical digital phase-locked loop [125].

Changing the phase detector and VCO can result in a system for which this model is very accurate. It is possible to learn quite a bit about the phase behavior of the PLL from linear analysis.

Again the difference with an I-Q demodulator is the latter is an open-loop device – never minimizing the phase difference between input and output signals – while a PLL closes the loop on the phase difference so as to drive it towards 0. In contrast to an I-Q demodulator, in a PLL, the magnitude of the demodulated signal is trivially available from the remaining baseband signal. The phase error can be viewed as a residual instantaneous phase difference between the input and the locked oscillator signal. In an IQ demodulator, the magnitude and phase are typically extracted via a rectangular to polar coordinate transformation as shown in Equation 7.82.

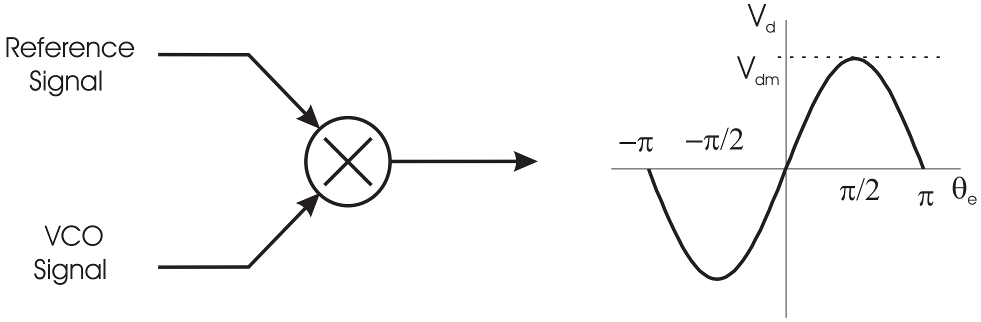


Figure 7.47: Classical mixing phase detector

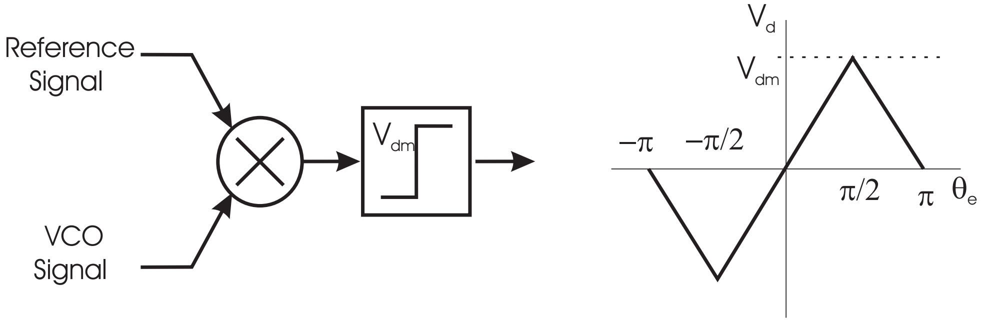


Figure 7.48: Over driven mixing phase detector

A classical mixing (multiplying) phase detector is shown in Figure 7.47. Once the $2X$ frequency component has been integrated out with the loop filter (and any high frequency low pass filter), the resulting phase characteristic is the sinusoidal one that we discussed earlier.

If one overdrives the circuit so that it saturates, then we get the phase response that is shown in Figure 7.48. Understanding the output of such a phase detector relies on a combination of averaging analysis and heuristics. However, one of the more interesting features of such a phase detector is that it can be implemented using an Exclusive-OR (XOR) gate as shown in Figure 7.49. One advantage of such a phase detector is that the loop gain is now independent of input signal amplitude. Furthermore, an XOR phase detector's response can have a larger linear range than a sinusoidal detector (mixer). The disadvantage is that the linearity of the baseband response is affected by the relative duty cycles of the input and VCO signals [155, 231]. The standard analysis done by PLL engineers involves drawing out square waves as shown in Figure 7.50 and then doing some heuristic "analysis" to convince themselves that the baseband (low frequency) component of the signal behaves with the triangular phase response shown in the right of Figure 7.49 (for a 50% duty cycle of the input signal).

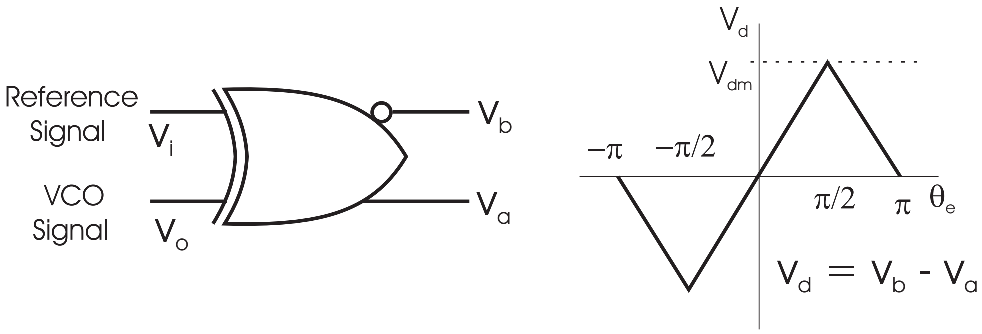


Figure 7.49: Phase detection using an XOR gate. Note that this accomplishes the same thing as an over driven mixer, but with digital circuitry.

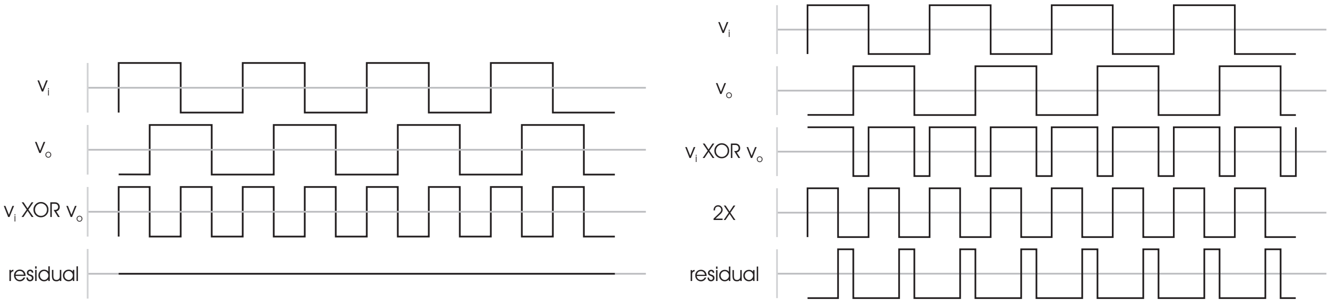


Figure 7.50: Phase detection using a XOR gate. On the left, a phase shift between reference and VCO output of $\pi/2$ produces an output of the phase detector whose baseband component is 0. On the right a relative phase shift of $\pi/4$ results in an output of the phase detector whose baseband component is $v_d/2$. The output is broken up here into a 2X frequency signal and a residual. The 2X signal averages to 0, while the residual averages to the baseband phase error.

Even more sophisticated digital circuits, such as a phase-frequency detector can integrate pulses to lock in not only the phase, but the frequency of an incoming signal. The analysis for such loops is often heuristic and graphical, but since most PLLs model the oscillator – either a voltage controlled oscillator (VCO) for analog loops or a numerically controlled oscillator (NCO) for digital loops – as an integrator and the loop filter often contains another integrator, the analysis often follows that of control of an integrator plant under PI control.

If all modulated signals were simply sinusoids, demodulated with analog circuits, the field of phase-locked methods would be a lot duller. Because accurate sinusoids are often difficult to produce in simple circuits and difficult to maintain across a circuit, deviations using non-sinusoidal shapes have arisen, and phase detectors beyond mixers (multipliers) have been employed. These circuits are even more nonlinear than the classical PLL, but they have clear advantages in simplicity and large scale reliability. A perfect exam-

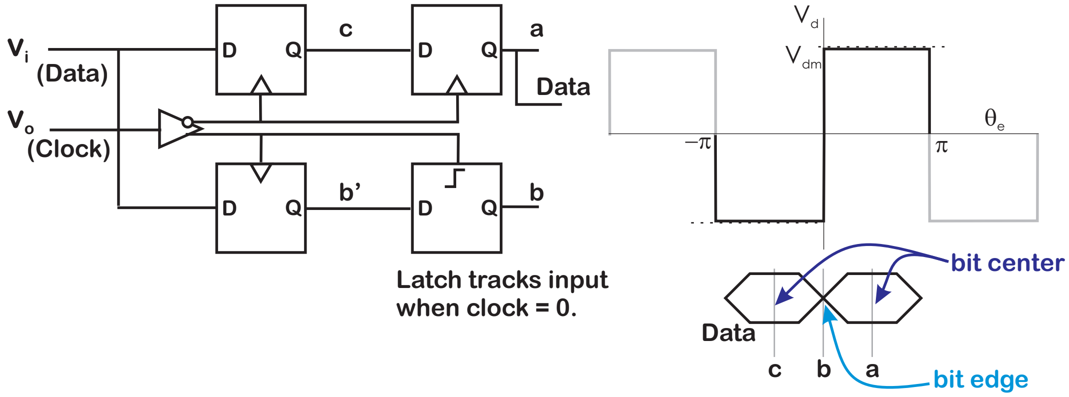


Figure 7.51: Block diagram for a Bang-Bang phase detector used in clock-data recovery PLLs.

ple involves replacing the sinusoids with square waves and the mixer phase detector with an Exclusive-OR circuit. The signals are binary, but the baseband behavior of the output of the XOR follows that first and third quadrant nonlinearity [30].

The point of this discussion is not to teach a lot of phase detector circuits (see [30] for that), but to point out that phase-lock methods – and the ideas behind them – can be used in all manner of signals that do not match our typical control system signals. Understanding that even these pulse trains and/or binary signals can yield information allowing us to synchronize an oscillator (or some mixing signals for an IQ demodulator) allow us to extract clean signals from all manner of encodings. As strange a device as the so-called Bang-Bang phase detector of Figure 7.51, can produce results that, when averaged to reveal the demodulated low frequency behavior, yield understandable phase relationships as shown in Figure 7.52 [236, 237].

It is worth understanding the Costas Loop, shown in Figure 7.53 because its progeny show up repeatedly in designs. We will see an example of one in the Laser Interferometers described in Section 7.23. The upper branch of the loop, labeled as quadrature, acts like a PLL, driving the phase between the input and the mixing signal to $0$. When the phase error in the upper branch is driven to near $0$, the oscillation in the lower branch is in phase with the input signal. In the absence of modulation, the two in-phase signals multiplied together essentially form a $\sin^2(\cdot)$ quantity which would produce an always positive value, especially when integrated via any low pass filter. Any amplitude modulation may be extracted almost trivially at this point. While the Costas Loop is most commonly associated with communication signals using Binary Phase Shift Keying (BPSK) this basic idea can be extremely useful in speeding up

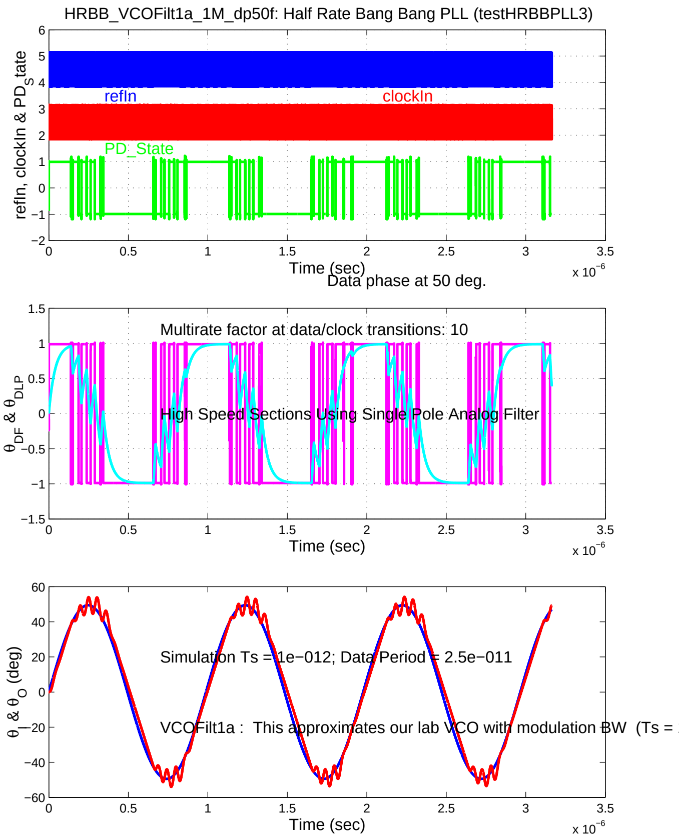


Figure 7.52: Time domain response of Half-Rate Bang Bang PLL simulation. Vertical fuzziness seen in phase detector output due to filters used by Matlab's decimation feature. The refIn signal is the data input. The clockIn is the recovered VCO clock. The PD State is the state of the phase detector. This is passed through two different low pass filters. The LP PD Out is low passed with a 4 GHz bandwidth. The VLP PD Out is the phase detector output passed through a 400 MHz bandwidth filter. In the bottom plot, the input phase is $\theta_i$ and the recovered clock phase is $\theta_o$.

precision IQ demodulation (Section 7.19) or in laser interferometers (Section 7.23).

Phase-lock methods allow us to synchronize an internal oscillator with some external reference signal. In doing so we use feedback to get phase alignment between signals, which simplifies a lot of other signal extraction. In some cases, we will see that the alignment provided by phase-lock methods allows a trivial extraction of other signal information, greatly reducing the computational load.

PLLs have made possible phase encoding of radial position on hard disk drives (HDDs), as shown in Figure 7.54. Once the PLL locks on the sync field, the phase detector provides left-right position information in the form of a phase delay or a phase advance.

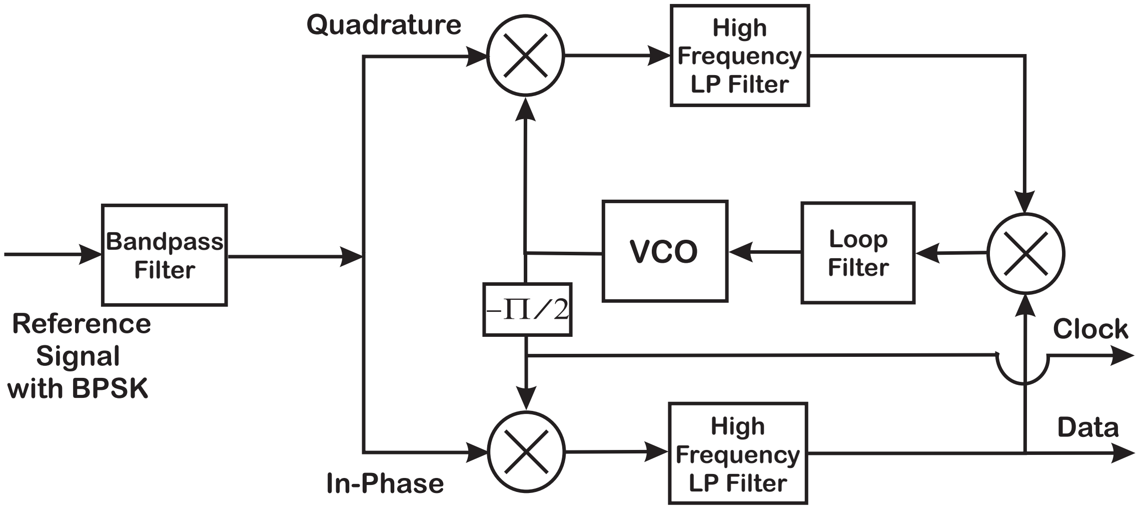


Figure 7.53: A Costas loop combines a PLL and an IQ demodulation function.

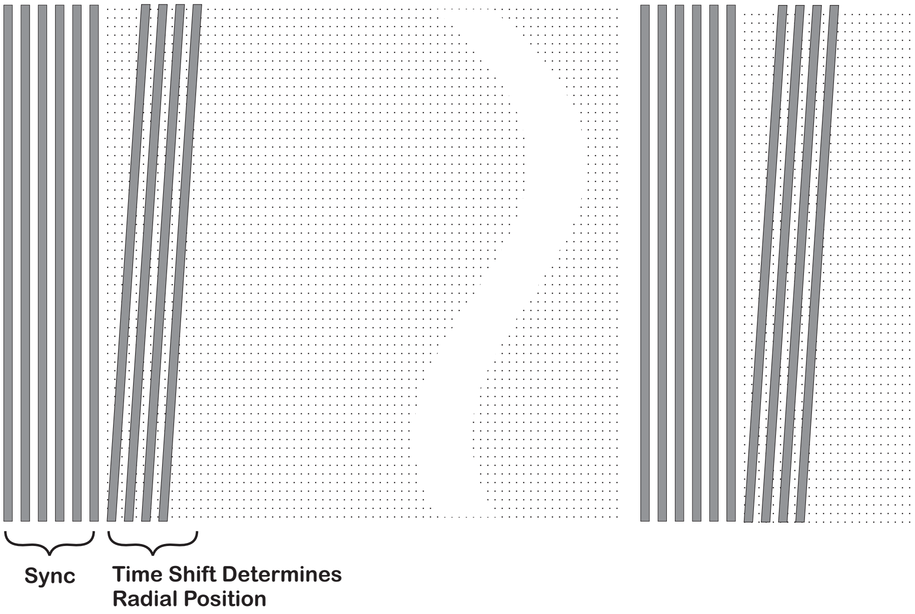


Figure 7.54: PLLs allow phase modulation of radial position on hard disk drives.

## 7.19 Precision Integration Lock-In

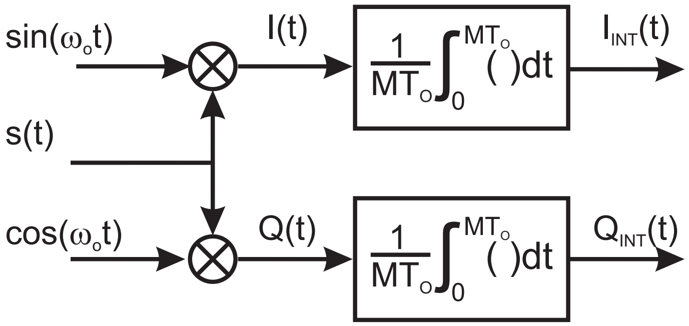


Figure 7.55: Lock-in amplifier with precision integration over an integer number of periods.

The lock-in amplifier methods described in Section 7.17 had the disadvantage – from a controls perspective – of having a fairly long delay, because their use models were not affected by delay. However, as we will see in the examples that follow this section, many of the uses of demodulation in control systems require a minimization of that delay.

One of the sources of delay is the long time constant in the LPF needed to minimize the effects of the higher harmonics produced in the mixing operation. In a pure circuit, this would only be the $2\omega_0$ harmonic, but as many circuits have small nonlinearities in practice, they had produce a wide set of higher harmonics that are hard to predict in advance. Returning to Figure 7.40, we can replace the generic LPF with a precision integral of Figure 7.55.

A given output signal, s(t), can be demodulated using a stepped-sine demodulator. Referring back to Figure 3 we can use Fourier series to decompose the signal, s(t), as

$$s(t) = A_0 + \sum_{k=1}^{\infty} \left(A_k \sin(k\omega_0 t) + B_k \cos(k\omega_0 t)\right). \tag{7.87}$$

We can expect that if the stimulus signal is single sinusoid, then $s(t)$ will have a strong first Fourier component:

$$s(t) \approx A_1 \sin(\omega_0 t) + B_1 \cos(\omega_0 t) + n(t) = C_1 \sin(\omega_0 t + \phi_1) + n(t), \tag{7.88}$$

where

$$C_1 = \sqrt{A_1^2 + B_1^2} \quad \text{and} \quad \phi_1 = arctan\frac{A_1}{B_1}. \tag{7.89}$$

The signals to be integrated, $I(t)$ for the in-phase and $Q(t)$ for the quadrature signal are

$$I(t) \quad = \quad s(t)\sin(\omega_0 t) \text{ and} \tag{7.90}$$

$$Q(t) \quad = \quad s(t)\cos(\omega_0 t). \tag{7.91}$$

Here $n(t)$ is the noise in $s(t)$.

Mixing with in-phase and quadrature signals as shown in Figure 7.55 yields

$$\begin{aligned} \int I(t)dt &= \int s(t)\sin(\omega_0 t)dt \\ &\approx \int A\sin(\omega_0 t+\theta)\sin(\omega_0 t)dt + \int n(t)\sin(\omega_0 t)dt \end{aligned} \tag{7.92}$$

and

$$\begin{aligned} \int Q(t)dt &= \int s(t)\cos(\omega_0 t)dt \\ &\approx \int A\cos(\omega_0 t+\theta)\cos(\omega_0 t)dt + \int n(t)\cos(\omega_0 t)dt. \end{aligned} \tag{7.93}$$

Here $n(t)$ is the noise in $s(t)$. Coherent demodulation (a.k.a. lock-in amplification) is based on the idea that if one sets the mixing signal to the same fundamental period as the drive signal, $T_0 = \frac{1}{f_0} = \frac{2\pi}{\omega_0}$, and integrates, then most of the terms drop out, leaving only a signals at baseband and at $2f_0$. The higher frequency signal can be removed with a post-integration notch or low-pass filter. Often, this low pass filter effect is achieved just by integrating over a large number of periods. With analog circuits, the difficulty in precisely knowing the fundamental frequency, $f_0$, means that it is difficult to place an analog notch at the exact $2f_0$ frequency (or those of any other harmonics). For this reason, the use of analog Lock-In-Amplifiers for coherent demodulation is usually accompanied by a broad low pass filter. The negative phase effects of using such a filter limits the closed-loop bandwidth of any system using such a demodulator.

Ideally, we will want to integrate over an integer, M, number of periods of the frequency that we wish to demodulate. Making the integrals definite and using well known trigonometric identities, yields:

$$\begin{aligned} \frac{1}{MT_0}\int_0^{MT_0} I(t)dt &= \frac{A}{2}\left(\cos\theta\frac{1}{MT_0}\int_0^{MT_0} dt - \frac{1}{MT_0}\int_0^{MT_0}\cos(2\omega_0 t+\theta)dt \right. \\ &\quad \left. + \frac{1}{MT_0}\int_0^{MT_0} n(t)\sin(\omega_0 t+\theta)dt\right) \text{ and} \end{aligned} \tag{7.94}$$

$$\begin{aligned} \frac{1}{MT_0}\int_0^{MT_0} Q(t)dt &= \frac{A}{2}\left(\sin\theta\frac{1}{MT_0}\int_0^{MT_0} dt - \frac{1}{MT_0}\int_0^{MT_0}\sin(2\omega_0 t+\theta)dt \right. \\ &\quad \left. + \frac{1}{MT_0}\int_0^{MT_0} n(t)\cos(\omega_0 t+\theta)dt\right). \end{aligned} \tag{7.95}$$

Equations 3.218 and 3.219 both have the properties that the second term on the right hand side goes to 0 for all positive $M$. The third term goes to 0 for increasing $MT_0$ as long as $n(t)$ is uncorrelated with the mixing sinusoids.

Such precise control of the integration period is difficult in an analog circuit but straightforward in a digital operation. As $MT_0$ gets large the contribution of $n(t)$ goes to 0, yielding the familiar relationships

$$I_{int} = \frac{1}{MT_0}\int_0^{MT_0} I(t)dt \approx \frac{A}{2}\cos(\theta) \tag{7.96}$$

and

$$Q_{int} = \frac{1}{MT_0}\int_0^{MT_0} Q(t)dt \approx \frac{A}{2}\sin(\theta). \tag{7.97}$$

### 7.19.1 Discrete Approximation of the Integral

There are several issues with standard methods of demodulation. The first is that imperfections in the integration approximation and noise in the signal require that $MT_0$ be large, relative to the period of the frequency at which demodulation is to take place, $T_0$, so $M$ must be large.

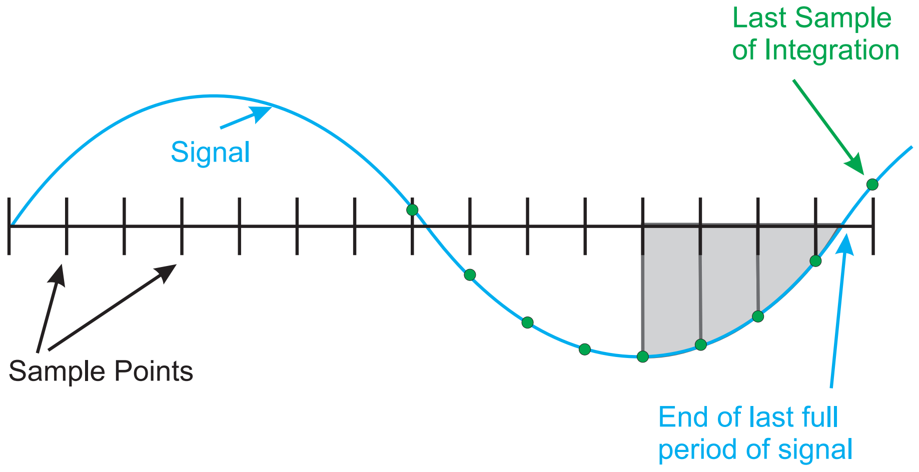


Figure 7.56: Integrating the partial sample of a sampled sinusoid.

The second is that with a digital controller, we have to be careful if we want to honor our desire to integrate over an integer number of periods of oscillation. We want

$$NT_S = MT_0, \tag{7.98}$$

where $N$ are are the number of samples in the integration, $T_S$ is the sample period, $M$ is the number of periods of oscillation, and $T_0$ is the period of oscillation. As illustrated in Figure 7.57 the data sample rate is rarely an integral multiple of the oscillation frequency, so it is difficult to make Equation 7.98

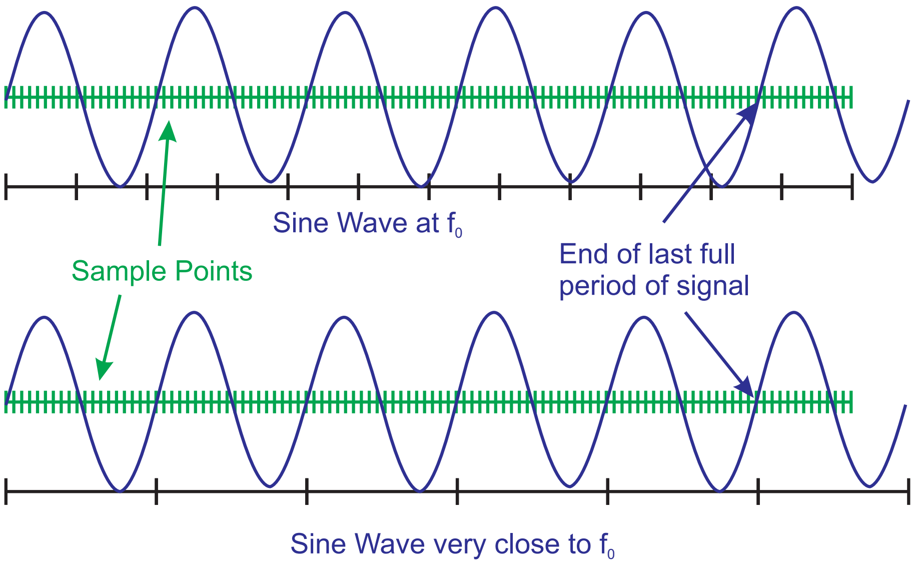

Sine Wave at $f_0$
Sample Points
End of last full period of signal
Sine Wave very close to $f_0$

Figure 7.57: The top drawing shows a sine wave which doesn't end up on an integer number of sample points. Adjusting $f_0$ slightly allows an integer number of periods to line up with an integer number of samples, as shown in the bottom drawing.

hold. Most digital systems are run at a fixed sample rate, $f_S = \frac{1}{T_S}$. The oscillation frequency, $f_0 = \frac{1}{T_0}$, comes from the frequencies at which we want to measure the FRF. That means $f_0$ will vary but $f_S$ will not. A solution is that for any desired $f_0$ and $M$, we can pick $N$ such that:

$$NT_S \leq MT_0 = N_{Real}T_S \leq (N+1)T_S. \tag{7.99}$$

We then round $N_{Real}$ to the nearest integer. We don't want to change $T_S$ or $M$, so we have two options. There is a fundamental difference between these methods when we have a fixed fundamental frequency, $f_0$ and when we can adjust it. In the most common case when equality does not hold, the last period of the integration is a partial one, as shown in Figure 7.56. This will require $N+1$ samples where the first $N$ samples of the integral integrate over the complete sample period and the last one is interpolated over a partial sample. If we are trying to precisely match a frequency, such as the resonant frequency of an AFM cantilever [85, 86], then we round $N_{Real}$ down to the closest integer below and then integrate over a partial fraction of an interval (Figure 7.56). The length of the partial fraction of an interval changes with every sample period and oscillation frequency, but would be fixed during any one measurement.

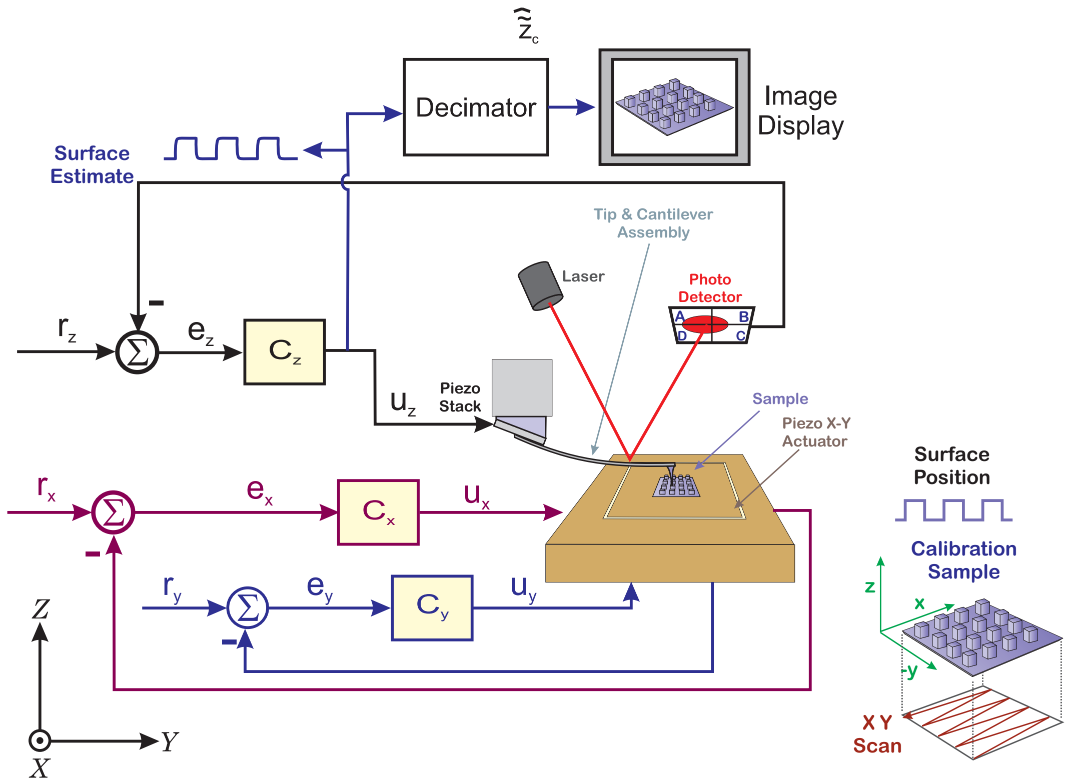


Figure 7.58: Schematic diagram of AFM control loop. Note that the loops driving the relative position of the sample and tip (put in the X-Y plane here) are often separate and slower than the loop controlling the interaction of the tip with the surface topology (put in the Z axis.

### 7.19.2 Coherent Demodulation for AFMs

One example of a practical implementation of these integrals was described in the author's earlier work on a low latency, high fidelity demodulation for atomic force microscopes (AFMs) [85, 86, 238].

These demodulation methods allow the system to extract signal information in as little as one cycle of the fundamental oscillation frequency. By having so little latency, the demodulator minimizes the time delay in the servo loop for an AC mode AFM. This in turn minimizes the negative phase effects of the demodulation allowing for higher speed scanning. There are two fundamental stages to this: mixing and integrating the signal efficiently [85] and extracting the magnitude and phase from those quantities [86]. To be useful, both should be done in real time. For the former, we will discuss the tradeoff between the accuracy of the integration and the relative sample frequency relative to that of the fundamental oscillation frequency. In the latter, we will discuss tradeoffs between the popular CORDIC method, optimized table lookup operations, and a phase-locked loop (PLL) based method. The latter two show

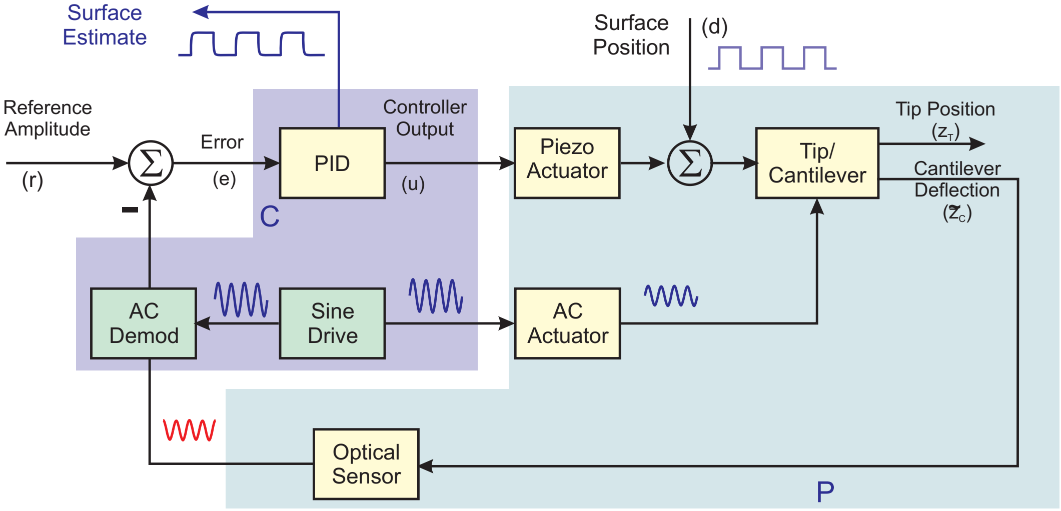


Figure 7.59: An AFM Control Block Diagram in dynamic mode. The digital controller generates a sinusoid and uses it to drive the AC actuator and feed the demodulation of the return signal.

a significant decrease in latency, with the PLL based method generating the magnitude and phase with virtually no extra latency and a significant savings in resources.

Control of Atomic Force Microscopes (AFMs) is depicted pictorially in Figure 7.58. This is often seen as three independent loops. The X-Y positioning of the sample relative tot he tip has a reference signal which makes this amenable to a large number of combined feedforward/feedback control schemes [239, 240, 241, 242, 243, 113, 244, 245, 246, 247, 248, 249, 250]. The Z-ais loop which controls the primary interaction of the cantilever and tip with the sample is only provided with a deflection signal in the case of contact mode measurement or a tip oscillation signal in the case of AC mode. Furthermore, the precision needed from the tip control loop often mandates significantly higher closed-loop bandwidth than with the X-Y control loops [27] (more references).

Dynamic mode AFM, which involves an oscillation of the cantilever in the proximity of the surface at a frequency close to the resonant frequency of the cantilever is depicted in Figure 7.59. In non-contact mode, the amplitude of the oscillation is slightly less than the nominal tip/surface distance so that while there is interaction between the tip and surface, this never enters into what would be considered contact. In the most common form of dynamic mode, also known as AC mode, or intermittent contact mode [216, 217], the amplitude of the free oscillation is slightly larger than the nominal tip/surface distance. When the tip comes into proximity with the surface, the oscillation amplitude, phase, and

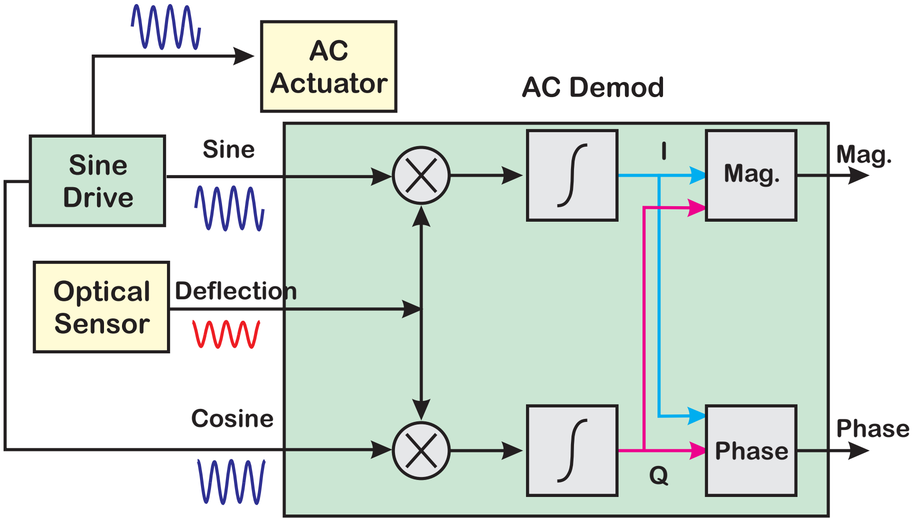


Figure 7.60: Coherent demodulation for AFM.

frequency are modulated, as shown (for amplitude) in Figure 7.61. By detecting this modulation and closing a feedback loop on the amplitude of the oscillation, this amplitude can be maintained at a constant level (modulo the closed-loop bandwidth of the z-axis loop [27]). Typically, the control signal represents the surface topography.

Dynamic mode imaging is done using cantilevers of various frequency ranges which are described in [27]. Often as the cantilever resonant frequency goes up, they get stiffer and have a higher quality factor ($Q$) [251]. The higher $Q$ provides greater amplitude amplification of the drive signal and better frequency discrimination for small shifts due to surface interaction. However, the extra stiffness of the cantilever might damage some materials, so there is a trade-off to be made on increasing the cantilever resonance. Because dynamic mode produces lower sheer forces on the sample than contact mode, the imaging of biological samples, is often done using this technique.

Although dynamic mode operation of AFMs is favored for imaging of soft samples, this operation is hampered by its slow speed. There are several reasons for this, as described in [27]:

- The $Q$ factor of the cantilever affects the time response. The cantilever is usually oscillated near its resonant frequency to get reasonable deflection amplitudes with low levels of input signal. Due to nonlinear interactions with the surface, the tip oscillation amplitude responds almost instantaneously to a step up in the surface. (see the left side of Figure 7.61). However, when there is a step down in the surface height, the response time of the cantilever oscillation will be proportional to $Q/\omega_o$, where $\omega_o$ is its resonant frequency (see the right sides of Figure 7.61 and 7.62) [252]. (The effect is exaggerated due to space considerations.) The flywheel action (which the author likes to call the Wiley E. Coyote effect), also introduces a limitation on the imaging speed without imaging artifacts. In general, imaging speed is limited by the slower rate.

- In AC mode, information about the surface is only available during the contact interval, which happens once every period of the oscillation. Consequently, the duty cycle of tip/surface interaction is considerably reduced as compared with contact mode. To have statistical significance, it is typical to average over multiple contact points, typically on the order of 10. Thus, the time constant of the vertical control loop is limited by the frequency of oscillation and the number of periods required.
- The surface information of interest is typically at a frequency well below that of the oscillation frequency and must be extracted from the oscillatory return signal via demodulation.

It is the role of the demodulator to extract surface information from the return signal. Typically there is a trade-off between the fidelity of the extracted information and shortening the demodulation time. Speeding up dynamic mode operation depends upon having a high fidelity, low latency demodulator. Consider driving the cantilever with a sine wave:

$$d(t) = D_0 \sin(\omega_0 t). \tag{7.100}$$

The signal, $s(t)$, from the optical sensor, is composed of harmonics of $\sin(\omega_0 t)$, i.e.

$$s(t) = A_0 + \sum_{k=1}^{\infty} \left( A_k \sin(k\omega_0 t) + B_k \cos(k\omega_0 t) \right). \tag{7.101}$$

We can expect that if the drive signal is large enough and the tip/surface interaction is set to be a small fraction of the free space oscillation, then the majority of the signal will be dominated by the first harmonic,

$$s(t) \approx A_1 \sin(\omega_0 t) + B_1 \cos(\omega_0 t) = C_1 \sin(\omega_0 t + \phi_1). \tag{7.102}$$

where

$$C_1 = \sqrt{A_1^2 + B_1^2} \text{ and } \phi_1 = \arctan\frac{A_1}{B_1}. \tag{7.103}$$

Because dynamic mode typically operates near the cantilever resonance [253], there is a relationship between the amplitude shift, phase shift, and frequency shift seen due to the surface/tip interaction. Thus, both the imaging and the Z-axis servo loop can be driven by one of several demodulated signals.

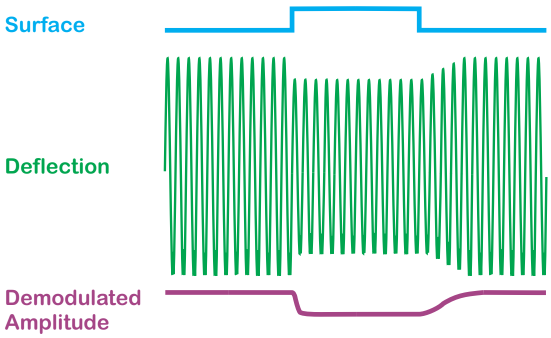


Figure 7.61: Open-loop deflection of the AFM tip in dynamic (AC) mode. Interaction with the surface will generally affect the amplitude and phase of the measured cantilever oscillation. The demodulated amplitude is shown here.

- **Amplitude Modulation (AM):** In this mode, the change in the amplitude of the return signal oscillation is detected and used as the error signal for the feedback loop. The speed of AM-AFM is often limited by the high $Q$-factor of the cantilever. , which slows the detection of surface features through the Wile E. Coyote effect seen in Figure 7.61, in which a the tip goes off a cliff on the surface but doesn't detect it for a while [27].
- **Phase Modulation (PM):** In this mode, the change in the difference between the reference phase of the cantilever drive oscillation and the phase of the returned deflection signal is detected. While feedback on the amplitude is easier to implement, the phase signal can be used to measure other surface properties like energy dissipation [254].
- **Frequency Modulation (FM):** In this mode, the change in the oscillation frequency of the returned deflection signal is detected. FM-AFM typically requires extremely high-$Q$ cantilevers so that the frequency shift can be detected. This has meant that FM-AFM is most often done in a vacuum where the lack of air damping makes the cantilever $Q$ seem much larger. However, non-vacuum operation has been made possible by recent improvements in instrumentation Reference.

Several standard forms of demodulation have been used for AFMs. Non-coherent, or non-synchronous demodulation, using an analog RMS-to-DC circuit [224](or its digital equivalent) can extract the signal magnitude, but not the phase. Both magnitude and phase can be extracted using a lock-in amplifier, which is a synchronous device that mixes in-phase and quadrature signals with the input signal and then integrates for a known period of time. Typical external lock-in amplifiers are slow, since tracking speed is often secondary to accuracy. Like the RMS-to-DC circuit, they often integrate over at least 10 periods of the input signal. For example, the 36 ms settling time of the AD736 [224] is 3,168 periods of the 88 kHz signal used in the examples of Sections 7.19.5 and 7.20. A 36 ms settling time sets the Nyquist frequency at 0.5*(1/36e-3) = 13.89 Hz. From this a reasonable closed-loop bandwidth limit would be

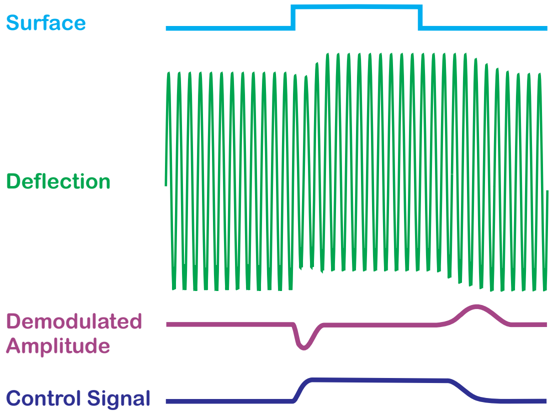


Figure 7.62: Deflection of the AFM tip in dynamic (AC) mode under feedback control. In this AM mode, the drop in oscillation amplitude results in the feedback loop raising the position of the actuator, which restores the oscillation amplitude. A rise in oscillation amplitude results in the controller lowering the position of the actuator. The control signal can then be used as a representation of the surface.

1/10 of the Nyquist frequency or about 1.4 Hz, which severely limits scanning speed. There has been a push among AFM manufacturers to include digital lock-in amplifiers in their AFM controllers. To be useful for high speed control, however, these lock-in amplifiers need to both have low latency and high fidelity. This is demonstrated in this paper.

Once the demodulation has been accomplished [85] producing in-phase and quadrature signals, these must be converted to magnitude and phase. This is essentially a transformation from rectangular to polar coordinates which seems simple until one has to compute it in real time. The second half of this paper will describe two methods for efficiently computing the magnitude and phase of those integrated signals. The first implements a minimal latency table lookup with automatic scaling of signals. The second method involves using a phase-locked loop (PLL) to align the mixing signal with the average phase of the return signal. In this case, the magnitude and phase calculations become trivial [86].

Minimal latency demodulation for AFMs presents unique challenges. Lock-in-amplifiers (LIAs) and coherent demodulation have typically been used in a variety of communication and measurement systems [75, 53, 70, 84, 87] (more communication, DSA references). The difference is that those systems did not close the feedback loop and thus were not affected by latency. In prior demodulation schemes for disk drive feedback systems, the author was able to use the fact that the timing of the magnetic servo patterns to be demodulated had already been recovered, allowing for amplitude demodulation with no need to extract magnitude from in-phase and quadrature integrals [213, 211, 212] In the AFM example, latency is one of the key limiting factors in overall loop bandwidth [19], and so the desire to limit latency means that our LIA has to minimize integration time. Furthermore, many LIAs require post-integration low pass filtering to minimize the effects of harmonics [225, 226]. However, this paper will also show two post-integration filtering methods that remove the effects of harmonics without significantly affecting the achievable signal bandwidth.

Note that since the conference version of this material was first published [85, 86], other papers have delved into similar areas, including [225], which essentially creates a high speed analog LIA, as opposed to the high speed digital LIA of this work. The results in this paper show several advantages of the digital technique, and when done properly, this digital LIA has significant noise immunity, in contrast to the limited description in [226].

### 7.19.3 Practical Implementation of the Discrete Integration

In those papers, a trapezoidal rule integration is used. This seems to provide a reasonable compromise between minimizing latency and integration accuracy. Consider the trapezoidal rule implementation of our integral:

$$\int_{t_0}^{t_N} y(t)dt \approx \sum_{k=0}^{N-1} \left(\frac{y_{k+1}+y_k}{2}\right) T_S, \tag{7.104}$$

where $T_S$ and $N$ are defined as in Equation 7.99. Between $N$ and $N+1$, we will have a partial interval integral that must be computed

$$\int_{t_N}^{t_k+T_S h} y(t)dt \approx \left(\frac{y_{N+1}+y_N}{2}\right) hT_S, \tag{7.105}$$

where $0 \leq h \leq 1$ and

$$h = \frac{MT_0 - NT_S}{T_S}. \tag{7.106}$$

Note that $hT_S$ is the integration time needed to complete the $M^{th}$ period of oscillations at $f_0$, so the fraction of a sample period that this represents is given by $h$. Putting these together and looking back in time rather than forward, we get

$$\int_{kT_S - MT_0}^{kT_S} y(t)dt \approx S_k \text{ where} \tag{7.107}$$

$$\frac{S_k}{T_S} = \sum_{j=0}^{N-1} \left(\frac{y_{k-j}+y_{k-(j+1)}}{2}\right) + \left(\frac{y_{k-N}+y_{k-(N+1)}}{2}\right) h. \tag{7.108}$$

$$\frac{S_k}{T_S} = \frac{y_k}{2} + \sum_{j=0}^{N-1} y_{k-j} + \frac{y_{k-N}}{2} + h\left(\frac{y_{k-N}+y_{k-(N+1)}}{2}\right). \tag{7.109}$$

Equation 7.109 is very instructive because it shows us that the integral can be simply constructed as a FIR filter. We can factor out a single scale factor, $T_S$, and then we have a main integral corresponding to the terms before the term scaled by $h$ and the fractional portion, scaled by $h$. It is also instructive that very little about this formula is dependent upon the sample interval, $f_S$, and the oscillation frequency, $f_0$. Basically, a change in $f_0$, $f_S$, and/or the number of periods in the integral, $M$, changes only $N$ and $h$. For a given $M$, $T_0$, and $T_S$, we pick $N$ from Equation 7.99 and $h$ from Equation 7.106.

It is worth noting the computational and practical importance of this result. A high speed, coherent demodulation problem has been turned into a simple "multiply and integrate" operation, and the integration has been turned into a simple finite impulse response (FIR) filter. Furthermore, most of the FIR coefficients are $1$, two of them are $1/2$ (easily accomplished via a right shift of the signal value by one bit for fixed point notation), leaving only one non-trivial multiplication. That multiplication has a coefficient that is less than $1$ ($h/2$), which is pre-computed a single time whenever the oscillation period ($T_0$) is adjusted. Even when $T_0$ changes, the only parameters of the integral that change are $N$ and $h$.

We see that this integration problem can be put in the general form of an FIR filter, providing we can handle the bookkeeping for the computation. At each time step, new data comes into one side of the FIR and old data is discarded from the other side. Actually doing this in real-time hardware presents three issues:

1) First, shifting all the values of $y(j)$ back one step in time can be expensive in terms of computation time. At each time step, $N$ values have to be shifted in their memory locations so that they line up with the coefficients of the filter at the next time step. For long values of $N$, this can be quite a chore. More importantly, as the value of $N$ changes, say due to the user selecting a new oscillation frequency or a new number of periods over which to integrate. Thus, the delay in the computation due to the bookkeeping changes with different frequencies. How this is handled depends upon the hardware architecture chosen for the task. For example, in a processor with a CPU and memory (such as a microprocessor or a Digital Signal Processor (DSP)), circular addressing can be used to change the starting point of the $\phi$ vector at each step. Thus, only one value needs to be updated in $\phi$ and the coefficient vector does not move.

2) Second, the longer the filter, the longer the number of computations. Varying latency is a problem for feedback systems and must be avoided. Simply computing the output of the filter as a new sample comes in means that there is a $N$ dependent delay in the time between the filter input and output. This particular issue can be handled using precalculation [15]. That is, all the computations that do not depend upon the current sample information are computed before the current sample arrives. When the current sample arrives, a last

computation is done and the filter output is generated. Not only does the precalculation method have less latency than methods without precalculation, but that latency does not vary with increasing $N$.

3) Third, any new frequency requires loading a new set of $N+1$ coefficients. How this is done changes dramatically, depending upon the hardware implementation of the algorithm. For example in a DSP, the coefficients can simply be loaded into RAM and the loop limits can be changed. However, in a FPGA implementation, this becomes much more difficult. To take advantage of the parallel computation nature of the FPGA, we have to avoid putting such operations through a serial process as much as possible.

In reference to the point (3) above, Equation 7.109 shows that the only one of the $N+1$ coefficients that changes is $h$. To address points (1) and (2),it is worth looking at an incremental or iterative method, that is, a method that uses a running sum for the integral and only makes adjustments to this running sum at each time step.

Advancing Equation 7.109, forward one step in time, and taking the difference with the current value yields

$$S_k = \frac{T_S}{2}\left[y_k + 2\sum_{j=1}^{N-1} y_{k-j} + y_{k-N} + h\left(y_{k+1} + y_k\right)\right], \text{ so} \tag{7.110}$$

$$S_{k+1} = \frac{T_S}{2}\left[y_{k+1} + 2\sum_{j=1}^{N-1} y_{k+1-j} + y_{k+1-N} + h\left(y_{k+2} + y_{k+1}\right)\right]. \tag{7.111}$$

Now,

$$\begin{aligned} S_{k+1} - S_k &= \frac{T_S}{2}\left[y_{k+1} - y_k + 2\sum_{j=1}^{N-1} y_{k+1-j} - 2\sum_{j=1}^{N-1} y_{k-j}\right. \\ &+ \left. y_{k-(N-1)} - y_{k-N} + h\left(y_{k-N} - y_{k-(N+1)}\right)\right] \end{aligned} \tag{7.112}$$

$$\begin{aligned} &= \frac{T_S}{2}\left[y_{k+1} - y_k + 2y_k - 2y_{k-(N-1)} + 2\sum_{j=2}^{N-1} y_{k+1-j}\right. \\ &\quad \left. -2\sum_{j=1}^{N-2} y_{k-j} + y_{k+1-N} - y_{k-N} + h\left(y_{k+2} - y_k\right)\right]. \end{aligned} \tag{7.113}$$

which reduces to

$$\Delta S_{k+1} = S_{k+1} - S_k = \frac{T_S}{2}\left[y_{k+1} + y_k + y_{k-(N-1)} - y_{k-N} + h\left(y_{k-N} - y_{k-(N+1)}\right)\right]. \qquad (7.114)$$

This is a wonderful result, because Equation 7.114 has a form that uses a small, fixed number of terms. Even if higher-order integration methods are used, this form would have a larger, but fixed number of terms, independent of $N$. We need to keep track of old values of the integral, but they are used sparingly in the calculation. If we start with $S_0 = 0$, then we can compute $S_k$ from

$$S_k = \Delta S_k + S_{k-1}. \qquad (7.115)$$

However, this incremental form is dependent upon having a precise calculation of $h$. Any errors in $h$ can accumulate resulting in a divergence between the true integral value and the calculated value. Setting $\hat{h}$ as our fixed point estimate of $h$,

$$\begin{aligned} \Delta S_{k+1} &= \frac{T_S}{2}\left[y_{k+1} + y_k + y_{k+1-N} - y_{k-N} + \hat{h}\left(y_{k+2} - y_k\right)\right] + \frac{T_S}{2}\left[\left(h - \hat{h}\right)\left(y_{k+2} - y_k\right)\right], && (7.116) \\ &= \hat{\Delta} S_{k+1} + \tilde{\Delta} S_{k+1}, && (7.117) \end{aligned}$$

where $\hat{\Delta} S_{k+1}$ is our estimate of $\Delta S_{k+1}$ based upon $\hat{h}$ and $\tilde{\Delta} S_{k+1}$ is the error introduced by the mismatch, $h - \hat{h}$. That is,

$$\tilde{\Delta} S_{k+1} = \frac{T_S}{2}\left[\left(h - \hat{h}\right)\left(y_{k+2} - y_k\right)\right]. \qquad (7.118)$$

Finally, the difference between the true sum, based on the true value of $h$, and the estimated sum, based on the estimate $\hat{h}$, will be

$$S_k - \hat{S}_k = \sum_{j=1}^{k} \tilde{\Delta} S_k, \qquad (7.119)$$

which will grow linearly (and without bound) with each time step, $k$.

Looking at Equation 7.110, we see that we can break up the sum for $S_k$ into

$$\begin{aligned} S_k &= \frac{T_S}{2}\left[I_k + h\left(y_{k+1} + y_k\right)\right], \text{ where} && (7.120) \\ I_k &= y_k + 2\sum_{j=1}^{N-1} y_{k-j} + y_{k-N} && (7.121) \end{aligned}$$

$I_k$ is based entirely on input and output samples that have not been multiplied by anything number that cannot be represented exactly in a digital system. That is, $1$ and $2$ are easy to represent exactly in any digital processor. Because of this, we can make the incremental portion of our integral based on a difference of $I_k$s.

$$
\begin{aligned}
I_{k+1} &= y_{k+1} + 2\sum_{j=1}^{N-1} y_{k+1-j} + y_{k+1-N}, \text{ so} \tag{7.122}\\
\Delta I_{k+1} &= I_{k+1} - I_k \tag{7.123}\\
&= y_{k+1} - y_k + 2\sum_{j=1}^{N-1} y_{k+1-j} - 2\sum_{j=1}^{N-1} y_{k-j} + y_{k+1-N} - y_{k-N}, \tag{7.124}\\
&= y_{k+1} - y_k + 2\sum_{j=1}^{N-1} y_{k-(j-1)} - 2\sum_{j=1}^{N-1} y_{k-j} + y_{k+1-N} - y_{k-N}, \tag{7.125}\\
&= y_{k+1} - y_k + 2\sum_{j=0}^{N-2} y_k - 2\sum_{j=1}^{N-1} y_{k-j} + y_{k+1-N} - y_{k-N}, \tag{7.126}\\
&= y_{k+1} - y_k + 2y_k + 2\sum_{j=1}^{N-2} y_k - 2\sum_{j=1}^{N-2} y_{k-j} - 2y_{k-(N-1)} + y_{k+1-N} - y_{k-N}, \tag{7.127}\\
&= y_{k+1} + y_k - (y_{k+1-N} + y_{k-N}). \tag{7.128}\\
& \tag{7.129}
\end{aligned}
$$

Now, we compute $I_k$ from $\Delta I_k$ by

$$I_k = \Delta I_k + I_{k-1}. \tag{7.130}$$

and $S_k$ is computed from Equation 7.120.

After a lot of algebra, we got to the remarkably simple incremental computation for the integral:

$$S_k = \frac{T_S}{2}\left[I_k + h\left(y_{k+1} + y_k\right)\right]. \tag{7.131}$$

This relationship was relatively straightforward to program into an FPGA, with the fractional portion of the integral removed from the iteration. Thus, the additions and subtractions from the incremental sum in Equation 7.128 are exact, preventing the possibility of small errors in $h$ accumulating in the recursion. Some will recognize that this form is essentially the same form as a Cascaded Integrator-Comb (CIC) Filter [164].

We can see that the recursion for computing $I_k$ from $\Delta_{I,k}$ involves no scaled values of the previous inputs. So, if we only keep old values of $y_k$ in memory, we can compute $I_k$ exactly, and any errors due to mismatch in $\hat{h}$ from $h$ are limited.

We still need to keep track of a potentially large set of prior sample values. If we were to move each sample back one time step in memory, this would create large amounts of bookkeeping computations.

Instead, we can use the same circular addressing methods mentioned above for the microprocessor/DSP case. A memory buffer is used as a storage mechanism for old sample values. New data is written into the memory and old data is read out of it. The memory addresses at which this happens are computed using circular addressing, with the indices of these addresses moving at each time step. Thus, the update of the memory buffer at any one time step requires reading the oldest value in the filter from memory and writing the newest value into memory.

In short, we have implemented a quadrature integration as an FIR, and compute the values of that FIR recursively as if it were an IIR. What keeps this an FIR is that at a later time we subtract off exactly the same value that was added to the cumulative sum. We still need to keep track of a potentially large set of prior sample values. If we were to move each sample back one time step in memory, this would create large amounts of bookkeeping computations. Instead, we can use circular addressing of a memory buffer of old sample values as is often done in DSP calculations. New data is written into the memory and old data is read out of it. The memory addresses at which this happens are computed using circular addressing, with the indices of these addresses moving at each time step. Thus, the update of the memory buffer at any one time step requires reading the oldest value in the filter from memory and writing the newest value into memory. This can be done as easily in a Field Programmable Gate Array (FPGA) as in a processor and is illustrated in Figure 7.63.

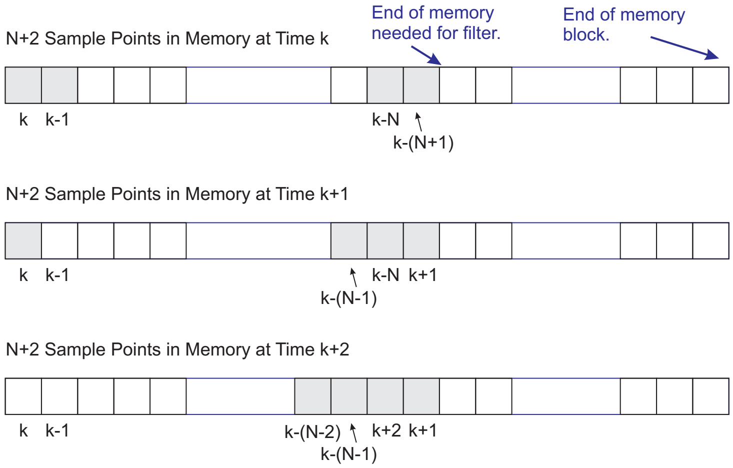

N+2 Sample Points in Memory at Time k
End of memory needed for filter.
End of memory block.
k
k-1
k-N
k-(N+1)
N+2 Sample Points in Memory at Time k+1
k
k-1
k-(N-1)
k-N
k+1
N+2 Sample Points in Memory at Time k+2
k
k-1
k-(N-2)
k-(N-1)
k+2
k+1

Figure 7.63: Circular addressing of filter memory

In Figure 7.63 we see that we can use a single large memory block and put the filter in part of it. The use of a larger block allows the filter length to be quite flexible, so that it is easy for the user to change oscillation frequency or the number of periods of oscillation that are needed for the calculation. Changing these simply results in a change in the size of the memory used in the block. We see here that for the $N$ chosen as above, we only have $N+2$ words of memory allocated for the filter. At any time $k$, a new sample will be written at a memory location denoted by $k$, and an old sample will be read from memory at location $k-(N+1)$. When the next sample comes in it should go to the left of sample $k$, but since our drawing shows this at the beginning of the memory block, the writing index is chosen to be at the end of the block, where the no longer useful data from $k-(N+1)$ is held. Thus, the sample from $k+1$ is written there, while the index for reading decrements so as to point to location $k-N$. We can see that this process can go on easily, simply by initializing the indices in the right locations and then moving them in synchrony together. Again, the importance of this is that the most difficult portion of the coherent demodulation has been turned into a computationally trivial circular memory shift.

The partial interval integral is necessitated in some cases by needing to precisely match a predetermined oscillation frequency. This is often due to a physical parameter – such as the cantilever resonance in the AFM demodulator. Note that if the exact frequency match is not critical, as with a built in sine-dwell [87], then we can adjust $T_0$ so that $h=0$ in Equation 7.106. This simplifies the sum terms above.

$$\tilde{N}T_S = M\tilde{T}_0. \tag{7.132}$$

In that case, $M$ was generally assumed to be at least 8, as implemented in the HP 3562A Dynamic Signal Analyzer [53]. Furthermore, the measurement frequencies in that use were servo system frequencies, generally significantly lower than the frequency of a cantilever tip oscillation. Finally, the generation of an accurate frequency response function measurement did not hinge on maintaining a single frequency. This allowed the set of oscillation frequencies to be adjusted slightly, so that for each measurement frequency, $f_0 = \frac{1}{T_0}$, equality in Equation 7.132 held.

The adjustments to $T_0$ to make equality hold in Equation 7.132 can be kept small if $N$ and $M$ are made large. While this approach may be feasible for an off-line measurement described above, or for producing signal processing results that are not used in the feedback loop, this choice will add to latency in the integral calculation, so we are better off integrating over the fractional interval as described above.

Digital quadrature is documented in many numerical computation texts [80, 255]. Generally, the algorithms for quadrature will make use of a polynomial fit over some number of sample points to approximate the function. The fit of a $L^{\text{th}}$ order polynomial will involve $L+1$ points. In applications where latency (time delay) is not an issue, one can achieve higher accuracy by conducting the integral between samples $k$ and $k+1$ using samples on either side of this interval.

For example, the Hewlett-Packard 3562A computes the integral of its mixed sinusoids by using a fifth order polynomial fit over 6 points [53]. It uses 3 points on either side of the interval in question. As the interval of integration slides forward in time, points to the left and right of it are used to give a more accurate approximation of the function being integrated. Note that the interval over which the integral is done is delayed by two and a half sample intervals (compared with simply using only the latest sample point).

For use in generating the error signal for a feedback controller, we want to minimize the latency of the integral and for this the simplest discrete integral approximations are the forward and backward rectangular rule approximations, and the trapezoidal rule approximation. Because of their small amount of delay, these are often used in generating discrete equivalents of analog controllers [15]. The forward rectangular rule has a single period delay. The backwards rectangular rule has zero delay. Finally, the trapezoidal rule has a half sample period delay.

Standard practice in digital lock-in amplifiers is to use one of the rectangular rule approximations and rely on integrating over many periods of oscillation to drive the error to 0. However, by using a higher-order approximation and a partial sample integral, we can cut the error down with significantly fewer periods of integration.

### 7.19.4 Pre and Post Integration Filtering

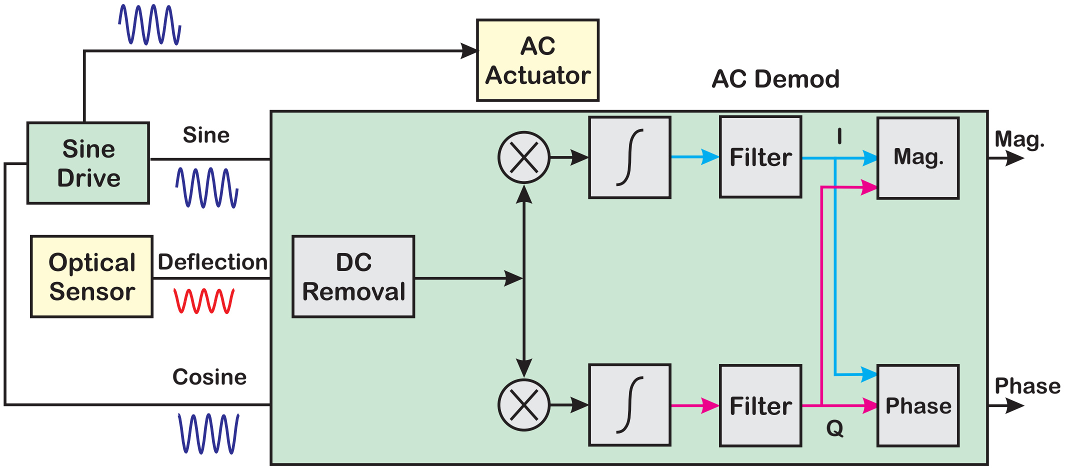


Figure 7.64: Coherent demodulation for AFM. DC removal and post integration filtering included.

If the integrals of Equations 7.142 and 7.143 were done in continuous time with infinite precision, then there would be need to filter DC components or harmonics of the input frequency from the integral. However, the sampling of the data means that the sinusoids are approximated by a stair step function, and the integration is approximated as described in Section 7.19.3 [85]. This means the rejection of

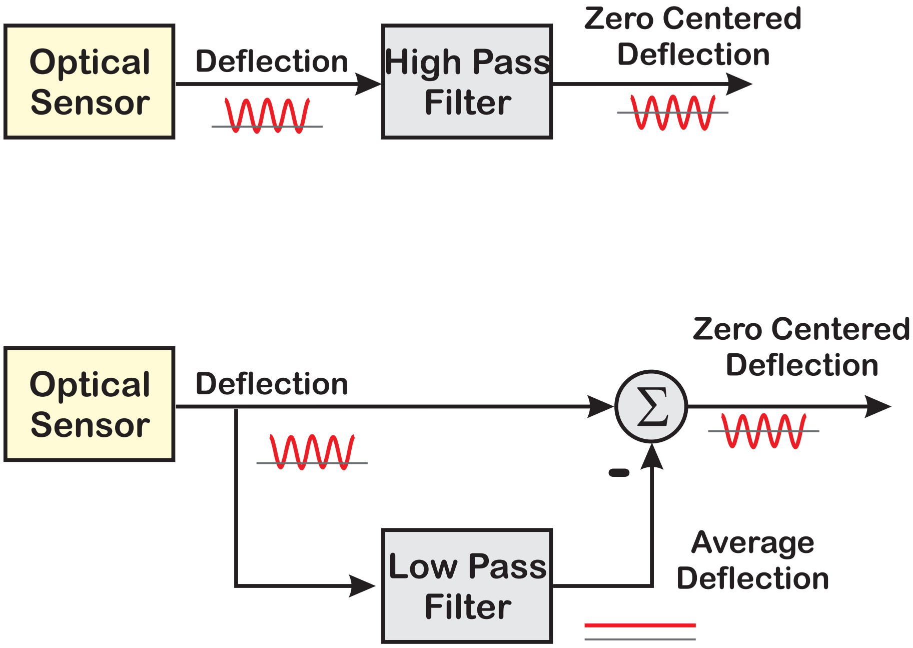


Figure 7.65: Different methods of removing DC value from return signal. While a high pass filter is conceptually simpler, it may incur more computational complexity.

DC offsets and higher harmonics is not complete. Several simple fixes, including DC removal and post integration filtering, significantly improve the behavior of the integration.

The DC removal can be accomplished using a simple high pass filter, and there are several different methods available for implementing this. For a discrete time ideal high pass filter, we would have

$$H(z) = \frac{\alpha(z-1)}{z-\alpha} = 1 - \frac{(1-\alpha)z}{z-\alpha} = 1 - L(z). \tag{7.133}$$

In other words, a high pass filter can be implemented by subtracting a low pass value from the original signal. While these are theoretically equivalent, there are some latency advantages to the second method in that the low pass value can be computed in the background (it changes slowly) and only the subtraction is in the "latency" path.

Post integration filtering helps remove artifacts of the integral approximation from the computed magnitude and phase. In particular, harmonics of the original drive frequency often show up, although at a greatly reduced level. Some authors make the case that this need to reduce higher harmonics requires a low pass filter that significantly limits the bandwidth of the demodulator and therefore the feedback loop [225, 226].

This illustrates one of the key advantages of an all digital demodulator built into the same processing chip as the oscillation signal generator. There is no guesswork or estimation about the drive signal since it can be routed to the demodulator as well as the cantilever driver. Furthermore, while digital quadrature must account for sampling issues, it is exactly repeatable, unlike multiplication, addition, integration, filtering, and therefore demodulation relying on an analog circuit implementation of the mathematics. In our case, the precise real time sinusoid generation and integration described here allows us to avoid using such an overly conservative filter. Since we have precise knowledge of the frequency, we can create a very narrow digital notch precisely at each harmonic we wish to attenuate.

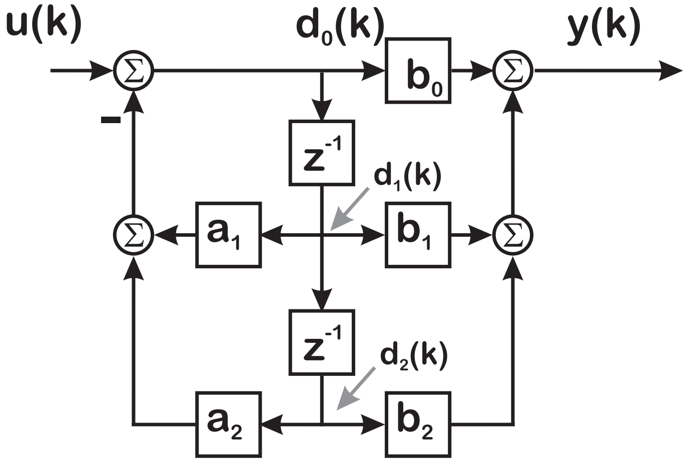


Figure 7.66: A second order polynomial filter in Direct Form II, known as a digital biquad filter.

A notch filter can be implemented simply using a single biquad digital filter as described in [54] and shown in Figure 7.66. Biquads are simple to program and relatively well behaved numerically.

$$N(s) = \frac{Y(s)}{U(s)} = \frac{\omega_d^2}{\omega_n^2}\left(\frac{s^2+2\zeta_n\omega_n s+\omega_n^2}{s^2+2\zeta_d\omega_d s+\omega_d^2}\right), \tag{7.134}$$

which can be discretized using the matched pole-zero mapping used in [54] and [33] to yield

$$N(z^{-1}) = \frac{Y(z)}{U(z)} = \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}. \tag{7.135}$$

This gets implemented in the time domain as:

$$\begin{aligned} y(k) &= -a_1y(k-1)-a_2y(k-2)+b_0u(k) \\ &\quad +b_1u(k-1)+b_2u(k-2) \end{aligned} \tag{7.136}$$

It turns out that it is easier to implement this using the delay format [169] which resembles a controller canonical form [171] in control or a direct form II IIR filter [167, 172, 173]:

$$d(k) = -a_1d(k-1)-a_2d(k-2)+u(k) \tag{7.137}$$

$$y(k) = b_0 d(k) + b_1 d(k-1) + b_2 d(k-2) \tag{7.138}$$

Biquads are nice because the growth in values can be limited by the short nature of the filter. Thus, finite word length problems are minimized as the sums from the numerator and denominator can balance each other out for a well designed filter [169].

While [54] describes minimal latency ways of cascading biquads, and [33] describes a method of improving the accuracy of the fixed point coefficients when the sample frequency is substantially larger than the dynamics to be filtered, neither of these should be necessary here. The main residual harmonic after the original integration appears at twice the mixing frequency, i.e. $2f_0$, and so a single notch can be generated by setting the numerator and denominator to have the same center frequency

$$\omega_n = 2\pi f_n = \omega_d = 2\pi f_d = 2\pi 2 f_0 = 2\omega_0, \tag{7.139}$$

and by setting

$$\zeta_n \ll \zeta_d. \tag{7.140}$$

The adjustments of [33] are not needed because the frequency of the notch, $2f_0$ is within two orders of magnitude of the sample frequency and therefore the jamming of filter poles and zeros near $z = 1$ does not occur.

The calculations in a single biquad, can be implemented with a handful of multiplies and additions (five of each), in fixed point arithmetic. Now, while the choice of the tip oscillation frequency, $f_0$ is determined by the physical characteristics of the particular cantilever, once this is set we know any harmonic, $Kf_0$, precisely because we are generating $f_0$ to drive the tip. This means that we can have great confidence in the accuracy of (7.139). For simplicity, this discussion will focus on the second harmonic, $2f_0$, but it is well understood that any harmonic can be filtered, up to the limitations placed on us by the Nyquist Rate. In principle, we could use a biquad cascade such as the multinotch [54], although it is unlikely that we need more than one or two biquads. Also, the fact that we are filtering harmonics of an AC drive frequency means that we are unlikely to encounter the sample rate issues which motivate $\Delta$ coefficients [33, 165].

In designing this filter, we should realize that the residual signal amplitude will be fairly small, so there is no need for extra precision in the calculation.

If we use (7.139), then at frequency $2f_0$, Equation 7.134 reduces to

$$N(j2\omega_0) = \frac{\zeta_n}{\zeta_d} = \frac{Q_d}{Q_n}, \tag{7.141}$$

where $Q = \frac{1}{2\zeta}$ is the well known resonance quality factor. Because we can be so confident in our knowledge of $2f_0$ we can set both $Q_d$ and $Q_n$ relatively high, limiting the effect of the notch filter to the frequency "vicinity" of $2f_0$. The phase effects of the notch filter will stay far away from $f_0$ and so have no real effect on the servo system. In the examples of Sections 7.19.5 and 7.20, $Q_d$ was set to $4$ and $Q_n$ was set to $40$, to produce an attenuation factor of $10$ $(20\text{ dB})$ at $f = 2f_0$. This is in contrast to the low pass filters alluded to in [225, 226], which are claimed to have a significant phase impact on the control system.

Alternately, an FIR filter similar to the demodulator integrator of Sections 7.19.1 and 7.19.3 can be used. The main difference in this FIR is that there is no mixing of the input signals. The output of the demodulator is fed into another FIR integrator, which integrates over a single period of the original wave. This is best understood as follows. For the original integral, it was necessary to mix the return signal from the tip with in-phase and quadrature signals at the same frequency in order to generate a baseband and a $2f_0$ signal. The integration over an integer number of periods of $f_0$ removed most but not all of the $2f_0$ component. Integrating the output signal of the integration averages both the baseband and the $2f_0$ components, leaving the baseband largely unchanged and nulling the not only the $2f_0$ component, but also any other residual higher harmonics of the integrator output signals. In the examples of Sections 7.19.5 and 7.20, an oscillation frequency of $f_0 = 88$ kHz was chosen with a sample frequency of $f_S = 1$ MHz, which meant that each oscillation was $11.\overline{36}$ sample periods long. This means that in Equation 7.99, if we choose $M = 1$ we get $N = 11$. From Equation 7.106, we get $h = \overline{0.36}$ which we need to represent in 2's complement arithmetic. The FIR averager has the same length as the $M = 1$ integration, but without the mixing of sinusoids.

This FIR averager lengthens the delay associated with the demodulator by at least half the period of the original oscillation (depending upon the length of the filter and the period of oscillation). A notch filter implemented as a digital biquad only removes a single harmonic at a time, but the added latency is fixed and minimal (typically a few clock periods). In Section 7.20, we see that combined with the DC removal, the notching of the $2f_0$ harmonic closely approximates the performance of the FIR.

For the amplitude modulation of the AFM cantilever, we are mostly concerned about the magnitude of the signal. In the case of a frequency-response function (FRF), we need both the magnitude and phase. However, while the integrals must be computed in real-time to keep up with the signals, these latter quantities can be computed off line on the results of the integrals, as they comprise a finite set of frequency results.

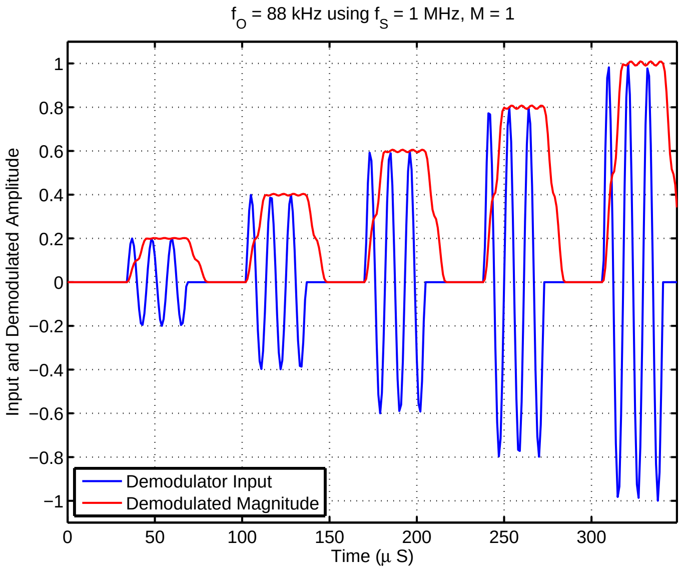


Figure 7.67: Matlab simulation demonstrating speed of convergence.

| Relevant Figures | Input Noise ($\sigma$) | Magnitude $\sigma$ | Phase $\sigma$ |
|---|---|---|---|
| Fig. 7.68 & 7.69 (no filt.) | 0 | 1.066493e-03 | 0.2428932 |
| Fig. 7.68 & 7.69 (FIR) | 0 | 3.410869e-06 | 7.756699e-04 |
| Fig. 7.70 & 7.71 (no filt.) | 0.01804220 | 1.172526e-03 | .3127864 |
| Fig. 7.70 & 7.71 (FIR) | 0.01804220 | 1.057135e-03 | .2950216 |

Table 7.2: Computed steady state noise values extracted from integrator simulations. The last 20% of the data was used to get steady state values. The input noise to the simulation was a uniform distribution.

### 7.19.5 AFM Demodulation Examples

Equations 7.114 and 7.115 are easy to implement simply in real-time processors such as DSPs or FPGAs. In this section are simulation examples to illustrate the behavior of the integration portion of the demodulator. Similar results are shown in Section 7.20 to illustrate the extraction of magnitude and phase from the integrated quantities.

An illustration of the convergence of the integrator is shown in the simple Matlab simulation of Figure 7.67. The input signal (blue curve) is at 88 kHz and is switched “on” and “off” with the magnitude stepped up with each “on” period. The demodulated magnitude (red curve) is plotted on the same axis as the input signal. As predicted by the analysis, the integrator yields I and Q in one integration period ($M = 1$), which is mapped to magnitude in the plot.

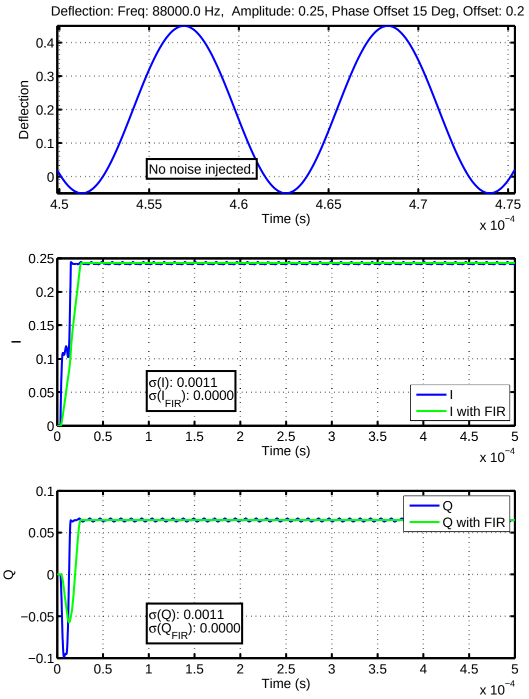


Figure 7.68: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is $0.25$. There is a normalized offset of 0.2 in the signal level, and the phase of the signal driving the deflection is $15^\circ$ ahead of that of the in-phase (sine) mixing signal at the beginning of the simulation.

The demodulator architecture, implemented in FPGA hardware, was simulated using ModelSim 6.6b [256], and signals were normalized back to real numbers. The signals of interest to this method were saved to an ASCII file, which was processed in Matlab. The last 20% of the data was used to compute the steady state averages ($\mu$) and standard deviations ($\sigma$) of these signals. The means for the $I$ and $Q$ signal do not mean much to the reader, but the small $\sigma$ for these signals does show the accuracy of the integrator. In the case of the magnitude and phase $\mu$s and $\sigma$s, the values can be compared to the original inputs. Note that the $\sigma$ value for phase is in degrees.

Figures 7.68 –7.71 show the results of the simulator when driven with an 88 kHz signal, which had an amplitude of $0.25$, an offset of $0.2$ and a phase advance (as compared to the in-phase mixing signal) of $15^\circ$ (in Figures 7.68 and 7.69) or a delay of 30° (in Figures 7.70 and 7.71). The top plots of Figures 7.68 and 7.70 are zoomed in to better show the effects of noise on the signal. The lower two plots show the results of the I and Q branch integrations. The in-phase (I) and quadrature (Q) branches converge quickly, although small imperfections in the integration result in some ripple in these signals. With no noise injected, the standard deviation ($\sigma$) is minuscule. The rejection of white noise while using a single integration period is small. The rejection can be increased by integrating over multiple oscillation

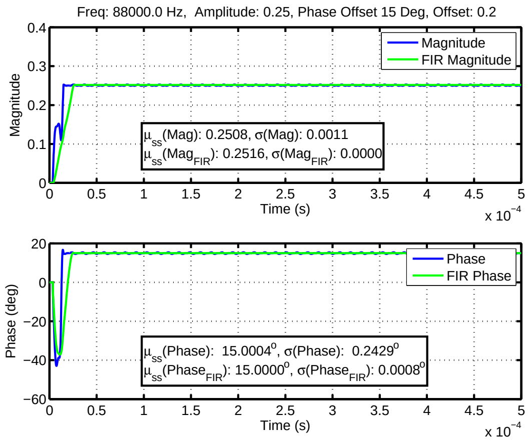


Figure 7.69: Floating point magnitude and phase extracted from the simulation in Figure 7.68.

periods, but at the cost of more latency.

While the simulation accurately simulates synthesizable blocks of the FPGA, it can also simulate blocks that cannot be put into logic. Thus, Figure 7.69 and 7.71 show real values of magnitude and phase extracted from the integrator outputs. Simulations that show magnitude and phase extracted using synthesizable blocks are shown in Section 7.20.

Note that with no noise injected, the deviation ($\sigma$) of the magnitude and phase from their steady state values ($\mu$) is extremely small (and virtually non-existent after the application of the FIR). The effect of signal noise, $n(t)$, is diminished despite the short integration time.

### 7.19.6 Magnitude and Phase Calculations

Prior sections have shown how to approximate the integrals of the in-phase and quadrature signals so as to obtain

$$I_{sum} \approx \frac{1}{MT_0}\int_0^{MT_0} I(t)dt \approx \frac{C_1}{2}\cos(\phi_1) \text{ and} \tag{7.142}$$

$$Q_{sum} \approx \frac{1}{MT_0}\int_0^{MT_0} Q(t)dt \approx \frac{C_1}{2}\sin(\phi_1), \tag{7.143}$$

where $C_1$ and $\phi_1$ are the magnitude and phase, respectively of the cantilever return signal from Equation 7.102.

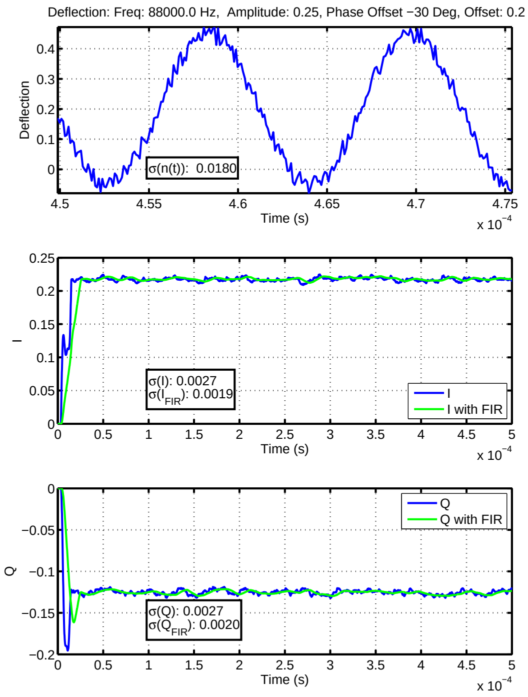


Figure 7.70: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is 0.25. There is a normalized offset of 0.2 in the signal level, and the phase of the signal driving the deflection is 30° behind of that of the in-phase (sine) mixing signal at the beginning of the simulation.

The classic method of computing magnitude and phase is from a coordinate transformation from rectangular to polar coordinates , *i.e.*,

$$C_1 = 2\sqrt{I_{sum}^2 + Q_{sum}^2} \text{ and } \phi_1 = \text{Arctan}\left(\frac{Q_{sum}}{I_{sum}}\right). \tag{7.144}$$

The difficulty comes in the resources needed to compute these relationships in real time with high sample rates. For example, a highly efficient algorithm is the so called CORDIC algorithms [227, 228, 257]. This algorithm computes magnitude and phase by rotating the frame of reference until the frame of reference and the signal have a matching magnitude and phase. The CORDIC algorithm is computationally simple, and is at the heart of the trigonometric calculations in the original HP-35 calculator [228]. It is even available now in logic cores for FPGAs, [258]. However, to compute magnitude and phase, a CORDIC algorithm requires one computational cycle per bit of accuracy, so a 16 bit accuracy would require an extra computational delay (on top of that done by the integral itself) of 16 clock cycles. In a standard computer, this might be considered fast, but in a DSP or FPGA which typically complete table lookup operations, additions, and multiplies in one or two cycles, this is considered slow. Alternately, some have chosen to offload the magnitude and phase calculation to a DSP chip once the I and Q branch

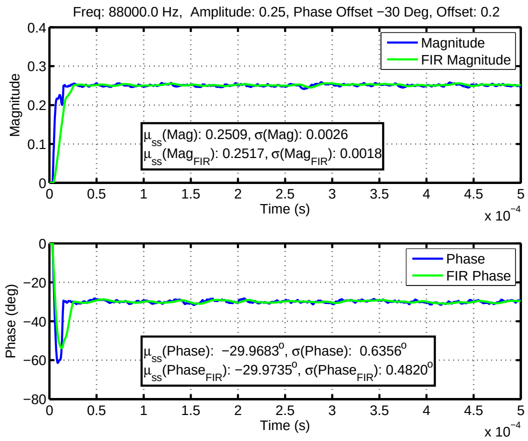


Figure 7.71: Floating point magnitude and phase extracted from the simulation in Figure 7.70.

demodulations are done, [259].

A more time efficient calculation involves a table lookup. However with two different numbers to look up and the high precision desired in these calculations, the tables can become huge. A 16-bit quantity would nominally require $2^{16}$ = 64K values. A few well placed adjustments to and restrictions of the calculation make it possible to use a relatively small table to give reasonably good estimates. The process for doing this involves two steps:

- Scaling the numbers so as to restrict the input range of the table, and then unscaling them after the table has been used.
- Interpolating between points in the lookup table using extra bits from the calculation.

### 7.19.7 Calculating Magnitude Using Table Lookup

It is certainly possible to do a simple table lookup to compute the magnitude, $C_1$, from the left side of Equation 7.144. The question becomes how to do this to efficiently make use of memory. The first thing that we notice from Equation 7.144 is that the number under the square root sign is a sum of squares. This means that the number inside is positive. Several other features of the calculation are useful to realize:

- Sums of squares of a number are easy to compute using in most real-time computer hardware, including DSPs and FPGAs which have built in hardware multiplies.
- Square roots are hard to compute quickly.
- The slope of the $\sqrt{x^2}$ changes very rapidly near $x^2 = 0$, which makes interpolation less accurate.
- Limiting the input range of the $\sqrt{x^2}$ lookup table to the range from 0.5 to 2.0, makes $\sqrt{x^2}$ well behaved.
- Since $I_{sum}^2 + Q_{sum}^2$ is in 2's compliment notation, it will have one or more leading 0$s$.

Furthermore, we want one lookup table for all values of $I_{sum}^2 + Q_{sum}^2$. We can minimize the entries in the table lookup by shifting to the left until the leading two bits are $01$, $10$, or $11$, and keeping track of the left shifts. A left shift by $2n$ bits effectively multiplies the number by $2^{2n}$. This pins the 2's compliment number in the table to be between $0.5$ and $2$, i.e. if

$$x_{in}^2 2^{2n} = \left(I_{sum}^2 + Q_{sum}^2\right) 2^{2n}, \text{ then} \tag{7.145}$$

$$0.5 \leq x_{in}^2 2^{2n} < 2. \tag{7.146}$$

Once the square root of the shifted sum of squares is looked up, we shift the result to the right by $n$ bits, since

$$y = \sqrt{\frac{x_{in}^2 2^{2n}}{2^{2n}}} \text{ then } y = \frac{\sqrt{x_{in}^2 2^{2n}}}{2^n}. \tag{7.147}$$

What was shifted by left by $2n$ bits (i.e. multiplied by $2^{2n}$), is shifted to the right by $n$ bits (i.e. divided by $2^n$) once the square root has been looked up.

With a memory that has $2^M$ locations and a smooth curve, improved accuracy can be achieved by splitting the address space and using part of the values for interpolation. For example, with $2^{10}$ locations, a lookup table of square roots of input values between 0 and 2 has an accuracy of only 4 bits. Using $2^9$ locations for lookup points and $2^9$ locations for slopes to do linear interpolation (for a total of $2^{10}$ locations) leads to an accuracy of 6 bits the same number of memory locations. Restricting the input range to being between between 0.5 and 2 for a lookup table with $2^{10}$ locations gives an accuracy of 9 bits. Using $2^9$ locations for lookup points and $2^9$ locations for slopes to do linear interpolation (for a total of $2^{10}$ locations) gives an accuracy of 19 bits as seen in Figure 7.72. This is made possible in large part by the shift away from values near 0 with steep slopes and by the smoothness of the square root function between input values of 0.5 and 2.

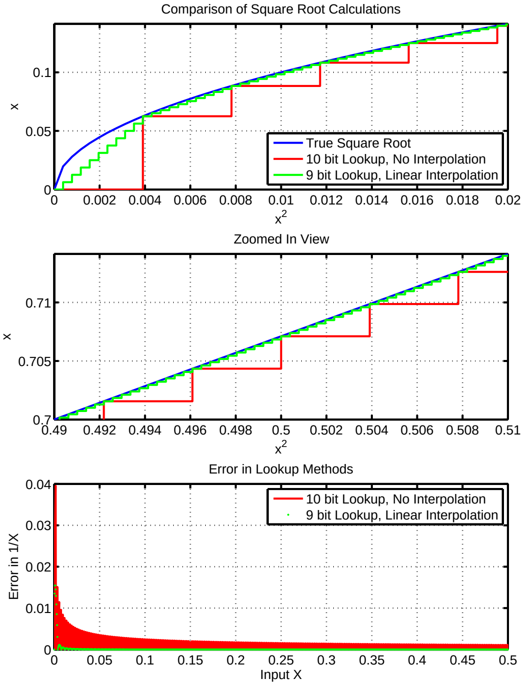


Figure 7.72: Comparison of table lookup and errors for $\sqrt{x}$, $0.0 \leq x \leq 2$. Plots are zoomed in to show more interesting aspects of the data. Note the significant drop in error away from $x = 0$.

Making a relatively mild assumption that a single computation can be done in a single clock cycle of the FPGA or DSP, the procedure and the estimated latency are:

1) Square $I_{sum}$ and $Q_{sum}$ (1 clock cycle).

2) Add them together yielding $x_{in}^2$ (1 clock cycle).

3) Shift left by the maximum even number of leading 0s, $(2n)$ to yield $x_{in,2n}^2$ where $0.5 \leq x_{in,2n}^2 < 2$ (1 clock cycle).

4) Look up the square root in a table (1 clock cycle).

5) (Optional) Look up interpolation slope in table (1 clock cycle, in parallel with other lookup).

6) (Optional) Interpolate square root value between lookup points (2 clock cycles, 1 for multiply and 1 for addition).

7) Shift square root value right by half as many bits as the previous left shift ($n$) (1 clock cycle).

So, this table lookup computes the square root in 7 clock cycles, irrespective of the number of bits in the input. For data widths of greater than 7 bits, this is faster than the CORDIC algorithm.

### 7.19.8 Calculating Phase Using Table Lookup

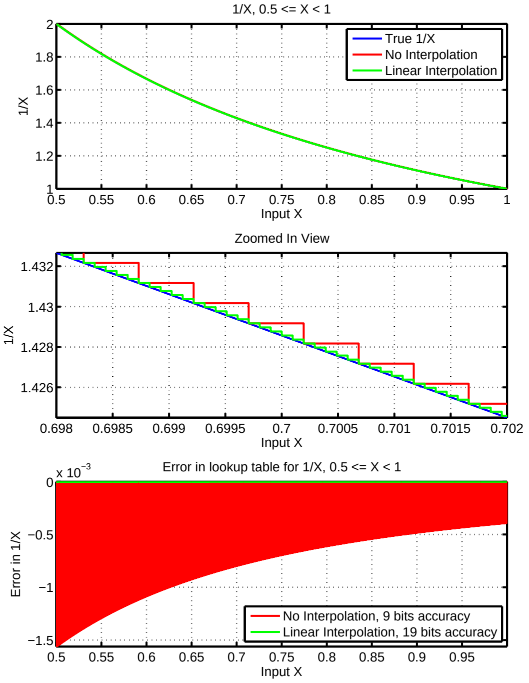


Figure 7.73: Comparison of table lookup and errors for $\frac{1}{x}$, $.5 \leq x \leq 2$. The lookup with interpolation once again significantly outperforms the straight table lookup.

We also need to compute the arctangent of $\frac{Q_{sum}}{I_{sum}}$ digitally. Again, the CORDIC may be too slow for our purposes and we want to use a table lookup. We want a unique table and a little bit of trigonometry knowledge allows us to use a single table. We can limit the operation to the first quadrant $(0 \leq \frac{Q_{sum}}{I_{sum}} < \frac{\pi}{2})$

by keeping track of the signs of the input values and then simply working with the absolute values. We then shift the result back into the proper quadrant by some simple math. The problem is that it is hard to be accurate in a table for the $\text{Arctan}\frac{Q_{sum}}{I_{sum}}$ when $I_{sum}$ is close to $0$. However, we can easily look up

$$\text{Arccot}\left(\frac{Q_{sum}}{I_{sum}}\right) = \frac{\pi}{2} - \text{Arctan}\left(\frac{I_{sum}}{Q_{sum}}\right), \tag{7.148}$$

and so we operate in the first half of the first quadrant $(0 \leq \frac{Q_{sum}}{I_{sum}} < \frac{\pi}{4})$:

- If $|I_{sum}| = |Q_{sum}|$, then $\text{Arctan}(1) = \frac{\pi}{4}$.
- If $|I_{sum}| > |Q_{sum}|$, then lookup $\text{Arctan}\left(\frac{Q_{sum}}{I_{sum}}\right)$.
- If $|I_{sum}| < |Q_{sum}|$, then lookup $\text{Arctan}\left(\frac{I_{sum}}{Q_{sum}}\right)$ and compute $\text{Arccot}\left(\frac{Q_{sum}}{I_{sum}}\right) = \frac{\pi}{2} - \text{Arctan}\left(\frac{I_{sum}}{Q_{sum}}\right)$.

Now, we see that before we can look up an $\text{Arctan}$, we need to compute either $\frac{|Q_{sum}|}{|I_{sum}|}$ or $\frac{|I_{sum}|}{|Q_{sum}|}$, depending upon whether $|Q_{sum}| \text{ or } |I_{sum}|$ is larger. This means looking up $\frac{1}{|Q_{sum}|}$ or $\frac{1}{|I_{sum}|}$.

Say we want to compute the value $\frac{Y}{X}$ where $X$ is the larger of $|Q_{sum}|$ and $|I_{sum}|$. We have to look up $\frac{1}{X}$ in a table and multiply this by $Y$. $\frac{1}{X}$ is badly behaved when $X$ is close to $0$, but we can shift both numerator and denominator:

$$\frac{Y}{X} = \frac{2^n Y}{2^n X}, \tag{7.149}$$

so that the leading $0$s in $X$ have been eliminated. We have already assumed that $X > Y$ so in a 2's complement format $Y$ should have at least as many leading $0$s as $X$. This means:

a) The value of $2^n X$ in the lookup table for $\frac{1}{2^n X}$ is always between $1$ and $2$ so The looked up value is always between $\frac{1}{2}$ and $1$.

b) $\left(\frac{1}{X}\right) Y$ is always between $0$ and $1$.

c) Therefore the looked up and interpolated values should be quite accurate as seen in Figure 7.73. Without interpolation, the $2^{10}$ location table achieves 9 bits of accuracy. Splitting the table into two $2^9$ location tables where the first part is for lookup and the second part is for linear interpolation results in 19 bits of accuracy. Again, we have made use of shifts in the original values and the smoothness of $\frac{1}{X}$ for $X \geq 1$.

For simplicity, we can look up values between $0$ and $1$ and scale them by $\frac{\pi}{4}$ in post processing. So, our procedure is as follows:

1) Compute $|I_{sum}|$, $|Q_{sum}|$, $\text{sgn}(I_{sum})$, and $\text{sgn}(Q_{sum})$ (1 cycle).

2) Determine if $|I_{sum}|$ or $|Q_{sum}|$ is larger. If $|I_{sum}| = |Q_{sum}|$, the $\text{Arctan is } 1$ (1 cycle).

3) Shift both $|I_{sum}|$ and $|Q_{sum}|$ left to eliminate leading zeros in larger value (1 cycle).

4) Look up $\frac{1}{X}$, where $X$ is the larger of $|I_{sum}|$ and $|Q_{sum}|$ (1 cycle).

5) (Optional) Look up interpolation slopes for $\frac{1}{X}$ (1 cycle).

6) (Optional) Interpolate values between lookup points of $\frac{1}{X}$ (2 cycles, 1 for multiply and 1 for addition).

7) Multiply $\frac{1}{X}$ by $Y$, where is the smaller of $|I_{sum}|$ and $|Q_{sum}|$ (1 cycle).

8) Look up $\text{Arctan}\frac{Y}{X}$ where $0 \leq \frac{Y}{X} < 1$ (1 cycle).

9) (Optional) Look up interpolation slopes for $\text{Arctan}\frac{Y}{X}$ (1 cycle).

10) (Optional) Interpolate values between lookup points of $\text{Arctan}\frac{Y}{X}$ (2 cycles, 1 for multiply and 1 for addition).

11) Calculate $\text{Arccot}\frac{X}{Y} = \frac{\pi}{2} - \text{Arctan}\frac{Y}{X}$ if needed (1 cycle).

12) Use $\text{sgn}(I_{sum})$ and $\text{sgn}(Q_{sum})$ to put the $\text{Arctan}$ or $\text{Arccot}$ in the proper quadrant (1 cycle).

This computation takes 14 cycles, which is more than the original integration from Section 7.19.3 [85]. However, it seems to be the fastest method of directly computing the phase for any numbers of bits greater than 14, unless one trivializes the magnitude and phase operation. The next section shows just how to do just that.

### 7.19.9 Using a PLL to Simplify Magnitude and Phase Calculations

The previous sections on computing magnitude and phase point out the issue that even the fastest methods of computation generate a significant amount of serial steps in the process. Assuming that each step can be done in one computation clock cycle (not always the case with FPGAs, and even more rare with DSPs), we still have quite a few cycles. For fast sampling, this can be a large fraction of a sample period. Furthermore, these calculations take significant resources, either in time – if the computation is done on a standard processor or a DSP – or in space if the computation is done in programmable logic – such as with a FPGA.

There is a way to circumvent these extra computations, and this is by noticing that if the mixing signal is in phase with the signal to be demodulated, then the magnitude drops out trivially from the integral. In other words, if the I mixing signal is in phase with the signal to be demodulated, then the Q mixing signal is $90^\circ$ out of phase meaning that on average, $Q_{sum}$ is very close to $0$. This means that the left side of Equation 7.144 is reduced to

$$C_1 \approx 2\sqrt{I_{sum}^2} = 2I_{sum}. \tag{7.150}$$

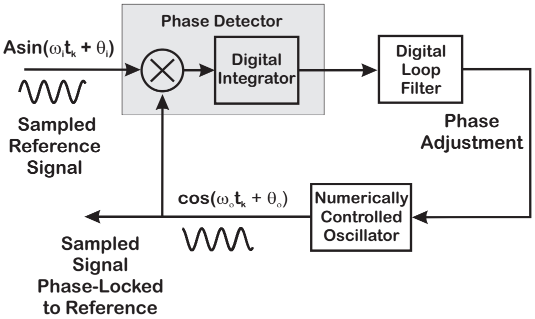


Figure 7.74: A digital mixing PLL including post mixing integration. Note that our demodulator mixes the input signal with numerically generated sinusoids and integrates them over an integer number of periods, [85]. The end result is an output that has very little of the 2X frequency component in the classical mixing loop. In other words, the Q branch of our demodulator calculates the phase error between the input signal and the sinusoid.

Furthermore, in place of the right side of Equation 7.144, we know that the average phase difference between the return signal and our driving signal is equal to the phase difference between our I mixing signal and our driving signal, and this can be read off trivially. The integral of $Q_{sum}$ has an elegant interpretation as the instantaneous phase difference between the return signal and the average phase. The difference between the phase of the mixing Numerically Controlled Oscillator (NCO) and the drive NCO represents the average phase.

However, since the average phase of the return signal is unknown, we need some way to identify it. The classic way is with a phase-locked loop (PLL). A block diagram of a digital PLL is shown in Figure 7.74. Each PLL has a reference signal and an oscillatory signal which will lock to it, [30]. The oscillatory signal is presumed to be at a frequency close enough to the reference signal so that differences between the frequencies can be considered phase errors. A phase detector extracts the baseband phase difference between the two signals as well as some higher frequency information to be filtered out. The properties of the loop, set by the combination of the phase detector and the loop filter, determine the phase changes that can be followed and which changes will be treated as disturbances to reject. Note that Figure 7.74 shows an NCO, but in general for a PLL, this can be replaced by a variety of oscillators including numerical sinusoid generators, numerical square wave generators, and clock generators. In our case, precise generation of numerical sinusoids greatly reduces the generation of extra harmonics and therefore lowers the integration error. Of equal importance is that there is a means for adjusting the phase of the oscillator.

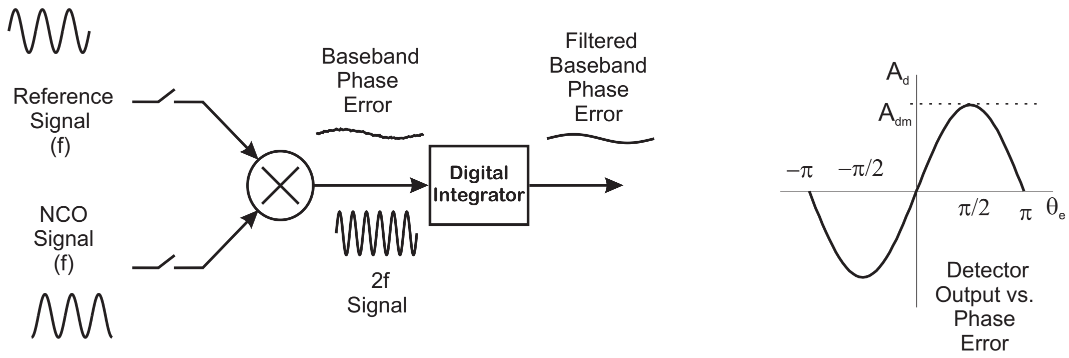


Figure 7.75: Input output properties of digital mixing phase detector. The multiplication (mixing) produces signals at the baseband and at twice the input frequency ($2f$). The digital integration over an integer number of periods of $f$ theoretically eliminates the $2f$ signal. On the right is the detector output versus the phase error. It can be shown that with the right choice of filters, this PLL will lock to any slowly changing phase.

Looking at the details of Figure 7.74, the input signal is sampled, just as our return signal from the AFM is sampled. A sampled oscillatory signal is multiplied (mixed) with this sampled signal and the output passes through the loop filter to adjust the phase of the NCO, which generates the mixing signal. It turns out that this is very similar to the form that we already have in the demodulator.

Our demodulator mixes the input signal with numerically generated sinusoids and integrates them over an integer number of periods. The end result is an output that has very little of the 2X frequency component in the classical mixing loop. In other words, the Q branch of our demodulator calculates the phase error between the input signal and the sinusoid. All we need to add to the pieces in our demodulator are the ability to adjust the phase of the sine drive (our NCO) and a loop filter to regulate the bandwidth of this loop adjustment.

Note that our sine drive/NCO produces two signals that are 90° out of phase with each other, a sine and a cosine. The cosine is used to generate and minimize the phase error as described above. The sine is used to extract the amplitude of the signal, since it will be in phase with the input signal.

The signal properties of a digital mixing phase detector are shown in Figure 7.75. The left diagram

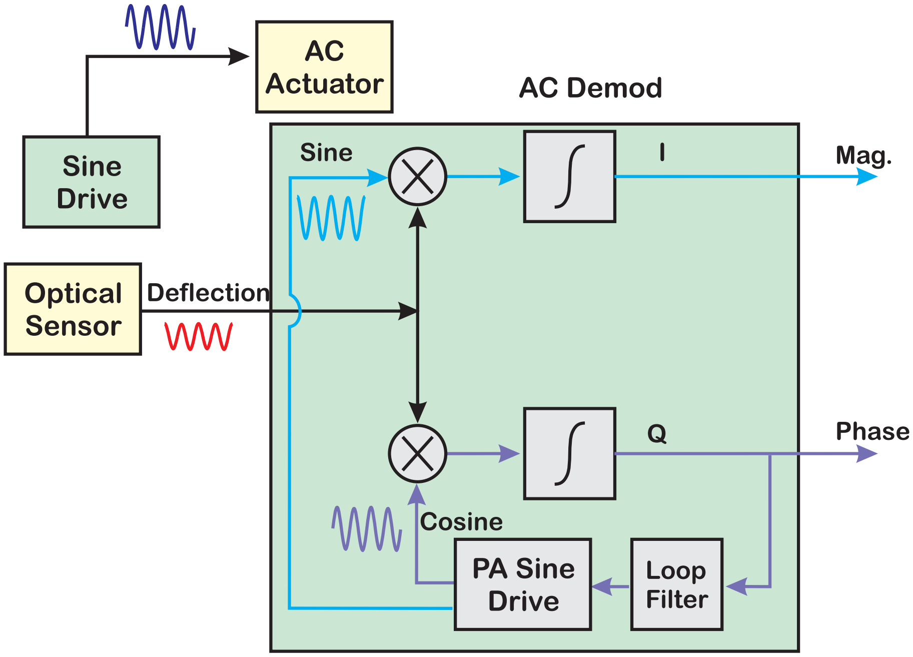


Figure 7.76: Coherent demodulation for AFM using a PLL. The phase adjustable (PA) sine drive is our NCO.

shows schematically the time domain operation of the mixer. Multiplication (mixing) produces signals at the baseband and at twice the input frequency $(2f)$. The digital integration over an integer number of periods of detector output versus the phase error. It can be shown that with the right choice of filters, this PLL will lock to any slowly changing phase.

A block diagram that shows the PLL as a part of the overall demodulator is shown in Figure 7.76. A block diagram that also includes the post integration filtering of Section 7.19.4 is shown in Figure 7.77. We now see that in the magnitude and phase computation blocks have been eliminated by this method. From a delay point of view, this means that the delay through the demodulator is governed only by the integration portion. Furthermore, the adjustment of the mixer's phase by the PLL is done as a sort of background process, outside the time critical flow. Finally, the resources occupied by this method, whether they be CPU time or space on a FPGA are far smaller than the previous method.

Finally, it is often the case with AFMs that the equations are written so that the driving sinusoid is considered a cosine rather than a sine. This doesn't change the behavior, but does change the style of the mathematical analysis.

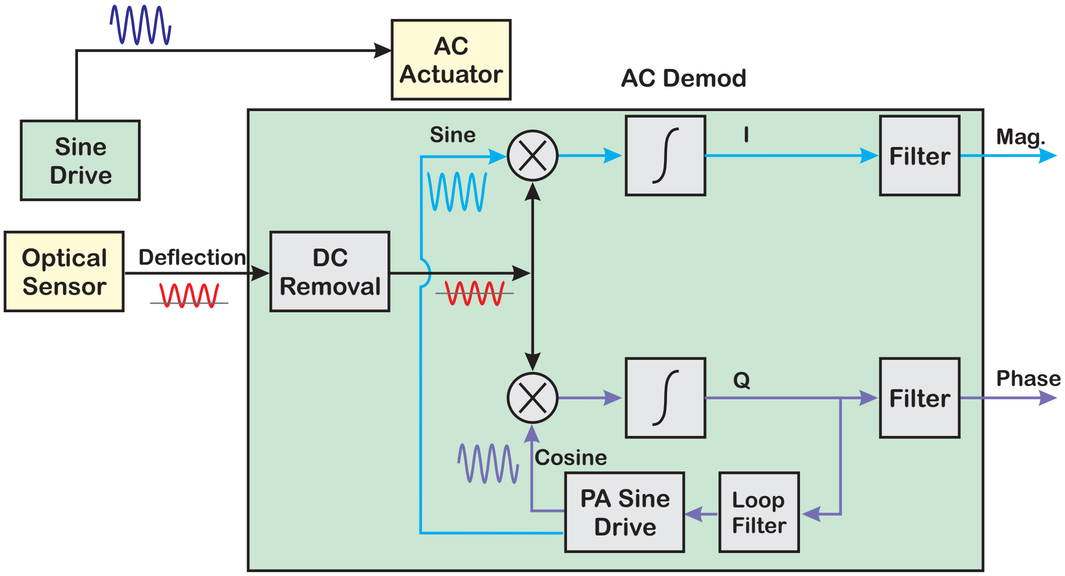


Figure 7.77: Coherent demodulation for AFM using a PLL. DC removal and post integration filtering included.

### 7.19.10 Surface Stick Detection

In responding to the amplitude of the tip oscillation, the controller responds by trying to keep the amplitude to some set fraction of the free oscillation amplitude, as diagrammed in Figure 7.62. A rise in the oscillation amplitude is interpreted as moving away from the surface, and the controller corrects by moving the tip closer. A drop in amplitude is viewed as moving closer to the surface, and the controller responds by moving the tip away. Coming completely off of the surface, while not desirable, does not damage the system and the tip will keep oscillating. However, if the tip sticks to the surface, all oscillations will stop, meaning the PLL based magnitude and phase detection described in Section 7.19.9 will lose lock. While the lack of oscillation should make the demodulated integrals go to $0$, there may be concerns about the transient behavior. The magnitude detection method of Section 7.19.7 will still work, but that involves a lot of extra computation for a safety measure. However, we can use the fact that for a vector, $x$,

$$||x||_2 \leq ||x||_1 \tag{7.151}$$

and so if the $l_1$ norm of $I_{sum}$ and $Q_{sum}$ goes to $0$, then the $l_2$ norm must also go to $0$. While the $l_1$ norm does not give an accurate amplitude measure, it does bound it from above and is trivial to compute, with no need for a CORDIC method or table lookup.

1) Compute $|I_{sum}|$ and $|Q_{sum}|$ (1 cycle).

2) Compute $|I_{sum}| + |Q_{sum}|$ (1 cycle).

3) Compare the sum to some lower bound threshold (1 cycle).

Not only is this calculation simple, but it can be done in parallel with the PLL based extraction of magnitude and phase. In fact, the 3-cycle stick detection is available much sooner than it would be with either a CORDIC method or the table lookup method of Section 7.19.7.

## 7.20 Magnitude and Phase Calculation Examples

The demodulator architecture, implemented in FPGA hardware, was simulated using ModelSim 6.6b [256], and signals of interest to this example were saved to an ASCII file. The ASCII data was normalized using MATLAB so that the number format was back in a more convenient form. Also, small gain differences between the I and Q phase outputs before and after filtering have been normalized out. The last 20% of the data was used to compute the steady state averages ($\mu$) and standard deviations ($\sigma$) of these signals. In these simulations, the units of $Q$ map to radians, since $Q$ is used as the phase error.

| Relevant Figures | Input Noise ($\sigma$) | $I_{sum}$ $\sigma$ | $Q_{sum}$ $\sigma$ |
|---|---|---|---|
| Fig. 7.78 & 7.79 (no filt.) | 0 | 1.774407e-03 | 1.763996e-03 |
| Fig. 7.78 & 7.79 (notch) | 0 | 0 | 6.920047e-05 |
| Fig. 7.78 & 7.79 (FIR) | 0 | 0 | 1.318177e-04 |
| Fig. 7.80 & 7.81 (no filt.) | 0 | 1.763196e-03 | 1.765161e-03 |
| Fig. 7.80 & 7.81 (notch) | 0 | 2.094685e-05 | 7.814222e-05 |
| Fig. 7.80 & 7.81 (FIR) | 0 | 0 | 1.340324e-04 |

Table 7.3: Computed steady state noise values extracted from integrator simulations. The last 20% of the data was used to get steady state values. The input noise to these simulations was set to 0 to better view the convergence of the PLL based $I_{sum}$ and $Q_{sum}$ to the magnitude and phase values, respectively.

Figure 7.78 shows an 88 kHz oscillation frequency, sampled at 1 MHz. The effect of the PLL based demodulator is that the mixing signals are shifted so that the in-phase (I) signal aligns with itself with the average phase of the input signal. The quadrature signal (Q) is aligns itself so that it is $90^\circ$ out of phase with the average of the input signal. This simulation has no offset in the level of the deflection (input) signal, but a $+30^\circ$ initial phase offset which the PLL portion quickly tracks. Thus, the magnitude calculation is obtained trivially from the in phase integral. In the lower two plots of Figure 7.78, we also see that the I term matches the computed magnitude and the Q term is very close to 0, which confirms

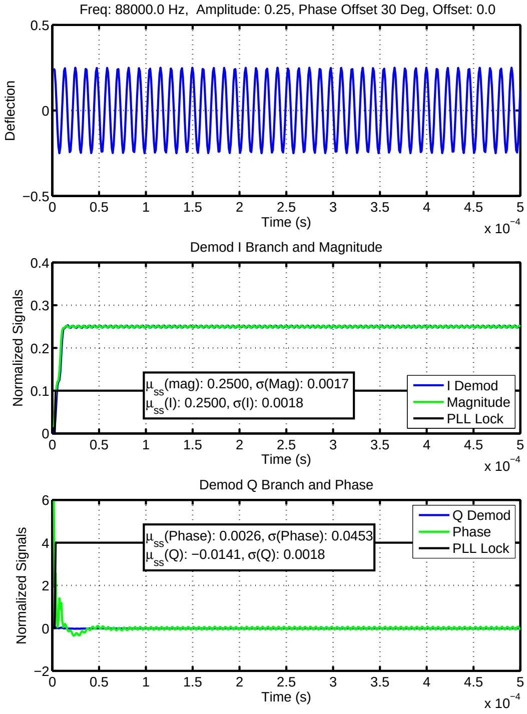


Figure 7.78: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is $0.25$. There is no offset in the signal level, but the phase of the signal driving the deflection is $30^\circ$ ahead of the in-phase (sine) mixing signal at the beginning of the simulation. Note how the I and Q phases converge to the magnitude and instantaneous phase, respectively.

phase tracking. The black curves there are the PLL lock indicator, which shows when the PLL error is considered small enough for it to be considered locked.

Figure 7.80 show a similar simulation but with an offset of $0.1$ and a $-179^\circ$ phase offset. Note how the PLL lock signal corresponds to the mixing signals being close enough to ideal in-phase and quadrature such that I and Q equal the magnitude and phase.

Figures 7.79 and 7.81 show the effects of using post integration filtering on the demodulated signals. In particular, Figure 7.79 which has particularly small phase error throughout, is scaled such that one can really see the improvement of the filtered signals.

The top plots of both figures show the I branch, while the lower plots show the Q branch. The unfiltered output of the demodulator is shown in blue. That output, post processed with an FIR filter which removes all the harmonics of the oscillation frequency, are shown in green. Looking at the Q outputs

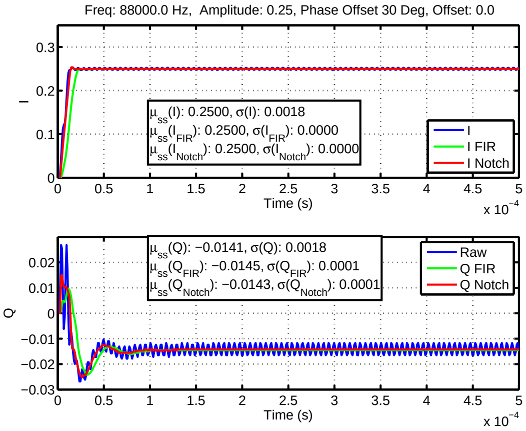


Figure 7.79: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is $0.25$. There is no offset in the signal level, but the phase of the signal driving the deflection is $30^{\circ}$ ahead of the in-phase (sine) mixing signal at the beginning of the simulation. This plot shows the effect of adding post post integration filtering to the demodulator.

which are zoomed in due to the small size of output signals, we can see that the unfiltered demodulator outputs have a component at twice the oscillation frequency, $f_0$. Thus, a simple notch filter, shown in red, can be used to remove this component from the output. The choice between the FIR and the notch depends upon the number of harmonics that one is concerned with versus the additional computational latency that one is willing to accept. However, in this case, the notch at $2f_0$ performs indistinguishably from the FIR, and the latency is clearly less than the FIR, although slightly more than the unfiltered results.

One more signal of interest is the PLL locked indicator. This is applied in the demodulator when the integral of the I branch stays positive and significantly larger than the absolute value of the Q branch integral. Note that once this signal becomes positive, the I branch integral is very close to the input signal magnitude.

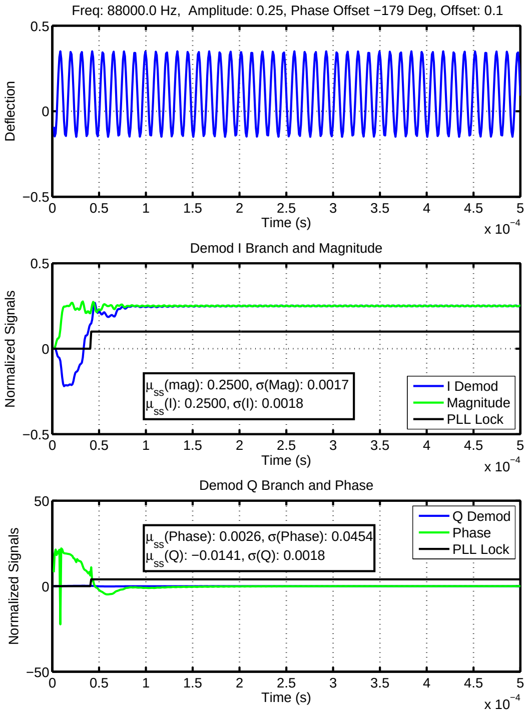


Figure 7.80: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is $0.25$. There is a normalized offset of 0.1 in the signal level, and the phase of the signal driving the deflection is $179^\circ$ behind that of the in-phase (sine) mixing signal at the beginning of the simulation. Again, the I and Q phases converge to the magnitude and instantaneous phase, respectively.

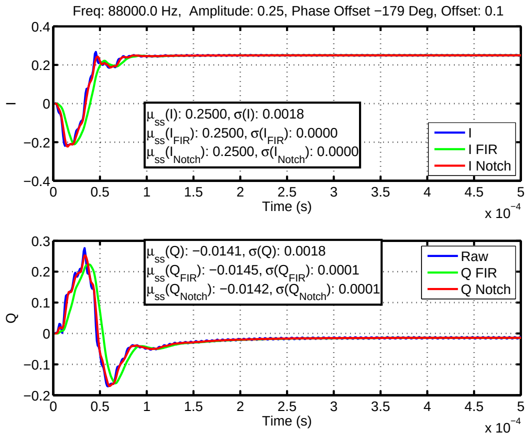


Figure 7.81: Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is $0.25$. There is a normalized offset of 0.1 in the signal level, and the phase of the signal driving the deflection is $179^\circ$ behind that of the in-phase (sine) mixing signal at the beginning of the simulation.

### 7.20.1 Summary of Coherent Demodulator for AFM

This section demonstrates a low latency, high accuracy, AC mode demodulator that is applicable for high speed, real-time applications, such as dynamics mode control of Atomic Force Microscopes (AFMs) The algorithm can be considered as either classical coherent demodulation or a lock-in-amplifier (LIA), but implemented in a clean and flexible structure with minimal computational overhead.

Since the original publications of these algorithms, they have been used in the Keysight 9500 AFM Controller [260]. This controller samples the Z-axis loop at 10 MHz. Using the demodulator described here, it is able to produce samples for a 1 MHz feedback rate. The method is robust to the variations involved in the mass manufacture of a commercial product.

The use of block RAM to hold old values of signal does provide a practical limit on how long our integration can be. For example, with a 50 MHz sample rate and a 100 kHz tip oscillation frequency, a single period of oscillation would require 500 samples, so 4K words of block RAM would allow us only 8 periods of oscillation. However, there are two mitigating factors. First of all, the entire scheme described here is designed to allow the number of oscillation periods to be small, providing for minimum latency. Secondly, 500 samples per oscillation is almost certainly overkill, and thus the demodulator sample rate could be kept closer to 20 times the oscillation frequency, allowing even a 1K word block to enable 500 periods of oscillation.

Furthermore, while not the original use of this work, the efficiency of the methodology allows it to be duplicated for multiple frequencies of oscillation on the same FPGA chip with little or no loss of bandwidth. That is, because the method is completely implementable in programmable logic, these blocks can be multiplexed in space on the chip, rather than in processing time, and with minimal recombination, can be used for stimulating, and demodulating multimode oscillations such as described in [261].

The other major difference between the precision IQ demodulation used in real-time feedback and that used in say a stepped-sine calculation is in extracting the magnitude and phase from the output of the precision integrators. In Figure 7.64, the standard rectangular to polar computations are done.

To extract the magnitude in real-time for a feedback calculation requires the computation of Equation 7.82. The difficulty comes in the resources needed to compute these relationships in real time with high sample rates. For example, a highly efficient algorithm is the so called CORDIC algorithms [227, 228]. This algorithm computes magnitude and phase by rotating the frame of reference until the frame of reference and the signal have a matching magnitude and phase. The CORDIC algorithm is computationally simple, and is at the heart of the trigonometric calculations in the original HP-35 calculator [228]. However, to compute magnitude and phase, a CORDIC algorithm requires one computational cycle per bit of accuracy, so a 16 bit accuracy would require an extra computational delay (on top of that done by the integral itself) of 16 clock cycles. In a standard computer, this might be considered fast, but in a DSP or FPGA which typically complete table lookup operations, additions, and multiplies in one or two cycles, this is considered slow.

A more time efficient calculation involves a table lookup. However with two different numbers to look up and the high precision desired in these calculations, the tables can become huge. A 16-bit quantity would nominally require $2^{16} = 64\text{K values}$. A few well placed adjustments to and restrictions of the calculation make it possible to use a relatively small table to give reasonably good estimates. A fuller discussion is found in [86].

A a faster way is to use the knowledge of phase-lock techniques to simplify the calculation. The quadrature branch (Q) of the integral is very close to a PLL, if we allow the mixing oscillator to have its phase adjusted in response to the phase error in that branch. With the mixing signal phase-locked, the in-phase (I) branch is aligned so that the output of that integral is proportional to the cosine of the phase difference. As that phase difference is driven to the vicinity of $0$, the cosine is approximately $1$, and the magnitude drops out trivially.

Some examples of the demodulator from Figure 7.77 in action are shown in Figures 7.78 and 7.79. We see in Figure 7.78, the original oscillation signal on top, with the I and Q branch integrals below. Because of the PLL structure of the lower, Q branch, we see that the Q integral converges to $0$ as the loop locks, leaving the I branch locked to the magnitude. In Figure 7.79, we examine the behavior of post integration filtering on the result, and we can see (especially in the lower Q plot) that a little bit of filtering removes a residual $2f_0$ frequency left over from imperfect integration.

Since this was originally published, even more exotic methods of demodulation for AFM oscillations have been published [218, 219, 262].

## 7.21 Example: Servo Signal Demodulation in Hard Disk Drives

Hard disk drives (HDD) are still in use to store the majority of online data in the world. The HDDs encode servo position information in magnetic domains offset from the track center. Hard disk drives (HDD)

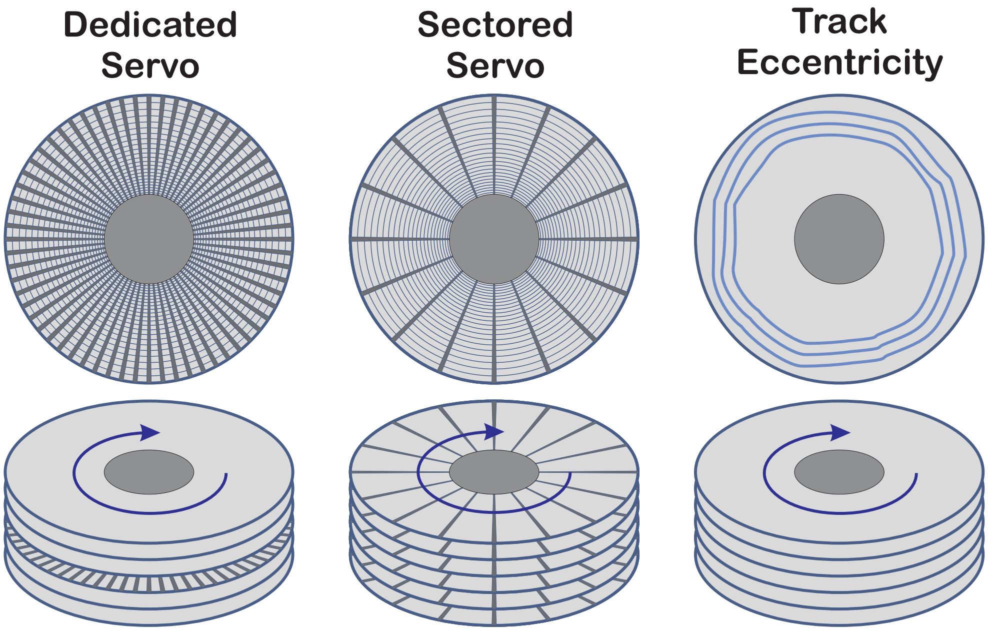


Figure 7.82: HDDs use either Dedicated Servo surfaces (left drawing) or Sectored Servo (center drawing) to provide position information so that the head can follow the track eccentricity (right drawing).

provide a breadth of examples of modulation and demodulation schemes used to encode position. We can trace an evolution of demodulation methodologies from looking at different methods we might apply to the same basic signals. Dedicated Servo uses one surface of disk stack (left drawing of Figure 7.82) and has a relatively high sample rate, but using a reference position on a center disk to track position on one of the outer surfaces can cause issues due to physical offsets caused by thermal expansion and contraction, as well as any small clamping slippage over time. Sectored or Sampled Servo (center drawing of Figure 7.82) multiplexes servo information with user data in periodic samples along the track known as servo bursts. Because the servo information takes away data storage capacity, there is always a tension between the desire to lower the sample rate to make more room for data and the need for higher sample rates. However, Sectored Servo has the physical advantage that the position sensing is co-located with data on the disk. In either case, the servo information should allow the head to follow the track eccentricities (right drawing of Figure 7.82).

To the best of my knowledge, the thermal offset issues mandate that almost all modern drives use Sectored Servo. This gives rise to a track layout pictured in Figure 7.83. The servo burst itself will consist of several fields: a clock sync field, servo position information, and an edit gap (Figure 7.83). The clock sync field is merely a pattern of alternating magnetic polarities along the track. These fields are consistent across the cross-track direction. This allows the PLL to minimize any phase drift that may have occurred during the data portion of the track. After this, there are the offset patterns that give the position information.

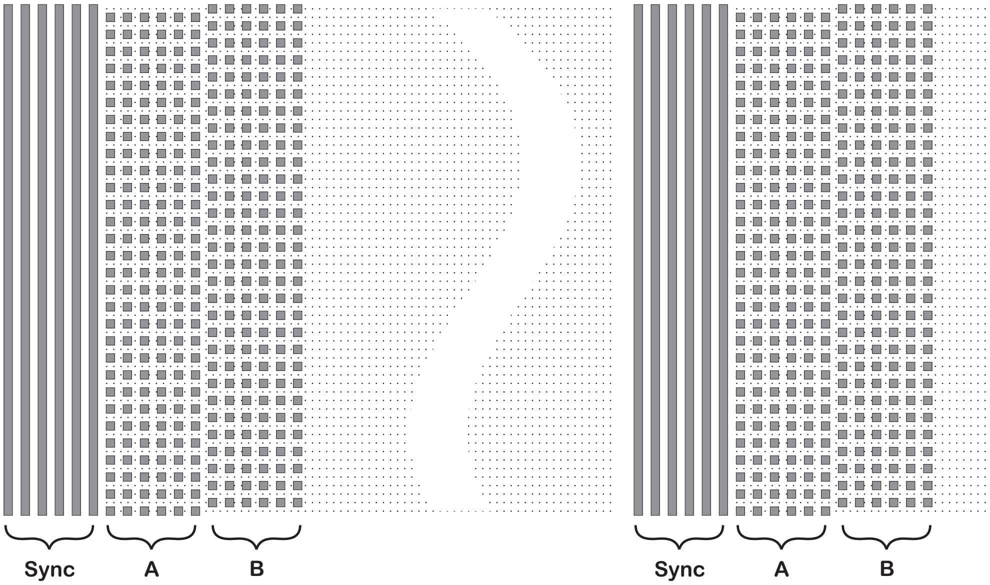


Figure 7.83: The layout of an HDD track. The sync field allows the PLL to recover its timing so that the fields can be detected at the right time. In this simplified form, there are just A and B fields offset by half a track from the track center. After the A and B fields comes the user data, which typically comprises a much larger proportion of the track.

The magnetic domains that encode position are at different cross track offsets. In this image we only show the A and B fields. but most drives have C and D fields offset by a quarter track width from A and B. Some of them divide these offsets into sixths, so there are E and F fields for better linearity. No matter how many fields are present for position linearity, the mixed signal chip that converts the servo burst opens up a timing window based upon a clock synchronized on the last servo burst, then looks for certain marks that delineate a burst. Finally, there is a space included (either at the beginning or end of the user data) known as the edit gap, which is there to make certain that any residual timing errors do not result in the user data overwriting the information in the next servo burst. (It is this lack of edit gap in the DVD-ROM format that made it difficult to create a rewritable DVD format that was compatible with the DVD-ROM [263].) A highly simplified view of this for our discussion is shown on the left side of Figure 7.84. The top drawing shows the relative position of the read/write head relative to the offset position information. The goal is to track the center of the track and if the head is too far over one field (A or B), then the return signal is larger for that portion, as shown in the lower drawing.

With the clock recovered, the servo processing portion of the chip opens up windows for the A field and the B field, separately. The output signals from the electronics are illustrated in the lower drawing. We can see that when the readback head is off to one side of track center (in this illustration more towards A), that the corresponding readback will produce a higher amplitude in the signals illustrated in the lower

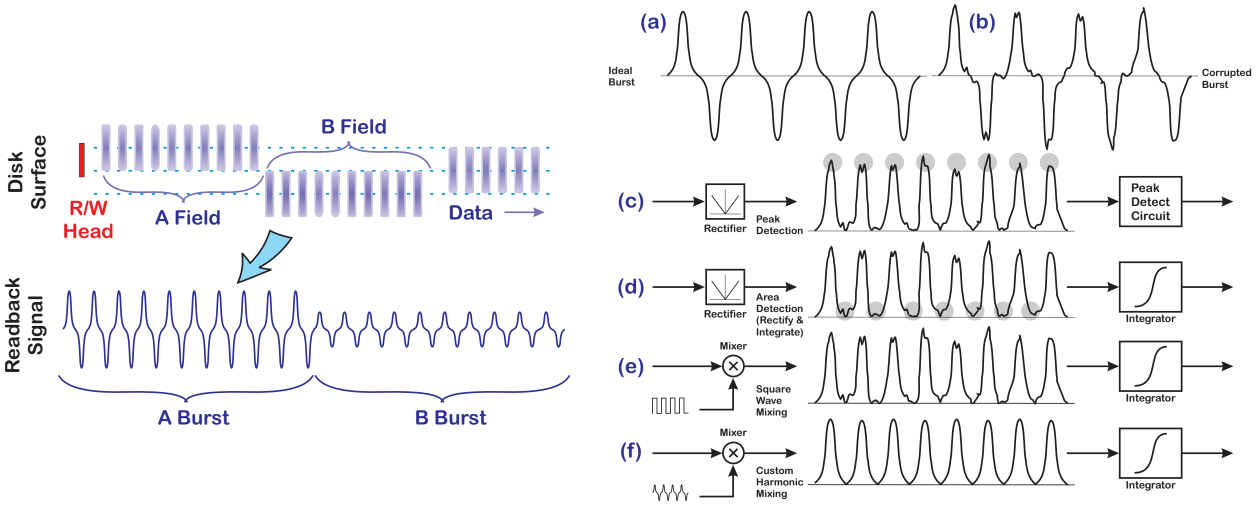


Figure 7.84: Modulated servo patterns on a hard disk drive and common demodulation methods used in decoding disk drive position bursts.

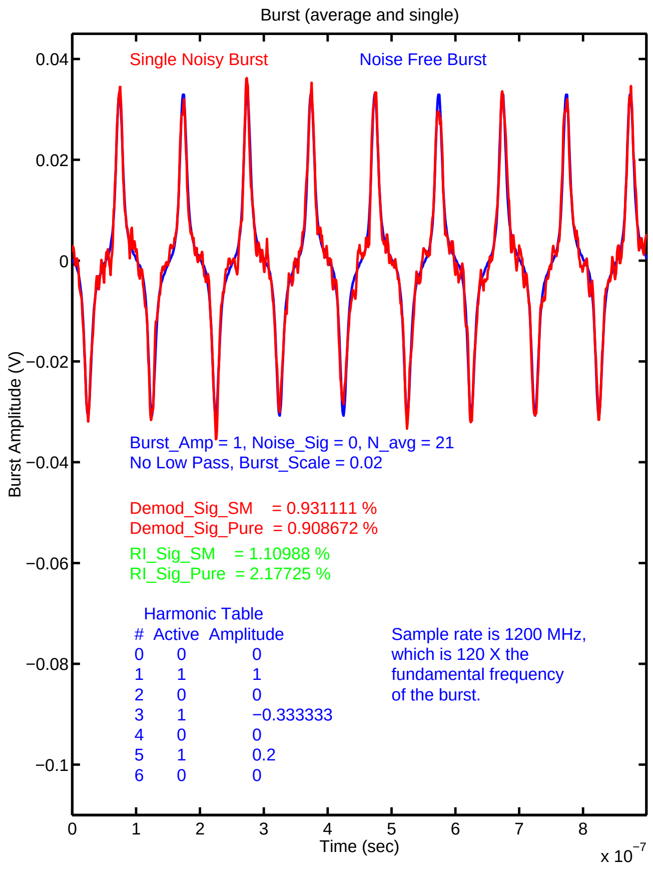
drawing. Now, assuming that the clocking allows us to separate the A "signal" from the B "signal", we have some options on how to extract their relative amplitude. A measurement from an HP Lynx 2 HDD made in 1995 is shown in Figure 7.85, along with a manual model fit. (HP exited the HDD business in 1996, so it has been harder to get HDD lab data since then.) The measurement itself was made using a digital oscilloscope at 120 MHz, which is fast enough to accurately record the shape of the signal. The signal itself is based on a 10 MHz fundamental oscillation. A manual model fit using the first three odd harmonics, the first, third, and fifth harmonics of a fundamental sine wave, is shown in blue.

Consider the hard disk drive servo signal example of Figure 7.84. In what was standard practice in hard disk control problems for many years, the position signals were written to straddle the track center. On the left we show a simplified pattern using only two fields, labeled A and B, but in practice more fields were used to help with cross track nonlinearity. The resulting signals were a combination of odd harmonics (ideally) and their amplitude was higher or lower depending upon the relative position of the read head. On the right are different candidate demodulation schemes:

a) Shows the ideal readback signal.

b) Shows a signal corrupted by noise and biases.

c) Shows a rectifier followed by **peak detection**. That is, the peak detector circuit picks the highest peak from the rectified pulses. The two-sided, zero-mean signal becomes a one-



Harmonic Table

| # | Active | Amplitude |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 0 | 0 |
| 3 | 1 | −0.333333 |
| 4 | 0 | 0 |
| 5 | 1 | 0.2 |
| 6 | 0 | 0 |

Figure 7.85: Measured Servo Burst on HP Lynx 2 HDD, Sampled at 120 MHz. Notice that the noise-free burst shape seems adequately modeled using the first, third, and fifth harmonics of the fundamental frequency of 10 MHz.

sided, non-zero-mean signal. This is not susceptible to noise around the zero crossings, but is highly sensitive to noise at the top of a peak, since there is no averaging there.

d) Shows a rectify and integrate circuit. The integration helps with the noise, but the rectifier has guaranteed that any noise or biases that might have been zero mean are now part of the integrated signal. Furthermore, every harmonic of the signal, good or bad, is included in the integral.

e) Shows a coherent demodulation in which a square wave of the same frequency and in phase with the servo carrier is mixed in with the return signal. This has one great advantage over the rectify and integrate circuit and that is that it does not give a nonzero bias to zero mean noise.

f) Shows a customizable coherent demodulator circuit. In this case, only the harmonics known to be clean (e.g. with a more linear cross track response) are mixed in with the return signal.

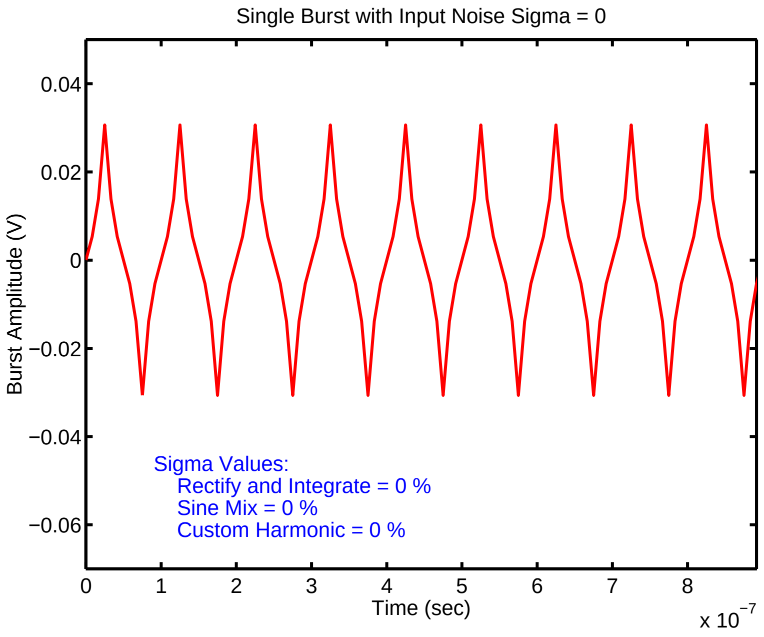


Figure 7.86: Ideal servo burst with no noise. Note that beyond the fundamental, the signal shape contains third and fifth harmonics.

It provides superior performance to all the others [211, 212, 213].

We can remove the rectification of noise by multiplying the signal in the window with a square wave aligned with the pulses. We see (in line (e)) that this rectifies the servo signal while the noise can still average towards its zero mean level. Besides requiring a coherent square wave, the main downside of this is that the square wave mixing admits all harmonics of returned servo burst signal. Finally (in line (f)) we see that if we are selective about which harmonics we use in the mixing signal, we can remove some of the nonlinear distortion while capturing the best features of the servo signal [213, 212, 211].

We can look at the relative improvement in the different demodulation schemes with some pretty straightforward simulations in which we generate the ideal noise free burst pattern (Figure 7.86) and then add things that distort it. In the simulation leading to the noisy pattern of Figure 7.87, we see that the "Rectify and Integrate" method allows in a lot of broadband noise. Coherently demodulating with a sine wave at the fundamental frequency (labeled Sine Mix) removes most of that dependency (by a factor of 4), and adding in the third and fifth harmonics (labeled Custom Harmonic) improves the performance slightly, as it admits in more true signal, while eliminating the same broadband noise. The effect on the noise admitted into the system can be summarized in Figure 7.88, which plots the computed demodulated noise sigmas versus the input noise sigmas.

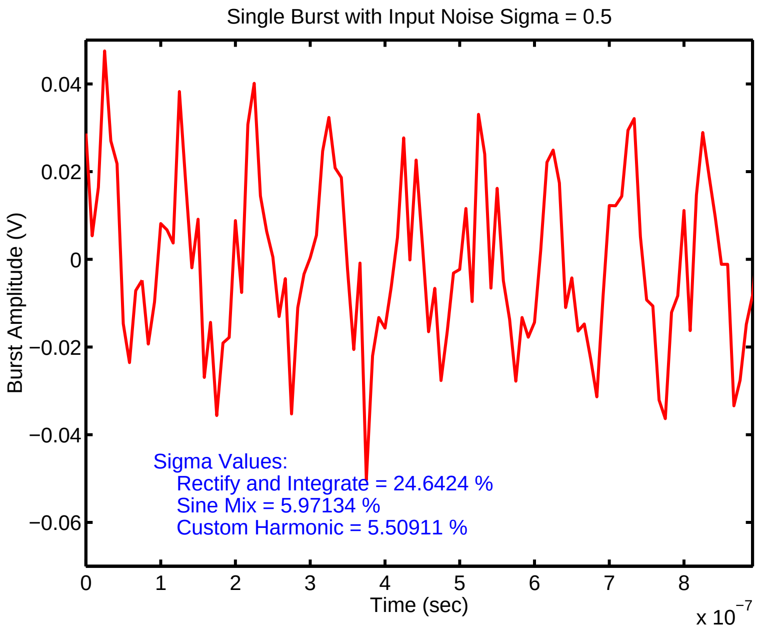


Figure 7.87: Ideal servo burst with significant noise. Note that beyond the fundamental, the signal shape contains third and fifth harmonics. We see how coherent demodulation adds significant immunity to noise in the servo position signal.

However, the advantages of intelligently picking harmonics to use in the demodulation mixing signal go well beyond noise immunity as shown in some of the following simulations. Figure 7.89 shows the effects of a thermal asperity – a heating of the readback head, usually caused by momentary contact or near with some minor bump on the disk surface – on the readback signal. This acts as a temporary offset which strongly affects Rectify and Integrate because this method admits signals at DC. Sine Mix and Custom Harmonic are almost completely immune to this, as they ignore signals at DC.

In the case of a baseline shift, shown in Figure 7.90, the DC level suddenly jumps, causing an offset in return signal which strongly affects anything that includes the baseband signal (in our case Rectify and Integrate). Advanced demodulation can reduce the effects of baseline shift in the burst signal. In this case a large baseline shift (100% of the fundamental signal amplitude) is added in. Again, we can see the effects of different levels of baseline shift on various demodulation scheme sigmas in Figure 7.91.

Another effect that happens to the servo burst is called baseline pop, in which part of the shape of part of the burst signal is altered. An example motivated by discussions with the HP drive team in the 1990s is shown in Figure 7.92. Again, the Sine Mix and Custom Harmonic methods are largely immune tot his.

Another dramatic demonstration of the advantages of being able to pick out "only good harmonics" for

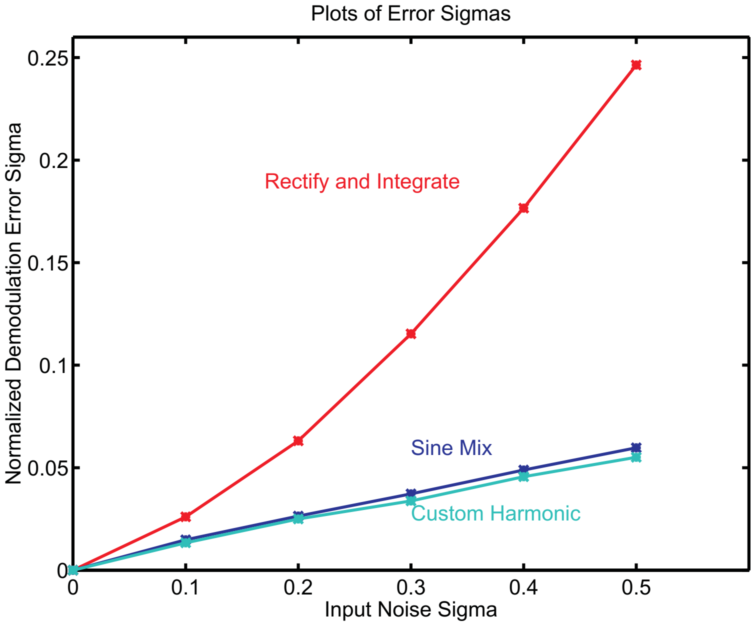


Figure 7.88: Advanced demodulation can reduce the noise of the signal, and therefore minimize the noise circulating in the feedback loop. Using coherent demodulation made up of specific harmonics of servo burst (modeled on HP Cougar I servo signals) dramatically diminishes effects of noise. Rectify and integrate admits far more noise than either demodulating with the first harmonic (sine wave) or multiple matched harmonics.

our mixing signal is shown when a second harmonic is added to the burst signal, as shown in Figure 7.93. Because the second harmonic is even, it corrupts the offtrack push-pull error signal, which should be strictly an odd signal to be most effective. This model was motivated by the disk drive readback heads at the time, where early magneto-resistive sensing (MR) were being used. These showed much higher data sensitivity than the older inductive read heads, but had nonlinear behavior in the offtrack signals. HP had alleviated this to first order with a Dual-Stripe MR (DSMR) head, where two MR sections were flipped relative to each other (thus, the dual stripe) resulting in an offtrack signal that was more linear. However, it still suffered from some nonlinearity manifesting itself in the second harmonic of the readback signal, which got worse as the head was more offset from the center of the magnetic region. The irony is that when the head is at track center, then it is 50% offset from both the A and the B patterns.

Intelligent coherent demodulation dramatically diminishes effects of bad harmonics. In the case of Figure 7.93 a large second harmonic (60% of the fundamental signal) is added in. This strongly affects Rectify and Integrate approach but had no effect on Sine Mix or Custom Coherent since both of these ignore the second harmonic. What surprised a lot of engineers well steeped in signal processing theory was that the Matched Filter approach (which provides the most immunity to AWGN) is clearly not a good choice

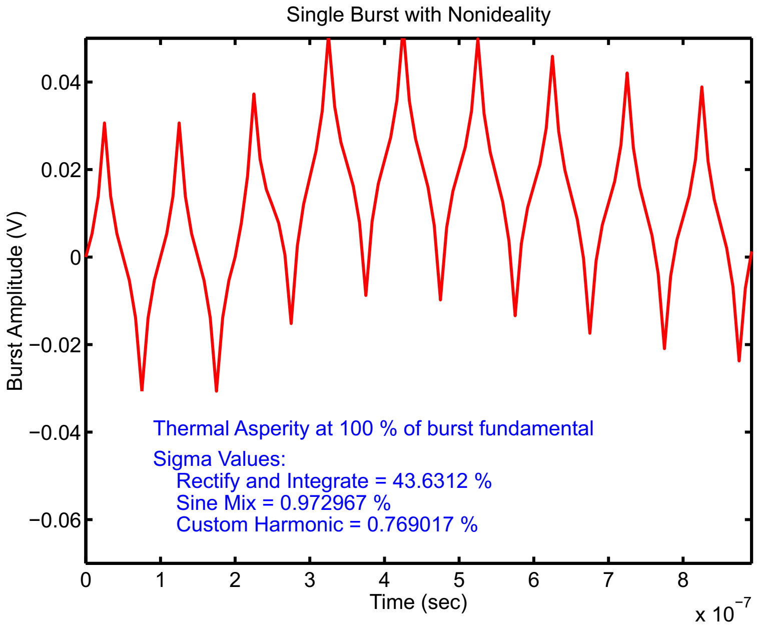


Figure 7.89: Thermal asperity acting on HDD servo burst. A thermal asperity is a heating of the readback head, usually caused by momentary contact or near with some minor bump on the disk surface.

here, since it would include the second harmonic in its mixing signal. The ability to intelligently pick and chose the components of the mixing signal based on the observed physical behavior of the system allows us a much cleaner response. A plot of the error sigmas versus the input amplitudes of the second harmonic (as a fraction of the fundamental) is shown in Figure 7.94. As for the Matched Filter approach, it isn't optimal if it doesn't work.

The point of reviving these old plots is not to revive the HDD industry, but to show that modulating sensed signals can provide immunity to a lot of bad effects, especially if the demodulation is done intelligently. High speed, real-time, digital signal processing algorithms give us the ability to remove a lot of the effects of noise and nonlinear or other non ideal behavior from the signals, thereby leaving our control design far freer to do what it is supposed to do. This particular demodulation scheme was able to rely on the timing being locked due to the presence of the sync field at the start of the servo burst. When this is not the case, we need to use some form of precision lock-in demodulation, as described in Section 7.19. If we can combine this with a PLL to minimize the phase difference, then the in-phase (I) signal can trivially extract the signal amplitude.

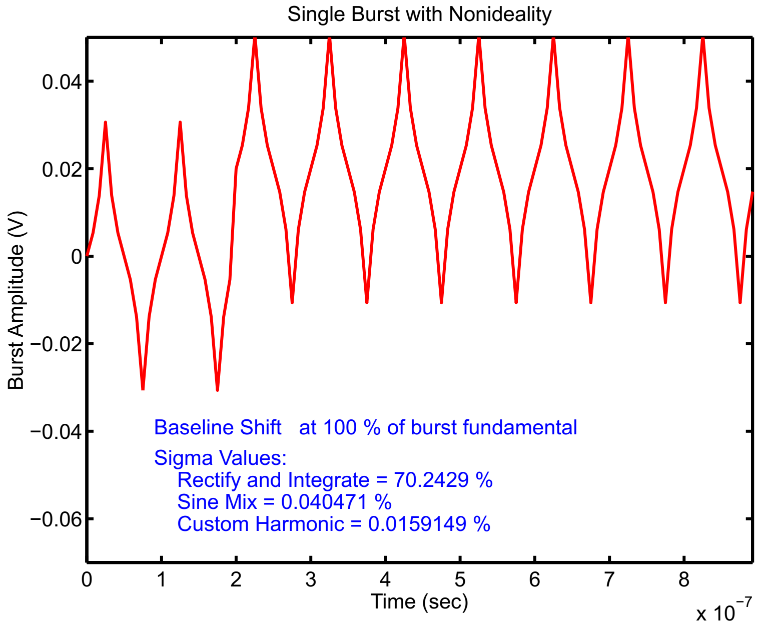


Figure 7.90: Ideal servo burst with significant baseline shift added. A baseline shift is essentially noise added at DC.

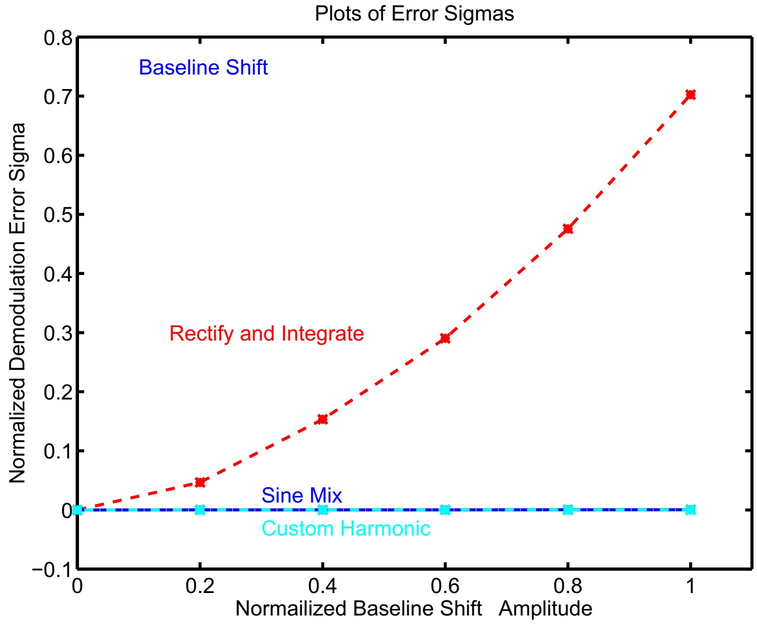


Figure 7.91: Advanced demodulation can reduce the effects of baseline shift in the burst signal. In this case a large baseline shift (100% of the fundamental signal amplitude) is added in. This strongly affects Rectify and Integrate demodulation.

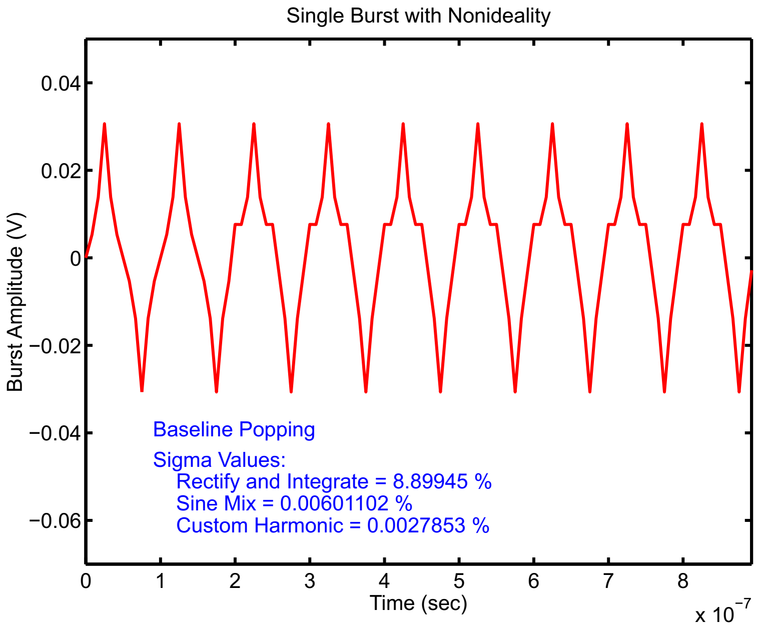


Figure 7.92: Ideal servo burst with significant baseline pop added. A baseline pop changes the shape of one polarity of the burst.

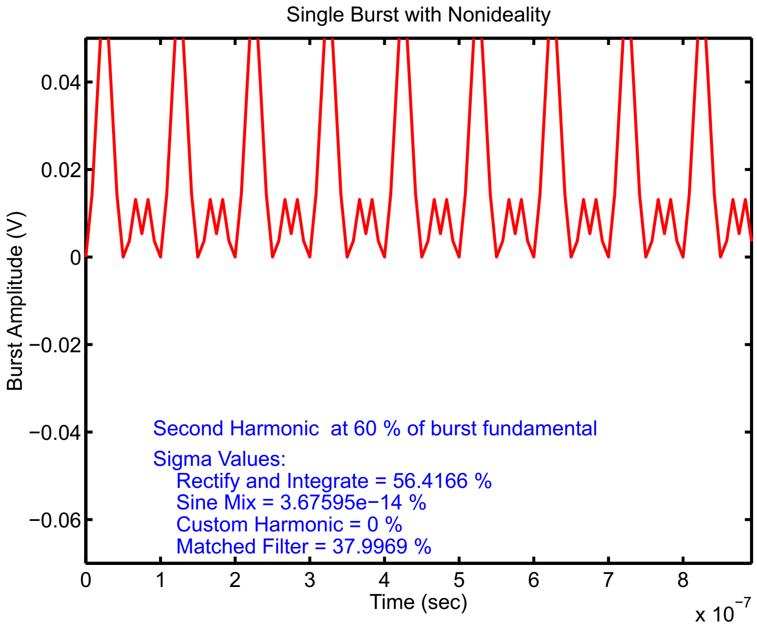


Figure 7.93: Ideal servo burst with significant second harmonic added. Because the second harmonic is even, it corrupts the offtrack push-pull error signal.

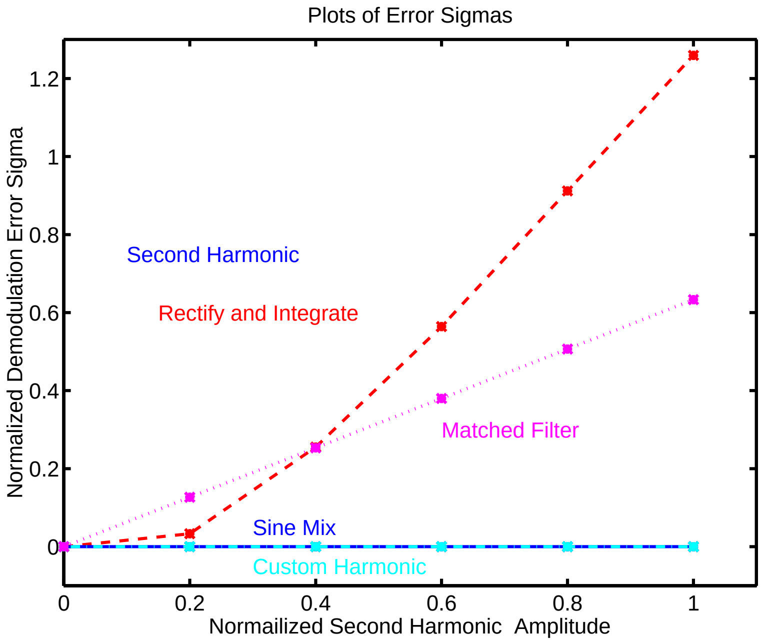


Figure 7.94: Advanced demodulation can reduce the effects of bad harmonics in the burst signal. In this case a large second harmonic (60% of the fundamental signal) is added in. This strongly affects Rectify and Integrate and a Matched Filter approach, as the latter would include all harmonics in the mixing signal.

## 7.22 Example: Optical Disk Precision Clocking: DVD+RW

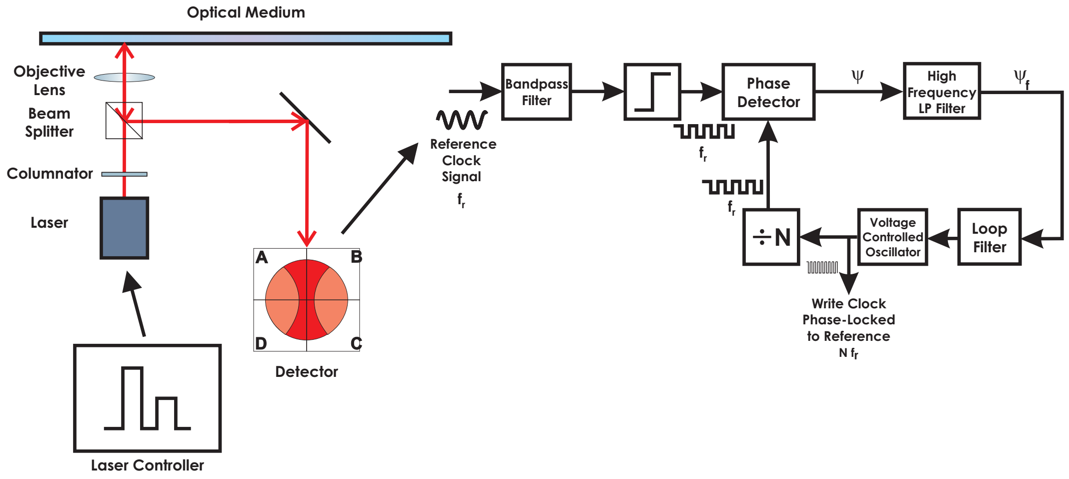


Figure 7.95: Rewritable optical disk drive system with a Harmonic Locking PLL to generate the write clock [264].

On optical drives such as CDs and DVDs, there are two main servo loops, one to maintain focus of the optical beam and the second to keep that beam over the correct track. However, because the tracks are distinguished by physical features on the disk, the tracking loop has the potential for much higher sample rates than are found in the HDD loops described in Section 7.21. In these problems the more difficult problem is that of establishing sub-bit accurate timing down the track for making rewritable optical disks that are compatible with the ubiquitous DVD ROM format. The latter has none of the edit gaps mentioned in the discussion of HDDs in Section 7.21, and so being able to write new data in the correct down-the-track location requires precise synchronization with sub-bit accuracy.

The essential technology is a high frequency, high fidelity reference signal embedded into the disk surface itself, as illustrated in Figure 7.96. With this so-called high frequency wobble, a digital phase-locked loop can correct the timing to enable read/write operations without edit gaps. A gapless edit of a 6T pattern into a 4T-8T pattern is shown in Figures 7.97–7.99. The lack of jumps in phase errors in a clock derived from the data (Figure 7.98) as well as the lack of any 5T or 7T patterns in the histograms of Figure 7.99 indicate a bit perfect edit, where T is the bit clock period. The continuous nature of the high fidelity reference signal simplified the PLL design which enabled the DVD+RW (and DVD+R) formats [264, 263, 265, 266]. By having a continuous, high fidelity reference clock signal across the entire length of the track, the internal timing of the read and write circuits was within fractions of a bit (allowing the desired read/write functionality with no edit gaps. By the year 2013 the DVD+RW and DVD+R businesses enabled by this amounted to 25% of a $559M market (according to a website called Storage.com which now has been re-purposed for finding self-storage units).

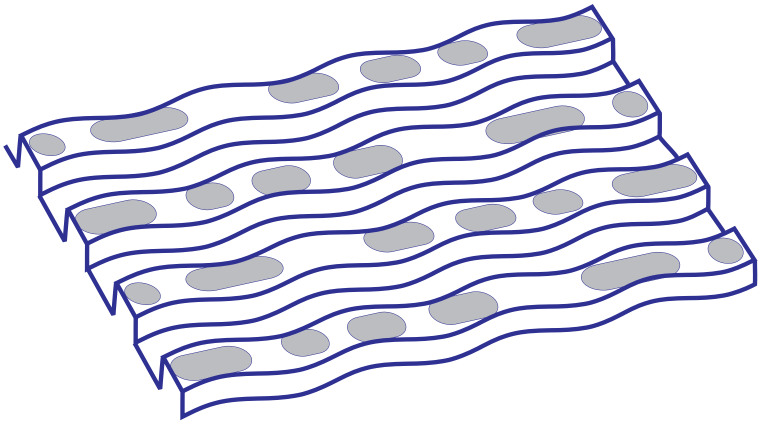


Figure 7.96: High frequency wobbles used in the DVD+RW optical disk format.

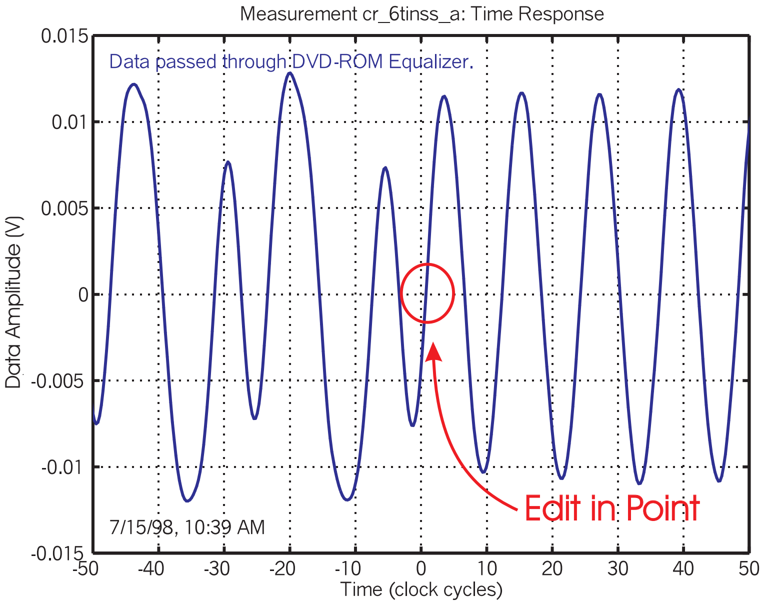


Figure 7.97: DVD+RW, gapless edit. A 6T pattern spliced into a 4T-8T pattern. This represents the time response at the edit-in point.

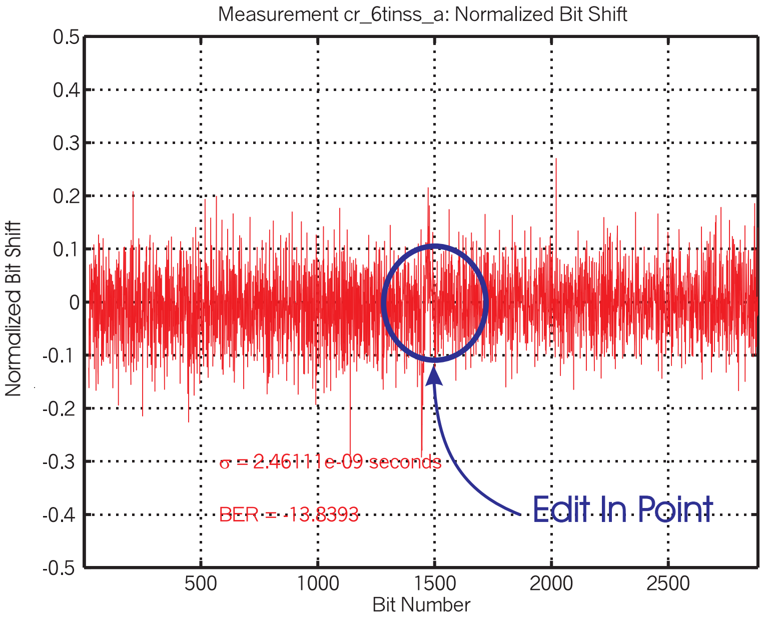


Figure 7.98: DVD+RW, gapless edit. A 6T pattern spliced into a 4T-8T pattern. This represents the phase error for a data clock generated from the data. (Note the absence of any phase jumps.)

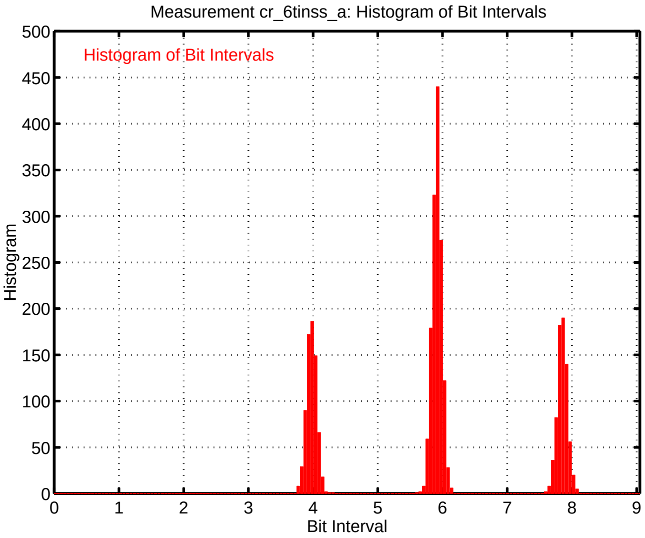


Figure 7.99: DVD+RW, gapless edit. This is a set of histograms of the bit intervals. The absence of any 5T or 7T bits is an indication that no bit errors have occurred.

## 7.23 Example: Laser Interferometry

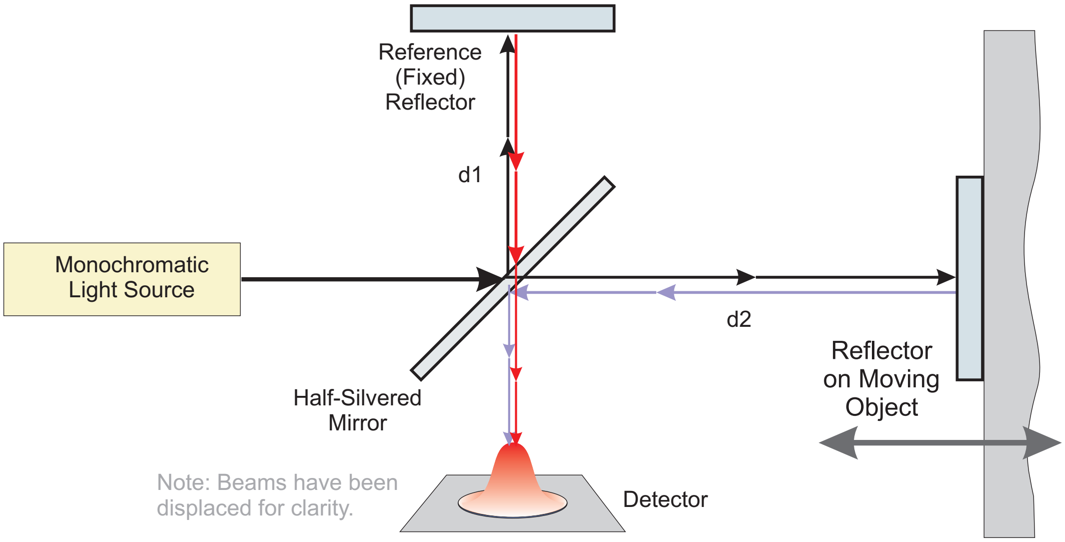


Figure 7.100: Basic Michelson interferometer.

This section is largely excerpted from the author's **A Tutorial on Laser Interferometry for Precision Measurements** [207]. The full equation derivations are in that reference, but a few key points will be used here to describe laser interferometers in the context of demodulation systems. The operation of an interferometer depends upon optics performing some of the same equations that we have been discussing above. It is in understanding the relationship of the optics equations that we see that the optics are performing a demodulation of two signals at the same frequency interfering with each other and therefore producing a phase difference that we can detect with something akin to a Costas loop.

The basic Michelson interferometer (Figure 7.100) uses a half silvered mirror to split a monochromatic light source into two beams. Each beam reflects off of a mirror, to be recombined at the half-silvered mirror. The recombined beam contains an interference pattern that changes when either of the mirrors move. Keeping one mirror fixed allows one to attribute all of the interference pattern changes to motion of the other mirror.

Many texts show the an interference pattern such as the one in the far left of Figure 7.101. However, in the absence of the beam being cropped, the detector will see a collimated beam on its center axis. Typically, this is modeled as a Gaussian beam and as the measure mirror moves the intensity of this pulse will vary, as shown in the right three figures of Figure 7.101. The detector then acts to integrate the

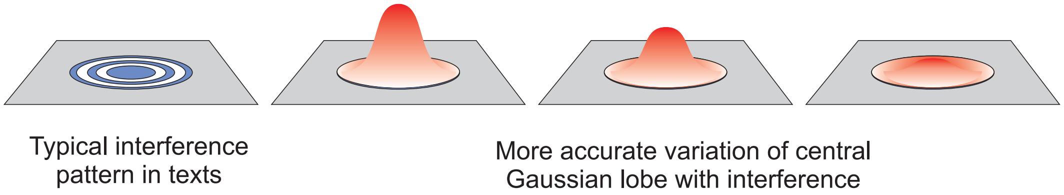

Typical interference pattern in texts
More accurate variation of central Gaussian lobe with interference

Figure 7.101: Effects of interference on detector. On the far left is the typical diagram one sees in books. However, the banding is typically caused by the beam being cropped and not the effect of the interferometry. A better picture comes from the three diagrams on the right, in which the intensity of the central Gaussian lobe is modulated by the interference pattern.

intensity of the beam over its spatial extent, and – assuming the integration is faster than the change in the interference pattern – this integrated intensity can be used to measure distance, modulo the wavelength of the laser used.

The Michelson interferometer is one of the most basic models of interferometry available. It is not a practical interferometer, in that there are significant issues with the actual implementation. However, it provides an easy to understand conceptual model for understanding precision measurement interferometers. Generally, for every imperfection of the Michelson interferometer, there is a practical fix that expands the range of usefulness of the interferometer [207]. Each of these fixes essentially returns the interferometer back to a more ideal Michelson behavior.

We start our analysis of the Michelson IF equations by looking at Figure 7.102. For our purposes, the source beam can be considered to originate at position 1, right before contact with the half silvered mirror. At the mirror, half of the beam is reflected to the reference mirror (path r2-r3-r4) where it is reflected back towards the half silvered mirror. At this interface, half of the beam is passed through to position 5, while half reflects back to the source. Meanwhile, the transmitted portion of the beam goes to the measurement mirror (path m2-m3-m4) and reflects back. At the half silvered mirror, half of the measure beam is reflected to position 5, while half passes back to the source. We are concerned with the two beams that meet at position 5 and are imaged on the detector.

A few things are important to understand interference as it is used in our measurements. First, since both the reference beam and the measure beam originate from the same laser, they are coherent with each other. Second, every time a beam goes through a reflection, it undergoes a $180^\circ$ phase shift. A look at the diagram of Figure 7.102 indicates that each beam at position 5 has gone through $360^\circ$ in phase shifts and thus they are still in phase with each other. Third, by the time both beams reach position 5, their amplitude has been reduced to $\frac{1}{4}$ of their original amplitude. (This is fixed in practical

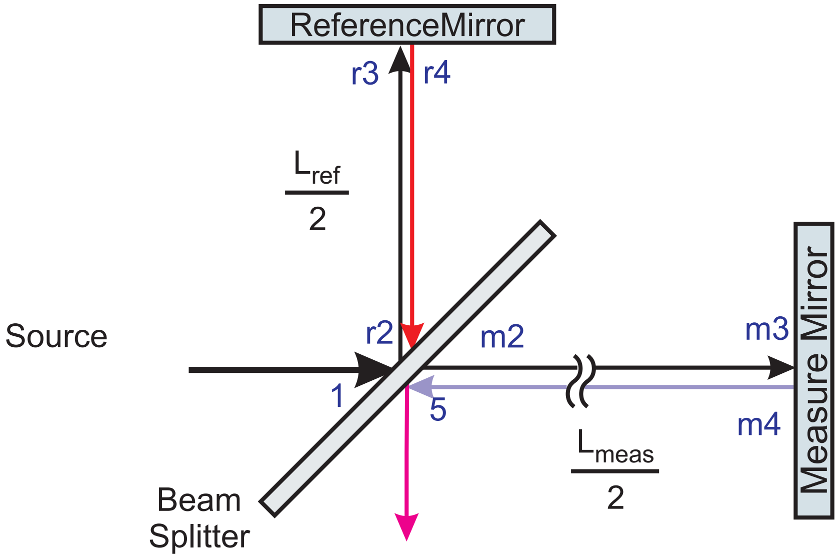


Figure 7.102: Some details on the beams of a Michelson interferometer.

interferometers via use of polarizing beam splitters, quarter wave plates, corner cubes, and a second frequency so that a far greater percentage of the beam power hits the detector [207].) Conceptually, if the reflection/transmission is exactly 50/50 and if the mirrors are perfectly aligned, then both beams add through linear superposition and have the same amplitude. Thus, we can attribute the variation at the detector to interference.

The equations for the interference pattern are derived in classic optical texts [267], [268] from application of the vector electromagnetic wave equations [269, 270]. Consider the electric field of the source beam at position 1:

$$E_{z,source}(z,t) = A\cos(kz - \omega t + \phi) \tag{7.152}$$

where $z$ is the direction of travel, $k = \frac{2\pi}{\lambda}$ is the wave number, $\lambda$ is the wavelength of the light, and $A$ is the amplitude of the beam. From position 1, the reference beam travels a distance $L_{ref} = 2d_1$ to get back to position 5, while the measure beam travels a distance $L_{meas} = 2d_2$ to get back to position 5. If we consider position 1 to be $z = 0$, then the two beams are thus,

$$E_{ref}(t) = \frac{A}{4}\cos(kL_{ref} - \omega t + \phi) \tag{7.153}$$

for the reference beam and

$$E_{meas}(t) = \frac{A}{4}\cos(kL_{meas} - \omega t + \phi) \tag{7.154}$$

for the measure beam. Through linear superposition, the beams add, so that the electric field of the combined beams at position 5 is

$$E_{tot}(t) = \frac{A}{4}\left[\cos(kL_{ref} - \omega t + \phi) + \cos(kL_{meas} - \omega t + \phi)\right]. \tag{7.155}$$

A word about notation is useful here. Normally, when one is working with wave equations [269, 270], the equations are set up as vector quantities along some frame of reference. This works very well in analysis of plane equations, point sources, etc. but in an interferometer, the direction of the beams are switched so often that keeping track of all the vector frames becomes confusing. For this tutorial, we will assume that the source electric field is in the X-Y plane, and the source magnetic field is rotated $90^\circ$ in that plane. This means that the Poynting vector which describes the energy density is in the Z direction. Every time we go through a reflection, polarizer, or beam splitter, the reference frame is changed, but our signals will end up so that the Poynting vector is normal to the detector plane. For the sake of simplicity, we will leave off the unit vector designations on the equations.

The detector is sensitive to signal intensity, not amplitude, and we can calculate this from the Poynting vector. If we assume that the electric field is in the $x$ direction and the magnetic field is in the $y$ direction, then

$$H_{tot}(t) = \frac{A}{4}\sqrt{\frac{\epsilon}{\mu}}\left[\cos(kL_{ref} - \omega t + \phi) + \cos(kL_{meas} - \omega t + \phi)\right]. \tag{7.156}$$

We now have two choices to simplify this: proceed with trigonometric identities or switch gears to saying that Equations 7.155 and 7.156 are the real parts of a complex exponential notation. For pedagogical purposes, we will plug through the trigonometric equations here. With the polarizations we have assumed, the Poynting vector, $\mathcal{P}(t)$, will be in the direction normal to the detector with

$$\begin{aligned}
\mathcal{P}_{tot}(t) &= \vec{E}_{tot}(t) \times \vec{H}_{tot}(t) \\
&= \frac{A^2}{16}\sqrt{\frac{\epsilon}{\mu}}\left[\cos\alpha + \cos\beta\right]^2 && (7.157) \\
&= \frac{A^2}{4}\sqrt{\frac{\epsilon}{\mu}}\left[\cos^2\alpha + \cos^2\beta + 2\cos\alpha\cos\beta\right] && (7.158)
\end{aligned}$$

where $\alpha = kL_{ref} - \omega t + \phi$ and $\beta = kL_{meas} - \omega t + \phi$. With this and some trigonometric identities, we end up with

$$\cos^2\alpha = \frac{1 + \cos 2(kL_{ref} - \omega t + \phi)}{2}, \tag{7.159}$$

$$\cos^2\beta = \frac{1 + \cos 2(kL_{meas} - \omega t + \phi)}{2}, \text{ and} \tag{7.160}$$

$$\begin{aligned}
2\cos\alpha\cos\beta &= \cos\left(k(L_{meas} + L_{ref}) - 2\omega t + 2\phi\right) \\
&\quad \times \cos\left(k(L_{meas} - L_{ref})\right).
\end{aligned} \tag{7.161}$$

Putting these all together,

$$\begin{aligned}\mathcal{P}_{tot}(t) &= \frac{A^2}{32}\sqrt{\frac{\epsilon}{\mu}}\Big[1+\cos 2(kL_{ref}-\omega t+\phi)+1+\cos 2(kL_{meas}-\omega t+\phi)\\ &\quad +2\cos\Big(k(L_{meas}+L_{ref})-2\omega t+2\phi\Big)+2\cos\Big(k(L_{meas}-L_{ref})\Big)\Big] \qquad (7.162)\end{aligned}$$

If we average over an integer number of periods, $T=\frac{1}{f}=\frac{2\pi}{\omega}$ then the time varying portion integrates out, leaving only the DC portion:

$$\mathcal{P}_{tot,avg} = \frac{A^2}{16}\sqrt{\frac{\epsilon}{\mu}}\Big[1+\cos\Big(k(L_{meas}-L_{ref})\Big)\Big] \qquad (7.163)$$

This rationale should remind the reader of the high precision IQ demodulator of Section 7.19. As a practical matter, the laser frequency is so much faster than the integration time of our detector that we are always getting the "DC portion". Thus, the relationship that is most commonly used for this type of interferometer is that for the intensity:

$$I \sim K\Big[1+\cos\Big(k(L_{meas}-L_{ref})\Big)\Big] \quad W/m^2 \qquad (7.164)$$

This is often rewritten in terms of the wavelength, $\lambda$, as

$$I \sim K\Big[1+\cos\Big(\frac{2\pi}{\lambda}\Big(L_{meas}-L_{ref}\Big)\Big)\Big] \quad W/m^2 \qquad (7.165)$$

This provides the power density at the detector in $Watts/m^2$. The detector integrates the energy density (intensity) over the detector surface. Thus, it is not the pattern on the surface that matters so much as the amount of intensity on that surface. Equation 7.164 gives the density at a given point. In fact for a highly collimated beam, the distribution is likely Gaussian and effect of a change in $L_{meas}$ is to cause the height of this Gaussian distribution to rise and fall. If all other variables are held constant, one can measure a change in distance by counting the passing of these light and dark times. From Equation 7.164 we see that we are still missing an ability to discern direction of motion. This is fixed in a single frequency IF by splitting the the beam and adding a phase delay to one portion, thus allowing for in-phase and quadrature demodulation (IQ), again reminiscent of Section 7.19.

We start with the reminder that all position measurements with an interferometer are relative. The fringes give a change in position from some starting position. This is analogous to trying to measure position from velocity measurements: one must assume a starting position.

IF measurements rely on knowing the wavelength of light, $\lambda$, and the wavelength being stable. This is why commercial interferometers did not emerge until lasers were invented. It is important to know that as pressure, temperature, humidity,and gas composition change, so does $\lambda$. Thus, an IF system is making measurements with a somewhat elastic ruler.

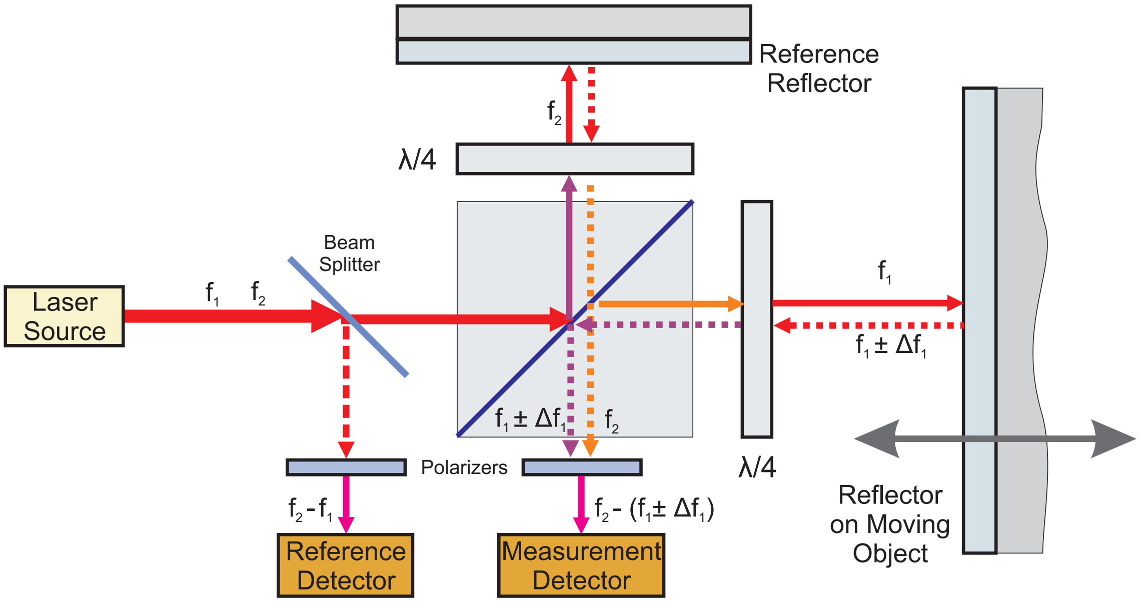


Figure 7.103: Two frequency (heterodyne) Michelson interferometer configuration.

All the modifications to the Michelson interferometer discussed thus far essentially are designed to desensitize the interferometer to non-ideal behavior and restore the accuracy of Equation 7.165. However, even when things are properly aligned, the interferometers described so far operate in the baseband. They use a single frequency of light, also known as homodyne interferometry, and the “difference” between measure and reference only shows up as a baseband phase and Equation 7.165 is a variation away from DC. DC detection is slow and suffers from $\frac{1}{f}$ and other noise in the detectors, mainly signal intensity variations (due to air turbulence or accumulated contaminants on mirror and optic surfaces) being indistinguishable from position changes.

Borrowing from the world of radio communications, it is more advantageous if the interference shows up at some intermediate frequency. To achieve this, modern IF measurement systems typically operate with multiple wavelengths [271], where the interference pattern is not a baseband signal, but in fact an AC signal, as diagrammed in Figure 7.103. Thus, distance becomes a measurement of the difference between two signals, one of which (known as the measurement signal) is modulated by the moving object, while the other (known as the reference signal) is generally fixed. The reference signal is usually composed of the difference between the two frequencies before one of them has been modulated, but can also be another modulated signal to create a differential measurement between two moving mirrors.

Assuming the two laser frequencies are $\omega_1$ and $\omega_2$, we get the equation for the Poynting vector [207]. If the measurement mirror is moving, then that movement will appear as a Doppler Shift in $\omega_1 = 2\pi f_1$, so that $\omega_1 \Longrightarrow \omega_1 + \Delta\omega_1$ becomes:

$$\vec{E}_{IF,LP} \times \vec{H}_{IF,LP} = \mathcal{P}_{IF,LP}(t) \approx \frac{A^2}{2}\sqrt{\frac{\epsilon}{\mu}}\left[1 + \cos\left(k_1 L_{meas} - k_2 L_{ref2} - (\omega_1 + \Delta\omega_1 - \omega_2)t\right)\right]. \tag{7.166}$$

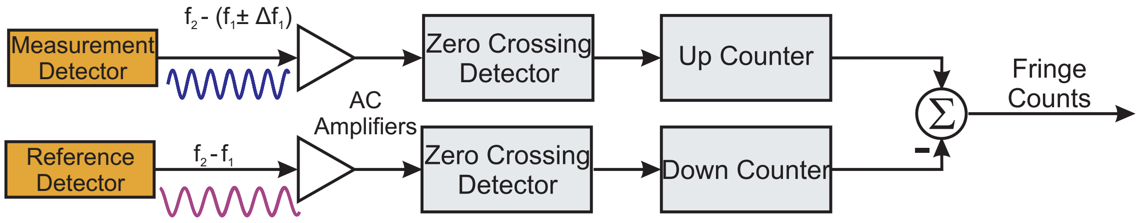


Figure 7.104: Generating distance from AC frequency differences.

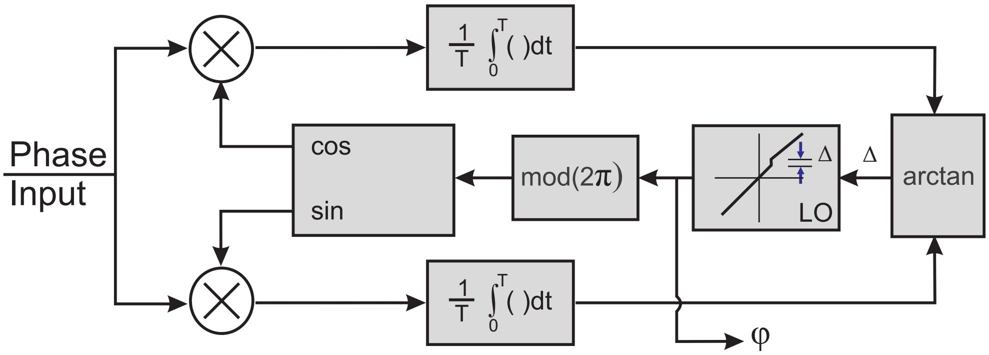


Figure 7.105: Phase generation from interference pattern input. Note the structural similarity to the Costas loop of Figure 7.53.

In the end, the optics have generated an intensity that is proportional to a $1 + cos(\theta)$ where $\theta$ is periodic and related to the distance between the reference reflector and the moving reflector. The fundamental accuracy is inversely proportional to the laser wavelength ($\lambda$).

In Equation 7.166, $\theta$ becomes a shift in the frequency difference between the two signals, and results in new oscillatory signal that once again needs to be demodulated. To be an instrument for measuring distance and velocity we need devices for counting either peaks in intensity (a peak finding demodulator, Figure 7.104) or to accurately and quickly measure the phase of the intensity signal (Figure 7.105). Note how Figure 7.105 closely matches the Costas loop of Figure 7.53. The peak counter (peak finding demodulator) has an accuracy limited to half the wavelength, while the IQ demodulator is capable of much finer resolution.

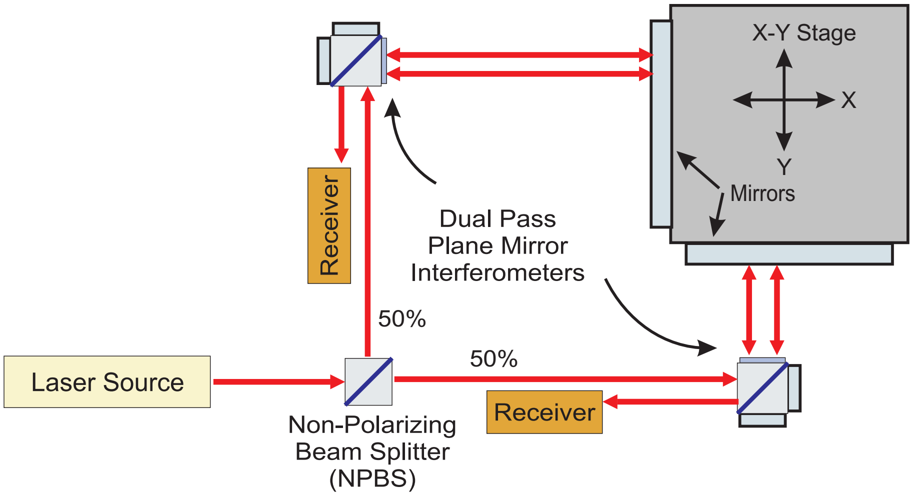


Figure 7.106: Two-axis plane mirror interferometer configuration.

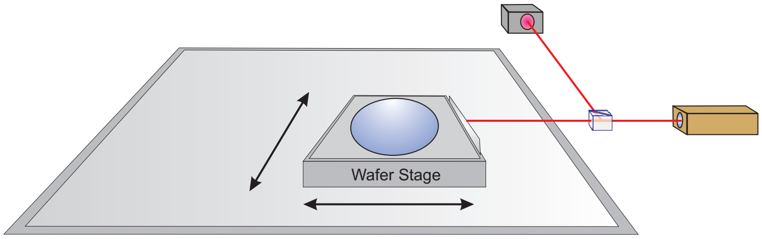


Figure 7.107: Wafer stage system measured with interferometer.

One of the great benefits of precision interferometry for position measurement is that because the measurements are done at a distance, multiple axes can be measured with the same system, by splitting the laser beam and directing it off of different surfaces and back to multiple receivers. This can be seen in the two-axis configuration shown in Figure 7.106, where a single beam is split and directed at polarizing beam splitter based interferometers. Each of these beams is reflected off of a planar mirrors on the side of a moving stage, resulting in position measurements for the x and y axes.

The bulk of interferometers systems are used in the IC photolithography industry [272, 273]. Here, very precise machines move an X-Y stage under an optical column. What is critical is the location of the stage relative to the optical column, and the repeatability of this measurement. So these systems use laser interferometers to measure the X and Y positions of the stage and the column, as well as the pitch and yaw of these items. Some even measure the vertical direction of the stage. Figure 7.107 shows the basic setup without the optical column which would obscure the stage.

This section has gone through more math than most of the others combined, but it shows that at the end, to a clean position and velocity signal, we again need to understand modulation and demodulation. A huge amount of the improvement in the interferometry measurements are made by an understanding of the optical paths, but at the end, the final bit of accuracy is limited by signal conversion done in the electronics. Mixing (multiplying) and digitally integrating/low-pass filtering a signal usually requires at least 10 samples per second. If we consider the upper range of FPGA fabric circuitry to be around 500 MHz, then we can consider processing a 50 MHz signal, but only if we use only one clock cycle per sample. More reasonably, we might expect a heavily pipelined algorithm to have 10 cycles of signals up to 5 MHz. Faster signals would require either custom signal processing circuitry or analog processing for the front end.

## 7.24 Demodulation Summary

Modulated signals are rarely considered a main part of control design. It is rare that the carrier itself has significant information for the loop. It is probably for this reason that this subject is not well studied by control engineers. However, the form of modulation, and the method by which we demodulate it can have a significant effect on the quality of the demodulated signal returned to the loop.

This tutorial has given a brief, non-exhaustive overview of demodulation methods for modulated signals found in control applications. The emphasis was placed on a conceptual understanding, but with a view to how to execute the different demodulation computations. There may well be a wide variety of cases for which the simplest modulation/demodulation – such as amplitude or pulse encoding demodulated with a simple rectifier and low-pass filter are sufficient. This is really a requirements question: when the speed and accuracy of the demodulation scheme is an order of magnitude above the needs of the loop, there is no reason to do anything more sophisticated. However, in understanding the more advanced methods, we have the option to speed and clean up the sensor signals before they get into the feedback loop, thus avoiding the limitations imposed by Bode's Integral Theorem [1, 158].

The demodulation methods we see often have a lot of commonality. An understanding of Fourier integrals is extremely helpful, but to make them practical, we have to seriously look at the information content of the signals and the computational structures needed to extract them. The payoff here is the potential to dramatically lower the sensor noise (and nonlinearities) injected into a loop, as well as dramatically speeding up the acquisition of the signal. As we know from our earliest controls principles that sensor noise goes right through to the output [13, 14], the extra work has a direct payoff.

## 7.25 Chapter Summary

Bode's Integral Theorem, as illustrated by Stein's Dirt Digging, tells us that noise we see at measurement points is really closed-loop filtered noise from sources around the loop. To understand which of these is most influential on our loop, we need some sort of regular methodology for measuring, isolating, and analyzing these noise inputs.

- PES Pareto gives us a way (under "mostly linear" assumptions) to measure, isolate, and quantify noises as inputs, and for their effects (relative and cumulative) at some measurement point.

- Measurements and models used to create Pareto can be used to extrapolate.

Once we have quantified the key sources of broadband noise, we can look for simple design changes that might limit this source or we can see if some improved processing can reduce it. In the former case and our disk drive example, redesigning the air flow inside the drive itself was seen to dramatically change the buffeting that the head experienced and therefore eliminate much of this input disturbance. In the latter case, applying coherent demodulation to the signal could dramatically reduce the effects of both broadband noise and nonlinear signal behavior. The cost of this type of work was doing more sophisticated math closer to the physical system.

In a separate discussion, the linear filtering done on our input signals also can have a dramatic effect on the latency and noise that enters the loop. Our noise will be shaped by our closed-loop, loop shaping filters (for better or worse) but there is no way to algorithmically remove latency once it has been inserted. Understanding the noise inputs and effects through a system and minimizing input noise and latency can often provide a much larger boost in achievable performance than any particular loop design improvement.

# Chapter 8

# Integrating in Feedforward Control

## 8.1 In This Chapter

One of the more particular features of feedforward control is that it has so many different meanings, depending upon which control engineer is using the term. In this chapter, we will discuss a wide variety of classes of control that all get refereed to as feedforward control. In this chapter, we will try to tie together as many of these various modes of "feedforward" control in an understandable way. Whether the term refers to adjusting the filters from the reference input into the closed-loop or the plant, or applying some sort of repetitive or harmonic correcting method, or using auxiliary sensors, feedforward control has three universal features.

- They work with far less phase delay than classic feedback loops allowing them to be far more aggressive in their region of operation.
- Noise amplification by the loop rarely affects the feedforward signal, again allowing for more aggressive control.
- They depend upon accurate modeling to work properly, but that modeling can be limited to only the range of the feedforward signal being used.

## 8.2 Chapter Introduction

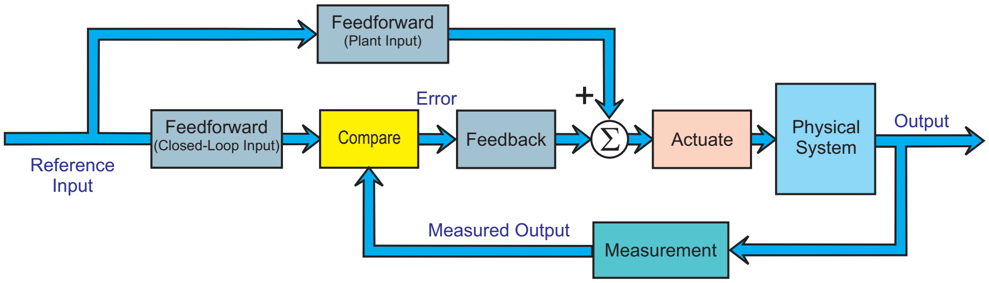


Figure 8.1: A generic feedforward/feedback loop configuration.

Feedforward control – especially as combined with feedback in feedforward-feedback controllers, has enjoyed a resurgence in the past three decades. The seemingly less rigorous problem – compared to feedback control only – found its way into many practical systems and then found itself into many official algorithms. In the first version of this chapter, I was trying to simply provide unifying looks at two forms of reference signal feedforward control, ones that inject the shaped reference at the plant input and ones that inject the shaped reference at the closed-loop input. However, I kept remembering different "feedforward" versions that deserved their own sections. Eventually, as the sections piled up like points made by Inspector Colombo before he needs to leave, I realized that I needed to take a step back and ask myself what feedforward control really is. With that in mind, my understanding is as follows:

When the controller's input signal comes from sensing the signal to be controlled, then we call it feedback control. When the controller's input signal comes from any other signal, be it another sensor, or the reference signal, or some heavily processed average of the controlled signal, then we call it feedforward. In the modern world, we end up with systems that have some feedback signal(s) and possibly one or more feedforward signals and that has become a thing.

The final overall question to answer is: Why is it a thing? What do we get from adding feedforward control to our feedback systems that we did not have before? Here is where adding all those sections actually paid off, because if one looks for a common advantage in all of these versions of feedforward control, it is that feedforward controllers allow us to apply control with minimum of delay, and as I've tried to point out in previous chapters, delay equals negative phase and negative phase erodes stability margins. Feedforward from auxiliary sensors often has far less delay that waiting for a disturbance to cause a loop error. In many cases, feedforward allows us to use acausal filtering because we "know the future" of that signal. We know what the reference signal should be ahead of time. When we extract repetitive components from an error signal, we can look forward or back on the cycle and so we can literally look back to the future for our control signal. The repetitive controller then becomes like a causal/acausal FIR filter with no net phase effects, which is pretty amazing. Phase lag is something we cannot eliminate in a feedback system, but if we can shift more of the control responsibility to the feedforward portion of the control system, we can gain some significant benefits. **The ability to work with less or no time delay for a significant fraction of the control actions is why I believe feedforward control is a thing and why I believe that it has worked so robustly in practical systems.**

## 8.3 What Do We Mean When We Say “Feedforward Control”?

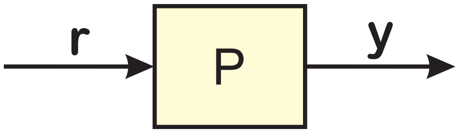


Figure 8.2: Plant only control. Generally, we want the output, $y$, of the plant (physical system) to follow the reference, $r$.

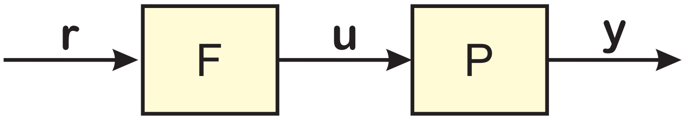


Figure 8.3: Feedforward only control. Now, we’ve added a prefilter to shape the reference, $r$, so that the output, $y$, can more easily follow it.

What is feedforward control? In some sense, it is what we designed before anyone though of using feedback. If we had a physical system denoted by $P$ in Figure 8.2, we would generally want to direct it to do something by giving it a push (a reference signal, $r$) and hoping that the measured output of that physical system, $y$ would some how have a relation to $r$. Often the ideal was that $y$ would exactly equal $r$. Clever people realized that if we shaped the input with some sort of filtering, denoted by $F$ in Figure 8.3, we might get a better result. In particular if we could set $F = P^{-1}$, then we would have $y = r$ and could go home.

At this point, the main issues were sensitivity to parameter variations, disturbance rejection, and the inability to truly invert $P$ for all inputs, conditions, and frequencies. Feedback, as simply diagrammed in Figure 8.4, allowed us to become less sensitive to plant variations and to disturbances, but the design

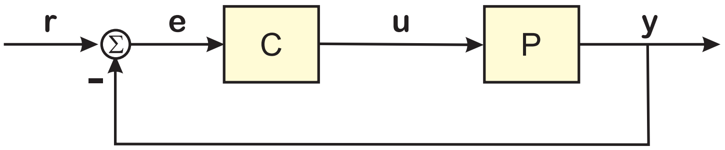


Figure 8.4: Basic feedback control often ignores the possibility of using feedforward.

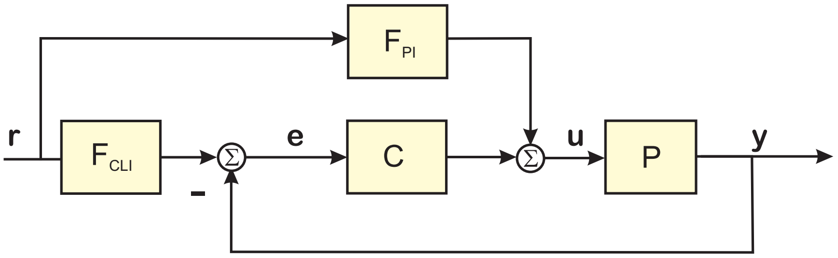


Figure 8.5: A more generic, but basic diagram of possible feedforward/feedback loop configurations. Here we see the two most likely feedforward blocks, so-called plant input (PI) and closed-loop input (CLI).

energy spent designing $C$ such that $y = \frac{PC}{1+PC}r$ behaved well almost seemed to relegate feedforward to the less of the two methods, often ignored in theoretical work – but finding many ad-hoc implementations in industry.

Since the late 1980s, feedforward has regained it's swagger in the academic/theoretical circles. Many of these methods have been formalizations of methods that had proven successful, reliable, and beneficial in practice. It's the model I love of **find a solution to the problem, then figure out if and why the solution works in general.**

So, what is feedforward control today? Well it is almost always used in conjunction with feedback control, hence the term: feedforward-feedback control. It generally means control involving a signal that is not from the sensor(s) used for the feedback portion. That is, we sense the controlled output and use it in the control, that's feedback, but if the signal is not the controlled signal, even if it's from a sensor, then it's feedforward. This generally holds so long as one allows in repetitive controllers and adaptive feedforward cancelers that use a quantity derived from the feedback signal (e.g. one or more repetitive components that have been isolated) to calibrate an add-on controller that feeds forward a canceling signal independently of the work of the feedback controller, $C$.

Looking at the generic feedforward-feedback controller diagram in Figure 8.5 we can describe many of the feedforward structures.

- From reference to the plant input, $(F_{PI})$, circumventing the feedback controller in the forward path.
- From the reference to the closed-loop input, $(F_{CLI})$, reshaping the reference to effectively incresase input-output bandwidth without amplifying sensor noise.
- From the reference to the closed-loop input, $(F_{CLI})$, reshaping the reference to remove certain poor components that adversely affect the closed-loop response (input shaping).
- From some repetitive component of the error signal, e.g. repetitive control (RC) or adaptive feedforward cancellation (AFC).
- From an auxiliary sensor, often used to detect and remove a disturbance before the feedback loop has to deal with it. (Noise canceling headphones do this, although the headphones themselves are not in a feedback loop.)

Now, a high bandwidth signal from any sensor can have noise while our reference signals are largely considered "noise free". High bandwidth feedback loops can amplify noise (see Bode's Integral Theorem) but as there is "none" in our reference that's not an issue for feedforward. Even if the signal comes from an auxiliary sensor, the insertion point in the loop and thereby the noise amplification is different (see PES Pareto ). Even with repetitive control or adaptive feedforward cancellation, the repetitive component is estimated via a slow averaging process, dramatically lowering the noise in the repetitive signal used by the feedforward portion of the control loop.

Finally, we could add iterative learning control here, in which a new reference path is recomputed at each time step based on the error results of the prior steps, but that is for another day or chapter.

The structure of this chapter is as follows:

- We start with the basic concepts of feedforward control in Section8.4.
- Section 8.5 describes measurements we can make on combined feedforward/feedback systems to characterize them for control.

-

- Section 8.6 starts with a practical example of input shaping feedforward control.

- Input Shaping Feedforward (Section 8.7) is simply a prefilter that can be thought of as either shaping or removing input signal components or can be thought of as adjusting the input output closed-loop transfer function of a system without broadening the feedback bandwidth.

- Another example of what is called feedforward involves Repetitive Control (RC) or Repetitive Feedforward Control, discussed in Section 8.8. In this case, a repetitive portion of the error signal is isolated – typically using some sort of learning or adaptation – and this is used in feedforward to cancel out the repetitive error. Even though the correction signal is calibrated using old data, it can be adjusted to minimize the error, and thus acts with little or no phase lag (for the repetitive component).

- Another form of feedforward control involves using auxiliary sensors to augment the feedback loop with some feedforward prediction. In this context, a filter is tuned to help predict a disturbance (by sensing it with the sensor) and then add a calibrated correction to the feedback loop. The correction is often calibrated using some sort of adaptive tuning scheme that adjusts the input filter so as to minimize the error induced by the disturbance signal. This correction is faster than what one might do in feedback because the sensor doesn't have to wait for the disturbance to cause an error in the primary feedback loop. Thus, there is far less phase lag between sensed disturbance and correction. In fact, in chemical process control (CPC) this is the main thing they mean when they say feedforward. This is described in Section 8.9.

- We summarize things in Section 8.10.

While these might seem disconnected, one thing that all feedforward schemes share is that their correction signals have much less phase than a correction signal generated as a result of an error in a feedback loop.

## 8.4 Basic Concepts in Feedforward Control

A generic SISO feedforward-feedback control system is diagrammed in Figure 8.5. We have included the two most common feedforward blocks, labeled $F_{PI}$ (plant-input) and $F_{CLI}$ (closed-loop input). These two feedforward blocks are different, but related. Usually, when someone uses the term feedforward, they are implying one or the other of these exclusively. Often, $F_{CLI}$ is termed prefiltering, and is used to remove dynamics from the input signal that might stimulate resonances in the close-loop system,

$$T_{CL} = T = \frac{PC}{1+PC}. \tag{8.1}$$

With some blatant disregard for nonlinearities, sampling, and frequency domain notation, we can get some general ideas about feedforward-feedback systems by generating the block diagram equations associated with Figure 8.5.

$$y = Pu, \tag{8.2}$$

$$u = F_{PI}r + Ce, \text{ and} \tag{8.3}$$

$$e = F_{CLI}r - y = F_{CLI}r - Pu. \tag{8.4}$$

From these we quickly get to:

$$e = F_{CLI}r - P\left(F_{PI}r + Ce\right), \tag{8.5}$$

$$e = \frac{F_{CLI} - PF_{PI}}{1+PC}r, \tag{8.6}$$

$$u = \left[\left(\frac{C}{1+PC}\right)\left[F_{CLI} - PF_{PI}\right] + F_{PI}\right]r, \text{ and} \tag{8.7}$$

$$y = \left[\left(\frac{PC}{1+PC}\right)\left[F_{CLI} - PF_{PI}\right] + PF_{PI}\right]r. \tag{8.8}$$

Generally, either $F_{CLI} = 1$ and $F_{PI}$ is used, or $F_{PI} = 0$ and $F_{CLI}$ is used. Looking at these two cases, we can summarize it as $F_{PI}$ being used to invert the plant, $P$, and $F_{CLI}$ being used to invert the closed-loop transfer function (technically, the closed-loop, complimentary sensitivity function), $T$. Here is the back of the envelope block diagram math to explain this.

Let $F_{CLI} = 1$ and $F_{PI} = P^{-1}$. We will ignore any possible stability, sampling, or nonlinearity issues for this “best we can hope for” ideal analysis. In this case,

$$e = \frac{1-PP^{-1}}{1+PC}r = \frac{1-1}{1+PC}r = 0, \tag{8.9}$$

$$u = F_{PI}r = \frac{1}{P}r, \text{ and} \tag{8.10}$$

$$y = PF_{PI}r = r. \tag{8.11}$$

What the heck could ever be wrong with that? Well, a couple of things:

1) If $P$ is a continuous-time plant, it may or may not have non-minimum-phase (NMP) zeros or even instabilities. Either of these, but especially the NMP causes a problem for inversion. Furthermore, returning a bit closer to the real world, our model of $P$, $\hat{P}$ – which we will use for inversion – may be a discrete-time model that has delays and NMP zeros not found in $P$.

2) There is a different issue many of the optimization methods ignore: for almost any physical system, $P$ is eventually low pass. This means that $P^{-1}$ is eventually high pass. At some point, even our perfect inversion of the plant must have tremendous high frequency amplification to make (8.10) true. It is unlikely that anyone really wants this, even if it could be achieved (which it probably can't). The idea of a responsive car is good, but one that follows every shake of the driver's hand is not. Practically speaking, $\hat{P}^{-1}$ should flatten out or even roll off at some frequency.

"Alright," you say. "You've convinced me. I'll set $F_{PI} = 0$ and use $F_{CLI}$. In perhaps the only time in which it is appropriate to channel our inner Lee Corso, "Not so fast, my friend!" Assume we do so. Then

$$e = \frac{F_{CLI}}{1 + PC} r, \tag{8.12}$$

$$u = \frac{CF_{CLI}}{1 + PC} r, \text{ and} \tag{8.13}$$

$$y = \frac{PC}{1 + PC} F_{CLI} r. \tag{8.14}$$

We are feeling good at this point and decide that inversion has to work here, so we set

$$F_{CLI} = \left(\frac{1 + PC}{PC}\right) \frac{F}{1 + F}. \tag{8.15}$$

Cool. At this point,

$$y = \frac{K}{1 + K} r, \tag{8.16}$$

and we have replaced the input-output behavior of $T_{CL}$ with a new $T_{CL,new}$ of arbitrary bandwidth.

1) We get the same issue as before that $T_{CL}$ may have NMP zeros.

2) Even if that is not the case, it is almost universal that $T_{CL}$ – unless it is designed by a severely twisted individual, is also low pass at some point and thus $\frac{1+PC}{PC}$ runs into our "infinite bandwidth high pass filter" issue. For any real system, we know that we can only invert the dynamics over a limited frequency range before we call off the dogs and limit what we ask of the system.

These considerations should be clear for even the simplest example of block diagram math laced with some common sense about control systems. Any high frequency amplification comes at the expense of amplifying noise, or slamming actuators around. The famous “chatter” of the bang-bang control problem for time-optimal control of a saturated double integrator is exactly this. The system asks for huge gain at high frequency to achieve its optimal result.

At the same time, our simple analysis (and plenty of experience) tells us that we can achieve significant improvements to real control systems with a little bit of feedforward and some common sense.

## 8.5 Measurements for Feedforward-Feedback Control

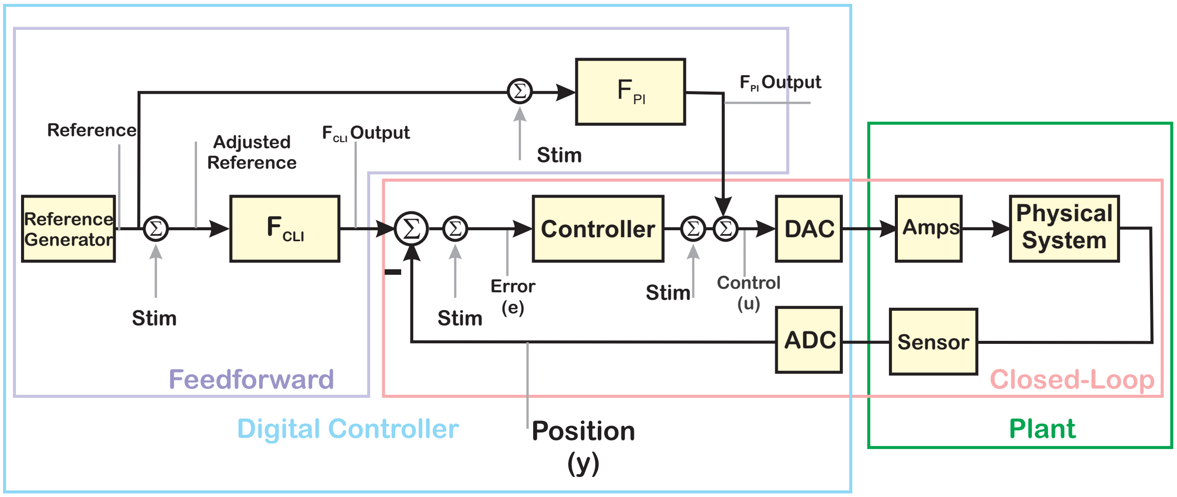


Figure 8.6: The basic diagram of possible feedforward/feedback loop configurations on steroids. Here we see the two most likely feedforward blocks, so-called plant input (PI) and closed-loop input (CLI). We also have locations where one might stimulate and measure the system in order to extract model information.

A key to being able to integrate feedforward control with feedback is once again, good system modeling based on good system measurements. In our newly augmented problem, we have added two extra blocks, plus their associated connections. We need to be able to make accurate loop measurements in order to do accurate design of these systems. An example of such a block diagram is shown in Figure 8.6.

In order to use the $F_{PI}$ block, we need a good measurement of the plant, either via opening a closed-loop measurement or via a three-wire measurement (Section 3.18). To make proper use of $F_{CLI}$, we need to set $F_{CLI} = 1$, $F_{PI} = 0$, and then do a straightforward measurement of the closed-loop response.

## 8.6 A Practical Example of Using $F_{CLI}$ Based Feedforward with Feedback

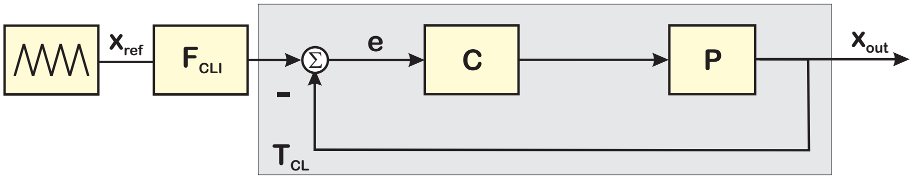


Figure 8.7: Combined feedback-feedforward control using the $F_{CLI}$ input.

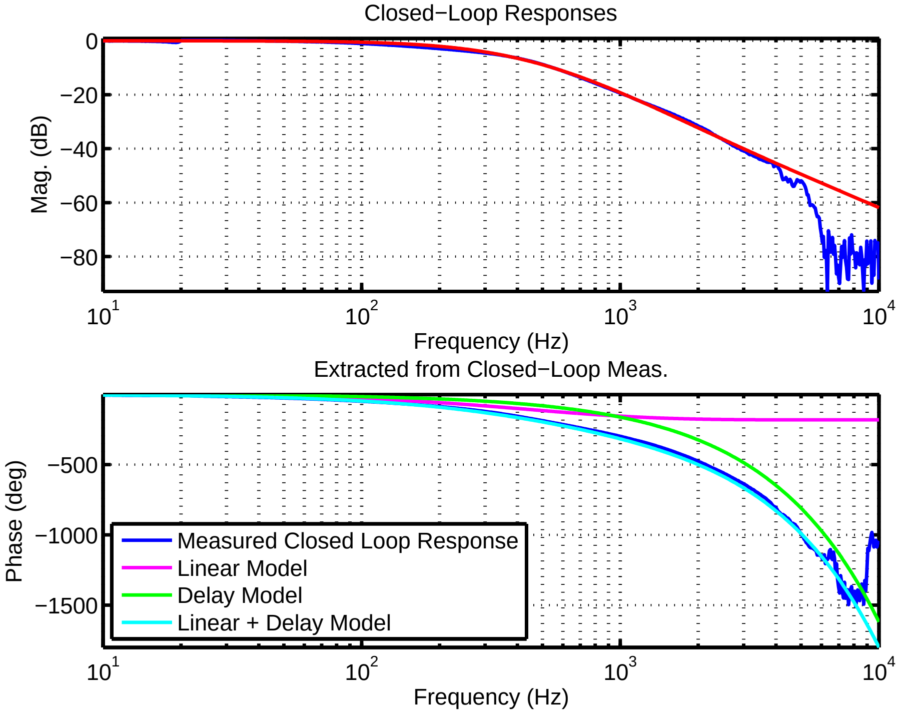


Figure 8.8: Measurement and curve fit of closed-loop response of nPoint NPXY30 x stage. The measured response can be fit well with a simple second-order linear filter plus delay.

One more section shows how these methods can be used to simplify, augment, and parallel model-based work. There is a fair body of work on feedforward control, including the Zero Phase Error Tracking Controller (ZPETC) of Tomizuka [193] and the Zero Magnitude Error Tracking Controller described but not called such in [274, 118] and discussed in Rigney et al. [275]. Feedforward has also been a driving effort in the X-Y control of AFMs [239, 240, 241, 242]. The series of work on combined feedforward-feedback control for mechatronic systems such as X-Y positioners for atomic force microscopes [243, 113, 244, 245, 246, 247, 248, 249, 250] point very strongly to the advantages of using feedforward when the system is presented with a reference signal. These methods largely depend upon

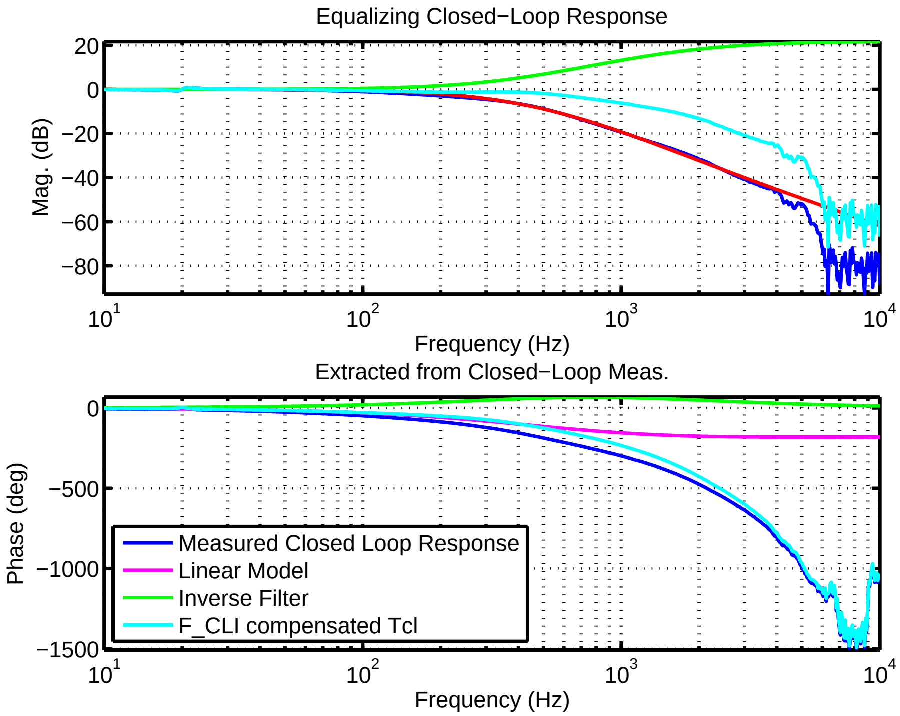


Figure 8.9: $F_{CLI}$ implemented as as simple double lead filter more that doubles the bandwidth while keeping the controller simple.

first generating an effective feedback controller based on a plant model and then designing a feedforward controller based on the closed-loop model generated from the plant and controller models. The work in [246] and [250] did not assume knowledge of the plant model, but only of the closed-loop response, as diagrammed in Figure 8.7. Still, as described in [244, 245] the objective was described as perfect tracking, in which $F_{CLI} = T_{CL}^{-1}$. While this might seem reasonable to do if $T_{CL}$ was minimum phase, most engineers would realize that this result requires infinite bandwidth from $F_{CLI}T_{CL}$, which would not only violate the Nyquist Criterion, but also cause the actuators to operate at high speed on amplified quantization noise. A more practical look at this yields a much simpler and more practical design method.

Realizing that the tools described earlier, particularly the built in stepped-sine of Section 3.26, with a digital patch panel that allows us to make measurements from wherever we want in Figure 3.30, we can easily make a stepped-sine measurement of our closed-loop system. If we have adhered to the design approach of making the open loop look like an integrator and preserving 60° of phase margin, then our closed-loop FRF, $T_{CL}(f)$ is likely to have the response of a low-pass filter.

From here, we can measure $T_{CL}$ and generate our desired transfer function shape (say a low-pass filter with more bandwidth). Our fitting routines can then adjust a multinotch to provide what ends up being

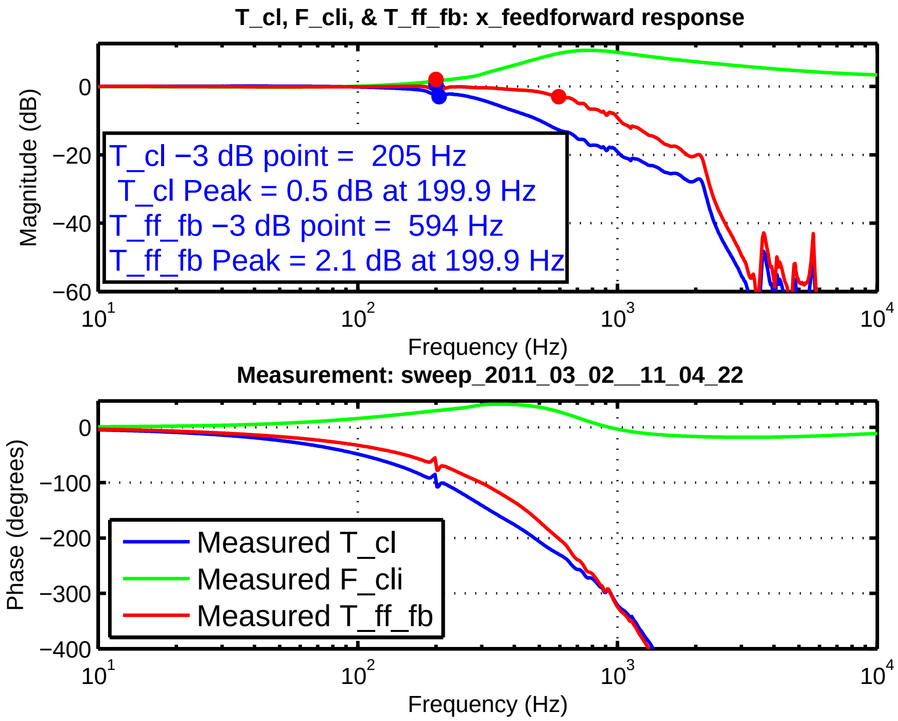


Figure 8.10: Measured $T_{CL}$, $F_{CLI}$, and $T_{FF,FB}$ on nPoint NPXY100 stage. The x axis feedback controller was autotuned to produce the $T_{CL}$ response. A new measurement was done, and from this a feedforward tune was done to generate $F_{CLI}$. The measurement was repeated to generate the combined $T_{FF,FB} = F_{CLI}T_{CL}$ response.

a combination of lead-lag filters and notch/bump filters. An example of this is shown in Figure 8.10, where an nPoint NPXY100 stage was measured in the X axis. A feedback controller was generated using a combination of PID and multinotch filters as described in Section 4.15. From there, the feedforward controller was generated as described above to almost triple the input-output bandwidth. Note that unlike the perfect tracking filter, we do not try to for infinite bandwidth. We can see this in the plots of Figures 8.11, where the improvement of reference tracking is very clear beyond the bandwidth of $T_{CL}$. The use of a double lead also means that the requested increased bandwidth can be limited to something reasonable for the physical system to achieve. Returning to our model-based approach, we see that this is equivalent to asking for $F_{CLI}T_{CL}$ to have a shape of a new low-pass filter with approximately triple the reference to position bandwidth of $T_{CL}$ alone.

$F_{CLI}T_{CL}$ with perfect tracking and infinite bandwidth is optimal, but sucks. $F_{CLI}T_{CL}$ with improved tracking and high bandwidth is excellent.

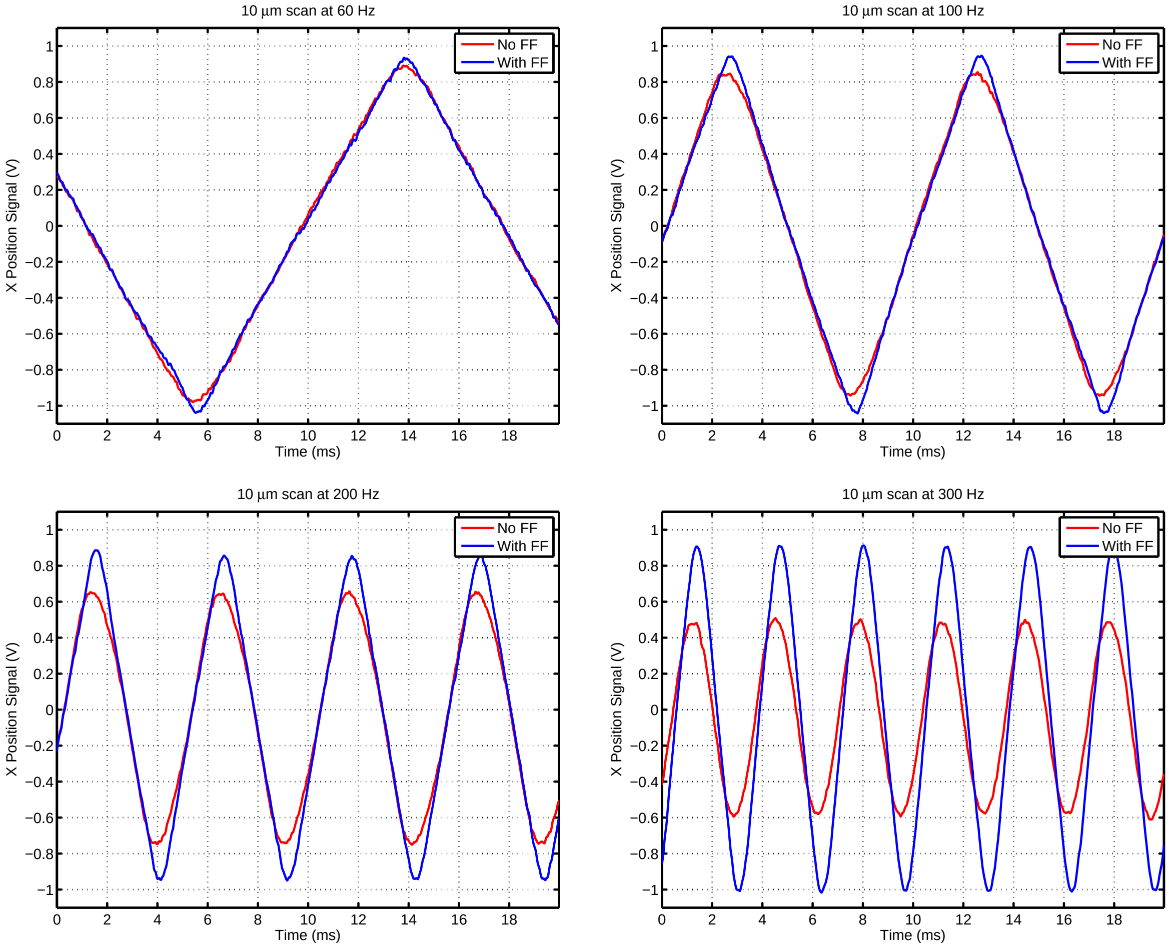


Figure 8.11: Time response data measured using an Agilent (now Keysight) Infiniium 54831M digital scope on the physical system shows the effect of the feedforward. From upper left to lower right, the reference input is a triangle wave of 60 Hz, 100 Hz, 200 Hz, and 300 Hz. At 60 Hz, both pure feedback and combined feedforward/feedback controllers can match the first and third harmonics constituting the triangle wave, but as the reference frequency goes up, only the combined controller can keep up. The combined controller can match the fundamental of the 300 Hz reference, but the third harmonic is about 10 dB down, leading to rounding of the response. On the left, the response differences between the pure feedback and combined feedforward-feedback controllers are minimal when the reference frequency is within the bandwidth of $T_{CL}$ at 100 Hz. The differences become severe when the triangle wave bandwidth is raised to 300 Hz. The combined $T_{FF,FB}$ response is able to match the fundamental, but the higher harmonics of the triangle wave are attenuated.

## 8.7 Input Shaping Feedforward

Up to now, we have thought of input-filter feedforward as a way to get the input into the closed-loop system either by bypassing the compensator in the forward path ($F_{PI}$) or by compensating for and extending the input-output behavior of the closed-loop system ($F_{CLI}$). In both cases, we have tried to be reasonable: we cannot perfectly invert most plants using $F_{PI}$ and we cannot perfectly invert most closed loops using $F_{CLI}$. Even when perfect inversion is theoretically possible, we want to roll off our transfer functions at higher frequency if only to not as for infinite input-output bandwidth or violate the Nyquist criterion.

Both of these have – in a large sense – relied on the ideas that we have:

a) identified all the main resonances and anti-resonance of the plant for $F_{PI}$ and

b) largely flattened out any critical resonances in the closed-loop response.

In doing so, we can improve the input-output response of the system without incurring extra sensor noise.

However, there is another use of the input filter, $F_{CLI}$, and that is to shape the input so as to remove signals and/or artifacts that might affect the closed-loop behavior. This use was somewhat obscured by our example where the original closed-loop response was shaped to be a nice low pass filter. Instead, many closed-loop responses have some artifacts at higher frequency (or even in their pass band) which might be stimulated by signals such as step functions at $r$. We can design $F_{CLI}$ then to remove those components from $r$so that the problematical closed-loop dynamics never get stimulated.

In the diagram of Figure 8.5, $F_{CLI}$ appears as a causal prefilter, but if the reference input is known in advance, the functionality of $F_{CLI}$ can be accomplished in a causal/acausal filter, removing any possible phase effects from the application of $F_{CLI}$. This prefiltered input can be applied in place of the original $r$ with the idea that it will not stimulate any of the residual resonances in $T_{CL}$.

One version of this input shaping, in which a specific set of pulses is convolved with the input to remove particular frequency components of that signal, was pioneered by Neal Singer and Warren Seering [276, 277]. However, it can be argued that this is simply applying a very specific prefilter to the input signal to remove those signal components.

## 8.8 Repetitive Feedforward Control

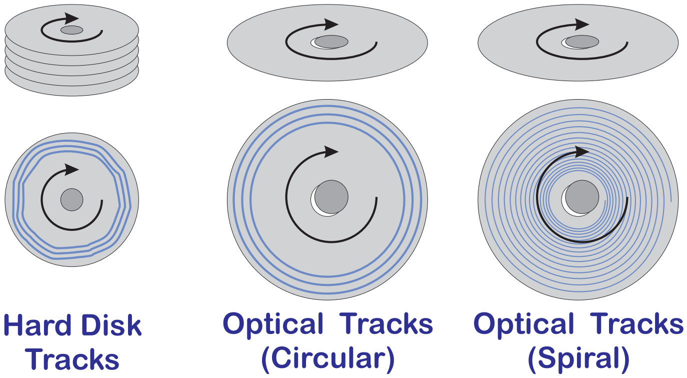


Figure 8.12: Track eccentricity in hard disks (left), optical disks with circular tracks (center), and optical disks with spiral tracks (right). In the case of the hard disk, the media is attached to the spindle and the tracks written, resulting in a smaller overall eccentricity but an eccentricity that includes the spindle frequency and harmonics, often related to the number of balls in the ball bearings. (Modern hard disks now have fluid bearing spindles, but mostly for audio noise minimization.) In the case of the optical drives, the tracks are nearly perfect around the disk hole, but this hole is can be put off center on the spindle – as it's removable media. Thus, there are few harmonics, but a large fundamental frequency eccentricity.

Rotating machinery, e.g. magnetic and optical disk drives, motors, milling machines, generators, jet engines, propellers, tend to have disturbances that are at the rotational frequency and harmonics of that frequency. In many of these machines, such as the disk drives and the milling machines, the disturbance itself is not so important (within bounds) as the error generated by the disturbance. A familiar example is that of track eccentricity in optical and magnetic disk drives, diagrammed in Figure 8.12 and described in the caption. While these disturbances are often within the controller bandwidth, a normal feedback controller may lack the gain at the harmonic frequency to reject the disturbance sufficiently. This looks like a job for repetitive control and unlike many advanced techniques, this has been simple and effective enough to enjoy wide usage in industry.

One of the most common versions of feedforward control is repetitive control, diagrammed in two different forms in Figures 8.13 and 8.14. Looking at either of these forms, one would immediately sputter that they are not feedforward, but augmented feedback controllers. One would be right, except that these controllers are designed to only act on the repetitive portion of the error signal, and by converging on and acting on the repetitive portion, they are in fact doing feedforward, causal/acausal control with no delay on the future errors which have already occurred in the past. (Somewhere, Yogi Berra is smiling.).

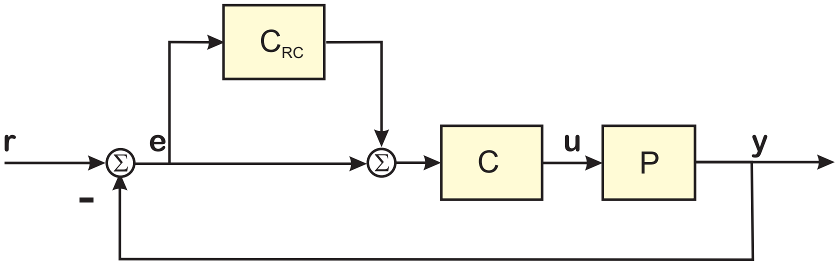


Figure 8.13: The model structure for add-in repetitive control (RC) when the designer can only read and inject at the error signal.

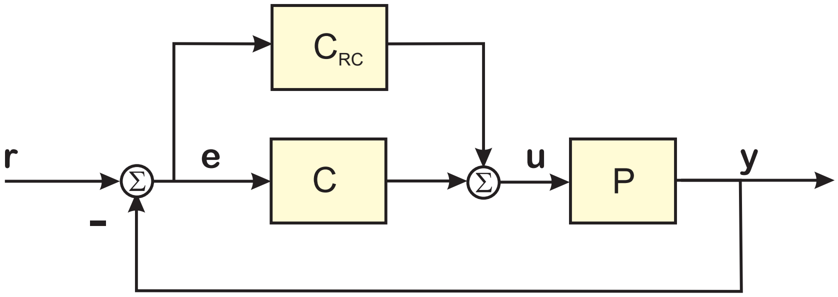


Figure 8.14: The model structure for add-in repetitive control (RC) when the designer can read the error signal but inject beyond the feedback (FB) controller.

The form in Figure 8.14 is that of an add-in repetitive controller, while the second form is most often associated with what is known as an adaptive feedforward canceler (AFC). In its most common usage, the Fourier coefficients of a periodic disturbance are estimated and then a feedforward cancellation term is generated using sinusoidal signals with the estimated coefficients [190, 278]. Note that this assumes that the disturbance is sinusoidal. If not multiple cancelers will be needed.

The form in Figure 8.13 is that of an add-in repetitive controller. The basic idea behind this is the internal model principle[279]. Briefly, the internal model principle states that to cancel a particular repetitive disturbance with a feedback loop, one should include a model of that disturbance within the loop. (Hence the name internal model.) The idea is to have an N-tap delay in positive feedback inside the loop, where N$T_S$ is the period of the disturbance. This integrates/averages the repetitive portion of the error signal generating an internal model of the periodic disturbance to cancel it out.

That being said, the two structures above augment the main controller, $C$, in different ways. In Figure

8.13,

$$C_{Tot} = (1 + C_{RC})\,C, \tag{8.17}$$

while Figure 8.14 produces

$$C_{Tot} = C_{RC} + C. \tag{8.18}$$

In either case, we need to make certain that the repetitive controller doesn't destabilize the loop. In Equation 8.17, the repetitive controller's multiplying effect on $C$, and so if in particular $1 + C_{RC}$ has an integrating action (as things that average do), we need to make certain that they integrator action happens at a low enough frequency not to affect loop stability.

In Equation 8.18, the structure adds on to $C$, and again we need to make certain that it doesn't affect the loop at crossover. The structure of an AFC is to tune a sine and cosine generator at the repetitive frequency to cancel out a sinusoidal disturbance at that frequency. From a frequency domain perspective, this is equivalent to having a very narrow filter with little or no phase effect. Thinking about this filter on the open-loop response, we would want it to be applied somewhere well away from the open-loop magnitude crossover frequency. This often can be handled because the repetitive signals to be canceled are typically at far lower frequency that the open-loop magnitude and phase crossover frequencies.

### 8.8.1 An Adaptive Feedforward Canceler

An adaptive feedforward canceler can be implemented as follows. The matching portion consists of modeling the disturbance as a Fourier series and identifying the relevant Fourier coefficients. From here, the cancellation portion merely injects this matched signal into the loop at an appropriate spot to either cancel the signal (ignoring the error) or follow the signal. The harmonic corrector for a once around disturbance (at the first harmonic) could take the form of

$$OA_{corr} = A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t) \tag{8.19}$$

For a Fourier series expansion of the disturbance, the first Fourier coefficients can be computed as

$$\begin{aligned} A_1 &= \frac{2}{T}\int_0^T \cos(\omega_0 t)\phi_0(t)dt && (8.20)\\ B_1 &= \frac{2}{T}\int_0^T \sin(\omega_0 t)\phi_0(t)dt, && (8.21) \end{aligned}$$

where $T = \frac{1}{f_0} = \frac{2\pi}{\omega_0}$. As this will almost certainly be done in a digital system, the integral needs to be approximated by a sum such as

$$a_1 \approx \frac{2}{T}\sum_{k=0}^{N-1}\cos(\omega_0 t_k)\phi_0(t_k) \qquad (8.22)$$

$$b_1 \approx \frac{2}{T}\sum_{k=0}^{N-1}\sin(\omega_0 t_k)\phi_0(t_k) \qquad (8.23)$$

where $N$ is the number of samples in a revolution. Similarly, the Fourier coefficients for the $n$th term of the Fourier series can be approximated by

$$a_n \approx \frac{2}{T}\sum_{k=0}^{N-1}\cos(n\omega_0 t_k)\phi_0(t_k) \qquad (8.24)$$

$$b_n \approx \frac{2}{T}\sum_{k=0}^{N-1}\sin(n\omega_0 t_k)\phi_0(t_k). \qquad (8.25)$$

The analysis here will be done for the first Fourier coefficients (i.e. for the first harmonic), but it is understood that it could easily be done for any linear combination of harmonics that were deemed important to cancel.

Some refinement of these terms is possible for programming a digital system. Given that $\omega_0 = 2\pi f_0$ is the spindle frequency in radians/second, then the period of a revolution would be $T_0$. Assuming that the phase error is sampled $N$ times per revolution, then for time sample $t_k$, the following are equivalent.

$$\omega_0 t_k = 2\pi f_0 t_k \qquad (8.26)$$

$$= 2\pi \frac{t_k}{T_0} \qquad (8.27)$$

$$= 2\pi \frac{k}{N}. \qquad (8.28)$$

Equations 8.22 and 8.23 become

$$a_1 = \frac{2}{N}\sum_{k=0}^{N-1}\cos\left(\frac{2\pi}{N}k\right) \qquad (8.29)$$

$$b_1 = \frac{2}{N}\sum_{k=0}^{N-1}\sin\left(\frac{2\pi}{N}k\right). \qquad (8.30)$$

This can be implemented by updating a running sum for $a_1$ and $b_1$ at each sample time around the disk. At the end of a revolution, the sums are scaled to give that revolution's estimate of the Fourier coefficient. This scaled sum is then used in a regression to adapt the actual coefficients.

Pseudo-code for a computer algorithm to implement this can take the following form:

```
At each step, $k$:
    if ((k mod N) == 0)                          % At the end of each revolution
        $a_1 = \frac{2}{N} * a_\Sigma$                % Scale running sum for cosine
        $b_1 = \frac{2}{N} * b_\Sigma$                % Scale running sum for sine
        $A = A + \mu a_1$                         % Adapt coefficients using portion of sums
        $B = B + \mu b_1$
        $a_\Sigma = 0$                             % Reset running sums for next revolution
        $b_\Sigma = 0$
    end
    $a_\Sigma = a_\Sigma + \cos\left(\frac{2\pi}{N}k\right)$       % Update the sums
    $b_\Sigma = b_\Sigma + \sin\left(\frac{2\pi}{Nm}k\right)$
    $OA_{corr}(k) = A * \cos\left(\frac{2\pi}{N}k\right) + B * \sin\left(\frac{2\pi}{N}k\right)$   % Update the corrector
    end
```

The advantages of a converged sine/cosine generator are:

1) The effect is very narrow about $\omega_0$ $(f_0)$. Remembering the concept of resolution bandwidth, the longer $T_0$ and the smaller $\mu$, the longer the "integration time," so the narrower the frequency width of the "filter."

2) Because it works on repetitive stuff, it has no phase delay. "Everything old (and periodic), is new again." It is akin to a causal/acausal FIR filter with symmetric taps.

3) Which means that we have a very narrow band filter spike at $\omega_0$ $(f_0)$ with no phase delay on the repetitive controller. A log of gain to reject the fundamental, which is great so long as $f_0$ is well below the open-loop gain crossover frequency or lots of notching (with no phase hit) if $f_0$ is well below the crossover.

4) The keys are to converge and generally stay away from the open-loop gain and phase crossover frequencies.

5) We can add as many of these as we need for each $kf_0$ harmonic.

6) Generally, the parallel structure has a simpler stability properties than the add in structure of Equation 8.17 and Figure 8.13.

The disadvantages of a converged sine/cosine generator compared to a repetitive controller are:

a) It requires one sine generator component for each harmonic of $f_0$. This can become computationally expensive if we have a lot of harmonics.

b) The computations here are relatively simple, except that one has to calculate sines and cosines. This is a library call in most computation cases, which can cost 30-70 CPU clock cycles, based on the CORIC algorithms [227, 228]. For fast real-time computation, these may need to be substituted with look-up-tables (LUTs) which can be used to get fairly accurate estimates of the sines and cosines in around 10 clock cycles.

### 8.8.2 Add-In Repetitive Controller Primer

The basic form [280, 281, 282] of the discrete time version of this controller comes from putting a delay chain in a positive feedback loop as shown in Figure 8.15. This comes from the notion that in discrete time, any periodic signal with period $N$ can be generated by a delay chain with a positive feedback loop.

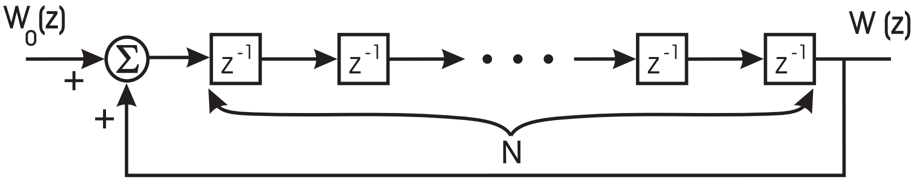


Figure 8.15: Repetitive control can be implemented by putting a string of $N$ unit delays in a positive feedback loop. Note that this version has the delays in the forward path of the feedback loop.

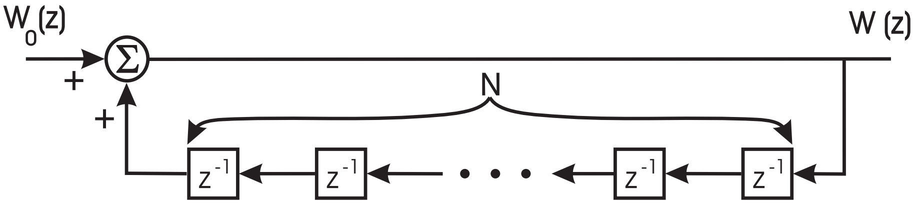


Figure 8.16: An alternate implementation of repetitive control puts the string of $N$ unit delays in the feedback path of the positive feedback loop.

Let the discrete time sequence for $\{w_0(k)\}$ be defined as

$$w_0(k) = \{w(0), w(1), w(2), \ldots, w(N-1), 0, 0, 0, \ldots\}. \tag{8.31}$$

Then the discrete time $\mathcal{Z}$ transform of this sequence will be

$$W_0(z) = w(0) + w(1)z^{-1} + w(2)z^{-2} + \ldots + w(N-1)z^{-(N-1)} \tag{8.32}$$

In Figure 8.15, W(z) will be a transform of $\{w(k)\}$ which is the repetition of $\{w_0(k)\}$. That is to say, when $\{w_0(k)\}$ is injected, nothing will emerge at $w(k)$ for $N$ time samples. At that point, the $\{w_0(k)\}$ sequence appears at $w(k)$ and reinjects back into the feedback summer. The $\{w_0(k)\}$ sequence going into the reference input is then 0, so all we have is the recirculating sequence. In the case were the $\{w_0(k)\}$ sequence repeats itself after $N$ samples, we see this positive feedback structure as providing a periodic integrator structure with period $N$. This periodic integrator, when coupled into a feedback loop will work to drive out all disturbances of period $N$, just a single integrator would drive out a constant disturbance.

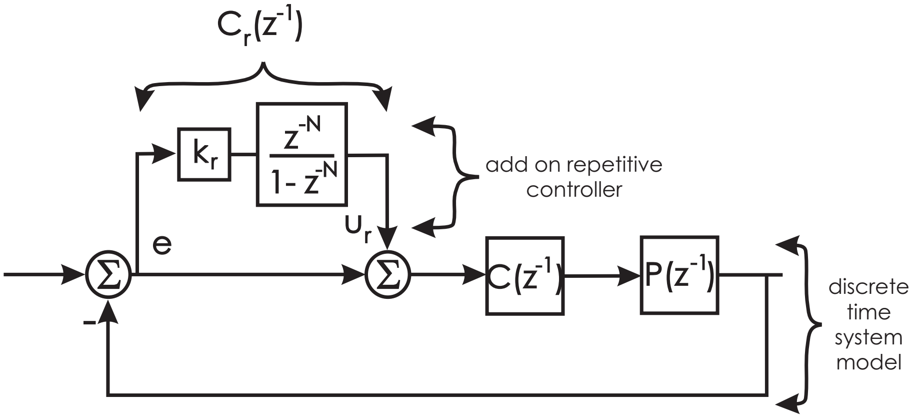

$C_r(z^{-1})$
$k_r$
$\frac{z^{-N}}{1-z^{-N}}$
add on repetitive controller
$u_r$
e
-
$C(z^{-1})$
$P(z^{-1})$
discrete time system model

Figure 8.17: Discrete time repetitive controller added to a discrete time closed-loop system. Note that the positive feedback loop is now compressed into the $\frac{z^{-N}}{1-z^{-N}}$ term.

Now,

$$W(z) = \frac{z^{-N}}{1-z^{-N}} W_0(z). \tag{8.33}$$

Note that if we want to avoid the first $N$ delays before the repetitive controller has an output, we can take the $W(z)$ signal off immediately after the summing junction. This can be drawn equivalently as shown in Figure 8.16. In this case,

$$W(z) = \frac{W_0(z)}{1-z^{-N}}. \tag{8.34}$$

This simple unit can be added as a repetitive control block to a nominal feedback system. This is shown in Figure 8.17 when the nominal system has already been discretized and in Figure 8.18 when the nominal system is viewed in a continuous time form. Note that in Figure 8.18, the repetitive controller has been included in the single term $C_r(z^{-1})$.

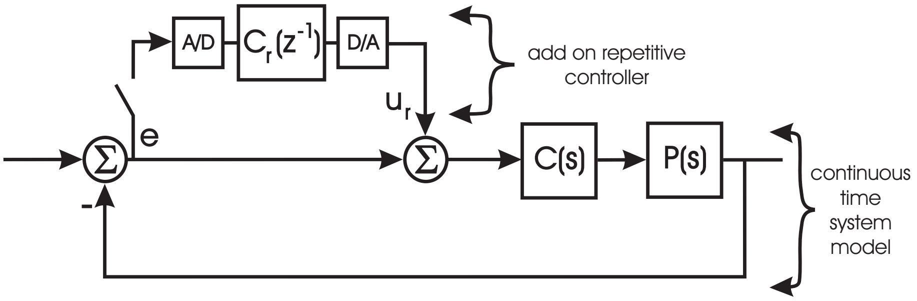


Figure 8.18: Discrete time repetitive controller added to continuous time closed-loop system. Note that from the perspective of $C_r(z^{-1})$ the system looks unchanged.

Note that in general, the only required component of $C_r(z^{-1})$ for complete cancellation is the $\frac{1}{1-z^{-N}}$ factor. Adding a term to the loop of the form

$$C_r(z^{-1}) = k_r \frac{z^{-N}}{1 - z^{-N}} \tag{8.35}$$

does not guarantee stability of the new closed loop system. This is analogous to the fact that adding an integrator into a standard control system does not guarantee that the new system is stable. In fact, changes may be necessary to provide stability. That is to say, that even if the original closed loop system were stable, the insertion of this repetitive controller may result in an unstable system. To compensate for this, a more general form of the repetitive controller

$$C_r(z^{-1}) = \frac{R(z^{-1})}{S(z^{-1})(1 - z^{-N})} \tag{8.36}$$

is often used.

Looking from input $u_r$ to "output" $e$, the repetitive controller $C_r(z^{-1})$ in Figure 8.19 sees the following discrete time transfer function

$$\frac{e}{u_r} = \frac{-P(z^{-1})C(z^{-1})}{1 + P(z^{-1})C(z^{-1})} = \frac{z^{-d}B(z^{-1})}{A(z^{-1})} = G_s(z^{-1}). \tag{8.37}$$

The control design problem then becomes to find a repetitive controller, $C_r(z^{-1})$ so that the overall closed

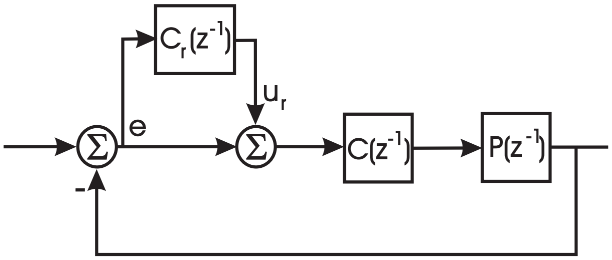


Figure 8.19: Generating a model of the system for stability analysis.

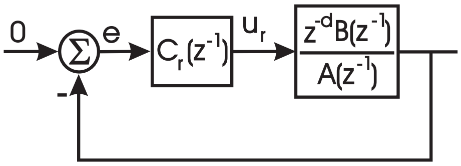


Figure 8.20: Stability analysis version of the system.

loop discrete time system,

$$T_{rc}(z^{-1}) = \frac{\frac{R(z^{-1})z^{-d}B(z^{-1})}{S(z^{-1})\left(1-z^{-N}\right)A(z^{-1})}}{1+\frac{R(z^{-1})z^{-d}B(z^{-1})}{S(z^{-1})\left(1-z^{-N}\right)A(z^{-1})}} \tag{8.38}$$

$$= \frac{R(z^{-1})z^{-d}B(z^{-1})}{S(z^{-1})\left(1-z^{-N}\right)A(z^{-1})+R(z^{-1})z^{-d}B(z^{-1})} \tag{8.39}$$

is stable. That means the denominator must have all it's roots inside the unit circle.

Since it is often hard to solve this problem in real time, the original work on applying repetitive control to disk drives [280] suggests a prototype repetitive controller. For a nominal closed loop system described by

$$G_s(z^{-1}) = \frac{z^{-d}B(z^{-1})}{A(z^{-1})} \tag{8.40}$$

and assuming that this nominal system is stable (the roots of $A(z^{-1})$ all lie within the unit circle), then as long as $B(z^{-1})$ and $(1-z^{-N})$ are co-prime (i.e. they do not have any common factors) then the following controller can provide perfect regulation for repetitive disturbances with a period of $N$:

$$C_r(z^{-1}) = \frac{k_r z^{-N+d}A(z^{-1})B^{-}(z)}{\left(1-z^{-N}\right)bB^{+}(z^{-1})} \tag{8.41}$$

where $b \geq \max\left|B^{-}(e^{-j\omega})\right|^2$ and $k_r \in (0,2)$. Note that $B(z^{-1}) = B^{-}(z^{-1})B^{+}(z^{-1})$ where $B^{+}(z^{-1})$ is the part of $B(z^{-1})$ that has roots inside the unit circle (so it can be safely inverted) and $B^{-}(z^{-1})$ is the part of

$B(z^{-1})$ that has roots on our outside the unit circle (so it cannot be inverted). Since the latter part can't be canceled properly, the trick of putting $B^{-}(z)$ in the numerator takes the reciprocal roots of the outside the unit circle portion and uses those to at least cancel the phase effects of the part of $B(z^{-1})$ that cannot be canceled completely.

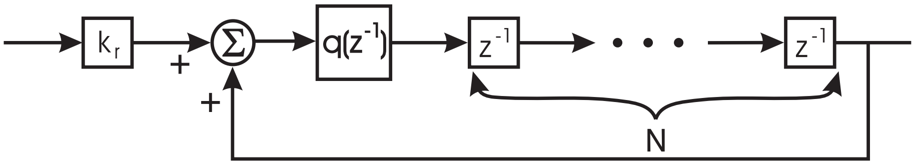


Figure 8.21: Addition of a "$q$ filter" to the repetitive controller.

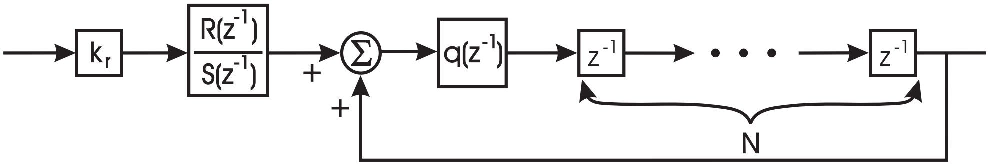


Figure 8.22: Generalized repetitive controller.

One final addition is generally made to relax the stability requirements in a practical system. In this case, a filter, $q(z^{-1})$ is inserted into the $\left(1 - z^{-N}\right)$ repetitive controller as shown in Figure 8.21 for the simple controller and in Figure 8.22 for the generalized controller. In the latter case, using the prototype controller means that

$$C_r(z^{-1}) = \frac{k_r z^{-N+d} q(z^{-1}) A(z^{-1}) B^{-}(z)}{\left(1 - q(z^{-1}) z^{-N}\right) bB^{+}(z^{-1})} \tag{8.42}$$

and

$$\frac{R(z^{-1})}{S(z^{-1})} = \frac{k_r z^{+d} A(z^{-1}) B^{-}(z)}{bB^{+}(z^{-1})}. \tag{8.43}$$

Typically, $q(z^{-1})$ is a low pass filter which is interpreted as giving up some rejection of the higher frequency harmonics for improved stability.

The Small Gain Theorem [283] states that the overall system will be stable if

$$\left|1 - k_r \frac{A(e^{-j\omega}) B_m^{-}(e^{j\omega}) B(-e^{j\omega})}{b_m B_m^{+}(e^{-j\omega}) A(e^{-j\omega})}\right| < \left|\frac{1}{q(e^{-j\omega})}\right|, \tag{8.44}$$

for $\omega \in [0, \pi]$, and with $A_m, B_m$, and $b_m$ representing the modeled quantities of the closed loop system parameters.

### 8.8.3 Repetitive Control Versus Adaptive Feedforward Correction

As mentioned in [190] repetitive control and adaptive feedforward correction are functionally equivalent to each other from a long term analysis perspective. What determines which one is the better candidate?

Fundamentally, in a digital implementation, it comes down to a choice between computer cycles and memory space. Typically, repetitive control takes fewer code cycles to implement. One merely implements the positive feedback loop in rewritable memory (RAM) with a few possible filter additions (the $R(z^{-1})$, $S(z^{-1})$, and $q(z^{-1})$ filters). The main cost is the memory storage to contain all the taps in this positive feedback loop. For a 50 kHz servo sample rate and a 50 Hz spindle rotation, one requires 1000 memory locations for this filter. On the other hand, adaptive feedforward cancellation requires far fewer memory locations. To cancel a single sinusoid, one would need two coefficients for the sinusoids (the $\bar{A}_1$ and $\bar{B}_1$ terms), one for the adaptation gain, and two for the running sums, for a total of 5 long term memory storage locations. The tradeoff here is that one has use more CPU cycles to generate the sine and cosine signals and do the parameter adaptation.

Both of these methods have design tradeoffs that can simplify them to bring them closer to the other. In the case of repetitive control, the repetitive controller can be sampled at a slower rate than that of the rest of the servo system. While this has a minor effect on the CPU cycles needed, the main savings is that the memory requirements also decrease by the subsample factor. In the case of adaptive feedforward cancellation the generation of the sine and cosine terms is the most computationally expensive and can be adjusted depending upon the accuracy required in generating those terms. Furthermore, a subsampling can also be used to save CPU cycles. (It will not cut the memory requirements.)

One of the advantages of the AFC is that one selects only those harmonics that one wants to cancel. This allows the repetitive spikes to be very, very narrow (depending on the quality of the integration) thus making the design very specific. If we stay away from the crossover frequencies, we can be pretty confident that this will work. In the repetitive controller, it tries to cancel anything that is integrated in by its delayed integrator. The analysis of the controller has to be done based on the nominal closed-loop system, that can be more complex.

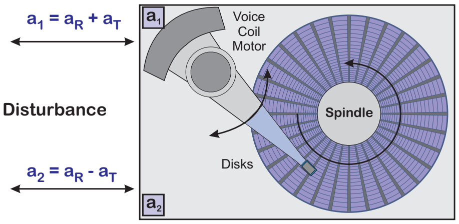


Figure 8.23: Diagram of hard disk drive (HDD) subjected to translational and rotational accelerations to the head disk assembly (HDA). The individual accelerometers can be thought of as detecting the sum and difference of rotational acceleration, $a_R$, and translational acceleration, $a_T$. If the accelerometer gains are equal, then their difference gives $2a_R$.

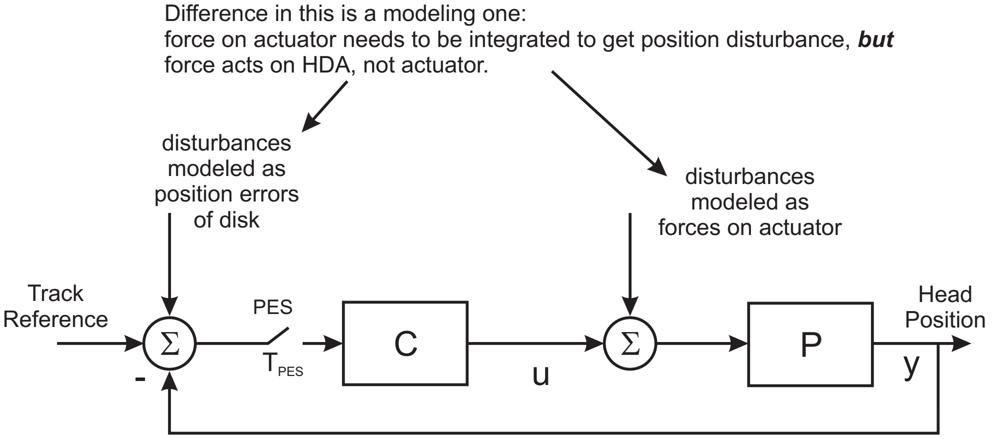

Difference in this is a modeling one:
force on actuator needs to be integrated to get position disturbance, ***but***
force acts on HDA, not actuator.
disturbances modeled as position errors of disk
disturbances modeled as forces on actuator
Track Reference
Σ
-
PES
$T_{PES}$
C
u
Σ
P
Head Position
y

Figure 8.24: Disturbances entering a hard disk drive control loop. They are often modeled as disturbances to the plant input, but in this case, the disturbances have no effect on the actuator and instead move the track out from under the actuator. In this case, it is better to model the disturbance as entering directly into the error signal (on the left).

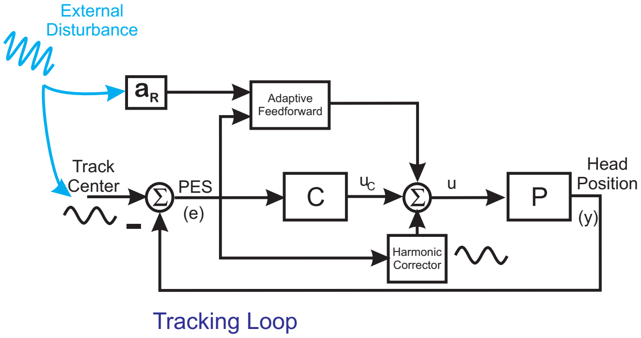


Figure 8.25: Disk drive servo loop showing auxiliary loops.

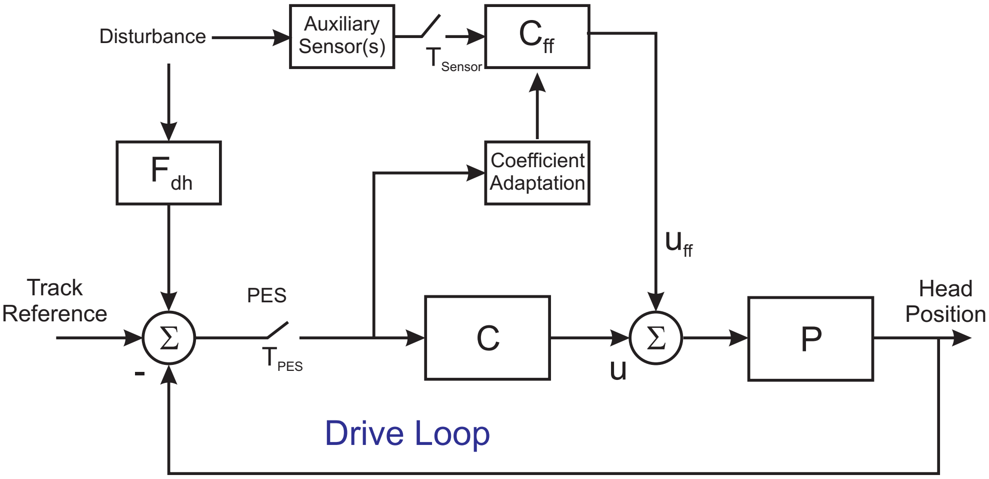


Figure 8.26: Using extra sensors to detect disturbances in a hard disk drive (HDD) control loop. The sensor detects the disturbance and compensates for it in the control signal. The auxiliary sensor feedforward loop is calibrate by decorrelating the sensed disturbance with the PES.

## 8.9 Feedforward from Auxiliary Sensors

One more form of feedforward is by using a signal that comes from an auxiliary sensor, not in the nominal feedback path, to improve disturbance rejection in the control system, mainly by “seeing” or sensing the disturbance and injecting a canceling signal into the feedback loop. This frees the feedback controller from every having to deal with the disturbance. An example I worked on comes from the hard disk industry, where the nominal loop sample rate is limited due to the “sectored servo” approach that multiplexes servo information with user data along the track. If you want to sample faster, you have to loose some user data, which gets to a point of diminishing return fairly quickly.

The drives are often subject to external disturbances that move the Head Disk Assembly (HDA) out from under the read/write head (which is at the end of the actuator arm). The servo can nominally follow some of the disturbance, but it works better with one or more auxiliary sensors. In the diagram of Figure 8.23, we see that this particular disk has two linear actuators mounted on the HDA. If properly configured and balanced, their average gives a measure of the translational disturbance (which should have minimal affect on the Position Error Signal, PES) while their difference gives a measure of the rotational disturbance (which greatly affects PES). If we could properly sense the rotational disturbance, we could reject the effects of this rotational disturbance. One idea is to use a rotational accelerometer [187]. However, rotational accelerometers are more expensive than two linear accelerometers, shown in Figure 8.23.

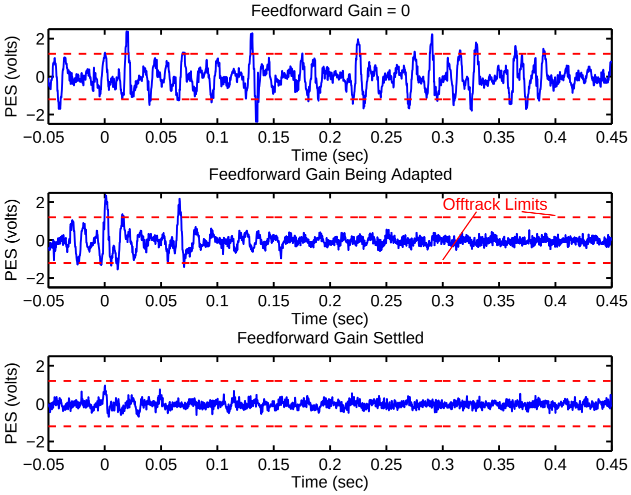


Figure 8.27: Adaptive feedforward accelerometer compensation of rotary vibration on a KittyHawk disk drive. The top plot shows the effects of rotary disturbance with no feedforward. The red dashed lines are the offtrack limits. When adaptive feedforward is turned on in the middle plot, the effects of the disturbance are quickly minimized. In the lower plot, the drive stays within the offtrack limits, despite the external rotary disturbance.

In practice, it is hard to adapt entire control systems for mechatronic systems, largely because of the previously discussed issues with physical properties being obscured in the discrete model. However, certain problems decompose this way, and this is the case for accelerometer feedforward on disk drives.

Hard disks multiplex position information with user data [31], so there is a desire to minimize the position information so as to maximize the user data. However, hard drives used in server farms encounter disturbances from mechanical interaction with other hard disks in the same server. In particular, while the drives have balanced actuators that makes the offtrack signal largely immune to translational shocks and vibration, the very nature of a rotary actuator makes it highly susceptible to rotary shocks and vibration in the plane of the disk. One would expect the main feedback loop to reject these disturbances, but the sample rate restrictions due to the trade-off between servo information and user data limit this capability.

One solution is to add an auxiliary loop using extra sensors to the basic drive control problem as shown in Figure 8.25. Furthermore, these sensors have no effect on user data and so they can be sampled faster than the main loop. These loops are adaptive [187, 284, 285]. Early on, it was thought that rotary accelerometers would be the best choice for these problems, since they would pass only the rotary disturbances and not the translational ones. However, rotary accelerometers are expensive, so two linear accelerometers are differenced, as shown in Figure 8.23. The linear accelerometers are quite inexpensive,

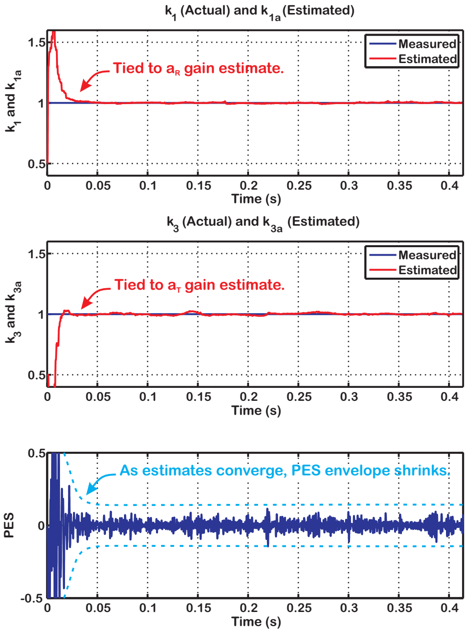


Figure 8.28: Adaptation simulation using noise driven $a_R$ and $a_T$, i.e., $a_R$ and $a_T$ are driven by noise only. The parameters adjust to minimize PES. Note the rapid convergence of $k_{3a}$ to $k_3$ (balancing accelerometer gains) and $k_{1a}$ to $k_1$ (adjusting feedforward gain from $a_R$). The effects of disturbance on PES rapidly goes away.

but that comes at the price of mismatch between the two, which can be in the range of ±15%. This often limits the benefits of such a feedforward system [286]. This problem was recently solved by the author by using the parasitic error signal caused by unbalanced accelerometers when subjected to translational shock and vibration to calibrate those same accelerometers to each other [188].

The HP KittyHawk 1.3" disk drive was slated for mobile applications where shock and vibration would be an issue. While translational disturbances were assumed to be decoupled because of the balanced rotary actuator, rotational disturbances entered directly into the tracking loop. Furthermore, the sectored servo [31, 287] employed by hard disks limited the sensor bandwidth of the PES signal. However, there was no limit on accelerometer sample rates. This allowed for multi-rate feedforward cancellation using the accelerometer, building on the development of [288, 289].

At a poster session at the 1996 IFAC World Congress in San Francisco [187, 284], Matt White was presenting work on rejecting rotational disturbances in full size (5 1/4" at the time) disk drives [285, 290]. Matt's work was a careful study with multiple adaptive parameters. At some point when the session got quiet he turned to me and asked something to the effect of, "How the heck does your work look so simple?" As we talked, a couple of differences emerged: First, the smaller KittyHawk disk had simpler dynamics and I was able to formulate the problem into a single parameter adaptation of the unknown rotary accelerometer gain. This allowed a simple Least Mean Squares (LMS) algorithm that switched on only when there were sufficient levels of detected rotary disturbance for the algorithm to have guaranteed persistence of excitation [45] (Figure 8.27). The other difference was that I had programmed the Banshee Multivariable Workstation (BMW, Section 10.12.1) to sample the accelerometer 4 times as fast as the PES. This simple exploitation of a physical feature of the system dramatically improved my rejection bandwidth.

Years later this "physically inspired approach" would pay off again when I was doing some consulting work on disk drives. Again the problem was rotary disturbance rejection, this time by differencing two inexpensive linear accelerometers that might not be balanced, as diagrammed in Figure 8.23. The mismatch between the two, which could be in the range of ±15% limited the benefits of such a feedforward system [286]. Eric Miller had built a shaker system that inadvertently shook the drive with both rotation and translation. He was chagrined by this, as the translational disturbances, $a_T$, showed up in PES – which should not have happened with a balanced actuator. The "Aha!" moment was when we realized that the only way that $a_T$ showed up in PES was parasitically through mismatched accelerometers, and this provided the key to an augmented adaptation algorithm. In this, the input from translational disturbance, $a_T$, to PES was used to equalize the accelerometer gains while the input from rotational disturbance, $a_R$, to PES could then minimize the effects of rotational disturbances [188], as shown in the plots of Figure 8.28. While these experiences are by no means exhaustive, they do point once again to the retention of physical parameters, this time for adaptation, as a means of dramatically simplifying control problems.

## 8.10 Feedforward Control Summary

When I began writing this chapter, I figured I should throw in a few basic feedforward concepts, such as the concept diagram of Figure 8.1 or the specific block diagram of Figure 8.5. Then, as things seem to go with me, I remembered, "Oh, there was repetitive control. And there was feedforward from auxiliary sensors. And . . ."

Generally, we end up calling something feedforward control when we want to use it to help our servo loop, but we don't want it to affect the feedback loop much. These techniques end up being used more than one might expect in industry because they have loads of physical intuition attached and they can be added into a functioning feedback control system.

**I want to reemphasize a key unifying thread in all these different forms of "feedforward" additions to a feedback loop. While these might seem disconnected, one thing that all feedforward schemes share is that their correction signals have much less phase than a correction signal generated as a result of an error in a feedback loop. To paraphrase Meghan Trainor, it's all about the phase . . .**

## 8.11 Change Log for Chapter 8

- 2019_07_03: Added in an outline for chapter near the beginning. Increased emphasis on unifying feature of feedfoward methods being the lack of phase lag in the correction signal.

## 8.12 Chapter Summary and Context

This chapter has been all about trying to get a unified view of what is called feedforward control in the context that we are normally discussing feedback as our fundamental driver.

One of the key unifying concepts in all the different control schemes that get labeled as "feedforward", we are talking about something that – when properly tuned – applies control signals with little or no phase lag. This is a very powerful realization in that it means that the control signal itself can be far more aggressive than what we would do if we had to worry about gain and phase margins. There is a catch, though, and this catch is that this aggressive correction depends on model accuracy. So long as we have an accurate (and representative of reality) model in the regions for which the signals will be applied, then this stuff is pretty great.

What this means is that if we take the same diligence to deriving the models we use in our feedforward methods as we do for those we use in our feedback methods, we have a very good chance of augmenting the controller performance, while leaving the feedback controller free to handle those things that cannot be predicted.

# Chapter 9

# State Space: The Good, the Bad, and the Practical

## 9.1 In This Chapter

In this chapter we will discuss state-space methods for control analysis and design. However, as there is a huge amount of material already published on the subject, this chapter will first try to give insights into what state space originally intended for (or sold as) and how it is used now.

We will introduce state-space control in a simple way that leads us to the methodology of full-state feedback. While feedback from every state in the system is great, the control signal still has to pass through the available system inputs, which leads us to the concept of controllability. Since we can only rarely directly measure every state in a system, it is almost always necessary to to reconstruct/estimate the state from available measurements. This leads observers and the concept of observability.

Observers are model-based measurement devices, and so this leads naturally to a tutorial on model-based measurements. The issue with these is that the treatment is very different across different applications. The observers take several forms, depending upon whether or not we know the input to the physical system. Again, we return to the filtering versus feedback views of the world discussed earlier. Include feedback versus filtering view in introduction, then reference that here. We will try to present a more unified view of these. Model-based measurements are very dependent upon the model, not only the accuracy of the model in representing input-output data, but also the internal parameterization from which we get any insight (or not) into the internal workings of our physical system.

From an implementation perspective, we will stick with computer-based, discrete methods. For modeling purposes, we will return to continuous-time representations, but as we are focused on implementing control methods in this book, we will assume that the state-space implementation will have to be digitized and will have to deal with "all that computer stuff".

All that is well and good, but the devil is in the details and the details get messy. The "details portion" have most recently been dominated by mathematics – matrix mathematics – in the texts. However, from a practical point of view, much of this severely affects the connection between the physical understanding and the discrete-time model states. That physical connection brings intuition and helps us debug our implementation. The normal treatments leave us stranded here. The back end of this chapter deals with this via state-space representations that strongly preserve physical intuition.

One more point made repeatedly throughout this book gets doubly emphasized in this chapter: modeling. State-space methods are model-based methods and model-based methods depend upon . . . wait for it . . . a good model. This seems like an absurd point to have to reiterate, but experience has shown that one of the key factors limiting the practical use of state-space methods is that folks are so focused on implementing "big ideas" that they do not go through the mind-numbing grind needed to get a computer based model that is both representative of the system behavior and can be readily debugged.

## 9.2 Chapter Ethos

Going back to the start, explaining dynamic systems starts with physical observations that are compressed into models via (usually for our purposes) differential equations. What we learned in those college math classes was that the most general differential equations were very hard to solve analytically. First-order equations were relatively easy and intuitive. Many second-order equations were difficult although we could still make progress with special cases, such as linear, constant coefficient systems.

From here, we found that to handle higher-order equations, we usually needed to either use transform methods or recast the equation as a system of first-order differential equations. That system turned the differential equation into a linear algebra problem. "And then came the matrices . . . "

When we move to state-space methods, we move to those systems of first order differential equations. To handle this in a digital computer, we approximate the system model with a discretized version of the system of differential equations.

Why go through all of this work? The three most obvious reasons are:

- If one could precisely measure each of the states in our system and apply feedback from all these states, one could (in principle and with certain assumptions) easily control the system arbitrarily well. There are reasons – both technical and practical – why we can't actually live this dream. There are specific technical conditions that we must meet in even the most ideal circumstances.
- Even if we cannot directly measure each of the states, we can often (again subject to certain technical conditions in even the most ideal circumstances) estimate/recreate the behavior of the states from signals that we can measure. If we are using this estimates for state-feedback control, then we must remember that almost always we want to estimate the states at a far higher bandwidth at which we want the full closed-loop system to operate
- This reconstruction of internal signals/states promises us something not easily accessible from transform (a.k.a. transfer function) methods: the chance to reconstruct the inner workings of our physical system based upon our available measurements and a model of the dynamic system.

There are lots of names for this reconstruction: estimator, observer, even the overly specific "Kalman Filter". We will tend to use the term, observer, since estimator is used in a lot of other contexts and Kalman Filter is a very specific form of observer. In the classic sense, an observer is essentially a model-based simulation of the system, where the simulation states are corrected by comparing the model/simulation outputs with the corresponding measured outputs of the physical system. We're going to pause here to take all that in:

- First of all, the model-based measurement supposedly gives us an understanding of the internal workings of the system. However, this internal understanding is entirely dependent upon the internal states and dynamics of the model having some connection with the physical system. Many versions of a model can give the same input-output behavior.
- The model quality matters. We cannot emphasize this enough. Any attempt to use a model to understand the workings of a dynamic system depends upon how well that model captures the behavior of that dynamic system.

- Deriving an accurate model from a combination of first principles and measurements of an actual physical system is hard. It is a massive data compression problem almost as soon as we are away from textbook examples.
- However, the benefits of having a good model (modulo all the work it takes to get to one) may very well make it worth it. If we didn't have at least some hope that this was true, we would dismiss state-space methods forever.

State space (SS) is the way to do things "right" in academia and yet seems to have limited used use in industry, depending usually upon the size and expense of the system, and of course, the number of resonances that have to be dealt with. To get industry folks to use SS, there has to be a clear set of advantages, and there has to be a clear path around the gotchas. It has always been pitched as a way to handle MIMO systems, but many MIMO systems are really handled as a decoupled set of SISO systems. To really make the case, we need to show the specific advantages of state space over transfer function methods in practical, ordinary SISO systems:

a) As noted above, state space potentially gives us a structural look inside the dynamics of a system having order $> 2$, beyond the ones that we can typically measure with sensors, and so the state space model should be realized in a way that the discrete form represents physical reality. (Note: This is almost never the case for any standard discretization method. It is a big part of why this chapter pushes the biquad state space (BSS) structure.)

b) State space can allow us to build a model framework for things like missing samples in a system or integrating together multiple sensors of different rates and qualities.

c) State space can allow us to embed nonlinear relationships in the states that are more difficult to represent by transfer functions. These modifications are more often useful in modeling and simulation of the plant than in controls analysis, but perhaps useful as non-linear observers (estimators).

At the same time, on the "non-fighter-jet problems", i.e. those for which one cannot assign a team of engineers to tune every single device, the use of state space is limited in physical systems. The exceptions seem to be:

1) Systems that are adequately characterized by simple second order models.

2) Expensive "fighter-jet" systems.

3) Slow dynamic systems for which analyzing with analog models and “sampling fast” work.

The common thread for all these systems is that the designers can get an adequate model, either by system slowness/simplicity or by massive effort. We would like a more graceful way to do this for most of our systems.

For these systems, a plant model is derived, typically via discrete-time model, time domain identification (ID) or via fast Fourier transform (FFT) based frequency response methods followed by a curve fit. Because these measurements lack the signal-to-noise ratios (SNR) of the stepped sine, there are a lot of noise blips that get included into the models. For some systems, it is not uncommon to have 30 or 40 states in a physical system that really might just have a rigid body and 3 dominant flexible modes.

At this point, the high-order model is unsuitable for control design and some form of numerical model reduction is done, typically by finding the principal components of the model – a.k.a. something singular value of the state-transition matrix related – and removing those that fall below a numerical threshold.

It is important to note that at no point in this process have I mentioned the physical model parameters entering into the model (except if one were to want to back out their values from the model-reduced discrete-time parameters). This is the issue that I have. The physical intuition vanished long before the first model reduction step. In fact, if one were to change the sample rate, there would be very little intuition about the relationship between one discrete time, model reduced model and one at twice or half the sample rate. I believe that this is the property that makes the use of state space a rarity rather than standard practice in physical systems.

The structure of this chapter is as follows:

- A high level, very physical example used to teach high school STEM students the ideas of feedback [291, 292]. It is the high level look at what we can do with full state feedback and why we need estimators (Section 9.3).
- A tutorial on model-based measurements, which is another way of describing state-space measurements. This section goes through the typical state-space structures, but adds a bit of completeness about what they are all about and why some things that are called the same thing are so structurally different (Section 9.7).
- A discussion on why actually implementing this stuff is a lot harder than it was in that first graduate

school linear systems class. Actually, this is an ongoing discussion, but it starts around (Section 9.10).

- An adaptation of the multinotch filter structure to state space is first discussed for a discrete-time implementation (Sections 9.15, 9.16, and 9.17). There are some patterns to the matrices, and those are discussed in Section 9.18.
- We put this structure together into a current estimator for state feedback in Section 9.19.
- In order to model actual structures, we need some rigid body models that are nor usually needed in filters such as the multinotch [54]. A first cut at this is made in Section 9.20,where we discretize a double integrator in a way so as to make it compatible with the rest of the BSS structure. We show examples of how well this models a mechatronic lab system in Section 9.21.
- Section 9.12 describes how to add integration into state space via the use of auxiliary states.
- Section 9.13 covers the often ignored subject of how to inject a reference input into a state-space controller.
- In Section 9.22, we start the move to discussing the analog BSS by first discussing continuous-time biquads. This is followed in Section 9.23 which describes the Analog Biquad State Space Form.
- Because we will almost certainly run these in a computer, it makes sense to discuss discretization of the Analog BSS (Section 9.24). We discuss some of the structural properties of the matrices that make a difference here in Section 9.25.
- Now, for discrete-time models, we were able to trick our way into adding a double integrator using a Trapezoidal Rule Equivalent which maintained the direct feedthrough that is characteristic of the BSS structure, but we cannot guarantee direct feedthrough in an analog/continuous structure. Thus, we discuss continuous-time rigid body dynamics and low pass filters in Section 9.26. In particular, we need to understand what handling the lack of direct feedthrough means to our structure (Section 9.27.
- The limitations of biquads, particularly in extracting the interior states of these rigid body systems has inspired a different structure that allows better access to what would be the interior states of a biquad. The bilinear state space (BLSS) form (Section 9.28, [5]) is particularly useful when the dynamics of this second order block are not complex, and we want to track the signals. As with biquads, we need to make some discretization choices in our structure (Section 9.29.
- We can then use the BLSS to generate new discrete-time rigid body models (Section 9.30). There are some examples of this and what kind of interior signal looks it provides (as opposed to a pure biquad) in Section 9.31.

- Section 9.32 provides some continuous time biquad examples, showing how closely the states of the continuous and discrete time BSS structures correspond. This is – as Joe Biden might say – a big f***ing deal.
- Finally, we summarize the chapter and what we believe are the key advantages of the biquad state space (BSS) structure in Section 9.33.

## 9.3 State Space Control for High Schoolers

An introduction to control principles for high school level students can be generated starting with the physical schematic of a double integrator in Figure 2.11, which gives the block diagram of Figure 9.1.

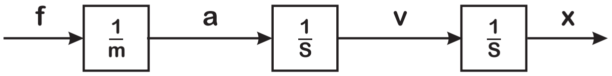


Figure 9.1: The block diagram of the double integrator.

Modeling spring and damper feedback gives us the schematic of Figure 2.14, repeated in Figure 9.2.

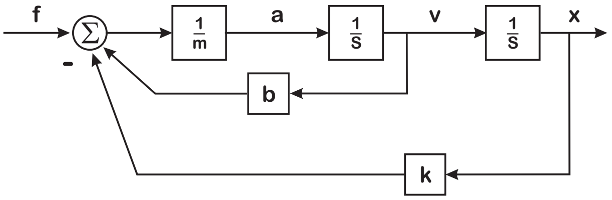


Figure 9.2: The block diagram of the double integrator with velocity and position feedback.

A great way to take a step back and understand state space control is from an example used to introduce control mathematics to high schoolers. We start with a simple double integrator diagram of Figure 9.1 and add spring and damper terms feeding back from the two integrators as in Figure 9.2. We are all familiar with both of these systems, but viewed this way, we have gone from an open-loop double integrator to one with feedback from both energy storage states in the model. That is, we have full state feedback. The characteristic equation is defined as:

$$\begin{array}{ccc} m\ddot{x} = f - b\dot{x} - kx & \longleftrightarrow & \dfrac{X(s)}{F(s)} = \dfrac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}}, \\ \text{Time Domain} & & \text{Transform Domain} \end{array} \tag{9.1}$$

where we see these feedback terms showing up explicitly in the time domain equation and this is transformed on the right into a relationship which we can introduce to them as a transfer function. On the transfer function side, they see that if we set $k$ and $b$ to $0$ we are back at our double integrator case. Furthermore we can relate the second order relationships of $k$ and $b$ and $m$ to those of an oscillatory system by matching coefficients in Equation 9.2 which relates the spring and damper parameters to that of a simple resonance:

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}} = \frac{K\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2}, \tag{9.2}$$

where

$$\sqrt{\frac{k}{m}} = \omega_d = 2\pi f_d \Longleftrightarrow f_d = \frac{1}{2\pi}\sqrt{\frac{k}{m}}, \tag{9.3}$$

and

$$\frac{b}{m} = 2\zeta_d\omega_d \Longleftrightarrow \zeta_d = \frac{b}{2\sqrt{km}}. \tag{9.4}$$

Now, we have gone, in very straightforward steps, from a double integrator system to one with feedback from the outputs of both integrators. We have shown using some pretty basic math that we can model the behavior of this very physical system. Equations 9.3 and 9.4 tell us about the behavior:

- $k/m$ tells us how fast it rings.
- $b/m$ (in relationship to $k$) tells us how long it rings.
- Making $k$ bigger means the spring is stiffer, which results in higher frequency ringing.
- Making $b$ bigger relative to $k$ causes the ringing to damp out faster.
- Because denominator polynomial is $2^{\text{nd}}$ order, we can get roots with quadratic equation.
- Any polynomials that are more than $2^{\text{nd}}$ order are a lot harder.

To illustrate the changes we can make by changing $k/m$ and/or $b/m$ we can show some relatively straightforward plots in Figure 9.3. Since we have described the damping and the oscillatory frequency

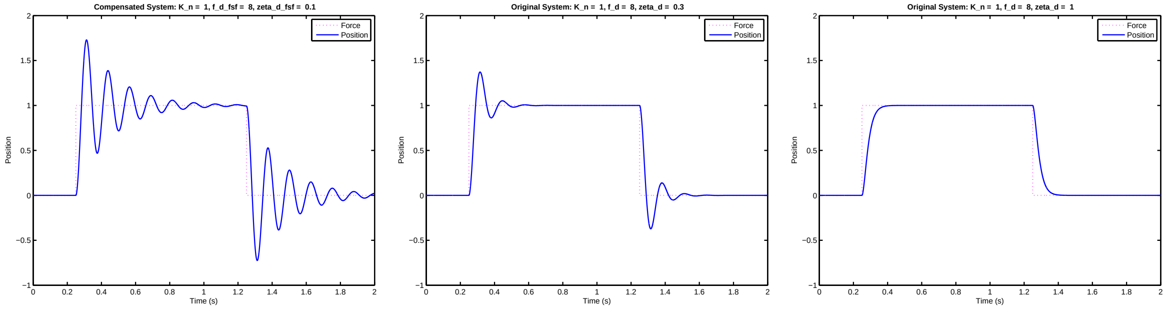


Figure 9.3: Double integrator with spring and damper feedback. Resonant frequency ($\omega_d$) at 8 Hz. We show the effect of changing the damping ($\zeta_d$) from 0.1 on the far left, to 0.3 in the center, to 1 on the far right.

as functions of $b$, $k$, and $m$, we can show how changing their relationship can change the damping and dramatically change the behavior of the system. We have introduced these as properties of the physical system itself. We are trying to understand/model the behavior with these equations and plots to gain insight.

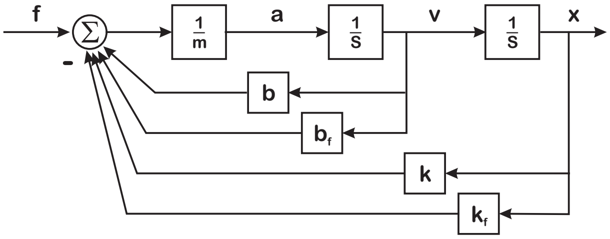


Figure 9.4: Adding our own feedback to the spring-mass-damper system.

At this point, we have shown them the effects of nature's feedback parameters on the behavior of our simple system, and it is natural to ask, "What if nature's $k$ and $b$ are lame? Can we compensate?" The obvious answer is to introduce augmented feedback into the system as shown in Figure 9.4. We can describe this as adding our own signals to feed back the output of each energy collector (which we call a state) back into the input of the system. We can analyze this to pick our parameters by looking at modifications of Equations 9.2–9.4.

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b+b_f}{m}s + \frac{k+k_f}{m}} = \frac{K\omega_d^2}{s^2 + 2\zeta_{df}\omega_{df}s + \omega_{df}^2}, \tag{9.5}$$

where

$$f_{df} = \frac{1}{2\pi}\sqrt{\frac{k+k_f}{m}} \text{ and } \zeta_{df} = \frac{b+bf}{2\sqrt{km}}. \tag{9.6}$$

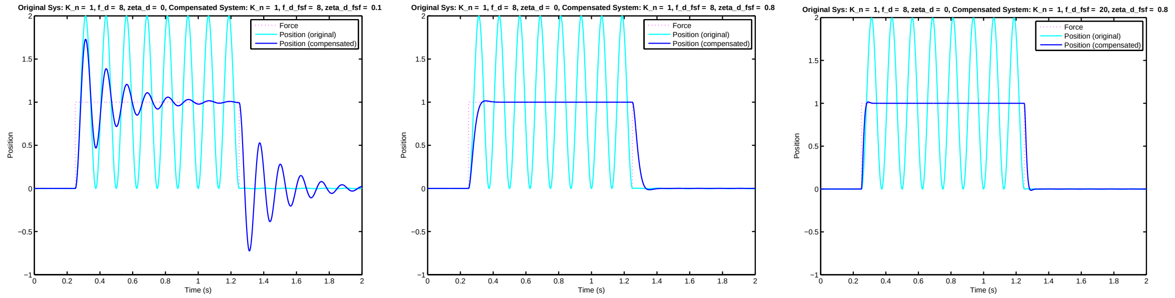


Figure 9.5: Our spring mass damper with $\zeta_d = 0$ and $f_d = 8Hz$. The cyan curve shows this response, which rings without stopping. The blue curve shows the response of the system to the same input, when we humans have augmented nature's feedback. We then use our augmented feedback to change $\zeta_d$ to 0.1 (far left), $\zeta_d$ to 0.8 (center), and even change the frequency, $f_d$ to 20 Hz.

The point is that this is very straightforward way that we can augment nature's feedback with our own. This is called "full-state feedback", which is the proverbial "800-pound gorilla" [293] of control. Full-state feedback, direct feedback from every energy storage node of the system (or state), is powerful because we can drive the roots of the closed-loop characteristic equation to anything we want. It is also relatively simple, conceptually. Now, for things beyond second order continuous time, linear time-invariant (LTI) systems, it gets more complicated, but most state-space theory is taught from an LTI perspective.

The point of this is to illustrate how we can manipulate the feedback loop almost trivially if we can access all the states and do full-state feedback. Almost all of the work in making state space methods work is about getting to some approximation to that. The other point of the above is to see the amount of physical intuition we can get into a feedback loop if we have this kind of a decomposition and are looking at physical parameters. Unfortunately, there is less work done for this latter part, but we will try to show a way out of that later in the chapter.

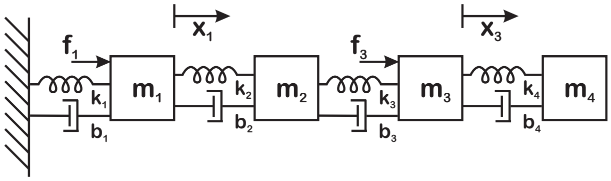


Figure 9.6: Extending our spring mass damper system to one with a lot more springs, masses, and dampers.

Although we could spend more time on the theories of how and where to place the poles, the really

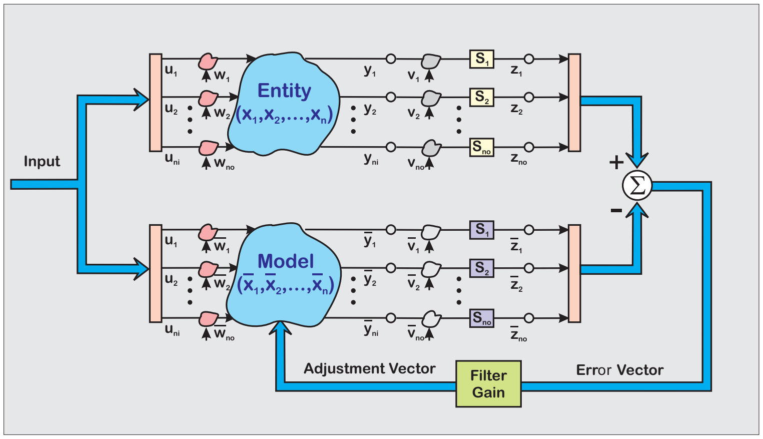


Figure 9.7: A generalized view of model based filtering. On top is a conceptual view of the physical system, being driven by a set of inputs (that we may or may not know). At the other end are outputs which we can measure. On the lower end is our simulation (a.k.a. model of the system). Our model is driven by the same inputs that we know and measured at the same outputs that we can measure on our physical system. The key is forming an error and using this error to correct the simulation to more closely predict the output of the physical system.

critical issue is that we almost never can feed back from every state because we do not have a perfect measurement of every state. Looking at the example of Figure 9.6, we can ask:

1) What if we can't measure every "state"?

2) What if physical system is more complicated than our model?

The trick is that since (2) is always true, then (1) is always true. This allows us to discuss really big, big systems with lots of stuff that we can't model exactly. Power grid, air traffic control, automated highways, systems biology, have many more things (states) than can be measured. The general process steps are:

- Build a simulation.
- Run that simulation in parallel with real world.
- Compare the simulation output(s) to the output(s) that we can measure in real world.

- Correct the simulation with measurements from the real world.

This is shown schematically in Figure 9.7. In doing so, we get a lot of advantages, including that the "inside" of our simulation will have useful information we could not measure in the real world, and even try some things that we might not dare try on a real world problem. The simulation corrected with real world estimates is an estimator. Building an estimator should allow us to try full-state feedback from the estimated states. Conceptually, we have covered the issues of state-space systems. What really comes up next is how we build them to make them useful.

It should come as no surprise that the quality of what one can do with state space control is governed by a handful of things:

1) How well the model of the system used in the "simulation" actually captures the critical behavior of the system.
2) How clean the measurements feeding the model corrections are.
3) How fast we can run the model and the measurements, relative to the critical time constants of the system.

Pretty much everything one can do with state space is limited by these because state-space filtering and control is model-based filtering and control and one cannot do good model-based filtering and control without a good model. Sad but true.

## 9.4 A Note on Notation

It is said that folks can often tell where a control engineer went to grad school by their choice of state matrix names. In one camp, state-space matrices are $\{A, b, c, d\}$ for SISO systems and $\{A, B, C, D\}$ for MIMO systems. Whether these matrices correspond to continuous or discrete models is often taken from the context. We have:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{9.7}$$

for continuous time and

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{9.8}$$

for discrete time. In another camp, state-space matrices are $\{F, G, H, J\}$ for both SISO and MIMO systems. However, for moving to discrete-time systems, the authors use the so-called Roman-to-Greek transformation where the state matrices for discrete-time representations become $\{\Phi, \Gamma, H, J\}$ for both SISO and MIMO systems. One might note that $H$ and $J$ don't go through the change in ancient alphabets transformation as they do not involve any time progression of the states and therefore do not need to transform (except that the states have been likely changed going from continuous to discrete form). Thus we have

$$\begin{aligned} \dot{x} &= Fx + Gu \\ y &= Hx + Ju \end{aligned} \tag{9.9}$$

for continuous time and

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Hx(k) + Ju(k) \end{aligned} \tag{9.10}$$

for discrete time.

I personally find that the $\{A, B, C, D\}$ matrices get used in a whole lot of contexts and is a bit confusing out of context while I am not fond of typing all the escape sequences to get to Greek letters for the $\{F, G, H, J\}/\{\Phi, \Gamma, H, J\}$ notation. I also thing that $J$ gets used a lot in state equations and transfer functions so, I prefer the $D$ for the seldom used direct feedthrough. Instead, I have found it less confusing to use $\{F, G, H, D\}$ for the state matrices and then add a $C$ or $D$ subscript if the context between continuous and discrete needed clarification. This gives us:

$$\begin{aligned} \dot{x} &= F_C x + G_C u \\ y &= G_C x + D_C u \end{aligned} \tag{9.11}$$

for continuous time and

$$\begin{aligned} x(k+1) &= F_D x(k) + G_D u(k) \\ y(k) &= G_D x(k) + D_D u(k) \end{aligned} \tag{9.12}$$

for discrete time. When the context is obvious, we can drop the $C$ and $D$ subscript and make both the typing and the reading a bit easier. Sometimes, the time indexes will also be subscripts, i.e.

$$\begin{aligned} x_{k+1} &= F_D x_k + G_D u_k \\ y_k &= G_D x_k + D_D u_k \end{aligned} \tag{9.13}$$

because sometimes that makes it easier to parse when reading.

## 9.5 The Separation Principle)

One of the things that makes state-space methods in any way tractable is the Separation Principle (cite Franklin's student?), which states that one can design a state-space controller by separately designing the observer and the state feedback separately. From a practical point of view we can only control as well as we measure and we must understand that the estimator is a filter from the measurement to the state. As such, we cannot have a control bandwidth that is faster than the observer/estimator bandwidth. In fact, it is reasonable to see the observer/estimator bandwidth several times the desired closed-loop system bandwidth so that the phase lag from the observer/estimator filter does not adversely affect the desired closed-loop bandwidth.

Proof is quite simple and on Wikipedia, so might show it here. End up with block triangular matrix with the closed-loop dynamics in one diagonal block and the closed-loop observer dynamics in the other diagonal block.

## 9.6 Full State Feedback and Its Evil Twin (the Dual Problem)

In the high school explanation of Section 9.3, we mentioned that directly measuring every state in the system and feeding these back would – in principle and with a linear, time-invariant (LTI) model – allow us to arbitrarily set the new closed-loop system dynamics. Very specifically, under certain conditions, directly feeding back all the states – called full-state feedback (FSFB) – can allow us to place the closed-loop poles of the dynamic system arbitrarily. Full-state feedback is the proverbial "800 pound gorilla" of control.

To be sure, if we could directly measure each state and inject the right feedback signal based on that measurement directly into the state, we would be able to do this. However, even our simple high school example of Section 9.3 had all the feedback signal going through a single input into the system. The property that allows us to still get the benefits of FSFB through a limited number of inputs is called controllability. Controllability is usually discussed in terms of matrix math (assuming again an LTI model), but we want to start more physically. Controllability asks: With this (these) input(s) and these state equations (system dynamics), can we get from here to there? Rephrasing, the fundamental question of controllability becomes: from the inputs we have, can we drive this dynamic system's states arbitrarily? The more rarely asked, more nuanced question one might ask is: How much effort is required to do that? How controllable is the system? For a specific example we might ask how much control effort it takes to move a railroad trestle's natural frequency from 1/100 Hz to 1 kHz?

The dual of this problem involves reconstructing the full state from a limited set of measurements. That is, since we cannot always – or even usually – measure all the states of a system, the observability question asks: from a given set of measured outputs, can we fully and accurately reconstruct the entire state? A follow on is whether or not we can do this as quickly (with as much bandwidth) as we would like? This follow on becomes significant when we are trading off between trying to filter out noise and trying to have an observer bandwidth far faster than the desired closed-loop bandwidth.

For the full-state feedback problem, controllability implies being able to set all the poles of the closed-loop system (assuming an LTI model) to arbitrary locations. For an observer, observability implies being able to set all the poles of the closed-lop observer (assuming an LTI model) to arbitrary locations: Three things:

- The controller problem is affecting the real physical system while the observer problem exists only inside the controller. In modern times, that further means that the controller is on a digital computer so the observer resides entirely inside a computer.

- Of course, if we use that observer to generate states for full-state feedback (a pretty common usage), then our observer dynamics do affect the closed-loop system dynamics.

- Finally, as much as anything in control relies on the quality of the model, state-space methods, both on the controller side and the observer side, rely dramatically on the accuracy of the model used for the system. (Yes, I am repeating this point incessantly because it is one of the universal failure points in trying to use state-space methods for anything other than proofs and simulations.)

- Noise; we always have some uncertainty/noise in our measurements and in the inputs driving our physical systems. We minimize the effects of noise by filtering (which is usually weighted averaging). The more we filter, the more time delay we add to the signal chain. In filtering contexts, we average away without worry. In feedback contexts, that delay eats phase margin, so there is always a tradeoff.

Okay, that was four.

### 9.6.1 System versus Realization

We have played fast and loose with the idea of a state-space **system**. While there is a physical system out there, that physical system can be represented in many ways. We can worry only about the input-output mappings in the transform domain (transfer functions) the input-output mappings in the transform measure (frequency) domain (frequency-response functions), or in the time domain, but even in the latter there are many, many different models that will give us the same input-output mappings.

Because of this, it is more accurate to talk about a state-space realization. A realization is simply one embodiment, one version, one parameterization of a system model from which we have mapped input-output behavior. A large part of the allure of state space is the idea that we can swap between different realizations using matrix math (linear algebra again). While this is absolutely true and often very useful, we want to repeat that moving a realization too far from something that is physically intuitive weakens our ability to debug the physical system.

### 9.6.2 Full State Feedback with an LTI Model

Almost always when the term, full-state feedback (FSFB) is used, we have assumed a model (realization) of the system and are using that state to generate our feedback signal. In the case of an LTI model, we have the state matrices defined as in Equation 9.11

$$\dot{x} = F_C x + G_C u. \tag{9.14}$$

For controllability, we are dealing with the input-to-state equation, not the output equation, so only the part in Equation 9.14. This defines a system of differential equations that constitute a realization of the dynamic system. The behavior of the homogeneous part is defined by the $F_C$ part while the particular solution (how the input gets into the states) requires both $F_C$ and $G_C$.

In discrete time, the most common form of the equations we will use are from Equation 9.13, where we again only care about the input to state portion:

$$x_{k+1} = F_D x_k + G_D u_k. \tag{9.15}$$

Here the model is predicting what the next state value will be from the current state and current input. How we get from $\{F_C, G_C\}$ to $\{F_D, G_D\}$ is all about discretization and as we will see later, the realization we choose can have a strong effect on how we discretize the realization.

Whether continuous or discrete, one way to check controllability is with the Controllability Gramian:

$$C = \left[ \begin{array}{cccc} G & FG & \cdots & F^{n-1}G \end{array} \right], \tag{9.16}$$

where there are $n$ states in the system. Note that we are not concerned with $H$ or $D$ as they do not affect the input-to-state results. We have also dropped the $C$ or $D$ subscripts, since the structure is the same for both continuous and discrete domains (although the matrices are clearly different). The basic check is that $C$ has to have full rank for the realization to be controllable. One interpretation of this is that each column of $C$ can be viewed as a step further into the realization states where the powers of $F$ either get us extra (filtered) derivatives or extra (filtered) time steps. If the matrix has full rank (nonsingular for SISO systems) then you can move to every state from the designated input(s).

### 9.6.3 The Dual Problem: Observers

From a physical system perspective, it is absurd to think we might be able to measure every state of the system. If we look closely enough, most systems are not really linear, not really time invariant, and not defined by a finite number of energy storage or vibrational modes. We just have to limit how closely we look to get anything done. Perhaps in certain well-engineered systems, we could measure all the critical storage/vibrational modes in the system, but generally measurements are opportunistic: What signals can we measure and what signals are critical to implementing feedback well?

Generally, this means we have a lot fewer measurement points than states, so this brings about the observability problem: From our limited set of measurements, can we fully and accurately reconstruct the state of our system realization? There are caveats hear, but this is the controllability problem flipped around. The measured outputs are defined by the lower portions of Equations 9.11 and 9.13, i.e.

$$y = G_C x + D_C u \tag{9.17}$$

and

$$y_k = G_D x_k + D_D u_k. \tag{9.18}$$

Note that there are no derivatives or steps forward in time,. Instead, the measured output(s), $y$, is a function of both the current state, $x$, and the input(s) to the system, $u$. Now, in many texts, the direct feedthrough term, $D$, is missing because there is an assumption that the system is inherently low pass eventually, and this will show up in the model as a lack of direct feedthrough. However, it is very helpful here to think in the frequency domain: Although we know that any physical system short of the Big Bang must have finite bandwidth (and therefore a zero $D$ matrix) it is entirely possible that in any bandwidth of interest, we might have direct feedthrough. We just are not modeling out to a high enough frequency to see the roll off. In this case, we have a non-zero $D$ matrix).

Even with a non-zero $D$ matrix, observability is about being able to extract the state information from the output and so in an LTI context is independent of the input. This means the direct feedthrough matrix, $D$ plays no role. The observability Gramian, $\mathcal{O}$ is defined as:

$$\mathcal{O} = \left[ \begin{array}{cccc} H^T & F^T H^T & \cdots & (F^T)^{n-1} H^T \end{array} \right]. \tag{9.19}$$

Practically speaking, if the contribution of the direct feedthrough to the output dominates the state-to-output signal, we may ave some issues separating out the components in finite time.

### 9.6.4 Some Simple Controllability/Observability Examples

It is useful to share a couple of trivial examples that illustrate some of these concepts without the Gramian matrices.

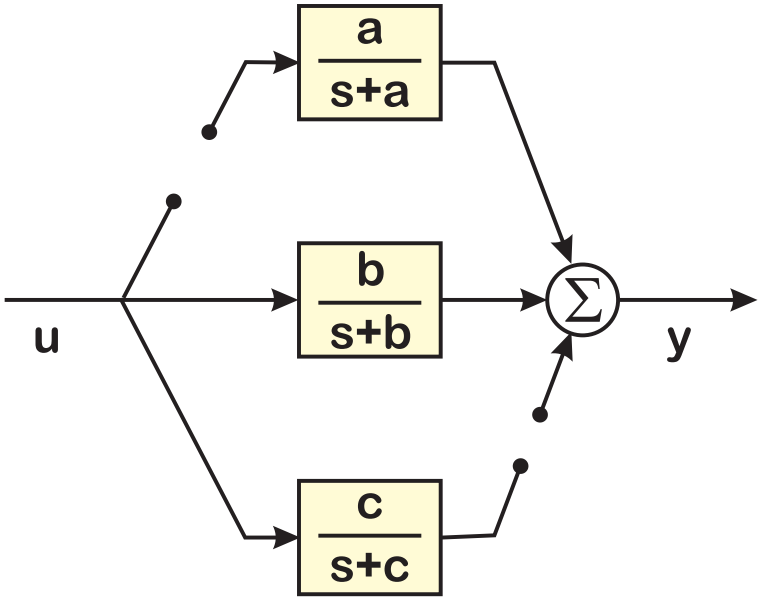


Figure 9.8: A simple controllability/observability example.

The system in figure 9.8 has an input-output transfer function of

$$\frac{Y(s)}{U(s)} = \frac{b}{s+b}, \tag{9.20}$$

but this realization has three states:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -a & 0 & 0 \\ 0 & -b & 0 \\ 0 & 0 & -c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ b \\ c \end{bmatrix} u \tag{9.21}$$

$$y = \begin{bmatrix} a & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{9.22}$$

where $a, b, c > 0$. In this trivial example, we see that:

- State 1, $x_1$, is not controllable from $u$, but is observable from $y$.
- State 2, $x_2$, is both controllable from $u$ and observable from $y$.
- State 3, $x_3$, is controllable from $u$, but not observable from $y$.

From a transfer function perspective, we have a first-order system, but from our state-space realization, we know that this really has three separate dynamic states. Perhaps we can observe State 1, but it has minimal effect on the output, so we do not worry that we cannot affect it with our input. Perhaps we cannot observe State 3, but know from first principles or measurements during manufacture, that it has no major effects on our system, so we need not worry about any parasitic coupling from our input to the state. On the other hand, we may find that pushing the closed-loop bandwidth also affects State 3 in a way that does not show up at our primary sensor, but manifests itself in damage to the system. This might prompt us to get another sensor on the system so that we can directly observe State 3. Perhaps we observe that State 1 dominates the response, but we have no way of doing anything about it. Maybe that makes us decide to add another actuator so that we can affect State 1.

Another simple example is as follows. The system in figure 9.9 has two states. We can give it a transfer function of

$$\frac{Y(s)}{U(s)} = \frac{2a(s+a) - \varepsilon^2}{(s+a-\varepsilon)(s+a+\varepsilon)}, \tag{9.23}$$

where $a \gg \varepsilon > 0$.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -a-\varepsilon & 0 \\ 0 & -a+\varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} a+\varepsilon \\ a-\varepsilon \end{bmatrix} u \tag{9.24}$$

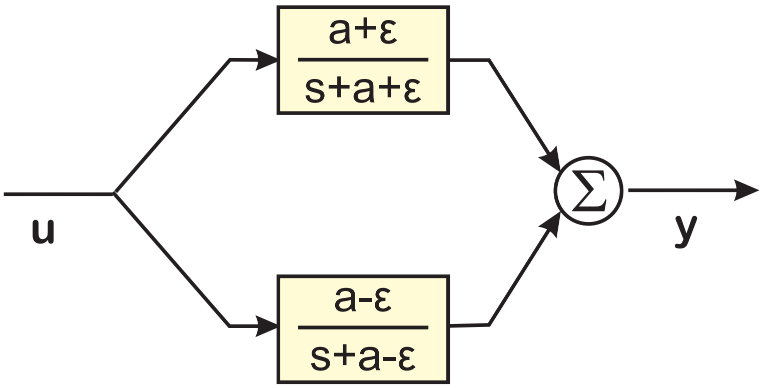


Figure 9.9: Another simple controllability/observability example.

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{9.25}$$

Technically, both states are controllable from $u$ and observable from $y$, but **how much** they are really depends upon the value of $\varepsilon$. When $\varepsilon = 0$, we can affect both $x_1$ and $x_2$ from $u$, but not separately. Similarly, both states are observable as a tandem, but not individually. We cannot separate these two using only one measurement when Equation 9.23 reduces to:

$$\frac{Y(s)}{U(s)} = \frac{2a(s+a)}{(s+a)^2} = \frac{2a}{s+a}. \tag{9.26}$$

Equation 9.26 shows our issue for controllability and observability: while the structure of Figure 9.9 shows two distinct states, we cannot separate one from the other once $\varepsilon$ gets small enough. As $\varepsilon$ moves away from $0$, the two roots become distinct. Still for values of $\varepsilon \ll a$, how distinct is the real issue. Both states might be separately controllable and observable, but that might be something that requires a lot of time or samples or massive inputs to sort out. If we factor in uncertainty at both the input (often called process noise) and the output (often called measurement noise) as well as quantization and finite word length issues,, the separation requirements on $\varepsilon$ grow quite a bit. This is something to consider when models are technically controllable or observable, but really not for all practical purposes. Again, maybe this is not an issue if we are only happy to affect the input-output response and not worry about the individual states. After all, the transfer function of Equation 9.26, especially with $a > 0$ should be an easy one to adjust.

The key takeaway from both of these examples is that it is important to know the context before we can determine how important controllability and/or observability of a particular state-space realization is to us.

## 9.7 A Model Based Measurement Tutorial

In some ways, state feedback seems simpler than state estimation, as there is only one real step. If our state is denoted by the vector, $x$ and if our controller output is a function of the states, $u = -Kx$, then we are simply picking $K$ based on our model of the system and where we want the closed-loop dynamics to end up.

For state estimation, we actually have to run the model inside the controller. Furthermore, while it is rare to try state feedback without an estimator, it is relatively common to have state estimators (observers) that are not used for state feedback.

The next section will present an overview tutorial on state estimation, otherwise known as model-based measurements.

We enter into state space by pulling our $n^{th} - order$ LTI differential equation into a system of $n\ 1^{st} - order$ differential equations, and then packing them into a matrix form. If we are starting with a discrete-time LTI difference equation with a uniform sample period, we pull our $n^{th} - order$ LTI difference equation into a system of $n\ 1^{st} - order$ difference equations, and pack them into their own matrix form.

Now, referencing our high school spring-mass-damper example from Equation 9.1 the "state" represents the internal memory of the system, the energy storage if you will. So, the $x$ state is a vector of $x$ subterms, in this case position ($x$), velocity ($\dot{x}$), and acceleration ($\ddot{x}$) although the latter is not considered part of the state, but the derivative of the state. A bit confusing, but extra notation would slow things down and be confusing. Now, a set of state equations for our simple, continuous time, linear, time invariant system (CT-LTI) are:

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -\frac{b}{m} & -\frac{k}{m} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} \frac{1}{m} \\ 0 \end{bmatrix} f \Longleftrightarrow \dot{\bar{x}} = F_C \bar{x} + G_C u. \tag{9.27}$$

Here we have used $\bar{x}$ as the state vector to reduce confusion with the position and velocity terms, while the force input, $f$, is our general input, $u$.

These equations are a particular form that correspond to our physics equations and retain some physical meaning. However, because matrix equations can be transformed through linear transformations, it turns out that there are an infinite number of representations. This gets somewhat confusing to both novices and advanced folks as they get wrapped around the notation. For my purposes in this book, I will try to stay as close as reasonable to some sort of well structured physical model. Hopefully, what I mean by that will be clear in the what follows.

I've defined the state above, but to correspond to an equivalent transfer function, we need to also have an output. Outputs are defined by the output matrices, which can simply be considered a choice of where we put our sensor (or which measurements are available). The most natural measurement is position, which results in the following output equation:

$$\left[\begin{array}{c} y \end{array}\right] = \left[\begin{array}{cc} 0 & 1 \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] + \left[\begin{array}{c} 0 \end{array}\right] f \Longleftrightarrow y = H_C\overline{x} + D_C u. \tag{9.28}$$

However, we can just as easily use a speedometer or tachometer to measure velocity, resulting in the equation:

$$\left[\begin{array}{c} y \end{array}\right] = \left[\begin{array}{cc} 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] + \left[\begin{array}{c} 0 \end{array}\right] f \tag{9.29}$$

Finally, although this is rarely seen in textbooks, we might choose to measure acceleration. After all, accelerometers exist, and so now, we have a problem because the "state" does not contain acceleration, $\ddot{x}$. The derivative of the state does however and if we choose to measure acceleration, we can use the state equations to redefine our acceleration output as:

$$\left[\begin{array}{c} y \end{array}\right] = \left[\begin{array}{c} \ddot{x} \end{array}\right] = \left[\begin{array}{cc} -\frac{b}{m} & -\frac{k}{m} \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] + \left[\begin{array}{c} \frac{1}{m} \end{array}\right] f \tag{9.30}$$

Note that neither position, $x$, nor velocity, $v = \dot{x}$ have direct feedthrough from the input force, $f$. However, acceleration, which might be measured with an accelerometer, does. That means that the input force instantly appears at the acceleration output of the system.

This state space form makes use of several matrices, and so the above equations can be rewritten as

$$\begin{array}{rcl} \dot{\overline{x}} & = & F_C\overline{x} + G_C u \\ y & = & H_C\overline{x} + D_C u \end{array} \tag{9.31}$$

When it is obvious that $x$ is the state vector and not the horizontal position, we will drop the vector bar from $\overline{x}$ and make the reading simpler by using $x$.

Note that through all of this, I haven't discussed noise at the inputs or the outputs. I will when we start talking about generating observers in Section 9.7.3.

### 9.7.1 A More Generic Continuous Time Model

A more generic second order system would be described as:

$$\begin{aligned} \ddot{x} &= -a_1\dot{x} - a_2 x + u && (9.32) \\ y &= b_0\ddot{x} + b_1\dot{x} + b_2 && (9.33) \end{aligned}$$

with the following transfer function description:

$$\begin{gathered} \left(s^2 + a_1 s + a_2\right) = U(s) \\ Y(s) = \left(b_0 s^2 + b_1 s + b_2\right) \end{gathered} \qquad (9.34)$$

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} \qquad (9.35)$$

We could put this into a state space form as well:

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \Longleftrightarrow \dot{\bar{x}} = F_C \bar{x} + G_C u. \qquad (9.36)$$

and with the output as in the transfer function we have

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} b_1 - b_0 a_1 & b_2 - b_0 a_2 \end{bmatrix} \begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} b_0 \end{bmatrix} u \Longleftrightarrow y = H_C \bar{x} + D_C u. \qquad (9.37)$$

It is worth noting that if $b_0 = 0$, then the system is strictly proper with no direct feedthrough and $D_C = 0$.

At this point, we have the same form as a "controller canonical form" [14, 171]. In another place in this chapter, or in one of my others, I will show a very useful digression from this form.

This system will show some well known representations of physical systems in state space forms. Again, we have packed the continuous-time physical system model into a standard state-space model:

$$\begin{aligned} \dot{\bar{x}} &= F_C \bar{x} + G_C u \\ y &= H_C \bar{x} + D_C u \end{aligned} \qquad (9.38)$$

### 9.7.2 Discrete-Time Version of Spring-Mass-Damper System

“It could easily be accomplished with a computer.” – Dr. Strangelove [294]

Starting with the late 1950s, and becoming far more universal since the 1980s, control systems of any sophistication (and by that I don’t mean much) have been implemented using a computer. It is of course possible to implement feedback mechanisms with analog means, e.g. op amp filters, and these are certainly prevalent in high frequency applications, but the maintainability of computer-based systems gradually eats up all the applications at low frequency moving to higher and higher frequencies every year. This will be discussed in much more depth in Chapter 10 which deals with computation for control.

To use a model in a computer, we need a discrete time version. Now, there are lots of ways of going from continuous time representation to discrete time representations, just as there are many ways to numerically integrate a curve. In this section, I will use a Trapezoidal Rule equivalent [15], which is related to the Trapezoidal Rule numerical integration method. It has some niceties to it and some complications, which I will hopefully discuss more fully below.

We need to have discrete equivalents of our differential equations to work on computers (at least digital computers and almost nobody uses analog computers anymore). This means that differential equations become difference equations and instead of state-space equations generating the derivative, they generate the next step in the progression. This is the most common form for digital control formulations, known as the step representation, but there are others.

The difficulty is understanding how the continuous time, physical parameters map to discrete time parameters. This gets a lot worse as the system gets more complicated.

Also, one has to worry about sampling fast enough to capture the essential behavior of the system. This is codified in Nyquist’s sampling theorem or the Nyquist-Shannon sampling theorem [295, 296, 15], which says you need to look at the data at least twice as fast as the highest frequency you are trying to recreate (some details omitted here). In actuality, we need to look at the data 10 to 20 times as fast as it changes to do a good job. This is because the Nyquist-Shannon rate is asymptotic, based upon Fourier integrals which go backwards and forwards infinitely in time. As most of us are far more limited, and as our control systems need to be causal and avoid the time delay associated with going back to the Big Bang (really hurts the phase margin), we need to sample faster than this asymptotic result.

The bottom line is that in order to use the models of Section 9.3 or the more generic one of Section 9.7.1

in a computer based system, we need a discrete equivalent, something that captures the essential behavior of the continuous time system in a discrete time model. We need to go from differential equation form, to difference equation form. That is, we need to go from (9.38) to a form:

$$\begin{aligned} x_{k+1} &= F_D x_k + G_D u_k \\ y_k &= H_D x_k + D_D u_k \end{aligned} \tag{9.39}$$

A discrete transfer function version of (9.35)

$$\frac{Y(z)}{U(z)} = \frac{b_{0,D}z^2 + b_{1,D}z + b_{2,D}}{z^2 + a_{1,D}z + a_{2,D}} = \frac{b_{0,D} + b_{1,D}z^{-1} + b_{2,D}z^{-2}}{1 + a_{1,D}z^{-1} + a_{2,D}z^{-2}} \tag{9.40}$$

and because it has the same structure, it has an equivalent state space form:

$$\begin{bmatrix} x_{k+2} \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} -a_{1,D} & -a_{2,D} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_{k+1} \\ x_k \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k \tag{9.41}$$

and with the output as in the transfer function we have

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} b_{1,D} - b_{0,D}a_{1,D} & b_{2,D} - b_{0,D}a_{2,D} \end{bmatrix} \begin{bmatrix} x_{k+1} \\ x_k \end{bmatrix} + \begin{bmatrix} b_{0,D} \end{bmatrix} u_k \tag{9.42}$$

The question is what the meaning of the new coefficients in relation to the old ones, and this is entirely related to both the original parameters and the discretization method. Exact discretization methods obscure the coefficient meaning and couple states in a very non-intuitive way. Some other approximations, in which the original transfer function is broken into a cascade of second order sections (biquads) can preserve much physical intuition. We saw this with the multinotch of Section 6.11 and will see it again with the biquad state space of Section 9.17.

There are lots of discretization methods and even when one does the "exact" math, one doesn't get a satisfying answer. In fact, the exact math can give an answer that is so convoluted as to obscure any hope of physical intuition and this is bad. The Trapezoidal Rule, also known as Tustin's Rule or a bilinear equivalent, substitutes discrete time operators (based on the Z transform) for the continuous time operator (based on the Laplace transform). Using the Trapezoidal Rule, we make the substitution:

$$s \longleftarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) \tag{9.43}$$

and if we substitute for $s$ in (9.35) to get to (9.40) then we end up with the following mappings:

$$\begin{array}{rclrcl} \Delta & = & 1+a_1\frac{T}{2}+a_2\frac{T^2}{4} & & & \\ b_{0,D} & = & \frac{1}{\Delta}\left(b_0+b_1\frac{T}{2}+b_2\frac{T^2}{4}\right) & a_{0,D} & = & 1 \\ b_{1,D} & = & \frac{2}{\Delta}\left(b_2\frac{T^2}{4}-b_0\right) & a_{1,D} & = & \frac{2}{\Delta}\left(a_2\frac{T^2}{4}-1\right) \\ b_{2,D} & = & \frac{1}{\Delta}\left(b_0-b_1\frac{T}{2}+b_2\frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1-a_1\frac{T}{2}+a_2\frac{T^2}{4}\right) \end{array} \tag{9.44}$$

For the simple spring-mass-damper system of (9.63), we end up with

$$\begin{array}{rclrcl} \Delta & = & 1+\frac{b}{m}\frac{T}{2}+\frac{k}{m}\frac{T^2}{4} & & & \\ b_{0,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{0,D} & = & 1 \\ b_{1,D} & = & \frac{2}{\Delta}\left(\frac{2}{m}\frac{T^2}{4}\right) & a_{1,D} & = & \frac{2}{\Delta}\left(\frac{k}{m}\frac{T^2}{4}-1\right) \\ b_{2,D} & = & \frac{1}{\Delta}\left(\frac{1}{m}\frac{T^2}{4}\right) & a_{2,D} & = & \frac{1}{\Delta}\left(1-\frac{b}{m}\frac{T}{2}+\frac{k}{m}\frac{T^2}{4}\right) \end{array} \tag{9.45}$$

If the example looks complicated, that's the point. The physical parameters get lost in the shuffle a lot and we need to fight to keep them in terms that are meaningful in our discrete time model. The problems get worse when the system order gets higher.

What I've come to realize is that breaking the problem down into blocks and discretizing the blocks makes a lot of sense in the sense that each block has it's own discretization error, but it also preserves the physical meaning of the original block (if you do it right, which still isn't trivial).

### 9.7.3 Linear, Time-Invariant, Discrete-Time Modeling of the Real World

Let's assume that I have a model of the physical world that is linear, time-invariant, and discrete time.

$$x_{k+1} = Fx_k + Gu_k + G_W w_k \tag{9.46}$$

$$z_k = Hx_k + Du_k + v_k \tag{9.47}$$

We are dropping the $D$ for discrete matrix subscripts as we have already specified that we are in discrete time here.

The matrices describe how the system evolves from one time step to the next. The terms, $w_k$ and $v_k$, denote noises that affect the system, with $w_k$ being a "process" noise at the input that actually affects where the actual states will go, while $v_k$ is a measurement noise at the output that does not affect the actual states, but affects our ability to measure the output.

We cannot measure noises but may know their distribution. For example, the noises are often assumed to be additive, white, Gaussian noise, i.e.

$$w_k \sim N\left(0, \sqrt{W}\right) \text{ and } v_k \sim N\left(0, \sqrt{V}\right). \tag{9.48}$$

That is, they have a Gaussian distribution with zero mean and variances of $W$ and $V$, respectively.

If we know the model, we can simulate the system:

$$\begin{aligned} \bar{x}_{k+1} &= F\bar{x}_k + Gu_k & (9.49) \\ \bar{y}_k &= H\bar{x}_k + Du_k. & (9.50) \end{aligned}$$

There are two clear issues with our simulation:

- it doesn't use noise and
- it's completely disconnected from measurements of the physical system.

Dave Luenberger and Rudy Kalman solved this problem in slightly different ways at the beginning of the 1960s. Luenberger created an "observer" or "estimator" essentially doing noise free analysis for his work. Kalman solved the problem for the case of additive, white, Gaussian noise. They both worked on discrete-time and continuous-time systems. For the sake of equivalence, a Kalman Filter takes the form of a Luenberger Current Observer in discrete-time, while a Kalman-Bucy Filter takes the form of a Luenberger Observer in continuous-time. Luenberger's observers have two different forms for discrete-time, one which corrects the estimate based on the previous measurement (and therefore has an exact 1 sample delay) and one that corrects the measurement based on the last measurement and therefore (due to finite conversion and computation time) has a fractional sample delay. The latter is known as a current observer, and is the form that I will explain.

The Luenberger Current Observer starts with the physical system model of Equations 9.46 and 9.47. The estimator itself is a two step process:

- Simulation: Predict next state and measurement

- Measurement Correction: adjust state estimate based on error with real measurement

That's it! That's a model-based measurement. The predict with model and correct with measurement is the key construct here.

Now, the textbooks [14, 171, 15, 208] teach these as separate steps, i.e.

$$\begin{array}{rcll} \bar{x}_{k+1} & = & F\hat{x}_k + Gu_k & \text{Time Update} \\ \bar{y}_k & = & H\bar{x}_k + Gu_k & \text{and} \\ \hat{x}_k & = & \bar{x}_k + L_C\left[z_k - \bar{y}_k\right] & \text{Measurement Update} \end{array} \tag{9.51}$$

Again, I have dropped the $D$ for discrete subscript. The $C$ subscript in $L_C$ denotes a current observer gain, as opposed to a predictive observer which would use $L_P$.

However, there are two confusing things about this form. First of all, it is not how one would program the filter, and so it confuses the poor soul who wants to actually implement the thing. I have reordered the above as follows:

$$\begin{array}{rcll} \bar{y}_k & = & H\bar{x}_k + Gu_{m,k} & \text{Estimate output using predicted state \& input} \\ \hat{x}_k & = & \bar{x}_k + L_C\left[z_k - \bar{y}_k\right] & \text{Adjust model state with feedback from measurement} \\ \bar{x}_{k+1} & = & F\hat{x}_k + Gu_{m,k} & \text{Predict next step's state} \end{array} \tag{9.52}$$

I have glossed over how we get to a linear discrete-time model, except for the example discussion in Section 9.7.2, as this would take a long time here and is discussed in more depth in Chapter 2.

What is essential is to realize that a lot of physical systems can be adequately modeled in this way. It's not perfect (the world is neither linear, nor time invariant), but it allows us to make a lot of progress on a lot of things. Those smart phones, digital communications, flat screen digital TVs, etc. are all in one way or another premised on the idea that digital (quantized and discrete time) models of things are adequate for a lot of purposes.

So, we have a physical system or process that is adequately described by an LTI model. Linear means if you double the input, you double the output. Time invariant means that if I wait 5 seconds to hit the system, I get the same response, only delayed by 5 seconds. Finally, the digital model presumes that we are looking at the data often enough so that we can capture all the essential behavior of the original continuous time, real number version. This once again is the Nyquist sampling theorem (and there is an equivalent by Widrow [38] for amplitude quantization), but again, the Nyquist rate is asymptotic, so for our purposes we need to sample far more than twice the highest frequency.

Oh, and the noise, the noise, the noise. Most of the time, folks assume it is additive, white, Gaussian noise (AWGN) . Additive, in that it gets summed in, not multiplied. White in that noise at time $k$ is independent from noise at time, $k-1$, etc. Gaussian says that the probability density function is Gaussian, which makes the math nice.

### 9.7.4 Error Dynamics of Current Observers

Often, the most fundamental way of understanding a loop's behavior is by understanding the error dynamics of the loop. $H$ represents what we choose to (or can) measure and $L_C$ is our "knob" to compromise between measurement and model. We can define the filter tracking error, do lots of math and get:

$$\begin{aligned} \tilde{x}_{k+1} = x_{k+1} - \hat{x}_{k+1} &= (I - L_C H) F \tilde{x}_k + (I - L_C H) G_w w_k - L_C v_k \\ &\quad + (I - L_C H) G \left[u_k - u_{m,k}\right] - L_C D \left[u_{k+1} - u_{m,k+1}\right] \end{aligned} \tag{9.53}$$

Somewhat surprisingly, we don't see this whole equation in textbooks and papers. Instead, in different contexts, we see parts of this equation with some of the terms missing. In most filtering contexts, we do not assume we have access to the input driving the system, so the $u$ terms are missing. Instead, the input/process noise, $w$ is given a much larger covariance. In feedback contexts, $u$ is almost always assumed to be known perfectly.

- The dynamic error analysis is typically presented without noise. This asserts the stability of the closed-loop observer, but $w$ and $v$ are left off.
- Almost all treatments assume there is no direct feedthrough, i.e. $D = 0$. We have discussed this earlier in the chapter, but what it means in Equation 9.53 is that a potential source of mismatch is ignored.
- They either assume they know the input ($u_{m,k} = u_k$) or ignore it ($u_k$ is left out. Either way, the last two terms of Equation 9.53 don't show up.

Seriously, I've looked at a lot of books on estimation [208, 297]. I've even opened a few of them. They treat all these cases as separate problems. For making real measurements, and certainly for debugging some real system, we don't want to do this. We want to know what all our error sources might be.

Breaking down Equation 9.53, we can isolate how the different portions lead to different parts of the analysis.

$$\tilde{x}_{k+1} = (I - L_C H) F \tilde{x}_k \tag{9.54}$$

$$+ (I - L_C H) G_w w_k - L_C v_k \tag{9.55}$$

$$+ (I - L_C H) G [u_k - u_{m,k}] - L_C D [u_{k+1} - u_{m,k+1}] \tag{9.56}$$

(9.54) represents the classic noise free analysis. This is the part that gets the linear algebra jocks all excited. They will tell you all about the eigenvalues and how stable the thing is. If

$$\max |eig ((I - L_C H) F)| < 1$$

then the discrete-time state estimates converge. In fact, the choice of $L_C$ chooses the filter dynamics (e.g. bandwidth), subject to mathematical condition (observability) [171]. Basically, observability is a matrix relationship involving $H$ and $F$ which says whether from the given set of measurements, one can recreate everything that is going on in that linear, time-invariant system. In other words, the relationship of $H$ and $F$ tell us if we can possibly reconstruct the internal state of the system model. (Remember, that we have already introduced an error by assuming that the system is LTI, but it really, helps us do something with the math.)

(9.55) represents the disturbance due to noise terms. This usually makes assumptions about the noise spectra (like it's additive, white, Gaussian noise). Additive means the noise is just an additive signal. White means that each noise input is independent from it's previous state. Gaussian means that the probability density function from which the white noise is drawn has a Gaussian shape. AWGN goes through linear filters and comes out as still Gaussian, but shaped by the filters. This makes the math a lot nicer.

(9.56) represents the disturbance due to not knowing the input driving the physical system. Call this the "unknown input disturbance", which is only there if $u_{m,k} \neq u_k$. Now, in most textbooks and papers, we don't see this term (I have never seen it in any textbook or paper) because if the direct feedthrough, $D$ is 0, then the second term drops out and if we know the input, both terms drop out.

What about problems when we do not know the input? These problems are surprisingly common: target tracking problems when the radar is trying to track an aircraft but has no access to the pilot's control inputs, etc. all fall into this category. In this case the authors seem to ignore the driving input or they consider it noise. This has consequences, in that if we assume that the input noise is so large that it can account for a large deterministic but unknown signal, then we have to trust our measurement more and our model less. The consequences of this are increased noise in the filtered output. However, if we do this, we can often get away with $u_{m,k} = 0$, provided that our filter is tracking much faster than the unknown deterministic input, $u_k$ is changing. Thus, the deterministic, noise free part has a strong effect on how we handle the noise disturbance and the unknown input disturbance.

### 9.7.5 What the heck is different about a Kalman Filter?

As noted earlier, a Kalman filter [208] takes the form of a Luenberger current observer [15]. Again, we have dropped the $D$ subscript on the state matrices as it is clear we are operating in discrete time. We still have our Time Update:

$$\bar{x}_{k+1} = F\hat{x}_k + Gu_k \tag{9.57}$$

and our Measurement Update:

$$\begin{aligned} \bar{y}_k &= H\bar{x}_k + Gu_k \\ \hat{x}_k &= \bar{x}_k + L_C\left[z_k - \bar{y}_k\right] \end{aligned} \tag{9.58}$$

but now, we modify $L_C$ to possibly vary every step, $L_{C,k}$ and it is chosen as a least squares balance between how the process noise, $w_k$, and the measurement noise, $v_k$, affect the measurement.

The Kalman Time Update is now:

$$\begin{aligned} \bar{x}_{k+1} &= F\hat{x}_k + Gu_k && \text{State Estimate Time Update} \\ M_{k+1} &= FP_kF^T + G_wWG_w^T && \text{State Uncertainty Time Update} \end{aligned} \tag{9.59}$$

and the Kalman Measurement Update is:

$$\begin{aligned} \bar{y}_k &= H\bar{x}_k + Gu_k, && \text{Predicted Measurement} \\ P_k &= M_k - M_kH^T\left(HM_kH^T + V\right)^{-1}HM_k, && \text{State Uncertainty Measurement Update} \\ L_{C,k} &= P_kH^TV^{-1}, && \text{Estimator Feedback Gain Update} \\ \hat{x}_k &= \bar{x}_k + L_{C,k}\left[z_k - \bar{y}_k\right]. && \text{State Estimate Measurement Update, \&,} \\ \hat{y}_k &= H\hat{x}_k + Gu_k. && \text{Best Estimate of Measurement} \end{aligned} \tag{9.60}$$

where $W$ is the covariance matrix of the process noise, $w_k$, and $V$ is the covariance of the measurement noise, $v_k$, i.e. $w_k \sim N(0, W)$ and $v_k \sim N(0, V)$.

Likewise, the steady-state Kalman Filter is can be written using the same state update equations, but now we have steady-state values of uncertainty, $P_k = P_{SS} = P$ and $L_{C,k} = L_{C,SS} = L_C$ as:

$$P = FPF^T + W - FPH^T \left[HPH^T + V\right]^{-1} HPF^T \tag{9.61}$$

$$\text{and } L_C = FPH^T \left[HPH^T + V\right]^{-1}, \tag{9.62}$$

where (9.61) is an Algebraic Riccati Equation (ARC) [208]. For a given system model, the noise covariances, $W$ and $V$, uniquely determine the feedback gains, whether they be evolving, $L_{C,k}$, or stead state, $L_C$.

Most of this is just as we had before, except for the propagation of uncertainty which is used to pick the feedback gain, $L_{C,k}$ at every step. The Kalman Filter is the least squares solution to a particular mathematical problem, but when one asks if it is optimal, the true answer is in the form a question: How well did we model reality? More specifically, How accurate are $\{F, G, G_w, H, D, W, \& V\}$ and does $u_{m,k} = u_k$?

The Dark Truth: People usually optimize the crap out of lousy models.

My point here is that Kalman almost seems like a sacred word sometimes, and I don't think it should be. The point I am trying to hammer home here is that the Kalman filter is just a Luenberger observer with a specific way to pick the feedback gain, $L_C$. Furthermore, the Kalman Filter is only as accurate as the model and the noise estimates allow it to be. However, when you don't have anything else to go on, it gives you a great starting point at picking $L_C$.

Note that Luenberger came up with two different estimators/observers [15]: the current observer I've shown you here, and the predictive observer (slightly different form, slightly more time delay and phase lag, but your real world system can behave much more like your model if you allow a full sample delay). Since I hate latency for other reasons, I am usually hesitant to use this form and the current observer is in the same form as the Kalman Filter.

Maybe a good place to give the Luenberger predictor observer and show the differences in the matrix equations.

Also, for continuous time systems, the Kalman-Bucy Filter is the equivalent least squares solution [208].

And for systems with nonlinearity or ones that are not time-invariant, the Extended Kalman Filter gets used [208].

Good place for a subsection on using PES Pareto to get the KF noise matrices. [298]

Here is the essential "model based filter/measurement/etc.":

We simulate and we measure, and we compare the two. The simulation might not be as exotic as one that doesn't have to deal with real-time data, but by comparing it to real-time data, we should be able to track the real time data.

### 9.7.6 Back to Our Simple Second Order System

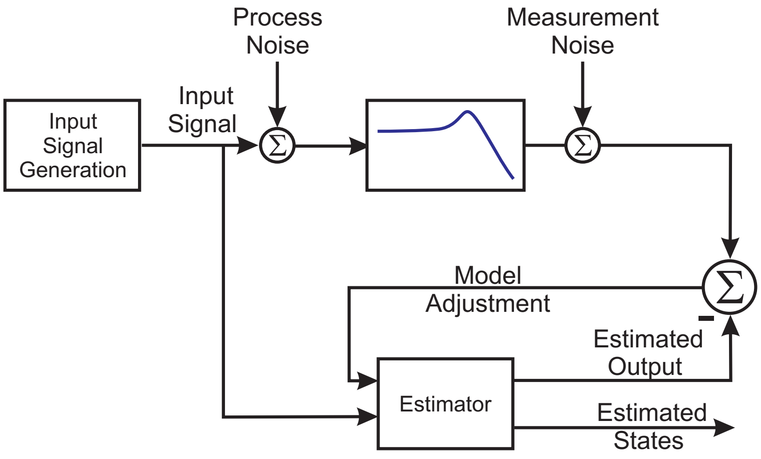


Figure 9.10: This system is a simple, resonance, driven by an input, with noise.

We return to our simple example, now diagrammed in Figure 9.10 with the estimator added. Our system is the second order resonance, of Equation 9.2, but now we translate this into resonance terms:

$$\frac{X(s)}{F(s)} = \frac{\frac{1}{m}}{s^2 + \frac{b}{m}s + \frac{k}{m}} = \frac{K\omega_d^2}{s^2 + 2\zeta_d\omega_d s + \omega_d^2} \tag{9.63}$$

We discretize the model, and then drive the model the same way the system is driven and as Figure 9.11 shows, it works. This is no big surprise. Note the wonderful rejection of noise in the green curve. The

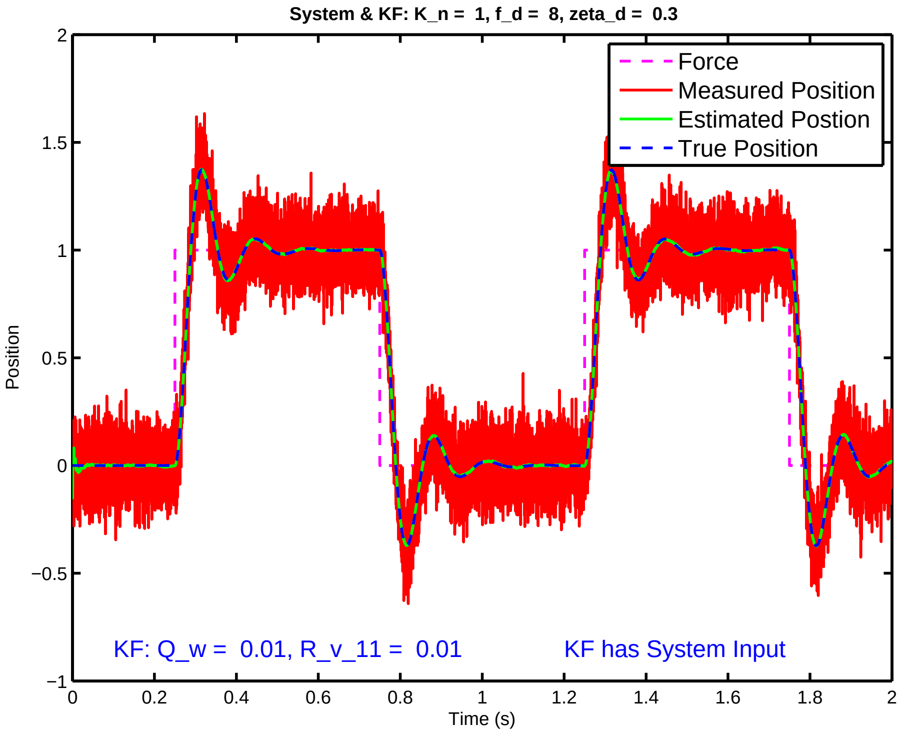


Figure 9.11: Noise driving the simple resonance.

filter is very narrow band around the model's projected output. This is the feedback context: we assume we have access to both the output and the input. In the filtering context, we do not have access to the input to the system.

If we can't see what is pushing the system around, diagrammed in Figure 9.12, we get a big error, as seen in Figure 9.13. Essentially, our filter lacks the bandwidth to track the unknown input. The filter is very narrow band around the model's projected output, but the model's projected output can't track the input it doesn't know about. Thus, the narrow band is in the wrong place. It's like an old Borscht Belt joke: The good news, we reject a lot of noise. The bad news, we don't follow the signal.

The typical fix goes as follows: We can pretend that driving input is a noise, so we artificially raise our process noise model (make $W$ bigger). What does it mean to make a positive definite matrix "bigger"? It means we make the singular values bigger. In this case, we have a single process noise, so $W$ is a $1 \times 1$ matrix, so we just make $W$ bigger – a lot bigger. This makes the input seem like noise, so we trust the model less, and the sensor more, which means we filter the sensor less. This is a simple tradeoff that we have to make to track the signal and it is shown in the results of Figure 9.14.

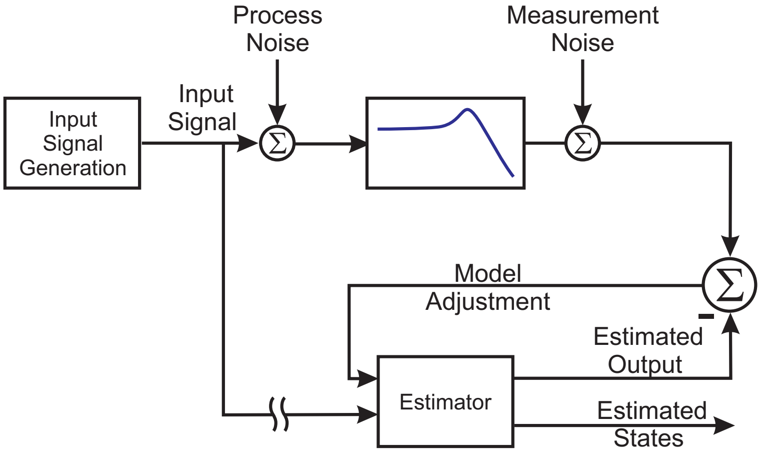


Figure 9.12: The system is a simple, resonance, driven by an input, with noise. However, in this case we do not have access to the input signal driving the system.

Note that there is an alternate method not shown in the talk, but in Figure 9.15, known as an unmodeled input observer, disturbance observer, disturbance estimator, extended state observer, etc. [299, 300, 301]. In this case an extra state is added to the model to account for the unmodeled input. This state is driven by its own "process noise" and so we tune that noise model to account for the input size. However, that allows for the internal states of the model to converge and if the model tracks faster than the input changes, we can then have a lower noise level on the real "process noise" input. This is what I did for Figure 9.15, and the result is that away from the input steps, it seems to get close to the original results of Figure 9.11. This is a bit beyond the scope of this tutorial, but I thought I'd mention it in the notes. The thing is that this does illustrate one of the great things about model based methods: the ability to add some extra structure to your model even if it doesn't correspond to something you can see in a transfer function.

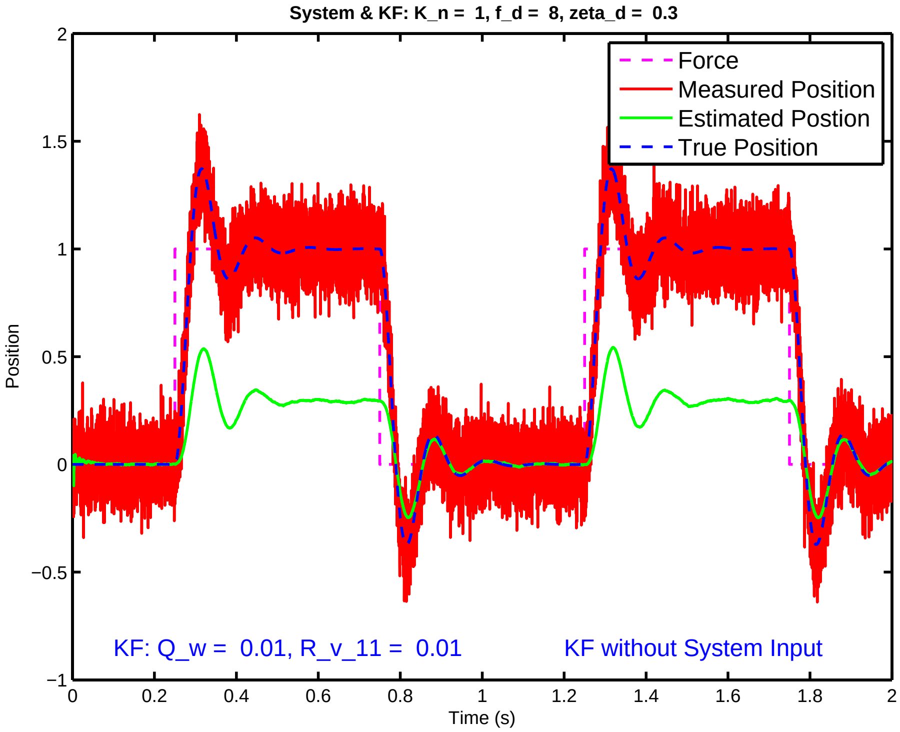


Figure 9.13: The system is a simple, resonance, driven by an input, with noise. We do not have access to the system input and so our filter does not track.

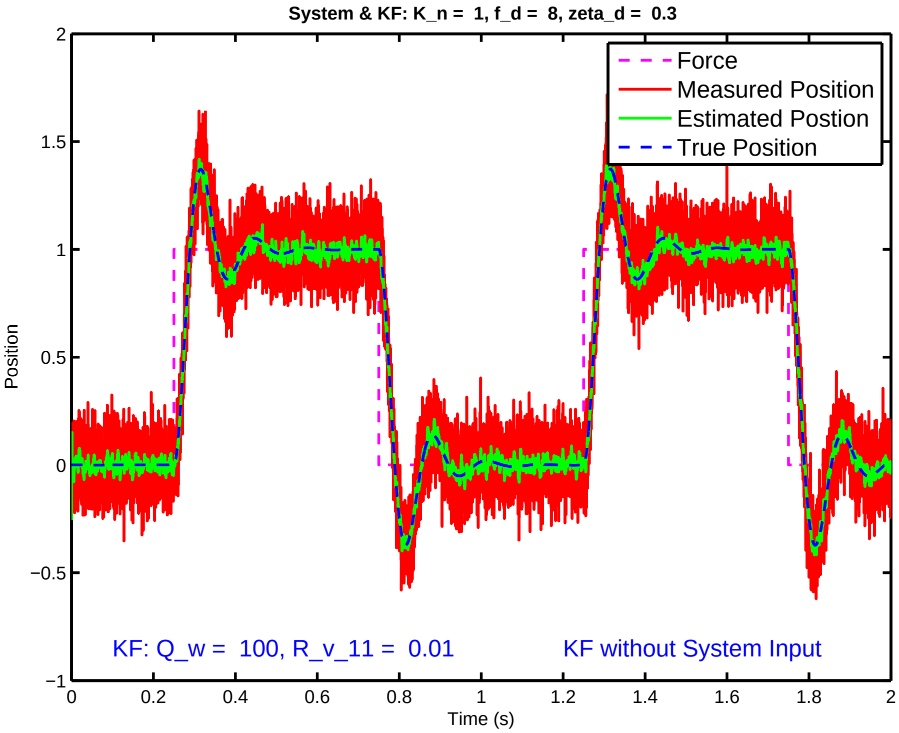


Figure 9.14: System is a simple, resonance, driven by an input, with noise. We do not have access to the system input, but we have increased the process noise model so that it treats the unknown input as a noise. Note the tracking at the expense of extra noise in the estimate.

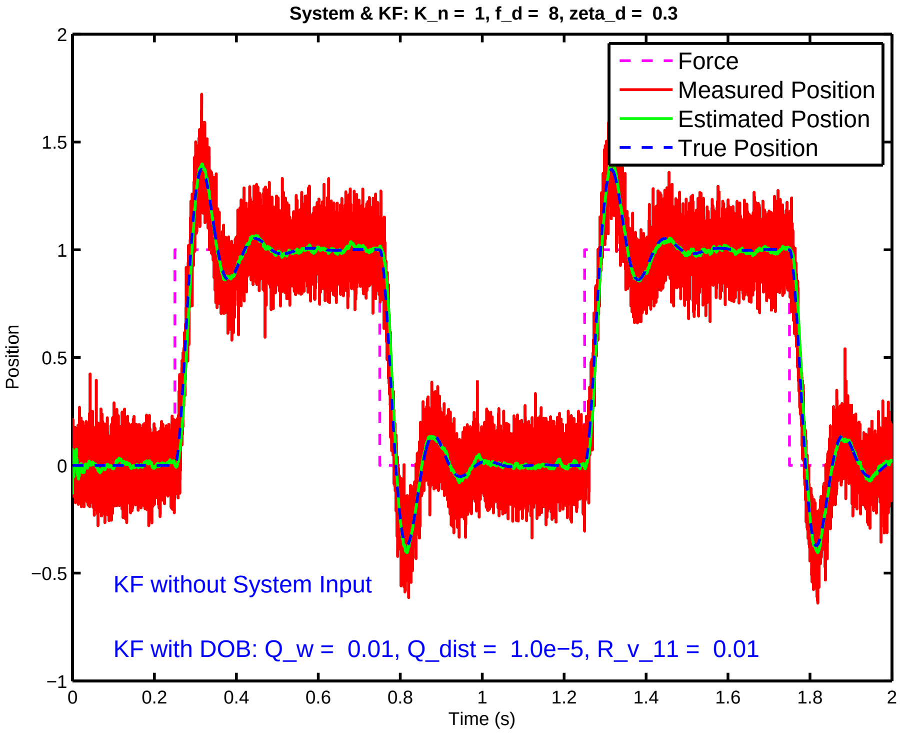


Figure 9.15: The system is a simple, resonance, driven by an input, with noise. We do not have access to the system input, and have left the process noise model unchanged. We have added an unmodeled input observer and the extra "input state" allows tracking with considerably less increase in the estimator noise.

### 9.7.7 What Makes Model Based Measurements Hard?

If model based methods are so darned good, why aren't they ubiquitous? (i.e. Why is this hard?)

- Model based-measurements require a model.
- Did I mention you need a model?
- Most folks really just phone in the model, so the error reduction is limited.

Bad models mean bad model based measurements. That simple. And the ways folks use to get models are often limited because they are trying to be computationally efficient (needlessly so in some cases). Or they are adhering to technology limits that were around 30 years ago and not around now.

## 9.8 The Canonical Forms

Related to polynomial form transfer functions

compression of parameter meaning but exposes controllability and/or observability

- controller
- observer
- controllability
- observability
- diagonal
- Jordan

## 9.9 State Space for MIMO Systems

One of the main selling points of state space methods is that they give a unified way of handling estimation and control for multi-input, multi-output (MIMO) systems. Instead of having a bunch of seemingly unrelated mappings between individual input and output pairs, the MIMO state-space model has one unified set of dynamics. What could go wrong?

As usual, the problem is not in the principle but in getting to such a unified model from measurements. We generally think of making measurements in a way to isolate systems when we can, so it is natural to try to derive our MIMO model from a collection of SISO measurements (and perhaps SISO models) between any input-output pair. Perhaps we are lucky enough to stimulate at one spot and measure at all the output measurement points, in a single-input, multi-output (SIMO) fashion. We still need to decide how and when to combine dynamics that are close to each other (e.g. poles or zeros that seem to differ only by a small numerical error). How close is close? Perhaps we can tell which are related by understanding the physical features and parameters of the system (e.g. the mass is generally constant no matter which input-output pair we use), but then again we trashed our physical parameters when we went to discrete-time polynomial transfer function type identification.

MIMO state-space models are not a bad idea, but again, getting there involves a lot of work.

## 9.10 What's Up with Implementing State Space?

So, why is it so hard to translate physical models into usable state space controllers? We get back to this issue that has run through the book of taking a complex model, turning it into a polynomial form transfer function in continuous time, then either discretizing that model and turning it into state space, or turning it into state space and then discretizing. Tools such as Matlab make that relatively easy from a computation perspective, but most physical understanding is lost there.

- How many terms need to be included in the original polynomial model?
- How should it be discretized?
- What form should the discrete-time state-space model take?

- How are the physical measurements tied into this discrete model?
- How does the discrete model – which we used in large part to give us more understanding of the system – relate to things we can measure in the real world?
- What kind of insight does the model actually give us?
- Perhaps most succinctly: what is the relationship between the continuous-time (CT) states and the discrete-time (DT) states? I mean, if we put in all the mathematical machinery to do a discrete state-space model, shouldn't we at the very least have simple access to position and velocity?

Repeating Equations 9.46 and 9.47 here:

$$\begin{aligned} x_{k+1} &= Fx_k + Gu_k + G_W w_k && (9.64) \\ z_k &= Hx_k + Du_k + v_k, && (9.65) \end{aligned}$$

we have the fundamental task of filling out the model matrices, $\{F, G, G_W, H, \text{ and } D\}$ in a sensible way.

Starting with a general polynomial digital filter form:

$$F(z^{-1}) = \frac{Y(z^{-1})}{U(z^{-1})} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}, \tag{9.66}$$

we make it a bit more unique by normalizing by the $a_0$ term so that $a_0 = 1$. As $a_0$ is the coefficient of $y_k$, this simplifies the process for computing $y_k$ as a function of past values of $y_k$ and past and current versions of $u_k$.

This underpins all of the textbook state-space methodologies. We will ignore $G_W$ here, as we are trying to understand the implications of the noise free structure (which represents a best case scenario). For the discussion right now, we use the polynomial filter coefficients to populate the matrices. The first form here is known as the controller canonical form [171].

$$F = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{9.67}$$

$$H = \begin{bmatrix} b_1 & b_2 & \cdots & b_{n-1} & b_n \end{bmatrix} \quad D = \begin{bmatrix} b_{0|} \end{bmatrix} \tag{9.68}$$

Note that if $b_0 = 0$ in our transfer function, we get no direct feedthrough, which we can directly see as $D = 0$. Equations 9.67 and 9.68 fill out the matrices for the controller canonical form. The input scaling is $1$, the polynomial transfer function denominator coefficients are across the top row of $F$, and the numerator coefficients populate the output vector. We might surmise that unless there is something pathological about the denominator, then the direct line from the input to the $F$ matrix means that this realization will always be controllable. Simply put, that means in this LTI system, we can drive the state to any value from the input, $u$:

$$u_k = -Kx = -k_1 x_1 - k_2 x_2 - \ldots - k_n x_n. \tag{9.69}$$

By the same token, the presence of the numerator polynomial in the output matrices, $H$ and $D$ implies that there might be some notches or zeros there that would make not all the states observable. That is, it is not obvious that we can all the states by simply observing the system output. Mathematically, some of the zeros of the $b$ polynomial may cancel some of the zeros of the $a$ polynomial and make that state unobservable.

An alternate canonical form, the observer canonical form guarantees this:

$$F = \begin{bmatrix} -a_1 & 1 & 0 & \cdots & 0 & 0 \\ -a_2 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ -a_{n-1} & 0 & 0 & & 1 & 0 \\ -a_n & 0 & & 0 & 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{9.70}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} b_0 \end{bmatrix} \tag{9.71}$$

Equations 9.70 and 9.71 fill out the matrices for the observer canonical form. Because the output matrix is directly from the states, the realization is always observable and one can theoretically reconstruct the stats by simply measuring the denoted output. It does not guarantees controllability (above) as the input vector may now hold notches or zeros. Find the math that shows this is observable.

It is worth pointing out that in most academic textbooks, $D$ is typically absent. This indicates that there is no direct feedthrough from the input to the output. When used in a model of a physical system, this is taking into account the general assumption that the physical system is – within the range included in the model – low pass and that this manifests itself in a transfer function as having more poles than zeros. In a state space model, the consequence is that the $D$ matrix is $0$ or nonexistent. This simplifies the analysis in many of the textbooks, but blinds those materials to a very fruitful construction of state-space matrices that I'll get to in Section 9.15.

As discussed earlier, there is a frequently used test for controllability, the controllability matrix, defined as:

$$\mathcal{C} = \left[ \begin{array}{cccc} G & FG & \cdots & F^{n-1}G \end{array} \right]. \tag{9.72}$$

Simply put, if $\mathcal{C}$ is nonsingular, then the state-space realization is controllable: one can use the defined input to drive the states to any value. Similarly, when the observability matrix,

$$\mathcal{O} = \left[ \begin{array}{cccc} H^T & F^T H^T & \cdots & (F^T)^{n-1} H^T \end{array} \right]. \tag{9.73}$$

is nonsingular (i.e. has full rank or is invertible) then the realization is observable. One can reconstruct all the states from the measured output.

In general, the theory says that a state-space realization is both observable and controllable, when it is minimal; that is when there are no pole-zero cancellations in the related transfer function. Okay, but what does this mean technically? In real systems it is not a binary question of observable or not, controllable or not. It is a floating point question of how controllable and how observable. The controllability matrix might be nonsingular but the spread in the singular values of the controllability matrix might mean that some states – while theoretically reachable – cannot practically be controlled from the input. The coupling might be so weak that the input needed would violate saturation or power limits. These don't show up in a LTI analysis, but they are real in implementing the system. Similarly, a state may couple to the output so weakly as to be piratically unobservable. Any noise or disturbance in the measurement would make accurately recovering this state almost impossible. Understanding the difference between theoretical and practical observability and/or controllability might lead us to improve, move, or simply add a sensor and/or an actuator.

Any real implementation of a state-space filter or controller (a.k.a. a model based filter or controller) has to take this into account. still, the real question may be: if we see one of these problems in a discrete-time state-space model, how to we fix it? Were to we put another sensor or move the current one if our model gives us no physical insight? Where might the physical system be tweaked to improve our modeling ability?

If you've paid any attention so far, you will realize that the realizations above give little or no help. As Brazilian Jujitsu instructor, Kurt Osiander is so fond of saying, "If you go to this point, you already f***ed up." These mathematically nice but nonphysical structures give us little hope of understanding what is going on in the physical system and little or no ability to debut them based on measurements. Generally, this means (excepting the exceptions at the beginning of the chapter), those of us that have used these canonical forms in real systems have already f***ed up.

## 9.11 The Transfer Function of a State Space Realization

$$H(s) = H_C(sI - F_C)^{-1}G_C + D_C \tag{9.74}$$

$$H(z) = H_D(zI - F_D)^{-1}G_D + D_D \tag{9.75}$$

## 9.12 Adding an Integrator to State Space

## 9.13 Adding Feedforward Control to State Space

## 9.14 State Space Midpoint Summary

Let's summarize where we have gotten with state-space methods. They are motivated by the notion of looking inside the system (via a model or realization) to get a deeper understanding of the internal behavior of the system via the behavior of the model "states". In principle, if we could accurately measure each of these states from the physical system, we would have a lot more design freedom in adjusting the closed-loop behavior of our system. This is subject to the notion of controllability – the idea that we can affect all the states at will from the chosen input(s). Compare this to the idea of loop shaping with a transfer function model of the system, where the main way that we separate the dynamics is from their pole and zero locations. If they are too close to each other, it becomes hard to deal with them individually. This is clearly related in some way to related to controllability and observability.

On the other hand, since we cannot usually measure every state, we work to observe (a.k.a. estimate) them from the available measurement(s) – hence the notion of observability. For example, if we are working with a SISO system, we have one measured output and one controlled input. In some sense, from an input/output (or output/input) perspective, our controller has the same tunnel as the transfer function version did. What, if anything did we accomplish here? Inside that measured output to system input tunnel, we expanded the signal space out (to states) based upon some model of the system.

Stated this way, whether or not the states are useful to our real-world application depends almost entirely on how well our model represents the aspects of our physical system that we need to understand to implement good feedback control.

If our system is extremely well behaved (e.g. passive, contractive, stable and well damped) then the specific physical parameters might not matter much. We might not care too much about what is under the hood if we are merely trying to flatten the response of a system that presents as a poor low-pass filter. That being said, it doesn't seem that doing that gives us much of the promised system insight. It is when the system is not too well behaved, is more "strenuous" with lightly damped dynamics, critical physical parameters, significant time delays, or nonlinearities that become significant when the performance is pushed, that we really want that system insight that state space methods were supposed to give us.

There are other issues with the canonical forms, stemming from their similarity to polynomial transfer functions. Besides the lack of connection between polynomial parameters and physical meaning, the polynomial parameters are often numerically sensitive. This is especially true for discrete-time polynomial coefficients. This shows up as poles and zeros compress in the $z$-plane towards $z = 1$. We have seen in Chapter 6 how fixed-point representations compound the numerical sensitivity of polynomial coefficients.

We do not have quite the same issue in continuous time, but we do have the issue that system dynamics spread across many decades can have a massive numerical range. If we simply consider resonances at $1$, $10$, $100$, $1000$, and $10,000$ Hz, we end up with the numerical spread of $\omega_0^2 = (2\pi f_0)^2$ terms as displayed in Table 9.1.

| | | | | Integer |
|---|---|---|---|---|
| | $f_0^2$ | $\omega_0^2$ | $\log_2(\omega_0^2)$ | Bits |
| $1$ | $1$ | | | |
| $10$ | $100$ | | | |
| $100$ | $10^4$ | | | |
| $1000$ | $10^6$ | | | |
| $10,000$ | $10^8$ | | | |

Table 9.1: Resonant frequencies and bits to represent them. Each resonance will have an $\omega_0^2$ term that filters into the continuous time coefficients. By taking the base-2 logarithm of this number, we get a gauge on the number of integer bits needed to hold the integer representation of that number.

Now, we may never see this kind of spread of dynamics in chemical process control (CPC), biological process control (BPC), thermal, or pressure systems. These tend to be modeled as low-order, slow systems, with overdamped dynamics. Much of the complexity of these problems arises in finding the appropriate or even any adequate means of sensing and/or actuation in these environments. Getting a

| | | | | | | Integer |
|---|---|---|---|---|---|---|
| | $f_0^2$ | $T_S$ | $e^{-2\omega_0 T_S}$ | $1-e^{-2\omega_0 T_S}$ | $\log_2(1-e^{-2\omega_0 T_S})$ | Bits |
| $1$ | $1$ | | | | | |
| | $1e-3$ | | | | | |
| | $1e-6$ | | | | | |
| $10$ | $100$ | | | | | |
| | $1e-3$ | | | | | |
| | $1e-6$ | | | | | |
| $100$ | $10^4$ | | | | | |
| | $1e-3$ | | | | | |
| | $1e-6$ | | | | | |
| $1000$ | $10^6$ | | | | | |
| | $1e-3$ | | | | | |
| | $1e-6$ | | | | | |
| $10,000$ | $10^8$ | | | | | |
| | $1e-3$ | | | | | |
| | $1e-6$ | | | | | |

Table 9.2: Resonant frequencies and bits to represent them in discrete time. Each resonance will have an $\omega_0^2$ term that filters into the discrete-time coefficients. In this case, we are concerned about how close the term is to $z = 1$. By taking the base-2 logarithm of the difference, we get a gauge on the number of integer bits needed to hold the integer representation of that number.

sensor inside a cell or a $1000°C$ reactor both have their own challenges. Right now, what I am driving towards is how state-space formulations fare on systems with multiple lightly damped resonances, possibly spaced far apart in frequency. We saw in Chapter 6 that such dynamics play havoc with the numerical fidelity of polynomial coefficient filters. This, in turn, motivated the development of the multinotch (Section 6.11, [54, 33]). If the inside of our state matrices are populated with the same polynomial filter coefficients that gave us trouble in the filter world, why would we not expect trouble in our state matrices?

Once can legitimately argue that the polynomials for the state matrices are different from the polynomials for filters, but for most high-performance control demands (e.g. pushing closed-loop bandwidths beyond the frequencies of some of the resonances), they are at least partially inverses of each other, if we neglect the rigid body dynamics that we address with a well-tuned PID. Thus, for high-performance applications our canonical form state matrices will face largely the same issues as our polynomial form filters. For these we might ask: can we adapt the multinotch to state-space forms? The answer is an emphatic, "Yes!" The answer one gets when we realize the full benefits of doing so is, "Heck yeah!!!"

While the preceding sections in this chapter were an attempt to provide some missing context for how and why we would use state-space methods, including the often neglected areas of integral control and feedforward matrices, the sections that follow will focus on formulating those state matrices in a way that enhances their numerical properties, increases insight into the physical system, and makes it easier to debug the real control loop on that physical system. I will introduce the biquad state space (BSS) [3, 4] and the bilinear state space (BLSS) [5] structures and show how they help with a lot of this.

## 9.15 The Biquad State Space Structure

One of the aspects of digital control that gets brief mention in control textbooks and research papers is the implementation of low latency control. It is well understood that latency, including computational latency, erodes phase margin by adding negative phase. Some textbooks mention precalculating operations which do not depend upon the current input in the preceding sample interval [15, 16]. The savings in latency are illustrated in Figure 9.16 (a repeat of Figure 6.1 to keep us from flipping pages).

In the single-input, single-output (SISO) case this is tedious, but relatively straightforward if the controller can be cast into the form of a high order polynomial filter. This is shown in Figure 9.17, and represented as transfer functions in the unit delay operator, $z^{-1}$:

$$\frac{Y(z^{-1})}{U(z^{-1})} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}. \tag{9.76}$$

This gets implemented in a filter as [169]:

$$\begin{aligned} y_k &= -a_1 y_{k-1} - a_2 y_{k-2} - \ldots - a_n y_{k-n} \\ &\quad + b_0 u_k + b_1 u_{k-1} + \ldots + b_n u_{k-n}. \end{aligned} \tag{9.77}$$

Looking at (9.77), we see that $y_k$ depends mostly on previous inputs and outputs. The only current value needed is $u_k$ and this is only multiplied by $b_0$. So we can break this up into [15]:

$$y_k = b_0 u_k + \text{prec}_k, \text{ where} \tag{9.78}$$

$$\begin{aligned} \text{prec}_k &= -a_1 y_{k-1} - \ldots - a_n y_{k-n} \\ &\quad + b_1 u_{k-1} + \ldots + b_n u_{k-n}. \end{aligned} \tag{9.79}$$

We can see that $\text{prec}_k$ depends only on previous values of $y_k$ and $u_k$. This means that $\text{prec}_k$ can be computed for step $k$ immediately after the filter has produced the output for time index, $k - 1$ [16].

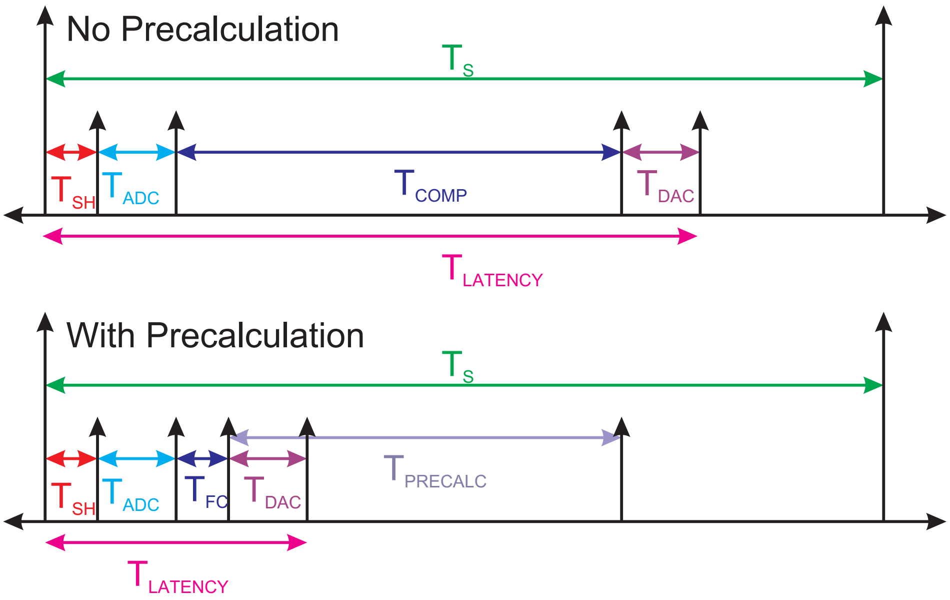


Figure 9.16: Input and output timing in a digital control system. The top drawing is without precalculation; the bottom drawing is with. Note that precalculation can be started as soon as the output has been sent to the DAC and therefore is in parallel with the DAC conversion time. The computation time, $T_{COMP}$, of the top diagram is now split into $T_{PRECALC} + T_{FC}$ where $T_{PRECALC}$ is the computation time needed for the precalculation and $T_{FC}$ is the time needed for the final calculation after the input sample. Modulo some small programming overhead, the split time should equal the total computation time. Here $T_{SH}$, $T_{ADC}$, and $T_{DAC}$ represent the sample and hold, ADC conversion, and DAC conversion times, respectively.

When the sample at time step $k$, $u_k$, comes into the filter, it need merely be multiplied by $b_0$ and added to prec$_k$ to produce the filter output. Thus, the delay between the input of $u_k$ and the output of $y_k$ is small and independent of the filter length. Small latency improves performance, but fixed latency implies predictable behavior, which may be more critical in debugging real time system.

In [54, 110], The Multinotch was introduced as a discrete time filter whose structure allowed for fixed and low latency between the most recent signal input and the filter output, while having the excellent numerical properties inherent in biquad structures. In [33] we demonstrated a filter coefficient adjustment, the $\Delta$ coefficients, which allowed high numerical fidelity even when the sample frequency was several orders of magnitude higher than that of the dynamic feature being filtered. Both of these papers implement the filter in a transfer function form.

This paper will demonstrate how to adapt The Multinotch for state space structures [183]. We will see that the same basic principles can be used to improve the computational latency and numerical fidelity

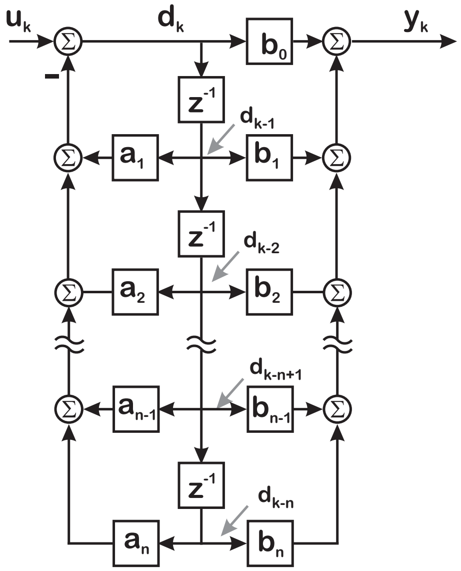


Figure 9.17: An $n^{th}$ order polynomial filter in Direct Form II configuration [167].

of current mode observers, thereby allowing state feedback with fixed and low latency. Furthermore, state space models of highly flexible systems can present severe numerical issues. The models derived from physical principles often lack structure. Canonical form models [171], are compact, but obscure any physical structure and can have coefficients that are highly sensitive to model parameters. What is needed is a form that has the compact representation of the canonical forms, the physicality of the forms derived from physical equations, and maintain numerical accuracy and physical intuition, even after discretization.

While The Multinotch was applied primarily to shaping loop dynamics with high Q resonances and anti-resonances, a good state space model also needs to be able to account for low frequency and rigid body dynamics. This will be demonstrated, using the classic double integrator as an example, in Section 9.20.

The numerical benefits of this form exist even when low latency is not a consideration, so we will show forms of the structure applicable in offline modeling and simulation in [4]. Finally, we will show a modeling example from experimental data of a mechatronics system where the Biquad State Space (BSS) form holds numerical accuracy far beyond conventional methods, as will become obvious in the examples of Section 9.21.

While the structure is quite regular and works for large or small numbers of biquads, the regular pattern becomes obvious in the three biquad case. Thus, most of the structural equations will be three biquad ones. The format considerations of this will mean that many of these matrix equations are in two column figures, but seeing the matrices in this way makes the structural properties fairly obvious. This will result in some of the larger equations being pushed into two column figures.

## 9.16 The Biquad Decomposition of Digital Filters

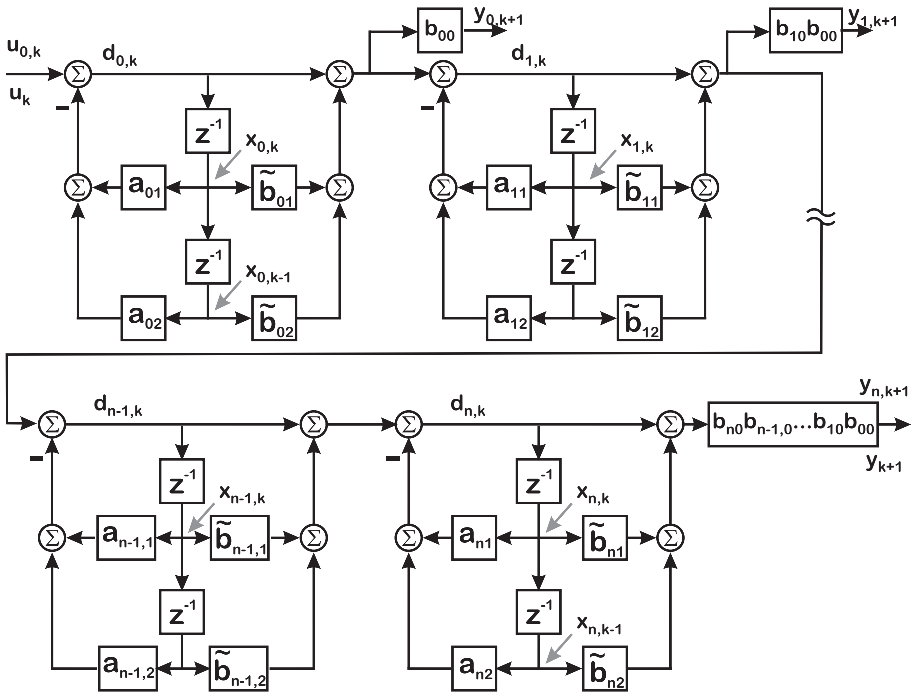


Figure 9.18: The updated discrete biquad cascade, with factored out $b_{i,0}$ terms and scaling the output of each block.

In [54], we discussed how a higher order single-input, single-output (SISO) digital filter, such as that in Equation 9.76, can be factored into a chain of second order filters known as biquads. This has been well established for a long time. However, until [54], using biquads [169] in digital feedback control meant that precalculation [15, 16] to reduce computational latency was limited to only the first biquad block since all downstream blocks needed the final output of the first block to do any computations. The Multinotch, by factoring out the direct feedthrough coefficients, and only multiplying them in at the output, removed that constraint, allowing the numerical advantages of biquad decomposition to be coupled with the low latency advantages of precalculation.

There is no need to repeat the equations of [54] here, but looking at the structure The Multinotch in Figure 9.18 there are a few things to note before generating our first state space form:

- The delay terms in the biquads are equivalent to states in a state space structure, but they are offset in time. Looking at Figure 9.18, $d_{i,k} = x_{i,k+1}$. That is, the digital filter approach defines delays on the input of time shifts $(z^{-1})$ while standard state space notation defines states on the outputs of time shifts.

- While $\tilde{y}_{i,k+1}$ depends on $x_{i,k+1}$, it can be recalculated as a weighted sum of prior delays and the current input. That is, we can calculate $\tilde{y}_{i,k+1}$ in parallel to $x_{i,k+1}$.

## 9.17 A Biquad State Space Form

So far, we have not done anything not already in [54]. However, we can look at each of these biquad sections as a state space realization. In this case:

$$\begin{bmatrix} x_{i,k+1} \\ x_{i,k} \end{bmatrix} \begin{bmatrix} -a_{i1} & -a_{i2} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_{i,k} \\ x_{i,k-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_{i,k} \tag{9.80}$$

while the state output equation is given by:

$$\begin{bmatrix} \tilde{y}_{i,k+1} \end{bmatrix} = \begin{bmatrix} \tilde{b}_{i1} - a_{i1} & \tilde{b}_{i2} - a_{i2} \end{bmatrix} \begin{bmatrix} x_{i,k} \\ x_{i,k-1} \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix} u_{i,k} \tag{9.81}$$

Finally, the properly scaled output is generated via:

$$\begin{bmatrix} y_{i,k+1} \end{bmatrix} = \begin{bmatrix} b_{i0} \end{bmatrix} \begin{bmatrix} \tilde{y}_{i,k+1} \end{bmatrix}. \tag{9.82}$$

The indexing of $\tilde{y}_{i,k+1}$ and $y_{i,k+1}$ are a bit odd because since we have direct feedthrough in our structure, $\tilde{y}_{i,k+1}$ depends on $x_{i,k+1}$ as well as $x_{i,k}$, $x_{i,k-1}$, and $u_{i,k}$. Thus, it's cleaner in what follows to call the biquad outputs, $\tilde{y}_{i,k+1}$ and $y_{i,k+1}$, respectively. We chain these together by noting that:

$$\begin{array}{lcll} u_{i+1,k} & = & \tilde{y}_{i,k+1}, & \text{for } 0 \leq i < n, \\ u_{0,k} & = & u_k, & \text{and} \\ \tilde{y}_{n,k+1} & = & \tilde{y}_{k+1}. & \end{array} \tag{9.83}$$

If one is willing to go through the algebraic pain and suffering of applying Equation 9.83 to each biquad structure a very regular state space structure results.

$$\begin{bmatrix} x_{2,k+1} \\ x_{2,k} \\ x_{1,k+1} \\ x_{1,k} \\ x_{0,k+1} \\ x_{0,k} \end{bmatrix} = \begin{bmatrix} -a_{21} & -a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -a_{11} & -a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -a_{01} & -a_{02} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{2,k} \\ x_{2,k-1} \\ x_{1,k} \\ x_{1,k-1} \\ x_{0,k} \\ x_{0,k-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_k \qquad (9.84)$$

$$\begin{bmatrix} \tilde{y}_{2,k+1} \\ \tilde{y}_{1,k+1} \\ \tilde{y}_{0,k+1} \end{bmatrix} = \begin{bmatrix} \tilde{b}_{21}-a_{21} & \tilde{b}_{22}-a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 0 & 0 & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \end{bmatrix} \begin{bmatrix} x_{2,k} \\ x_{2,k-1} \\ x_{1,k} \\ x_{1,k-1} \\ x_{0,k} \\ x_{0,k-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u_k \qquad (9.85)$$

For a 3-biquad model, we get the state equation of 9.84. The unscaled output is in Equation 9.85. Finally, the properly scaled outputs are generated via:

$$\begin{bmatrix} y_{2,k+1} \\ y_{1,k+1} \\ y_{0,k+1} \end{bmatrix} = \begin{bmatrix} b_{20}b_{10}b_{00} & 0 & 0 \\ 0 & b_{10}b_{00} & 0 \\ 0 & 0 & b_{00} \end{bmatrix} \begin{bmatrix} \tilde{y}_{2,k+1} \\ \tilde{y}_{1,k+1} \\ \tilde{y}_{0,k+1} \end{bmatrix}. \qquad (9.86)$$

One key of this form is that the generation of the state update (output vector) involves:

- Multiplication of the prior state vector by the state transition matrix (state output matrix) – none of which involves the current input. This can therefore be done in a precalculation step.
- Addition of the unscaled current input to each product row of the above multiplication. This can be parallelized so that the latency once the current input is available is that of a single addition.

Looking at this critically, the state transition and output matrices are always multiplied by the old state, and therefore could be precalculated in any form. If there is no direct feedthrough from the input to the output, such a model can be used without incurring much delay. However, the BSS is structured so that direct feedthrough from the input to the output needs one addition and one multiplication per output. This is a big benefit for using state space in real time control. The fact that the BSS also provides excellent numerical properties as will be seen in the example of Section 9.21.

The generation of the final, scaled output takes a single multiplication per output. Therefore, updating the state and output using the BSS using precalculation has a computational latency of two operations: one addition and one multiplication.

Our state equations are slightly different here, due to where the scaling happens: The general form of the discrete time, linear state equations is:

$$X_{k+1} = F_D X_k + G_D u_k \text{ and} \tag{9.87}$$

$$\tilde{Y}_k = H_D X_k + D_D u_k, \tag{9.88}$$

where $\tilde{Y}$ is the unscaled output vector. For simplicity, we will drop the $D$ subscript and assume digital state matrices unless otherwise noted. Also, we will move between the notations of $x_i(k)$ and $x_{i,k}$ because the space considerations in some of the larger equations.

State space models of highly flexible systems can present severe numerical issues. The models derived from physical principles often lack structure and have large parameter sets. On the other hand, canonical form models [171] reduce the number of parameters (and therefore computational operations) to a minimum set equivalent to those in a transfer function form. However, in doing so for anything more sophisticated than a second order model, most – if not all – physical intuition is lost. Furthermore, the compaction of these parameters into a canonical set often results in parameters that are highly sensitive to small changes in the underlying physical parameters. Such models often fail when used with systems of higher order. Furthermore, even if the models are usable in continuous time, they can become even more sensitive and far less physical once the system is discretized. This is particularly true for mechatronic systems, which often are characterized by a "rigid body" model followed by multiple sharp resonances and anti-resonances.

All of this creates a situation where state space approaches are used only by experts in the field, while more basic, physically intuitive approaches continue to dominate in industrial applications. These intuitive methods may work fine when the system is low order, but they break down as the system complexity rises. What is needed is a form that can capture higher order dynamics in a way that maintains physical intuition and preserves numerical accuracy through the discretization process.

This section presents a new state space form, the analog Biquad State Space, based on The Multinotch structure [54, 33]. The biquad state space (BSS) [1] has several desirable characteristics:

- It uses a structure based on a serialized biquad filters which can be physically matched to resonance/anti-resonance pairs observed in measurements.

[1] Some have suggested Abramovitch State Space, but it suffers from a bad acronym.

- The number of parameters is comparable to that of a canonical form, although many appear multiple times in the matrix structures.
- The basic structure remains the almost the same through discretization.
- The underlying biquad structure leads to a state space structure that is numerically very stable, even through discretization. The $\Delta$-parameters from The Multinotch [33] can be used to improve the numerical accuracy of discretized coefficients [3], allowing this form to be implemented in fixed point math, such as that found on inexpensive DSP chips and FPGAs.

The biquad state space (BSS) was introduced in [3] discrete time, state space form that allowed the numerical resiliency of serial cascades of biquad filters to be moved into the state space world, while allowing for precalculation (Figure 9.18). This gave the structure excellent numerics and fixed and low latency, as previously described in [54]. The numerical robustness shown there is useful, even when minimal latency control was not an issue.

The rest of the chapter will be organized as follows. Section 9.18 discusses the discrete time matrix structure. Section 9.20 shows how to modify the BSS to add rigid body modes. Section 9.19 discusses a current mode BSS estimator. Continuous time biquads will be discussed in Section 9.22. The analog Biquad State Space form will be described in Section 9.23. The invariance of the BSS under discretization will be discussed in Section 9.24. Some discussion of the matrix form will be in Section 9.25. Discussion of lack of direct feedthrough will be in Section 9.27. Finally, Section 9.32 gives some examples that show the effectiveness of the BSS in modeling flexible dynamic systems, such as mechatronics.

While the structure is quite regular and works for large or small numbers of biquads, the regular pattern becomes obvious in the three biquad case. Thus, most of the structural equations will be three biquad ones. The format considerations of this will mean that many of these matrix equations are in two column figures, but seeing the matrices in this way makes the structural properties fairly obvious. This will result in some of the larger equations being pushed into two column figures.

## 9.18 The Matrices, Reloaded

Generating coefficients from continuous time biquad parameters is discussed in some detail in [54] and [33]. Suffice it to say that continuous time, physical parameters can be mapped into the discrete time biquads which form the basis of our state matrices.

The state transition matrix in Equation 9.84 has a very regular, block upper triangular form. On the block diagonals are $2 \times 2$ blocks with the biquad denominator parameters (from which we can extract the model poles). Below the diagonal blocks are empty, while above the diagonal blocks is a repeated set of $2 \times 2$ blocks with $0s$ on the lower rows and

$$\begin{bmatrix} \tilde{b}_{i,1} - a_{i,1} & \tilde{b}_{i,2} - a_{i,2} \end{bmatrix} \tag{9.89}$$

on the top row. The top rows of these blocks represent the feedthrough of the biquad states to the other states. Likewise in the output matrix of Equation 9.85, these same subsections in (9.89) represent the feedthrough of the biquad states to the outputs. Note that in both of these matrix equations, the input is passed unscaled to the states and unscaled outputs. The gain scaling is applied in (9.86).

Note that while these matrices are denser than a typical canonical form, many of the needed multiplications and additions are repeated, so that proper coding of the state and unscaled output updates makes this form no more computationally intense than a canonical form.

The above the block diagonal blocks are governed by the terms in (9.89), and these terms are determined by how the overall system model is partitioned into biquads. One way to minimize these terms is to arrange the pole-zero groupings so that each biquad consists of poles and zeros that are closest to each other.

We may wish to have low pass and/or high pass filters in our Multinotch, or single or double integrators in our BSS model of a physical system. We might note that while it is theoretically possible to add integrators into a Multinotch, say as part of a controller that includes integral action, practical implementation of integrators usually involves useful nonlinearities, such as integrator anti-windup [302] which necessitated the integrators being broken off from the rest of the filter. However, in linear state space models, single or double integrators are common in rigid body models. .

The common feature of most of these filters is the lack of direct feed through from the input to the output of any one filter stage, at least in the continuous time model. As noted briefly in [303, 304] this lack of direct feedthrough affects the propagation of states and the structure of the state space matrices.

## 9.19 Current Estimator and State Feedback

In a prediction estimator, the measurement error is formed using the previous measurement and a state output generated entirely from quantities available before the current measured output. This means that the BSS does not have a significant latency advantage in a predictor form observer, simply because the latter already has a full sample of latency. A current estimator, on the other hand, depends on the current measurement. It is for this type of estimator where we can get some latency savings as shown in Figure 9.16.

In order to use our form in an observer, we need to generate time update and measurement update equations. For the 3-biquad case, the time update equation is given by Equation 9.90. For the SISO case, there will only be a single output, and so the output equations become what is shown in Equation 9.93.

$$\begin{bmatrix} \bar{x}_{2,k} \\ \bar{x}_{2,k-1} \\ \bar{x}_{1,k} \\ \bar{x}_{1,k-1} \\ \bar{x}_{0,k} \\ \bar{x}_{0,k-1} \end{bmatrix} = \begin{bmatrix} -a_{21} & -a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -a_{11} & -a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -a_{01} & -a_{02} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \hat{x}_{2,k-1} \\ \hat{x}_{2,k-2} \\ \hat{x}_{1,k-1} \\ \hat{x}_{1,k-2} \\ \hat{x}_{0,k-1} \\ \hat{x}_{0,k-2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_{k-1}. \tag{9.90}$$

The state output equation is given by:

$$\begin{bmatrix} \bar{\tilde{y}}_{2,k} \\ \bar{\tilde{y}}_{1,k} \\ \bar{\tilde{y}}_{0,k} \end{bmatrix} = \begin{bmatrix} \tilde{b}_{21}-a_{21} & \tilde{b}_{22}-a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 0 & 0 & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \end{bmatrix} \begin{bmatrix} \bar{x}_{2,k} \\ \bar{x}_{2,k-1} \\ \bar{x}_{1,k} \\ \bar{x}_{1,k-1} \\ \bar{x}_{0,k} \\ \bar{x}_{0,k-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u_k \tag{9.91}$$

Finally, the properly scaled outputs are generated via:

$$\begin{bmatrix} \bar{y}_{2,k} \\ \bar{y}_{1,k} \\ \bar{y}_{0,k} \end{bmatrix} = \begin{bmatrix} b_{20}b_{10}b_{00} & 0 & 0 \\ 0 & b_{10}b_{00} & 0 \\ 0 & 0 & b_{00} \end{bmatrix} \begin{bmatrix} \tilde{y}_{2,k} \\ \tilde{y}_{1,k} \\ \tilde{y}_{0,k} \end{bmatrix}. \tag{9.92}$$

For the SISO case, there will only be a single output, and so the output equations become:

$$\left[ \; \bar{\tilde{y}}_{2,k} \; \right] = \left[ \; \tilde{b}_{21} - a_{21} \quad \tilde{b}_{22} - a_{22} \quad \tilde{b}_{11} - a_{11} \quad \tilde{b}_{12} - a_{12} \quad \tilde{b}_{01} - a_{01} \quad \tilde{b}_{02} - a_{02} \; \right] \begin{bmatrix} \hat{x}_{2,k-1} \\ \hat{x}_{2,k-2} \\ \hat{x}_{1,k-1} \\ \hat{x}_{1,k-2} \\ \hat{x}_{0,k-1} \\ \hat{x}_{0,k-2} \end{bmatrix} + \left[ \; 1 \; \right] u_{k-1} \tag{9.93}$$

Finally, the properly scaled time update output is generated via a single multiplication of concatenated feedthrough coefficients, in a similar manner to [54].

$$\overline{y}_k = \overline{y}_{2,k} = b_{20} b_{10} b_{00} \tilde{y}_{2,k}. \tag{9.94}$$

For the SISO measurement update, the equations are quite simple:

$$e_k = y_{meas,k} - \overline{y}_k, \text{ and} \tag{9.95}$$

$$\hat{x}_k = \overline{x}_k + L_c e_k. \tag{9.96}$$

Now, Equation 9.95 involves one subtraction. Equation 9.96 involves one multiply and addition for each state, but these are independent and so can be done in parallel. The latency then, for the state state update, is that of 2 multiplies, plus 3 add/subtract operations, independent of the size of the state. To use the state estimate in state feedback would require

$$u_{fb,k} = K_{fb} \hat{x}_k = K_{fb} \overline{x}_k + K_{fb} L_c e_k, \tag{9.97}$$

in which the $K_{fb}\overline{x}_k$ and the $K_{fb}L_c$ products can be precalculated. Thus for a SISO system, state feedback involves one more multiply and one addition.

## 9.20 Adding Rigid Body Dynamics: Double Integrator

In filtering problems, there is rarely need for an integrator. Even a PID constructed as a filter will rarely use one of these blocks because of the desire to add features such as anti-windup to the integrator. However, when modeling any real mechatronic system, there will have to be some sort of rigid body or low frequency resonance model. In this section, we will show how to add a double integrator to this biquad structure. The simplest way, of course, would be if the double integrator could just be modeled as a biquad. Defining our double integrator as $D(s) = K/s^2$ and applying the Trapezoidal rule yields

$$D_T(z^{-1}) = K\left(\frac{T}{2}\right)^2 \left(\frac{1+z^{-1}}{1-z^{-1}}\right)^2 . \tag{9.98}$$

Neglecting the gain, $K\left(\frac{T}{2}\right)^2$, we define

$$\tilde{D}_T(z^{-1}) = \left(\frac{1+z^{-1}}{1-z^{-1}}\right)^2 = \frac{1+2z^{-1}+z^{-2}}{1-2z^{-1}+z^{-2}} \tag{9.99}$$

from which we can extract the time domain equations

$$d_k - 2d_{k-1} + d_{k-2} = u_k. \tag{9.100}$$

Remembering that in the traditional state-space notation $x_{k+1} = d_k$ we get

$$x_{k+1} = 2x_k - x_{k-1} + u_k \tag{9.101}$$

and

$$\tilde{y}_{k+1} = x_{k+1} + 2x_k + x_{k-1}. \tag{9.102}$$

Note that $\tilde{y}_{k+1}$ depends upon $x_{k+1}$ which we have defined in terms of previous values of $x_k$ and the current input, $u_k$, so we can make the substitutions to get

$$\begin{aligned} \tilde{y}_{k+1} &= x_{k+1} + 2x_k + x_{k-1} \\ &= 2x_k - x_{k-1} + u_k + 2x_k + x_{k-1} \\ &= 4x_k + u_k. \end{aligned} \tag{9.103}$$

We put this in state space form as:

$$\begin{bmatrix} x_{k+1} \\ x_k \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ x_{k-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k. \tag{9.104}$$

The output is defined as:

$$[\tilde{y}_{k+1}] = \begin{bmatrix} 4 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ x_{k-1} \end{bmatrix} + \begin{bmatrix} 1 \end{bmatrix} u_k. \tag{9.105}$$

Finally,

$$[y_{k+1}] = \left[KT^2/4\right] [\tilde{y}_{k+1}] . \tag{9.106}$$

This is great news. What we have seen is that we can treat a double integrator as a digital biquad, and so we can drop it right into our structure, simply by choosing

$$\begin{aligned} &a_1 = -2, \quad a_2 = 1, \\ &\tilde{b}_1 = 2, \quad\ \ \tilde{b}_1 = 1, \quad \text{and } b_0 = \tfrac{KT^2}{4}. \end{aligned} \tag{9.107}$$

What we have done in this section is really only the first cut at adding rigid body dynamics to a biquad state space (BSS) structure. In this, we have discretized the double integrator with a trapezoidal rule (TR) equivalent , in part because this discretization of the double integrator preserves the direct feedthrough structure that we use in the BSS. This presents three problems. First, it limits our discretization choices in a way that we may not want. We have seen that for a PID controller, a seemingly non-ideal discretization (the backwards rectangular rule (BR) equivalent) has some unexpected benefits that make it the most likely choice. The second issue is that if we want to have continuous time BSS models (which are about to be introduced below, so spoiler alert, we do, Section 9.23), then we will have some filters such as integrators and low pass filters, or some model structures that are modeled as those, that will not have direct feedthrough. We will need to show how to work those into our BSS structure if we want to make this universally useful. The third is that if we look at the discrete double integrator produced above, we do not have direct access to the velocity term. We should consider the irony of adding in all this mathematical machinery under the promise that it would give us access to more of the real world signals, and then generating models that do not even give us both position and velocity. Were this a cartoon, we would be playing the part of Wiley E. Coyote here. A structure that helps us be a bit more clever like Bugs Bunny will be explained in Section 9.28.

## 9.21 Discrete Time Examples

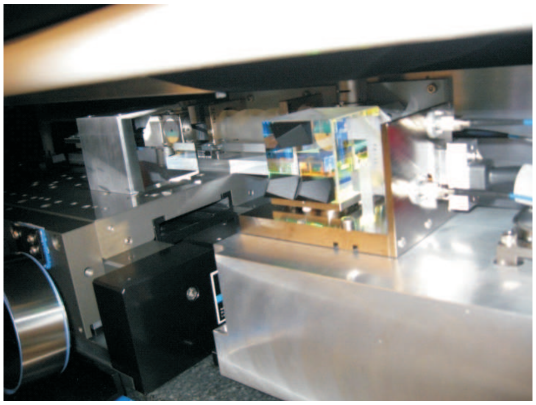


Figure 9.19: Laboratory system: Aerotech air bearing linear stage, including linear grating for position measurement. The Aerotech system implements a PID controller and samples the data at 8 kHz. It has a built in swept-sine measurement. In the center of the image is a laser interferometer (IF) to provide an alternate measurement of the stage position. The stage itself is to the left of the IF.

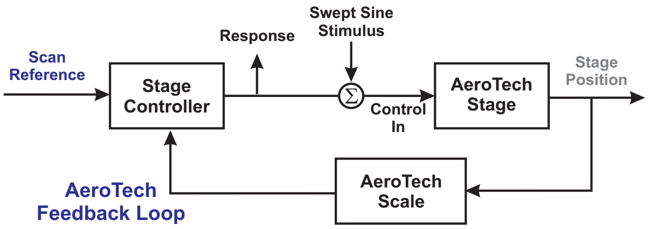


Figure 9.20: Conceptual block diagram of AeroTech stage measurement for frequency response measurements.

In order to demonstrate the numerical improvements arising from the biquad state space structure, an example is take from measurements of an Aerotech linear stage used in experiments for the Quintessential Phase project [305]. The Aerotech single axis stage as shown in Figure 9.19. The basic measurement set-up is diagrammed in Figure 9.20. The Aerotech 3300 stage controller includes a PID like feedback controller along with a feedforward portion. The sample rate for these is 8 kHz. In order to obtain a clean frequency response, the Eric Johnstone [305] turned off the feedforward compensator and then used the Aerotech controller's built in swept-sine functionality. A 1000 point swept-sine frequency response function (FRF) was taken on a logarithmic frequency axis from 10 Hz to 4 kHz. The Aerotech controller returned an open loop FRF, which was uploaded to MATLAB . There a model of the Aerotech PID was constructed using Aerotech parameters. A FRF for this controller was synthesized on the same frequency axis as the stage open loop response measurement, and this controller FRF was divided out of the open loop FRF to obtain a "plant" FRF. This plant FRF was fit to a stage model that consisted of a double integrator plus 20 analog biquads. The biquads are ranked in order of significance on the frequency response so that if one wants to simplify the model, one removes the latter biquads. The identified model parameters are in Table 9.3.

In order to compare the biquad state space to more conventional methods, the fit parameters were then used to generate both transfer function models and state space models in Matlab. The linear system concatenation functions were used for both of these. From these high order models, Bode plots were generated to compare to the original measurement. Similarly, model terms were used to construct a biquad state space structure and again, a Bode plot was generated. Note that these plots are not made using fixed point math, but with all terms represented in Matlab's dual precision floating point format.

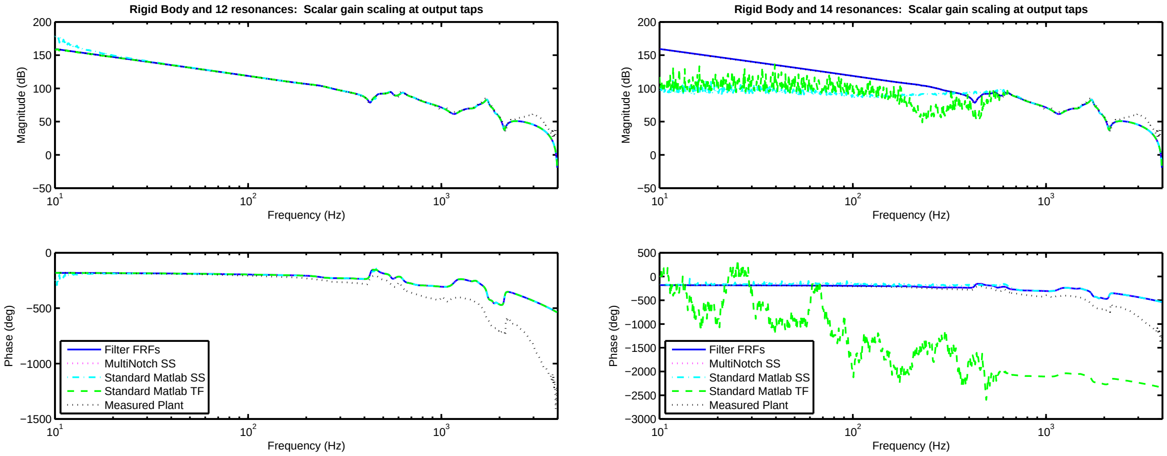
On the left side of Figure 9.21, we see that with up to 12 biquads and a rigid body, all the methods produce essentially equivalent Bode plots, that match the magnitude data exceptionally well. The phase features are matched, with the exception of the general rolloff that can be attributed to time delay not modeled in the rigid body or the biquads.

| **20 Biquad Fit Parameters for Aerotech Stage** | | | | |
|---|---|---|---|---|
| **Biquad #** | $f_{N,n}$ **(Hz)** | $Q_n$ | $f_{N,d}$ **(Hz)** | $Q_d$ |
| 1 | 2116.5 | 46.9420 | 1871.2 | 9.0188 |
| 2 | 1162.6 | 9.3768 | 1301.8 | 1.9112 |
| 3 | 619.9 | 4.8004 | 631.5 | 13.2948 |
| 4 | 1792.6 | 13.1546 | 1726.6 | 27.3882 |
| 5 | 702.0 | 0.3261 | 1374.2 | 0.1245 |
| 6 | 428.7 | 25.2154 | 449.4 | 7.3544 |
| 7 | 559.7 | 10.4940 | 549.4 | 18.6003 |
| 8 | 248.3 | 2.5601 | 241.9 | 3.2069 |
| 9 | 1891.1 | 31.8588 | 1874.3 | 21.0130 |
| 10 | 1484.5 | 14.0540 | 1506.3 | 11.2718 |
| 11 | 720.5 | 5.0254 | 718.5 | 7.0045 |
| 12 | 458.0 | 24.6218 | 459.3 | 19.0552 |
| 13 | 287.8 | 9.7413 | 286.8 | 9.8888 |
| 14 | 225.7 | 14.7047 | 225.4 | 14.0349 |
| 15 | 3590.2 | 7.4186 | 3203.0 | 10.2562 |
| 16 | 2159.3 | 60.0000 | 2143.5 | 21.8389 |
| 17 | 1947.3 | 11.7009 | 1954.3 | 9.3325 |
| 18 | 1982.2 | 9.2703 | 1982.1 | 9.2825 |
| 19 | 1936.4 | 9.2163 | 1936.5 | 9.2002 |
| 20 | 2128.2 | 60.0000 | 2121.7 | 84.6660 |

Table 9.3: Model parameters from curve fit of Aerotech frequency response data.

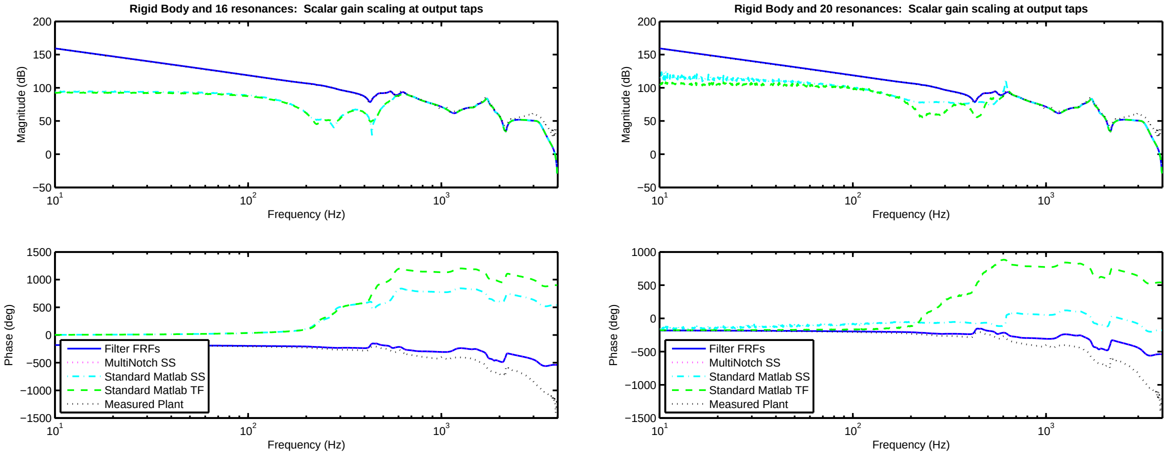
However, just the addition of two more biquads (on the right side of Figure 9.21), we see that the two "conventional" methods deviate significantly from the measured frequency response.

At 16 biquads (left side of Figure 9.22), the two conventional methods are more self consistent, but in some ways worse than at 14 biquads. At 20 biquads plus the rigid body as shown on the right side of Figure 9.22, it is very clear that the conventional methods are so affected by numerical issues that they cannot come close to representing the measurement, either a low frequency or high frequency. In both of these cases, we see that the biquad state space continues to match the original measurement.


Rigid Body and 12 resonances:  Scalar gain scaling at output taps

Rigid Body and 14 resonances:  Scalar gain scaling at output taps

Figure 9.21:   Comparing state space forms to AeroTech stage frequency response.  Modeling the system with first 12 biquads and a rigid body, there is no discernible difference in the plots.  The measured plant exhibits a phase roll off at high frequency not fit by the biquads.Modeling the system with first 14 biquads and a rigid body, we start seeing significant differences in the different methods of realizing the state space form.  The conventional methods are clearly not matching the measured Aerotech frequency response, while the biquad state space method is.


Rigid Body and 16 resonances:  Scalar gain scaling at output taps

Rigid Body and 20 resonances:  Scalar gain scaling at output taps

Figure 9.22: Comparing state space forms to Aerotech stage frequency response.  On the left:  Modeling the system with first 16 biquads and a rigid body.  On the right:  Modeling the system with first 20 biquads and a rigid body, Both plots show the significant differences in the different methods of realizing the state space form get worse.  Again, the conventional methods are clearly not matching the measured Aerotech frequency response, while the biquad state space method is.

## 9.22 Continuous Time Biquads

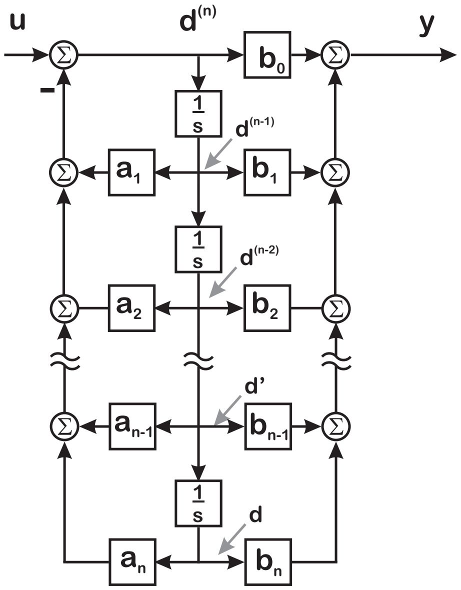


Figure 9.23: An n<sup>th</sup> order continuous-time, polynomial filter in Direct Form II configuration similar to the discrete-time filter form in [3, 167].

A standard Single-Input, Single-Output (SISO) transfer function is shown in Figure 9.23, and represented as transfer function by

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \ldots + a_n}. \tag{9.108}$$

We can consider such high order polynomial transfer functions as filter models and factor these into a series of biquad filters such as:

$$\frac{Y(s)}{U(s)} = \left(\frac{b_{00}s^2 + b_{01}s + b_{02}}{s^2 + a_{01}s + a_{02}}\right)\left(\frac{b_{10}s^2 + b_{11}s + b_{12}}{s^2 + a_{11}s + a_{12}}\right)\cdots\left(\frac{b_{m0}s^2 + b_{m1}s + b_{m2}}{s^2 + a_{m1}s + a_{m2}}\right). \tag{9.109}$$

If $n$ is even, then the number of biquads, $m$ is set to $n/2$. If $n$ is odd, then there are are $(n+1)/2$ biquads, but the last one is first order filter (by setting $b_{m,2} = a_{m,2} = 0$. As was shown in [54, 3], there can be advantages to factoring out the direct feedthrough gains, resulting in

$$\frac{Y(s)}{U(s)} = b_{00}\left(\frac{s^2 + \tilde{b}_{01}s + \tilde{b}_{02}}{s^2 + a_{01}s + a_{02}}\right) b_{10}\left(\frac{s^2 + \tilde{b}_{11}s + \tilde{b}_{12}}{s^2 + a_{11}s + a_{12}}\right)\cdots b_{m0}\left(\frac{s^2 + \tilde{b}_{m1}s + \tilde{b}_{m2}}{s^2 + a_{m1}s + a_{m2}}\right), \tag{9.110}$$

where $\tilde{b}_{ij} = \frac{b_{ij}}{b_{i0}}$. Note that if $b_{i0} = 0$ and $b_{i1} \neq 0$, then the factored out gain is $b_{i1}$. Likewise if both $b_{i0}$ and $b_{i1}$ are $0$, then then the factored out gain is $b_{i2}$.

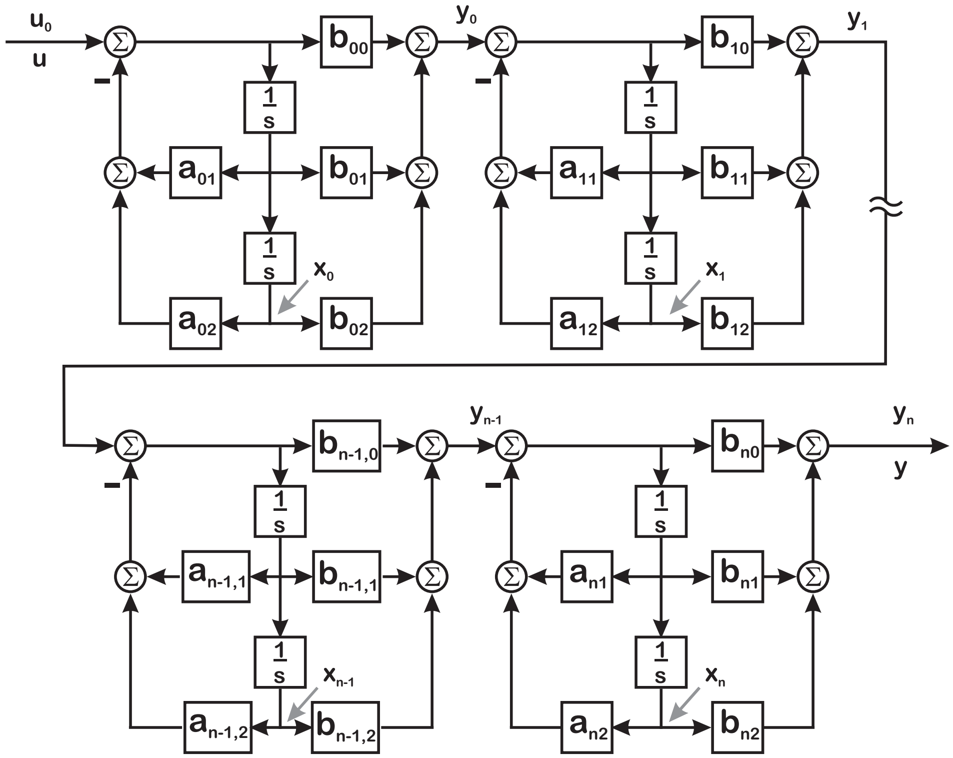


Figure 9.24: Continuous biquad blocks in controller canonical form with states noted.

If we call this transfer function $H(s)$, then $H(s) = b_{m0} \cdots b_{10} b_{20} \tilde{H}(s)$ which gives:

$$\tilde{H}(s) = \left(\frac{s^2 + \tilde{b}_{01}s + \tilde{b}_{02}}{s^2 + a_{01}s + a_{02}}\right)\left(\frac{s^2 + \tilde{b}_{11}s + \tilde{b}_{12}}{s^2 + a_{11}s + a_{12}}\right)\cdots\left(\frac{s^2 + \tilde{b}_{m1}s + \tilde{b}_{m2}}{s^2 + a_{m1}s + a_{m2}}\right). \tag{9.111}$$

Again, if one of the $b_{i0}$ terms is $0$ it is replaced by the first non-zero $b_{i1}$ or $b_{i2}$ term.

Returning to a more modal representation, a single biquad can be represented as:

$$B(s) = K\left(\frac{s^2 + 2\varsigma_n\omega_n s + \omega_n^2}{s^2 + 2\varsigma_d\omega_d s + \omega_d^2}\right) \tag{9.112}$$

which in turn can be represented in a two step differential form as:

$$\begin{aligned} \ddot{x} &+ 2\varsigma_d\omega_d\dot{x} + \omega_d^2 x = u \\ y &= K\left(\ddot{x} + 2\varsigma_n\omega_n\dot{x} + \omega_n^2 dx\right) \end{aligned} \tag{9.113}$$

This can be represented in state space form as:

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -2\varsigma_d\omega_d & -\omega_d^2 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \tag{9.114}$$

and

$$y = K\left[\begin{array}{cc} 2\varsigma_n\omega_n & \omega_n^2 \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] + K\ddot{x} \tag{9.115}$$

but we need to get rid of $\ddot{x}$ and get the output in terms of the actual state vector:

$$\begin{aligned} y &= K\left[\begin{array}{cc} 2\varsigma_n\omega_n & \omega_n^2 \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] \\ &\quad -K\left[\begin{array}{cc} 2\varsigma_d\omega_d & \omega_d^2 \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] + \left[\begin{array}{c} K \\ 0 \end{array}\right] u \end{aligned} \tag{9.116}$$

$$\begin{aligned} y &= \left[\begin{array}{cc} K(2\varsigma_n\omega_n - 2\varsigma_d\omega_d) & K(\omega_n^2 - \omega_d^2) \end{array}\right]\left[\begin{array}{c} \dot{x} \\ x \end{array}\right] \\ &\quad + \left[\begin{array}{c} K \\ 0 \end{array}\right] u \end{aligned} \tag{9.117}$$

What is important in this structure is that the output depends on the first two states and the input. In this case, the input can feed through directly. Now, we would like to move this to a more general form such as we had in [54] and [3], so we replace these resonance parameters with filter coefficients:

$$\left[\begin{array}{c} \ddot{x}_i \\ \dot{x}_i \end{array}\right] = \left[\begin{array}{cc} -a_{i1} & -a_{i2} \\ 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{x}_i \\ x_i \end{array}\right] + \left[\begin{array}{c} 1 \\ 0 \end{array}\right] u_i \tag{9.118}$$

while the state output equation is given by:

$$\left[\begin{array}{c} \tilde{y}_i \end{array}\right] = \left[\begin{array}{cc} \tilde{b}_{i1} - a_{i1} & \tilde{b}_{i2} - a_{i2} \end{array}\right]\left[\begin{array}{c} \dot{x}_i \\ x_i \end{array}\right] + \left[\begin{array}{c} 1 \end{array}\right] u_i \tag{9.119}$$

Finally, the properly scaled output is generated via:

$$\left[\begin{array}{c} y_i \end{array}\right] = \left[\begin{array}{c} b_{i0} \end{array}\right]\left[\begin{array}{c} \tilde{y}_i \end{array}\right]. \tag{9.120}$$

## 9.23 The Analog Biquad State Space Form

If we have multiple biquads of the form shown in Equations 9.118, 9.119, and 9.120, we can chain these together by noting that:

$$\begin{aligned} u_i &= \tilde{y}_i, & &\text{for } 0 \le i < n, \\ u_0 &= u, & &\text{and} \\ \tilde{y}_n &= \tilde{y}. \end{aligned} \tag{9.121}$$

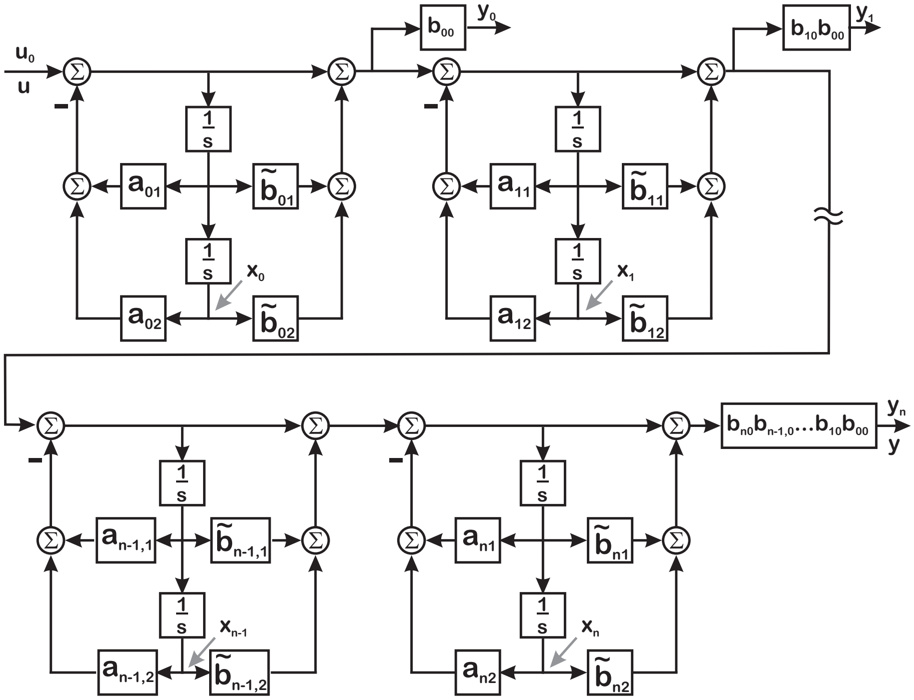


Figure 9.25: The analog biquad cascade, with factored out $b_{i,0}$ terms and scaling the output of each block. This is completely analogous to the digital form of Figure 9.18

If one is willing to go through the algebraic pain and suffering of applying Equation 9.121 to each biquad structure a very regular state space structure results.

$$\begin{bmatrix} \ddot{\tilde{x}}_2 \\ \dot{\tilde{x}}_2 \\ \ddot{\tilde{x}}_1 \\ \dot{\tilde{x}}_1 \\ \ddot{\tilde{x}}_0 \\ \dot{\tilde{x}}_0 \end{bmatrix} = \begin{bmatrix} -a_{21} & -a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -a_{11} & -a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -a_{01} & -a_{02} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\tilde{x}}_2 \\ \tilde{x}_2 \\ \dot{\tilde{x}}_1 \\ \tilde{x}_1 \\ \dot{\tilde{x}}_0 \\ \tilde{x}_0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} u \qquad (9.122)$$

$$\begin{bmatrix} \tilde{y}_2 \\ \tilde{y}_1 \\ \tilde{y}_0 \end{bmatrix} = \begin{bmatrix} \tilde{b}_{21}-a_{21} & \tilde{b}_{22}-a_{22} & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & \tilde{b}_{11}-a_{11} & \tilde{b}_{12}-a_{12} & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \\ 0 & 0 & 0 & 0 & \tilde{b}_{01}-a_{01} & \tilde{b}_{02}-a_{02} \end{bmatrix} \begin{bmatrix} \dot{\tilde{x}}_2 \\ \tilde{x}_2 \\ \dot{\tilde{x}}_1 \\ \tilde{x}_1 \\ \dot{\tilde{x}}_0 \\ \tilde{x}_0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u. \qquad (9.123)$$

Finally, the properly scaled outputs are generated via:

$$\begin{bmatrix} y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} b_{20}b_{10}b_{00} & 0 & 0 \\ 0 & b_{10}b_{00} & 0 \\ 0 & 0 & b_{00} \end{bmatrix} \begin{bmatrix} \tilde{y}_2 \\ \tilde{y}_1 \\ \tilde{y}_0 \end{bmatrix}. \tag{9.124}$$

This structure has a very regular iteration which continues with the addition of extra biquads. It is worth noting several properties of this structure.

- First of all, it is a relatively sparse structure where a lot of the multiplies are 1.
- Secondly, we have put off multiplying by gain terms until the end. This provides the same input output behavior as the transfer function model, although the internal states may not be scaled the same way as the internal signals in the biquad chain. We will discuss alternate choices of where to assign gain scaling in [306].
- The eigenvalues of the state matrix are still defined by the denominator terms of the transfer function, and these show up in the "block diagonals" of the state matrix.
- The off diagonals contain differences of the numerator and denominator coefficients. Proper selection of these terms can minimize these differences and keep the size of the off diagonal terms well constrained.

It is worth discussing what it means to select these terms, the $\tilde{b}_{i1} - a_{i1}$ and $\tilde{b}_{i2} - a_{i2}$ terms. In the case of a biquad,

$$\tilde{b}_{i1} - a_{i1} = 2\varsigma_{in}\omega_{in} - 2\varsigma_{id}\omega_{id} \text{ and } \tilde{b}_{i2} - a_{i2} = \omega_{in}^2 - \omega_{id}^2. \tag{9.125}$$

This structure then allows the designer to pick pole/zero or resonance/ant-resonance combinations that minimize the off diagonal terms in the system matrix, the $\tilde{b}_{i1} - a_{i1}$ and $\tilde{b}_{i2} - a_{i2}$ terms, as well as their effect on the output.

What we will see in the next section, is that the biquad matrix structure is the same for discrete time biquads, although the physical interpretation of the coefficients is different. However, it is helpful to keep in mind the similarity of the numerical structures.

## 9.24 Discretization of the Analog BSS

One major difference in using the BSS compared to general textbook methods is that we choose to discretize the BSS on a biquad by biquad basis. While this looses the satisfaction of analytical mathematical exactness, it does have the following positive properties:

1) Discretization approximations, and therefore discretization errors, are on a biquad by biquad basis. This has the potential to bound the error growth as the number of biquads (and therefore the number of states) grows.

2) The discretization method most appropriate to any one biquad can be applied independently of how adjacent biquads are discretized. For example, with lightly damped resonance/anti-resonance pairs, the pole zero mapping used in [54] and the $\Delta$ coefficients of [33] work extremely well. On the other hand, representing a double integrator as a discrete biquad can be accomplished using a Trapezoidal Rule equivalent [15] as described in [3].

3) Moreover, discretizing on a biquad by biquad basis means that the digital BSS for a given system has largely the same block structure as its analog BSS.

If one considers debugging a physical system, the importance of the last item cannot be overstated. The "invariance under discretization" means that a discrete state space model can be compared to an analog state space model or to modal test points on a physical system. It means that we can closely relate the digital state space model to the physics of the problem, and therefore to physical measurements of real systems.

This means that analog biquads are mapped to digital biquads, and while the meanings of the coefficients change, the matrix structure is largely invariant.

The simple reason is that each biquad in the matrix structure is preserved – with different coefficients. Thus, taken two at a time, the states are invariant under discretization. The consequence of this cannot be understated, as an analog biquad decomposition is very a "physical" model of a system, similar to a modal decomposition. As the obfuscation of physical, analog parameters by discretization is contained within each biquad,

## 9.25 The Matrices, Reloaded, Part Deux

Generating coefficients from continuous time biquad parameters is discussed in some detail in [54] and [33]. Suffice it to say that continuous time, physical parameters can be mapped into the discrete time biquads which form the basis of our state matrices.

The state transition matrix in Equation 9.122 has a very regular, block upper triangular form. On the block diagonals are $2 \times 2$ blocks with the biquad denominator parameters (from which we can extract the model poles). Below the diagonal blocks are empty, while above the diagonal blocks is a repeated set of $2 \times 2$ blocks with $0s$ on the lower rows and

$$\left[ \begin{array}{cc} \tilde{b}_{i,1} - a_{i,1} & \tilde{b}_{i,2} - a_{i,2} \end{array} \right] \tag{9.126}$$

on the top row. The top rows of these blocks represent the feedthrough of the biquad states to the other states. Likewise in the output matrix of Equation 9.123, these same subsections in (9.126) represent the feedthrough of the biquad states to the outputs. Note that in both of these matrix equations, the input is passed unscaled to the states and unscaled outputs. The gain scaling is applied in (9.124).

Note that while these matrices are denser than a typical canonical form, many of the needed multiplications and additions are repeated, so that proper coding of the state and unscaled output updates makes this form no more computationally intense than a canonical form.

The above the block diagonal blocks are governed by the terms in (9.126), and these terms are determined by how the overall system model is partitioned into biquads. One way to minimize these terms is to arrange the pole-zero groupings so that each biquad consists of poles and zeros that are closest to each other.

## 9.26 Continuous Time Rigid Body Dynamics and Low Pass Filters

The most common rigid body models we might see would be a double integrator as shown in the tortured biquad form of Figure 9.26 with $a_1 = 0$ or an integrator plus low pass form ($a_1 > 0$), where there is a real stable pole in place of one of the integrators. Spring-mass-damper actuators, such as those in an atomic force microscope (AFM) [307, 308] would require a more difficult access to velocity. The double

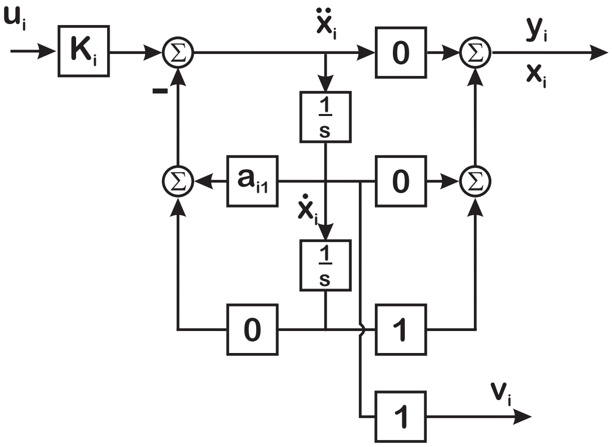


Figure 9.26: Continuous time (CT) rigid body biquad. Setting $a_1 = 0$ turns it into a CT double integrator.

integrator is modeled as:

$$D(s) = \frac{K}{s^2}, \tag{9.127}$$

while the single pole rigid body is modeled as

$$D(s) = \frac{Ka_1}{s(s + a_1)}. \tag{9.128}$$

where $a_1$ might be viscous friction applying velocity feedback.

Let's consider a few forms of continuous time low pass filters (CT-LPF). When possible, we will set the DC gain to $1$ as a common scaling. A pair of first order models are presented in:

$$L_{1,a}(s) = \frac{a}{s+a} \quad \text{and} \tag{9.129}$$

$$L_{1,b}(s) = \left(\frac{a}{b}\right)\frac{s+b}{s+a}. \tag{9.130}$$

In the case of (9.129), it is low pass because the "zero" is at infinite frequency. At some point, for positive $a$, it has to roll off. Equation (9.130) is only a low pass filter if $0 \leq a < b$. It doesn't have infinite rejection at infinite frequency. It is a lag filter, where the response at low frequency is higher than the response at high frequency, and the level of attenuation is set by the distance between $a$ and $b$. Our method of translating filters from continuous time to discrete time in the multi-notch is based on pole-zero mapping, and this has worked fine as long as the zeros were finite, so there is no problem with (9.130). Likewise, there would be no problem with:

$$L_{2,b2}(s) = \left(\frac{a_2}{b_2}\right)\frac{s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}. \tag{9.131}$$

Equation (9.129) is a different matter as are:

$$L_{2,b0}(s) = \left(\frac{a_2}{b_2}\right)\frac{b_2}{s^2 + a_1 s + a_2}, \text{ and} \tag{9.132}$$

$$L_{2,b1}(s) = \left(\frac{a_2}{b_1}\right)\frac{s + b_1}{s^2 + a_1 s + a_2}. \tag{9.133}$$

Equations (9.129), (9.132), and (9.133) all have zeros when $|s| \longrightarrow \infty$ or when $s$ is evaluated on the $j\omega$ axis, when $|\omega| \longrightarrow \infty$. A general form for such structures is diagrammed in Figure 9.27 and will be discussed in Section 9.27.

## 9.27 Handling the Lack of Direct Feedthrough

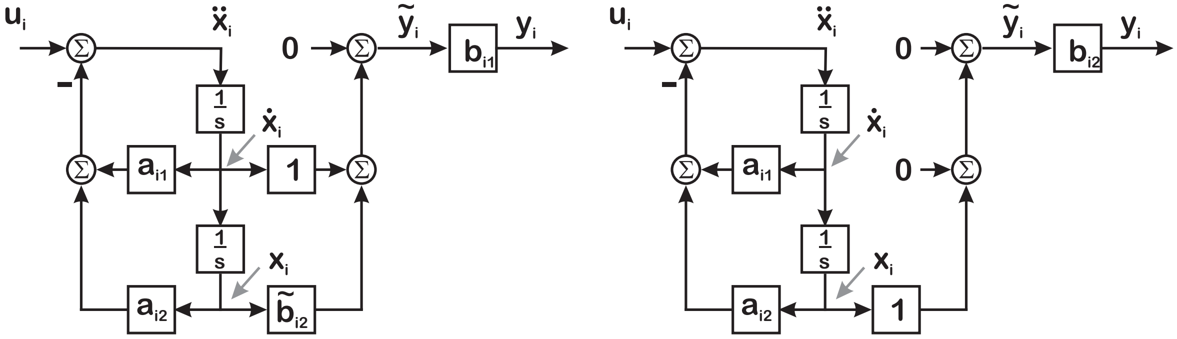


Figure 9.27: Analog biquads without direct feedthrough. On the left, $b_{i0} = 0$. On the right, both $b_{i0}$ and $b_{i1} = 0$. In either case, the leading gain is the gain of the highest order numerator term that has a non-zero coefficient. In either case, the lack of direct feedthrough means that the output is only determined by the state of the block. All downstream blocks from this one will not have direct feedthrough from the cascade input to the cascade output.

One of the nice properties of the BSS is that it handles direct feedthrough from the input to the output in a systematic structure. In the discrete time world, we can provide direct feedthrough for models of analog systems by choice of discretization method. For example, the analog double integrator inserted into the discrete BSS in [3] was discretized with the Trapezoidal Rule approximation, which gave it direct feedthrough. In the analog world, the rationale for this does not exist and since most mechatronic systems have some sort of low frequency behavior that has a pole zero excess (e.g. double integrator, simple resonance), we need to know how to accommodate this.

Our focus on LPF and rigid body models raised the importance of entering models with no direct feedthrough into the BSS and MNF. One of the nice properties of the BSS is that it handles direct feedthrough from the input to the output in a systematic structure. In the discrete time world, we can provide direct feedthrough for models of analog systems by choice of discretization method, as described in Section 9.29. In the analog world, the rationale for this does not exist and since most mechatronic systems have some sort of low frequency behavior that has a pole-zero excess, we need to know how to accommodate this.

Figure 9.27 shows two examples of biquads tasked with modeling such systems. On the left side is a biquad model for a system where only $b_{i0} = 0$. This would model a pole zero excess of $1$. On the right, both $b_{i0}$ and $b_{i1}$ are $0$. In either case, we factor out the non-zero $b_{ij}$ corresponding to the highest order. This will be used in our downstream gain calculations. Note that when any such biquad is in the chain, the direct feedthrough from the system input, $u$, to any of the downstream inputs and outputs, $u_j$ and $y_k$ for $j > i$ and $k \geq i$, is $0$. This affects the form of our state matrices.

In both cases, the state equation from (9.118) is unchanged. However, the state output equations change a lot. In the left hand case, Equation 9.119 becomes

$$\left[\ \tilde{y}_i\ \right] = \left[\ 1 \quad \tilde{b}_{i2}\ \right]\left[\begin{array}{c} \dot{x}_i \\ x_i \end{array}\right] + \left[\ 0\ \right] u_i \tag{9.134}$$

where $\tilde{b}_{i2} = b_{i2}/b_{i1}$ and (9.120) becomes:

$$\left[\ y_i\ \right] = \left[\ b_{i1}\ \right]\left[\ \tilde{y}_i\ \right]. \tag{9.135}$$

In the right hand case, Equation 9.119 becomes

$$\left[\ \tilde{y}_i\ \right] = \left[\ 0 \quad 1\ \right]\left[\begin{array}{c} \dot{x}_i \\ x_i \end{array}\right] + \left[\ 0\ \right] u_i \tag{9.136}$$

and (9.120) becomes:

$$\left[\ y_i\ \right] = \left[\ b_{i2}\ \right]\left[\ \tilde{y}_i\ \right]. \tag{9.137}$$

This may seem like an awful lot of bookkeeping for such a simple concept, but doing this bookkeeping allows us to maintain a the overall system structure, which allows us to write scripts and programs to build up BSS matrices from individual biquad models.

To illustrate this, consider a 4-biquad system model. We choose $b_{i0} = 0$ for biquad $1$. (Here the first biquad in the chain is biquad $0$ and the last one is biquad $3$. Again, algebraic pain and suffering results in a very regular state space structure. For our 4-biquad model, we get the state equation of 9.138. The unscaled output is in Equation 9.139, both displayed in Figure 9.28 due to their size. Finally, the

$$\begin{bmatrix} \ddot{\tilde{x}}_3 \\ \dot{\tilde{x}}_3 \\ \ddot{\tilde{x}}_2 \\ \dot{\tilde{x}}_2 \\ \ddot{\tilde{x}}_1 \\ \dot{\tilde{x}}_1 \\ \ddot{\tilde{x}}_0 \\ \dot{\tilde{x}}_0 \end{bmatrix} = \begin{bmatrix} -a_{31} & -a_{32} & \tilde{b}_{21} - a_{21} & \tilde{b}_{22} - a_{22} & \tilde{b}_{11} & \tilde{b}_{12} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -a_{21} & -a_{22} & \tilde{b}_{11} & \tilde{b}_{12} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -a_{11} & -a_{12} & \tilde{b}_{01} - a_{01} & \tilde{b}_{02} - a_{02} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -a_{01} & -a_{02} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\tilde{x}}_3 \\ \tilde{x}_3 \\ \dot{\tilde{x}}_2 \\ \tilde{x}_2 \\ \dot{\tilde{x}}_1 \\ \tilde{x}_1 \\ \dot{\tilde{x}}_0 \\ \tilde{x}_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} u \tag{9.138}$$

$$\begin{bmatrix} \tilde{y}_3 \\ \tilde{y}_2 \\ \tilde{y}_1 \\ \tilde{y}_0 \end{bmatrix} = \begin{bmatrix} \tilde{b}_{31} - a_{31} & \tilde{b}_{32} - a_{32} & \tilde{b}_{21} - a_{21} & \tilde{b}_{22} - a_{22} & \tilde{b}_{11} & \tilde{b}_{12} & 0 & 0 \\ 0 & 0 & \tilde{b}_{21} - a_{21} & \tilde{b}_{22} - a_{22} & \tilde{b}_{11} & \tilde{b}_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \tilde{b}_{11} & \tilde{b}_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \tilde{b}_{01} - a_{01} & \tilde{b}_{02} - a_{02} \end{bmatrix} \begin{bmatrix} \dot{\tilde{x}}_3 \\ \tilde{x}_3 \\ \dot{\tilde{x}}_2 \\ \tilde{x}_2 \\ \dot{\tilde{x}}_1 \\ \tilde{x}_1 \\ \dot{\tilde{x}}_0 \\ \tilde{x}_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u. \tag{9.139}$$

Figure 9.28: State equations for continuous time biquad state space with scalar output scaling. Biquad 1 lacks direct feedthrough.

properly scaled outputs are generated via:

$$\begin{bmatrix} y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} b_{30}b_{20}b_{1x}b_{00} & 0 & 0 & 0 \\ 0 & b_{20}b_{1x}b_{00} & 0 & 0 \\ 0 & 0 & b_{12}b_{00} & 0 \\ 0 & 0 & 0 & b_{00} \end{bmatrix} \begin{bmatrix} \tilde{y}_3 \\ \tilde{y}_2 \\ \tilde{y}_1 \\ \tilde{y}_0 \end{bmatrix}, \tag{9.140}$$

where $b_{ix} = b_{i1}$ if $b_{i1} \neq 0$ and $b_{ix} = b_{i2}$ if $b_{i1} = 0$. In Equations 9.138 and 9.139 $\tilde{b}_{11} = 1$ and $\tilde{b}_{12} = b_{12}/b_{11}$ if $b_{10} = 0$ and $b_{11} \neq 0$. Similarly, if $b_{10} = 0$ and $b_{11}00$, then $\tilde{b}_{11} = 0$ and $\tilde{b}_{12} = 1$. Note that the direct feedthrough from the input to any outputs downstream of biquad 1 is blocked. Also note that direct feedthrough of any states upstream of biquad 1 to any states downstream of biquad 1 is also blocked. Like the input, those states affect the downstream states through the output of biquad 1. However, it is clear that they still have a regular structure.

While lack of direct feedthrough may be unavoidable in continuous time, the real design choice comes when we wish to discretize continuous time models without direct feedthrough and represent them in an equivalent discrete time BSS or MNF.

## 9.28 Bilinear State-Space Form

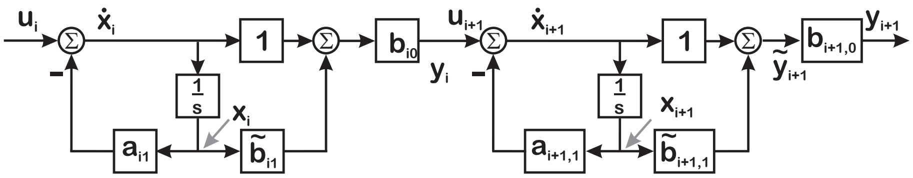


Figure 9.29: Continuous time bilinear state space (CT-BLSS) form.

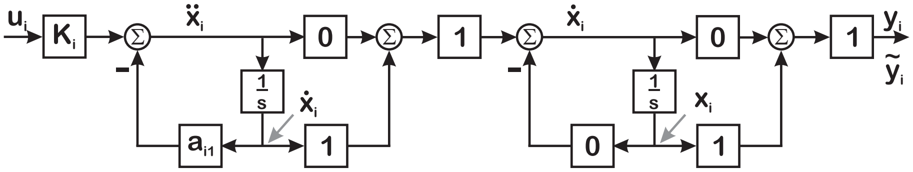


Figure 9.30: Continuous time rigid body BLSS model. Note that rather than indexing the second stage as $i + 1$, we stick with $i$ but label the level of integration on the signals.

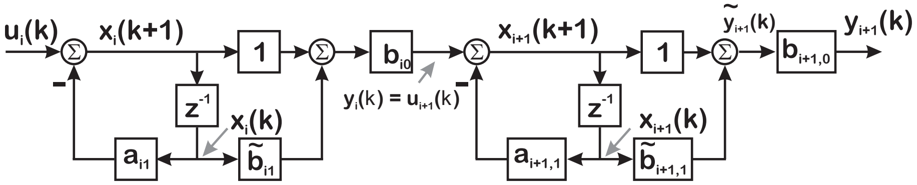


Figure 9.31: Discrete time bilinear state space form (DT-BLSS).

One of the issues with biquads is that while the continuous time biquads map to the discrete time biquads, and the input-output relationships hold, the internal states might not represent the physical states. In rigid body models, there is often an advantage to accessing the individual physical states, and in having these states map to discrete time models. To do this, we suggest the bilinear state space form (BLSS), which opens up the biquad in the case of real and distinct poles and zeros. The familiar CT and DT state equations are:

$$\dot{x} = F_C x + G_C u,\ y = H_C x + D_C u \text{ and} \tag{9.141}$$

$$x(k+1) = F x(k) + G u(k),\ y = H x(k) + D u(k), \tag{9.142}$$

respectively. The continuous and discrete matrices have the same structure, but the interpretation of the internal $\{a_{i1}, a_{i2}, \tilde{b}_{i0}, \tilde{b}_{i1}, \text{ and } \tilde{b}_{i2}\}$ coefficients change in going from continuous to discrete time. The DT matrices, $\{F, G, H, \text{ and} D\}$ are given by:

$$F = \begin{bmatrix} -a_{i+1,1} & b_{i,0}(\tilde{b}_{i1} - a_{i1}) \\ 0 & -a_{i,1} \end{bmatrix}, \tag{9.143}$$

$$H = \begin{bmatrix} b_{i+1,0}(\tilde{b}_{i+1,1} - a_{i+1,1}) & b_{i+1,0}b_{i,0}(\tilde{b}_{i,1} - a_{i,1}) \\ 0 & b_{i,0}(\tilde{b}_{i,1} - a_{i,1}) \end{bmatrix}, \tag{9.144}$$

$$G = \begin{bmatrix} b_{i,0} \\ 0 \end{bmatrix}, \text{ and } D = \begin{bmatrix} b_{i+1,0}b_{i,0} \\ b_{i,0} \end{bmatrix}, \text{ where} \tag{9.145}$$

$$u_i = y_{i-1} \text{ for } i = 1, \ldots, n. \tag{9.146}$$

The CT matrices, $\{F_C, G_C, H_C, \text{ and} D_C\}$ are given by:

$$F_C = \begin{bmatrix} -a_{i+1,1} & b_{i,0}(\tilde{b}_{i1} - a_{i1}) \\ 0 & -a_{i,1} \end{bmatrix} \tag{9.147}$$

$$H_C = \begin{bmatrix} b_{i+1,0}(\tilde{b}_{i+1,1} - a_{i+1,1}) & b_{i+1,0}b_{i,0}(\tilde{b}_{i,1} - a_{i,1}) \\ 0 & b_{i,0}(\tilde{b}_{i,1} - a_{i,1}) \end{bmatrix} \tag{9.148}$$

$$G_C = \begin{bmatrix} b_{i,0} \\ 0 \end{bmatrix}, \text{ and } D_C = \begin{bmatrix} b_{i+1,0}b_{i,0} \\ b_{i,0} \end{bmatrix}, \text{ where} \tag{9.149}$$

$$u_i = y_{i-1} \text{ for } i = 1, \ldots, n. \tag{9.150}$$

The general continuous time BLSS model is diagrammed in Figure 9.29 which simplifies to Figure 9.30 for the rigid body models we have discussed. The general discrete time model is diagrammed in Figure 9.31. We will see that these forms are extremely useful with adding a rigid body section to BSS models, since we can access the states such as velocity and position easily.

## 9.29 Discretization Choices

With the MNF, and the BSS, discretization was easily done via pole-zero mapping so long as the continuous time numerator and denominator were of the same order [309, 310, 303, 304]. The Bode plot comparisons in [304] gave confidence that this captured the zero behavior. When we are dealing with a pole-zero excess of 1 or 2 in a block – such as we have with CT low-pass filters and CT rigid-body models, we need to consider some choices for placing the CT "zeros at $\infty$" [133]. We have essentially 4 choices:

- Map one or two CT zeros at $s = -\infty$ to $z = -\infty$. While this takes sampling delay into account, it is the least favored of these methods for any filter that will be used in a feedback mechanism as the zeros at $-\infty$ will pull a corresponding number of closed-loop poles towards them and out of the unit circle.

- Map the CT zeros at $s = -\infty$ to $z = 0$. These tend to cancel pure delays and are conservative in that they minimize the phase effect of the denominator. This is what is done in typical PID discretization [311] and the conservatism of the zero at $z = 0$ helps stabilize the overall loop. However, they are not the most accurate match to the continuous time filter.
- Map the CT zeros at $s = -\infty$ to $z = -1$. This corresponds to the Trapezoidal Rule equivalent and is the most accurate match for the continuous time model.
- Do some combination of the above choices.

While assigning one of the CT zeros at $s = -\infty$ to $z = -\infty$ is a traditional way of incorporating delay [133] and results in no direct feedthrough. Assigning both DT zeros this way results in a discrete biquad that looks like

$$B_{i,FR}(z) = \frac{b_{i,2}z^{-2}}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}}, \tag{9.151}$$

which is similar to a forward rectangular rule equivalent, $\frac{1}{s} \longrightarrow \frac{Tz^{-1}}{1-z^{-1}}$. This will not have direct feedthrough and thus will require a discrete time block structure similar to the continuous time ones shown in Section 9.27. Recent work suggests designing for a system with minimum delay and then backing off bandwidth to accommodate the phase due to measured delay [112, 312].

If we consider the conservative backwards rule construction, $\frac{1}{s} \longrightarrow \frac{T}{1-z^{-1}}$, we end up with

$$B_{i,BR}(z) = \frac{b_{i,0}}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}}, \tag{9.152}$$

where we see that the numerator delay from (9.151) has been completely eliminated. The BSS block will have a standard structure, just with $\tilde{b}_{i,1}$ and $\tilde{b}_{i,2} = 0$. Finally, if we choose the Trapezoidal Rule equivalent, $\frac{1}{s} \longrightarrow \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}}$, we end up with

$$B_{i,TR}(z) = b_{i,0}\frac{1 + 2z^{-1} + z^{-2}}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}}, \tag{9.153}$$

with two zeros at $z = -1$. Finally, we might try tweaking the phase of the model by choosing one zero at $z = 0$ and one at $z = -1$, in which case we would have

$$B_{i,TBR}(z) = b_{i,0}\frac{1 + z^{-1}}{1 + a_{i,1}z^{-1} + a_{i,2}z^{-2}}. \tag{9.154}$$

Again, these are standard BSS blocks with direct feedthrough, but with particular values for $\tilde{b}_{i,1}$ and $\tilde{b}_{i,2}$. If we want a rigid body model with the BSS or a low pass filter that works well with the BSS or MNF, it is best to avoid zeros at $-\infty$.

## 9.30 Discrete Time Rigid Body Models

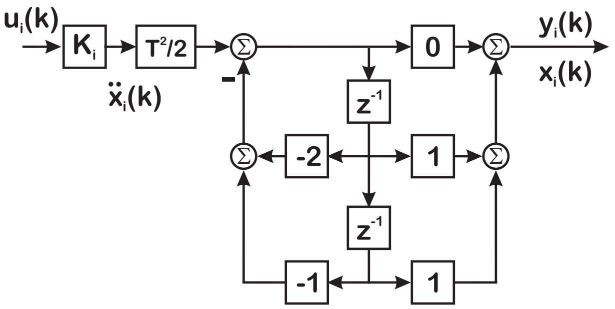


Figure 9.32: Discrete double integrator biquad model (ZOH equivalent). Notice when viewed this way, the integrators are unbalanced in that the first one is implemented differently from the second.

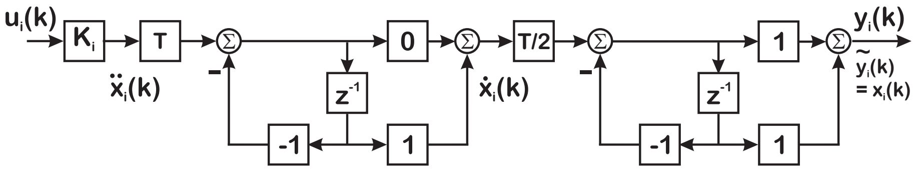


Figure 9.33: Discrete double integrator BLSS model (ZOH equivalent). In this drawing, we've chosen to make the index, i, as with a biquad stage, but we are explicitly labeling the different integration levels.

In Section 9.20 and in [303], we showed how using a Trapezoidal Rule equivalent on a double integrator preserved the feedthrough. With discrete equivalent forms of the model in Figure 9.26, we run into the issue that we cannot readily access a reasonable velocity estimate from these models. Looking at the zero-order hold (ZOH) equivalent [133] model in Figure 9.32 or the Trapezoidal Rule equivalent in Figure 9.34, we can easily extract an acceleration estimate and/or a position estimate, but velocity would require some new combination of the states. In practical use of state space models for motion control of mechatronic systems, it seems highly illogical to go to the trouble of generating a state space model and not be able to easily access velocity.

A word about indexing here. While normally we would want to index these blocks as we do with any of our biquad blocks, say block $i$, all the subscripts can make the text Byzantine at first glance. Instead, we use index $0$ for the first integrator, and $1$ for the second integrator, realizing that the readers will be able to add the appropriate offset to the equations. For clarity, the drawings index the stages all as $i$, but noting the integration level of each block.

The BLSS model of Figure 9.30 exposes velocity. Discretizing this model with a ZOH equivalent leads to the model of Figure 9.33, while a Trapezoidal Rule equivalent can be found in Figure 9.35.

We break up the ZOH equivalent as

$$\begin{aligned} D_{ZOH}(z) &= K\frac{T^2}{2}\frac{(z+1)}{(z-1)^2} \\ &= KT\left(\frac{1}{z-1}\right)\left(\frac{T}{2}\right)\left(\frac{z+1}{z-1}\right). \end{aligned} \tag{9.155}$$

The blocks end up with the equations of:

$$x_{0,k+1} = KTu_{0,k} + x_{0,k} \text{ and} \tag{9.156}$$

$$y_{0,k} = x_{0,k}, \tag{9.157}$$

where $u_0 = u$ and $y_1 = y$. With $u_{1,k} = y_{0,k}$ we have:

$$x_{1,k+1} = \frac{T}{2}u_{1,k} + x_{1,k} = x_{1,k} + \frac{T}{2}x_{0,k} \text{ and} \tag{9.158}$$

$$y_{1,k} \quad = \quad x_{1,k} + x_{1,k+1} = 2x_{1,k} + \tfrac{T}{2}x_{0,k}. \tag{9.159}$$

Together, these become:

$$\begin{bmatrix} x_{1,k+1} \\ x_{0,k+1} \end{bmatrix} = \begin{bmatrix} 1 & \frac{T}{2} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} 0 \\ KT \end{bmatrix} u_k. \tag{9.160}$$

The output is defined as:

$$\begin{bmatrix} y_{1,k} \\ y_{0,k} \end{bmatrix} = \begin{bmatrix} 2 & \frac{T}{2} \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u_k. \tag{9.161}$$

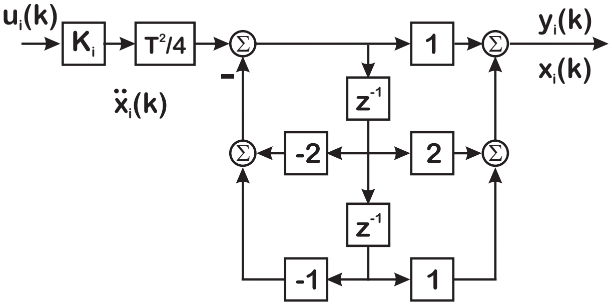


Figure 9.34: Discrete double integrator biquad model (trapezoidal rule equivalent).

This is the textbook model for a double integrator [133], but now we can access the velocity output, $y_{0,k}$ directly. This does not have direct feedthrough, unlike the trapezoidal rule model of (9.162). We break that up as follows:

$$\begin{aligned} D_{TR}(z) &= K\frac{T^2}{4}\frac{(z+1)^2}{(z-1)^2} \\ &= K\left(\frac{T}{2}\right)\left(\frac{z+1}{z-1}\right)\left(\frac{T}{2}\right)\left(\frac{z+1}{z-1}\right). \end{aligned} \tag{9.162}$$

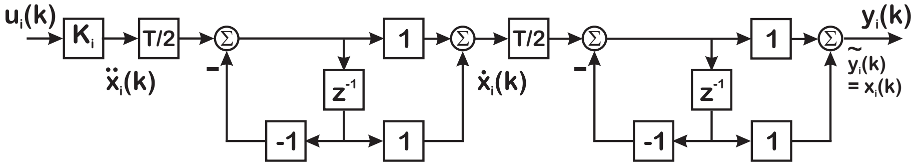


Figure 9.35: Discrete double integrator BLSS model (trapezoidal rule equivalent) . In this drawing, we've chosen to make the index, $i$, as with a biquad stage, but we are explicitly labeling the different integration levels.

The blocks end up with the equations of:

$$x_{0,k+1} = K\frac{T}{2}u_{0,k} + x_{0,k} \text{ and} \tag{9.163}$$

$$y_{0,k} = x_{0,k} + x_{0,k+1} = 2x_{0,k} + K\frac{T}{2}u_{0,k}, \tag{9.164}$$

where $u_0 = u$ and $y_1 = y$. With $u_{1,k} = y_{0,k}$ we have:

$$\begin{aligned} x_{1,k+1} &= \tfrac{T}{2}u_{1,k} + x_{1,k} \\ &= 2x_{1,k} + Tx_{0,k} + K\tfrac{T^2}{4}u_k \text{ and} \end{aligned} \tag{9.165}$$

$$\begin{aligned} y_{1,k} &= x_{1,k} + x_{1,k+1} = 2x_{1,k} + K\tfrac{T}{2}u_{1,k} \\ &= 2x_{1,k} + Tx_{0,k} + K\tfrac{T^2}{4}u_k. \end{aligned} \tag{9.166}$$

Together, these become:

$$\begin{bmatrix} x_{1,k+1} \\ x_{0,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} K\left(\frac{T}{2}\right)^2 \\ K\left(\frac{T}{2}\right) \end{bmatrix} u_k. \tag{9.167}$$

The output is defined as:

$$\begin{bmatrix} y_{1,k} \\ y_{0,k} \end{bmatrix} = \begin{bmatrix} 2 & T \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} K\left(\frac{T}{2}\right)^2 \\ K\left(\frac{T}{2}\right) \end{bmatrix} u_k. \tag{9.168}$$

With this implementation of the trapezoidal rule (TR) equivalent, we can also access the velocity output, $y_{0,k}$ directly. This has direct feedthrough, and is probably the closest simple equivalent to to continuous time form from a frequency response perspective.

Finally, our discussion of discretization might make us consider using a backwards rule equivalent to model the double integrator. This would be broken up as

$$\begin{aligned} D_{BR}(z) &= KT^2\tfrac{(z)^2}{(z-1)^2} \\ &= KT\left(\tfrac{z}{z-1}\right)T\left(\tfrac{z}{z-1}\right). \end{aligned} \tag{9.169}$$

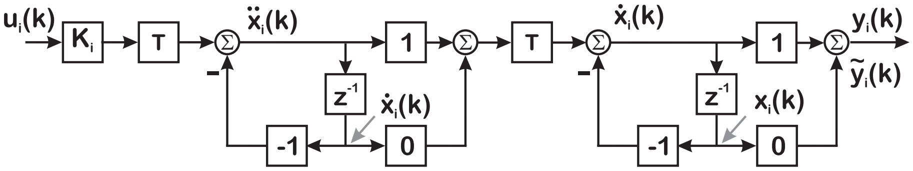


Figure 9.36: Discrete double integrator BLSS model (backwards rectangular rule equivalent ).

The blocks end up with the equations of:

$$x_{0,k+1} = KTu_{0,k} + x_{0,k} \text{ and} \tag{9.170}$$

$$y_{0,k} = x_{0,k+1} = x_{0,k} + KTu_{0,k}, \tag{9.171}$$

where $u_0 = u$ and $y_1 = y$. With $u_{1,k} = y_{0,k}$ we have:

$$\begin{aligned} x_{1,k+1} &= Tu_{1,k} + x_{1,k} \\ &= x_{1,k} + Tx_{0,k} + KT^2u_k \text{ and} \end{aligned} \tag{9.172}$$

$$y_{1,k} = x_{1,k+1} = x_{1,k} + Tx_{0,k} + KTu_{1,k}. \tag{9.173}$$

Together, these become:

$$\begin{bmatrix} x_{1,k+1} \\ x_{0,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} KT^2 \\ KT \end{bmatrix} u_k. \tag{9.174}$$

The output is defined as:

$$\begin{bmatrix} y_{1,k} \\ y_{0,k} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{0,k} \end{bmatrix} + \begin{bmatrix} KT^2 \\ KT \end{bmatrix} u_k. \tag{9.175}$$

While this is not as true to the continuous time model as the trapezoidal rule, it does have the nice property of minimizing latency with the two zeros at $z = 0$. This property makes the model more robust to time delay, much in the same way that the backwards rule equivalent discretization used in most digital PID controllers makes the controller more robust.

Note the key difference internally is that we have scaled the integration of the intermediate state structure to more closely match the continuous time form. This kind of scaling was scrupulously avoided for high Q filters in [309, 304], but should present no problem with the rigid body modes.

## 9.31 Rigid Body to BSS Examples

Figure 9.37 demonstrates the Bode plot of a BSS model with a low pass filter (LPF) as part of a 3-biquad model. Note the close match between the composite responses (individual biquad frequency responses

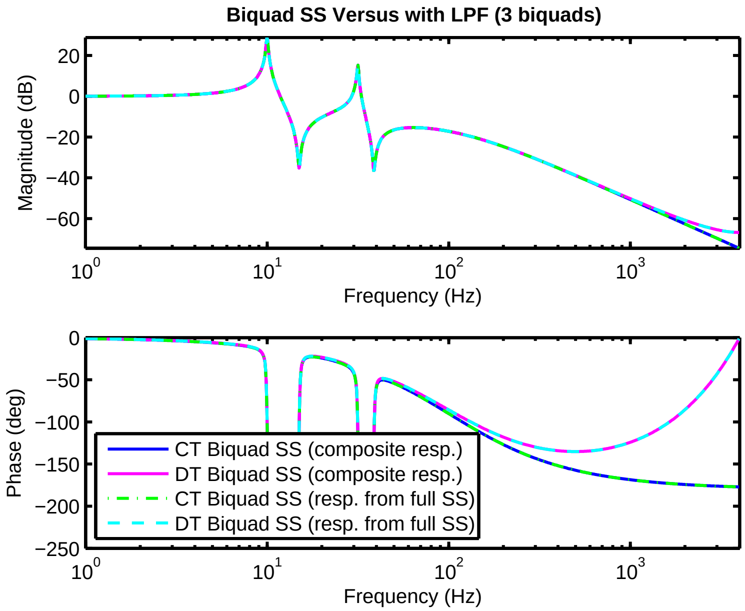


Figure 9.37: Biquad State Space with three biquads including a low pass filter. The CT biquad plots include a composite of the individual CT biquad Bode plots (blue) and a Bode of the complete CT BSS structure (green). The DT biquad plots also include a composite of individual DT biquad Bode plots (magenta) and a Bode of the complete DT BSS structure (cyan). The match of the complete structures to the composites show that the CT and DT BSS structures have properly represented the series connection of the individual biquads. The match between CT and DT show that we have properly discretized the low pass filter in our BSS. The curl up of the phase back to 0 in the DT curves is based on the zeros at $z = -1$ due to the use of a Trapezoidal rule equivalent discretization.

combined) versus responses extracted directly from the full BSS structure. The significance of this is that while the discrete time LPF can be modeled with direct feedthrough, the continuous time LPF cannot. Nevertheless, they produce responses that match very well. Figure 9.38 shows that the CT and DT match is across all the biquad outputs, as previously demonstrated without the LPF in [304].

Figure 9.39 shows at double integrator, discretized using a ZOH equivalent, implemented as a biquad (top) and a BLSS (bottom). The input-output behavior is consistent, but we now have access to the internal intermediate state with the BLSS. Figure 9.40, repeats the simulation using a Trapezoidal rule equivalent. In both cases, the BLSS gives us access to the output of the first stage, which is can be interpreted as velocity. We see further, that using the Trapezoidal Rule equivalent adjusts the scale of the internal state of the first bilinear block to make the integrators balanced. The BLSS block is a logical addition to state space models needing access to both position and velocity.

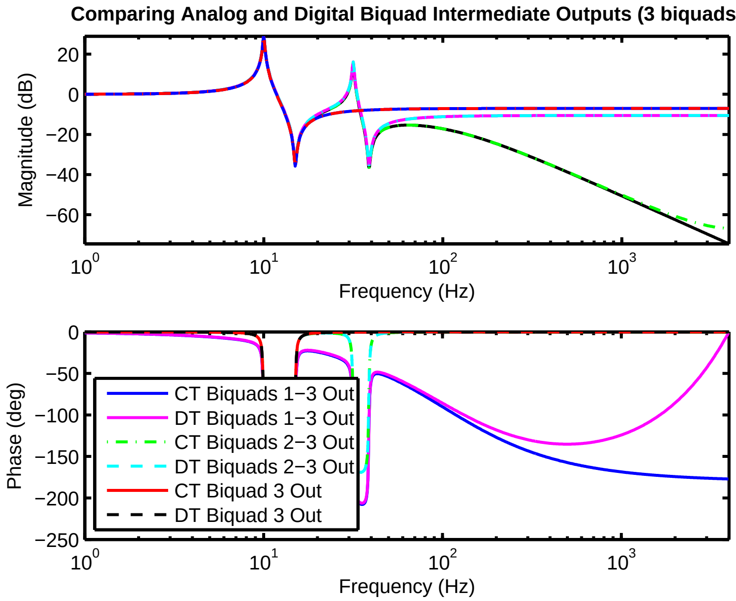


Figure 9.38: BSS with three biquads including a low pass filter in Biquad 1. This plot compares the Bode responses of the individual CT and DT biquad sections. The outputs of biquad 3 and biquad 2 show the magnitude and phase flattening out at high frequency (due to the matched number of poles and zeros). Once the response of biquad 1 is added in, we see the low pass rolloff of Figure 9.37. At each biquad output, the match between continuous and discrete responses is incredibly close, a unique and useful feature of this structure.

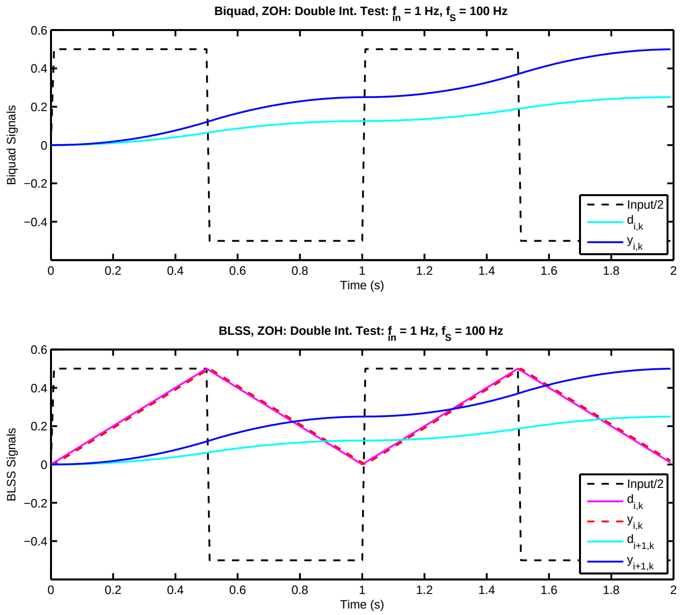


Figure 9.39: Double integrator with square wave input. Implemented using a ZOH equivalent biquad (top) and BLSS (bottom).

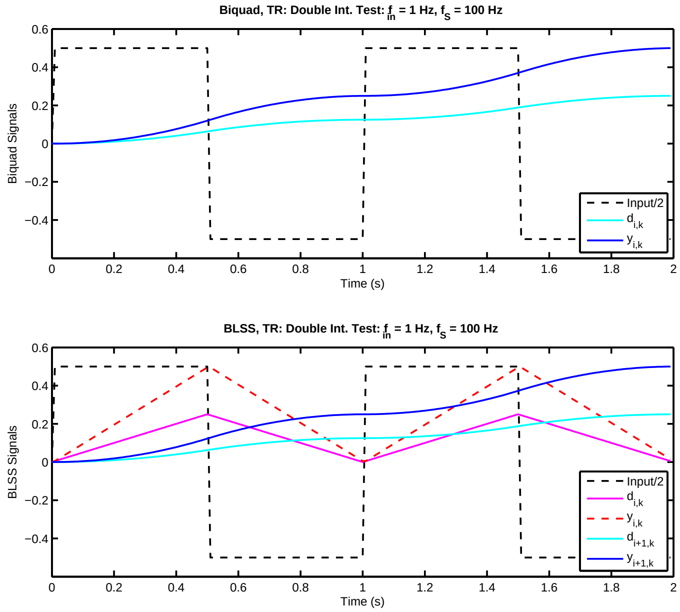


Figure 9.40: Double integrator with square wave input. Implemented using a trapezoidal rule equivalent biquad (top) and BLSS (bottom).

## 9.32 Continuous Time Examples

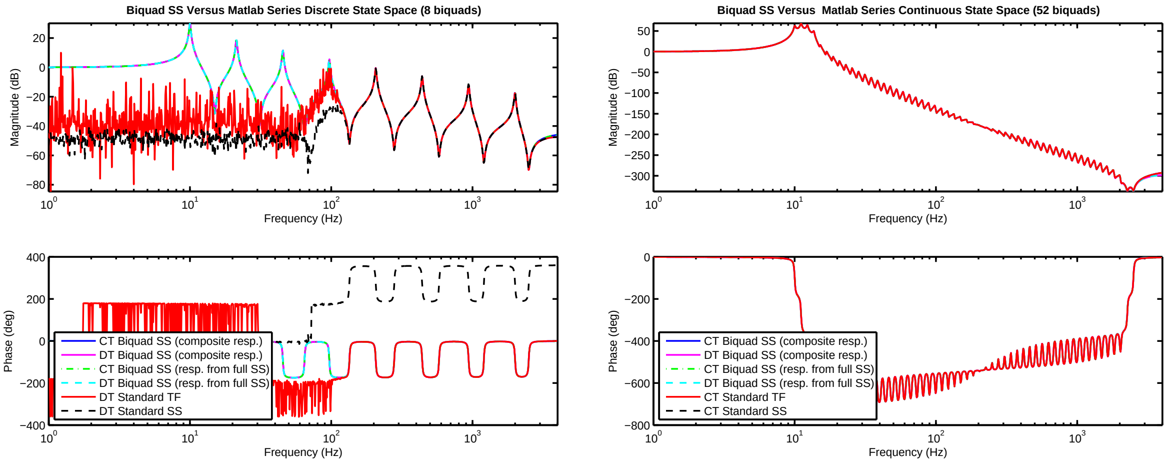


Figure 9.41: Comparison of Bode plots from continuous and discrete BSS, as well as standard discrete transfer function and state space forms. The conventional methods fall apart with 8 biquads, while the BSS methods retain their numerical integrity.

In order to demonstrate the numerical improvements arising from the biquad state space structure, some simple examples were generated in Matlab. A serial biquad structure was generated from resonance and anti-resonance parameters. The natural frequencies of the resonances were logarithmically spaced between 10 Hz and 2000 Hz, while the natural frequencies of the anti-resonances were spaced between 15 Hz and 2500 Hz. Numerator and denominator damping factors were set at 0.01. The sample frequency was chosen at 8 kHz for the discrete biquads. The script could then specify any number of resonance/anti-resonance pairs to fill that frequency range. As the baseline, the frequency responses of each individual biquad was generated and these responses were summed to create a composite response. Since the responses were generated from individual biquads, it was thought that they would be less susceptible to numerical issues.

In order to compare the biquad state space to more conventional methods, the resonance/anti-resonance parameters were then used to generate both transfer function models and state space models in MATLAB . The linear system concatenation functions were used for both of these. From these high order models, Bode plots were generated to compare to the composite Bode plots described above. These are the “standard” or “conventional” methods. Similarly, model terms were used to construct both continuous and discrete biquad state space structures and again, Bode plots were generated. Note that these plots are not made using fixed point math, but with all terms represented in MATLAB 's dual precision floating point format.

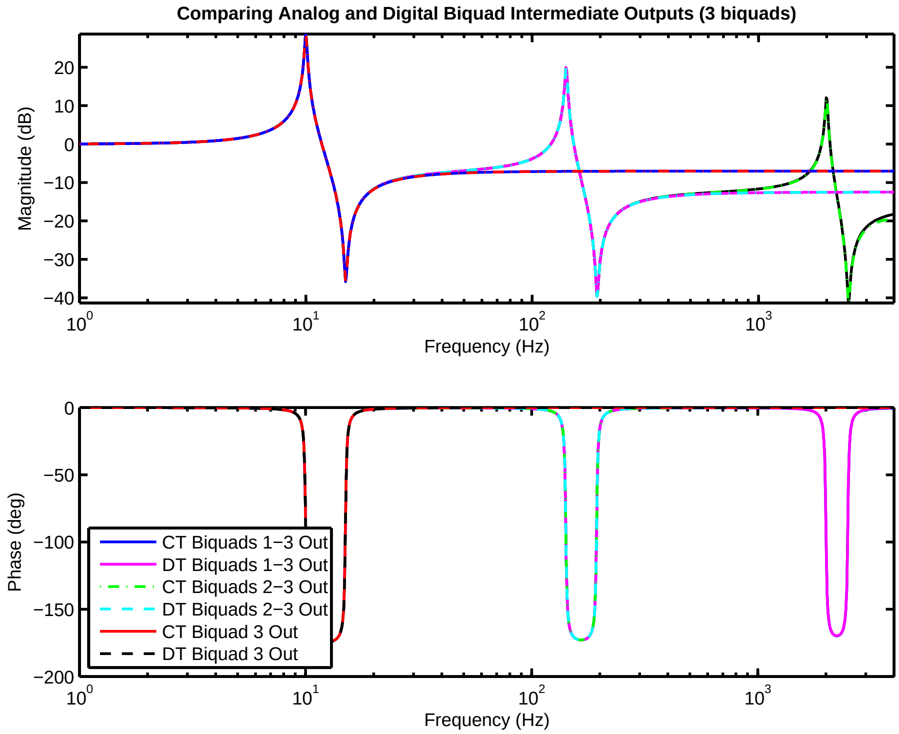


Figure 9.42: Comparing analog and discrete BSS outputs. With 3 biquads, the outputs of each analog biquad section was plotted against it's digital version. This demonstrates the invariance of the biquad outputs under discretization.

The left side of Figure 9.41 shows an 8 biquad case where the conventional discrete time structures (transfer function and state space) have numerical difficulties. Note that both continuous and discrete BSS structures produce plots right on top of the composite plots.

On the right side of Figure 9.41 the composite plots and continuous and discrete BSS structure plots are compared to conventional continuous methods. In this case all methods produce identical plots, even with a 52 biquad structure.

Figure 9.42 plots a 3-biquad structure, and in this case we plot neither conventional methods nor the composite plots. Instead we tap off the individual biquad outputs of the first, second, and third biquads so that we can demonstrate the almost exact match of the discrete biquads to the continuous biquads.

As mentioned earlier, this "invariance under discretization" is a very useful property. In particular, it allows one to construct an analog model from physical principles, convert this model to an analog Biquad State Space form, convert that to a discrete time Biquad State Space form for implementation, and then easily extract information about the continuous model from the discrete model results.

## 9.33 Biquad State Space Summary

One of the more peculiar aspects of this development is that the solution to adding physical intuition back into state-space methods started in the discrete-time formulation, and then bled into continuous time. It was only by realizing that we needed to give up the idea of combining all the model parameters into some sort of polynomial form realization (e.g. controller canonical form or observer canonical form) and realizing that we could discretized the analog structures one biquad at a time, did we finally get to a state-space construction where the states of the continuous time model and the discrete time model were tightly connected.

The biquad state space (BSS) form adapts The Multinotch filter [54, 33] for state space use, preserving the latter's excellent numerical properties [3]. One form of the BSS has the same minimum latency behavior that makes the Multinotch so useful for real-time control. Like the Multinotch, this "scalar output gain" version of the BSS has computational latency after the input sample that is independent of the number of states. When used as an observer for state feedback in a SISO system, the extra latency is due to $n_{states}$ multiplications (which can be done in parallel) and a sum of these products. Depending upon the computational architecture and the number of products, the addition can also be accomplished in $1$ to $n_{states}-1$ operations, the latter if addends must be summed $2$ at a time.

Even when doing off line modeling, the examples in Sections 9.21 and 9.32 demonstrate how the BSS preserves numerical fidelity in the state space model. It also preserves the physical intuition of the analog parameters in the digital state space matrices, which is extremely helpful in debugging physical systems. Moreover, both the continuous BSS [4] and the discrete BSS [3] have a regular and repeatable structure, as with a canonical form. This makes it relatively straightforward to generate the matrices from modal parameters in an automated fashion, and transfer those parameters into an implementable discrete-time form.

Perhaps most important is the fact that the discrete-time BSS matrices are closely related to the continuous-time BSS matrices, allowing for an extremely "physical" understanding of complex, discrete-time models. One can envision using state space in industrial environments by:

a) Generate a basic physical system model from first principles, physics, experience, and previously measured dynamics.

b) Make measurements (Chapter 3) to verify/correct these models to tighten parameters, etc. for an initial feedback loop.

c) (If needed) Repeat measurements under feedback for more detailed continuous time model.

d) Transform that model into a modal, continuous-time biquad state space (CT-BSS) form.

e) Discretize the continuous-time biquad state space (CT-BSS) to a discrete-time biquad state space (DT-BSS), preserving the essential structure.

f) Verify model by applying similar "measurements" to Discrete-Time Biquad State Space and Continuous-Time Biquad State Space forms, as in Figure 9.42.

g) Use the discrete-time biquad state space for design/control.

The steps may seem pedantic, but they each verify te viability and physicality of the Discrete-Time Biquad State Space model. Furthermore, the test points o the Discrete-Time Biquad State Space should correspond to the test points on the Continuous-Time Biquad State Space, which should be very relatable to those on the physical system.

What this means – and this is a big deal – is that we can debug our system. We can compare physical and digital model test points to match the signals. This kind of physical "debugability" of at discrete-time state space model is almost unheard of outside of second order systems, but this method gives us that possibility.

## 9.34 Chapter Summary and Context

## 9.35 Change Log for Chapter 9

- 2019_06_29: Added Section 9.3 as a simple introduction of state space concepts and what is important.
- 2019_06_29: Added Section 9.7 as a model based measurement tutorial.

- 2019_06_30: Added discussion of observability and controllability, and its practical meaning, to Section 9.7.
- 2019_07_02: Added sections on BLSS and other useful structures for BSS from [5]. Tried to harmonize them with the rest of the chapter.
- 2019_07_03: Cleaned up text. Added more structure to the conclusions. Added a quasi-chapter TOC nearrow the beginning of the chapter.

# Chapter 10

# Real-Time Computing Issues for Control Systems

## 10.1 In This Chapter

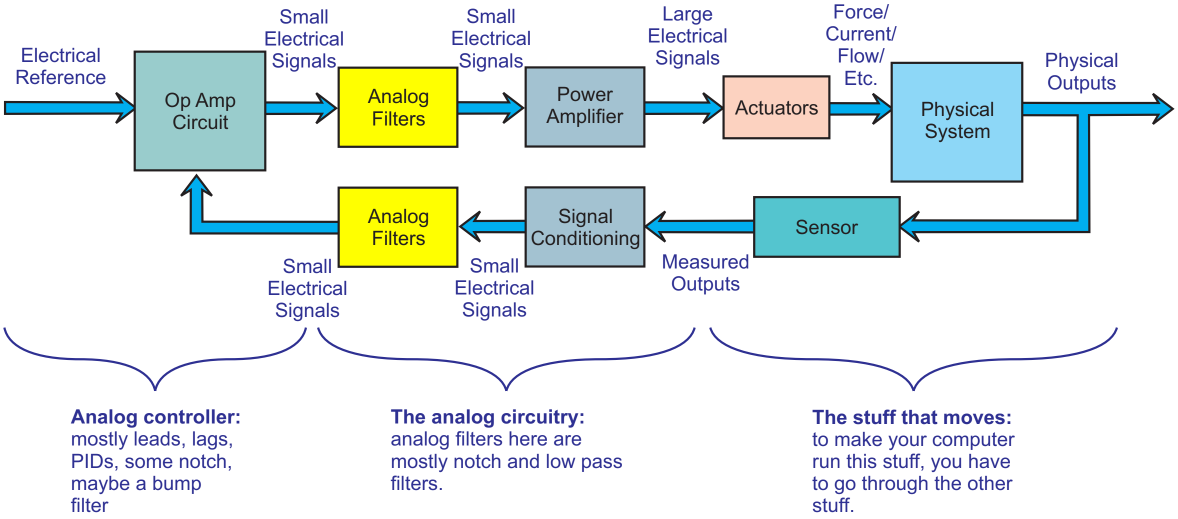
In this chapter we will focus on the pieces of computation that need to be in place in any real-time control system. While we are focusing on control systems where the main computation is done by a digital computer (broadly any computation done via programmable digital logic, including processors and FPGAs ), there is always a role for analog electronics in interfacing these means to the real world. This chapter will have a handful of major themes:

- The physical system to computation “input” signal chain and its computations.
- The computation to physical system “output” signal chain and its computations.
- The physical system “computations” and discussions of various model types.
- The computation itself: how to think about computer architectures and programming in the context of real-time control systems.

In this chapter, we will often refer to Moore’s law, the general rule proposed by Intel Co-Founder Gordon Moore [42] that the amount of logic one can pack onto a Silicon chip doubles roughly every 18 months.

This prediction has been remarkably accurate over the decades, not only through the march of technology but also that chip makers themselves have felt pressured to make sure they meet the line (in logarithmic space). In this chapter, Moore's law will be used as a generic stand in for the rapid advance of computer and electronic technology. Similarly, Newton's laws, which usually refer to Newton's laws of motion [314], but in this book the term is shorthand for scientific modeling and inference, i.e. what drives the real world system. Substantial portions of this chapter have been published by the author in [315] and [17].

## 10.2 Motivation: Why Talk About Computation?



Electrical Reference
Op Amp Circuit
Small Electrical Signals
Analog Filters
Small Electrical Signals
Power Amplifier
Large Electrical Signals
Actuators
Force/ Current/ Flow/ Etc.
Physical System
Physical Outputs
Sensor
Measured Outputs
Signal Conditioning
Small Electrical Signals
Analog Filters
Small Electrical Signals
Analog controller: mostly leads, lags, PIDs, some notch, maybe a bump filter
The analog circuitry: analog filters here are mostly notch and low pass filters.
The stuff that moves: to make your computer run this stuff, you have to go through the other stuff.

Figure 10.1: A generic analog control loop.

Let's be pedantic for a moment. The main difference between an analog control loop in Figure 10.1 and the digital loop in Figure 10.2 is the stuff on the left side, namely the computation and the conversion chips. Within reason, one can consider the stuff to the right of the converters to be the same, although it is true that there are some control loops that nobody would ever attempt without digital controllers.

Still, the necessity of these devices for digital control implies that one should at least know something about them. One might not need to fully understand the internal combustion engine to be a car owner, but it helps to have at least a thumbnail idea of what is going on when you need to take that car to the shop.

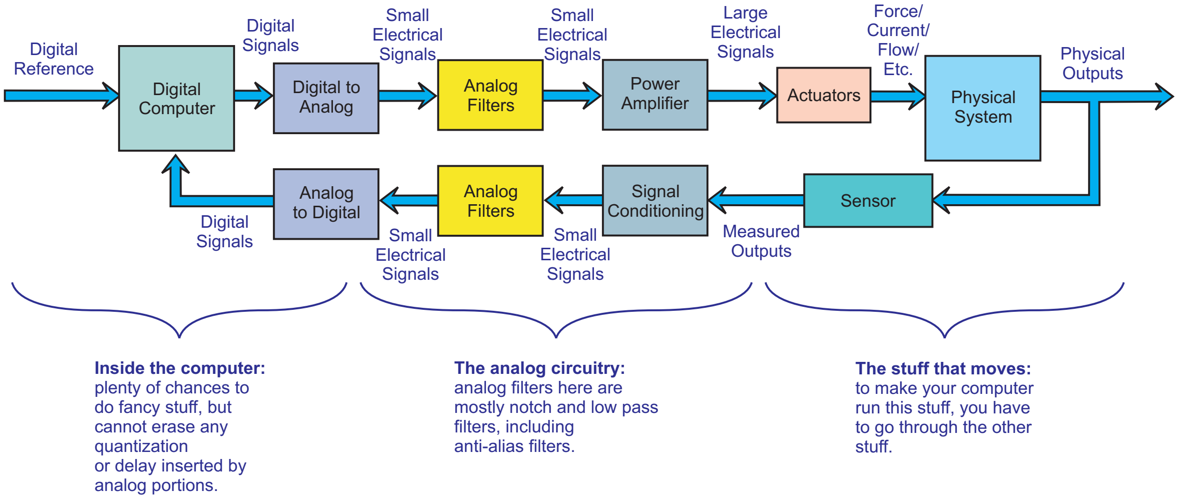

Digital Reference
Digital Computer
Digital Signals
Digital to Analog
Small Electrical Signals
Analog Filters
Small Electrical Signals
Power Amplifier
Large Electrical Signals
Actuators
Force/ Current/ Flow/ Etc.
Physical System
Physical Outputs
Sensor
Measured Outputs
Signal Conditioning
Small Electrical Signals
Analog Filters
Small Electrical Signals
Analog to Digital
Digital Signals
Inside the computer: plenty of chances to do fancy stuff, but cannot erase any quantization or delay inserted by analog portions.
The analog circuitry: analog filters here are mostly notch and low pass filters, including anti-alias filters.
The stuff that moves: to make your computer run this stuff, you have to go through the other stuff.

Figure 10.2: A generic digital control loop.

The choices of computational models, sample rates, converters, and analog filters usually affect the results in the real world far more than any 5% difference in algorithm performance. This is, in fact one of the difficulties of putting advanced methods into physical implementation. Applying 10× as much mathematical machinery may or may not get as much improvement as choosing an ADC with a pipeline delay of 1 sample rather than 17 samples.

The point is that these choices often determine what your algorithms can and can't do. The second point – and this may be more important – is that somebody will make those choices. Channeling our inner Georges Clemenceau [316], the choice of hardware and computing platform for our control systems is too important to be left to people who are not control engineers. These system level choices are at the heart of what it means to translate the "beautiful math" of control theory into the "bad ass engineering" of control systems.

It is important to realize that often the assumptions about what can and can't be done in a control loop in any institution are based on examples of what has worked before. Many of these date from a time when processing was far more expensive. The relentless path of Moore's law means that we need to challenge the computational assumptions at least every 5 years.

While computation is critical to any digital control system, our CAD tools have gotten so effective that fewer and fewer control engineers are competent programmers outside of Matlab or Python. This leaves us to rely on the prepackaged real-time implementation tools provided by manufacturers. While these tools allow one to go from model to real-time without writing C or C++ code, their need for generality and for hiding the computational complexity from the user often consume much of the real-time resources of a chip. The control engineer who hits that limit without understanding computation is left relatively helpless. A basic understanding of how computation-for-feedback issues affect the performance of real-time systems will greatly expand the performance achievable by many algorithms.

## 10.3 Why is Discussing Computation for Feedback Systems Hard?

We should ask at this point what makes general discussions and guidelines of computation for real-time control so rare? The texts that discuss them are either focused on one technology level for one particular class of operations, e.g. running rigid body mechatronic systems from a standard laptop, or they are at such a high level as to have only vague recommendations about processor speed with respect to time constants.

1) There are few closed-form answers. That is, there is little coherent theory which allows us to plug in numbers and get definitive yes/no answers. Instead, we are left to piece together information from data sheets, prior experience, and best practices. Furthermore, everything seems very technology and application specific. This makes it hard to have a general discussion.

2) The technology is constantly changing (again Moore's law), which means anything chip or technology specific will be out of dat in one to ten years. Mass market devices are made obsolete on a 1–2 year schedule. Connection chips, embedded processors, etc. have a longer lifespan, typically in the 4–10 year range. Still, this is uncomfortable for those of use used to theories that largely stand the test of time.

3) There are many levels, many time constants, and many technologies. Consequently, it is again hard to make definitive statements.

We take the view that while technology changes constantly, the principles on how we choose that technology does not. The feedback framework of Figure 10.5 provides certain essential limitations and requirements that provide critical insights on how we build our real-time systems.

**"The more you want to do this stuff in real life, the more you need your opponent's reaction to help you." – Rickson Gracie, at a seminar in Oakland, CA in 1994 discussing Brazilian Jiu Jitsu (BJJ) and Mixed Martial Arts (MMA).** Similarly, the more you want to design control systems for the real world, the more you need insights from the physical system to help you. With that in mind:

- Put in extra sensors, even if they are not providing data for the main feedback loop. They can often provide important meta data for the conditions affecting the main feedback loop.
- Whenever possible, compute low computation cost reality checks in parallel to the main computation. These are rarely sophisticated enough to run the main control loop(s), but they can often provide insight as to when the main loop is malfunctioning.
- Make the computer models of the system as physical as possible. This makes it far easier to compare the signals in the model with the signals one can measure from the physical system.

**"The pitch is a strike until the ball tells you it's not." – Old baseball coach saying.** One of the characteristics that seems to align with youth (it certainly aligned with mine) is to assume that systems using old technology were so bad as to not consider. Perhaps this is why enthusiasts of classic cars tend to be older. However, the classic car enthusiasts may have a point. They are assuming that these are well engineered machines for the technology of their day; that the designers made rational choices and solved problems well, given their technology constraints. In a similar way, we should be willing to examine and understand successful real-world design examples, even old ones. Assume the engineers made intelligent choices, given their technological environment until you understand it well enough to say why they did not. (The pitch is a strike until the ball tells you it's not.) With that understanding, we can intelligently assess what our current technology environment changes about the problem and how much stays the same. This allows us to be more intelligent about preserving the (possibly considerable) problem solving effort made by those with more primitive tools, while allowing us to improve on those design with the gifts of Moore's law.

**Match the algorithm to the data and the physical system, not to one's preferred programming environment.** As much as possible, algorithms should be processor and environment agnostic, while being sensitive to the particular needs and restrictions of each computational layer in the Three-Layer Model of Section 10.12.1.

**Know when it matters.** Many, many problems can be solved without much consideration to many of the issues here. Their dynamics are so benign and slow compared to our computation, their physical parameters are not needed for adequate control, and it is more important to enable some feedback, than to get hung up on analysis. Some of the discussions about time constants, time delay, jitter, and computational architecture should give the reader a lot more guidance on how to know when they are in one of those situations and they can follow Nike's maxim to "Just do it."

**Solutions to real-world problems are far more iterative than they are sequential.** That is, there is a lot to be gained from iterating on prototype efforts rather than assuming that the problem can be solved in a purely sequential way. It is the feedback from those prototypes that reveals the key adjustments to the design.

**Specific technologies and chips change. Principles for computation remain.** The latter are what we discuss in the rest of this chapter.

## 10.4 A High-Level View of Computing for Feedback Systems

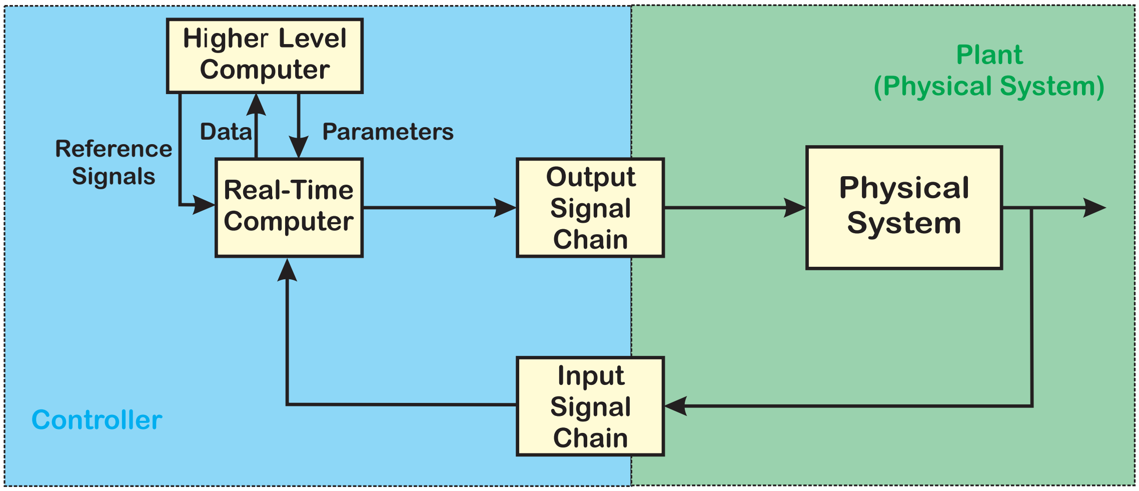


Figure 10.3: An abstracted view of the main computational divisions in a feedback system.

In this chapter we will focus on the pieces of computation that need to be in place in any real-time control system. While we are focusing on control systems where the main computation is done by a digital computer (broadly any computation done via programmable digital logic, including processors and field programmable gate arrays (FPGAs) ), there is always a role for analog electronics in interfacing these means to the real world. Figure 10.3 shows the main "computations" done in a feedback loop as abstracted blocks. We can think of four chains where processing happens:

- The physical system to computation "input" signal chain and its computations.

- The computation to physical system "output" signal chain and its computations.
- The physical system "computations" and discussions of various model types.
- The computation itself: how to think about computer architectures and programming in the context of real-time control systems.

Each has its own potential latency, jitter, and noises. We typically only think about those of the plant. Bode's Integral Theorem [105, 1, 2] teaches us that once noise gets into a loop we can only adjust where we amplify it. Causality means that once latency enters a loop, we cannot eliminate it. Jitter just makes all of this worse. Finally, it's all relative to the physical system time constants.

In this chapter, we will often refer to Moore's law [42], the general rule proposed by Intel Co-Founder Gordon Moore that the amount of logic one can pack onto a Silicon chip doubles roughly every 18 months. This prediction has been remarkably accurate over the decades, not only through the march of technology but also that chip makers themselves have felt pressured to make sure they meet the line (in logarithmic space) [42]. In this chapter, Moore's law will be used as a generic stand-in for the rapid advance of computer and electronic technology. Similarly, Newton's laws, usually refer to Newton's laws of motion [314], but in this chapter the term is shorthand for scientific modeling and inference, i.e. what drives the real world system.

Each of the blocks in Figure 10.3 affects the performance of the feedback loop. Moore's Law has made the left side of Figure 10.3 much more powerful. Modulo being able to attack far more physical problems than before (again Moore's Law), the right side is governed by Newton's Laws. The following should be considered fundamental to understanding computation for feedback control systems.

> Newton's Laws take precedence over Moore's Law, and they always will. However, Moore's Law helps us read the fine print of Newton's Laws, helps us get computation close to the real physics, close to the real model, but only if we are ready to do the work.

Finally, it is worth returning to the discussions of the mental framework for filtering, diagrammed in Figure 10.4, with that of feedback, diagrammed in Figure 10.5. This was first presented, and in more depth, in Section 2.4 of the Introduction. To summarize what was there, the key differentiator of the filtering framework is that we can never assume any access to the input driving the physical system. Consequently:

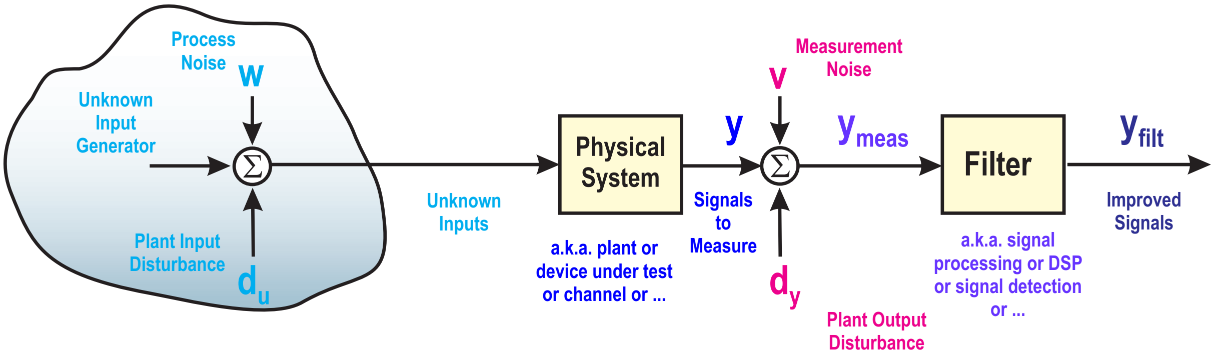


Figure 10.4: A filtering structure for looking at processes. This is a repeat of Figure 2.16.

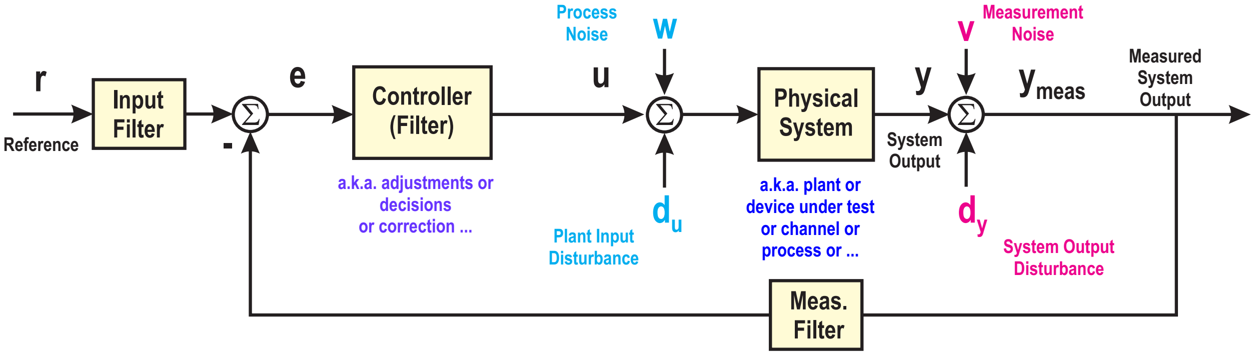


Figure 10.5: A feedback structure for physical processes. This is a repeat of Figure 10.5.

⇒ Noise and disturbances are modeled solely at the output.

⇒ This fundamentally limits input-output modeling.

⇒ Nothing we do in our filtering will affect the process.

⇒ Because of this, we have to assume that the physical process has to be reasonably behaved on its own.

⇒ The filtering context is insensitive to latency.

Conversely, a feedback framework only exists if we assume that we have access to some inputs to drive the physical system.

⇒ Noise and disturbances are modeled at both input and output of the physical system.

⇒ Our models use both plant outputs and inputs.

⇒ We adjust our measurements to better drive our inputs.

⇒ Our inputs can and most likely will change the behavior of the physical system.

⇒ We are sensitive to latency.

This matters because much of what has been written about real-time computation is from a filtering framework. Access to inputs to the system drives sensitivity to latency, and this changes the entire perspective.

## 10.5 Time Delay and Sampling

Time delay (latency) in a feedback loop is one of the key limiting factors of closed-loop performance [19]. Latency in time is negative phase in frequency, and without phase margin, feedback control is untenable. In a feedback loop, we can think of four general sources of time delay:

a) physical properties of the system,

b) sensor/actuator effects,

c) conversion delays, and

d) computational and sample rate delays.

We will focus on computation and conversion delays elsewhere, since these are things that we may affect by better real-time system design (hardware and software). Here we group sensor, actuator, and plant delays together as delays that we cannot alter merely with electronics. We will discuss the consequences of this delay. We especially want to make obvious what fast sampling can and cannot do to handle this delay.

A delay of $\Delta$ seconds is typically modeled in the s-plane as:

$$D(s) = e^{-s\Delta}. \tag{10.1}$$

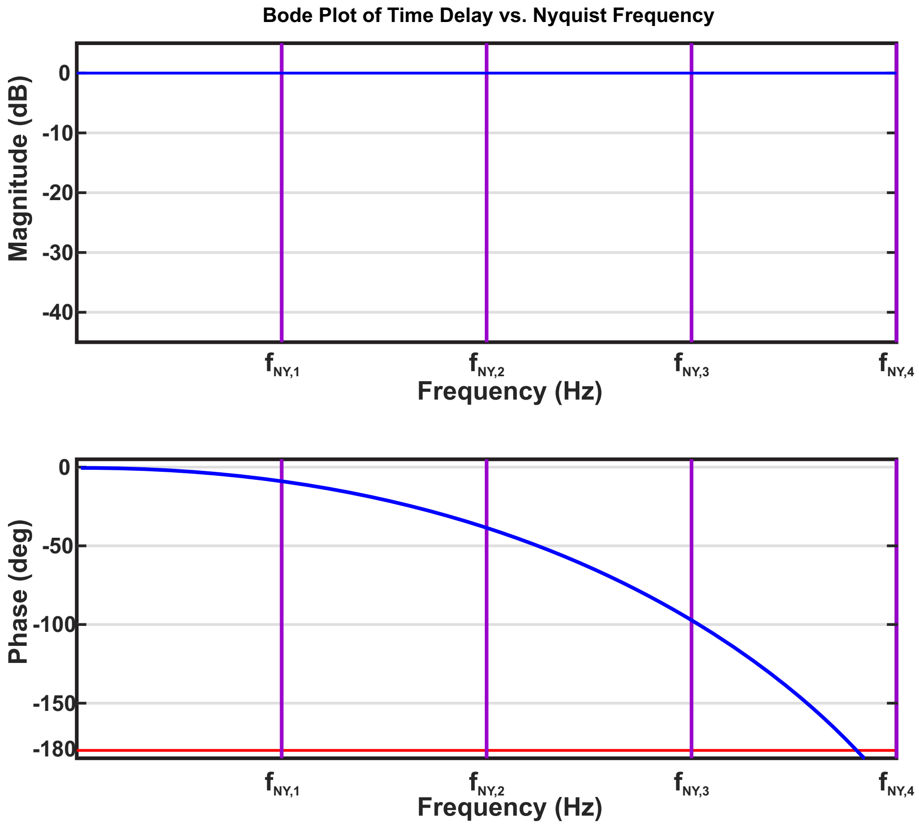


Figure 10.6: Bode plot of physical time delay versus sampling rate. (Repeat of Figure 5.13.)

We can evaluate Equation 10.1 at $s = j\omega$ to generate a Bode plot as shown in Figure 10.6, but for our analysis we usually like to have a rational transfer function. It is common to use a Padé approximant [114] for this, and while many variations are possible, a first order numerator and denominator effectively illustrate the issues we must contend with. For a first order Padé approximation of (10.1), we get

$$e^{-s\Delta} \approx \frac{1 - \frac{s\Delta}{2}}{1 + \frac{s\Delta}{2}} = \frac{\frac{2}{\Delta} - s}{\frac{2}{\Delta} + s} \tag{10.2}$$

Now, we see that this simple and reasonable approximation of delay has given us a stable pole and non-minimum phase (NMP) zero, as diagrammed in Figure 10.7. Were we to use a higher order Padé approximant, we would have more NMP zeros.

It is common for control engineers to say that we need simply sample faster to deal with the time delay, but fast sampling does not make physical delay disappear. This is illustrated in Figure 10.6, which shows a Bode plot of physical delay versus different Nyquist frequencies. The faster sampling has just pushed

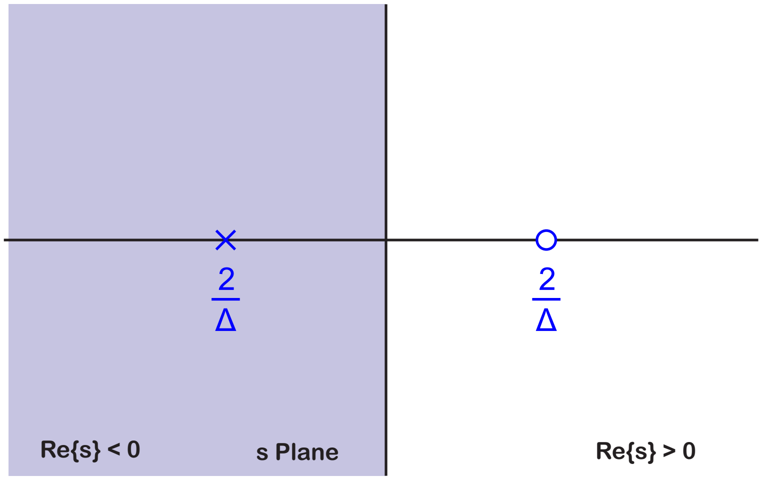


Figure 10.7: First order Padé approximation of time delay on the s-plane. Note the non-minimum phase (NMP) zero.

the Nyquist frequencies to the right, up to areas of higher phase lag due to delay. That does not solve anything, although it gives us more room in the frequency space to apply phase-lead to compensate for some of this delay.

We can return to the Padé approximation and discretize it to look at this in the z-plane. However, we will first break up $\Delta$ into full and partial sample periods, i.e.

$$e^{-s\Delta} = e^{-s(MT_S+\delta)} \tag{10.3}$$

where $\Delta = MT_S + \delta$ and $0 \leq \delta < T_S$. We will use $z^{-1}$ for full sample period delays and the Padé approximant for the partial delay, $\delta$.

It turns out that using a trapezoidal rule on (10.2) (replacing $\Delta$ with $\delta$) is satisfying in that for $\delta = 0, D_\delta(z) = 1$ and for $\delta = T_S, D_\delta(z) = z^{-1}$. In between these values we get:

$$e^{-s\delta} \approx \frac{\frac{2}{\delta} - s}{\frac{2}{\delta} + s} \xrightarrow{TR} \frac{(T_S - \delta)z + T_S + \delta}{(T_S - \delta)z + T_S + \delta} = D_\delta(z) \tag{10.4}$$

for $0 \leq \delta < T_S$. The poles and zeros get exposed by reordering this as:

$$D_\delta(z) = \left(\frac{T_S - \delta}{T_S + \delta}\right)\left(\frac{z + \frac{T_S+\delta}{T_S-\delta}}{z + \frac{T_S-\delta}{T_S+\delta}}\right). \tag{10.5}$$

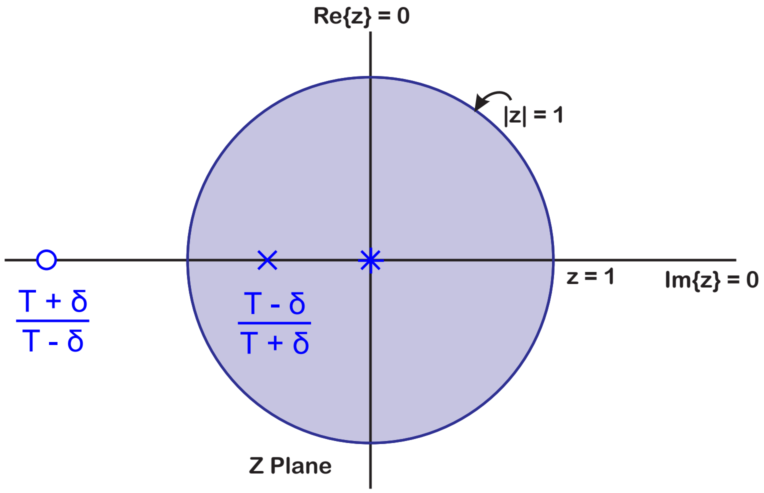

Re{z} = 0
|z| = 1
$\frac{T + \delta}{T - \delta}$
$\frac{T - \delta}{T + \delta}$
z = 1
Im{z} = 0
Z Plane

Figure 10.8: Padé approximation of time delay on the z-plane. The full sample delays result in a pole at $z = 0$. The partial sample delay is handled by a first order Padé Approximation. If we have $M$ unmatched poles at $z = 0$, then we will have $M$ zeros at $|z| = \infty$.

The result of this discretization is diagrammed in Figure 10.8. We see that we have mapped the sub-sample portion of our delay to a stable pole and NMP zero. However, we have $M$ poles at $z = 0$. These are so benign in DSP environments (the filtering context of Figure 10.4) that signal processing engineers like to refer to their Finite Impulse Response (FIR) filters as "all-zeros" filters as they only have zeros in $z^{-1}$. For them, it is a harmless misstatement; a simplification.

In fact, in the z-plane, the lack of matching finite zeros for the $M$ poles means that there are $M$ zeros at $|z| = \infty$. Those of us that close feedback loops know that on any version of the root locus [13, 317, 15, 14] the closed-loop poles go from the open-loop poles to the open-loop zeros. This means that at some point, $M$ closed-loop poles will be going to those $M$ open-loop zeros at $|z| = \infty$. It is worth noting for the record that $|z| = \infty$ is outside the unit circle, and so fast sampling has not saved us from closed-loop poles streaking towards instability. It merely allows us an opportunity to put more compensation inside the unit circle to allow us to push our closed-loop system a bit further. It is up to us to properly use that opportunity.

## 10.6 Understanding Phase Delay, Phase Noise, and Jitter

We should now discuss timing uncertainty, a quantity known alternately in different fields as phase noise [318] or jitter [319]. Phase noise is a frequency domain term related to time through the relation:

$$\Delta\theta = e^{j\omega\Delta t}. \tag{10.6}$$

Usually, this is considered relative to a particular frequency, such as a carrier frequency: $\omega = \omega_C$. Jitter is a time domain term, usually relative to some sample period, $T_S$:

$$\text{jitter} = \frac{\Delta t}{T_S}. \tag{10.7}$$

For digital systems, jitter is a more common concept than phase noise. Much of what we try to minimize is time delay and jitter because each of these can badly affect our control systems and can be the result of bad computer architecture.

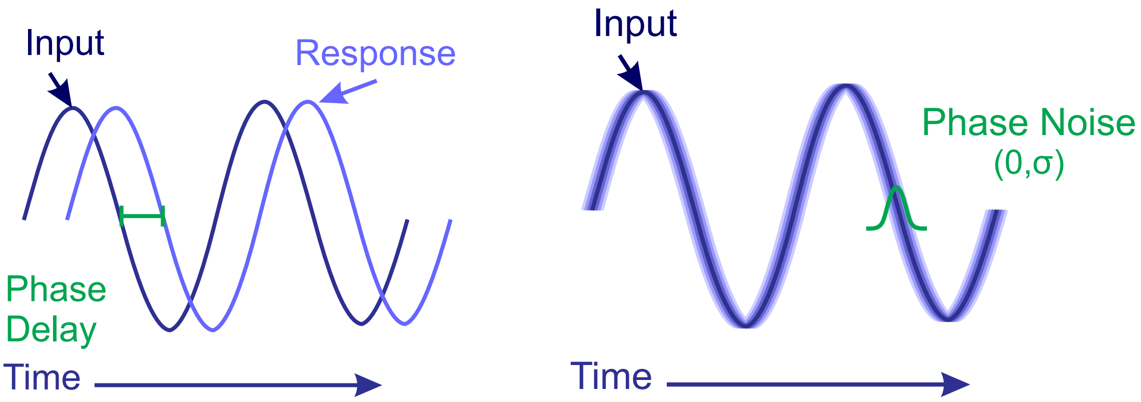


Figure 10.9: Phase delay and noise as seen in a sinusoid.

A few more figures illustrate this idea. In Figure 10.9 we see the difference between phase delay and phase noise in a sinusoidal signal. Phase delay is a predictable lag, phase noise is not. Phase noise makes the exact timing of any part of the signal unknowable. Phase noise is usually characterized by distribution, e.g. a Gaussian. Note that neither phase delay nor phase noise change the maximum or minimum level of the signal, but uncertainty in when signal happens translates into uncertainty in the signal value.

Moving to Figure 10.10, we see many of the same properties of phase delay and phase noise seen in sinusoids apply to square waves. Now, the sharp edges mean that timing uncertainty results in uncertainty in a logic level. Logic levels trigger operations inside programmable logic (PL) or a processor, so jitter

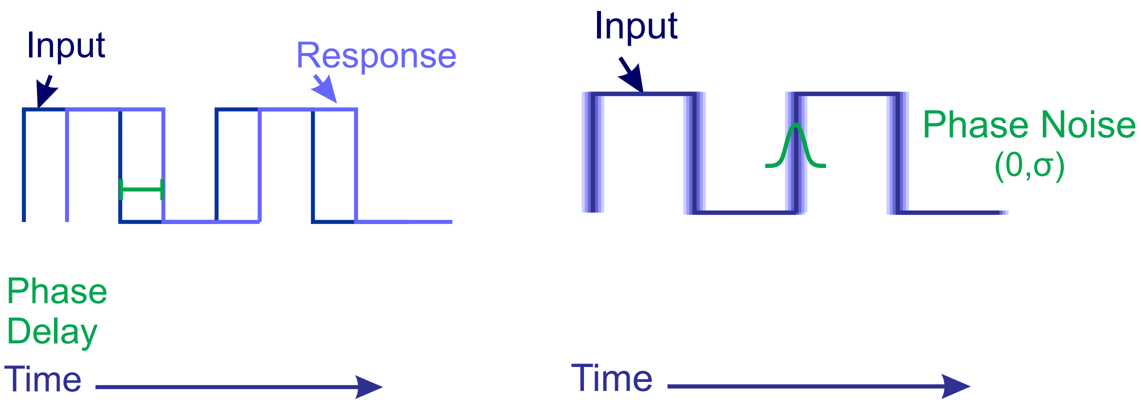


Figure 10.10: Phase delay and noise as seen in a square wave.

results in uncertainty in when operations will be triggered. Whether or not this is important to us is highly dependent on the amount of jitter relative to our sample period. Figure 10.11 illustrates how the same amount of timing uncertainty that would be insignificant for the longer sample period, $T_{S1}$, covers the majority of shorter sample period, $T_{S2}$. The moral of this is that if our physical system time constants allow for a slower sample rate, than we may be far less sensitive to jitter than we would be in a system that requires a sample rate several orders of magnitude higher than the first one.

We get to the last visualization of how jitter can affect us with Figure 10.12. Our illustration here is to show that if our controller computation times not predictable, we may miss the next sample instants and samples. In the illustration of Figure 10.12, the predictable portion of controller computations, $T_{comp}$, takes up most of a sample period, $T_S$. Jitter in the exact calculation time makes missing samples a probabilistic problem. This can even be seen in student laboratory systems [320]. Some systems are passive enough that this does not cause problems for our loop performance. However, for lightly damped, high speed, unstable, and/or nonlinear systems, this could be disastrous.

The take away here is that for high performance systems with relatively short sample periods, we want to minimize computational jitter. As much as possible, we want to have:

- deterministic computations,
- deterministic memory access,
- deterministic sampling, and
- deterministic communications in the digital system.

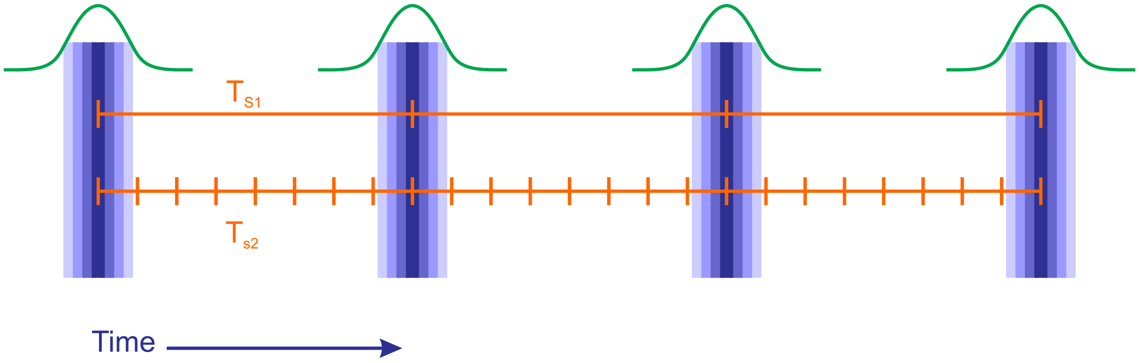


Figure 10.11: Jitter is usually defined relative to a sample period.

These desires frame how we discuss each of the computation chains previewed in Figure 10.3 of Section 10.2.

## 10.7 The Input Signal Chain: The Real World to Computation

For most real systems, there are multiple inputs and outputs, but , for simplicity of concept and visualization, we will stay with a single-input, single-output explanation here.

Figure 10.13 shows a lot of component blocks we often ignore in considering the implementation of getting measurements from the physical system into digital control systems. Sometimes they are simply bundled inside a turn-key system, but for our purposes we want to discuss them individually. Note that the technology changes over time, but the basic functionality of the blocks does not. The path from the physical system to our algorithm inside a computer starts with a sensor.

**Sensor/Transducer:** This is what touches the real world. A lot of science and engineering are employed to convert some physically sensed phenomenon into a calibrated, repeatable, electrical signal. Sensors have their own dynamics, in terms of linearity, time constants, and noise properties. The physical environment often determines what sensors are available and what they cost.

**Sensor Electronics:** These are especially suited to the sensor and environment of the sensor. They

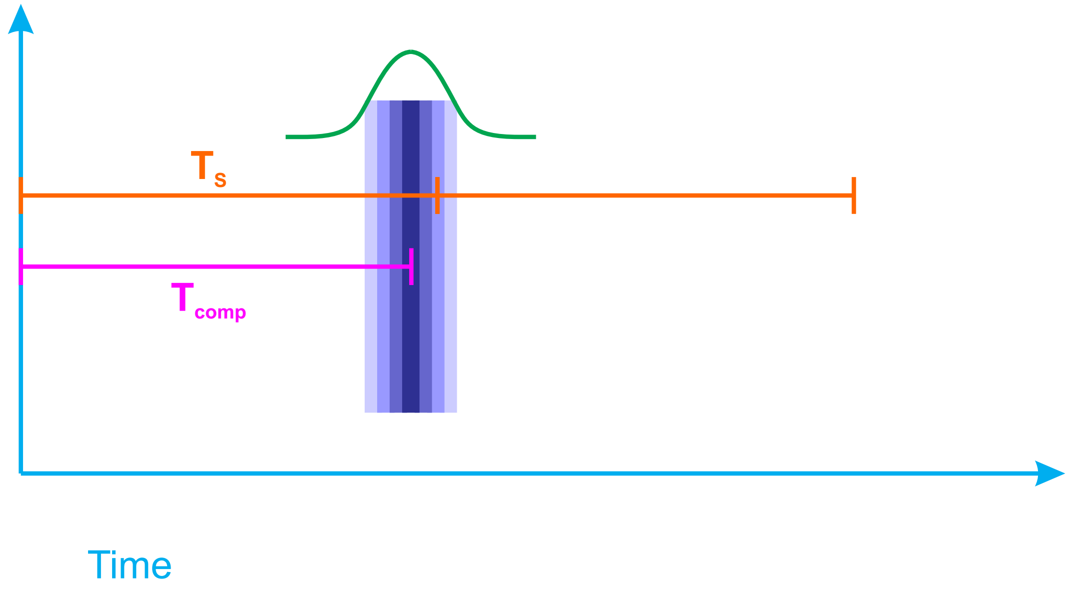


Figure 10.12: Jitter, added onto computational time, may make us miss samples.

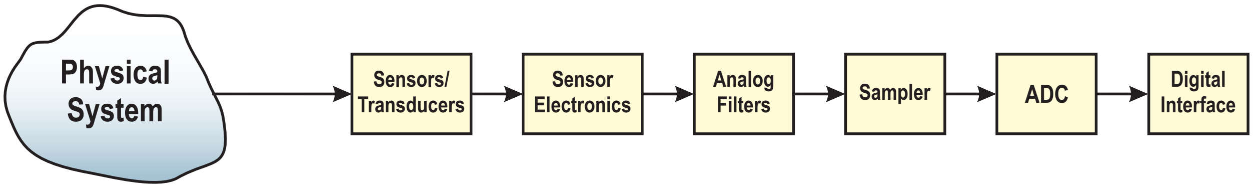


Figure 10.13: An abstracted view of the input signal chain from the physical system to our computation. This one is specific to use in a feedback loop.

typically are packaged with the sensor, but their characteristics and limitations sometimes need to be considered apart from the sensor itself. They are often specialized to handle tough environments, extreme levels of temperature, pressure, moisture, ambient noise, speed, and voltages and currents. Ideally, they get signals into the low voltage, well regulated electronics where we like to do our small-signal filtering.

**Analog Filters:** While there are many elements that can perform a “filtering function”, we focus here on analog electronic filters. These are combinations of operational amplifiers (op amps) , resistors, capacitors, inductors, diodes, transistors, and other components that allow us to implement mathematical filtering functions inside of circuitry.

Why do we need analog filters when we can digitally filter signals inside our computers using only high level languages and none of that wiring and solder? One short answer is that we cannot digitally filter everything. A second short answer is that we can do a much better job of digitally filtering signals that have been cleaned up and normalized by some analog filters. Signals outside of our sampling bandwidth need to be managed with analog filters. Of particular interest to control engineers are anti-alias filters (Section 10.7.1).

**Sampler:** The role of a sampler or sample-and-hold is to capture the well conditioned analog signal (while minimally affecting it) and hold it long enough for the analog-to-digital conversion (ADC) of that signal is done.

**ADC:** Analog-to-Digital Converters (ADCs) are often grouped with samplers when discussed in controls textbooks, but there are many cases in which the sampler can capture signals at a higher rate than a single ADC can convert them. (This phenomena led to a class of digital oscilloscopes called "equivalent time" digital oscilloscopes [321].)

ADCs convert sampled signals into a digital computer compatible signals. Note that we tend to think in terms of floating point numbers when we analyzer control signals, but ADCs (and DACs deal with fixed-point representations. Their quantization is nonlinear, but usually modeled as noise [38]. The number of bits of accuracy ties to cost and conversion time. We will discuss this more in Section 10.7.2.

**Digital Interface:** How the converted signal is delivered to the controller is another fundamental source of possible delay and jitter. There are significant design tradeoffs between the number of signal lines to a converter chip. While we may dream of having dedicated, parallel interfaces to each ADC in a system, the costs of laying out 16+ parallel lines for each analog input and keeping their timing aligned at high speed means that many of the interfaces are serial, with the inherent delay of serialization at the ADC and deserialization at the processor. Is each ADC dedicated to one signal or multiplexed? These issues will be discussed in more detail in Section 10.7.2.

### 10.7.1 Anti-Alias and Oversampling

PES Pareto [2, 298, 322] gave us a way to find the most critical noise sources affecting the error. While noise gets shaped once it's in the loop, we can read "Bode's fine print" if we attack it at the source, before it enters the loop. Almost always, this requires an understanding of analog electronics and how they interact with sampled data system. Here, we focus on the issues posed by anti-alias filters, which are typically implemented as analog low-pass filters (LPFs) with cutoff frequencies at or above the Nyquist

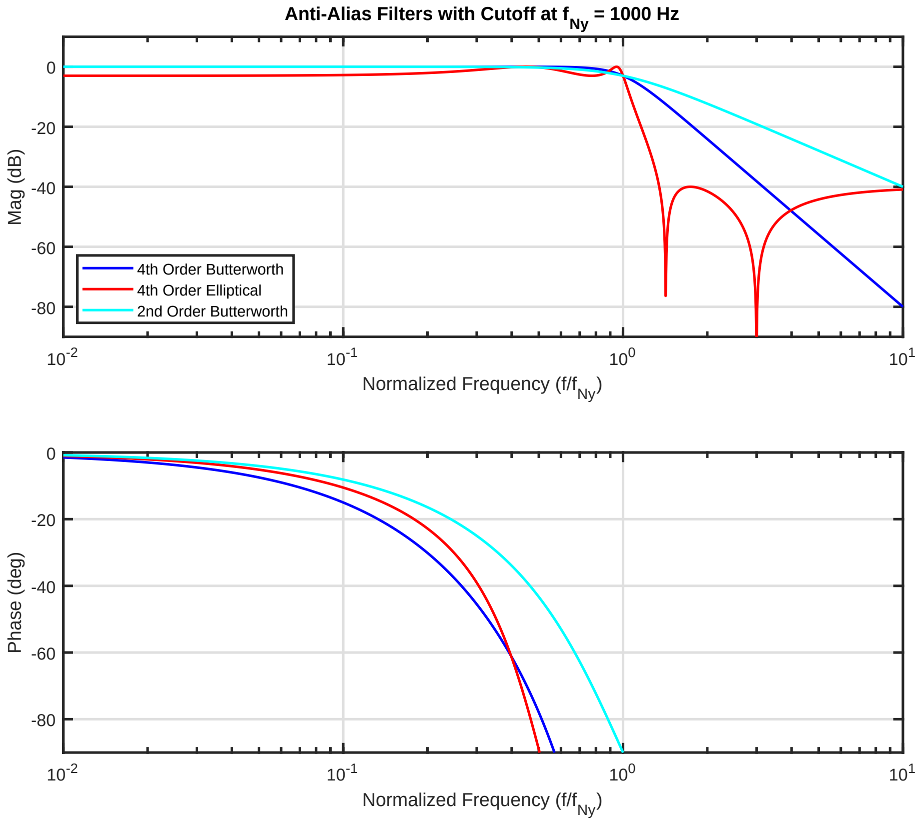


Figure 10.14: Frequency responses of various anti-alias filters. All filters have a DC gain of 1, with the passband ending at the Nyquist Frequency ($f_{Ny} = f_S/2$ = 1kHz here). Under an assumption that the sample frequency is $10$ or $20\times$ the open loop crossover frequency, we can examine the filter phase response, which can significantly degrade the phase margin of the system, as documented in Table 10.1. (Repeat of Figure 7.36.)

frequency.

From our first digital controls class, we are told that we need to apply anti-alias filters in order to avoid aliasing of higher frequency signals into our control bandwidth [15, 323]. What is often overlooked are the effects, particularly in terms of negative phase in the passband, of such filters. Table 10.1 and Figure 10.14 show the effects of three simple anti-alias filters, with their corner frequencies at the Nyquist rate. The simple take away from these is to note the substantial amount of negative phase imparted in order to get attenuation above the Nyquist frequency. The loss of $30^\circ$ of phase margin at $1/10$ the sample rate is nothing to take lightly. The anti-alias filter can have strong phase lag, and this itself can severely limit the achievable bandwidth or destabilize the system. The selection of anti-alias filters must be combined with the selection of sample rate and in this case it seems that what we might think of as "oversampling"

| Filter | Phase at $f_{Ny}/10$ | Phase at $f_{Ny}/5$ | Attenuation at $10 f_{Ny}$ | Peak Gain Beyond Roll Off |
|---|---|---|---|---|
| 4th Order Butterworth | $-15.0276°$ | $-30.1223°$ | $-80.1201$ dB | NA |
| 4th Order Elliptical | $-10.5523°$ | $-22.8086°$ | $-40.8932$ dB | $-40$ dB |
| 2nd Order Butterworth | $-8.1486°$ | $-16.4211°$ | $-40.0605$ dB | NA |

Table 10.1: Phase penalty of representative anti-alias filters. The corner frequency is chosen to be at the Nyquist frequency, half the sample frequency, $f_{Ny} = f_S/2$. Comparisons are made with respect to the Nyquist frequency, as it is considered the limit of intentional digital control action. The two Butterworth filters are flat in the passband, but they incur a larger phase penalty relative to the elliptical filter for stopband gain attenuation they provide. On the other hand, the elliptical filter has up to 3 dB magnitude distortion in the passband. Outside the passband, beyond the Nyquist frequency, the Butterworth filters drop off monotonically, while the Elliptical filter has lobes with the peak gain at approximately $-40$ dB at frequencies up to $100 f_{Ny}$ (beyond the range of the plot). Ultimately, the choice of anti-alias filter structure should not be separated from the available sample rate options, nor the robustness of the system to gain and phase distortions. (Repeat of Table 7.1.)

is fundamental to achieving desired closed-loop performance.

### 10.7.2 Analog to Digital Converters

Figure 10.15 illustrate a few different options for the discussion of Analog to Digital Converters (ADCs). Saying it's an ADC isn't specific enough.

Simply defining a block as an Analog to Digital Converter (ADCs) is not specific enough. The lowest latency version ADC could be a single dedicated converter on a parallel digital bus, eliminating any delay based on serialization/deserialization. Parallel operation "expensive" for two reasons. First of all, it requires far more digital lines be laid out on the circuit board containing the ADC and the path to the processor consuming valuable board real estate. The second cost is that as signaling speeds have gone up, keeping parallel digital signals phase aligned on these buses has gotten substantially more difficult. For these reasons, high-speed serial buses have become increasingly popular inside computing environments, and their tradeoffs may make them faster than the available parallel solutions. Unlike the filtering context (Figure 10.4), those of us in the feedback context (Figure 10.5) need to be highly aware of both the transmission speed and the latency of these channels.

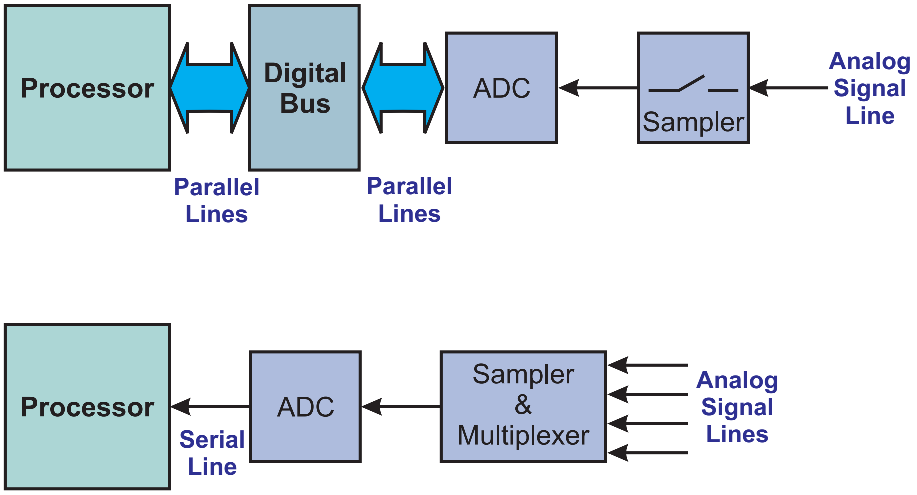


Figure 10.15: Options in analog-to-digital conversions (ADC).

At the other end of the speed/dedicated line spectrum is the shared ADC, comprising a single multiplexed sample and hold handling many input lines and presenting these sequentially to the converter. The ADC does each conversion and then puts the result on a serial line to the processor. The resulting architecture likely has significantly higher latency than the top one. Control and system theory knowledge must guide the design: the more cost effective but slower architecture may have delays orders of magnitude shorter than the physical system time constants. In such a case, demanding the top architecture is wasteful and unnecessary. It is better to yield on these channels quoting the lack of need so that we can be taken more seriously when we need to press for the top architecture.

It is also worth looking inside of the ADC itself. The ADC and sample and hold timing can be diagrammed as in Figure 10.16. Here, $T_{SH}$ represents the sample and hold time, $T_{OUT}$ represents the transmission on the digital interface to the processor, and $T_C$ represents the digital computation time in the ADC conversion. What is often overlooked is that many ADCs speed up their sample rate by pipelining this conversion time into smaller digital processing blocks. Even though the time for any individual sample to reach the processor is longer, the sample period, $T_S$, can be shortened. This is yet another example of architectures that are excellent for DSP applications (filtering context), but have very negative effects on a system that is sensitive to latency (feedback context). When someone not attuned to latency makes the choice, they can unknowingly blow 90% of the phase margin and bandwidth. Such errors cannot be fixed by any algorithm, and so it is critical that folks informed by a knowledge of control principles be involved in the design of the input signal chain. It is not necessary to be the expert in the latter, but only to be conversant enough so as to inform the experts of the latency sensitivity.

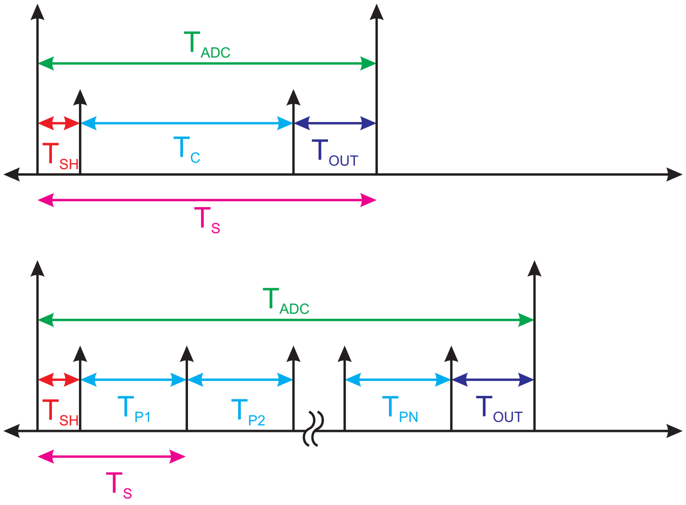


Figure 10.16: Diagrams of sample timing. The lower diagram shows the pipelining of the digital computation needed for conversion.

## 10.8 Quantization "Noise"

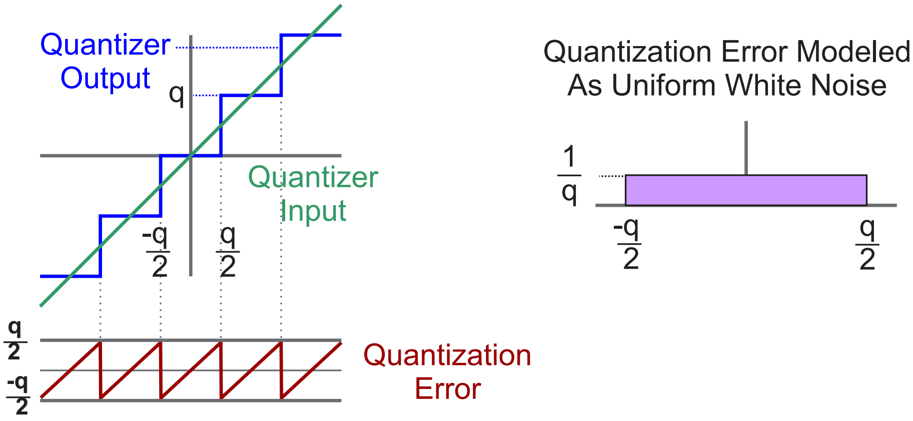


Figure 10.17: Diagram of quantization and Widrow model. (Repeat of Figure 7.12.)

In this section, we return to what is covered more deeply in Section 7.6.3, but now focusing on the role of quantization inside the computer implementation of control systems.

Quantization in a digital system can occur at many levels: in certain sensors and actuators, the numerical computations, or the input and output signal chains. We normally focus on the quantization in the ADC itself, perhaps due to the prevalence of the filtering context in the literature. The Widrow model [38] of quantization is based on an analog of the Nyquist sampling theorem [203]. A conceptual quantizer is shown on the left side of Figure 10.17. Quantization is not a random process, but a deterministic, nonlinear operation. This makes it hard to do anything with passing the math through a filter. Widrow's insight was that while quantization was deterministic and nonlinear, if the signal excited enough of the scale of the quantizer, and the quantization bins were fine enough, that the probability of the quantized signal falling anywhere in the quantization bin could be modeled as a uniform density white noise on the interval $[-\frac{q}{2}, \frac{q}{2}]$, where $q$ is the size of a minimum quantization interval. This is displayed on the right side of Figure 7.12 . With this model, computing the mean and the variance of the quantization "noise" reveals that the mean and variance are:

$$\mu_q = 0 \text{ and } \sigma_q^2 = \frac{q^2}{12}. \tag{10.8}$$

This number for variance is used in texts all over the world [15]. This is one measure of understanding the effective "noise" inherent in the quantizer. The PES Pareto work points out that this is often one of the least important noise sources [2].

There are other time domain measurements that yield only a single number, such as the variance due to quantization in the Widrow model . This variance must be spread across the frequency band in some logical way, so the authors chose to normalize it by the frequency bandwidth. This is consistent with the texts [201, 202] on quantization devices and with will be described briefly here. Note that [202] is erroneous in that the normalization by $T_S = 1/f_S$ has been omitted.

## 10.9 The Output Signal Chain: Computation to the Real World

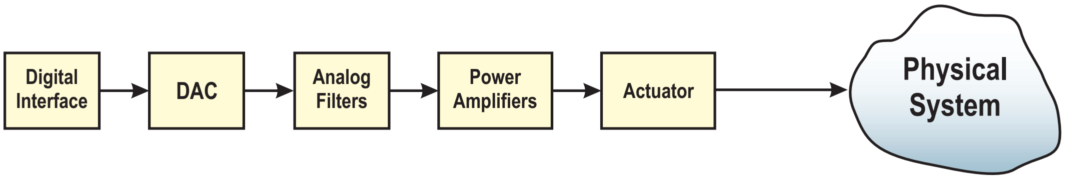


Figure 10.18: An abstracted view of the output signal chain from our computation to the physical system.

A diagram of an output signal chain using a Digital to Analog Converter (DAC) is shown in Figure 10.18. Its components usually include:

**Digital Interface:** How the converted signal is delivered from controller to the DAC is another fundamental source of possible delay and jitter. There are significant design tradeoffs between the number of signal lines to a converter chip. While we may dream of having dedicated, parallel interfaces to each DAC in a system, the costs of laying out 16+ parallel lines and keeping their timing aligned at high speed means that many of the interfaces are serial, with the inherent delay of serialization at the processor and deserialization at the DAC. Is each DAC dedicated to one signal or multiplexed? These issues will be discussed in more detail in Section 10.9.1.

**DAC:** Digital-to-Analog Converters (DACs) convert the computer signal into a well regulated analog voltage. They face many of the same quantization issues as ADCs. We will discuss these more in Section 10.9.1.

**Analog Filters:** On the output chain analog filters are often used to remove digital artifacts or smooth the analog version of digital signals. For example, a digitally produced sine wave will have steps if we look closely enough, but passing it through a low-pass filter (LPF) or band-pass filter (BPF) can considerably smooth that signal.

**Power Amplifier/Drive Electronics:** Theses scale up the voltages and currents produced by the DACs to drive the actuators.

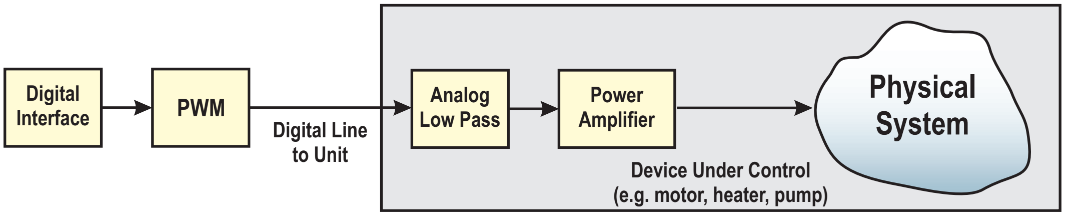


Figure 10.19: An abstracted view of the output signal chain from our computation to the physical system using pulse width modulation (PWM).

**Pulse-Width Modulation (PWM):** A slightly different output chain substitutes pulse width modulation (PWM) in place of DACs. This is diagrammed in Figure 10.19. This is surprisingly common in many industrial environments. The use of PWM depends on the dynamics of plant being much, much slower than the electronics (often true). It's utility lies in part in the fact that a single, serial, binary line is very useful in noisy industrial environments. In most of these applications, the drive electronics have been made part of the actuator/device itself.

**Actuator:** This part pushes on the real world.

### 10.9.1 Digital to Analog Converters

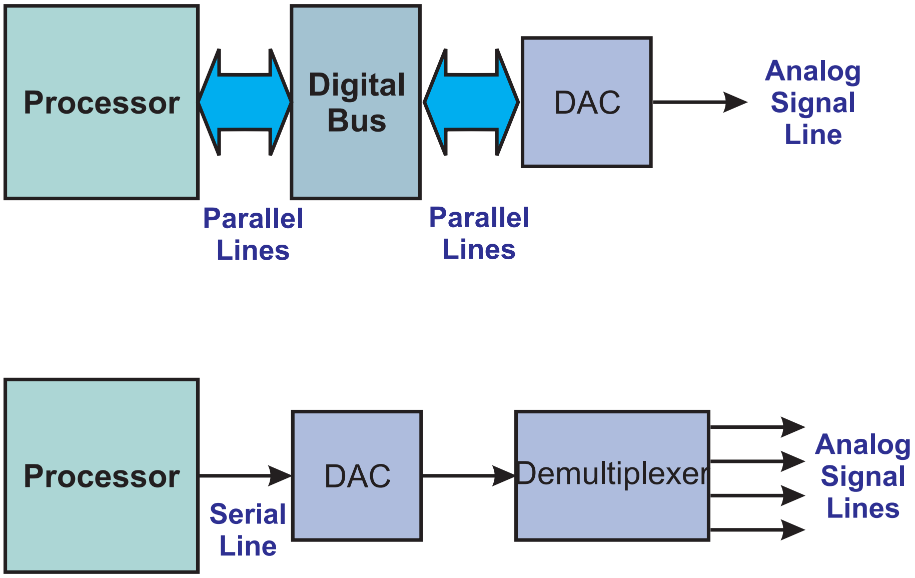


Figure 10.20: Options in digital-to-analog conversions (DAC) .

Some options for Digital to Analog Conversion (DAC) are diagrammed in Figure 10.20. In many respects, they are similar to and duals of the issues for ADCs discussed in Section 10.7.2.

Generally speaking, DACs seem architecturally simpler than ADCs, but this does not mean they do not often include similar pipelining to that shown in Figure 10.16. Again, control engineers need to be involved in the selection of these components so that we can avoid having our phase margin wrecked by digital pipelining in the conversion circuits.

## 10.10 Pulse Width Modulation

We discussed Pulse Width Modulation (PWM) previously in Section 4.19. We return to this discussion now as a portion of the output signal chain. A classic view of PWM is shown in Figure 10.21. A multi-bit number is modulated onto a carrier on a 0-1 digital line such that the duty cycle represents the number. It has the advantages that the single line is cheap and noise immune. The systems driven by this are slow and the system itself integrates/low-pass filters the modulated signal to restore the the original multi-bit number. In essence, the PWM then becomes an inexpensive DAC, but it finds application is a lot of industrial systems such as pumps, heaters, and motors.

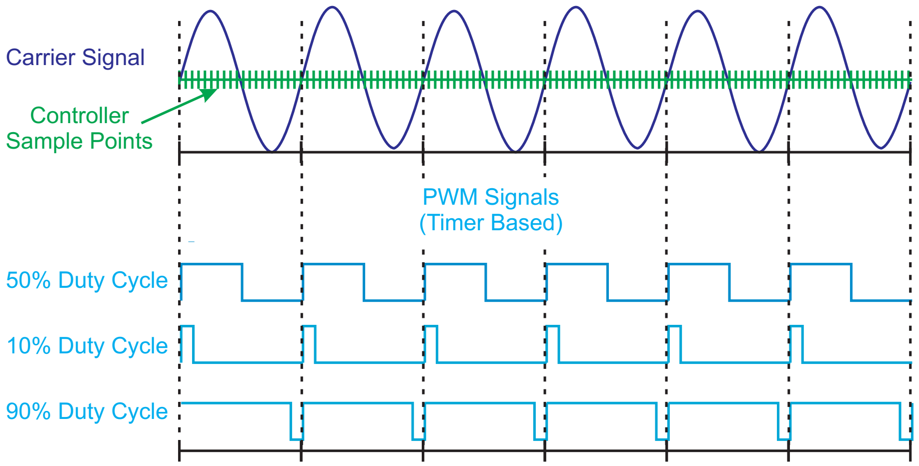


Figure 10.21: Classic PWM converts a multi-bit number into a stream of 1s and 0s whose duty cycle on the carrier encodes the number.

## 10.11 The Plant's "Computation"

The plant itself is the most fundamental piece of computation in our loop, and the one that sets and limits what all the other pieces can and must do. In actuality, the plant is doing some form of physical computation, which we try to model with a combination of first principles (a.k.a. "physics based" or "Newton's laws") and data driven approaches. The plant itself is – for this discussion – fixed, but the models we choose to apply to it are quite varied. We can have:

- A linear, time-invariant (LTI) model for control design.
- A model used for parameter identification.
- A linear, time-varying model, or one with uncertain data sampling, for observer design and operation.
- A complex, nonlinear model on dedicated hardware as a digital twin of the system for health monitoring and simulation.

The point of mentioning these (and many other) forms is that Moore's law allows us to have many of these running in parallel on the same system.

## 10.12 The Computer Itself

We have delayed getting to this section until the other prerequisites computational pieces had been described. At this point, we focus on the broad issues of computation in and out of the real-time environment. Section 10.12.1 will introduce and discuss a highly useful Three-Layer Model of computation in a real-time environment. Section 10.14.3 will get more basic, in the sense of how our choices of filter structure can also affect jitter, latency, and numerical stability.

### 10.12.1 The Three-Layer Model

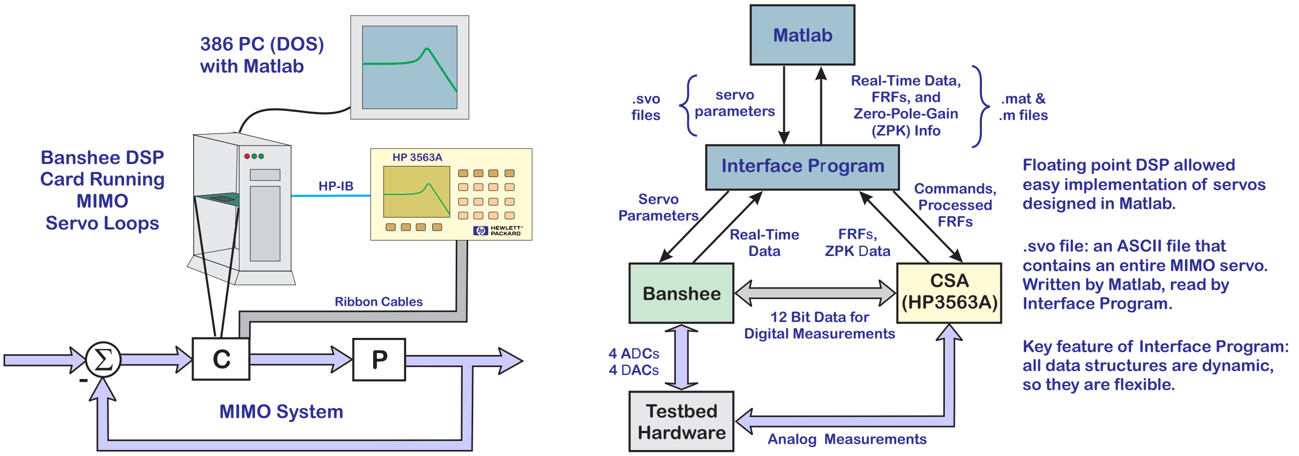


Figure 10.22: The Banshee Multivariable Workstation (BMW). On the left is the physical layout of the system used for enable control systems research for optical and magnetic disk drives at Hewlett-Packard Labs. On the right is the three-layer view of the computation.

This section will present a three-layer model of computation for real-time systems. This split in functionality seems quite common when one examines many real-time systems, but without abstracting out the different layers and their purposes, the smearing of the boundaries can add a lot of confusion about how they should be programmed. In other words, it's always been there, but we didn't see it.

The first author's first experience with this model was in building something called the Banshee Multivariable Workstation in the early 1990s at Hewlett Packard Labs (HPL) [76]. The hardware was an old DOS based PC to hold the upper layer, with a Banshee Floating Point DSP board produced by

Atlanta Signal Processing (ASPI) to run the real-time operations. The intent was to have the floating point DSP run the real-time computations while in the PC we could run Matlab. This is diagrammed on the left of Figure 10.22. With the initial work of Carl Taussig, it was interfaced to the HP 3563A Control Systems Analyzer (CSA), an augmented version of HP's 3562A DSA, that was capable of both analog and digital frequency response function measurements and curve fits [70, 104, 69, 102]. The CSA had a superset of the features of the DSA, specifically enhanced to work with discrete-time systems. Significantly, measurements could be coordinated from the host computer and completed measurements and/or parametric curve fits were uploaded to the host computer to be used in Matlab.

Schematically, the software architecture diagrammed on the right of Figure 10.22 reveals three distinct layers. The most ad-hoc was the Interface Program, used so that Matlab could interact with a real-time DSP. The considerable work to build the Interface Program (mezzanine layer) smoothed out the interactions with the upper level decision functions (Matlab) and the Hard-Real-Time (Banshee DSP Board interacting with disk drive testbed).

This first seemed like a special case, but over the years, the three separate layers kept showing up in all my work projects. in the projects I saw

They were there on Agilent's first 40Gb bit error rate tester (BERT) (sometimes called bit error ratio tester), and in the Agilent (now Keysight) high speed atomic force microscope (AFM). Furthermore, looking at other systems that professed to dealing with the real world, some form of those three layers were always there. However, an elegant exploitation of the three layer model eluded most of what I saw for what I think are the following reasons:

- Technology limits: As I will discuss below, having these separate layers either required putting them on one chip – requiring multitasking of even the hard-real-time, or on multiple chips (or boards) – requiring cross chip/board interfaces that often became the bottleneck.
- Recognition: The three layer model is like one of those old Magic Eye pieces of art work. It is often hard to see the separation, once you see it you can't unsee it, and it's hard to talk about it to someone who hasn't seen it.
- Even when one recognized the three layers, each one required a unique set of programming/logic design tools making it very difficult for any one person to work their way up and down the model and the signal chain. If multiple programming environments were needed, it became increasingly daunting to move algorithms and processing around when one realized something might do better in a different layer.

Over the years, the three-layer-model has remained, and the understanding of what key functionality each of those layers provides has grown. The rest of this section will provide a more universal vision for it.

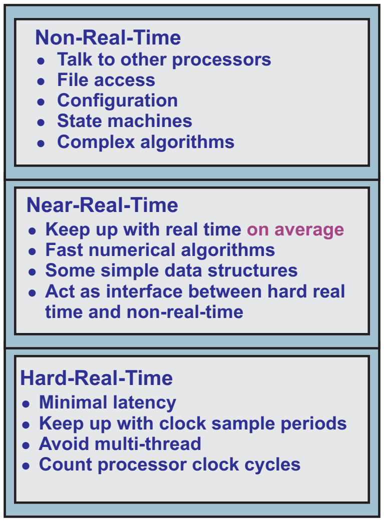

Non-Real-Time
• Talk to other processors
• File access
• Configuration
• State machines
• Complex algorithms
Near-Real-Time
• Keep up with real time on average
• Fast numerical algorithms
• Some simple data structures
• Act as interface between hard real time and non-real-time
Hard-Real-Time
• Minimal latency
• Keep up with clock sample periods
• Avoid multi-thread
• Count processor clock cycles

Figure 10.23: An abstracted view of the three-layer computing model useful for understanding the computing needs of control systems.

Figure 10.23 presents a more general form of a three-layer compute model that serves well for understanding programming in real-time systems. Each layer presents different data, timing, and programming problems. At the bottom is the Hard-Real-Time, in which we are counting clock cycles to make certain we complete computations between sample points. At the top is the Non-Real-Time, which is the computing environment most programming classes are oriented towards. In between is the least well known mezzanine layer, which functions to keep the two other layers happy. It must be able to outrun the Hard-Real-Time on average, but perhaps in a burst mode. It must also have enough memory flexibility to deal with the Non-Real-Time. We will discuss the different forms of this model and how programming against it improves the performance of embedded systems and their real-time layers.

In order to more fully understand this model, it is useful to delve into the layers and the issues that each presents. However, the spoiler alert here is that each layer has a unique set of timing requirements that lead to different programming and memory access methods. Where people get into trouble is when they fail to recognize that programming the different layers requires different mind sets.

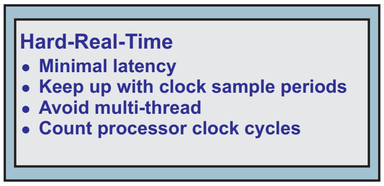

Hard-Real-Time
• Minimal latency
• Keep up with clock sample periods
• Avoid multi-thread
• Count processor clock cycles

Figure 10.24: The hard-real-time layer of computation.

### 10.12.2 Doing Time: Hard Real Time

The first layer to discuss is the real-time layer, which we further delineate as hard-real-time, and diagrammed in Figure 10.24. This is the layer that interacts with system hardware, that "touches the real world" through sensors, actuators, circuits, and data converters. These computations are almost always digital, using discrete math, and while there may come a day when there is good intuition of complex systems sampled at a non-uniform rate, to paraphrase Aragorn, today is not that day. Instead, we almost always rely on a sense of uniform time sampling and of linear transducers (or of quickly linearizing the data once digitized). To miss samples corrupts the model, but to miss samples without realizing that a sample has been missed makes a lot of the data meaningless. The added qualification is intended to emphasize that hard-real-time processing must meet the physical system's timing, not only to maintain physical meaning of the data collected, but also potentially for critical system operational issues.

Missed timing/clocks are bad, especially when controlling devices or trying to make sense of sampled signals. This is the reason for the discussion of Section 10.6. The faster the physical world – relative to the processing – the simpler and more deterministic the processing must be. This layer often features a lot of simple but time-critical tasks. It is possible that the tasks are not that fast, but they do have to happen within a tight time window.

To assign one "large" processor to one of these simple tasks is wasteful. (Metaphorically of using an sledge hammer to push in a thumbtack.) To handle all of these time-critical tasks, we either want a lot of lightweight processors in parallel or want to multiplex in time using a single powerful processor.

FPGAs shine here as one can build small "single task" processors, i.e. application specific processing. In this layer we avoid fancy memory access/caching due to uncertainty in timing. Memory management using caches to hold the most frequently/recently accessed data is fine in other layers, but the possibility of a cache miss – when one requests data that happens to not be in the cache – can increase the time of that memory access by several orders of magnitude. This is a major potential contributor to the operational jitter diagrammed in Figure 10.12. For systems with dynamics that must be tightly controlled, this is unacceptable. Simple, deterministic memory access is critical and so we write algorithms to rely on on-chip memory whenever possible. With modern FPGA tools, we can craft "hardware subroutines" as accelerators for some of our most critical processing steps.

For example, memory access of on chip memory or FPGA RAM blocks takes 1-4 clock cycles, guaranteed. The amount is small and fixed. Accessing memory managed RAM for which the required memory is in cache is also typically 2-6 clock cycles. However, if that desired memory is not in cache, then the access through the memory management unit (MMU) can be 50–100 cycles. This timing uncertainty can be disastrous to a real-time control system if that uncertainty is a substantial fraction of a sample period (again Figure 10.12). If the uncertainty all falls below say 1/10 to 1/20 of a sample period, then for most systems, it can be ignored. This is an example of where understanding the relative speeds of the computation and the physical device to be controlled are critical to understanding whether one can swish in and "just do it", or whether some careful consideration of timing is needed.

The lots of small processors approach is what drives neural network based systems. The lots of medium processors approach is what drives graphics processing unit (GPU) processing algorithms. In all these cases, one of the main issues here is how one parameterizes the problems to break them up into parallel chunks. On the fast AFM FPGA, we manually parallelized tasks into independent streams on different chunks of programmable logic. The other issue for these specialty processing schemes is packaging the data to move in and out of the processor array. For example, the GPU approach can be blindingly fast, but data must be grouped together and passed into the GPU array, then the processed data unpackaged. Often, this works much more easily with batch processing. The alternative is to have some streaming mechanism to get the data through the processing unit. For these kinds of problems, the GPU array is often thought of as a co-processor more than a real-time lightweight processing answer.

One advantage of this was that it took existing, working methods and broke them up appropriately. Also, there was a lot one could do to parallelize a process using pipelining, if one was willing to take an overall latency hit. However, this method made it a lot harder to use automatic tools. Each Lego block had to be hand built and tested.

### 10.12.3 Non Real Time: What we learn in computer science (CS) classes

We next move up to the Non-Real-Time layer, added to the top of the diagram in Figure 10.25. This is the layer that we learn about in most programming and computer science classes. Non-Real-Time

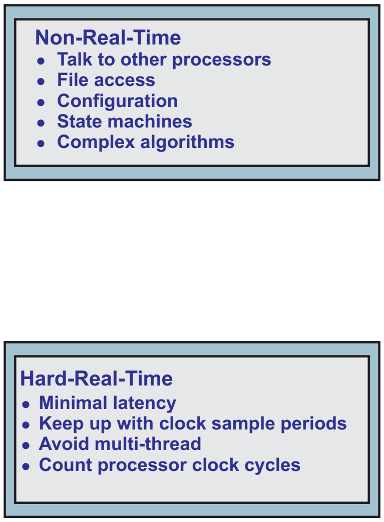


Figure 10.25: Adding the top layer of computing to the hard-real-time.

systems are what most people think of as computers and smart devices. Most users will only interact with embedded systems through this layer, and as it has no critical timing, it needs only be responsive enough to not annoy users. Tasks in the Non-Real-Time layer interact with other systems, interact with users, and generally have a lot of multi-tasked functionality. This is the land of multi-tasking operating systems (often a form of Linux) and Graphical User Interfaces (GUIs).

This is the land of multi-tasking operating systems (OS) (e.g. Linux or Windows) and graphical user interfaces (GUIs). Programming-wise, this is the land of lists, Python servers, web pages, recursive algorithms, database searches, programs that manage lots of dynamic memory, and caching. We can (and should) use much more complex code/algorithms/memory management in this layer than we could ever do in the Hard-Real-Time layer. We can afford to represent our signals and numbers in single or double precision floating point representations.

It's what most folks consider programming, and there are lots of tools to aid the developer. Furthermore, nothing that we program is time critical. In part because folks are so much more comfortable working at this level, lots of companies make money offering to take care of the lower layers [324]. This is certainly a viable approach in many situations, but giving up knowledge of how to program for the lower layers makes it nearly impossible for us to port simpler versions of our algorithms down closer to the data. Giving up the ability to design the lower layers kills our ability to make algorithms agnostic.

One of my old man/curmudgeon rants against how computer science (CS) is taught is that the layers of abstraction in a high level OS and programming language have completely obfuscated the real world.

In some sense that is good. However, for embedded software (ESW), firmware (FW), hardware (HW), and real-world interaction, it is a real problem. Nevertheless, we do not want to be kept away from all these powerful tools, environments, and abstractions. For these reasons, it's good to have those tools available in their own layer that is not responsible for Hard-Real-Time.

### 10.12.4 When Non Real Time is so much faster than the real world

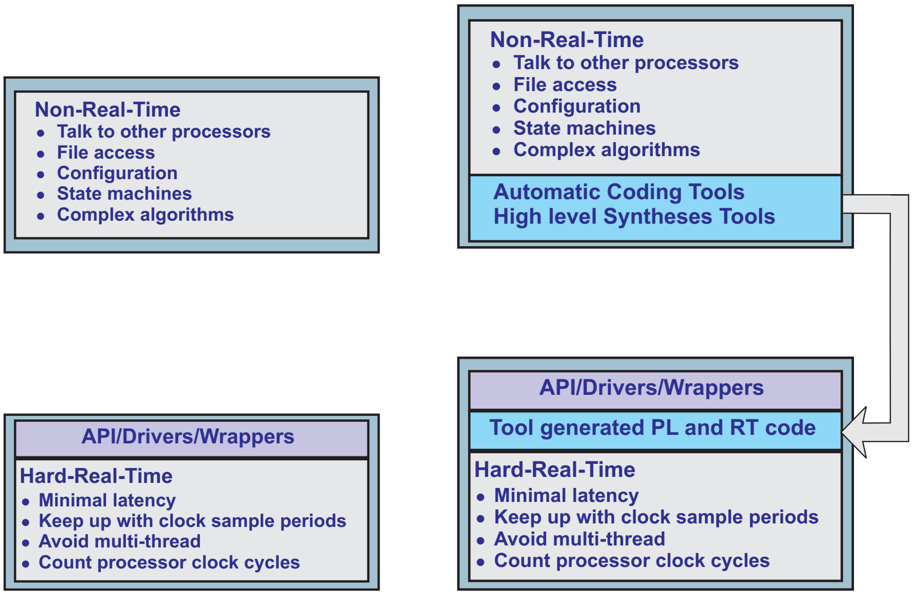


Figure 10.26: Two views of a “heavy-top” layer that does most of the work.

There are times when the processing power is so fast – compared to the physical world dynamics being handled – that folks try to do everything from the top level. In this case many of our computational latency and jitter concerns from Sections 10.5 and 10.6 are not significant for our closed-loop system. Certainly, as the cost of chips has come down, inexpensive, relatively powerful systems on a chip (SOC) such as the Arduino or Raspberry Pi have made it reasonable to address small problems with relatively massive computational power (compared to the physical problem dynamics). Even so, the real-time, “touch the world” blocks are encapsulated into little blocks with application program interfaces (APIs) and drivers. This is diagrammed on the left side of Figure 10.26. There are still real-time, “touch the world” blocks are encapsulated into little blocks with application programming interfaces (APIs). In some cases, small, inexpensive processors (e.g. Arduino, Raspberry Pi) are so cheap that it is worth trying to do all parts of a simple problem on them.

These systems certainly save us from having to deeply consider the middle layer and the cycle-counting programming typical of the Hard-Real-Time layer, but there are two inherent dangers here. The first is that this architecture only really works if the speed of the "main processor" swamps the physical system dynamics and the processing needed. The second is that there is always a temptation to add more applications, threads, and processing requests to main processor. All of a sudden, the we are violating the speed assumptions. Almost everyone who has owned a computer or a smart phone has experienced exactly this latter phenomenon: they were fast when first purchased, but seemed to slow to a crawl in the years that followed. It is unlikely that the circuits got slower. It wasn't that the processing clock got slower (except for apparently certain fruit themed company smart phones); it was that too much stuff had been packed into the once powerful device.

If Sun Tzu was an expert in real-time-systems, he likely would have written: "Know your time constants, and know your dynamics, and you can close one hundred loops without disaster."

### 10.12.5 The advanced tool approach

Yet another common approach is the advanced tool approach, diagrammed on the right of Figure 10.26. The advanced too. approach, which is that many tool supplying companies have realized the difficulty in getting good designs into the Hard-Real-Time layer, and in passing data between the top layer and the bottom layer. They have chosen to solve this through tools that create designs for The idea is that to get more Hard-Real-Time processing without having to code in a Hard-Real-Time way, there are increasing numbers of advanced development tools. When it works, it gives programmers with little or no real-time programming skills access to having algorithms running in Hard-Real-Time Examples include:

- Simulink to FPGA synthesis (HDL Coder).
- MathWorks RT Workshop (MATLAB /Simulink /Stateflow ⟹ real-time hardware)
- Xilinx High Level Synthesis Tools
- dSpace HIL and auto-code (MATLAB /Simulink /Stateflow ⟹ real-time hardware)
- National Instruments HIL simulation
- Hardware targets designed to use these flexible tools

(Mathworks RT Workshop, MATLAB, Simulink, and Statefow are registered trademarks of Mathworks.)

One of the key features of these systems is that they are not fully general. The have specific boards, hardware targets, that are designed for interactions with these systems. Alternately, they develop wrappers for existing, popular board systems. This is great for getting started with development, but the move from development system to product hardware often becomes the bottleneck. This is perhaps not an issue for academics or research labs, but is not truly an answer for products.

The blocks these tools access on the real-time targets are very standardized "Lego " blocks. This has its own overhead in amount of PL used. When it is at a fundamental Xilinx block layer, the overhead is low, but often the blocks associated with the advanced tools and the generality that they need comes with a significant speed hit.

At the same time, the tools have gotten cheaper and more powerful. The speed of processing is getting faster while the real world has mostly the same time constants. (One caveat: as the speed of processing goes up, the set of physical systems that one considers prime for digital interaction also goes up. Hence, the Internet of Things, UAVs, autonomous vehicles, etc.) These tools allow one to stay in their thinking set of tools (e.g. MATLAB , **Simulink**, LabView) and not have to deal with the much of the real world as much. However, doing engineering without touching the real world has a lot of downsides. The more I have worked, the more strongly I have believed that the person who designs it should have a big role in making it work. This methodology is available and as the tools improve more and more chunks of our design might be implemented this way, at least in the first cut.

### 10.12.6 Issues with connecting the layers

Having passed through these variants, we are now in a position to have a deeper discussion of the full three-layer-model displayed in Figure 10.27. What has been missing from our previous examples has been a fully fledged middle/mezzanine layer. The BMW had a definitive mezzanine layer, as denoted by the Interface Program of Figure 10.22. In that example, it served as a bridge between the highly sophisticated layer of Matlab design and data analysis and the Hard-Real-Time of the DSP board and testbed.

It turns out that the processing, data, timing, and memory models of the Non-Real-Time and Hard-Real-Time are so different that some sort of rubber/glue layer is almost always needed. A term that serves

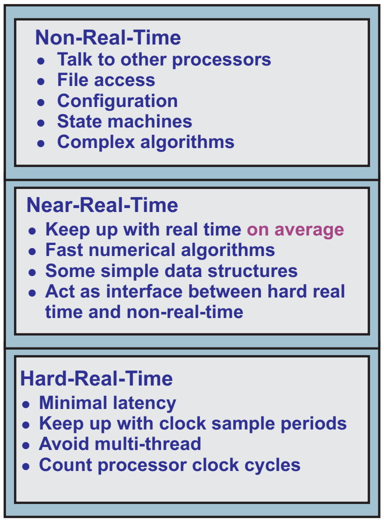

Non-Real-Time
• Talk to other processors
• File access
• Configuration
• State machines
• Complex algorithms
Near-Real-Time
• Keep up with real time on average
• Fast numerical algorithms
• Some simple data structures
• Act as interface between hard real time and non-real-time
Hard-Real-Time
• Minimal latency
• Keep up with clock sample periods
• Avoid multi-thread
• Count processor clock cycles

Figure 10.27: The near-real-time (mezzanine) layer connects the two.

for this is Near-Real-Time, or Mezzanine Level Processing. It has to outrun the Hard-Real-Time – but only on average. It features a lot of memory buffers and message queues between the two other layers. This middle layer wants to ensure that it never leaves the Hard-Real-Time layer without inputs in its queue and that it clears the Hard-Real-Time's output queues fast enough that they are never completely full. Doing so means that the most time-critical layer, the one that has to keep up with Newton's laws, never has to slow down because of the rest of the embedded computing system. We can use more complex memory, processing, algorithms in the Near-Real-Time. Once tasks in this layer get past their initialization and allocate their memory, they go faster than the Hard-Real-Time (hence the need for buffers). Programs in this layer can also handle a lot of monolithic streaming data.

Depending upon the speed of the physical system dynamics, this Mezzanine Layer can be handled with either a dedicated Linux thread, a Real-Time Operating System (RTOS), or a bare metal (no operating system) implementation. However, unlike the layers on either side, far less is formalized about this layer. Some of this is due to the very purpose of this layer: to form a smooth interface between the others, however this can be viewed as the Wild, Wild West of programming.

Even if one recognizes the layers of Figure 10.27 and programs them differently, there are the issues of how to combine them and how to share data between them. Traditionally, the choices have been:

- To combine multiple layers on one processor. However, the swapping and the multi-tasking could cause complexity and timing problems. By putting everything on one processor, it is difficult to give the Hard-Real-Time layer the priority it needs and wasteful to have a single large processor handling so many simple real-time tasks.

- To put different layers on different processors/chips/boards. While this preserves the independence between the layers, the hand-offs of data and control between layers was usually a bottleneck. The interfaces end up being either low overhead and slow (old parallel buses) or higher overhead and fast (fast serial buses). (Fast parallel buses between boards are disappearing from everything but video controllers.) Crossing boundaries is often a nightmare, as we are having to create compatible channels between very different levels of programming. Furthermore, being on different boards usually implies that the processors are on different clock domains. This gives yet another opportunity for longer and less predictable delay between different computation layers.

However, without moving data and processing easily between the layers, it was very hard for processing to be agnostic, for algorithms to work at whatever level then need to for the data. Moving the data between the different layers is a key enabler to being able to program for each of the layers. Moving the data around cleanly is also necessary for making processing agnostic, that is, moving the processing to the layer which is most appropriate for the data and the sample rate. If data moves easily, and the development tools are similar at the different layers, then the choice of where to process data becomes driven by the data itself, not by some "layer preference".

The key to a solution allows for separate hardware segments for each layer while still enabling fast transfer of data between layers. In this respect, chips which feature a System on a Chip (SoC) provide such a solution. Multiple chip families from makers such as Altera (now part of Intel) and Xilinx (now part of AMD), provide multiple high level ARM processors on the same die as programmable logic (PL), on-chip-memory (OCM) as well as communication and interfaces for off chip memory [325, 326]. This allows multiple levels of computing on one piece of Silicon (processing, programmable logic, memory, buses), and the chip makers are moving towards advanced synthesis tools to "compile" hardware to meet the system timing requirements (or alert the designer when they cannot be met).

Each of these layers has unique timing constraints. The hard-real-time layer must match up to the physical world timing needs, and that affects how the programming is done. One might want to write code with simple math in it, but it turns out that division is about 8 times as computationally expensive as multiplication. When possible, this means precomputing $x_{inv} = 1/x$ instead of dividing by $x$ at each time step. One may be able to access a library for trigonometric functions, but again, it is easier to put them in a look-up-table. One gives up mathematical exactness for computational speed and a known latency. Algorithms that require large amounts of memory are replaced with ones that use fixed, direct access memory. All of these compromises are things that one would never do on "big iron" or even an Intel or AMD processor, but they are the key to making computations fit into real-time processors.

On the other hand, such programming lacks the ability to add the kind of intelligence that most modern controllers should have. There is no user interface, no file access, no tuning algorithms. There is no access to libraries or GPU processing. There is little communication and exchange with other software. There is little ability to explore strange new algorithms, to see out new methods, and new computations. This type of programming should be there, but it has to be in a separate layer.

The Rodney Dangerfield of the group (the one that gets no respect) is the middle, Near-Real-Time layer. This is likely to be the most customized layer in the system, because it has to joint the rich data centric world of the non-real-time upper layer with the physical world timing constraints of the hard-real-time layer.

While one may use the same set of tools to program the different layers, each has a unique set of timing requirements that lead to different programming and memory access methods. Where people get into trouble is when they fail to recognize that programming the different layers requires different mind sets.

## 10.13 Control Algorithm Programming

The prior sections have focused at a high level on the many issues that can encumber the proper execution of control algorithms. In this section, we will discuss specifics in how we code our filters and our state-space structures to minimize issues such as latency, jitter, and numerical instability in the code. There are many wonderful texts describing the different mathematical benefits of different controller structures. What is sometimes left to the reader is first how to implement those structures (a.k.a. write code) and how the particular properties of the compute engine place limits on how we write that code.

We cannot emphasize enough the principle that for algorithms to make it from the notepad, through Matlab or Python, and into systems that work, they must be debuggable. This goes beyond commenting one's code (although this helps) but goes into the filter and state-space structures into which we fit our schemes. Structure, compartmentalization, and documentation matter. The clueless schmuck who looks at your code in 6 months or 2 years will likely be you. Throw that schmuck a bone.

### 10.13.1 It's a Filter

In most human-built implementations of feedback controllers, the computations take the form of weighted values of functions of prior inputs to the plant, outputs from the plant, reference signals, and auxiliary sensor inputs to the controller block. In other words, they implement one or more filters. Whether implemented using analog circuits [327], digital logic, or computer code, controllers involve filters (and decision trees). We will focus on the digital versions of these, starting with computer code but branching into digital logic as a special "hardware implementation" of code. Even a state-space structure can be considered under this "filter" rubric [14, 15, 328, 329].

When we talk about filters in code, we most often mean that the filter is in a subroutine (or function or subprogram, but for our purposes here, these are the same). While filters may be coded in-line in small systems, best practices of compartmentalizing coding dictate that most filters will be implemented in subroutines. The subroutines might themselves have their own subprograms, but for what we are trying to explain, one level of subprogram is enough. Considering the most simple (and common) case of a linear, constant-coefficient filter, we will have parameters into the routine that need to be shared from the top level down to the routine and some that go back. As subroutines generally have a separate data space from that of the calling routine, the parameters need to be either:

- **Global:** Shared between all routines and the top level program.
- **Passed:** These are passed down via the stack during the subroutine call.
- **Static/persistent:** These keep their value even after the routine ends so that on the next call they remember their previous value.
- **Part of an instance of a class:** This is static data local to an instance of a class that gets initialized through the class, but also maintains its value until the instance is deleted.

Of issue here is what data needs to be known in the filter routine and which does not. In our LTI filter example, we can imagine the common form for a polynomial form IIR filter as:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) \\ &\quad +b_0 u(k) + b_1 u(k-1) + \ldots + b_n u(k-n). \end{aligned} \tag{10.9}$$

In this filter, we have have three sets of parameters:

- **Filter Coefficients:** Our values of $\{a_i, b_j\}$ define what we think of as a filter as they are the filter coefficients. Most often, these are defined before the filter is ever run and they do not change.

- **Previous filter inputs and outputs:** This signifies the $\{y(k-i)\}$ and $\{u(k-i)\}$ values that are known prior to the current time step.

- **Current filter inputs and outputs:** This signifies $u(k)$, the current input at step $k$, and $y(k)$, the output of the filter to be computed based upon the coefficients, the prior inputs and outputs, and the most recent input to the filter.

In a very simplified understanding, only $u(k)$ is a new input to the filter, and only $y(k)$ is a new output. Given the right programming methods, no other parameters from above need to be passed in to or out of the filter at any one step.

This is a very good spot to mention the difference between batch mode programming and sequential programming. In batch mode programming, we present all of the data to the subroutine or program at the entry point. It applies the parameters and generates all of the data to be output. In this mode, the routine would be called only once. It would execute its operations on the large data space, then return the result in (probably) an equally large data space. We are used to this type of programming for much of our off-line data processing, e.g. when we run filter() or filtfilt() in Matlab. While the filter of Equation 10.9 might only process one new input at a time, in batch mode processing all the data would be passed down to the filter routine which would loop through repeatedly before returning a column of results.

However, this type of programming makes no sense in the most critical operations of a feedback controller. Instead of a batch of data, we have new data coming in at each time step (probably) and need to generate a response at each time step (probably). The filter routine only sees a tiny bit of new data at each step and must respond to it. It is this incremental use of filters and programs that we will focus on, since these are the ones that will be running in a feedback loop that must run "forever". In this mode of programming, passing the coefficients and old inputs and outputs on the parameter stack every time the filter routine is called is tremendously wasteful of computation time. It does not advance our algorithm; only increasing our computation delay while adding no functionality. In such programs, it is far more efficient to have both filter coefficients and signal values (delays/states) as static/persistent data so that only new information gets passed on the parameter stack.

### 10.13.2 The Wire

Besides being a classic HBO show, The Wire is the name the first author gives to the first filter subroutine one writes in any computer environment. The role of The Wire is akin to the role of the various "Hello world!" programs in programming environments: it is the first proof of concept that helps debug the basic structure of the code. The Wire is simply a filter that takes the input and passes it to the output. In terms of Equation 10.9, all coefficients are $0$ except for $b_0$ which is $1$. By first coding The Wire engineers can debug their filter code structure without worrying about numerics, time constants, and discretization.

Once The Wire is working, the next version involves having all coefficients at $0$ except for $b_0$ and $b_1$ which are both set to $0.5$. Once this simple FIR averager is working, the next version sets all coefficients to $0$ except for $b_0 = 1$ and $a_1 = -0.5$ to set up a simple IIR low-pass filter. With these, one can test impulse and step responses and generally debug most of the filter structure. After that, one can get more complicated with the numerics.

## 10.14 Numerics, Parameterization, and Operations

The successful implementation of algorithms into filter routines relies on doing a variety of different things well, or at the very least, not screwing up a whole bunch of small things. Experienced programmers (just like experienced practicing engineers) realize that every implementation will need to be debugged, and so a key aspect of coding is to design the code in such a way that it can be easily debugged. This is one of the reasons for breaking code up into smaller subprograms or subroutines. Taken to the next level, we get to Object-Oriented Programming (OOP) where much of the data and code specific to that data, are wrapped up into a class. (Discussing OOP in detail is beyond the scope of this tutorial, but it some parts are keenly relevant to this discussion.)

Those used to programming in feature rich, Non-Real-Time environments, can often not see the need for such careful attention to structure, but for any practical programming system, structure comes before numerics. Once the data handling is properly handled, then one can look at how the algorithm handles the numerics. This is not an either-or situation: the structure puts one in a position to better understand the numerics and the numerics then may well mandate a change to the structure.

A feature of programs that needs to be understood is the cost of different mathematical operations.

In most modern computation architectures, additions, subtractions, and multiplications take 1–5 clock cycles while operations such as division or computation of transcendental functions take on the order of 30+ clock cycles (depending upon the particular processor). Accesses to on-chip memory take 1–2 clock cycles and so certain real-time operations are far better accomplished via a local look-up-table (LUT) and interpolation, than by following the complete algorithm.

When working with Hard-Real-Time, one might often need to give up on floating-point operations in the name of resources and speed. For example, using Xilinx's DSP48E blocks in PL, one can perform a multiply of a 25 and 18-bit twos-complement number and accumulate with another 25-bit number in 5 clock cycles with one block. To perform the same calculation using floating-point requires 10 clock cycles and four of these same blocks [88, 176]. Thus, when the precision is less critical than speed and resources, one may very well opt for fixed-point operations.

### 10.14.1 Understanding Sampling and Discretization Methods

While there are many ways to discretize a linear model, the Zero-Order Hold Equivalent (ZOH) [15] has been the most used form for many years. As it is the default method for Matlab's c2d() function, this dominance has only increased. The ZOH equivalent provides exact matches at the sample instances, but with it comes a loss of physical understandability [322]. Its use only makes sense when one is discretizing the entire plant model in one step. However, as John Madden famously taught, "One size doesn't fit all" [330].

For example, most PID controllers are discretized using a Backwards Rectangular Rule equivalent [322]. This will be discussed in Section 10.14.2, but it is worth noting that the only place in Matlab where one finds the Backwards Rule built in is in the PID design tools. Matching higher end resonant structures is far more intuitive if one breaks them into second order biquads. For such lightly damped second order sections, discretizing them with pole-zero matching is not only highly accurate, but preserves the physical intuition for each biquad (Sections 10.14.4 and 10.14.5). If one wishes to model the rigid body modes of a system in such a way as to directly extract both position and velocity directly from the discrete-time state space, a Trapezoidal Rule equivalent makes a lot of sense (Section 10.14.6).

Finally, while the "sampling fast" mantra does bring a lot of benefits, it can also bring forth several issues. The first is that if one samples so rapidly that the measurement and quantization noise is larger than any signal change in the sample instant, the signal-to-noise ratio (SNR) of our measurements drop, with the consequentially bad results. Secondly, because all discretization is calculating or approximating some version of $e^{sT_S}$, the shrinking value of $T_S$ will squeeze more poles and zeros into a pack around $z = 1$. This may not matter when the numbers are represented in double-precision floating point, but for real-time systems with single-precision floating point or even fixed-point coefficients, this can be disastrous. Trimming a few bits of the numeric representation can flip poles and zeros from inside of to outside of the unit circle in unpredictable ways [33]. Several schemes have been proposed to shift the design space back to a more continuous time like representation, including the $\delta$ parameterization [177, 179] and the $\tau$ parameterization [331]. The $\Delta$ coefficients used in the Multinotch (Section 10.14.4) [33], change the coefficients to be more accurate but do not alter the signal space. For biquad structures, they provide greater accuracy of coefficients [165] but do not do anything about signal growth. However, for the signal growth limiting properties of the $\delta$ parameterization can actually be traced to the fact that $T_S$ ends up scaling many of the signals. As this shrinks, it compensates for having more values in the accumulator [166, 331]. The issue with this scaling by $T_S$ is negligible when the numbers are represented in floating point, but potentially disastrous when $T_S$ is in an unscaled fixed point number. All of this is to say that sampling fast should be done fully aware of the potential downsides.

### 10.14.2 PIDs

While many controls researchers see the Proportional, Integral, Derivative (PID) controller as “Brand X” controller against which to compare their research [322], it is still the case that most practical controllers are (or start with) some variant of a PID. There are many good discussions on various forms, aspects, and tuning of PID controllers [115, 118, 120, 323], we will focus here on some of the important computational features. One point to be clarified is that while PIDs are considered “standard” there is no one standard PID form. There is an attempt to consolidate many of the forms into one of 4 different archetypes in [40], and [17] adds the ISA standard form PID equation [332], will focus in one one form from [40] that has nice discretization properties

**Design:** Modern PIDs are typically designed in continuous time and implemented in discrete time. Almost universally, discretization is done using the Backwards Rectangular Rule (or Backwards Rule), where

$$s \longrightarrow \frac{z-1}{T_S z} = \frac{1-z^{-1}}{T_S}, \text{ and} \tag{10.10}$$

$T_S$ is the sample period. Taking one continuous time form, Explicit Time with No Derivative Filtering from [40]:

$$C(s) = K_P + \frac{K_I}{T_I s} + K_D T_D s, \tag{10.11}$$

where $K_P, K_I$, and $K_D$ is the proportional, integral, and derivative gains, while $T_I$ and $T_D$ are integration and differentiation times. Setting $T_D = T_I = T_S$ and applying the Backwards Rule from (10.10) yields

$$C(z) = K_P + \frac{K_I}{z-1} + K_D\left(\frac{z-1}{z}\right). \tag{10.12}$$

We see that modulo the $T_I$ and $T_D$ factors there is a tight correspondence between (10.12) and (10.11). Using this parameterization and the Backwards Rule has not only led to a very intutive correspondence between the continuous and discrete time PID parameters, but it has also made the non-proper derivative term in (10.11) proper in (10.12). In essence, the conservatism of the Backwards Rule has inserted a needed low-pass filter. Operationally, this means that the discrete PID is internally stable, so long as the integrator is not operated when the actuator is in saturation. This last issue is one of the likely reasons why PIDs are broken out from the rest of the filter blocks in most industrial systems. Being able to isolate the integrator for anti-windup purposes is extremely helpful.

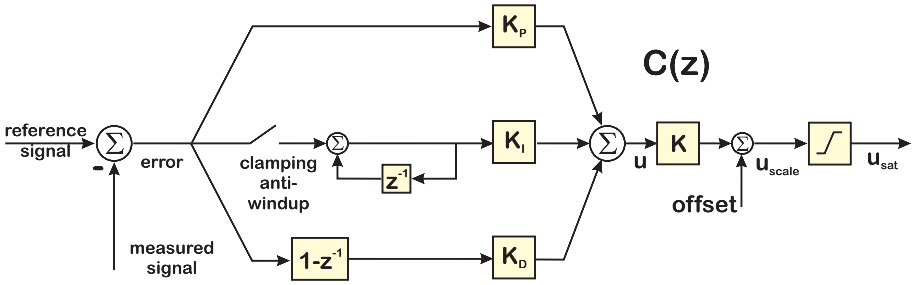


Figure 10.28: Conditional integration/integrator clamping.

**Anti-Windup:** We know from the Final Value Theorem that in order to track an input step with 0 steady state error, we need an integrator in the forward portion of the loop [14]. The issue for an integrator in the controller is that it is only stable in the presence of the feedback loop. Saturation – typically at the actuator – breaks that feedback. The breaking of the loop can cause a buildup of error in the integrator (wind up) causing the actual error to take much longer to settle when the system comes out of saturation.

An advantage of the PID structure is that the integrator can be isolated so that anti-windup schemes can be implemented. The workaround of integrator anti-windup involves some method of detecting the saturation and then using this to change/limit the input to the integrator. The two prevalent methods are back-calculation and conditional integration (also known as integrator clamping). Back calculation aims to remove from the integrator input a portion of the controller output that did not get past the saturation block. If properly scaled, this should drive the input to the integrator to 0, eliminating the windup. Most descriptions of this seem to focus on the integrator or the PI action, giving the impression this is usually reserved for systems where the D portion is not enabled. Back calculation has an advantage in that it can be readily implemented in continuous or discrete-time implementations. Conditional integration, diagrammed in Figure 10.28, implements a decision tree to turn zero out the input to the integrator during saturation. As such, it is more easily understood with discrete-time implementations. It also has the advantage that even in saturation, input to the integrator may be allowed if the sign of the error is different from the output of the integrator, thereby moving away from saturation.

**Derivative Filtering:** Depending upon the application, various forms of filtering are often recommended. When the plant dynamics are substantially slower than the computation, one can apply fairly aggressive low-pass filtering to the entire PID to limit electronics noise while not affecting the needed closed-loop performance [118]. For higher speed systems such as mechatronics, we may stick with only filtering the derivative term [322, 40].

The point of this is that the most common PID blocks should be implemented in their own special filter subroutine. The form of discretization matters can affect the intuition one keeps about the overall control system. Furthermore, this block can contain some anti-windup code, not found in most other filter blocks.

One more feature of integrators is that even without saturation and windup, their internal signals can get quite large compared to other filter signals. When using fixed point number formats, we often need to allocate extra bits just for the integrator accumulation.

### 10.14.3 Filter Structures and Latency

As Section 10.7.1 showed the phase-margin killing effects of careless selection of anti-alias filters, this section deals with computational latency. In particular, Figure 10.29 illustrates how the lack of precalculation makes the closed-loop latency dependent on the controller filter size. Restructuring the calculation to push as much as possible into precalculation makes the computational latency fixed and shorter. It is relatively straightforward to apply precalculation on a controller implemented as an IIR filter as in Figure 10.30, but polynomial filters have poor numerical properties, particularly when the filter has lightly damped poles and zeros. Since these are common in mechatronic systems, we want to implement our control filters using a biquad cascade that has better numerical properties than a polynomial filter [54].

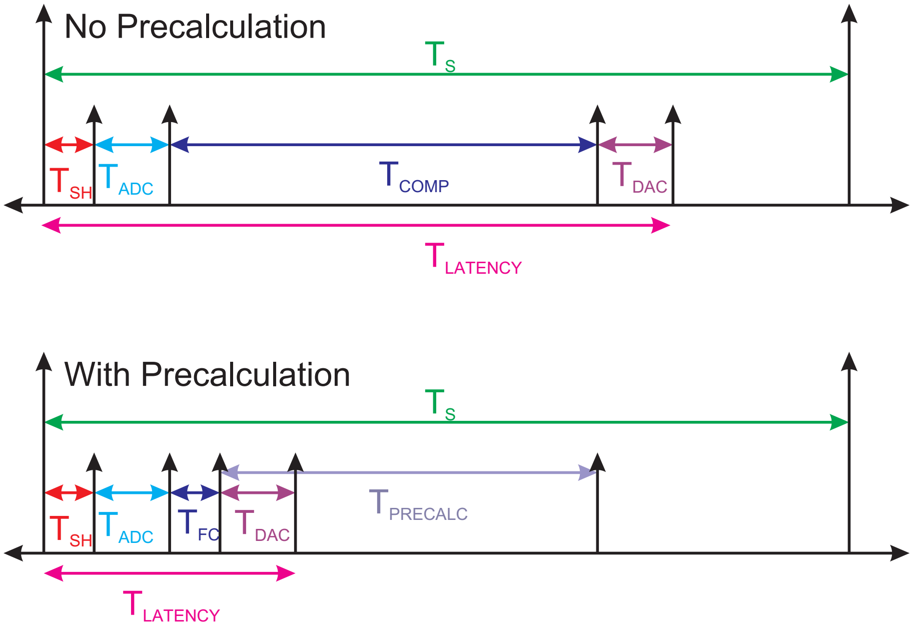


Figure 10.29: Input and output timing in a digital control system. The top drawing is without precalculation; the bottom drawing is with. (Repeat of Figure 6.1.)

### 10.14.4 The Multinotch

This section returns to the discussion of Section 6.11, but with a focus on the computation itself.

The development of the BSS starts with the Multinotch, a way of turning a polynomial form IIR filter as diagrammed in Figure 10.30, into a cascade of biquads with the direct feedthrough coefficients factored out to the end, as shown in Figure 10.31 [33, 54]. With this factorization, one can discretize each biquad individually so that the discretized biquads have a one-to-one correspondence with the continuous-time biquads. By judicious choices of the which poles and zeros from the physical model are assigned to each biquad, one can minimize effects of the signals of any one biquad on the others. The Multinotch is a highly efficient digital filter because it not only has greater numerical stability than standard polynomial and state-space forms, but also allows for precalculation of most of the filter, minimizing the latency between reading a sample and responding to it (Figure 10.29). This particular form of the direct feedthrough scaling allows for precalculation, but others are available if we want the internal states of the filter to be scaled as they are in the system.

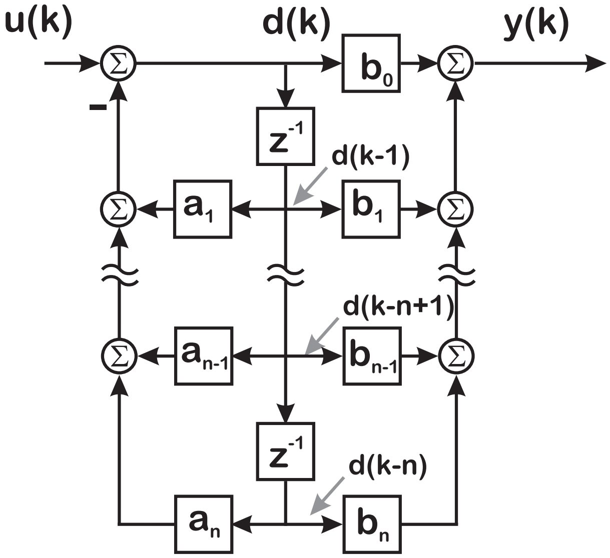


Figure 10.30: $n^{th}$ order polynomial filter in Direct Form II configuration [167]. (Repeat of Figure 6.2.

### 10.14.5 The Biquad State-Space (BSS)

In this section, we repeat and summarize the discussion of Section 9.15, again with a focus on computation.

One of the easiest ways to clear a room full of practicing mechatronic control engineers is to suggest that they employ state-space methods for the control of their structure with many lightly damped resonances. State-space models of highly flexible systems can present severe numerical issues. The models derived from physical principles often lack structure. Canonical form models, are compact, but obscure any physical structure and can have coefficients that are highly sensitive to model parameters. What is needed is a form that has the compact representation of the canonical forms, the physicality of the forms derived from physical equations, and maintain numerical accuracy and physical intuition, even after discretization.

The first of these is the Biquad State Space (BSS) [3, 4], based on the Multinotch of Section 6.11. The BSS captures the endearing characteristics of the Multinotch while providing the flexibility of model based control. A significant feature of the BSS is the ability to move easily between the states of the continuous and discrete time forms.

The digital version of the BSS shown in Figure 10.32 looks very similar to the Multinotch, although as we are more focused on accurate modeling than precomputation, we scale the outputs of each biquad to

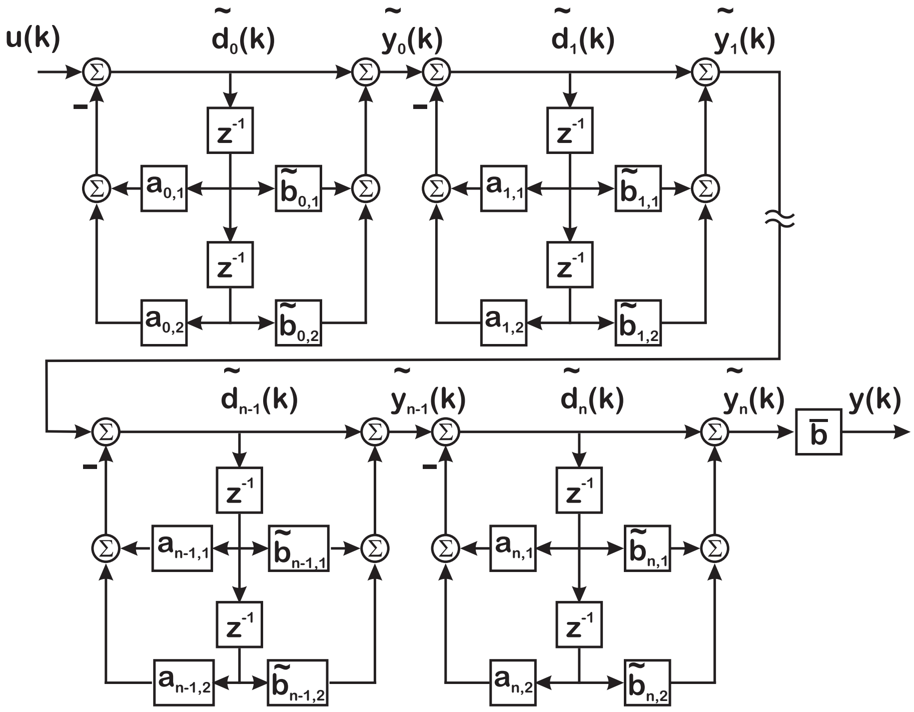


Figure 10.31: The updated biquad cascade, with factored out $b_0$ terms. (Repeat of Figure 6.20.)

get to the proper output states. This form results in a block upper triangular state transition matrix [3]. If one were to mistakenly substitute $1/s$ for $z^{-1}$, one would end up with the continuous time structure of Figure 10.33. Furthermore, if one were to discretize the structure of Figure 10.33 one biquad at a time, then one would end up with the structure of Figure 10.32, with the added advantage that the signals at the outputs of the biquads would correspond between the analog and digital versions. Figure 10.34 demonstrates this with a 3-biquad system, where one of the biquads implements an LPF [5]. Note the tight correspondence between the outputs of both analog and digital biquads. The roll up in phase in the DT plot is due to mapping the LPFs continuous time zeros at $s = -\infty$ to $z = -1$.

### 10.14.6 Rigid Body Modes and the Bilinear State-Space (BLSS) Structure

In this section, we repeat and summarize the discussion of Section 9.28, again with a focus on computation.

For all the advantages of the BSS for flexible modes, we still need to find some way to not only represent

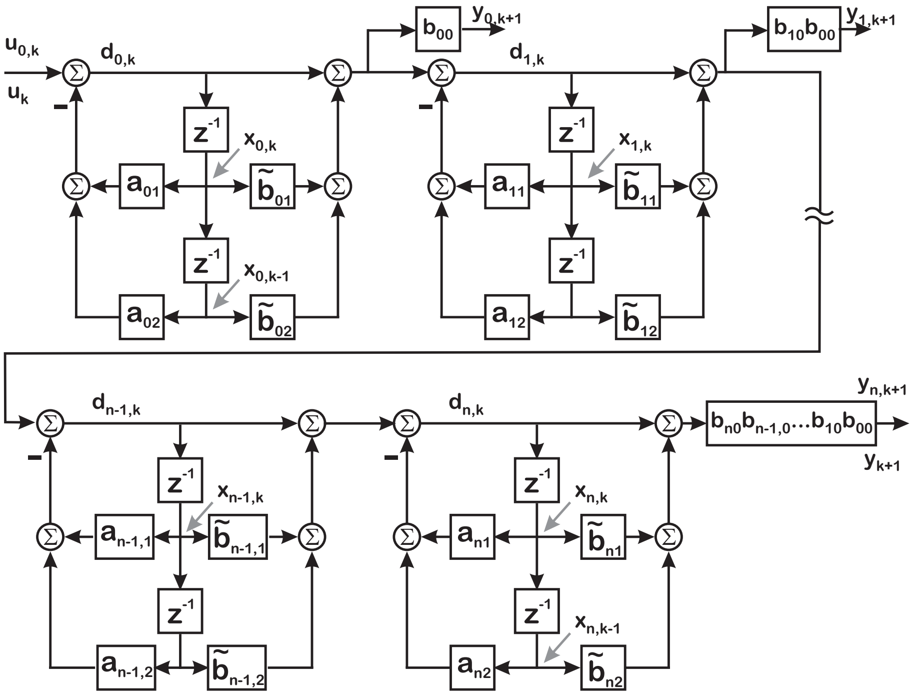


Figure 10.32: The updated discrete biquad cascade, with factored out $b_{i,0}$ terms and scaling the output of each block.

rigid body modes, but also to have the internal states of those structures correspond to the internal states of the rigid body, e.g. velocity and position. Furthermore, we would also like that rigid body state-space structure to have an equivalence between the continuous and discrete time forms. This is accomplished via the Bilinear State-Space (BLSS) structure [5]. Figures 10.35 and 10.36 show continuous and discrete versions of the Bilinear State-Space form (BLSS) [5] which accomplishes that, and can be combined with a cascade of biquads into one overall state-space structure. This is shown in the simulation of Figure 10.37. After all, it is a bit embarrassing to go through all the mathematical machinery of state space, and not be able to access the discrete states for both position and velocity.

## 10.15 Example Bandwidth Ranges, Applications, and Platforms

In this as in many other sections, there are general statements based upon where the state of the art of technology ("Moore's law"[42]) is at the time of the writing. We expect that the specific numbers and

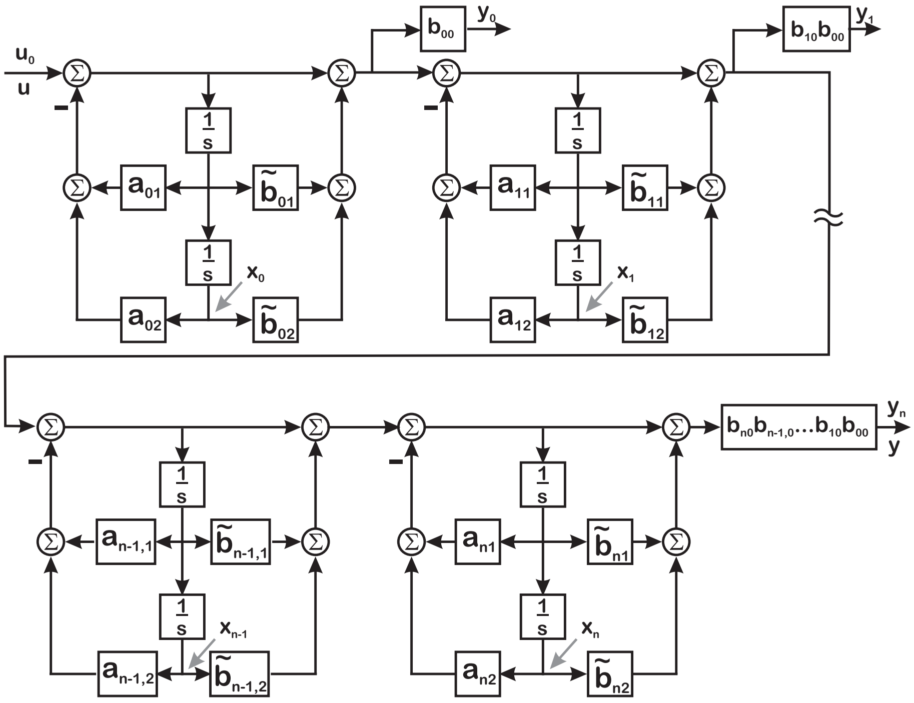


Figure 10.33: The analog biquad cascade, with factored out $b_{i,0}$ terms and scaling the output of each block. This is completely analogous to the digital form of Figure 10.32. (Repeat of Figure 9.25.)

ranges will change over time, but the basic idea of what defines a range should remain. The best advice here seems to be to paraphrase Sun Tzu [333]: "Know your time constants, and know your dynamics, and you can close 100 loops without disaster."

We will provide some platform examples, in a direction of ever increasing speed. Of common interest is the tradeoff between preemption – being able to run multiple tasks by multiplexing – and timing certainty. As we move down the list, we move towards more and more dedicated hardware to a specific computation and less opportunities for preemption. The second trend as we move down the list is that there are a decreasing number of electronic components between the physical system and the electronic computation. The increase in speed comes along with a potential increase in cost and almost always decrease in flexibility. However, the march of technology means that as the years pass, more physical systems with faster time constants move up these layers. One can look across the different computation technologies as follows:

**On a Linux Thread:** Linux is a free, customizable operating system. Among the many variants are complex versions for servers and simple versions that run much of the embedded systems in the world.

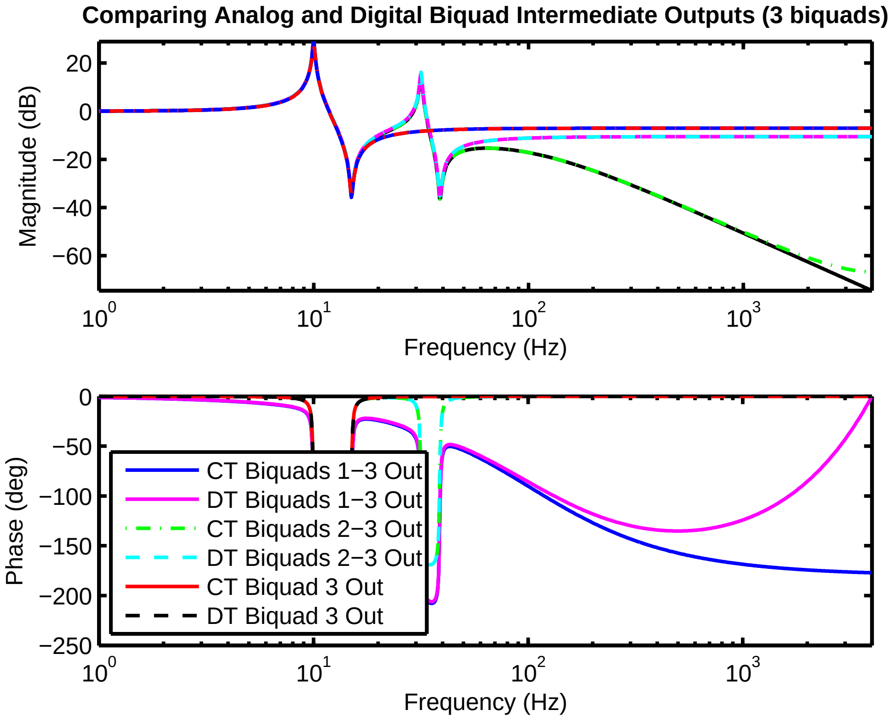


Figure 10.34: BSS with three biquads including a low-pass filter in biquad 1. This plot compares the Bode responses of the individual CT and DT biquad sections. The outputs of biquad 3 and biquad 2 show the magnitude and phase flattening out at high frequency (due to the matched number of poles and zeros). Once the response of biquad 1 is added in, we see the low pass roll off. At each biquad output, the match between continuous and discrete responses is incredibly close, a unique and useful feature of this structure. (Repeat of Figure 9.38.

It is not uncommon to select a Linux thread to run some of the slower real-time applications. Because a real-time Linux thread can be given higher priority than Non-Real-Time threads, this can reduce the delay and jitter to acceptable levels for relatively slow applications.

**On a Real-Time Operating System (RTOS):** An RTOS is the next level up in capability. This is a compromise between maintaining a preemptive operating system with its ability to manage memory, communications, and scheduling, with the priority of real-time tasks. Many applications that use Digital Signal Processing (DSP) chips will fall into this area.

**On a bare metal (minimal/no OS) chip:** In this case, our need for precise control of timing has overridden our desire for the convenience of an operating system. Many of these processors are relatively small and made for running a single process with minimal interruptions.

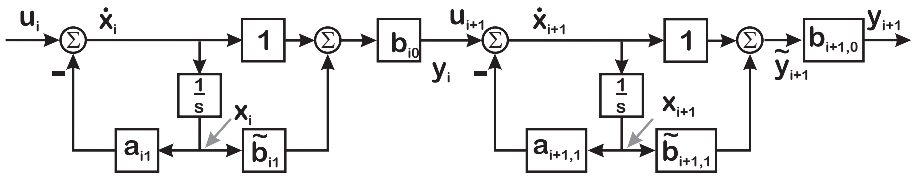


Figure 10.35: Continuous time bilinear state-space (CT-BLSS) form. (Repeat of Figure 9.29.)

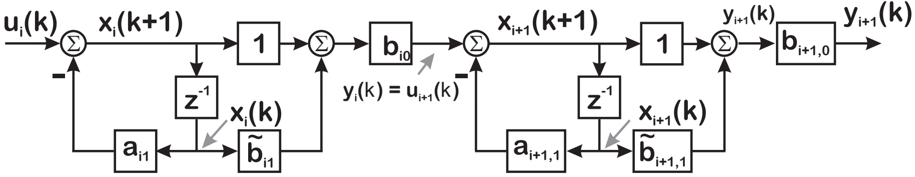


Figure 10.36: Discrete time bilinear state-space form (DT-BLSS). (Repeat of Figure 9.31.)

**On an FPGA or other PL:** Programmable logic (PL) started as a way of prototyping custom integrated circuits (custom ICs) or of generating the glue logic that tied many processing chips together. As the capabilities have grown, so have the PL chips and market. What FPGAs give is a chance to generate custom hardware processors for specific mathematical tasks. Instead of multiplexing these tasks in time as one would have to do on a single processor, they are multiplexed in space on the FPGA . One of the key features that of algorithms that makes this highly valuable is that many small algorithms are relatively simple and can be implemented with relatively simple logic. Thus, we are not wasting the processing power of a large processor by having it switch to these tasks.

**On custom digital or mixed signal IC:** A custom chip can include mixed analog and digital signals without incurring any of the (albeit small) overhead of an FPGA chip. The cost of laying out a custom chip means that this solution is only feasible for either high-cost applications or for mass market applications. In either case, something has to make up the cost of chip layout to gain that extra speed.

**In analog electronics:** The fastest speed systems often require us to give up on digital methods in the Hard-Real-Time layer. With this move to fully analog implementation comes the loss of flexibility, reproducibility, and updatability that are a key advantage of digital methods.

An alternate view starts with the general speed ranges that we can group by sample rates. Note that the edges often overlap.

**Low End Speed $f_S \leq 1Hz$:** The typical applications include thermal systems, pressure control, biological reactors, and chemical process control. Such slow systems can often be handled as a Linux

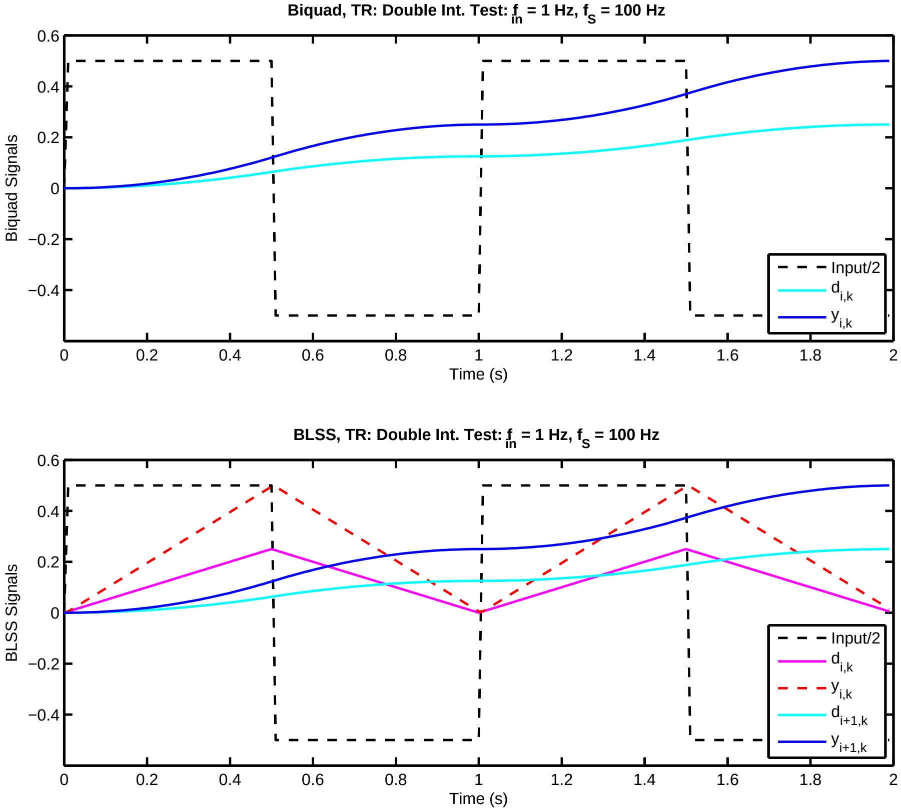


Figure 10.37: Double integrator with square wave input. Implemented using a trapezoidal rule equivalent biquad (top) and BLSS (bottom). (Repeat of Figure 9.40.)

thread with the API approach diagrammed on the left of Figure 10.26 or using the advanced tools method diagrammed on the right of Figure 10.26.

**Next Level** $1Hz \leq f_S \leq 100Hz$**:** In this zone we currently have rigid systems; typically medium to large mechatronic systems, or small toy class systems. We might find personal robots in this region. While the sample rates are pushing the boundaries, one might still find some of these handled by Linux threads. The more complex or safety critical systems might run on an RTOS, and the chips involved might move from low end processors to DSP chips.

**Next Level** $10Hz \leq f_S \leq 50kHz$**:** Typical applications here might include fast rigid body and/or mechatronic systems, such as motion stages, fast robotics, flight, safety, automotive, and disk storage. These would be usually handled with an RTOS or bare medal computing environment. For hardware, we would have moved away from conventional or inexpensive processors fully in the range of DSP and/or FPGA chips.

**The Need for Speed $50kHz \leq f_S \leq 50MHz$:** Typical applications requiring these sample rates might include high-end instrumentation, mid-level electronic test, and high-speed small mechatronics, such as atomic force microscopes (AFMs). These might use DSP chips at the lower end FPGA implementations at the higher end and run with minimal operating system interference.

## 10.16 Business Models and Bandwidth

The one-line version of this section reads [31], "You can't put a $200 FPGA into a $50 disk drive. Sad, but true."

Control theory might seem unified, but the space to implement is dramatically varied. As was described in Section 10.15, the physical system time constants are a main determinant of the required sampling rates, and these in-turn affect the version of the Three-Layer-Model from Section 10.12.1 that we will program against.

For very slow applications (e.g. pressure, temperature, or chemical and biological process control) the computer is so much faster than the process dynamics that we can forget about latency inside the digital unit. This is where lots of compute intensive learning algorithms are first tried. A great example of this is Model Predictive Control [29, 334], which finds a natural home performing optimization between the relatively-slow sample instants of chemical process control systems. At the same time, the types of dynamic structures for which the control system must compensate are different. Chemical engineers rarely think about resonances, but are keenly aware of transport delay and the limitations on their sensors and actuators. Similarly, the long time constants mean that frequency domain measurements are almost meaningless to this group. What this means is that the raw processing costs will be relatively low. Such bandwidth requirements can often be met with the API method diagrammed on the left of Figure 10.26. Where the money will be spent is on the input and output signal chains, where – depending upon the system – the environment in which those electronics operate often determine the cost.

At the other end of the spectrum are intense, expensive applications (e.g. fighter planes, space launch and spacecraft, wafer scanners). For these systems, the cost of processing is a tiny part of the machine cost. For these types of systems, there might be substantial numbers of lightly damped dynamics, as well as substantial instrumentation challenges. Time constants may be short, prompting higher sample rates and more stringent computational requirements. However for these systems, the engineering teams are large and (relatively speaking) resources are flush. Latency, noise, performance limits all matter, but the proverbial checkbook is open. These systems are characterized in part by being so expensive that each device can be tuned by a team of engineers. Such systems afford high-end, high-cost processing, and plenty of engineers to design and program each of the layers in Figure 10.27.

Perhaps the largest price-performance demands come from consumer level systems. This may present the largest challenge and opportunity. The prevalence of feedback-based devices in our everyday use require a lot of performance in relatively inexpensive processing solutions. These must be increasingly self-tuning and self-diagnostic. The low unit cost mandates that we cannot afford to have engineers touch every device. That being said, their penetration to the public is far more visible, so reliability is critical. The path for these computation systems is usually to start with more complex, powerful chips and algorithms in the early test phases, and then port the simplified versions down into the low-cost hardware. While the Hard-Real-Time layer is made as efficient as possible, it is common in these systems to thin out the top two layers to reduce cost. After all, when it is cheaper to replace a device than to diagnose and fix it, this makes economic sense. (We are ignoring the more complete accounting costs on the environment due to throwing away, rather than repairing or recycling broken devices. We do not endorse this incomplete accounting; we simply view it as a current – and flawed – business practice for many operations.)

Consumer level devices where the feedback loop is used as a selling point include fuzzy logic washing machines and rice cookers. The two salient features of these are that the system dynamics are relatively simple and benign and that the fuzzy logic control is advertised as a feature. They are highly unlikely to do damage to anyone or anything even with a flawed or failing feedback loop. Such systems feature hard limits on internal temperature, cooking or cleaning time, and the like, so that even if the feedback fails, the system will shut down with nothing more than a ruined dinner. If one views fuzzy control as more of a control implementation method than a control design method [335], then we can realize that a key feature of these devices was the implementation of feedback on a benign but improvable process.

Few consumers are aware of the key role of feedback in their hard disk drives and optical disks [287, 336, 337, 338]. As the use of optical disks for consumers gets replaced by streaming media and hard disk drives move out of personal computers and into server farms, there is nothing to indicate that this trend will reverse. At the same time, devices that visibly depend upon feedback for their are rapidly becoming prevalent in our society, from all types of robots, to drones, to self-driving modes in vehicles, to self-driving vehicles themselves. These are not systems with benign dynamics and so the proper implementation of feedback on a low-cost embedded platform is critical. As control engineers we need to understand both how to to fit our algorithms into such inexpensive platforms and how to justify our push for some head space in those systems for improved diagnostics and code revisions. The team will likely only have one or two engineers with any knowledge of feedback. Being able to communicate design considerations to everyone from business types, to chemists, to software designers is a critical skill.

Finally, with all the discussion of machine learning (ML) and artificial intelligence (AI) in the public consciousness, we can neither shy away from these discussions, turn off our connection to the laws of physics, nor pout in the corner because we are not getting the same attention. Anya Tsalenko, an expert in machine learning, big data, and artificial intelligence at Agilent Labs often points out that the main advancement between the neural network methods of the 1980s and 1990s and today are a massive increase in the amount of training data and computer power available now. What was once a parallel scheme run on a single Intel processor is now a massive parallel scheme tuned with terabytes of data on GPUs (Graphics Processing Units) and implemented in parallel on FPGA s. Still, the demonstrations all seem to focus on systems for which the dynamics are orders of magnitude slower than the processing. This is best expressed in a high school mathematical word problem:

Q: If Train A leaves Chicago traveling East at 50 MPH and Train B leaves Buffalo traveling west at 60 MPH . . .

What is the probability that the AI demonstration will involve image classification?

The answer is almost always 1. The answer for our field is that we need to be there with fundamental systems thinking when ML/AI systems try to make something physical move.

## 10.17 Chapter Summary and Context

This chapter has been all about how we think about computation for control systems. Most of this has focused on the real-time portion of the loop, the part that needs to meet nature's timing constraints. However, once we enter this real-time world, we realize that we have to consider all the signal chains coming into and going out of the processing engine. Those necessarily involve analog electronics and physical computation. We need to consider 4 main blocks of computation in a feedback system: the plant's "computation", the input signal chain, the output signal chain, and the computer itself.

Each of these is a potential source of delay, noise, and jitter. They can each wreck the performance of even the best control algorithm. Addressing them is each part of successful control design.

Time delay through the system, which we closely associate with phase margin. While this is not a major issue for a pure filtering context, it is fundamental to a feedback context.

Our awareness of Bode's integral theorem discussed in Chapter 7 and how that makes us sensitive to any noise that enters the loop was also brought up. Understanding the role of real-time computation for different time constant problems is critical. The three-layer computer model helps us understand how to program for different parts of the loop.

Finally, none of this can be separated from the business model versus bandwidth tradeoffs.

# Chapter 11

# Closing Thoughts

My father was an organic chemist, and once, when I was about 5 years old, I asked him why he had a Doctor of Philosophy degree if he never actually did any philosophy. (Of course, he did the usual dad stuff, but I was thinking tweed jackets with elbow patches here.) Well, looking back on a lot of the stuff in this book and the workshop, I realize I have been dropping a lot of control systems philosophy bombs along the way. I am going to own that whole philosophy part of the Ph.D., so in this chapter I will try to summarize some of the control systems philosophy pushed throughout the book. Hopefully, that will give a bit more coherence to what is in here. If not, it's okay. I like saying them.

## 11.1 What Different Perspectives Want

I am going to look back here and offer one more view of what two different control perspectives, the academic/theoretical (AT) and the implementation/industrial (II) are looking for. Hopefully, this view has been taught in what came before this and now it's just a summary. The reader should understand that in order to present these two perspectives I will make very broad generalizations (as I did in Chapter 2) that won't apply to any one individual exactly. Call it fuzzy reasoning. The two perspectives I will consider here are textbook and practical control (or academic/theoretical (AT) and industrial/implementation (II) in Chapter 2).

Broadly stated, textbook (academic/theoretical (AT)) control wants to:

- Take a system model that describes something in the real world.
- Find properties of that model that allow for a new and improved method of control.
- Analyze the new method on the model and show it is "optimal" on some metric.
- Simulate the model in MATLAB /**Simulink**(or something equivalent) and make it "real" by adding in Additive White Gaussian Noise (AWGN) and/or some sort of bounded uncertainty.
- Say it will work in practice.
- Hopefully get a paper out of it.

Broadly stated, practical (implementation/industrial (II) ) control wants to:

- Hook some real-time controller box into a test system or prototype.
- Push a button that generates excellent measurements of the system and produce model options.
- Push another button that generates an accurate, robust, parametric model for controller design, and gives the user a design choices menu.
- Push the design button that produces a high performance, robust control design and projects its behavior against the original measurement.
- Push one more button to transfer the design into a low overhead, high sample rate, real-time control system.
- Tell their manager they did something innovative and "optimal" only to have their manager ask if four buttons are really necessary.

I do not want to abandon either of these views. Instead, I want to keep alternating our perspective going from one side to the other so that we can get to a real solution somewhere in the middle. This is where I want them to meet:

- Every button push results in a step that is both physically intuitive and mathematically smart (although not necessarily "optimal").

- Models are heavily measurement driven and measurements can be rapidly iterated with data being passed easily to and from control systems Computer Aided Design (CAD) software. Parametric models for control design can be rapidly and reliably extracted from measurements.
- Implementation choices and trade-offs are reflected back into the system and design model.
- Measurement, model, and design improvements are easily iterated on the experimental system. Designs are easily transferred to the real-time experimental system. Experimental and measurement data are easily transferred back into the CAD program.
- Experiment and design results easily compared to "optimal" model-based projections to see how close implementation is to theoretical best.
- The ability to reflect "non-control" design choices into the model in an intuitive way for co-design with other fields.
- Experiments, models, and designs easily saved in a form that is easy to retrieve, easy to display, and easy to export to many other formats.

The unified approach does not see superiority in either theoretical or practical results, but works to iterate between the two to achieve a practical system that may not be theoretically optimal but is guided by optimization to be excellent. Much of the work to connect these two worlds involves a lot of grunt work in programming, but this programming cannot be done without a system view that stretches from the physics to the web page. Not only that, but the programming has to take the structure of the physical system.

For those perspectives to meet, someone has to be the ambassador, the one who ventures forth into other areas. In the sections that follow, I will try to make the case that it is the person trained in and skilled with a systems perspective that is best equipped to do this. However, we systems engineers cannot accomplish this, cannot fulfill the future that our background allows, unless we are willing to venture beyond the comfortable world of proofs and optimality, into the messy world of physical systems and their limitations. After all, Francis of Assisi wasn't sainted for illuminated texts. He only became important when he ventured out.

In 2008 I was invited down to the University of California at Santa Barbara (UCSB) by Mustafa Khammash. This was my first visit there, although I would make many more in a few years as my oldest son pursued an undergraduate degree in Chemical Engineering there. I did not know Mustafa at the time, and so I told him that I had a few hour long talks already prepared ("in the can" as they say in Hollywood). However, no sooner had I said yes, than I got an email from Professor Karl ström, who spent the winter quarter at UCSB at that time, saying essentially, "Now that you're coming down, you can give a talk to my advanced controls class." Okay, now I needed a new talk, which I did create and later turned into the ACC tutorial paper on "business and bandwidth" [31]. The talks themselves went well, and interaction with one advanced graduate student in Karl's class led me to the desperate but insightful realization that, "Well state-space methods are model based methods and model-based methods require . . . a good model."

After the talk, we had a discussion in his office and I was relaying my growing realization that something had been missing in how we had done stuff over the years. He pointed out to me that in the early days of control, the control engineer knew the entire system, from the physical plant, to the electronics, to whatever processing was being done to implement the control law. They worked on the entire loop. Now (in 2008), he said, most control engineers had a platonic love of control: they wanted to talk about it, write about it, but they didn't want to touch a real one.

It seems that many controls researchers view their role as to hand off wisdom from on high, without actually participating in the messiness of implementation. Unfortunately, this misses the point, because just as many theoretical concepts cannot be truly learned until one has to derive or apply them in a homework problem, many of the issues of the physical world can only be appreciated once one is personally trying to make a real system work. Being there matters. We need to get beyond a platonic love of control and touch real systems.

To have a seat at the table, control engineers need to once again become the Jacks and Jills of All Trades that characterized the early days of the field. We need to be comfortable with being dilettantes in related areas so that we can be ambassadors for our field to our fellow engineers, scientists, and, yes, even to managers and MBAs. I get it, the real stuff is messy and involves a lot more than just sitting at your computer. Then again, making a control system to work in the real world is a lot more fun than watching videos of control systems on YouTube.

## 11.2 Real Control Design Work Cannot Be Separated from Implementation Details

It would be nice to think that theory and optimization can exist and make progress on their own, but separation from actual problems creates another issue of relevance: can disconnected math still be relevant? Of course, there will always be examples of old theory that has been rediscovered and applied to a particular class of problems that didn't exist when the theory was first proposed. The methods of Lyapunov [339, 340] provide a great example for this. It is my general guess that most researchers would prefer that their work become applicable earlier than 40 years after their deaths [341].

Contrast this with the Bode Lecture of Gunter Stein [151] for which the video went viral during his career and for which the companion paper [1], once coaxed out of him, was awarded the IEEE Control Systems Magazine Best Paper Award. Stein's seminal talk did not reveal any new theory: it taught some forgotten theory and tied it to real world examples of things that went wrong when one did not heed the guidance that Bode's Integral Theorem [158] afforded. Chapter 7 and the PES Pareto methodology would not exist, but for that talk. Of course, these are anecdotes, but Stein's work – by explaining how to apply a particular theory to guide real world designs – had tremendous impact on the field during his career. There are many problems in the world for which a fundamental and intuitive understanding of system theory provides tremendous guidance, but it is up to us to tie this guidance to real problems, up to us to show people who don't know all the theorems how they can be useful. To have such an impact, **real control design methodologies cannot be separated from implementation details**.

In practice, the controller has often been the last step in the design. This can be for a variety of reasons, but none of them mitigate the damage that such an approach can have. Often the project leaders are from different fields and they once "did control." While not to be trivialized, a few questions usually reveal that they wrapped a PI loop around a simple system and got it to work (usually at lower performance than we would think reasonable), and declared victory; that the performance limits achievable via feedback control were whatever caused their system to ring. Unfortunately, until we can discuss their simple solution with them using their nomenclature, we are not in any position to explain to them that substantial improvements can be achieved with the given hardware and science through some attention to system theory. Not having a discussion about how you can improve things because you think that PID control is too basic for your level of expertise is a really foolish reason to be excluded from the conversation.

However, if you've gotten anything from this tome or the workshop it accompanies, you know that when we push for more performance we end up pushing bandwidths, which means pushing actuators, sensors, and timing. To do this without understanding the limits of our models gets us into trouble. To do this when our design space has been already limited by scientists and MBAs is a flawed methodology that we have been reluctant to fight.

To be truly successful, control design needs to be part of the initial design concept. It is at this stage that slight changes in the mechanical or chemical process design, or of the product instrumentation (where the sensors and actuators are placed and how many of them they are) can dramatically affect what can be achieved through the use of control and system theory. We often evaluate the controllability and observability of our state-space models, but this is designing in controllability and observability as the Wright Brothers and the Apollo Program did. Designing these in gives us more than a mathematical assurance that we can somehow get from here to there (and know where we are along the way); it gives us a way to make it a lot easier to get from here to there and a lot easier to check our path.

To do that, to have a seat at that product definition and initial design table, we cannot just do math. We need to understand the entire loop (at least a little bit of all the aspects). It is certain that someone will have overall responsibility for the entire design and we cannot know everything, but we need to be curious about and become conversant in most of the aspects of the system. We need to be dilettantes in the other areas of the design, so we can have the experts add the tweaks that make the control design space far more manageable. The buzzword for this is co-design, but we need to think more plainly and understand that this is just good engineering.

Stepping up to this does not come instantly and it does not come without work to understand other fields we might not have studied since college. Still, it can save a lot of heartache from and performance for a project. A few examples are presented here simply to illustrate the concept.

A lot of what I have discussed in this tome has its origins in trying to solve control problems related to high speed Atomic Force Microscopy on the Agilent Labs AFM Project. To be certain, many of the developments and examples in earlier chapters stem from this project and yet some of the most important results affecting the achievable control performance were not controls related.

At some point in the Agilent AFM project, I started breaking out of my "role." I found out that they ADCs and DACs chosen had huge amounts of pipeline delay, simply because the master circuit designer didn't know that delay was a consideration. Had he known, he could have adjusted the design in a straightforward way. I found out that the analog filtering being done (including anti-alias) made this problem an order of magnitude worse. (We were able to tweak this, although not without giving up on some potential aliasing.) No amount of algorithm redesign was going to take away the fact that my phase margin (and therefore effective bandwidth) had been knocked down by a factor of 10-20.

The lesson I learned from this – and which I still kick myself from not seeing earlier is that we need to understand enough of the circuits to help drive the choice of input and output systems. We need to understand enough of the mechanics to shift the resonances to places where they can be more effectively damped. We need to understand enough of the chemistry or biology to know which processes can moderate (i.e. act as actuators) the overall system.

To have a seat at this table, be a dilettante in areas outside of control, and be an ambassador for our field. As Trevor Noah pointed out in his autobiography, Born a Crime, p. 236, [342]:

> Nelson Mandela once said, "If you talk to a man in a language he understands, that goes to his head. If you talk to him in his language, that goes to his heart." He was so right. When you make the effort to speak someone else's language, even if it's just basic phrases here and there, you are saying to them, "I understand that you have a culture and identity that exists beyond me. I see you as a human being."

When we try to learn about the fields of our co-workers, when we try to speak their language, most (but not all) will appreciate it enough to start listening to us in ours. People who travel around the world know this to be true. We need to apply the same idea to our work.

Understand the computing, know how to write code, but also know when you need to push the professional software types to do something they don't understand. You cannot do this if the only code you write will never get close to the implementation (at least when you want to push performance). There are lots of environments to allow us to forget the implementation. MATLAB has code generation and real-time blocks in **Simulink**. Xilinx and Altera (Intel) providing libraries to let AI Python jocks get their designs into FPGAs without ever understanding one bit of the hardware. For many, many problems, this is enough. **When it fails because of some performance/size/memory/speed limit, you have no prayer of fixing it unless you know something about what it's been doing. That's just engineering.**

I prototype and debug my algorithms in MATLAB . That accelerates the proving out and debugging of algorithms by a factor of ten or more in my estimation. I am also the one to rewrite them in C/C++/C# (the collection of which I refer to as C*). I share the C* data back into Matlab and do side by side comparisons. This helps me debug the C* implementation. When they both produce the same results, I have a pretty good confidence that the code is right. That being said, I have learned something unexpected from this:

- For the first chunk of time, my MATLAB implementation is debugging my C* implementation.
- But eventually, my C* implementation always finds bugs in the MATLAB implementation.
- More importantly, I have code "cred" (credibility) with the professional software engineers, so I get to drive what they do and I get more of their help. (Channel Nelson Mandela and Trevor Noah.)

Some of my co-workers have suggested that I could spend more time writing algorithms if I left the end programming to a pure software type, but I have resisted that. The control engineer, the algorithm person knows what they need the software to do, not just for a set of numbers, but from a fundamental insight of what the behavior should be. Cut that tie to the actual implementation and the control engineer has had their direct measurement taken away, which weakens their ability to correct their code with measurement based feedback. Put another way, I know what the code is supposed to do but I also need to fix my algorithm with a direct understanding of how it will function. Otherwise, I do not have direct feedback to fix my algorithms.

When I suggest an anti-alias op-amp circuit to the analog designers on my team, they will instantly be amused by the crude simplicity of the circuit, but all the pieces I left out that are needed to make it practical. They will also see what I am trying to get at and suggest another design that should do the same thing and a design conversation ensues between what the loop designer wants and the circuit designer knows they can reasonably provide. I've spoken to them in their language and now I can give them the design constraints and objectives in mine. Sure, I feel stupid for my simplistic circuit diagram. It will pass.

**At the same time, understand enough of the system to know when the problem has nothing to do with software or control laws** In my undergraduate days, I was part of the Co-Op Program at Clemson, spending four semesters working at Milliken & Company, a textile manufacturer based out of Spartanbrg, SC. During that time span, one of the software developers in our area left to join a company making textile machinery. I occasionally caught up with him and found out about how he had a rough first year trying to get the embedded platform to work on this particular machine. On one of my semesters back at Clemson, I found out that one of my Electrical Engineering professors was also consulting for that same company. Upon finding out that I knew this software engineer, this generally reserved and under appreciated professor launched into a rant about the issues at that company and how the problem with the software was that it wasn't a software problem. Apparently, the vibrations of the machine caused the socketed chips on the embedded systems board to wiggle loose, causing intermittent connection problems. There were also issues with moisture getting onto the circuit boards, but as these were outside the expertise of the software designer, he ignored them, and had a very tough time. As a rule, it's hard to fix fundamental hardware problems by writing code and it's hard to debug code on a chip that isn't always connected to the motherboard.

Never be afraid to be a real engineer. Feeling stupid will pass. Missing a good design will stay with you for years. Co-design is the right term. Control/system engineers are the ones best equipped to guide this.

- We look at the whole loop. Few disciplines are built upon that.
- We are in a position to be dilettantes about the design of other blocks in the loop.
- Computer Science, for all its benefits, doesn't have a concept of physics or timing.
- Most of the sciences don't have the structural discipline of engineering built in.
- Control spans a lot of different implementation fields. It's your entry pass to being able to ask questions of the experts in those fields.

Understand the business opportunity/tradeoffs and marketing. Georges Clemenceau said "War is too serious a matter to entrust to military men," [316] although the variant of, "War is too important to be left to the generals," is more famous, perhaps because of its use in the movie, Dr. Strangelove [294]. For our purposes, the translation would be that marketing is too important to leave solely in the hands of marketing people. This is not a disparagement of marketing: good marketing involves predicting what future products will meet the previously unknown needs of customers. This is hard; probably a lot harder than good engineering because it is predicting the behavior of humans which have a tendency to be nonlinear and time-varying. Deep respect is due to anyone who can repeatedly execute good marketing. The issue is that to someone who has not learned anything about business models or marketing, the difference between good marketing and bad marketing (the latter one requiring very little work) is virtually unobservable during the crucial first few years of a product's development. Shrewd bad marketers will push unreasonable market demands on the technical team, often gaining a promotion before anyone has been able to see that what they were asking for was not feasible with the physics of the next ten years.

Again, I realized this late in the game on the AFM project. We had a marketing lead that insisted on a certain size of the image, with a particular scan speed and resolution. No one piece of it was unreasonable as one could find examples of each of these pieces being done in research laboratories around the world. This individual had a list of about 100 features that the market demanded, 90% of them being must have, while most of the rest were strong wants. Any question of why a particular item was on the list was met with a response of, "The market demands . . ." After a few years of a fairly smart team working very diligently to make the items on this list reality, I finally did some simple unit conversion calculations (as in freshman chemistry). His "market demands" required circuit speeds that would not be physically possible for another 20 years. I tried to tell the master circuit designer mentioned above what the "market demanded" of his circuits, but his suggestion in response to me is not printable. I then took the same back-of-the-envelope spreadsheet analysis and emailed it to the entire project team. Within a week, the market did not seem to be demanding the physically impossible anymore. Go figure.

Anyone on the project could have done these simple calculations, and yet none of us had. We took what the marketing person told us at face value, never questioning whether we should check for perpetual motion machines on their list of must-haves. Some of these checks are like the so called "idiot lights" on the car dashboard. One doesn't need an MBA to know that we cannot put a $200 FPGA into a device that sells for $50 (sad, but true). The consequences of not doing such simple checks often lead us to algorithms that cannot run on the hardware available. The consequences of doing development without envisioning more available computation are that one needlessly restricts the design space when another one might open up in a year. There is no perfect prediction, no perfect answer, but we as engineers should be as comfortable and competent as any with understanding the limits of our predictions and making engineering tradeoffs.

Willie Mays was a legendary American baseball player from the late 1940s to the early 1970s. He was unique among superstar players of that time, being known for both his hitting and fielding prowess. It was once said of a long ball hit by Mays, "The only man who could have caught that ball just hit it." To have those Willie Mays moments, we have to be willing to play in all aspects of the game.

## 11.3 "Rommel . . . I Read Your Book!"

In the 1970 movie, Patton [343], in which actor George C. Scott would win the Academy Award for Best Actor for his portrayal of American General George S. Patton, there is an initial battle between Rommel's and Patton's troops. As the battle is clearly turning in favor of the Americans, Patton says to himself, "Rommel, you magnificent bastard, I read your book!" While a fictionalized account of what could have happened [344], it's a teachable moment that we can learn from.

When I introduce feedback systems to middle and high school students at STEM Student Workshops at the American Control Conferences (ACCs) I often describe feedback in terms of things that we humans do all the time: adjust the temperature of the water in shower, drive cars, ride bikes. Once they are familiar with the behaviors I am pointing out, I tell them that we are simply trying to teach machines to do the kinds of feedback we do all the time. As such, we do fall into one corner of machine intelligence and machine learning (ML). That being said, anyone taking a broader look at all the hype surrounding machine learning (ML) and artificial intelligence (AI) knows that we are far from the center of attention. Like the unnamed scientist described in Section 11.2, much of the technical world thinks of feedback control as something that they did once. Most of the non-technical world hardly knows that we exist, until there is some disaster in the news related to a flawed feedback system (or a sensor failure in a feedback system leading to a disaster in the news).

When one visits Piazza San Marco (Stain Mark's Square) in Venice, one can marvel at St. Mark's Clock Tower, where the automated bronze figures bang out the hours on the large bell, never giving much thought to the fact that their banging would be meaningless without the feedback provided by the escapement within the mechanism. Similarly, those automated highways, self-driving vehicles, easy-to-fly quadcopters, delivery drones, domestic assistance robots, and virtual assistants all fail almost instantly without pervasive understanding and application – both at the top level and down in individual devices – of the fundamentals of feedback.

Part of this is the "gap" between control theory and the practice of control, something that I have tried to bridge in my career and somewhat in this book. Since the 1960s, the "gap" between theory and practice has been bemoaned by those of us in the controls research community, and yet in that time, the control journals and conferences have become increasingly theoretical. The introduction of new journals that are geared towards applications have not stemmed the tide of the papers in these journals being almost exclusively written by academics.

A saying usually mis-attributed to Einstein [345], but useful nonetheless, is that, "The definition of insanity is doing the same thing over and over again and expecting different results." Even if Einstein had said that (no evidence of that), we would have to weigh in his lack of belief in purely random processes ("God does not play dice with the universe.") and chaos theory. Still, on a macro scale, and with general ideas of time-invariance over the measurement period, we can expect that the same large-scale inputs will yield the same large-scale outputs. It should be no great surprise, then that after 60 years we are still having this discussion about the "gap." I was a member of the IEEE CSS Member Activities Board (MAB) during the 2000s, and I kept asking the other MAB members – all academics – a single question: "Of your Ph.D. students who did not go into academia upon graduation, how many (by number or percentage) were still doing control after five years?" The fact that none of them would ever give me a direct answer – and they repeatedly shot down the idea of doing a survey of controls professors – spoke volumes. This was almost two decades ago and the situation has not improved. As it currently stands, one famous friend at a top engineering school cannot fill their advanced controls classes as all the students are packing into the machine learning (ML) and artificial intelligence (AI) classes. Industry research jobs at a project leadership level are lacking, while data scientists are in high demand. **In a world that increasingly needs physical and operational safety from ever increasing numbers of autonomous measurements, agents, and robots how are we minor players?** It is worth taking a look at the emerging world:

- Autonomy is springing up in isolated/independent systems as well as in large proprietary systems.
- Judging from past industrial booms, there won't be one unified system for many years. This raises a serious question about how we make heterogeneous, distributed, autonomous systems

interact gracefully together. Beyond the nice pictures in presentations, how do we get the low level stuff to communicate and agree to play nice? (My experience with automating and connecting measurements does not fill me with optimism.)

- This involves lots of networking and large efforts in fields generally viewed as Computer Science and network provisioning, but where are the feedback principles?
- While it is true that many large projects include control folks, there is a general feeling that the thought leaders picked to head these efforts are much more likely to be AI folks.
  - AI (artificial intelligence) is now almost exclusively neural networks (NN).
  - Most inference is forward propagation convolutional neural networks.
  - These are the adaptive FIR filters of the AI world.
  - They are applicable to many, many problems but they have no state or memory, and no feedback.
  - Being controls folks we know that they will hit a wall on certain problems: ones that require feedback to operate with any effectiveness and safety.
- Projects depend on connectivity, network layers, understanding large scale distributed feedback, understanding small scale, localized feedback, programming, civil engineering, and some machine learning. Still, the algorithm leads, the so-called thought leaders, are more likely to be folks from AI than what the controls community would call systems engineers.
- You can't really ask an AI expert how feedback affects their models, or to quantify the effects of time delay and how that affects the engineering requirements. You can't ask them basic control concepts such as how integral action features in the system or when it is necessary. (There is integral action in almost all iterative learning.)

Now, those of us who are old enough can remember the over hyping of the first wave of AI, and the second wave fueled by neural networks in the 1980s and 1990s, as well as the time when we were supposed to have a "fuzzy future." I myself have benefited from poking holes in the hype for some of this [335], and while everything I wrote then applies today and I probably saved my company millions of dollars for a dead end not chased, there is something that was missed along the way: that Moore's law would make the massive computational needs of neural network tuning less of an issue, that the small and parallel computations enabled by FPGAs would make implementation of the inference engines for neural networks practical, and that everybody taking pictures with their smart phones would give Apple, Google, Amazon, FaceBook, and who knows who else tons of training data on which to pre-train many layers of their networks. It would appear that AI in the form of neural networks has been enabled in a practical way, and because of the successes that are made possible with massive amounts of computation and optimization (when fed with massive amounts of data), there have been some impressive successes, particularly in visual problems for which the model based methods seemed to struggle. Again, we should not decry problems that can be solved without the benefit of a fundamental understanding of feedback principles, but we should be able to map their boundaries and limits, and explain to the general public why those boundaries exist.

In problems where decent dynamic models are hard to generate, these methods give a chance for success. That being said, as soon as they are being applied to dynamic systems with time constants that are less than days and serious consequences for mis-tuning, we are right back at dynamic systems problems. How do we manage this? How do we establish to the general public that knowledge of fundamental feedback principles is critical to enabling this world of autonomous devices doing our bidding? We need to be those dilettantes that delve into other fields, because while a typical control engineer will have a background that allows them to understand the underlying science of the physical system and the computational problems posed in making them work. Many AI scientists are skilled algorithm developers and programmers, but there is a huge potential market for tools that take AI constructs into hardware implementation automatically because those large groups have no inclination to learn about hardware. The data scientist will have an impressive resume based on statistics, Bayesian and otherwise, but I have yet to meet one that questions the functioning of the circuits and ADCs collecting the small data that makes up their big data. Few of the individuals working with tuning algorithms for neural networks can tell us how they would function in a real-time embedded system.

We know that these are issues, but we are unlikely to be able to make our case if we cannot communicate the importance of fundamental control principles to not only these individuals, but the MBAs and company founders that control the money. We are unlikely to convince this latter group if – when asked to assist or consulting with their in-house control engineers – we are unable to translate what we know into improved outcomes for these engineers. We should consider all of these interactions as auditions for the roles we think we should play.

To take on the larger role, to put the system theory into these systems, we need to do like Patton and “read their books.” It is well enough to dismiss the hype of fuzzy logic and fuzzy control [335], but those engineers actually put feedback loops on rice cookers and washing machines and other consumer products that the controls community ignored. They earned their exposure to the general public because there were devices they enabled that made life better for the general public. We can chide the adaptive filtering world for not needing to be concerned with stability in their adaptive FIR filters [24], but let’s not forget how many devices and technologies that they have enabled. We know they can’t solve tough feedback problems, but there are plenty of problems where the time constants are slow, or feedback isn’t critical or the physical parameters don’t need to be known, where their approach has worked fabulously.

I remember a conversation with a DSP specialist at Xilinx, when I was complaining that they had tremendous tools for synthesizing FIR filters but very few for IIR filters. His response was simply, "Dude, you [control] guys are 5% of our business. Sorry." Rather than feeling like losers, we need to appreciate all the problems that have been addressed by those technologies. Like Patton, we need to read their book. As Nelson Mandela said, we need to speak to them in their language. When we do this, when we take this effort, my bet is that we will find plenty of places where feedback principles are critical. Perhaps more importantly, we will know how to explain this to the ML and AI communities, as well as the MBAs and the general public.

We must also be attuned to another issue often voiced in the media, that this world of machine learning and artificial intelligence, of automation and robotics, will displace millions of jobs. There is no doubt of this, and it can paint a dystopian picture of the future. However, the same arguments were made about the invention of mechanization, in which steam shovels replaced individuals digging ditches by hand. Even the American folk hero, John Henry, who beat the steam drill only to die thereafter [346], could not stop the march of technology. Steam ships fueled by coal burners gave way to ships with internal combustion engines running on bunk oil because the latter could be pumped and did not require strong people shoveling fuel into an open fire box. Digging ditches went from being strong folks with shovels to a device powered by a steam engine (the steam shovel) to the modern backhoe that any homeowner can rent for the weekend.

Each of these technologies was seen as the destroyer of jobs in their day, but history teaches us that as those jobs went away, new (and usually better) ones emerged. The democratization of technology, moving from the few to the many, opens avenues for more humans to be more productive in new ways. A generation ago, there were programs called Control-C and Matrix-X, based on the original FORTRAN public domain version of Matlab. When System Control Technology, the company that created the former, was bought out by British Petroleum, several of the engineers cashed in their stock and spent a year rewriting the underlying code from FORTRAN to C. Those engineers were Jack Little, Steve Bangert, and Cleve Moler. The product they created, PC-Matlab, brought high level control and signal processing CAD to the average engineer's desktop computer. Even the high end product, Pro-Matlab (for workstations) had a unique feature in that the program scripts (m-files) and data (mat-files) could be shared between the two. Those products democratized control and signal processing design for a generation of engineers. What we are talking about here is another democratization of control and system theory, not in a new design tool, but in making that fundamental understanding available to folks from middle school to practicing engineers to the lay public. Today we just know it as MATLAB , and almost all STEM students in college use it. It is so powerful and useful that even the public domain imitation, Octave, is a tremendously useful teaching tool in and of itself.

Another everyday example of democratization of technology is the digital camera. Nobody took pictures of their dinners when the cameras used 35mm chemical process film, cost $10 per roll of 36 exposures and another $10 and two weeks to get developed. It was the ubiquity of camera phones, of the free digital film (memory), and of the instant access to easy to use applications that made everyone an amateur food photographer.

Even the people most fearful of being replaced, the driver of delivery vehicles, trucks, buses, and even airplanes, owe their current careers to a technology that was not available 130 years ago. The advent of the internal combustion engine has created a lot of secondary problems of pollution and global warming, but it replaced the much dirtier steam engine for large applications and made smaller vehicles practical. The automobile which was practically unheard of at the turn of the twentieth century was a means to individual liberation by the middle of that century, as explained so beautifully in Tom Wolfe's **Last American Hero** [347]:

> To a great many good old boys a hot car was a symbol of heating up life itself. The war! Money even for country boys! And the money bought cars. In California they suddenly found kids of all sorts involved in vast drag-racing orgies and couldn't figure out what was going on. But in the South the mania for cars was even more intense, although much less publicized. To millions of good old boys, and girls, the automobile represented not only liberation from what was still pretty much a land-bound form of social organization but also a great leap forward into twentieth-century glamor, an idea that was being dinned in on the South like everywhere else. It got so that one of the typical rural sights, in addition to the red rooster, the gray split-rail fence, the Edgeworth Tobacco sign and the rusted-out harrow, one of the typical rural sights would be . . . you would be driving along the dirt roads and there beside the house would be an automobile up on blocks or something, with a rope over the tree for hoisting up the motor or some other heavy part, and a couple of good old boys would be practically disappearing into its innards, from below and from above, draped over the side under the hood. It got so that on Sundays there wouldn't be a safe straight stretch of road in the county, because so many wild country boys would be out racing or just raising hell on the roads. A lot of other kids, who weren't basically wild, would be driving like hell every morning and every night, driving to jobs perhaps thirty or forty miles away, jobs that were available only because of automobiles. In the morning they would be driving through the dapple shadows like madmen.

These were not people with advanced education, but 50 years into its existence, the car had become regular enough, understandable enough, so that individuals with an interest but little formal education could be automobile mechanics. This technology liberated them from the land, allowing them access jobs that had been out of reach a decade before. We should not fault those who worry that some "new technology" (in this case automation and AI) will replace their old "new technology." We should transition them and ourselves to the new jobs that these technologies will enable. In every step, whether it is the steam engine, the automobile, or an advanced CAD tool that could run on underpowered personal computers, the democratization of technology moves us all upstream, so long as we are willing to teach more and more people how to use that technology.

The challenge with the new technologies are never truly about adoption or eventual benefit to society, but about the responsibility to educate the public so that the jobs lost by some technological advance are replaced by newer and better opportunities enabled by that advance. Those of us who create new technologies have a social responsibility not just to make those technologies useful, but to make them tools for greater democratization and greater economic freedom.

## 11.4 You Said You Were a Doctor of Philosophy

An understanding of industrial research is not something that happens instantaneously, but over years of practice. My first job at HP Labs was to do MIMO control on an optical disk system. Having been weaned on MATLAB throughout graduate school, I decided that I wanted to be able to dump MIMO designs from MATLAB direction into my real time DSP system. It took 15 months of tool building to make this work, so that I could start really doing my original job. At the time, it seemed as if this tool building time might have been wasted, but it resulted in a sophisticated real time system that was I used for research over the next seven years. The difficulty of having to implement a system to translate my algorithmic work into something useful was a first insight into the differences between the academic and industrial worlds. This insight was broadened when I was about seven years out of graduate school. In about 1994, two events happened that led me to a much fuller understanding of the philosophical differences between academic and industrial research.

During that year, I was doing research on the control of hard disk drives at Hewlett-Packard Labs. I got a phone call from a Ph.D. student at an East Coast school. (If I ever find out who it was, I will apologize for what happened next.) The graduate student told me they were finishing their Ph.D. on this and that subject and then said, "I am very interested in working on optimal control." Almost reflexively, the words leaped out of my mouth, "Me, too, pal."

While I feel bad that this might have sounded callous, the difference in points of view was right on. The graduate student had a tool that they wanted to apply to do whatever amazing things the tool could do. I, as an industrial researcher, had a tool bucket and a problem to solve. While it would have been nice to be able to concentrate on one particular tool, I have found that it is more important to understand how to select the right tool to make headway on the physical problem. While successful academic researchers often chase a particular algorithm, successful industrial researchers rarely have that luxury. The student is stuck following Hanlon's Razor, "When all you have is a hammer, every problem looks like a nail." The industrial guy is handed something that with high probability, bears no resemblance to a nail.

The second thing that happened is that some friends and I attended a martial arts seminar on Brazilian Jiu Jitsu taught by Rickson Gracie. Those who know about mixed martial arts competition know him as one of the all time greats in that sport. In person, he is physically imposing. And yet, techniques aside, he made two philosophical statements about competition against an opponent [348]:

- "You can't do what you want. You must do what they give you."
- "The more you want to use this stuff for the real thing, the more you need your opponent's reaction to help you."

Now, coming from such a physically imposing person, this made us all think, "Well if he has to react to each situation, what do the rest of us have to do?" For me, though, I realized that those same thoughts could be rewritten for industrial engineering research:

- You can't do what you want. You must solve the problem they give you.
- The more you want to use this stuff (e.g. control theory) for the real thing, the more you need your problem's characteristics to help you.

Furthermore, it is important to look up from our algorithms and realize the truth in the words of Star Trek's Lt. Montgomery Scott (Scotty) "I can't change the laws of physics![349]" This fundamental realization that no algorithm can make the physics of the problem disappear, is key to making progress on applying control to physical problems. Whenever anyone says, "You don't need to understand the problem; X will take care of it," alarms should go off in the listener's head. Hogwarts [350], The Force [351], perpetual motion machines, warp speed [352], etc. are not in evidence on this planet. More importantly, knowing when someone is relying on one of those is useful. Put in today's terms, magical thinking doesn't end pandemics; listening to science and responsible scientists does.

So, what use are algorithms in real problems? Well, like a surfer on a big wave, good algorithms ride the physics of the problem, rather than trying to change them. The problems determine the control algorithm. No one "hammer" can solve them all. However, the prepared engineer is most likely to find the right tool, especially if they are agnostic in their selection. By taking this philosophy, they are more likely to find the fracture points in a real problem: those hidden places where significant progress can be made. One of the hardest lessons for a highly skilled controls person to learn and accept is that often, the best solution to the servo problem often has nothing to do with control. Like Han Solo, we are not looking for a mystical energy field, but applying our simple tricks and nonsense [10][1] . And like Pope John Paul I, once we realize what algorithm can help with a particular problem, we always wish we had studied it harder [353][2].

In the book and the movie, "A Bridge Too Far," [8, 9] an American paratrooper group is on one side of the bridge at Nijmegen. The general in command comes up to one of his colonels and asks him, "What's the best way to take a bridge?" The response, "Both ends at once." That's when the colonel finds out he is leading his men across the river to assault the far end of the bridge.

The point of this workshop was not to teach control theory anew, but to show how we can make our physical control systems (moving metal and sloshing goop) better by attacking the bridge from both sides, looking at both the theory that helps explain most of what is going on and the practical limitations that we must deal with in any real problems. I have started calling this group the Pareto Theory, that 20% of the theory that gets us 80% of the understanding of what is going on. Like a wrestler's or judoka's three favorite moves, we keep coming back to these Pareto Theories to give us insight. Without them we would be blind, but without trying to apply them to understanding physical problems we would have 20-20 vision with no visual reference points. We need both.

Throughout this workshop, we have seen a lot of individual problems, loop components, side issues, etc. that can limit how we design our control loops. We have analog designers making circuits, programmers writing code, mechanical engineers designing structures, scientists wanting to make measurements, digital designers wanting to work with logic. Although engineers like to tackle these problems in isolation, and managers like having an individual who "owns" one aspect of the project, all of the above need to be tackled from a systems point of view. The systems/control engineer need not be an expert in all areas, but they are better off when they are conversant in all these areas.

Only someone with a systems view can balance the different design constraints. However, we cannot do that simply by putting some equations on a white board (or worse in a PowerPoint presentation). We have to be the craftspeople that get into the lab and measure things, that go and ask the scientists about the process, that get advice from the professional programmers, that have our op-amp diagrams

[1] "There's no mystical energy field that controls my destiny. It's all a bunch of simple tricks and nonsense. – Han Solo

[2] "If someone had told me I would be Pope one day, I would have studied harder." – Pope John Paul I

laughed at by the circuits experts. If we want to call ourselves systems engineers, then we cannot be afraid to walk around the entire system. And if we want someone to pay for it, be it a company or a government funding agency, we had better be prepared to share this system view with people who are smart but not trained in our field.

Perhaps one more theme can be expressed in the mantra I have been using for a few years now, **Optimality sucks.[3].** Let me explain: Optimality, as we think of it, is based on a cost criterion applied to a limited model of the system. This limited model is by definition always flawed when describing the real world. The net effect of pushing for optimality then is often to cause the design to fail when applied to anything real. On the other hand, knowing optimal conditions for a model, and knowing how closely that model describes reality allows us to get excellent control. This is why on real physical systems, optimality conditions are, as Captain Barbossa would say, "more of what you'd call 'guidelines' than actual rules."[354] Thus, the above mantra, Version 2.0, would read **Optimality sucks – but excellence rocks[4].** Still, as Bill and Ted might say [355], "Being excellent is not bad."

If there is anything that I have learned on this journey it is that lack of analysis and optimality doesn't stop most people from working on "control systems." In fact, the explosion of cheap sensors and actuators, as well as inexpensive computation platforms such as the Raspberry Pi and the MicroZed mean that we have witnessed only the tip of the iceberg. Guidance from the control community would be welcomed, but it has to be in clear, physically intuitive terms which are extensions of what they are doing now, not wholesale replacement.

I have never been a fan of the KISS (Keep It Simple, Stupid) acronym because it implied the inability to do complex things, either from the speaker or the listener. It also implies that one or both lack intelligence. However, it is my belief that we often make complex things overly intimidating and even the brightest of us are keenly aware of the number of times we have done boneheaded things. Hence, **I would like to propose the KICK acronym, which stands for Keep It Clear, Knucklehead.** We need to keep our explanations clear and physical, as the folks listening are our customers. And we need to make sure that the advanced methods provide at least as much bang for the computational buck as the simple methods.

We must also internalize the fact that **methods that kept working in practice did so for one or more fundamental reasons.** The advanced control background that caused me to look for those underlying factors revealed a lot that, in retrospect, seems quite intuitive. Furthermore, if there is one absolute take away from this work, it is that putting in the work up front to make high fidelity measurements

[3] ©that.

[4] ©that, too.

something that is repeatable and easy, whether in the design stage or in the operation of the system, pays dividends far beyond many of our most sophisticated optimization methods. A long time ago, Gene Franklin quipped to me, “Well, you can only control as well as you can measure.” Likewise, you can only model what you can accurately measure.

**There are tremendous benefits of model-based methods.** They promise a close tie to the physical dynamic equations, something that has been restored with the connection between analog and digital BSS models. They appeared to give a systematic way to handle MIMO systems. It seems, though that the intuition preserving approach may yield benefits there. Measuring a $2 \times 2$ mechatronic system still requires input-output FRFs of the four SISO systems and only after those individual systems have been curve fit, can one really talk about combining them into a more compact model. In such systems, how close do dynamics have to be to be considered common? By what metric will these be evaluated? If the BSS preserves the model precision of a SISO system, how do we choose between close biquad pairs to reduce the model of our MIMO system? These seem like worthwhile directions to pursue. The goal is not to ignore model-based or optimization methods, but to provide a rapprochement between them and the physical intuition of classical methods. The goal is not to ignore practicing engineers and hobbyists for dealing with a too trivial set of mathematics, but to have a set of tools that starts with their intuitive understanding and can be iteratively improved to take care of more and more dynamic features. We might call this approach Optimization Inspired Classical Control.

American football coaching legend Lou Holtz was once asked if the small town he was working in was the end of the world. His response was, “No, but you can see it from here.” Are we at the point where a few button pushes lead to measurement based, mathematically excellent designs? No, but we can see it from here.

I have been fond of quoting Han Solo’s “There’s no mystical energy field that controls my destiny. It’s all a lot of simple tricks and nonsense.”[10] When one is down in the weeds trying to make individual pieces of a complex design work, it may seem like a bunch of simple tricks, nonsense, and sweat. That may be, but if we have some guiding intuition and understanding, if our design is tied closely to our physical system, then when we take a step back, all those simple tricks and nonsense tend to look like a mystical energy field. **Sure, we can’t defeat the laws of physics, but we can read the fine print. Surf’s up.**

# Bibliography

[1] G. Stein, "Respect the unstable," IEEE Control Systems Magazine, vol. 23, no. 4, pp. 12–25, August 2003.

[2] D. Y. Abramovitch, "A tutorial on PES Pareto methods for analysis of noise propagation in feedback loops," in Proceedings of the 2020 IEEE Conference on Control Technology and Applications, IEEE. Montreal, Canada: IEEE, August 2020.

[3] ——, "The discrete time biquad state space structure: Low latency with high numerical fidelity," in Proceedings of the 2015 American Control Conference, AACC. Chicago, IL: IEEE, July 2015, pp. 2813–2818.

[4] ——, "The continuous time biquad state space structure," in Proceedings of the 2015 American Control Conference, AACC. Chicago, IL: IEEE, July 2015, pp. 4168–4173.

[5] ——, "Adding rigid body modes and low-pass filters to the biquad state space and multinotch," in Proceedings of the 2018 American Control Conference, AACC. Milwaukee, WI: IEEE, June 2018, pp. 6024–6030.

[6] J. Burke, Day the Universe Changed. Little Brown & Co, 1986, iSBN: 978-0316116954.

[7] S. Hawking, A Brief History of Time. New York, NY: Bantam Dell Publishing Group, 1988, iSBN: 978-0-553-10953-5.

[8] C. Ryan, A Bridge Too Far. Simon & Schuster, 1966.

[9] W. Goldman, A Bridge Too Far. MGM, 1977.

[10] H. Solo, "Musings on mystical energy fields," in Star Wars, Episode IV, G. Lucas, Ed., Lucasfilm. A galaxy far away from here: 20th Century Fox, 1977.

[11] D. Y. Abramovitch, "Measurements for the design of control systems: Step and frequency response methods," in Presented at Applications Friday during the 2016 American Control Conference, Boston, MA, July 2016.

[12] D. Y. Abramovitch and S. B. Andersson, "Understanding and tuning PID controllers," in Presented at Applications Friday during the 2016 American Control Conference, Boston, MA, July 2016.

[13] K. Ogata, Modern Control Engineering, 3rd ed., ser. Prentice-Hall Instrumentation and Controls Series. Englewood Cliffs, New Jersey: Prentice-Hall, 1970.

[14] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, 5th ed. Upper Saddle River, New Jersey: Prentice Hall, 2006.

[15] G. F. Franklin, J. D. Powell, and M. L. Workman, Digital Control of Dynamic Systems, 3rd ed. Menlo Park, California: Addison Wesley Longman, 1998.

[16] K. J. ström and B. Wittenmark, Computer Controlled Systems, Theory and Design, 2nd ed. Englewood Cliffs, N.J. 07632: Prentice Hall, 1990.

[17] D. Abramovitch, "Thoughts on furthering the control education of practicing engineers," The IEEE Control Systems Magazine, vol. 43, no. 1, February 2023.

[18] D. A. Mindell, Digital Apollo: Human and Machine in Spaceflight. Cambridge, MA: The MIT Press, September 30 2011.

[19] D. Y. Abramovitch, "Trying to keep it real: 25 years of trying to get the stuff I learned in grad school to work on mechatronic systems," in Proceedings of the 2015 Multi-Conference on Systems and Control, IEEE. Sydney, Australia: IEEE, September 2015, pp. 223–250.

[20] B. W. Bequette, Process Control: Modeling, Design, and Simulation. Prentice Hall, 2006.

[21] MathWorks. (2016) Temperature control in a heat exchanger. [Online; accessed June 21, 2016]. [Online]. Available: http://www.mathworks.com/help/control/examples/temperature-control-in-a-heat-exchanger.html

[22] S. Padhee, "Controller design for temperature control of heat exchanger system: Simulation studies," WSEAS Transactions on Systems and Control, vol. 9, pp. 485–491, 2014.

[23] D. Y. Abramovitch, "Using feedback control principles as guiding metaphors for business processes," in Proceedings of the 2022 American Control Conference, AACC. Atlanta, GA: IEEE, June 2022, pp. 3088–3093.

[24] B. Widrow and S. D. Stearns, Adaptive Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, 1985.

[25] D. J. Hand, Dark Data: Why What You Don't Know Matters: Illustrated Edition. Princeton University Press, February 18 2020, iSBN-10: 069118237X ISBN-13: 978-0691182377.

[26] R. N. Bracewell, The Fourier Transform and Its Applications, 2nd ed. New York: McGraw-Hill, 1978.

[27] D. Y. Abramovitch, S. B. Andersson, L. Y. Pao, and G. Schitter, "A tutorial on the mechanisms, dynamics, and control of atomic force microscopes," in Proceedings of the 2007 American Control Conference, AACC. New York, NY: IEEE, July 11–13 2007, pp. 3488–3502.

[28] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle, Process Dynamics and Control, 4th ed. Wiley, 2016.

[29] J. B. Rawlings, "Tutorial overview of model predictive control," IEEE Control Systems Magazine, vol. 20, no. 3, pp. 38–52, June 2000.

[30] D. Y. Abramovitch, "Phase-locked loops: A control centric tutorial," in Proceedings of the 2002 American Control Conference, AACC. Anchorage, AK: IEEE, May 2002.

[31] ——, "A tale of three actuators: How mechanics, business models and position sensing affect different mechatronic servo problems," in Proceedings of the 2009 American Control Conference, AACC. St. Louis, MO: IEEE, June 10-12 2009, pp. 3357–3371.

[32] S. Tzu, The Art of War: Translated and with an Introduction by Samuel B. Griffith. London, Oxford, New York: Oxford University Press, 1971, iSBN 0-19-501476-6.

[33] D. Y. Abramovitch, "The Multinotch, Part II: Extra precision via $\Delta$ coefficients," in Proceedings of the 2015 American Control Conference, AACC. Chicago, IL: IEEE, July 2015, pp. 4137–4142.

[34] D. Abramovitch, T. Hurst, and D. Henze, "An overview of the PES Pareto Method for decomposing baseline noise sources in hard disk position error signals," IEEE Transactions on Magnetics, vol. 34, no. 1, pp. 17–23, January 1998.

[35] ——, "The PES Pareto Method: Uncovering the strata of position error signals in disk drives," in Proceedings of the 1997 American Control Conference, AACC. Albuquerque, NM: IEEE, June 1997, pp. 2888–2895.

[36] T. Hurst, D. Abramovitch, and D. Henze, "Measurements for the PES Pareto Method of identifying contributors to disk drive servo system errors," in Proceedings of the 1997 American Control Conference, AACC. Albuquerque, NM: IEEE, June 1997, pp. 2896–2900.

[37] D. Abramovitch, T. Hurst, and D. Henze, "Decomposition of baseline noise sources in hard disk position error signals using the PES Pareto Method," in Proceedings of the 1997 American Control Conference, AACC. Albuquerque, NM: IEEE, June 1997, pp. 2901–2905.

[38] B. Widrow, "A study of rough amplitude quantization by means of Nyquist sampling theory," IRE Transactions on Circuit Theory, vol. 3, pp. 266–276, 1956.

[39] Wikipedia. (2018) Winston Churchill. [Online; accessed March 28, 2018]. [Online]. Available: https://en.wikipedia.org/wiki/Winston_Churchill#%22We_shall_never_surrender%22

[40] D. Y. Abramovitch, "A unified framework for analog and digital PID controllers," in Proceedings of the 2015 Multi-Conference on Systems and Control, IEEE. Sydney, Australia: IEEE, September 2015, pp. 1492–1497.

[41] L. Carroll, Alice's Adventures in Wonderland. Macmillan, 1898.

[42] Wikipedia. (2022) Moore's law. [On line; accessed September 21, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/Moore's_law

[43] B. Widrow, K. M. Duvall, R. P. Gooch, and W. C. Newman, "Signal cancellation phenomena in adaptive antennas: Causes and cures," IEEE Trans. on AP, vol. 30, pp. 469–478, 1982.

[44] L. Ljung and T. Glad, Modeling of Dynamic Systems. Upper Saddle River, NJ: Prentice Hall, 1994.

[45] L. Ljung, System Identification: Theory for the User, ser. Prentice-Hall Information and System Sciences Series. Englewood Cliffs, New Jersey 07632: Prentice-Hall, 1987.

[46] L. Ljung and T. Söderström, Theory and Practice of Recursive Identification, ser. MIT Press Series in Signal Processing, Optimization, and Control. Cambridge, MA 02142: MIT Press, 1983.

[47] G. C. Goodwin and K. S. Sin, Adaptive Filtering Prediction and Control, ser. Information and Systems Science Series. Englewood Cliffs, N.J. 07632: Prentice-Hall, 1984.

[48] K. J. ström, "Theory and applications of adaptive control – a survey," Automatica, vol. 9, pp. 471–486, 1983, this paper is an expanded and updated version of a plenary lecture given at the 8th IFAC Congress in Kyoto 1981.

[49] L. Ljung, System Identification Toolbox for Use with Matlab, The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760, May 1995, 3rd Printing.

[50] EDN. (2014, November 6) S-parameters basics. [On line; accessed December 3, 2022]. [Online]. Available: https://www.edn.com/s-parameters-basics/

[51] J. R. Ragazzini and G. F. Franklin, Sampled-Data Control Systems. New York, N. Y.: McGraw-Hill Book Company, 1958.

[52] G. F. Franklin and J. D. Powell, Digital Control of Dynamic Systems, 1st ed. Menlo Park, California: Addison-Wesley, 1980.

[53] R. C. Blackham, J. A. Vasil, E. S. Atkinson, and R. W. Potter, "Measurement modes and digital demodulation for a low-frequency analyzer," Hewlett-Packard Journal, vol. 38, no. 1, pp. 17–25, January 1987.

[54] D. Y. Abramovitch, "The Multinotch, Part I: A low latency, high numerical fidelity filter for mechatronic control systems," in Proceedings of the 2015 American Control Conference, AACC. Chicago, IL: IEEE, July 2015, pp. 2161–2166.

[55] B. Wdrow, J. John R. Glover, J. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, J. Eugene Dong, and R. C. Goodlin, "Adaptive noise cancelling: Principles and applications," Proceedings of the IEEE, vol. 63, no. 12, pp. 1692–1716, December 1975.

[56] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, 3rd ed. Menlo Park, California: Addison-Wesley, 1994.

[57] K. Ogata, Modern Control Engineering, 4th ed., ser. Prentice-Hall Instrumentation and Controls Series. Englewood Cliffs, New Jersey: Prentice-Hall, 2001.

[58] R. A. Witte, Electronic Test Instruments: Theory and Applications. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993.

[59] M. S. Holcomb, S. O. Hall, W. S. Tustin, P. J. Burkart, and S. D. Roach, "Design of a mixed-signal oscilloscope," Hewlett-Packard Journal, pp. 13–22, April 1997.

[60] E. Upton and G. H. and, Raspberry Pi User Guide, 3rd ed. John Wiley & Sons, 2014.

[61] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media, 2014.

[62] F. Wang, D. Abramovitch, and G. Franklin, "A method for verifying measurements and models of linear and nonlinear systems," in Proceedings of the 1993 American Control Conference, AACC. San Francisco, CA: IEEE, June 1993, pp. 93–97.

[63] D. Abramovitch, F. Wang, and G. Franklin, "Disk drive pivot nonlinearity modeling Part I: Frequency Domain," in Proceedings of the 1994 American Control Conference, AACC. Baltimore, MD: IEEE, June 1994, pp. 2600–2603.

[64] E. Levy, "Complex-curve fitting," IRE Transactions on Automatic Control, vol. AC-4, pp. 37–43, 1959.

[65] J. L. Adcock, "Curve fitter for pole-zero analysis," Hewlett-Packard Journal, vol. 38, no. 1, pp. 33–37, January 1987.

[66] M. D. Sidman, F. E. DeAngelis, and G. C. Verghese, "Parametric system identification on logarithmic frequency response data," IEEE Transactions on Automatic Control, vol. 36, no. 9, pp. 1065–1070, September 1991.

[67] R. L. Dailey and M. S. Lukich, "MIMO transfer function curve fitting using Chebyshev polynomials," October 1987, presented at the SIAM $35^{\text{th}}$ Anniversary Meeting, Denver, CO.

[68] J. S. Epstein, G. R. Engel, D. R. Hiller, J. Glen L. Purdy, B. C. Hoog, and E. J. Wicklund, "Hardware design for a dynamic signal analyzer," Hewlett-Packard Journal, vol. 35, no. 12, pp. 12–17, December 1984.

[69] Control System Development Using Dynamic Signal Analyzers: Application Note 243-2, Hewlett-Packard, 1984.

[70] HP 3563A Control Systems Analyzer, Hewlett-Packard, 1990.

[71] Multichannel Analysis System Type 3550, Brüel & Kjær, 1984.

[72] The Fundamentals of Signal Analysis: Application Note 243, Hewlett-Packard, 1994.

[73] J. S. Bendat and A. G. Piersol, Random Data: Analysis and Measurement Procedures, 3rd ed., ser. Wiley Series on Probability and Statistics. New York, NY: John Wiley & Sons, 2000.

[74] ——, Engineering Applications of Correlation and Spectral Analysis, 2nd ed. New York, NY: John Wiley & Sons, 1993.

[75] ——, Random Data: Analysis and Measurement Procedures, 2nd ed. New York, NY: John Wiley & Sons, 1986.

[76] D. Abramovitch, "The Banshee Multivariable Workstation: A tool for disk drive servo research," in Proceedings of the ASME Winter Annual Meeting, ASME. Anaheim, CA: ASME, November 1992.

[77] D. Y. Abramovitch and C. P. Taussig, "Determination of open loop responses from closed loop measurements," Hewlett-Packard, Palo Alto, CA USA, United States Patent 5,446,648, August 29 1995.

[78] R. N. Bracewell, The Fourier Transform and Its Applications, 1st ed. New York: McGraw-Hill, 1965.

[79] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing. Englewood Cliffs, N. J.: Prentice Hall, 1970.

[80] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1988.

[81] R. A. Witte, Spectrum and Network Measurements. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993.

[82] H. W. McKinney, "Band-selectable fourier analysis," Hewlett-Packard Journal, vol. 64, pp. 20–24, April 1975.

[83] N. Thrane, "ZOOM-FFT," Brüel & Kjær Technical Review, no. 2, pp. 3–43, 1980.

[84] Wikipedia. (2012) Lock-in amplifier. [Online; accessed December 30, 2014]. [Online]. Available: http://en.wikipedia.org/wiki/Lock-in_amplifier

[85] D. Y. Abramovitch, "Low latency demodulation for atomic force microscopes, Part I: Efficient real-time integration," in Proceedings of the 2011 American Control Conference, AACC. San Francisco, CA: IEEE, June 29–July 1 2011.

[86] ——, "Low latency demodulation for atomic force microscopes, Part II: Efficient calculation of magnitude and phase," in Proceedings of the IFAC 18th World Congress, IFAC. Milan, Italy: IFAC, August 28–September 2 2011.

[87] ——, "Built-in stepped-sine measurements for digital control systems," in Proceedings of the 2015 Multi-Conference on Systems and Control, IEEE. Sydney, Australia: IEEE, September 2015, pp. 145–150.

[88] 7 Series DSP48E1 Slice Users Guide, Ug479 (v1.6) ed., Xilinx, August 7 2013.

[89] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput., vol. 19, no. 2, pp. 297–301, April 1965, reprinted in Digital Signal Processing, ed. L. R. Rabiner and C. M. Rader, pp. 223-227, New York: IEEE Press, 1972.

[90] R. Pintelon and J. Schoukens, System Identification: A Frequency Domain Approach, 1st ed. Piscataway, NJ: IEEE Press, 2001.

[91] T. Oomen, R. van Herpen, S. Quist, M. van de Wal, O. Bosgra, and M. Steinbuch, "Connecting system identification and robust control for next-generation motion control of a wafer stage," IEEE Control Systems Magazine, vol. 22, no. 1, pp. 102–118, January 2014.

[92] J. B. Hoagg, S. L. Lacy, V. Babuška, and D. S. Bernstein, "Sequential multisine excitation signals for system identification of large space structures," in Proceedings of the American Control Conference, AACC. Minneapolis, MN: IEEE, June 2006.

[93] J. Schoukens, R. M. Pintelon, and Y. J. Rolain, "Broadband versus stepped sine FRF measurements," IEEE Transactions on Instrumentation and Measurement, vol. 49, no. 2, pp. 275–278, April 2000.

[94] Wikipedia. (2016) IEEE-488. [Online; accessed June 26, 2016]. [Online]. Available: https://en.wikipedia.org/wiki/IEEE-488

[95] N. Instruments. (2012, August 14) History of GPIB. [Online; accessed June 26, 2016]. [Online]. Available: http://www.ni.com/white-paper/3419/en/

[96] LabView Internet Toolkit Home Page, National Instruments, http://www.natinst.com/labview/internet/, main starting page for information on the LabView software package Internet Developers Toolkit for adding web based functionality to LabView virtual instruments.

[97] Wikipedia. (2016) Standard Commands for Programmable Instruments. [Online; accessed June 26, 2016]. [Online]. Available: https://en.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments

[98] M. Borrello, "Measurements for the design of control systems: Dynamic signal analyzers: The forgotten tool of control systems engineering," in Presented at Applications Friday during the 2016 American Control Conference, Boston, MA, July 2016.

[99] Wikipedia. (2016) Cuneiform script. [Online; accessed June 26, 2016]. [Online]. Available: https://en.wikipedia.org/wiki/Cuneiform_script

[100] nPoint. (2015) Multi-axis stages. [Online; accessed February 5, 2015]. [Online]. Available: http://www.npoint.com/multi-axis-stages

[101] R. Bracewell, The Fourier Transform and Its Applications, 3rd ed., ser. McGraw-Hill Series in Electrical and Computer Engineering. Circuits and Systems. McGraw-Hill, June 8 1999.

[102] Curve Fitting in the HP 3562A, Product note HP 3562A-3 ed., Hewlett-Packard, 1989.

[103] D. Y. Abramovitch and C. R. Moon, "Automatic tuning of atomic force microscope," USPTO, Keysight Technologies Santa Rosa, CA USA, United States Patent 9,678,103, June 13 2017.

[104] z-Domain Curve Fitting in the HP 3563A Analyzer, Hp 3563a-1 product note ed., Hewlett-Packard, 1989.

[105] H. W. Bode, “Relations between attenuation and phase in feedback amplifier design,” Bell System Technical Journal, vol. 19, pp. 412–454, July 1940.

[106] J. A. Butterworth, L. Y. Pao, and D. Y. Abramovitch, “Fitting discrete-time models to frequency responses for systems with transport delay,” in ASME 2011 International Mechanical Engineering Congress and Exposition, ASME. ASME, 2011.

[107] D. Y. Abramovitch, “Task list for fully automated PID tuning,” Agilent Laboratories, Internal Report/Memo, September 21 2007.

[108] D. Y. Abramovitch, S. Hoen, and R. Workman, “Semi-automatic tuning of PID gains for atomic force microscopes,” in Proceedings of the 2008 American Control Conference, AACC. Seattle, WA: IEEE, June 11–13 2008.

[109] D. Y. Abramovitch, S. T. Hoen, and R. K. Workman, “Automatic generation of PID parameters for a scanning probe microscope,” Agilent Technolgies, Inc., Santa Clara, CA USA, United States Patent 7,987,006, July 26 2011.

[110] D. Y. Abramovitch and C. R. Moon, “Cascaded digital filters with reduced latency,” World Intellectual Property Organization, International Application Published Under the Patent Cooperation Treaty WO 2012/118483, September 9 2012.

[111] D. Y. Abramovitch, “Curve fits on high-Q mechatronic systems in the presence of noise,” in To be submitted to the 2016 American Control Conference, AACC. Boston, MA: IEEE, July 2016.

[112] J. A. Butterworth, L. Y. Pao, and D. Y. Abramovitch, “Fitting discrete-time models to frequency responses for systems with transport delay,” in ASME Int. Mechanical Engr. Congress & Exposition. ASME, 2011.

[113] ——, “The effect of nonminimum-phase zero locations on the performance of feedforward model-inverse control techniques in discrete-time systems,” in Proceedings of the American Control Conference, AACC. Seattle, WA: IEEE, June 2008.

[114] Wikipedia. (2018) Padé approximant. [Online; accessed June 6, 2018]. [Online]. Available: https://en.wikipedia.org/wiki/Pade_approximant

[115] K. J. ström and T. Hägglund, PID Controllers: Theory, Design, and Tuning. ISA Press, 1995.

[116] Wikipedia. (2016) Apollo guidance computer. [Online; accessed June 26, 2016]. [Online]. Available: https://en.wikipedia.org/wiki/Apollo_Guidance_Computer

[117] J. Tylko, “MIT and navigating the path to the moon,” AeroAstro, 2008–2009. [Online]. Available: http://web.mit.edu/aeroastro/news/magazine/aeroastro6/mit-apollo.html

[118] K. J. ström and T. Hägglund, Advanced PID Control, ser. Oxford Series on Optical and Imaging Sciences. ISA Press, August 15 2005.

[119] M. A. Johnson and M. H. Moradi, Eds., PID Control: New Identification and Design Methods. London: Springer-Verlag, 2005.

[120] T. Wescott, "PID without a PhD," Embedded Systems Programming, pp. 86–108, October 2000.

[121] W. Messner, "The development, properties, and application of the complex phase lead compensator," in Proceeding of the 2000 American Control Conference, AACC. Chicago, IL: IEEE, June 2000, pp. 2621–2626.

[122] W. C. Messner, M. D. Bedillion, L. Xia, and D. C. Karns, "Lead and lag compensators with complex poles and zeros," IEEE Control Systems Magazine, vol. 27, no. 1, pp. 44–54, February 2007.

[123] W. Messner, "Formulas for asymmetric lead and lag compensators," in Proceeding of the 2009 American Control Conference, AACC. St. Louis, MO: IEEE, June 2009, pp. 3769–3774.

[124] K. M. Moudgalya, Digital Control. John Wiley & Sons, 2007.

[125] D. Abramovitch, "Lyapunov Redesign of classical digital phase-lock loops," in Proceedings of the 2003 American Control Conference, AACC. Denver, CO: IEEE, June 2003, pp. 2401–2406.

[126] J. G. Ziegler and N. E. Nichols, "Optimum settings for automatic controllers," Transactions of the ASME, vol. 64, pp. 759–768, November 1942.

[127] J. Ziegler and N. B. Nichols, "Process lags in automatic control circuits," Transactions of the ASME, vol. 65, no. 5, pp. 433–444, November 1942.

[128] D. Croft, G. Shed, and S. Devasia, "Creep, hysteresis, and vibration compensation for piezoactuators: Atomic force microscopy application," ASME J. Dyn., Sys., Meas., & Ctrl., vol. 128, no. 35, pp. 35–43, 2001.

[129] T. Ando, T. Kodera, E. Takai, D. Maruyama, K. Saito, and A. Toda, "A high-speed atomic force microscope for studying biological macromolecules," PNAS, vol. 98, pp. 12 468–12 472, 2001.

[130] G. Schitter, K. J. ström, B. DeMartini, G. E. Fantner, K. Turner, P. J. Thurner, and P. K. Hansma, "Design and modeling of a high-speed scanner for atomic force microscopy," in Proc. Amer. Ctrl. Conf., Minneapolis, MN, June 2006, pp. 502–507.

[131] MFP-3D Atomic Force Microscope Controller – Fully Digital, Fast, Low Noise for High Performance, Asylum Research, www.asylum.com, August 2004.

[132] The New NanoScope V Controller: New Power and Capabilites for Multimode V, Dimension V, NanoMan VS and PicoForce, Veeco Instruments, www.veeco.com, 2006.

[133] G. F. Franklin, J. D. Powell, and M. L. Workman, Digital Control of Dynamic Systems, 3rd ed. Add. Wesl. Long., 1998.

[134] K. J. ström and T. Hägglund, Advanced PID Control. ISA Press, 2005.

[135] R. C. Smith, M. V. Salapaka, A. Hatch, J. Smith, and T. De, "Model development and inverse compensator design for high speed nanopositioning," in Proc. IEEE Conf. Dec. & Ctrl., Dec. 2002, pp. 3652–3657.

[136] D. Croft and S. Devasia, "Vibration compensation for high speed scanning tunneling microscopy," Rev. Sci. Instrum., vol. 70, no. 12, pp. 4600–4605, December 1999.

[137] G. Schitter and A. Stemmer, "Identification and open-loop tracking control of a piezoelectric tube scanner for high-speed scanning-probe microscopy," IEEE T. Contr. Syst. T., vol. 12, no. 3, pp. 449–454, 2004.

[138] T. Sulchek, R. Hsieh, J. D. Adams, G. G. Yaralioglu, S. C. Minne, C. F. Quate, J. P. Cleveland, A. Atalar, and D. M. Adderton, "High-speed tapping mode imaging with active $Q$ control for atomic force microscopy," Applied Physics Letters, vol. 76, p. 1473, 2000.

[139] N. Kodera, H. Yamashita, and T. Ando, "Active damping of the scanner for high-speed atomic force microscopy," Rev. Sci. Instrum., vol. 76, p. 053708, 2005.

[140] N. Kodera, M. Sakashita, and T. Ando, "Dynamic proportional-integral-differential controller for high-speed atomic force microscopy," Rev. Sci. Instrum., vol. 77, p. 083704, 2006.

[141] G. Schitter, P. Menold, H. Knapp, F. Allgöwer, and A. Stemmer, "High performance feedback for fast scanning atomic force microscopes," Rev. Sci. Instrum., vol. 72, no. 8, pp. 3320–3327, 2001.

[142] A. Sebastian, M. V. Salapaka, and J. P. Cleveland, "Robust control approach to atomic force microscopy," in Proc. IEEE Conf. Dec. & Ctrl., Maui, HI, Dec. 2003, pp. 3443–3444.

[143] A. Sebastian and S. Salapaka, "Design methodologies for robust nano-positioning," IEEE T. Contr. Syst. T., vol. 13, no. 6, pp. 868–876, 2005.

[144] S. Salapaka, A. Sebastian, J. P. Cleveland, and M. V. Salapaka, "High bandwidth nano-positioner: A robust control approach," Rev. Sci. Instrum., vol. 73, no. 9, pp. 3232–3241, 2002.

[145] G. Schitter, R. W. Stark, and A. Stemmer, "Fast contact-mode atomic force microscopy on biological specimen by model-based control," Ultramicroscopy, vol. 100, no. 3-4, pp. 253–257, 2004.

[146] J. M. Rieber, G. Schitter, A. Stemmer, and F. Allgöwer, "Experimental application of $\ell_1$-optimal control in atomic force microscopy," in Proc. IFAC World Congress, Prague, Czech Republic, July 2005.

[147] G. Schitter, F. Allgöwer, and A. Stemmer, "A new control strategy for high-speed atomic force microscopy," Nanotechnology, vol. 15, no. 1, pp. 108–114, 2004.

[148] L. Y. Pao, J. A. Butterworth, and D. Y. Abramovitch, "Combined feedforward/feedback control of atomic force microscopes," in Proc. Amer. Ctrl. Conf., New York, NY, July 2007.

[149] M. Khammash and H. El-Samad, "Systems biology: From physiology to gene regulation," IEEE Control Systems Magazine, pp. 62–76, August 2004.

[150] Wikipedia. (2018) Sandro Botticelli's "The Birth of Venus". [Online; accessed June 10, 2018]. [Online]. Available: https://en.wikipedia.org/wiki/The_Birth_of_Venus

[151] G. Stein, "Respect the unstable," December 1989, bode Lecture presented at the 1989 IEEE Conference on Decision and Control, Tampa FL.

[152] D. Y. Abramovitch, "Lyapunov Redesign of analog phase-lock loops," The IEEE Transactions on Communication, vol. 38, no. 12, pp. 2197–2202, December 1990.

[153] P. C. Parks, "Liapunov redesign of model reference adaptive control systems," IEEE Trans. on Automatic Control, vol. AC-11, no. 3, July 1966.

[154] R. E. Best, Phase-Locked Loops: Design, Simulation, and Applications, 4th ed. New York: McGraw-Hill, 1999.

[155] D. H. Wolaver, Phase-Locked Loop Circuit Design, ser. Advanced Reference Series & Biophysics and Bioengineering Series. Englewood Cliffs, New Jersey 07632: Prentice Hall, 1991.

[156] F. M. Gardner, Phaselock Techniques, 2nd ed. New York, NY: John Wiley & Sons, 1979, iSBN 0-471-04294-3.

[157] D. Y. Abramovitch, "Analysis and design of a third order phase-lock loop," in Proceedings of the IEEE Military Communications Conference. IEEE, October 1988.

[158] H. W. Bode, Network Analysis and Feedback Amplifier Design. New York: Van Nostrand, 1945.

[159] S. Boyd and C. A. Desoer, "Subharmonic functions and performance bounds on linear time-invariant feedback systems," IMA J. of Mathematical Control and Information, vol. 2, pp. 153–170, 1985, also in *Proc. 1984 Conf. on Decision and Control*.

[160] I. M. Horowitz, Quantitative Feedback Theory (QFT). 4470 Grinnel Ave., Boulder, CO 80303: QFT Publications, 1992.

[161] C. Mohtadi, "Bode's integral theorem for discrete-time systems," Proceedings of the IEE, vol. 137, no. 2, pp. 57–66, March 1990.

[162] J. C. Doyle and G. Stein, "Multivariable feedback design: Concepts for a classical/modern synthesis," IEEE Trans. Aut. Control, vol. AC-26, no. 1, pp. 4–16, Feb. 1981.

[163] R. A. Mueller, "Optimizing the performance of the pilot control loaders at the NASA vertical motion simulator," American Institute of Aeronautics and Astronautics (AIAA) Journal of Aircraft, vol. 47, no. 2, pp. 682–693, March-April 2010.

[164] Wikipedia. (2020) Cascaded integrator-comb filter. [On line; accessed April 19, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Cascaded_integrator-comb_filter

[165] D. Y. Abramovitch, "A comparison of $\Delta$ coefficients and the $\delta$ parameterization, Part I: Coefficient accuracy," in Proceedings of the 2017 American Control Conference, AACC. Seattle, WA: IEEE, May 2017.

[166] ——, "A comparison of $\Delta$ coefficients and the $\delta$ parameterization, Part II: Signal growth," in Proceedings the 2018 American Control Conference, AACC. Milwaukee, WI: IEEE, June 2018, pp. 5231–5237.

[167] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing. Englewood Cliffs, N. J.: Prentice Hall, 1975.

[168] P. Horowitz and W. Hill, The Art of Electronics, 1st ed. Cambridge University Press, October 31 1980.

[169] T. Instruments, TMS320C4x User's Guide, Texas Instruments, 1993.

[170] Wikipedia. (2020) Data processing inequality. [On line; accessed June 16, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Data_processing_inequality

[171] T. Kailath, Linear Systems. Englewood Cliffs, N.J. 07632: Prentice-Hall, 1980.

[172] J. O. Smith, Introduction to Digital Filters with Audio Applications. http://www.w3k.org/books/: W3K Publishing, 2007.

[173] P. M. Embree, C Algorithms for Real-Time DSP. Upper Saddle River, NJ 07458: Prentice Hall PTR, 1995.

[174] Spartan-6 FPGA DSP48A1 Slice User Guide, Ug389 (v1.1) ed., August 13 2009.

[175] Virtex-5 FPGA XtremeDSP Design Considerations User Guide, Ug193 (v3.5) ed., January 26 2012.

[176] LogiCORE IP Floating-Point Operator v6.0, Ds816 (v1.2) ed., Xilinx, January 18 2012.

[177] R. H. Middleton and G. C. Goodwin, "Improved finite word length characteristics in digital control using $\delta$ operators," IEEE Transactions on Automatic Control, vol. 31, no. 11, pp. 1015–1021, November 1986.

[178] R. M. Goodall and B. J. Donoghue, "Very high sample rate digital filters using the $\delta$ operator," IEE Proceedings-G, vol. 140, no. 3, pp. 199–206, June 1993.

[179] G. Li and M. Gevers, "Comparative study of finite wordlength effects in shift and delta operator parameterizations," IEEE Transactions on Automatic Control, vol. 38, no. 5, pp. 803–807, May 1993.

[180] G. C. Goodwin, J. I. Yuz, J. C. Agüero, and M. Cea, "Sampling and sampled-data models," in Proceedings of the 2010 American Control Conference, AACC. Baltimore, MD: IEEE, June 2010.

[181] J. Kauraniemi, T. I. Laakso, I. Hartimo, and S. J. Ovaska, "Delta operator realizations of direct-form IIR filters," IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing, vol. 45, no. 1, pp. 41–52, January 1998.

[182] D. Y. Abramovitch and C. R. Moon, "Cascaded digital filters with reduced latency," USPTO, United States Patent Application US 2014/0040339 A1, February 6 2014.

[183] D. Y. Abramovitch and E. Johnstone, "State space system simulator utilizing bi-quadratic blocks to simulate lightly damped resonances," World Intellectual Property Organization, International Application Published Under the Patent Cooperation Treaty WO 2013/130076, September 6 2013.

[184] D. Y. Abramovitch and E. S. Johnstone, "State space system simulator utilizing bi-quadratic blocks to simulate lightly damped resonances," USPTO, Patent Application Publication US 2015/0006133 A1, January 1 2015.

[185] D. Y. Abramovitch, "The Demod Squad: A tutorial on the utility and methodologies for using modulated signals in feedback loops," in Proceedings of the 2020 IEEE Conference on Control Technology and Applications, IEEE. Montreal, Canada: IEEE, August 2020.

[186] S. A. C. Doyle, The Sign of Four. London: Lippincott's Monthly Magazine, 1890.

[187] D. Abramovitch, "Rejecting rotational disturbances on small disk drives using rotational accelerometers," in Proceedings of the 1996 IFAC World Congress, IFAC. San Francisco, CA: IEEE, July 1996, pp. 483–488 (Volume O).

[188] D. Y. Abramovitch and G. Hsu, “Mitigating the effects of disturbances of a disk drive,” in Proceedings of the 2015 Multi-Conference on Systems and Control, IEEE. Sydney, Australia: IEEE, September 21-23 2015, pp. 1473–1478.

[189] A. Sacks, M. Bodson, and W. Messner, “Advanced methods for repeatable runout compensation (disc drives),” in Digests of The Magnetic Recording Conference 1994. Pittsburg, PA: IEEE, August 1994.

[190] M. Bodson, A. Sacks, and P. Khosla, “Harmonic generation in adaptive feedforward cancellation schemes,” IEEE Transactions on Automatic Control, vol. 39, no. 9, pp. 1939–1944, September 1994.

[191] L. Y. Pao, J. A. Butterworth, and D. Y. Abramovitch, “Combined feedforward/feedback control of atomic force microscopes,” in Proceedings of the 2007 American Control Conference, AACC. New York, NY: IEEE, July 11–13 2007, pp. 3509–3515.

[192] H. Perez, Q. Zou, and S. Devasia, “Design and control of optimal feedforward trajectories for scanners: Stm example,” in Proceedings of the 2002 American Control Conference, AACC. Anchorage, AK: IEEE, May 2002, pp. 2305–2312.

[193] M. Tomizuka, “Zero phase error tracking algorithm for digital control,” ASME Journal of Dynamic Systems, Measurement, and Control, March 1987.

[194] L. Y. Pao and K. E. Johnson, “Control of wind turbines,” IEEE Control Systems Magazine, vol. 31, no. 2, pp. 44–62, April 2011.

[195] J. S. McAllister, “The effect of disk platter resonances on track misregistration in 3.5 inch disk drives,” IEEE Transactions on Magnetics, vol. 32, no. 3, pp. 1762–1766, May 1996.

[196] ——, “Characterization of disk vibrations on aluminum and alternate substrates,” IEEE Transactions on Magnetics, vol. 33, no. 1, p. 968, May 1996.

[197] ——, “Disk flutter: Causes and potential cures,” Data Storage, vol. 4, no. 6, pp. 29–34, May/June 1997.

[198] 3585A Spectrum Analyzer Operating Manual, Part number 03585-90003 ed., Hewlett Packard, February 1979.

[199] CXA X-Series Signal Analyzer, Multi-touch N9000B: 9 kHz to 3.0, 7.5, 13.6, or 26.5 GHz Data Sheet, Part number 5992-1274en ed., KeySight, February January.

[200] 8566B Spectrum Analyzer: 100 Hz–2.5GHz/2-22GHz, Manual, Part number 08566-90040 ed., Hewlett Packard, March 1984.

[201] R. G. Lyons, "Reducing ADC quantization noise," Microwaves and RF Magazine, pp. 10–21, June 2005.

[202] W. Kester, Ed., Linear Design Seminar Notes, 1st ed. Norwoood, MA: Analog Devices, Inc., 1995.

[203] C. E. Shannon, "A mathematical theory of communication," Bell Systems Technical Journal, vol. 27, pp. 379–423,623–656, 1948.

[204] Wikipedia. (2020) Parseval's theorem. [On line; accessed May 22, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Parseval's_theorem

[205] T. Wolfe, The Right Stuff. Farrar, Straus and Giroux, January 1 1979.

[206] D. Y. Abramovitch, "The Banshee Multivariable Workstation: A tool for disk drive research," in Advances in Information Storage Systems, Vol. 5, B. Bhushan, Ed. New York, NY: ASME Press, 1993, pp. 59–72.

[207] R. Loughridge and D. Y. Abramovitch, "A tutorial on laser interferometry for precision measurements," in Proceedings of the 2013 American Control Conference, AACC. Washington, DC: IEEE, June 17–19 2013.

[208] A. E. Bryson and Y. C. Ho, Applied Optimal Control. 1010 Vermont Ave., N. W., Washington, D.C. 20005: Hemisphere Publishing Co., 1975.

[209] M. Spock, "Only nixon could go to china," in Star Trek, VI: The Undiscovered Country, L. Nimoy, L. Konner, and M. Rosenthal, Eds., Star Fleet. Alpha Quadrant: Paramount Pictures, 1991.

[210] J. Burke, The Day the Universe Changed. Boston and Toronto: Little, Brown and Company, 1985.

[211] D. Abramovitch, "Customizable coherent servo demodulation for disk drives," IEEE/ASME Transactions on Mechatronics, vol. 3, no. 3, pp. 184–193, September 1998.

[212] ——, "Customizable coherent servo demodulation for disk drives," in Proceedings of the 1998 American Control Conference, AACC. Philadelphia, PA: IEEE, June 1998, pp. 3043–3049.

[213] D. Y. Abramovitch, "Disk drive servo demodulation system which supresses noise on the position error signal," Hewlett-Packard, Palo Alto, CA USA, United States Patent 5,801,895, September 1 1998.

[214] W. H. Calvin, "Normal repetitive firing and its pathophysiology," in Epilepsy: A Window to Brain Mechanisms, J. Lockard and A. A. Ward, Eds. New York: Raven Press, 1980, pp. 97–121.

[215] B. Widrow, Y. Kim, and D. Park, "The Hebbian-LMS learning algorithm," IEEE Computational Intelligence Magazine, pp. 37–53, November 2015.

[216] Q. Zhong, D. Inniss, K. Kjoller, and V. Ellings, "Fractured polymer/silica fiber surface studied by tapping mode atomic force microscopy," Surf. S. Lett., vol. 290, no. 1-2, pp. L688–L692, Jun. 1993.

[217] V. B. Elings and J. A. Gurley, "Jumping probe microscope," Digital Instruments, Santa Barbara, CA USA, United States Patent 5,266,801, November 30 1993.

[218] D. M. Harcombe, M. G. Ruppert, and A. J. Fleming, "A review of demodulation techniques for multifrequency atomic force microscopy," Beilstein Journal of Nanotechnology, vol. 11, pp. 76–91, January 2020.

[219] M. R. P. Ragazzon, S. Messineo, J. T. Gravdahl, D. Harcombe, and M. G. Ruppert, "Generalized lyapunov demodulator for amplitude and phase estimation by the internal model principle," in Conference: 8th IFAC Symposium on Mechatronic Systems, IFAC. Vienna, Austria: IFAC, September 2019.

[220] Wikipedia. (2020) Amplitude modulation. [On line; accessed April 12, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Amplitude_modulation

[221] ——. (2020) Phase modulation. [On line; accessed April 12, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Phase_modulation

[222] ——. (2020) Frequency modulation. [On line; accessed April 12, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Frequency_modulation

[223] ——. (2020) Quadrature amplitude modulation. [On line; accessed April 12, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Quadrature_amplitude_modulation

[224] C. Kitchin and L. Counts, "RMS to DC conversion application guide, $2^{ND}$ edition," Analog Devices, Inc., Product Guide, 1986.

[225] K. S. Karvinen and S. O. R. Moheimani, "A high-bandwidth amplitude estimation technique for dynamic mode atomic force microscopy," Review of Scientific Instruments, vol. 85, no. 2, p. 023707, 2014.

[226] M. G. Ruppert, K. S. Karvinen, S. L. Wiggins, and S. O. R. Moheimani, "A Kalman filter for amplitude estimation in high-speed dynamic mode atomic force microscopy," IEEE Transactions on Control Systems Technology, vol. 24, no. 1, pp. 276–284, January 2016.

[227] J. E. Volter, "The CORDIC trigonometric computing technique," IRE Transactions on Electronic Computation, vol. 8, pp. 330–334, 1959.

[228] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," IEEE Transactions on Circuits and Systems – I:Regular Papers, vol. 56, no. 9, pp. 1893–1907, September 2009.

[229] A. Blanchard, Phase-Locked Loops. New York, NY: John Wiley & Sons, 1976.

[230] A. J. Viterbi, Principles of Coherent Communication, ser. McGraw-Hill Series in Systems Science. New York, NY: McGraw-Hill, 1966.

[231] R. E. Best, Phase-Locked Loops: Design, Simulation, and Applications, 3rd ed. New York: McGraw-Hill, 1997.

[232] J. A. Crawford, Frequency Synthesizer Design Handbook. Norwood, MA 02062: Artech House, 1994.

[233] D. Abramovitch and G. Franklin, "Disk drive control: The early years," Annual Reviews in Control, vol. 26, no. 2, pp. 229–242, February 2003.

[234] D. Abramovitch, "Lyapunov Redesign of analog phase-lock loops," in Proceedings of the 1989 American Control Conference, AACC. Pittsburg, PA: IEEE, June 1989, pp. 2684–2689.

[235] H. De Bellescize, "La réception synchrone," L'onde électrique, vol. 11, pp. 225–240, May 1932.

[236] D. Y. Abramovitch, "Efficient and flexible simulation of phase locked loops, Part I: Simulator design," in Proceedings of the 2008 American Control Conference, AACC. Seattle, WA: IEEE, June 11–13 2008, pp. 4672–4677.

[237] ——, "Efficient and flexible simulation of phase locked loops, Part II: Post processing and a design example," in Proceedings of the 2008 American Control Conference, AACC. Seattle, WA: IEEE, June 11–13 2008, pp. 4678–4683.

[238] ——, "Coherent demodulation with reduced latency adapted for use in scanning probe microscopes," Agilent Technologies, Santa Clara, CA USA, United States Patent 7,843,627, November 30 2010.

[239] D. Croft, G. Shed, and S. Devasia, "Creep, hysteresis, and vibration compensation for piezoactuators: Atomic force microscopy application," ASME Journal of Dynamic Systems, Measurement, and Control, vol. 128, no. 35, pp. 35–43, March 2001.

[240] D. Croft and S. Devasia, "Vibration compensation for high speed scanning tunneling microscopy," Review of Scientific Instruments, vol. 70, no. 12, pp. 4600–4605, December 1999.

[241] K. K. Leang, Q. Zou, and S. Devasia, “Feedforward control of piezoactuators in atomic force microscope systems: Inversion-based compensation for dynamics and hysteresis,” IEEE Control Systems Magazine, vol. 19, pp. 70–82, 2009.

[242] K. K. Leang and S. Devasia, “Feedback-linearized inverse feedforward for creep, hysteresis, and vibration compensation in AFM piezoactuators,” IEEE Transactions on Control Systems Technology, vol. 15, no. 5, pp. 927–935, 2007.

[243] J. A. Butterworth, L. Y. Pao, and D. Y. Abramovitch, “Combined feedforward/feedback control of atomic force microscopes,” in Proceedings of the American Control Conference, AACC. New York, NY: IEEE, June 2007.

[244] ——, “A comparison of control architectures for atomic force microscopes,” in Proceedings of the 2008 IFAC Triennial World Congress, AACC. Seoul, Korea: IEEE, July 2008.

[245] ——, “A comparison of control architectures for atomic force microscopes,” Asian Journal of Control, vol. 11, no. 2, pp. 175–181, March 2009.

[246] ——, “Adaptive-delay combined feedforward/feedback control for raster tracking with applications to AFMs,” in Proceedings of the American Control Conference, AACC. Baltimore, MD: IEEE, June 2010.

[247] ——, “Analysis and comparison of three discrete-time feedforward model-inverse control techniques for nonminimum-phase systems,” Mechatronics, vol. 22, no. 5, pp. 577–587, August 2012.

[248] ——, “A discrete-time single-parameter combined feedforward/feedback adaptive-delay algorithm with applications to piezo-based raster tracking,” IEEE Transactions on Control Systems Technology, vol. 20, no. 5, pp. 416–423, March 2012.

[249] ——, “A comparison of ILC architectures for nanopositioners with applications to AFM raster tracking,” in Proceedings of the American Control Conference, AACC. San Francisco, CA: IEEE, June 2011.

[250] ——, “Dual-adaptive feedforward control for raster tracking with applications to AFMs,” in Proceedings of the IEEE International Conference on Control Applications, CCA 2011, IEEE. Denver, CO, USA: IEEE, September 28-30 2011.

[251] Wikipedia. (2017) Q factor. [Online; accessed February 6, 2017]. [Online]. Available: https://en.wikipedia.org/wiki/Q_factor

[252] T. Sulchek, G. G. Yaralioglu, S. C. Minne, and C. F. Quate, “Characterization and optimization of scan speed for tapping mode atomic force microscopy,” Rev. Sci. Instrum., vol. 73, p. 2928, 2002.

[253] R. W. Stark, G. Schitter, and A. Stemmer, "Tuning the interaction forces in tapping mode atomic force microscopy," Physical Review B, vol. 68, pp. 085 401–1–085 401–5, 2003.

[254] B. Anczykowski, B. Gotsmann, H. Fuchs, J. P. Cleveland, and V. B. Elings, "How to measure energy dissipation in dynamic mode atomic force microscopy," Appl. Surf. Sci., vol. 140, pp. 376–382, 1999.

[255] P. E. Gill, W. Murray, and M. H. Wright, Practical Optimization. London: Academic Press, 1981.

[256] Mentor Graphics, ModelSim, Mentor Graphics, www.mentor.com/products/fpga/simulation/modelsim, 2011.

[257] S. Vadlaman and W. Mahmoud, "Comparison of CORDIC algorithm implementations on FPGA families," in Proceedings of the Thirty-Fourth Southeastern Symposium on System Theory, 2002., 2002, pp. 192–196.

[258] Xilinx, CORDIC v4.0, Xilinx Corporation, www.xilinx.com, April 29 2009.

[259] R. Proksch, J. Cleveland, D. Bocek, T. Day, M. Viani, and C. Callahan, "Fully digital controller for cantilever-based measurements," Asylum Research Corporation, Santa Barbara, CA USA, United States Patent 7,234,342, June 26 2007.

[260] K. Technologies, "Keysight 9500 AFM," Keysight Technologies, Data Sheet, 2016.

[261] M. G. Ruppert and S. O. R. Moheimani, "Multimode Q control in tapping-mode afm: Enabling imaging on higher flexural eigenmodes," IEEE Transactions on Control Systems Technology, 2016.

[262] M. R. P. Ragazzon, Parameter Estimation in Atomic Force Microscopy: Nanomechanical Properties and High-speed Demodulation. Trondheim: Norwegian University of Science and Technology, April 2018.

[263] D. Y. Abramovitch, "Turning the tracking problem sideways: Servo tricks for DVD+RW clock generation," in Proceedings of the 2000 American Control Conference, AACC. Chicago, IL: IEEE, June 2000, pp. 2615–2620.

[264] D. Abramovitch, D. Towner, C. Perlov, J. Hogan, M. Fischer, C. Wilson, I. Çokgör, and C. Taussig, "High Frequency Wobbles: A write clock generation method for rewritable DVD that enables near drop-in compatibility with DVD-ROMs," in Technical Digest of ISOM/ODS 1999: Joint International Symposium on Optical Memory and Optical Data Storage 1999 Conference, IEEE/LEOS,OSA,SPIE,JSAP. Koloa, HI: SPIE, July 1999, pp. 56–58.

[265] D. Y. Abramovitch, "Magnetic and optical disk control: Parallels and contrasts," Agilent Labs, Palo Alto, CA 94304, On-Line Full Draft of Paper, June 2001, http://www.labs.agilent.com/-personal/Danny_Abramovitch/pubs/hd_vs_od_servo.pdf.

[266] ——, "References for: Magnetic and Optical Disk Control: Parallels and Contrasts," Agilent Labs, Palo Alto, CA 94304, On-Line Bibliography of Paper, June 2001, http://www.labs.agilent.com/-personal/Danny_Abramovitch/pubs/hd_vs_od_servo_refs.pdf.

[267] E. Hecht and A. Zajac, Optics, ser. Addison-Wesley Series in Physics. Reading, MA: Addison-Wesley, 1979.

[268] M. Born and E. Wolf, Principles of Optics: Electromagnetic Theory of Propagation, Interference, and Diffraction of Light, 6th ed. Oxford, England: Pergamon Press, 1980.

[269] S. Ramo, J. R. Whinnery, and T. V. Duzer, Fields and Waves in Communication Electronics, 2nd ed. New York, NY: John Wiley & Sons, 1984.

[270] J. William H. Hayt, Engineering Electromagnetics, 4th ed., ser. Electrical & Electronic Engineering. McGraw-Hill Inc., 1981.

[271] J. N. Dukes and G. B. Gordon, "A two-hundred-foot yardstick with graduations every microinch," Hewlett-Packard Journal, pp. 2–9, August 1970.

[272] H. Butler, "Position control in lithographic equipment: An enabler for current-day chip manufacturing," IEEE Control Systems Magazine, pp. 28–47, October 2011.

[273] R. M. Schmidt, G. Schitter, and J. V. Eijk, The Design of High Performance Mechatronics: High-Tech Functionality by Multidisciplinary System Integration. Delft, NL: IOS Press (Delft University Press), September 15 2011.

[274] J. Wen and B. Potsaid, "An experimental study of a high performance motion control system," in Proceedings of the 2004 American Control Conference, AACC. Boston, MA: IEEE, June 2004, pp. 5158–5163.

[275] B. P. Rigney, L. Y. Pao, and D. A. Lawrence, "Nonminimum phase dynamic inversion for settle time applications," IEEE Transactions on Control Systems Technology, vol. 17, no. 5, pp. 989–1005, 2009.

[276] N. C. Singer and W. P. Seering, "Using acausal shaping techniques to reduce robot vibration," in Proceedings of the 1989 IEEE International Conference on Robotics and Automation, IEEE. Scottsdale, AZ: IEEE, April 24-29, 1988, pp. 1434–1439.

[277] ——, "Design and comparison of command shaping methods for controlling residual vibration," in Proceedings of the 1989 IEEE International Conference on Robotics and Automation, IEEE. Scottsdale, AZ: IEEE, May 14-19, 1989, pp. 888–893.

[278] A. Sacks, M. Bodson, and W. Messner, "Advanced methods for repeatable runout compensation (disc drives)," IEEE Transactions on Magnetics, vol. 31, no. 2, pp. 1031–1036, March 1995.

[279] B. Francis and W. M. Wonham, "The internal model principle for linear multivariable regulators," Applied Mathematics and Optimization, vol. 2, pp. 170–194, 1975.

[280] M. Tomizuka, T.-C. Tsao, and K.-K. Chew, "Discrete-time domain analysis and synthesis of repetitive controllers," in Proceedings of the 1988 American Control Conference, AACC. Atlanta, GA: IEEE, June 1988, pp. 860–866.

[281] T.-C. Tsao and M. Tomizuka, "Adaptive and repetitive digital control algorithms for noncirculating machining," in Proceedings of the 1988 American Control Conference, AACC. Atlanta, GA: IEEE, June 1988, pp. 115–120.

[282] K. K. Chew and M. Tomizuka, "Digital control of repetitive errors in disk drive systems," IEEE Control Systems Magazine, vol. 10, no. 1, pp. 16–20, January 1990.

[283] C. A. Desoer and M. Vidyasagar, Feedback Systems: Input-Output Properties. New York: Academic Press, 1975.

[284] D. Y. Abramovitch, "Rejection of disturbances on a disk drive by use of an accelerometer," Hewlett-Packard, Palo Alto, CA USA, United States Patent 5,663,847, September 2 1997.

[285] M. White and M. Tomizuka, "Increased disturbance rejection in magnetic disk drives by acceleration feedforward control," Control Engineering Practice, vol. 5, no. 6, pp. 741–751, 1997.

[286] A. Jinzenji, T. Sasamoto, K. Aikawa, S. Yoshida, and K. Aruga, "Acceleration feedforward control against rotational disturbance in hard disk drives," IEEE Transactions on Magnetics, vol. 37, no. 2, pp. 888–893, March 2001.

[287] D. Y. Abramovitch, "Magnetic and optical disk control: Parallels and contrasts," in Proceedings of the 2001 American Control Conference, AACC. Arlington, VA: IEEE, June 2001, pp. 421–428.

[288] D. B. Davies and M. D. Sidman, "Active compensation of shock, vibration, and wind-up in disk drives," in Advances in Information Storage Systems, Vol. 5, B. Bhushan, Ed. New York, NY: ASME Press, 1993, pp. 5–20.

[289] V. L. Knowles and D. M. Hanks, "Shock and vibration disturbance compensation system for disc drives," Hewlett-Packard Co., Corporate Patent Department, M/S 20B-O, 3000 Hanover Street, Palo Alto, CA 94304 USA, European Patent Application 871065555.3, June 1987.

[290] M. White and M. Tomizuka, "Increased disturbance rejection in magnetic disk drives by acceleration feedforward control," in Proceedings of the 1996 IFAC World Congress, IFAC. San Francisco, CA: IEEE, July 1996, pp. 489–494 (Volume O).

[291] D. Y. Abramovitch, “Introducing feedback control to middle and high school stem students, Part 1: Basic concepts,” in Proceedings of the 12th IFAC Symposium on Advances in Control Education, IFAC. Philadelphia, PA: IFAC, July 2019.

[292] ——, “Introducing feedback control to middle and high school stem students, Part 2: Control system math,” in Proceedings of the 12th IFAC Symposium on Advances in Control Education, IFAC. Philadelphia, PA: IFAC, July 2019.

[293] Wikipedia. (2022, December 11) 800-pound gorilla. [On line; accessed December 11, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/800-pound_gorilla

[294] S. Kubrick, Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb. Columbia Pictures, 1964.

[295] C. E. Shannon, “Communication in the presence of noise,” Proceedings of the IRE, vol. 37, no. 1, pp. 10–21, January 1949.

[296] Wikipedia. (2022, December 11) Nyquist-Shannon sampling theorem. [On line; accessed December 12, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

[297] M. S. Grewal and A. P. Andrews, Kalman Filtering: Theory and Practice Using MATLAB, 2nd ed., ser. Wiley. New York, NY: Addison-Wesley, 2001.

[298] D. Y. Abramovitch, “Determining Kalman filter input noises using PES Pareto,” in Proceedings of the 2021 American Control Conference, AACC. New Orleans, LA: IEEE, May 2021, pp. 4292–4298.

[299] H.-W. Kim and S.-K. Sul, “A new motor speed estimator using kalman filter in low speed range,” IEEE Transactions on Industrial Electronics, vol. 43, no. 4, pp. 498–504, August 1996.

[300] M. T. White, M. Tomizuka, and C. Smith, “Improved track following in magnetic disk drives using a disturbance observer,” IEEE/ASME Transactions on Mechatronics, vol. 5, no. 1, pp. 3–11, March 2000.

[301] R. Miklosovic, A. Radke, and Z. Gao, “Discrete implementation and generalization of the extended state observer,” in Proceedings of the 2006 American Control Conference, AACC. Minneapolis, MN: IEEE, June 2006, pp. 2209–2214.

[302] K. J. ström and R. M. Murray, Feedback Systems, 2nd ed. Princeton Univ. Press, 2016.

[303] D. Y. Abramovitch, “The discrete time biquad state space structure: Low latency with high numerical fidelity,” in Proc. Amer. Ctrl. Conf. Chicago: IEEE, 2015.

[304] ——, “The continuous time biquad state space structure,” in Proc. Amer. Ctrl. Conf. Chicago: IEEE, 2015.

[305] E. Johnstone and D. Y. Abramovitch, “Quintessential Phase: A method of mitigating turbulence effects in interferometer measurements of precision motion,” in Proceedings of the 2013 American Control Conference, AACC. Washington, DC: IEEE, June 17–19 2013.

[306] D. Abramovitch, “Notes on the peak finding in mass spectrometry,” Agilent Mass Spectrometry Division, 5301 Stevens Creek Blvd., MS: 3U, Santa Clara, CA 95051, White Paper, December 6 2017. [Online]. Available: http://dj.scs.agilent.com/danny/tex/mass_spec/architecture_notes/peak_finding/peak_finding_notes.pdf

[307] G. Schitter, G. E. Fantner, P. Thurner, J. Adams, and P. K. Hansma, “Design and characterization of a novel scanner for high-speed atomic force microscopy,” in Proc. 4th IFAC-Symp. Mech. Sys., 2006.

[308] D. Y. Abramovitch, S. B. Andersson, L. Y. Pao, and G. Schitter, “A tutorial on the mechanisms, dynamics, and control of atomic force microscopes,” in Proc. Amer. Ctrl. Conf., New York, NY, July 2007.

[309] D. Y. Abramovitch, “The Multinotch, Part I: A low latency, high numerical fidelity filter for mechatronic control systems,” in Proc. Amer. Ctrl. Conf. Chicago: IEEE, 2015.

[310] ——, “The Multinotch, Part II: Extra precision via $\Delta$ coefficients,” in Proc. Amer. Ctrl. Conf. Chicago: IEEE, 2015.

[311] ——, “A unified framework for analog and digital PID controllers,” in Proc. Multi-Conf. Sys. & Ctrl. Sydney: IEEE, 2015.

[312] ——, “Trying to keep it real: 25 years of trying to get the stuff I learned in grad school to work on mechatronic systems,” in Proc. Multi-Conf. Sys. & Ctrl. Sydney: IEEE, 2015.

[313] D. Y. Abramovitch and E. Johnstone, “State space system simulator utilizing bi-quadratic blocks to simulate lightly damped resonances,” Agilent Technologies, Santa Clara, CA USA, Patent Application PCT/US1227149, February 29 2012.

[314] Wikipedia. (2022) Newton’s laws of motion. [On line; accessed September 21, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/Newton’s_laws_of_motion

[315] D. Y. Abramovitch, S. B. Andersson, K. K. Leang, W. S. Nagel, and S. D. Ruben, “A tutorial on real-time computing issues for control systems,” in Proceedings of the 2023 American Control Conference, AACC. San Diego, CA: IEEE, May 31–June 2 2023.

[316] G. Clemenceau. (1932) War is too important a matter to be left to the military. [Online; accessed May 7, 2018]. [Online]. Available: https://en.wikiquote.org/wiki/Georges_Clemenceau

[317] K. Ogata, Discrete-Time Control Systems, 2nd ed. Englewood Cliffs, New Jersey: Prentice-Hall, 1994.

[318] Wikipedia. (2022) Phase noise. [On line; accessed September 28, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/Phase_noise

[319] ——. (2022) Jitter. [On line; accessed September 28, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/Jitter

[320] S. D. Ruben, "Respecte the implementation: Using NI myRIO in undergraduate control education," in Proceedings of the 2016 American Control Conference, AACC. Boston, MA: IEEE, July 6-8 2016, pp. 7315–7320.

[321] Keysight Technologies, "What is the difference? between an equivalent time sampling oscilloscope and a real-time oscilloscope," Keysight Technologies, Santa Rosa, CA USA, Application Note 5989-8794, April 9 2021, [On line; accessed September 28, 2022]. [Online]. Available: https://www.keysight.com/us/en/assets/7018-01852/application-notes/5989-8794.pdf

[322] D. Y. Abramovitch, Practical Methods for Real World Control Systems. Self, December 1 2022.

[323] K. J. ström and B. Wittenmark, Computer Controlled Systems, Theory and Design, 3rd ed. Englewood Cliffs, N.J. 07632: Prentice Hall, 1997.

[324] S. B. Andersson, "Lessons from the advanced tool world," in Proceedings of the 2023 American Control Conference, AACC. San Diego, CA: IEEE, May 31–June 2 2023.

[325] Xilinx. (2022) Xilinx adaptive SoCs. [On line; accessed October 4, 2022]. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc.html

[326] Intel. (2022) Intel FPGAs and Soc FPGAs. [On line; accessed October 4, 2022]. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga.html

[327] S. D. Ruben, "Controller implementation via analog computers," in Proceedings of the 2023 American Control Conference, AACC. San Diego, CA: IEEE, May 31–June 2 2023.

[328] W. S. Nagel, A. Mitrovic, G. M. Clayton, and K. K. Leang, "Discrete input-output sliding-mode control with range compensation: Application in high-speed nanopositioning," in Proceedings of the 2022 American Control Conference, AACC. Atlanta, GA: IEEE, May 31–June 2 2023, pp. 4371–4376.

[329] W. S. Nagel and K. K. Leang, "Discrete input-output state-space models for real-time control," in Proceedings of the 2023 American Control Conference, AACC. San Diego, CA: IEEE, May 31–June 2 2023.

[330] J. Madden and D. Anderson, One Size Doesn't Fit All. Jove, October 1 1989.

[331] D. Y. Abramovitch, "A comparison of the $\delta$ parameterization and the $\tau$ parameterization," in Proceedings of the 2019 American Control Conference, AACC. Philadelphia, PA: IEEE, July 2019.

[332] V. VanDoren, "Understanding PID control and loop tuning fundamentals," Control Engineering Magazine, 2023, [On line; accessed September 30, 2022]. [Online]. Available: https://www.controleng.com/articles/understanding-pid-control-and-loop-tuning-fundamentals/

[333] S. Tzu, S. B. Griffith, and B. H. L. Hart, The Art of War, reissue ed. Oxford University Press, January 1 1963, iSBN-10: 0195015401 ISBN-13: 978-0195015409.

[334] R. R. Negenborn and J. M. Maestre, "Distributed model predictive control: An overview and roadmap of future research opportunities," IEEE Control Systems Magazine, vol. 34, no. 4, pp. 87–97, August 2014.

[335] D. Y. Abramovitch, "Some crisp thoughts on fuzzy logic," in Proceedings of the 1994 American Control Conference, AACC. Baltimore, MD: IEEE, June 1994, avalable at dabramovitch.com.

[336] W. Messner and R. Ehrlich, "A tutorial on controls for disk drives," in Proceedings of the 2001 American Control Conference, AACC. Arlington, VA: IEEE, June 2001, pp. 408–420.

[337] D. Y. Abramovitch and G. F. Franklin, "A brief history of disk drive control," IEEE Control Systems Magazine, vol. 22, no. 3, pp. 28–42, June 2002.

[338] ——, "Disk drive control: The early years," in Proceedings of the 2002 IFAC World Congress, IFAC. Barcelona, ES: IEEE, July 2002.

[339] R. E. Kalman and J. E. Bertram, "Control system analysis and design via the "Second Method" of Lyapunov, Part 1: Continuous-Time Systems," Transactions of the ASME, 1959.

[340] ——, "Control system analysis and design via the "Second Method" of Lyapunov, Part 2: Discrete-Time Systems," Transactions of the ASME, 1959.

[341] Wikipedia. (2020) Aleksandr lyapunov. [On line; accessed July 6, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Aleksandr_Lyapunov

[342] T. Noah, Born a Crime: Stories from a South African Childhood. Random House Publishing Group, November 15 2016, iSBN-10: 0399588183; ISBN-13: 978-0399588181.

[343] Wikipedia. (2020) Patton (film). [On line; accessed July 6, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Patton_(film)

[344] ——. (2020) Infantry attacks. [On line; accessed July 6, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Infantry_Attacks

[345] Garson O'Toole Quote Investigator. (2017, March 23) Insanity is doing the same thing over and over again and expecting different results. [On line; accessed May 6, 2020]. [Online]. Available: https://quoteinvestigator.com/2017/03/23/same/

[346] Wikipedia. (2020) John henry (folklore). [On line; accessed July 6, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/John_Henry_(folklore)

[347] T. Wolfe, "The last American hero," in The Kandy-Kolored Tangerine-Flake Streamline Baby. New York: Farrar, Straus and Giroux, 1965, ch. 1, pp. 121–166.

[348] R. Gracie, "Gracie jiu jitsu," Martial Arts Seminar, March 1994, various quotations from Rickson's seminar.

[349] L. M. Scott, "The naked time," Star Trek Episode 7, Season 1, September 29 1966, stardate 1704.2.

[350] J. Rowling, Harry Potter and the Philosopher's Stone. Bloomsbury, Scholastic, 1997.

[351] G. Lucas, Star Wars. Skywalker Ranch, CA: Lucasfilm, 1977.

[352] G. Roddenberry, Star Trek: The Original Series. Los Angeles, CA: Desilu Productions, 1966.

[353] Pope John Paul I, "If someone had told me ..." http://www.saidwhat.co.uk/quotes/famous/pope_john_pau
August 1978.

[354] G. Verbinski, Pirates of the Caribbean: The Curse of the Black Pearl. Walt Disney Pictures, 2003.

[355] S. Herek, Bill & Ted's Excellent Adventure. De Laurentiis Entertainment Group (DEG), 1989.

# Alphabetical Index