Controls Engineering in the FIRST Robotics Competition

Tyler Veness

Controls Engineering in the FIRST Robotics Competition

Tyler Veness

https://github.com/calcmogul/controls-engineering-in-frc

Generated from commit 6d4cc6c made on May 30, 2025. The latest version can be downloaded from https://file.tavsys.net/control/controls-engineering-in-frc.pdf.

Contents

IV System modeling

Flower bush near Spencer's Market in Santa Maria, CA

Preface

When I was a high school student on FIRST Robotics Competition (FRC) team 3512, I had to learn control theory from scattered internet sources that either weren't rigorous enough or assumed too much prior knowledge. After I took graduate-level control theory courses from University of California, Santa Cruz for my bachelor's degree, I realized that the concepts weren't difficult when presented well, but the information wasn't broadly accessible outside academia.

I wanted to fix the information disparity so more people could appreciate the beauty and elegance I saw in control theory. This book streamlines the learning process to make that possible.

I wrote the initial draft of this book as a final project for an undergraduate technical writing class I took at UCSC in Spring 2017 (CMPE 185). It was a 13-page IEEE-formatted paper intended as a reference manual and guide to state-space control that summarized the three graduate controls classes I had taken that year. I kept working on it the following year to flesh it out, and it eventually became long enough to make into a proper book. I've been adding to it ever since as I learn new things.

I contextualized the material within FRC because it's always been a significant part of my life, and it's a useful application sandbox. I've since contributed implementations of many of this book's tools to the FRC standard library (WPILib) and maintain them.

Acknowledgements

Thanks to my controls engineering instructors Dejan Milutinović and Gabriel Elkaim for introducing me to the field of control theory. They taught me what it means to be a controls engineer, and I appreciated their classes's pragmatic perspective focused on intuition and application.

Trees by Baskin Engineering building at UCSC

0. Notes to the reader

0.1 Prerequisites

Knowledge of basic algebra and complex numbers is assumed. Some introductory physics and calculus will be taught as necessary.

0.2 Structure of this book

This book consists of five parts and a collection of appendices that address the four tasks a controls engineer carries out: derive a model of the system (kinematics), design a controller for the model (control theory), design an observer to estimate the current state of the model (localization), and plan how the controller is going to drive the model to a desired state (motion planning).

Part I "Fundamentals of control theory" introduces the basics of control theory and teaches the fundamentals of PID controller design.

Part II "Modern control theory" first provides a crash course in the geometric intuition behind linear algebra and covers enough of the mechanics of evaluating matrix algebra for the reader to follow along in later chapters. It covers state-space representation, controllability, and observability. The intuition gained in part I and the notation of linear algebra are used to model and control linear multiple-input, multiple-output (MIMO) systems and covers discretization, LQR controller design, LQE observer design, and feedforwards. Then, these concepts are applied to design and implement controllers for

real systems. The examples from part IV are converted to state-space representation, implemented, and tested with a discrete controller.

Part II also introduces the basics of nonlinear control system analysis with Lyapunov functions. It presents an example of a nonlinear controller for a unicycle-like vehicle as well as how to apply it to a two-wheeled vehicle. Since nonlinear control isn't the focus of this book, we mention other books and resources for further reading.

Part III "Estimation and localization" introduces the field of stochastic control theory. The Luenberger observer and the probability theory behind the Kalman filter is taught with several examples of creative applications of Kalman filter theory.

Part IV "System modeling" introduces the basic calculus and physics concepts required to derive the models used in the previous chapters. It walks through the derivations for several common FRC subsystems. Then, methods for system identification are discussed for empirically measuring model parameters.

Part V "Motion planning" covers planning how the robot will get from its current state to some desired state in a manner achievable by its dynamics. It introduces motion profiles with one degree of freedom for simple maneuvers. Trajectory optimization methods are presented for generating profiles with higher degrees of freedom.

The appendices provide further enrichment that isn't required for a passing understanding of the material. This includes derivations for many of the results presented and used in the mainmatter of the book.

The Python scripts used to generate the plots in the case studies double as reference implementations of the techniques discussed in their respective chapters. They are available in this book's Git repository. Its location is listed on the copyright page.

0.3 Ethos of this book

This book is intended as both a tutorial for new students and as a reference manual for more experienced readers who need to review a thing or two. While it isn't comprehensive, the reader will hopefully learn enough to either implement the concepts presented themselves or know where to look for more information.

Some parts are mathematically rigorous, but I believe in giving students a solid theoretical foundation with emphasis on intuition so they can apply it to new problems. To achieve deep understanding of the topics in this book, math is unavoidable. With that said, I try to provide practical and intuitive explanations whenever possible.

Most teaching resources separate linear and nonlinear control with the latter being reserved for a different course. Here, they are introduced together because the concepts

of nonlinear control apply often, and it isn't that much of a leap (if Lyapunov stability isn't included). The control and estimation chapters cover relevant tools for dealing with nonlinearities like linearization when appropriate.

0.4 Mindset of an egoless engineer

Engineering has a mindset, not just a skillset. Engineers have a unique way of approaching problems, and the following maxim summarizes what I hope to teach my robotics students (with examples drawn from controls engineering).

> "Engineer based on requirements, not an ideology."

Engineering is filled with trade-offs. The tools should fit the job, and not every problem is a nail waiting to be struck by a hammer. Instead, assess the minimum requirements (min specs) for a solution to the task at hand and do only enough work to satisfy them; exceeding your specifications is a waste of time and money. If you require performance or maintainability above the min specs, your min specs were chosen incorrectly by definition.

Controls engineering is pragmatic in a similar respect: *solve. the. problem*. For control of nonlinear systems, plant inversion is elegant on paper but doesn't work with an inaccurate model, yet using a theoretically incorrect solution like linear approximations of the nonlinear system works well enough to be used industry-wide. There are more sophisticated controllers than PID, but we use PID anyway for its versatility and simplicity. Sometimes the inferior solutions are more effective or have a more desirable cost-benefit ratio than what the control system designer considers ideal or clean. Choose the tool that is most effective.

Solutions need to be good enough, but do not need to be perfect. We want to avoid integrators as they introduce instability, but we use them anyway because they work well for meeting tracking specifications. One should not blindly defend a design or follow an ideology, because there is always a case where its antithesis is a better option. The engineer should be able to determine when this is the case, set aside their ego, and do what will meet the specifications of their client (e.g., system response characteristics, maintainability, usability). Preferring one solution over another for pragmatic or technical reasons is fine, but the engineer should not care on a personal level which sufficient solution is chosen.

0.5 Request for feedback

While we have tried to write a book that makes the topics of control theory approachable, it still may be dense or fast-paced for some readers (it covers three classes of

feedback control, two of which are for graduate students, in one short book). Please send us feedback, corrections, or suggestions through the GitHub link listed on the copyright page. New examples that demonstrate key concepts and make them more accessible are also appreciated.

Fundamentals of control theory

This page intentionally left blank

Road near walking trail off of Rice Ranch Road in Santa Maria, CA

1. Control system basics

Control systems are all around us and we interact with them daily. A small list of ones you may have seen includes heaters and air conditioners with thermostats, cruise control and the anti-lock braking system (ABS) on automobiles, and fan speed modulation on modern laptops. Control systems monitor or control the behavior of systems like these and may consist of humans controlling them directly (manual control), or of only machines (automatic control).

How can we prove closed-loop controllers on an autonomous car, for example, will behave safely and meet the desired performance specifications in the presence of uncertainty? Control theory is an application of algebra and geometry used to analyze and predict the behavior of systems, make them respond how we want them to, and make them robust to disturbances and uncertainty.

Controls engineering is, put simply, the engineering process applied to control theory. As such, it's more than just applied math. While control theory has some beautiful math behind it, controls engineering is an engineering discipline like any other that is filled with trade-offs. The solutions control theory gives should always be sanity checked and informed by our performance specifications. We don't need to be perfect; we just need to be good enough to meet our specifications.[1]

R Most resources for advanced engineering topics assume a level of knowledge

[1] See section 0.4 for more on engineering.

well above that which is necessary. Part of the problem is the use of jargon. While it efficiently communicates ideas to those within the field, new people who aren't familiar with it are lost. Therefore, it's important to define terms before using them. See the glossary for a list of words and phrases commonly used in control theory, their origins, and their meaning. Links to the glossary are provided for certain words throughout the book and will use this color.

1.1 What is gain?

Gain is a proportional value that shows the relationship between the magnitude of an input signal to the magnitude of an output signal at steady-state. Many systems contain a method by which the gain can be altered, providing more or less "power" to the system.

Figure 1.1 shows a system with a hypothetical input and output. Since the output is twice the amplitude of the input, the system has a gain of 2.

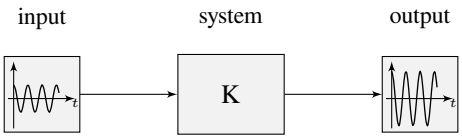


Figure 1.1: Demonstration of system with a gain of $K = 2$

1.2 Block diagrams

When designing or analyzing a control system, it is useful to model it graphically. Block diagrams are used for this purpose. They can be manipulated and simplified systematically (see appendix A). Figure 1.2 is an example of one.

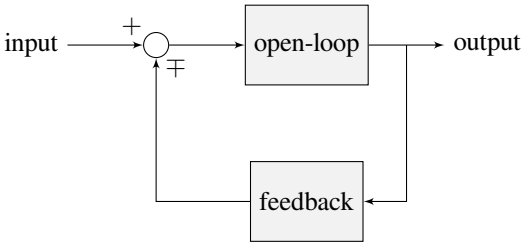


Figure 1.2: Block diagram with nomenclature

The open-loop gain is the total gain from the sum node at the input (the circle) to the output branch. This would be the system's gain if the feedback loop was disconnected. The feedback gain is the total gain from the output back to the input sum node. A sum node's output is the sum of its inputs.

Figure 1.3 is a block diagram with more formal notation in a feedback configuration.

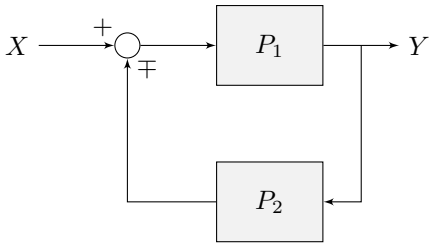


Figure 1.3: Feedback block diagram

$\mp$ means "minus or plus" where a minus represents negative feedback.

1.3 Open-loop and closed-loop systems

The system or collection of actuators being controlled by a control system is called the plant. A controller is used to drive the plant from its current state to some desired state (the reference). We'll be using the following notation for relevant quantities in block diagrams.

$r(t)$	reference	$u(t)$	control input
$e(t)$	error	$y(t)$	output

Controllers which don't include information measured from the plant's output are called *open-loop* or *feedforward* controllers. Figure 1.4 shows a plant with a feedforward controller.

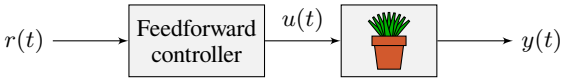


Figure 1.4: Open-loop control system

Controllers which incorporate information fed back from the plant's output are called *closed-loop* or *feedback* controllers. Figure 1.5 shows a plant with a feedback controller.

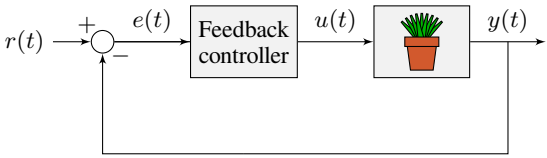


Figure 1.5: Closed-loop control system

Note that the input and output of a system are defined from the plant's point of view. The negative feedback controller shown is driving the difference between the reference and output, also known as the error, to zero.

Figure 1.6 shows a plant with feedforward and feedback controllers.

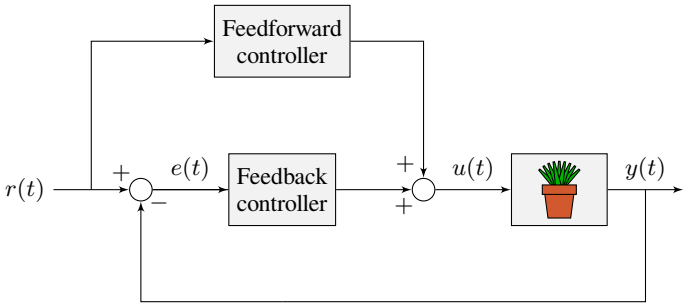


Figure 1.6: Control system with feedforward and feedback

1.4 Feedforward

Feedback control can be effective for reference tracking (making a system's output follow a desired reference signal), but it's a reactionary measure; the system won't start applying control effort until the system is already behind. If we could tell the controller about the desired movement and required input beforehand, the system could react quicker and the feedback controller could do less work. A controller that feeds information forward into the plant like this is called a feedforward controller.

A feedforward controller injects information about the system's dynamics (like a model does) or the desired movement. The feedforward handles parts of the control actions

we already know must be applied to make a system track a reference, then feedback compensates for what we do not or cannot know about the system's behavior at runtime.

There are two types of feedforwards: model-based feedforward and feedforward for unmodeled dynamics. The first solves a mathematical model of the system for the inputs required to meet desired velocities and accelerations. The second compensates for unmodeled forces or behaviors directly so the feedback controller doesn't have to. Both types can facilitate simpler feedback controllers; we'll cover examples of each in later chapters.

1.5 Why feedback control?

Let's say we are controlling a DC motor. With just a mathematical model and knowledge of all current states of the system (i.e., angular velocity), we can predict all future states given the future voltage inputs. Why then do we need feedback control? If the system is disturbed in any way that isn't modeled by our equations, like a load was applied to the armature, or voltage sag in the rest of the circuit caused the commanded voltage to not match the actual applied voltage, the angular velocity of the motor will deviate from the model over time.

To combat this, we can take measurements of the system and the environment to detect this deviation and account for it. For example, we could measure the current position and estimate an angular velocity from it. We can then give the motor corrective commands as well as steer our model back to reality. This feedback allows us to account for uncertainty and be robust to it.

This page intentionally left blank


On trail between McHenry Library and Media Theater at UCSC

2. PID controllers

The PID controller is a commonly used feedback controller consisting of proportional, integral, and derivative terms, hence the name. This chapter will build up the definition of a PID controller term by term while trying to provide intuition for how each of them behaves.

First, we'll get some nomenclature for PID controllers out of the way. The reference is called the setpoint (the desired position) and the output is called the process variable (the measured position). Below are some common variable naming conventions for relevant quantities.

$r(t)$	setpoint	$u(t)$	control input
$e(t)$	error	$y(t)$	output

The error $e(t)$ is $r(t) - y(t)$.

For those already familiar with PID control, this book's interpretation won't be consistent with the classical intuition of "past", "present", and "future" error. We will be approaching it from the viewpoint of modern control theory with proportional controllers applied to different physical quantities we care about. This will provide a more complete explanation of the derivative term's behavior for constant and moving setpoints, and this intuition will carry over to the modern control methods covered later in this book.

The proportional term drives the position error to zero, the derivative term drives the

velocity error to zero, and the integral term accumulates the area between the setpoint and output plots over time (the integral of position error) and adds the current total to the control input. We'll go into more detail on each of these.

2.1 Proportional term

The *Proportional* term drives the position error to zero.

Definition 2.1.1 — Proportional controller.

$$u(t) = K_p e(t) \tag{2.1}$$

where K_p is the proportional gain and $e(t)$ is the error at the current time t.

Figure 2.1 shows a block diagram for a system controlled by a P controller.

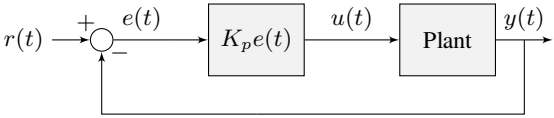


Figure 2.1: P controller block diagram

Proportional gains act like "software-defined springs" that pull the system toward the desired position. Recall from physics that we model springs as $F = -kx$ where F is the force applied, k is a proportional constant, and x is the displacement from the equilibrium point. This can be written another way as $F = k(0 - x)$ where 0 is the equilibrium point. If we let the equilibrium point be our feedback controller's setpoint, the equations have a one-to-one correspondence.

$$\begin{aligned} F &= k(r - x) \\ u(t) &= K_p e(t) = K_p(r(t) - y(t)) \end{aligned}$$

so the "force" with which the proportional controller pulls the system's output toward the setpoint is proportional to the error, just like a spring.

2.2 Derivative term

The *Derivative* term drives the velocity error to zero.

Definition 2.2.1 — PD controller.

$$u(t) = K_p e(t) + K_d \frac{de}{dt} \tag{2.2}$$

where K_p is the proportional gain, K_d is the derivative gain, and $e(t)$ is the error at the current time t.

Figure 2.2 shows a block diagram for a system controlled by a PD controller.

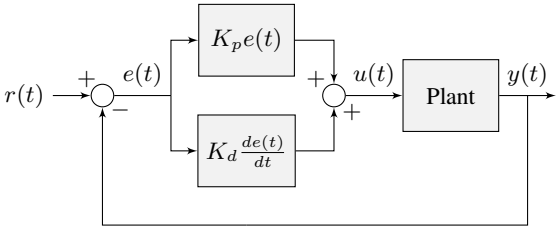


Figure 2.2: PD controller block diagram

A PD controller has a proportional controller for position (K_p) and a proportional controller for velocity (K_d). The velocity setpoint is implicitly provided by how the position setpoint changes over time. Figure 2.3 shows an example for an elevator.

To prove a PD controller is just two proportional controllers, we will rearrange the equation for a PD controller.

$$u_k = K_p e_k + K_d \frac{e_k - e_{k-1}}{\Delta t}$$

where u_k is the control input at timestep k, e_k is the error at timestep k, and Δt is the timestep duration. e_k is defined as $e_k = r_k - y_k$ where r_k is the setpoint at timestep k and y_k is the output at timestep k.

$$\begin{aligned}
u_k &= K_p(r_k - y_k) + K_d \frac{(r_k - y_k) - (r_{k-1} - y_{k-1})}{\Delta t} \\
u_k &= K_p(r_k - y_k) + K_d \frac{r_k - y_k - r_{k-1} + y_{k-1}}{\Delta t} \\
u_k &= K_p(r_k - y_k) + K_d \frac{r_k - r_{k-1} - y_k + y_{k-1}}{\Delta t} \\
u_k &= K_p(r_k - y_k) + K_d \frac{(r_k - r_{k-1}) - (y_k - y_{k-1})}{\Delta t}
\end{aligned}$$

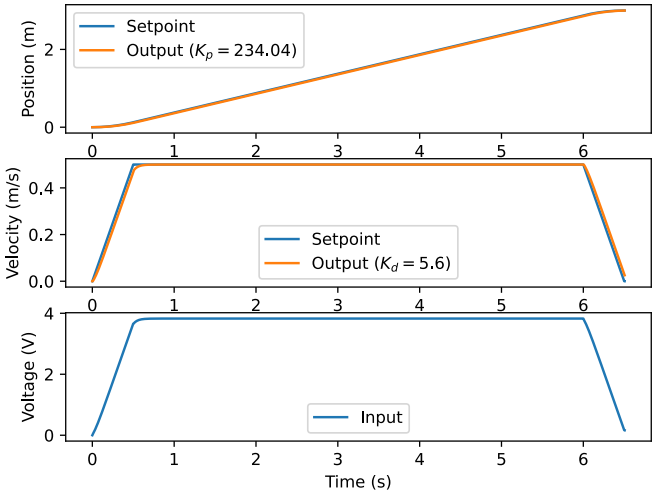


Figure 2.3: PD controller on an elevator

$$u_k = K_p(r_k - y_k) + K_d\left(\frac{r_k - r_{k-1}}{\Delta t} - \frac{y_k - y_{k-1}}{\Delta t}\right)$$

Notice how $\frac{r_k - r_{k-1}}{\Delta t}$ is the velocity of the setpoint and $\frac{y_k - y_{k-1}}{\Delta t}$ is the estimated velocity of the system. This means the K_d term of the PD controller drives the estimated velocity to the setpoint velocity.

If the setpoint is constant, the implicit velocity setpoint is zero, so the K_d term slows the system down if it's moving. This acts like a "software-defined damper". These are commonly seen on door closers, and their damping force increases linearly with velocity.

2.3 Integral term

The *Integral* term accumulates the area between the setpoint and output plots over time (i.e., the integral of position error) and adds the current total to the control input. Accumulating the area between two curves is called integration.

Definition 2.3.1 — PI controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau \tag{2.3}$$

where K_p is the proportional gain, K_i is the integral gain, $e(t)$ is the error at the current time t, and τ is the integration variable.

The integral integrates from time 0 to the current time t. We use τ for the integration because we need a variable to take on multiple values throughout the integral, but we can't use t because we already defined that as the current time.

Figure 2.4 shows a block diagram for a system controlled by a PI controller.

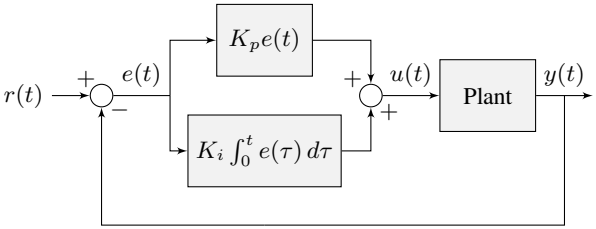


Figure 2.4: PI controller block diagram

When the system is close to the setpoint in steady-state, the proportional term may be too small to pull the output all the way to the setpoint, and the derivative term is zero. This can result in steady-state error, as shown in figure 2.5.

A common way of eliminating steady-state error is to integrate the error and add it to the control input. This increases the control effort until the system converges. Figure 2.5 shows an example of steady-state error for a flywheel, and figure 2.6 shows how an integrator added to the flywheel controller eliminates it. However, too high of an integral gain can lead to overshoot, as shown in figure 2.7.

R There are better approaches to fix steady-state error like using feedforwards or constraining when the integral control acts using other knowledge of the system. We will discuss these in more detail when we get to modern control theory.

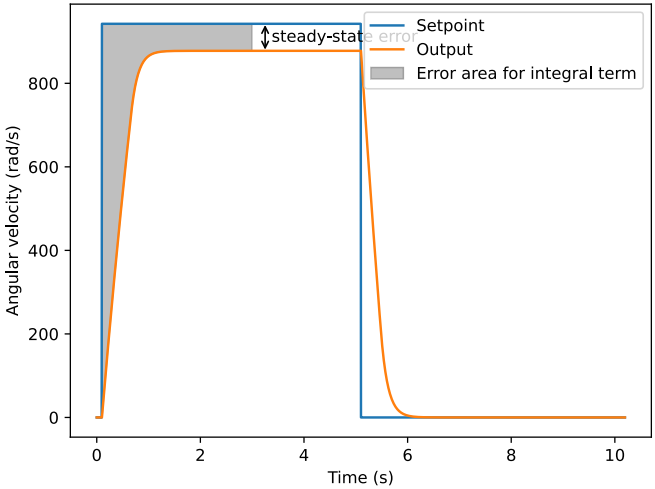


Figure 2.5: P controller on a flywheel with steady-state error

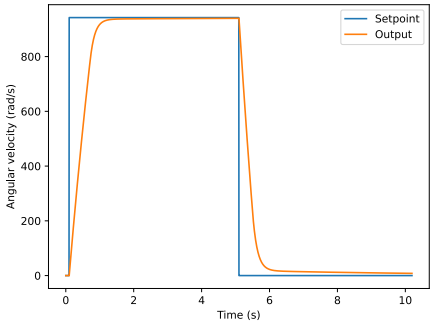


Figure 2.6: PI controller on a flywheel without steady-state error

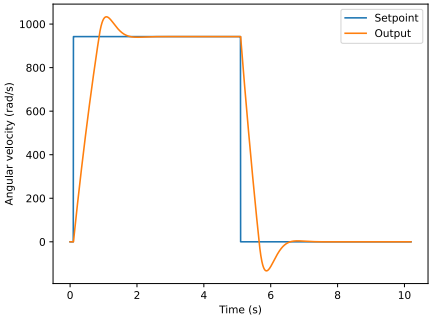


Figure 2.7: PI controller on a flywheel with overshoot from large K_i gain

2.4 PID controller definition

When these three terms are combined, one gets the typical definition for a PID controller.

Definition 2.4.1 — PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt} \tag{2.4}$$

where K_p is the proportional gain, K_i is the integral gain, K_d is the derivative gain, $e(t)$ is the error at the current time t, and τ is the integration variable.

Figure 2.8 shows a block diagram for a system controlled by a PID controller.

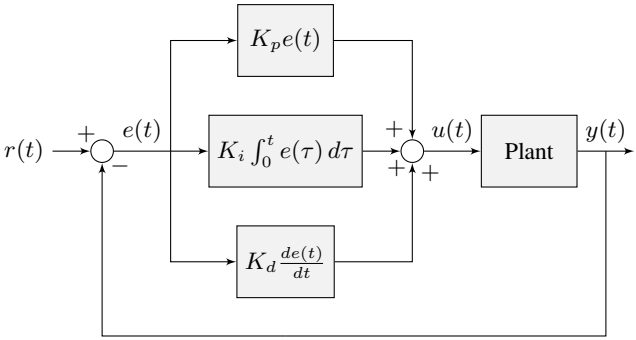


Figure 2.8: PID controller block diagram

2.5 Response types

A system driven by a PID controller generally has three types of responses: underdamped, overdamped, and critically damped. These are shown in figure 2.9.

For the step responses in figure 2.9, rise time is the time the system takes to initially reach the reference after applying the step input. Settling time is the time the system takes to settle at the reference after the step input is applied.

An *underdamped* response oscillates around the reference before settling. An *overdamped* response is slow to rise and does not overshoot the reference. A *critically damped* response has the shortest rise time without oscillating around the reference (i.e., overshooting then undershooting).

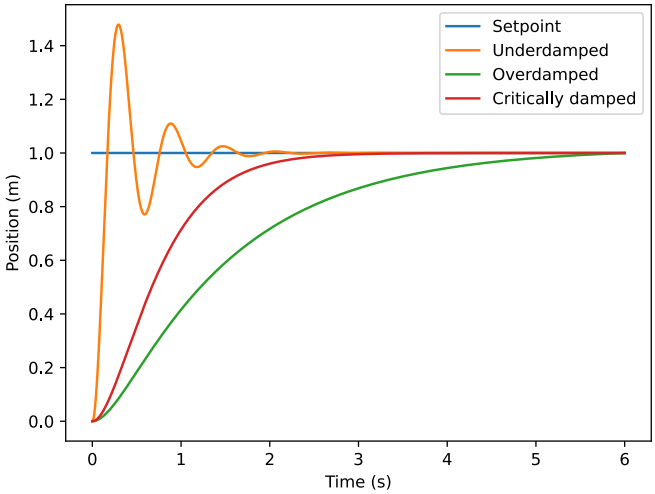


Figure 2.9: PID controller response types

2.6 Manual tuning

These steps apply to position PID controllers. Velocity PID controllers typically don't need K_d.

1. Set K_p, K_i, and K_d to zero.
2. Increase K_p until the output starts to oscillate around the setpoint.
3. Increase K_d as much as possible without introducing jittering in the system response.

If the setpoint follows a trapezoid profile (see section 15.1), tuning becomes a lot easier. Plot the position setpoint, velocity setpoint, measured position, and measured velocity. The velocity setpoint can be obtained via numerical differentiation of the position setpoint (i.e., $v_{desired,k} = \frac{r_k - r_{k-1}}{\Delta t}$). Increase K_p until the position tracks well, then increase K_d until the velocity tracks well.

If the controller settles at an output above or below the setpoint, one can increase K_i such that the controller reaches the setpoint in a reasonable amount of time. However, a steady-state feedforward is strongly preferred over integral control (especially for velocity PID control).

R *Note:* Adding an integral gain to the controller is an incorrect way to eliminate

steady-state error. A better approach would be to tune it with an integrator added to the plant, but this requires a model. Since we are doing output-based rather than model-based control, our only option is to add an integrator to the controller.

Beware that if K_i is too large, integral windup can occur. Following a large change in setpoint, the integral term can accumulate an error larger than the maximal control input. As a result, the system overshoots and continues to increase until this accumulated error is unwound.

2.7 Limitations

PID's heuristic method of tuning is a reasonable choice when there is no *a priori* knowledge of the system dynamics. However, controllers with much better response can be developed if a dynamical model of the system is known. Furthermore, PID only applies to single-input, single-output (SISO) systems; we'll cover methods for multiple-input, multiple-output (MIMO) control in part II of this book.

This page intentionally left blank

Treeline by Crown/Merril bus stop at UCSC

3. Application advice

3.1 Mechanical vs software solutions

While this book focuses on controls engineering applications in FRC, controls isn't necessary to have a successful robot; FRC is primarily a mechanical competition, not a software competition. We spend over six weeks designing and building a robot, but we often allocate just two days for software testing. Despite this, teams still field competitive robots, which is a testament to how simple and easy to use the FRC software ecosystem is nowadays. Focus on a reliable mechanical design first because a good mechanical design can succeed with simple software, but the robot can't succeed with unreliable hardware.

The solution to a design problem may be a tradeoff between mechanical and software complexity. For example, for a mechanism that only needs two positions, a solenoid connected to a pneumatic actuator may take less effort and be more reliable than a motor, rotary encoder, and software feedback control. If one can get the software solution working though, the robot may not need the added space and weight of a compressor and air tanks.

My rule of thumb for evaluating designs is to prefer elegant mechanical solutions over comprehensive software solutions because it's easier to make mechanical solutions reliable in competition. Well-placed sensors can also drastically improve robot performance and reduce driver cognitive load. An example would be a limit switch and match timer for automatically deploying minibots in the 2011 FRC game as soon as

the endgame starts. In many cases, manual processes can be automated later and given a manual fallback if the associated software or sensor fails. Be cautious with designs that require closed-loop control to function.

When should problems be solved in hardware instead of software with clever controls? Controls can handle disturbances like battery voltage drop or measurement noise, but there are limits. For example, there's nothing software can do to work around a drivetrain gearbox seizing up or throwing a chain. Sometimes, you're better off just fixing the root cause in hardware.

Design robot mechanisms for controllability. FRC team 971's "Mechanical Design for Controllability" seminar goes into more detail. Two of the important takeaways from it are:

- Reduce gearbox backlash
- Choose motors and gear reductions that provide adequate control authority

"Spartan Series / Mechanical Design for Controllability" (1 hour, 31 minutes)
Travis Schuh
https://youtu.be/VNfFn-gcfFI

Remember, "fix it in software" isn't always the answer. The remaining chapters of this book assume you've done the engineering analysis and concluded that your chosen design would benefit from more sophisticated controls, and you have the time or expertise to make it work.

3.2 Actuator saturation

A feedback controller calculates its output based on the error between the reference and the current state. Plant in the real world don't have unlimited control authority available for the feedback controller to apply. When the actuator limits are reached, the feedback controller acts as if the gain has been temporarily reduced (i.e., it will have reduced control authority).

We'll try to explain this through a bit of math. Let's say we have a controller $u = k(r - x)$ where u is the control effort, k is the gain, r is the reference, and x is the current state. Let u_{max} be the limit of the actuator's output which is less than the

uncapped value of u and k_{max} be the associated maximum gain. We will now compare the capped and uncapped controllers for the same reference and current state.

$$\begin{aligned} u_{max} &< u \\ k_{max}(r - x) &< k(r - x) \\ k_{max} &< k \end{aligned}$$

For the inequality to hold, k_{max} must be less than the original value for k. This reduced gain is evident in a system response when there is a linear change in state instead of an exponential one as it approaches the reference. This is due to the control effort no longer following a decaying exponential plot. Once the system is closer to the reference, the controller will stop saturating and produce realistic controller values again.

This page intentionally left blank

4. Calculus

This book uses derivatives and integrals occasionally to represent small changes in values over small changes in time and the infinitesimal sum of values over time respectively. The formulas and tables presented here are all you'll need to carry through with the math in later chapters.

If you are interested in more information after reading this chapter, 3Blue1Brown does a fantastic job of introducing them in his *Essence of calculus* series. We recommend reading lesson chapters 1 through 3, 5, and 10 through 13 for a solid foundation.

"Essence of calculus"
3Blue1Brown
https://www.3blue1brown.com/topics/calculus

4.1 Derivatives

Derivatives are expressions for the slope of a curve at arbitrary points. Common notations for this operation on a function like $f(x)$ include

Leibniz notation	Lagrange notation
$\frac{d}{dx} f(x)$	$f'(x)$
$\frac{d^2}{dx^2} f(x)$	$f''(x)$
$\frac{d^3}{dx^3} f(x)$	$f'''(x)$
$\frac{d^4}{dx^4} f(x)$	$f^{(4)}(x)$
$\frac{d^n}{dx^n} f(x)$	$f^{(n)}(x)$

Table 4.1: Notation for derivatives of $f(x)$

Lagrange notation is usually voiced as “f prime of x”, “f double-prime of x”, etc.

4.1.1 Power rule

$$f(x) = x^n$$
$$f'(x) = nx^{n-1}$$

4.1.2 Product rule

This is for taking the derivative of the product of two expressions.

$$h(x) = f(x)g(x)$$
$$h'(x) = f'(x)g(x) + f(x)g'(x)$$

4.1.3 Chain rule

This is for taking the derivative of nested expressions.

$$h(x) = f(g(x))$$
$$h'(x) = f'(g(x)) \cdot g'(x)$$

For example,

$$h(x) = (3x + 2)^5$$
$$h'(x) = 5\,(3x + 2)^4 \cdot (3)$$
$$h'(x) = 15\,(3x + 2)^4$$

4.1.4 Partial derivatives

A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant (as opposed to the total derivative, in which all variables are allowed to vary). Partial derivatives use ∂ instead of d in Leibniz notation.

For example, let $h(x, y) = 3xy + 2x$. For the partial derivative with respect to x, y is treated as a constant.

$$\frac{\partial h(x, y)}{\partial x} = 3y + 2$$

For the partial derivative with respect to y, x is treated as a constant, so the second term becomes zero.

$$\frac{\partial h(x, y)}{\partial y} = 3x$$

4.2 Integrals

The integral is the inverse operation of the derivative and calculates the area under a curve. Here is an example of one based on table 4.2.

$$\int e^{at}\, dt$$

$$\frac{1}{a}e^{at} + C$$

The arbitrary constant C is needed because when you take a derivative, constants are discarded because vertical offsets don't affect the slope. When performing the inverse operation, we don't have enough information to determine the constant.

However, we can provide bounds for the integration.

$$\int_0^t e^{at}\, dt$$

$$\left(\frac{1}{a}e^{at} + C\right)\Bigg|_0^t$$

$$\left(\frac{1}{a}e^{at} + C\right) - \left(\frac{1}{a}e^{a\cdot 0} + C\right)$$

$$\left(\frac{1}{a}e^{at} + C\right) - \left(\frac{1}{a} + C\right)$$

$$\frac{1}{a}e^{at} + C - \frac{1}{a} - C$$

$$\frac{1}{a}e^{at} - \frac{1}{a}$$

When we do this, the constant cancels out.

4.3 Change of variables

Change of variables is a technique for simlifying problems in which expressions are replaced with new variables to make the problem more tractible. This can mean either the problem is more straightforward or it matches a common form for which tools for finding solutions are readily available. Here's an example of integration which utilizes it.

$$\int \cos \omega t \, dt$$

Let $u = \omega t$.

$$du = \omega \, dt$$
$$dt = \frac{1}{\omega} \, du$$

Now substitute the expressions for u and dt in.

$$\int \cos u \frac{1}{\omega} \, du$$
$$\frac{1}{\omega} \int \cos u \, du$$
$$\frac{1}{\omega} \sin u + C$$
$$\frac{1}{\omega} \sin \omega t + C$$

Another example, which will be relevant when we actually cover state-space notation ($\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$), is a closed-loop state-space system.

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r}$$
$$\dot{\mathbf{x}} = \mathbf{A}_{cl}\mathbf{x} + \mathbf{B}_{cl}\mathbf{u}_{cl}$$

where $\mathbf{A}_{cl} = \mathbf{A} - \mathbf{B}\mathbf{K}$, $\mathbf{B}_{cl} = \mathbf{B}\mathbf{K}$, and $\mathbf{u}_{cl} = \mathbf{r}$. Since it matches the form of the open-loop system, all the same analysis tools will work with it.

4.4 Tables

4.4.1 Common derivatives and integrals

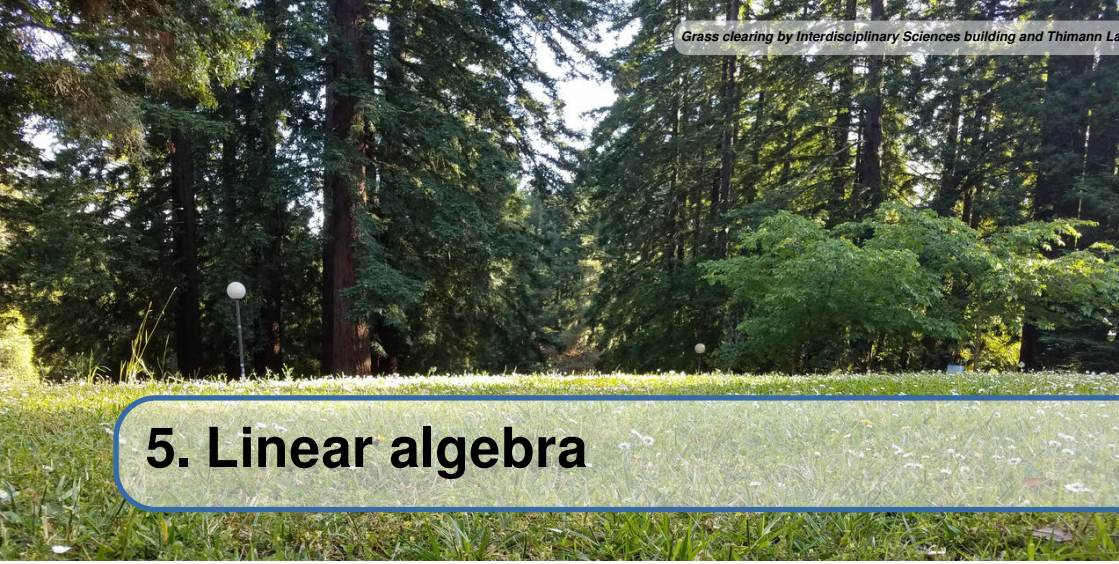
$\int f(x)\,dx$	$f(x)$	$f'(x)$
ax	a	0
$\frac{1}{2}ax^2$	ax	a
$\frac{1}{a+1}x^{a+1}$	x^a	ax^{a-1}
$\frac{1}{a}e^{ax}$	e^{ax}	ae^{ax}
$-\cos(x)$	$\sin(x)$	$\cos(x)$
$\sin(x)$	$\cos(x)$	$-\sin(x)$
$\cos(x)$	$-\sin(x)$	$-\cos(x)$
$-\sin(x)$	$-\cos(x)$	$\sin(x)$

Table 4.2: Common derivatives and integrals

This page intentionally left blank

II Modern control theory

This page intentionally left blank



5. Linear algebra

Modern control theory borrows concepts from linear algebra. At first, linear algebra may appear very abstract, but there are simple geometric intuitions underlying it. We'll be linking to 3Blue1Brown's *Essence of linear algebra* video series because it's better at conveying that intuition than static text.

"Essence of linear algebra preview" (5 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/eola-preview

5.1 Vectors

"Vectors, what even are they?" (5 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/vectors

5.2 Linear combinations, span, and basis vectors

"Linear combination, span, and basis vectors" (10 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/span

5.3 Linear transformations and matrices

"Linear transformations and matrices" (11 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/linear-transformations

5.4 Matrix multiplication

"Matrix multiplication as composition" (10 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/matrix-multiplication

5.5 The determinant

"The determinant" (10 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/determinant

5.6 Inverse matrices, column space, and null space

"Inverse matrices, column space, and null space" (12 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/inverse-matrices

5.7 Nonsquare matrices

"Nonsquare matrices as transformations between dimensions" (4 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/nonsquare-matrices

5.8 Eigenvectors and eigenvalues

"Eigenvectors and eigenvalues" (17 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/eigenvalues

5.9 Miscellaneous notation and operators

This book works with two-dimensional matrices in the sense that they only have rows and columns. The dimensionality of these matrices is specified by row first, then column. For example, a matrix with two rows and three columns would be a two-by-three matrix. A square matrix has the same number of rows as columns. Matrices commonly use capital letters while vectors use lowercase letters.

5.9.1 Special constant matrices

$\mathbf{I}$ is the identity matrix, a typically square matrix with ones along its diagonal and zeroes elsewhere. $\mathbf{0}$ is a matrix filled with zeroes and $\mathbf{1}$ is a matrix filled with ones. An optional subscript $m \times n$ denotes the matrix having m rows and n columns.

$$\mathbf{I}_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{0}_{3\times2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{1}_{3\times2} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

5.9.2 Operators

Transpose

The $^\mathsf{T}$ in $\mathbf{A}^\mathsf{T}$ denotes transpose, which flips the matrix across its diagonal such that the rows become columns and vice versa.

A symmetric matrix is equal to its transpose.

Pseudoinverse

The $^+$ in $\mathbf{A}^+$ denotes the Moore-Penrose pseudoinverse, which is a generalization of the inverse matrix to nonsquare matrices. The pseudoinverse is an approximate inverse in the least-squares sense, so it can solve least-squares problems of the form $\mathbf{Ax} = \mathbf{b}$ via $\mathbf{x} = \mathbf{A}^+\mathbf{b}$.

System type	Shape of A	Name of $\mathbf{A}^+$	Value of $\mathbf{A}^+$
Well-determined	Square	Inverse	$\mathbf{A}^{-1}$
Overdetermined	Tall	Left pseudoinverse	$(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}$
Underdetermined	Wide	Right pseudoinverse	$\mathbf{A}^\mathsf{T}(\mathbf{A}\mathbf{A}^\mathsf{T})^{-1}$

Table 5.1: Pseudoinverses by system type.

Diagonal

A diagonal matrix has elements along its diagonal and zeroes elsewhere (e.g., the identity matrix). Let $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\mathsf{T}$.

$$\operatorname{diag}(\mathbf{x}) = \operatorname{diag}(x_1, \dots, x_n) = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & x_n \end{bmatrix}$$

A block diagonal matrix has matrices along its diagonal. $\operatorname{diag}()$ works similarly for constructing one. Let $\mathbf{A} = \left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$ and $\mathbf{B} = \left[\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{smallmatrix}\right]$.

$$\operatorname{diag}(\mathbf{A}, \mathbf{B}) = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 & 6 \\ 0 & 0 & 7 & 8 & 9 \end{bmatrix}$$

Operations on the $\mathrm{diag}()$ argument are applied element-wise.

$$\mathrm{diag}\left(\frac{1}{\mathbf{x}^2}\right) = \begin{bmatrix} \frac{1}{x_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{x_n^2} \end{bmatrix}$$

Trace

$\mathrm{tr}(\mathbf{A})$ denotes the trace of the square matrix $\mathbf{A}$, which is defined as the sum of the elements on the main diagonal (top-left to bottom-right).

5.10 Matrix definiteness

Type	Eigenvalues	Notation
Negative definite	All $\lambda < 0$	$\mathbf{M} < \mathbf{0}$
Negative semidefinite	All $\lambda \leq 0$	$\mathbf{M} \leq \mathbf{0}$
Positive semidefinite	All $\lambda \geq 0$	$\mathbf{M} \geq \mathbf{0}$
Positive definite	All $\lambda > 0$	$\mathbf{M} > \mathbf{0}$
Indefinite	Positive and negative	N/A

Table 5.2: Eigenvalue distribution and notation for each type of matrix definiteness. Let $\mathbf{M}$ be a matrix.

Type	Definition
Negative definite	$\mathbf{x}^T\mathbf{M}\mathbf{x} < 0$ for all $\mathbf{x}$
Negative semidefinite	$\mathbf{x}^T\mathbf{M}\mathbf{x} \leq 0$ for all $\mathbf{x}$
Positive semidefinite	$\mathbf{x}^T\mathbf{M}\mathbf{x} \geq 0$ for all $\mathbf{x}$
Positive definite	$\mathbf{x}^T\mathbf{M}\mathbf{x} > 0$ for all $\mathbf{x}$
Indefinite	$\exists \mathbf{x}, \mathbf{y} \ni \mathbf{x}^T\mathbf{M}\mathbf{x} < 0 < \mathbf{y}^T\mathbf{M}\mathbf{y}$

Table 5.3: Rigorous definition for each type of matrix definiteness. Let $\mathbf{M}$ be a matrix and let $\mathbf{x}$ and $\mathbf{y}$ be nonzero vectors.

5.11 Common control theory matrix equations

Here's some common matrix equations from control theory we'll use later on. Solvers for them exist in Drake (C++) and SciPy (Python).

5.11.1 Continuous algebraic Riccati equation (CARE)

The continuous algebraic Riccati equation (CARE) appears in the solution to the continuous time LQ problem.

$$\mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{A} - \mathbf{X}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{X} + \mathbf{Q} = 0 \tag{5.1}$$

5.11.2 Discrete algebraic Riccati equation (DARE)

The discrete algebraic Riccati equation (DARE) appears in the solution to the discrete time LQ problem.

$$\mathbf{X} = \mathbf{A}^T\mathbf{X}\mathbf{A} - (\mathbf{A}^T\mathbf{X}\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{X}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{X}\mathbf{A} + \mathbf{Q} \tag{5.2}$$

Snippet 5.1 computes the unique stabilizing solution to the discrete algebraic Riccati equation. A robust implementation should also enforce the following preconditions:

1. $\mathbf{Q} = \mathbf{Q}^T \geq \mathbf{0}$ and $\mathbf{R} = \mathbf{R}^T > \mathbf{0}$.
2. $(\mathbf{A}, \mathbf{B})$ is a stabilizable pair (see subsection 6.12.4).
3. $(\mathbf{A}, \mathbf{C})$ is a detectable pair where $\mathbf{Q} = \mathbf{C}^T\mathbf{C}$ (see section 6.12.8).

```
#include <Eigen/Cholesky>
#include <Eigen/Core>
#include <Eigen/LU>

template <int States, int Inputs>
```

```
Eigen::Matrix<double, States, States> DARE(
    const Eigen::Matrix<double, States, States>& A,
    const Eigen::Matrix<double, States, Inputs>& B,
    const Eigen::Matrix<double, States, States>& Q,
    const Eigen::Matrix<double, Inputs, Inputs>& R) {
  // [1] E. K.-W. Chu, H.-Y. Fan, W.-W. Lin & C.-S. Wang
  //     "Structure-Preserving Algorithms for Periodic Discrete-Time
  //     Algebraic Riccati Equations",
  //     International Journal of Control, 77:8, 767-788, 2004.
  //     DOI: 10.1080/00207170410001714988
  //
  // Implements SDA algorithm on p. 5 of [1] (initial A, G, H are from (4)).
  using StateMatrix = Eigen::Matrix<double, States, States>;

  StateMatrix A_k = A;
  StateMatrix G_k = B * R.llt().solve(B.transpose());
  StateMatrix H_k;
  StateMatrix H_k1 = Q;

  do {
    H_k = H_k1;

    StateMatrix W = StateMatrix::Identity() + G_k * H_k;

    auto W_solver = W.lu();
    StateMatrix V_1 = W_solver.solve(A_k);
    StateMatrix V_2 = W_solver.solve(G_k.transpose()).transpose();

    G_k += A_k * V_2 * A_k.transpose();
    H_k1 = H_k + V_1.transpose() * H_k * A_k;
    A_k *= V_1;
  } while ((H_k1 - H_k).norm() > 1e-10 * H_k1.norm());

  return H_k1;
}
```

Snippet 5.1. Discrete algebraic Riccati equation solver in C++

5.11.3 Continuous Lyapunov equation

The continuous Lyapunov equation appears in controllability/observability analysis of continuous time systems.

$$\mathbf{AX} + \mathbf{XA}^\mathsf{T} + \mathbf{Q} = 0 \tag{5.3}$$

5.11.4 Discrete Lyapunov equation

The discrete Lyapunov equation appears in controllability/observability analysis of discrete time systems.

$$\mathbf{AXA}^\mathsf{T} - \mathbf{X} + \mathbf{Q} = 0 \tag{5.4}$$

5.12 Matrix calculus

Matrix calculus uses partial derivatives. See subsection 4.1.4 for how they work.

We'll need a vector-valued function to demonstrate some common operations in matrix calculus. Let $\mathbf{f}(\mathbf{x}, \mathbf{u})$ be a vector-valued function defined as

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} f_1(\mathbf{x}, \mathbf{u}) \\ \vdots \\ f_m(\mathbf{x}, \mathbf{u}) \end{bmatrix} \text{ where } \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

5.12.1 Jacobian

The Jacobian is the first-order partial derivative of a vector-valued function with respect to one of its vector arguments. The columns of the Jacobian of $\mathbf{f}$ are filled with partial derivatives of $\mathbf{f}$'s rows with respect to each of the argument's elements. For example, the Jacobian of $\mathbf{f}$ with respect to $\mathbf{x}$ is

$$\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial x_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix}$$

$\frac{\partial f_1}{\partial x_1}$ is the partial derivative of the first row of $\mathbf{f}$ with respect to the first row of $\mathbf{x}$, and so on for all rows of $\mathbf{f}$ and $\mathbf{x}$. This has m^2 permutations and thus produces a square matrix.

The Jacobian of $\mathbf{f}$ with respect to $\mathbf{u}$ is

$$\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial u_1} & \cdots & \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial u_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial u_1} & \cdots & \frac{\partial f_m}{\partial u_n} \end{bmatrix}$$

$\frac{\partial f_1}{\partial u_1}$ is the partial derivative of the first row of $\mathbf{f}$ with respect to the first row of $\mathbf{u}$, and so on for all rows of $\mathbf{f}$ and $\mathbf{u}$. This has $m \times n$ permutations and can produce a nonsquare matrix if $m \neq n$.

5.12.2 Hessian

The Hessian is the second-order partial derivative of a vector-valued function with respect to one of its vector arguments. For example, the Hessian of $\mathbf{f}$ with respect to $\mathbf{x}$ is

$$\frac{\partial^2 \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}^2} = \begin{bmatrix} \frac{\partial^2 \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial x_1^2} & \cdots & \frac{\partial^2 \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial x_m^2} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f_1}{\partial x_1^2} & \cdots & \frac{\partial^2 f_1}{\partial x_m^2} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_m}{\partial x_1^2} & \cdots & \frac{\partial^2 f_m}{\partial x_m^2} \end{bmatrix}$$

and the Hessian of $\mathbf{f}$ with respect to $\mathbf{u}$ is

$$\frac{\partial^2 \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}^2} = \begin{bmatrix} \frac{\partial^2 \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial u_1^2} & \cdots & \frac{\partial^2 \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial u_n^2} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f_1}{\partial u_1^2} & \cdots & \frac{\partial^2 f_1}{\partial u_n^2} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_m}{\partial u_1^2} & \cdots & \frac{\partial^2 f_m}{\partial u_n^2} \end{bmatrix}$$

5.12.3 Useful identities

Here's some useful matrix calculus identities pulled from Wikipedia's table [3].

Theorem 5.12.1 $\frac{\partial \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is symmetric.

Theorem 5.12.2 $\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e})}{\partial \mathbf{x}} = \mathbf{A}^\mathsf{T}\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e}) + \mathbf{D}^\mathsf{T}\mathbf{C}^\mathsf{T}(\mathbf{A}\mathbf{x}+\mathbf{b})$

Corollary 5.12.3 $\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = 2\mathbf{A}^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})$ where $\mathbf{C}$ is symmetric.

Proof:

$$\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = \mathbf{A}^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b}) + \mathbf{A}^\mathsf{T}\mathbf{C}^\mathsf{T}(\mathbf{A}\mathbf{x}+\mathbf{b})$$
$$\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = (\mathbf{A}^\mathsf{T}\mathbf{C} + \mathbf{A}^\mathsf{T}\mathbf{C}^\mathsf{T})(\mathbf{A}\mathbf{x}+\mathbf{b})$$

$\mathbf{C}$ is symmetric, so

$$\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = (\mathbf{A}^\mathsf{T}\mathbf{C} + \mathbf{A}^\mathsf{T}\mathbf{C})(\mathbf{A}\mathbf{x}+\mathbf{b})$$
$$\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = 2\mathbf{A}^\mathsf{T}\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})$$

Night sky above Dufour Street in Santa Cruz, CA

6. Continuous state-space control

When we want to command a system to a set of states, we design a controller with certain control laws to do it. PID controllers use the system outputs with proportional, integral, and derivative control laws. In state-space, we also have knowledge of the system states so we can do better.

Modern control theory uses state-space representation to model and control systems. State-space representation models systems as a set of state, input, and output variables related by first-order differential equations that describe how the system's state changes over time given the current states and inputs.

6.1 From PID control to model-based control

As mentioned before, controls engineers have a more general framework to describe control theory than just PID control. PID controller designers are focused on fiddling with controller parameters relating to the current, past, and future error rather than the underlying system states. Integral control is a commonly used tool, and some people use integral action as the majority of the control action. While this approach works in a lot of situations, it is an incomplete view of the world.

Model-based control has a completely different mindset. Controls designers using model-based control care about developing an accurate model of the system, then driving the states they care about to zero (or to a reference). Integral control is added with u_{error} estimation if needed to handle model uncertainty, but we prefer not to use it be-

cause its response is hard to tune and some of its destabilizing dynamics aren't visible during simulation.

Why use model-based control in FRC? Poor build season schedule management often leads to the software team:

1. Not getting enough time to verify basic functionality and test/tune feedback controllers.
2. Spending dedicated software testing time troubleshooting mechanical/electrical issues within recently integrated subsystems instead.

Model-based control (one of the focuses of this book) avoids both problems because it lets software teams test basic functionality in simulation much earlier in the build season and tune their feedback controllers automatically.

6.2 What is a dynamical system?

A dynamical system is a system whose motion varies according to a set of differential equations. A dynamical system is considered *linear* if the differential equations describing its dynamics consist only of linear operators. Linear operators are things like constant gain multiplications, derivatives, and integrals. You can define reasonably accurate linear models for pretty much everything you'll see in FRC with just those relations.

But let's say you have a DC motor hooked up to a power supply and you applied a constant voltage to it from rest. The motor approaches a steady-state angular velocity, but the shape of the angular velocity curve over time isn't a line. In fact, it's a decaying exponential curve akin to

$$\omega = \omega_{max}\left(1 - e^{-t}\right)$$

where ω is the angular velocity and ω_{max} is the maximum angular velocity. If DC motors are said to behave linearly, then why is this?

Linearity refers to a system's equations of motion, not its time domain response. The equation defining the motor's change in angular velocity over time looks like

$$\dot{\omega} = -a\omega + bV$$

where $\dot{\omega}$ is the derivative of ω with respect to time, V is the input voltage, and a and b are constants specific to the motor. This equation, unlike the one shown before, is actually linear because it only consists of multiplications and additions relating the input V and current state ω.

Also of note is that the relation between the input voltage and the angular velocity of the output shaft is a linear regression. You'll see why if you model a DC motor as a voltage source and generator producing back-EMF (in the equation above, bV corresponds to the voltage source and $-a\omega$ corresponds to the back-EMF). As you increase the input voltage, the back-EMF increases linearly with the motor's angular velocity. If there was a friction term that varied with the angular velocity squared (air resistance is one example), the relation from input to output would be a curve. Friction that scales with just the angular velocity would result in a lower maximum angular velocity, but because that term can be lumped into the back-EMF term, the response is still linear.

6.3 Continuous state-space notation

6.3.1 What is state-space?

Recall from last chapter that 2D space has two axes: x and y. We represent locations within this space as a pair of numbers packaged in a vector, and each coordinate is a measure of how far to move along the corresponding axis. State-space is a Cartesian coordinate system with an axis for each state variable, and we represent locations within it the same way we do for 2D space: with a list of numbers in a vector. Each element in the vector corresponds to a state of the system.

In state-space notation, there are also input and output vectors with corresponding input and output spaces. Inputs drive the system to other points in the state-space, and outputs are sensor measurements that have a linear relationship to the state and input. Since the mapping from state and input to change in state is a system of equations, it's natural to write it in matrix form.

6.3.2 Definition

Below is the continuous version of state-space notation.

Definition 6.3.1 — Continuous state-space notation.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{6.1}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{6.2}$$

$\mathbf{A}$	system matrix	$\mathbf{x}$	state vector
$\mathbf{B}$	input matrix	$\mathbf{u}$	input vector
$\mathbf{C}$	output matrix	$\mathbf{y}$	output vector
$\mathbf{D}$	feedthrough matrix		

Matrix	Rows × Columns	Matrix	Rows × Columns
$\mathbf{A}$	states × states	$\mathbf{x}$	states × 1
$\mathbf{B}$	states × inputs	$\mathbf{u}$	inputs × 1
$\mathbf{C}$	outputs × states	$\mathbf{y}$	outputs × 1
$\mathbf{D}$	outputs × inputs		

Table 6.1: State-space matrix dimensions

The change in state and the output are linear combinations of the state vector and the input vector. The $\mathbf{A}$ and $\mathbf{B}$ matrices are used to map the state vector $\mathbf{x}$ and the input vector $\mathbf{u}$ to a change in the state vector $\dot{\mathbf{x}}$. The $\mathbf{C}$ and $\mathbf{D}$ matrices are used to map the state vector $\mathbf{x}$ and the input vector $\mathbf{u}$ to an output vector $\mathbf{y}$.

6.4 Eigenvalues and stability

If a system is stable, its output will tend toward equilibrium (steady-state) over time. For a general system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, equilibrium points are points where $\dot{\mathbf{x}} = \mathbf{0}$. If we let $\mathbf{x} = \mathbf{0}$ and $\mathbf{u} = \mathbf{0}$ in $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, we can see that $\dot{\mathbf{x}} = \mathbf{0}$, so $\mathbf{x} = \mathbf{0}$ is an equilibrium point.

We'd like to know whether all possible unforced system trajectories ($\mathbf{u} = \mathbf{0}$) move toward or away from the equilibrium point. If we solve the system of linear differential equations $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, we get $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$.[1] $e^{\mathbf{A}t}$ is the superposition of $e^{\lambda_j t}$ terms where $\{\lambda_j\}$ is the set of $\mathbf{A}$'s eigenvalues.[2]

For now, let's consider when all the eigenvalues are real numbers.

$$\begin{cases} \lambda_j < 0, & e^{\lambda_j t} \text{ decays to zero (stable)} \\ \lambda_j = 0, & e^{\lambda_j t} = 1 \text{ (marginally stable)} \\ \lambda_j > 0, & e^{\lambda_j t} \text{ grows to infinity (unstable)} \end{cases}$$

So the system tends toward the equilibrium point (i.e., it's stable) if $\lambda_j < 0$ for all j.

Now let's consider when the eigenvalues are complex numbers. What does that mean for the system response? Let $\lambda_j = a_j + b_j i$. Each of the exponential terms in the solution can be written as

$$e^{\lambda_j t} = e^{(a_j + b_j i)t} = e^{a_j t} e^{i b_j t}$$

[1] Section 7.3 will explain why the matrix exponential $e^{\mathbf{A}t}$ shows up here.

[2] We're handwaving why this is the case, but it's a consequence of $e^{\mathbf{A}t}$ being diagonalizable.

The complex exponential can be rewritten using Euler's formula.[3]

$$e^{ib_j t} = \cos(b_j t) + i \sin(b_j t)$$

Therefore,

$$e^{\lambda_j t} = e^{a_j t}(\cos(b_j t) + i \sin(b_j t))$$

When the eigenvalue's imaginary part $b_j \neq 0$, it contributes oscillation to the real part's response.

The eigenvalues of $\mathbf{A}$ are called *poles*.[4] Figure 6.1 shows the impulse responses in the time domain for systems with various pole locations in the complex plane (real numbers on the x-axis and imaginary numbers on the y-axis). Each response has an initial condition of 1.

Poles in the left half-plane (LHP) are stable; the system's output may oscillate but it converges to steady-state. Poles on the imaginary axis are marginally stable; the system's output oscillates at a constant amplitude forever. Poles in the right half-plane (RHP) are unstable; the system's output grows without bound.

R Imaginary poles always come in complex conjugate pairs (e.g., $-2 + 3i$, $-2 - 3i$).

[3]Euler's formula may seem surprising at first, but it's rooted in the fact that complex exponentials are rotations in the complex plane around the origin. If you can imagine walking around the unit circle traced by that rotation, you'll notice the real part of your position oscillates between -1 and 1 over time. That is, complex exponentials manifest as oscillations in real life.

"What is Euler's formula actually saying? | Ep. 4 Lockdown live math" (51 minutes)
3Blue1Brown
https://youtu.be/ZxYOEwM6Wbk

[4]This name comes from classical control theory. See subsection E.2.2 for more.

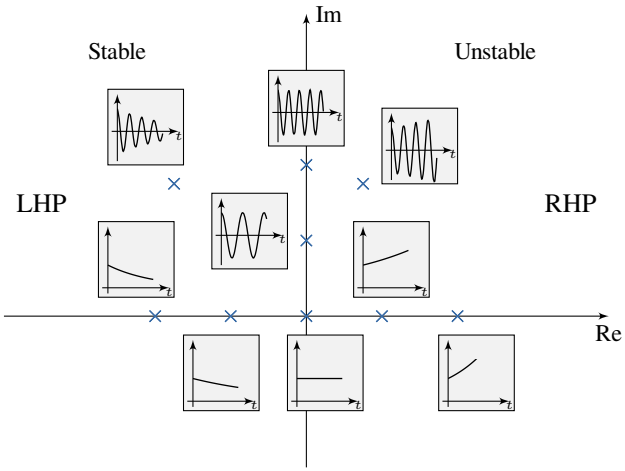


Figure 6.1: Continuous impulse response vs pole location

6.5 Closed-loop controller

With the control law $\mathbf{u} = \mathbf{K}(\mathbf{r}-\mathbf{x})$, we can derive the closed-loop state-space equations. We'll discuss where this control law comes from in subsection 7.7.

First is the state update equation. Substitute the control law into equation (6.1).

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{K}(\mathbf{r} - \mathbf{x}) \\ \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r} - \mathbf{B}\mathbf{K}\mathbf{x} \\ \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r}\end{aligned} \tag{6.3}$$

Now for the output equation. Substitute the control law into equation (6.2).

$$\begin{aligned}\mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}(\mathbf{K}(\mathbf{r} - \mathbf{x})) \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{K}\mathbf{r} - \mathbf{D}\mathbf{K}\mathbf{x} \\ \mathbf{y} &= (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x} + \mathbf{D}\mathbf{K}\mathbf{r}\end{aligned} \tag{6.4}$$

Instead of commanding the system to a state using the vector $\mathbf{u}$ directly, we can now specify a vector of desired states through $\mathbf{r}$ and the controller will choose values of $\mathbf{u}$ for us over time that drive the system toward the reference.

The eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ are the poles of the closed-loop system. Therefore, the rate of convergence and stability of the closed-loop system can be changed by moving

the poles via the eigenvalues of $\mathbf{A} - \mathbf{BK}$. $\mathbf{A}$ and $\mathbf{B}$ are inherent to the system, but $\mathbf{K}$ can be chosen arbitrarily by the controller designer. For equation (6.3) to reach steady-state, the eigenvalues of $\mathbf{A} - \mathbf{BK}$ must be in the left-half plane. There will be steady-state error if $\mathbf{Ar} \neq \mathbf{0}$.

Symbol	Name	Rows × Columns
$\mathbf{A}$	system matrix	states × states
$\mathbf{B}$	input matrix	states × inputs
$\mathbf{C}$	output matrix	outputs × states
$\mathbf{D}$	feedthrough matrix	outputs × inputs
$\mathbf{K}$	controller gain matrix	inputs × states
$\mathbf{r}$	reference vector	states × 1
$\mathbf{x}$	state vector	states × 1
$\mathbf{u}$	input vector	inputs × 1
$\mathbf{y}$	output vector	outputs × 1

Table 6.2: Controller matrix dimensions

6.6 Model augmentation

This section will teach various tricks for manipulating state-space models with the goal of demystifying the matrix algebra at play. We will use the augmentation techniques discussed here in the section on integral control.

Matrix augmentation is the process of appending rows or columns to a matrix. In state-space, there are several common types of augmentation used: plant augmentation, controller augmentation, and observer augmentation.

6.6.1 Plant augmentation

Plant augmentation is the process of adding a state to a model's state vector and adding a corresponding row to the $\mathbf{A}$ and $\mathbf{B}$ matrices.

6.6.2 Controller augmentation

Controller augmentation is the process of adding a column to a controller's $\mathbf{K}$ matrix. This is often done in combination with plant augmentation to add controller dynamics

relating to a newly added state.

6.6.3 Observer augmentation

Observer augmentation is closely related to plant augmentation. In addition to adding entries to the observer matrix $\mathbf{K}$,[5] the observer is using this augmented plant for estimation purposes. This is better explained with an example.

By augmenting the plant with a bias term with no dynamics (represented by zeroes in its rows in $\mathbf{A}$ and $\mathbf{B}$), the observer will attempt to estimate a value for this bias term that makes the model best reflect the measurements taken of the real system. Note that we're not collecting any data on this bias term directly; it's what's known as a hidden state. Rather than our inputs and other states affecting it directly, the observer determines a value for it based on what is most likely given the model and current measurements. We just tell the plant what kind of dynamics the term has and the observer will estimate it for us.

6.6.4 Output augmentation

Output augmentation is the process of adding rows to the $\mathbf{C}$ matrix. This is done to help the controls designer visualize the behavior of internal states or other aspects of the system in MATLAB or Python Control. $\mathbf{C}$ matrix augmentation doesn't affect state feedback, so the designer has a lot of freedom here. Noting that the output is defined as $\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$, the following row augmentations of $\mathbf{C}$ may prove useful. Of course, $\mathbf{D}$ needs to be augmented with zeroes as well in these cases to maintain the correct matrix dimensionality.

Since $\mathbf{u} = -\mathbf{K}\mathbf{x}$, augmenting $\mathbf{C}$ with $-\mathbf{K}$ makes the observer estimate the control input $\mathbf{u}$ applied.

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$
$$\begin{bmatrix}\mathbf{y} \\ \mathbf{u}\end{bmatrix} = \begin{bmatrix}\mathbf{C} \\ -\mathbf{K}\end{bmatrix}\mathbf{x} + \begin{bmatrix}\mathbf{D} \\ \mathbf{0}\end{bmatrix}\mathbf{u}$$

This works because $\mathbf{K}$ has the same number of columns as states.

Various states can also be produced in the output with $\mathbf{I}$ matrix augmentation.

[5] Observers use a matrix $\mathbf{K}$ to steer the state estimate toward the true state in the same way controllers use a matrix $\mathbf{K}$ to steer the current state toward a desired state. We'll cover this in more detail in part III.

6.6.5 Examples

Snippet 6.1 shows how one packs together the following augmented matrix in Python using `np.block()`.

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5], [6]])
C = np.array([[7, 8]])
D = np.array([[9]])

tmp = np.block([[A, B], [C, D]])
```

Snippet 6.1. Matrix augmentation example: block

Snippet 6.2 shows how one packs together the same augmented matrix in Python using array slices.

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5], [6]])
C = np.array([[7, 8]])
D = np.array([[9]])

tmp = np.empty((3, 3))
tmp[:2, :2] = A  # tmp[0:2, 0:2] = A
tmp[:2, 2:] = B  # tmp[0:2, 2:3] = B
tmp[2:, :2] = C  # tmp[2:3, 0:2] = C
tmp[2:, 2:] = D  # tmp[2:3, 2:3] = D
```

Snippet 6.2. Matrix augmentation example: array slices

Section 6.7 demonstrates model augmentation for different types of integral control.

6.7 Integral control

A common way of implementing integral control is to add an additional state that is the integral of the error of the variable intended to have zero steady-state error.

We'll present two methods:

1. Augment the plant. For an arm, one would add an "integral of position" state.
2. Estimate the "error" in the control input (the difference between what was applied versus what was observed to happen) via the observer and compensate for it. We'll call this "input error estimation".

In FRC, avoid integral control unless you have a very good reason to use it. Integral control adds significant complexity, and steady-state error can often be avoided with a motion profile, a well-tuned feedforward, and proportional feedback (i.e., more deterministic options you should be using anyway).

6.7.1 Plant augmentation

Caveats

First, unconstrained integral control exhibits integral windup on a unit step input. When the error reaches zero, the integrator may still have a large positive accumulated error from the initial ramp-up. The integrator makes the system overshoot until the accumulated error is unwound by negative errors. Poor tuning can lead to instability.[6] Integrating only when close to the reference somewhat mitigates integral windup.

Second, unconstrained integral control is a poor choice for modeled dynamics compensation. Feedforwards provide more precise compensation since we already know beforehand how to counteract the undesirable dynamics.

Third, unconstrained integral control is a poor choice for unmodeled dynamics compensation. To choose proper gains, the integrator must be tuned online when the unmodeled dynamics are present, which may be inconvenient or unsafe in some circumstances. Furthermore, accumulation even when the system is following the model means it still compensates for modeled dynamics despite our intent otherwise. Prefer the approach in subsection 6.7.2.

Implementation

We want to augment the system with an integral term that integrates the error $\mathbf{e} = \mathbf{r} - \mathbf{y} = \mathbf{r} - \mathbf{C}\mathbf{x}$.

$$\mathbf{x}_I = \int \mathbf{e}\, dt$$
$$\dot{\mathbf{x}}_I = \mathbf{e} = \mathbf{r} - \mathbf{C}\mathbf{x}$$

The plant is augmented as

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_I \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_I \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{u} + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{r}$$
$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_I \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_I \end{bmatrix} + \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{r} \end{bmatrix}$$

[6] With proper tuning, proportional control should generally be doing much more work than integral control.

The controller is augmented as

$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x}) - \mathbf{K}_I\mathbf{x}_I$$
$$\mathbf{u} = \begin{bmatrix}\mathbf{K} & \mathbf{K}_I\end{bmatrix}\left(\begin{bmatrix}\mathbf{r}\\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\mathbf{x}\\ \mathbf{x}_I\end{bmatrix}\right)$$

6.7.2 Input error estimation

Models can predict system behavior, but unmodeled disturbances make the observed system behavior deviate from the model. We want to react to these disturbances quickly and improve the model's predictive power in the face of these disturbances.

Input error estimation uses a state observer (covered in chapter 9) to estimate the difference between the provided model input and a hypothetical input that makes the model match the observed behavior. Subtracting this value from the provided input compensates for unmodeled disturbances, and adding it to the state observer's input makes the model better predict the system's future behavior.

First, we'll consider the one-dimensional case. Let u_{error} be the difference between the hypothetical input with disturbances and the provided input. The u_{error} term is then added to the system as follows.

$$\dot{x} = Ax + B\left(u + u_{error}\right)$$

$u + u_{error}$ is the hypothetical input actually applied to the system.

$$\dot{x} = Ax + Bu + Bu_{error}$$

The following equation generalizes this to a multiple-input system.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{B}_{error}u_{error}$$

where $\mathbf{B}_{error}$ is a column vector that maps u_{error} to changes in the rest of the state the same way $\mathbf{B}$ does for $\mathbf{u}$. $\mathbf{B}_{error}$ is only a column of $\mathbf{B}$ if u_{error} corresponds to an existing input within $\mathbf{u}$.

Given the above equation, we'll augment the plant as

$$\begin{bmatrix}\dot{\mathbf{x}}\\ u_{error}\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{B}_{error}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ u_{error}\end{bmatrix} + \begin{bmatrix}\mathbf{B}\\ \mathbf{0}\end{bmatrix}\mathbf{u}$$
$$\mathbf{y} = \begin{bmatrix}\mathbf{C} & 0\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ u_{error}\end{bmatrix} + \mathbf{D}\mathbf{u}$$

Notice how the state is augmented with u_{error}. With this model, the observer will estimate both the state and the u_{error} term. The controller is augmented similarly. $\mathbf{r}$ is augmented with a zero for the goal u_{error} term.

$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x}) - \mathbf{k}_{error} u_{error}$$
$$\mathbf{u} = \begin{bmatrix}\mathbf{K} & \mathbf{k}_{error}\end{bmatrix} \left(\begin{bmatrix}\mathbf{r} \\ 0\end{bmatrix} - \begin{bmatrix}\mathbf{x} \\ u_{error}\end{bmatrix} \right)$$

where $\mathbf{k}_{error}$ is a column vector with a 1 in a given row if u_{error} should be applied to that input or a 0 otherwise.

This process can be repeated for an arbitrary error which can be corrected via some linear combination of the inputs.

6.8 Double integrator

The double integrator has two states (position and velocity) and one input (acceleration). Their relationship can be expressed by the following system of differential equations, where x is position, v is velocity, and a is acceleration.

$$\dot{x} = v$$
$$\dot{v} = a$$

We want to put these into the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$. Let $\mathbf{x} = \begin{bmatrix}x & v\end{bmatrix}^\mathsf{T}$ and $\mathbf{u} = \begin{bmatrix}a\end{bmatrix}^\mathsf{T}$. First, add missing terms so that all equations have the same states and inputs. Then, sort them by states followed by inputs.

$$\dot{x} = 0x + 1v + 0a$$
$$\dot{v} = 0x + 0v + 1a$$

Now, factor out the constants into matrices.

$$\begin{bmatrix}\dot{x} \\ v\end{bmatrix} = \begin{bmatrix}0 & 1 \\ 0 & 0\end{bmatrix} \begin{bmatrix}x \\ v\end{bmatrix} + \begin{bmatrix}0 \\ 1\end{bmatrix} \begin{bmatrix}a\end{bmatrix}$$
$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

6.9 Elevator

This elevator consists of a DC motor attached to a pulley that drives a mass up or down.

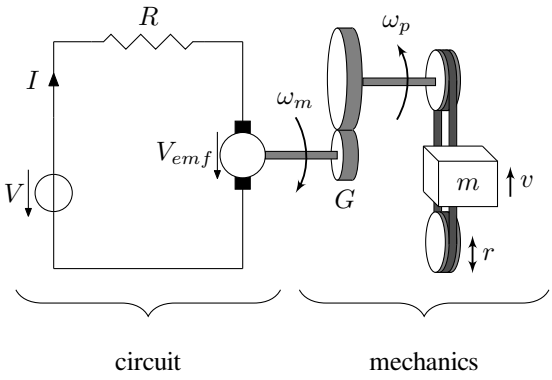


Figure 6.2: Elevator system diagram

6.9.1 Continuous state-space model

Using equation (12.15), the position and velocity derivatives of the elevator can be written as

$$\dot{x} = v \tag{6.5}$$

$$\dot{v} = -\frac{G^2 K_t}{Rr^2 m K_v} v + \frac{G K_t}{Rrm} V \tag{6.6}$$

Factor out v and V into column vectors.

$$\begin{bmatrix}\dot{v}\end{bmatrix} = \begin{bmatrix}-\frac{G^2 K_t}{Rr^2 m K_v}\end{bmatrix} \begin{bmatrix}v\end{bmatrix} + \begin{bmatrix}\frac{G K_t}{Rrm}\end{bmatrix} \begin{bmatrix}V\end{bmatrix}$$

Augment the matrix equation with the position state x, which has the model equation $\dot{x} = v$. The matrix elements corresponding to v will be 1, and the others will be 0 since they don't appear, so $\dot{x} = 0x + 1v + 0V$. The existing rows will have zeroes inserted where x is multiplied in.

$$\begin{bmatrix}\dot{x} \\ \dot{v}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ 0 & -\frac{G^2 K_t}{Rr^2 m K_v}\end{bmatrix} \begin{bmatrix}x \\ v\end{bmatrix} + \begin{bmatrix}0 \\ \frac{G K_t}{Rrm}\end{bmatrix} \begin{bmatrix}V\end{bmatrix}$$

Theorem 6.9.1 — Elevator state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix} \quad \mathbf{y} = x = \text{position} \quad \mathbf{u} = V = \text{voltage}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{G^2 K_t}{Rr^2 m K_v} \end{bmatrix} \tag{6.7}$$
$$\mathbf{B} = \begin{bmatrix} 0 \\ \frac{G K_t}{Rrm} \end{bmatrix} \tag{6.8}$$
$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{6.9}$$
$$\mathbf{D} = 0 \tag{6.10}$$

6.9.2 Model augmentation

As per subsection 6.7.2, we will now augment the model so a u_{error} state is added to the control input.

The plant and observer augmentations should be performed before the model is discretized. After the controller gain is computed with the unaugmented discrete model, the controller may be augmented. Therefore, the plant and observer augmentations assume a continuous model and the controller augmentation assumes a discrete controller.

$$\mathbf{x}_{aug} = \begin{bmatrix} x \\ v \\ u_{error} \end{bmatrix} \quad \mathbf{y} = x \quad \mathbf{u} = V$$

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0}_{1\times 2} & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{6.11}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{6.12}$$

This will compensate for unmodeled dynamics such as gravity. However, using a constant voltage feedforward to counteract gravity is preferred over u_{error} estimation in this case because it results in a simpler controller with similar performance.

6.9.3 Gravity feedforward

Input voltage is proportional to force and gravity is a constant force, so a constant voltage feedforward can compensate for gravity. We'll model gravity as an acceleration disturbance $-g$. To compensate for it, we want to find a voltage that is equal and opposite to it. The bottom row of the continuous elevator model contains the acceleration terms.

$$Bu_{ff} = -(\text{unmodeled dynamics})$$

where B is the motor acceleration term from $\mathbf{B}$ and u_{ff} is the voltage feedforward.

$$\begin{aligned} Bu_{ff} &= -(-g) \\ Bu_{ff} &= g \\ \frac{GK_t}{Rrm}u_{ff} &= g \\ u_{ff} &= \frac{Rrmg}{GK_t} \end{aligned}$$

6.9.4 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[7] creates and tests a controller for it. Figure 6.3 shows the closed-loop system response.

6.9.5 Implementation

C++ and Java implementations of this elevator controller are available online.[8][9]

[7] https://github.com/calcmogul/frccontrol/blob/main/examples/elevator.py

[8] https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/StateSpaceElevator/cpp/Robot.cpp

[9] https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/statespaceelevator/Robot.java

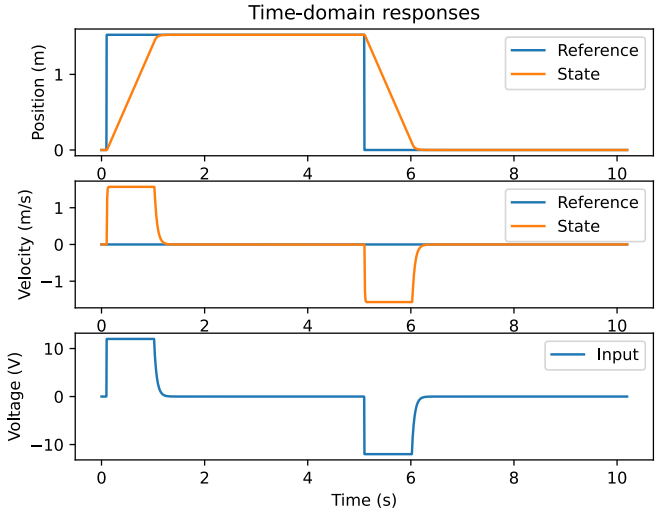


Figure 6.3: Elevator response

6.10 Flywheel

This flywheel consists of a DC motor attached to a spinning mass of non-negligible moment of inertia.

6.10.1 Continuous state-space model

By equation (12.18)

$$\dot{\omega} = -\frac{G^2 K_t}{K_v R J}\omega + \frac{G K_t}{R J} V$$

Factor out ω and V into column vectors.

$$\begin{bmatrix}\dot{\omega}\end{bmatrix} = \begin{bmatrix}-\frac{G^2 K_t}{K_v R J}\end{bmatrix}\begin{bmatrix}\omega\end{bmatrix} + \begin{bmatrix}\frac{G K_t}{R J}\end{bmatrix}\begin{bmatrix}V\end{bmatrix}$$

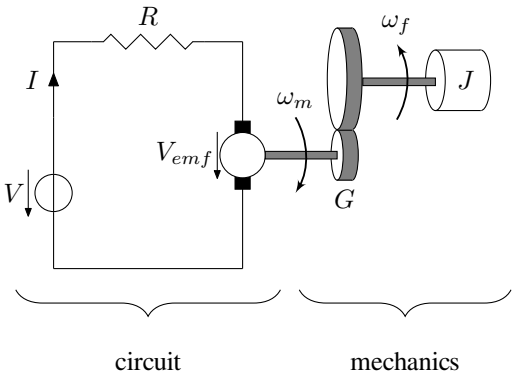


Figure 6.4: Flywheel system diagram

Theorem 6.10.1 — Flywheel state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$\mathbf{x} = \omega$ = angular velocity $\quad \mathbf{y} = \omega$ = angular velocity $\quad \mathbf{u} = V$ = voltage

$$\mathbf{A} = -\frac{G^2 K_t}{K_v R J} \tag{6.13}$$
$$\mathbf{B} = \frac{G K_t}{R J} \tag{6.14}$$
$$\mathbf{C} = 1 \tag{6.15}$$
$$\mathbf{D} = 0 \tag{6.16}$$

6.10.2 Model augmentation

As per subsection 6.7.2, we will now augment the model so a u_{error} state is added to the control input.

The plant and observer augmentations should be performed before the model is discretized. After the controller gain is computed with the unaugmented discrete model, the controller may be augmented. Therefore, the plant and observer augmentations assume a continuous model and the controller augmentation assumes a discrete con-

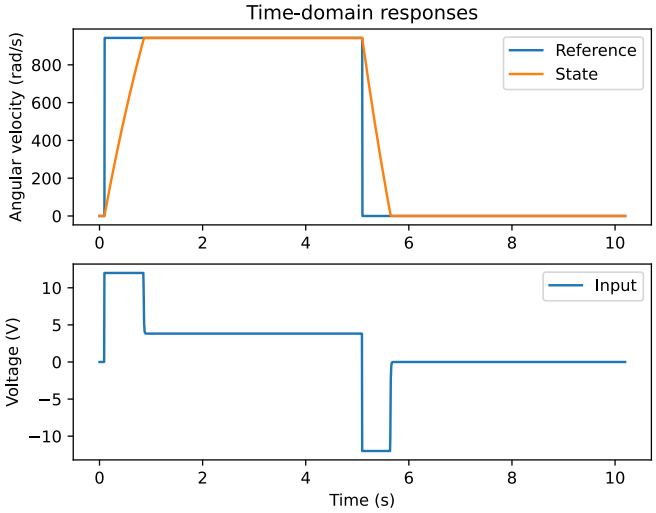


Figure 6.5: Flywheel response

troller.

$$\mathbf{x} = \begin{bmatrix} \omega \\ u_{error} \end{bmatrix} \quad \mathbf{y} = \omega \quad \mathbf{u} = V$$

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ 0 & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{6.17}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{6.18}$$

This will compensate for unmodeled dynamics such as projectiles slowing down the flywheel.

6.10.3 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[10] creates and tests a controller for it. Figure 6.5 shows the closed-loop system response.

Notice how the control effort when the reference is reached is nonzero. This is a plant inversion feedforward compensating for the system dynamics attempting to slow the flywheel down when no voltage is applied.

[10] https://github.com/calcmogul/frccontrol/blob/main/examples/flywheel.py

6.10.4 Implementation

C++ and Java implementations of this flywheel controller are available online.[11][12]

6.10.5 Flywheel model without encoder

In the FIRST Robotics Competition, we can get the current drawn for specific channels on the power distribution panel. We can theoretically use this to estimate the angular velocity of a DC motor without an encoder. We'll start with the flywheel model derived earlier as equation (12.18).

$$\dot{\omega} = \frac{GK_t}{RJ}V - \frac{G^2K_t}{K_vRJ}\omega$$
$$\dot{\omega} = -\frac{G^2K_t}{K_vRJ}\omega + \frac{GK_t}{RJ}V$$

Next, we'll derive the current I as an output.

$$V = IR + \frac{\omega}{K_v}$$
$$IR = V - \frac{\omega}{K_v}$$
$$I = -\frac{1}{K_vR}\omega + \frac{1}{R}V$$

Therefore,

[11]https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/StateSpaceFlywheel/cpp/Robot.cpp

[12]https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/statespaceflywheel/Robot.java

Theorem 6.10.2 — Flywheel state-space model without encoder.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\mathbf{x} = \omega = \text{angular velocity} \quad \mathbf{y} = I = \text{current} \quad \mathbf{u} = V = \text{voltage}$$

$$\mathbf{A} = -\frac{G^2 K_t}{K_v R J} \tag{6.19}$$
$$\mathbf{B} = \frac{G K_t}{R J} \tag{6.20}$$
$$\mathbf{C} = -\frac{1}{K_v R} \tag{6.21}$$
$$\mathbf{D} = \frac{1}{R} \tag{6.22}$$

Notice that in this model, the output doesn't provide any direct measurements of the state. To estimate the full state (also known as full observability), we only need the outputs to collectively include linear combinations of every state[13]. We'll revisit this in chapter 9 with an example that uses range measurements to estimate an object's orientation.

The effectiveness of this model's observer is heavily dependent on the quality of the current sensor used. If the sensor's noise isn't zero-mean, the observer won't converge to the true state.

6.10.6 Voltage compensation

To improve controller tracking, one may want to use the voltage renormalized to the power rail voltage to compensate for voltage drop when current spikes occur. This can be done as follows.

$$V = V_{cmd} \frac{V_{nominal}}{V_{rail}} \tag{6.23}$$

where V is the controller's new input voltage, V_{cmd} is the old input voltage, $V_{nominal}$ is the rail voltage when effects like voltage drop due to current draw are ignored, and V_{rail} is the real rail voltage.

[13]While the flywheel model's outputs are a linear combination of both the states and inputs, inputs don't provide new information about the states. Therefore, they don't affect whether the system is observable.

To drive the model with a more accurate voltage that includes voltage drop, the reciprocal can be used.

$$V = V_{cmd}\frac{V_{rail}}{V_{nominal}} \tag{6.24}$$

where V is the model's new input voltage. Note that if both the controller compensation and model compensation equations are applied, the original voltage is obtained. The model input only drops from ideal if the compensated controller voltage saturates.

6.10.7 Do flywheels need PD control?

PID controllers typically control voltage to a motor in FRC independent of the equations of motion of that motor. For position PID control, large values of K_p can lead to overshoot and K_d is commonly used to reduce overshoots. Let's consider a flywheel controlled with a standard PID controller. Why wouldn't K_d provide damping for velocity overshoots in this case?

PID control is designed to control second-order and first-order systems well. It can be used to control a lot of things, but struggles when given higher order systems. It has three degrees of freedom. Two are used to place the two poles of the system, and the third is used to remove steady-state error. With higher order systems like a one input, seven state system, there aren't enough degrees of freedom to place the system's poles in desired locations. This will result in poor control.

The math for PID doesn't assume voltage, a motor, etc. It defines an output based on derivatives and integrals of its input. We happen to use it for motors because it actually works pretty well for it because motors are second-order systems.

The following math will be in continuous time, but the same ideas apply to discrete time. This is all assuming a velocity controller.

Our simple motor model hooked up to a mass is

$$V = IR + \frac{\omega}{K_v} \tag{6.25}$$

$$\tau = IK_t \tag{6.26}$$

$$\tau = J\frac{d\omega}{dt} \tag{6.27}$$

For an explanation of where these equations come from, read section 12.1.

First, we'll solve for $\frac{d\omega}{dt}$ in terms of V.

Substitute equation (6.26) into equation (6.25).

$$V = IR + \frac{\omega}{K_v}$$
$$V = \left(\frac{\tau}{K_t}\right) R + \frac{\omega}{K_v}$$

Substitute in equation (6.27).

$$V = \frac{\left(J\frac{d\omega}{dt}\right)}{K_t} R + \frac{\omega}{K_v}$$

Solve for $\frac{d\omega}{dt}$.

$$\begin{aligned} V &= \frac{J\frac{d\omega}{dt}}{K_t} R + \frac{\omega}{K_v} \\ V - \frac{\omega}{K_v} &= \frac{J\frac{d\omega}{dt}}{K_t} R \\ \frac{d\omega}{dt} &= \frac{K_t}{JR}\left(V - \frac{\omega}{K_v}\right) \\ \underbrace{\frac{d\omega}{dt}}_{\dot{\mathbf{x}}} &= \underbrace{-\frac{K_t}{JRK_v}}_{\mathbf{A}} \underbrace{\omega}_{\mathbf{x}} + \underbrace{\frac{K_t}{JR}}_{\mathbf{B}} \underbrace{V}_{\mathbf{u}} \end{aligned} \tag{6.28}$$

There's one stable open-loop pole at $-\frac{K_t}{JRK_v}$. Let's try a simple P controller.

$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$$
$$V = K_p(\omega_{goal} - \omega)$$

Closed-loop models have the form $\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{BKr}$. Therefore, the closed-loop poles are the eigenvalues of $\mathbf{A} - \mathbf{BK}$.

$$\begin{aligned} \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{BKr} \\ \dot{\omega} &= \left(\left(-\frac{K_t}{JRK_v}\right) - \left(\frac{K_t}{JR}\right)(K_p)\right)\omega + \left(\frac{K_t}{JR}\right)(K_p)(\omega_{goal}) \\ \dot{\omega} &= -\left(\frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)\omega + \frac{K_t K_p}{JR}\omega_{goal} \end{aligned}$$

This closed-loop flywheel model has one pole at $-\left(\frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)$. It therefore only needs one P controller to place that pole anywhere on the real axis. A derivative term

is unnecessary on an ideal flywheel. It may compensate for unmodeled dynamics such as accelerating projectiles slowing the flywheel down, but that effect may also increase recovery time; K_d drives the acceleration to zero in the undesired case of negative acceleration as well as well as the actually desired case of positive acceleration.

This analysis assumes that the motor is well coupled to the mass and that the time constant of the inductor is small enough that it doesn't factor into the motor equations. The latter is a pretty good assumption for a CIM motor with the following constants: $J = 3.2284 \times 10^{-6}\ kg\text{-}m^2$, $b = 3.5077 \times 10^{-6}\ N\text{-}m\text{-}s$, $K_e = K_t = 0.0181\,V/rad/s$, $R = 0.0902\,\Omega$, and $L = 230$ µH. Notice the slight wiggle in figure 6.6 compared to figure 6.7. If more mass is added to the motor armature, the response timescales increase and the inductance matters even less.

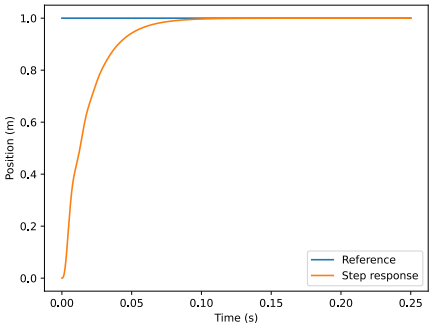


Figure 6.6: Step response of second-order DC motor plant augmented with position ($L = 230$ µH)

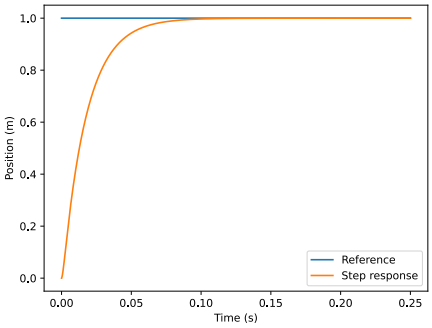


Figure 6.7: Step response of first-order DC motor plant augmented with position ($L = 0$ µH)

Subsection 6.7.2 covers a superior compensation method that avoids zeroes in the controller, doesn't act against the desired control action, and facilitates better tracking.

6.11 Single-jointed arm

This single-jointed arm consists of a DC motor attached to a pulley that spins a straight bar in pitch.

6.11.1 Continuous state-space model

Using equation (12.23), the angle and angular rate derivatives of the arm can be written as

$$\dot{\theta}_{arm} = \omega_{arm} \tag{6.29}$$

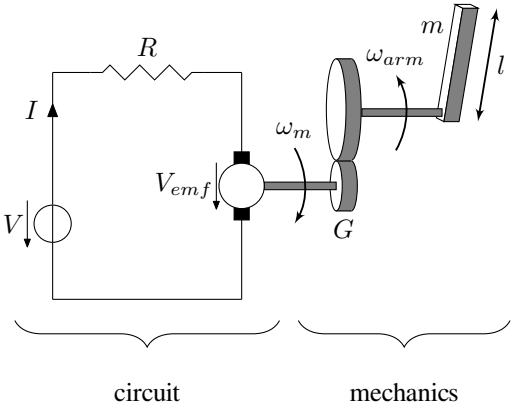


Figure 6.8: Single-jointed arm system diagram

$$\dot{\omega}_{arm} = -\frac{G^2 K_t}{K_v R J}\omega_{arm} + \frac{G K_t}{R J} V \tag{6.30}$$

Factor out ω_{arm} and V into column vectors.

$$\left[\dot{\omega_{arm}}\right] = \left[-\tfrac{G^2 K_t}{K_v R J}\right]\left[\omega_{arm}\right] + \left[\tfrac{G K_t}{R J}\right]\left[V\right]$$

Augment the matrix equation with the angle state θ_{arm}, which has the model equation $\dot{\theta}_{arm} = \omega_{arm}$. The matrix elements corresponding to ω_{arm} will be 1, and the others will be 0 since they don't appear, so $\dot{\theta}_{arm} = 0\theta_{arm} + 1\omega_{arm} + 0V$. The existing rows will have zeroes inserted where θ_{arm} is multiplied in.

$$\begin{bmatrix}\dot{\theta_{arm}} \\ \dot{\omega_{arm}}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ 0 & -\frac{G^2 K_t}{K_v R J}\end{bmatrix}\begin{bmatrix}\theta_{arm} \\ \omega_{arm}\end{bmatrix} + \begin{bmatrix}0 \\ \frac{G K_t}{R J}\end{bmatrix}\left[V\right]$$

Theorem 6.11.1 — Single-jointed arm state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} \theta_{arm} \\ \omega_{arm} \end{bmatrix} = \begin{bmatrix} \text{angle} \\ \text{angular velocity} \end{bmatrix} \quad \mathbf{y} = \theta_{arm} = \text{angle} \quad \mathbf{u} = V = \text{voltage}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{G^2 K_t}{K_v R J} \end{bmatrix} \tag{6.31}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ \frac{G K_t}{R J} \end{bmatrix} \tag{6.32}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{6.33}$$

$$\mathbf{D} = 0 \tag{6.34}$$

6.11.2 Model augmentation

As per subsection 6.7.2, we will now augment the model so a u_{error} state is added to the control input.

The plant and observer augmentations should be performed before the model is discretized. After the controller gain is computed with the unaugmented discrete model, the controller may be augmented. Therefore, the plant and observer augmentations assume a continuous model and the controller augmentation assumes a discrete controller.

$$\mathbf{x}_{aug} = \begin{bmatrix} \mathbf{x} \\ u_{error} \end{bmatrix} \quad \mathbf{y} = \theta_{arm} \quad \mathbf{u} = V$$

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0}_{1\times 2} & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{6.35}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{6.36}$$

This will compensate for unmodeled dynamics such as gravity or other external loading from lifted objects. However, if only gravity compensation is desired, a feedforward of the form $u_{ff} = V_{gravity} \cos\theta$ is preferred where $V_{gravity}$ is the voltage required to hold the arm level with the ground and θ is the angle of the arm with the ground.

6.11.3 Gravity feedforward

Input voltage is proportional to torque and gravity is a constant force, but the torque applied against the motor varies according to the arm's angle. We'll use sum of torques to find a compensating torque.

We'll model gravity as an acceleration disturbance $-g$. To compensate for it, we want to find a torque that is equal and opposite to the torque applied to the arm by gravity. The bottom row of the continuous elevator model contains the angular acceleration terms, so Bu_{ff} is angular acceleration caused by the motor; JBu_{ff} is the torque.

$$\begin{aligned} JBu_{ff} &= -(\mathbf{r} \times \mathbf{F}) \\ JBu_{ff} &= -(rF\cos\theta) \end{aligned}$$

Torque is usually written as $rF\sin\theta$ where θ is the angle between the $\mathbf{r}$ and $\mathbf{F}$ vectors, but θ in this case is being measured from the horizontal axis rather than the vertical one, so the force vector is $\frac{\pi}{4}$ radians out of phase. Thus, an angle of 0 results in the maximum torque from gravity being applied rather than the minimum.

The force of gravity mg is applied at the center of the arm's mass. For a uniform beam, this is halfway down its length, or $\frac{L}{2}$ where L is the length of the arm.

$$\begin{aligned} JBu_{ff} &= -\left(\left(\frac{L}{2}\right)(-mg)\cos\theta\right) \\ JBu_{ff} &= mg\frac{L}{2}\cos\theta \end{aligned}$$

$B = \frac{GK_t}{RJ}$, so

$$\begin{aligned} J\frac{GK_t}{RJ}u_{ff} &= mg\frac{L}{2}\cos\theta \\ u_{ff} &= \frac{RJ}{JGK_t}mg\frac{L}{2}\cos\theta \\ u_{ff} &= \frac{L}{2}\frac{Rmg}{GK_t}\cos\theta \end{aligned}$$

$\frac{L}{2}$ can be adjusted according to the location of the arm's center of mass.

6.11.4 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[14] creates and tests a controller for it. Figure 6.9 shows the closed-loop system response.

[14] https://github.com/calcmogul/frccontrol/blob/main/examples/single_jointed_arm.py

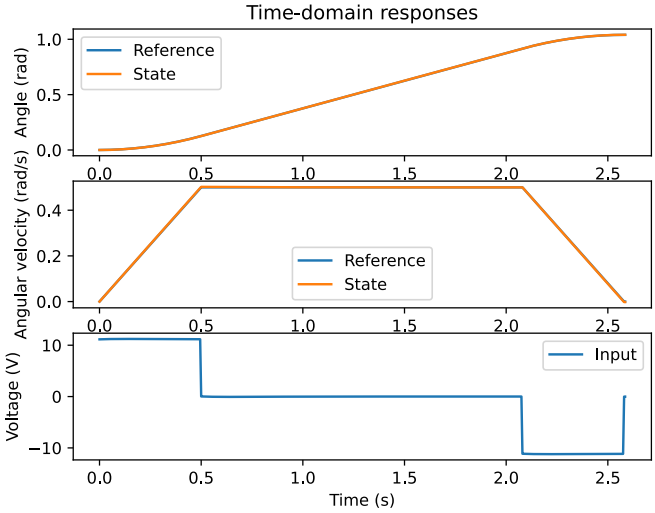


Figure 6.9: Single-jointed arm response

6.11.5 Implementation

C++ and Java implementations of this single-jointed arm controller are available online.[15][16]

[15]https://github.com/wpilibsuite/allwpilib/blob/main/wpilibcExamples/src/main/cpp/examples/StateSpaceArm/cpp/Robot.cpp

[16]https://github.com/wpilibsuite/allwpilib/blob/main/wpilibjExamples/src/main/java/edu/wpi/first/wpilibj/examples/statespacearm/Robot.java

6.12 Controllability and observability

6.12.1 Controllability matrix

A system is controllable if it can be steered from any state to any state by a finite sequence of admissible inputs.

The controllability matrix can be used to determine if a system is controllable.

Theorem 6.12.1 — Controllability. A continuous time-invariant linear state-space model is controllable if and only if

$$\operatorname{rank}\begin{pmatrix}\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}\end{pmatrix} = n \tag{6.37}$$

where rank is the number of linearly independent rows in a matrix and n is the number of states.

The controllability matrix in equation (6.37) being rank-deficient means the inputs cannot apply transforms along all axes in the state-space; the transformation the matrix represents is collapsed into a lower dimension.

The condition number of the controllability matrix $\mathcal{C}$ is defined as $\frac{\sigma_{max}(\mathcal{C})}{\sigma_{min}(\mathcal{C})}$ where σ_{max} is the maximum singular value[17] and σ_{min} is the minimum singular value. As this number approaches infinity, one or more of the states becomes uncontrollable. This number can also be used to tell us which actuators are better than others for the given system; a lower condition number means that the actuators have more control authority.

6.12.2 Controllability Gramian

While the rank of the observability matrix can tell us whether the system is controllable, it won't tell us which specific states are controllable or how controllable. The controllability Gramian can be used to determine these things.

If $\mathbf{A}$ is stable, the controllability Gramian $\mathbf{W}_c$ is the unique solution to the following continuous Lyapunov equation.

$$\mathbf{AW}_c + \mathbf{W}_c\mathbf{A}^\mathsf{T} + \mathbf{BB}^\mathsf{T} = 0 \tag{6.38}$$

Alternatively,

$$\mathbf{W}_c = \int_0^\infty e^{\mathbf{A}\tau}\mathbf{B}\mathbf{B}^\mathsf{T}e^{\mathbf{A}^\mathsf{T}\tau}\,d\tau \tag{6.39}$$

[17] Singular values are a generalization of eigenvalues for nonsquare matrices.

If the solution is positive definite, the system is controllable. The eigenvalues of $\mathbf{W}_c$ represent how controllable their respective states are (larger means more controllable).

6.12.3 Controllability of specific states

If you want to know if a specific state is controllable, first find its corresponding eigenvalue λ in $\mathbf{A}$. Then, that state is controllable if

$$\operatorname{rank}\begin{pmatrix}\begin{bmatrix}\lambda\mathbf{I} - \mathbf{A} & \mathbf{B}\end{bmatrix}\end{pmatrix} = n \tag{6.40}$$

where n is the number of states.

6.12.4 Stabilizability

Stabilizability is a weaker form of controllability. A system is considered stabilizable if one of the following conditions is true:

1. All uncontrollable states can be stabilized
2. All unstable states are controllable

6.12.5 Observability matrix

A system is observable if the state, whatever it may be, can be inferred from a finite sequence of outputs.

Observability and controllability are mathematical duals; controllability proves that a sequence of inputs exists that drives the system to any state, and observability proves that a sequence of outputs exists that drives the state estimate to any true state.

The observability matrix can be used to determine if a system is observable.

Theorem 6.12.2 — Observability. A continuous time-invariant linear state-space model is observable if and only if

$$\operatorname{rank}\left(\begin{bmatrix}\mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{n-1}\end{bmatrix}\right) = n \tag{6.41}$$

where rank is the number of linearly independent rows in a matrix and n is the number of states.

The observability matrix in equation (6.41) being rank-deficient means the outputs do not contain contributions from every state. That is, not all states are mapped to a linear

combination in the output. Therefore, the outputs alone are insufficient to estimate all the states.

The condition number of the observability matrix $\mathcal{O}$ is defined as $\frac{\sigma_{max}(\mathcal{O})}{\sigma_{min}(\mathcal{O})}$ where σ_{max} is the maximum singular value[17] and σ_{min} is the minimum singular value. As this number approaches infinity, one or more of the states becomes unobservable. This number can also be used to tell us which sensors are better than others for the given system; a lower condition number means the outputs produced by the sensors are better indicators of the system state.

6.12.6 Observability Gramian

While the rank of the observability matrix can tell us whether the system is observable, it won't tell us which specific states are observable or how observable. The observability Gramian can be used to determine these things.

If $\mathbf{A}$ is stable, the observability Gramian $\mathbf{W}_o$ is the unique solution to the following continuous Lyapunov equation.

$$\mathbf{A}^\mathsf{T}\mathbf{W}_o + \mathbf{W}_o\mathbf{A} + \mathbf{C}^\mathsf{T}\mathbf{C} = 0 \tag{6.42}$$

Alternatively,

$$\mathbf{W}_o = \int_0^\infty e^{\mathbf{A}^\mathsf{T}\tau}\mathbf{C}^\mathsf{T}\mathbf{C}e^{\mathbf{A}\tau}\,d\tau \tag{6.43}$$

If the solution is positive definite, the system is observable. The eigenvalues of $\mathbf{W}_o$ represent how observable their respective states are (larger means more observable).

6.12.7 Observability of specific states

If you want to know if a specific state is observable, first find its corresponding eigenvalue λ in $\mathbf{A}$. Then, that state is observable if

$$\text{rank}\left(\begin{bmatrix}\lambda\mathbf{I} - \mathbf{A} \\ \mathbf{C}\end{bmatrix}\right) = n \tag{6.44}$$

where n is the number of states.

6.12.8 Detectability

Detectability is a weaker form of observability. A system is considered detectable if one of the following conditions is true:

1. All unobservable states are stable
2. All unstable states are observable

7. Discrete state-space control

The complex plane discussed so far deals with continuous systems. In decades past, plants and controllers were implemented using analog electronics, which are continuous in nature. Nowadays, microprocessors can be used to achieve cheaper, less complex controller designs. Discretization converts the continuous model we've worked with so far from a differential equation like

$$\dot{x} = x - 3 \tag{7.1}$$

to a difference equation like

$$\begin{aligned} \frac{x_{k+1} - x_k}{\Delta T} &= x_k - 3 \\ x_{k+1} - x_k &= (x_k - 3)\Delta T \\ x_{k+1} &= x_k + (x_k - 3)\Delta T \end{aligned} \tag{7.2}$$

where x_k refers to the value of x at the k^{th} timestep. The difference equation is run with some update period denoted by T, by ΔT, or sometimes sloppily by dt.[1]

While higher order terms of a differential equation are derivatives of the state variable (e.g., $\ddot{x}$ in relation to equation (7.1)), higher order terms of a difference equation are delayed copies of the state variable (e.g., x_{k-1} with respect to x_k in equation (7.2)).

[1]The discretization of equation (7.1) to equation (7.2) uses the forward Euler discretization method.

7.1 Continuous to discrete pole mapping

When a continuous system is discretized, its poles in the LHP map to the inside of a unit circle. Table 7.1 contains a few common points and figure 7.1 shows the mapping visually.

Continuous	Discrete
$(0, 0)$	$(1, 0)$
imaginary axis	edge of unit circle
$(-\infty, 0)$	$(0, 0)$

Table 7.1: Mapping from continuous to discrete

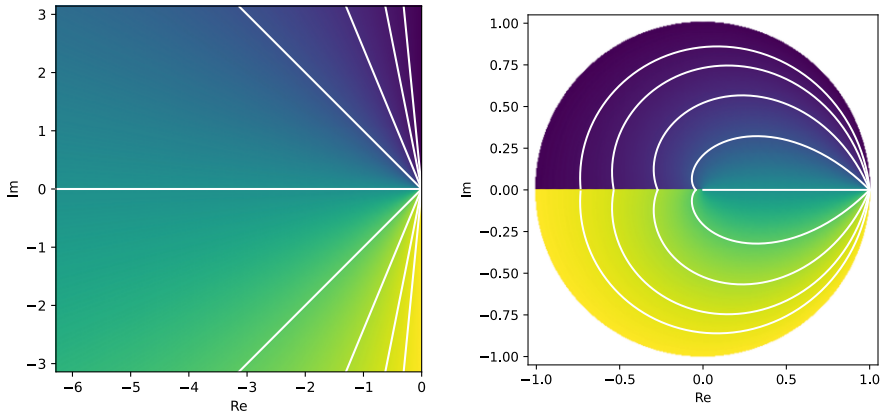


Figure 7.1: Mapping of complex plane from continuous (left) to discrete (right)

7.1.1 Discrete system stability

Eigenvalues of a system that are within the unit circle are stable. To demonstrate this, consider the discrete system $x_{k+1} = ax_k$ where a is a complex number. $|a| < 1$ will make x_{k+1} converge to zero.

7.1.2 Discrete system behavior

Figure 7.2 shows the impulse responses in the time domain for systems with various pole locations in the complex plane (real numbers on the x-axis and imaginary numbers

on the y-axis). Each response has an initial condition of 1.

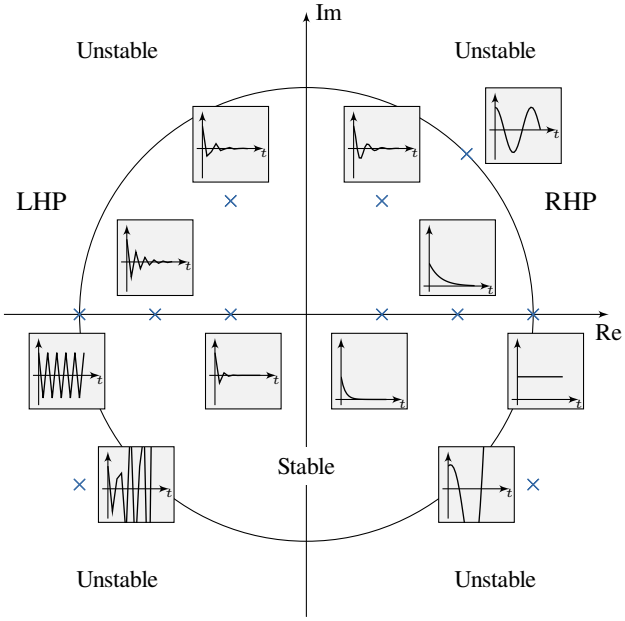


Figure 7.2: Discrete impulse response vs pole location

As ω increases in $s = j\omega$, a pole in the discrete plane moves around the perimeter of the unit circle. Once it hits $\frac{\omega_s}{2}$ (half the sampling frequency) at $(-1, 0)$, the pole wraps around. This is due to poles faster than the sample frequency folding down to below the sample frequency (that is, higher frequency signals *alias* to lower frequency ones).

Placing the poles at $(0, 0)$ produces a *deadbeat controller*. An N^{th}-order deadbeat controller decays to the reference in N timesteps. While this sounds great, there are other considerations like control effort, robustness, and noise immunity.

Poles in the left half-plane cause jagged outputs because the frequency of the system dynamics is above the Nyquist frequency (twice the sample frequency). The discretized signal doesn't have enough samples to reconstruct the continuous system's dynamics. See figures 7.3 and 7.4 for examples.

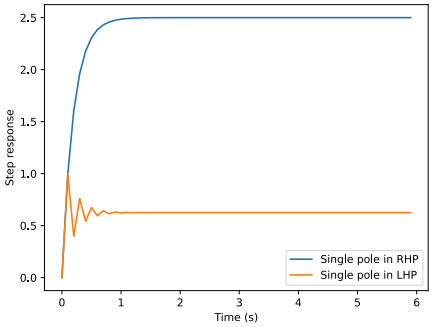


Figure 7.3: Single poles in various locations in discrete plane

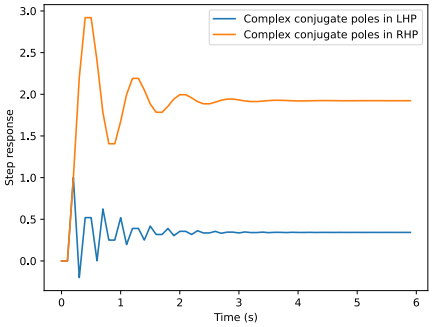


Figure 7.4: Complex conjugate poles in various locations in discrete plane

7.1.3 Nyquist frequency

To completely reconstruct a signal, the Nyquist-Shannon sampling theorem states that it must be sampled at a frequency at least twice the maximum frequency it contains. The highest frequency a given sample rate can capture is called the Nyquist frequency, which is half the sample frequency. This is why recorded audio is sampled at 44.1 kHz. The maximum frequency a typical human can hear is about 20 kHz, so the Nyquist frequency is 20 kHz and the minimum sampling frequency is 40 kHz. (44.1 kHz in particular was chosen for unrelated historical reasons.)

Frequencies above the Nyquist frequency are folded down across it. The higher frequency and the folded down lower frequency are said to alias each other.[2] Figure 7.5 demonstrates aliasing.

The effect of these high-frequency aliases can be reduced with a low-pass filter (called an anti-aliasing filter in this application).

[2]The aliases of a frequency f can be expressed as $f_{alias}(N) \stackrel{def}{=} |f - Nf_s|$. For example, if a 200 Hz sine wave is sampled at 150 Hz, the observer will see a 50 Hz signal instead of a 200 Hz one.

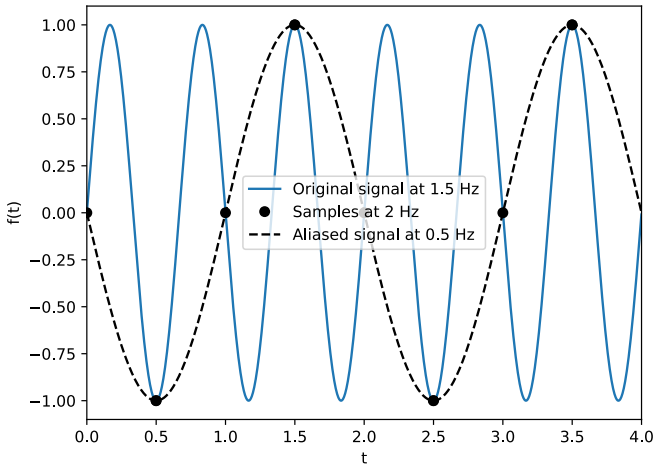


Figure 7.5: The original signal is a 1.5 Hz sine wave, which means its Nyquist frequency is 1.5 Hz. The signal is being sampled at 2 Hz, so the aliased signal is a 0.5 Hz sine wave.

7.2 Effects of discretization on controller performance

7.2.1 Sample delay

Implementing a discrete control system is easier than implementing a continuous one, but discretization has drawbacks. A microcontroller updates the system input in discrete intervals of duration T; it's held constant between updates. This introduces an average sample delay of $\frac{T}{2}$. Large delays can make a stable controller in the continuous domain become unstable in the discrete domain since it can't react as quickly to output changes. Here are a few ways to combat this.

- Run the controller with a high sample rate.
- Designing the controller in the analog domain with enough phase margin to compensate for any phase loss that occurs as part of discretization.
- Convert the plant to the digital domain and design the controller completely in the digital domain.

7.2.2 Sample rate

Running a feedback controller at a faster update rate doesn't always mean better control. In fact, you may be using more computational resources than you need. However, here are some reasons for running at a faster update rate.

Firstly, if you have a discrete model of the system, that model can more accurately approximate the underlying continuous system. Not all controllers use a model though.

Secondly, the controller can better handle fast system dynamics. If the system can move from its initial state to the desired one in under 250 ms, you obviously want to run the controller with a period less than 250 ms. When you reduce the sample period, you're making the discrete controller more accurately reflect what the equivalent continuous controller would do (controllers built from analog circuit components like op-amps are continuous).

Running at a lower sample rate only causes problems if you don't take into account the response time of your system. Some systems like heaters have outputs that change on the order of minutes. Running a control loop at 1 kHz doesn't make sense for this because the plant input the controller computes won't change much, if at all, in 1 ms.

7.3 Linear system discretization

We're going to discretize the following continuous time state-space model

$$\dot{\mathbf{x}} = \mathbf{A}_c\mathbf{x} + \mathbf{B}_c\mathbf{u} + \mathbf{w}$$
$$\mathbf{y} = \mathbf{C}_c\mathbf{x} + \mathbf{D}_c\mathbf{u} + \mathbf{v}$$

where $\mathbf{w}$ is the process noise, $\mathbf{v}$ is the measurement noise, and both are zero-mean white noise sources with covariances of $\mathbf{Q}_c$ and $\mathbf{R}_c$ respectively. $\mathbf{w}$ and $\mathbf{v}$ are defined as normally distributed random variables.

$$\mathbf{w} \sim N(0, \mathbf{Q}_c)$$
$$\mathbf{v} \sim N(0, \mathbf{R}_c)$$

Discretization is done using a zero-order hold. That is, the input is only updated at discrete intervals and it's held constant between samples.

7.3.1 Taylor series

R Watch the "Taylor series" video from 3Blue1Brown's *Essence of calculus* series for an explanation of how the Taylor series expansion works.

The definition for the matrix exponential and the approximations below all use the *Taylor series expansion*. The Taylor series is a method of approximating a function

"Taylor series" (22 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/taylor-series

like e^t via the summation of weighted polynomial terms like t^k. e^t has the following Taylor series around $t = 0$.

$$e^t = \sum_{n=0}^{\infty} \frac{t^n}{n!}$$

where a finite upper bound on the number of terms produces an approximation of e^t. As n increases, the polynomial terms increase in power and the weights by which they are multiplied decrease. For e^t and some other functions, the Taylor series expansion equals the original function for all values of t as the number of terms approaches infinity.[3] Figure 7.6 shows the Taylor series expansion of e^t around $t = 0$ for a varying number of terms.

We'll expand the first few terms of the Taylor series expansion in equation (7.3) for $\mathbf{X} = \mathbf{A}T$ so we can compare it with other methods.

$$\sum_{k=0}^{3} \frac{1}{k!}(\mathbf{A}T)^k = \mathbf{I} + \mathbf{A}T + \frac{1}{2}\mathbf{A}^2T^2 + \frac{1}{6}\mathbf{A}^3T^3$$

Table 7.2 compares discretization methods for the matrix case, and table 7.3 compares their Taylor series expansions. These use a more complex formula which we won't present here.

Each of them has different stability properties. The bilinear transform preserves the (in)stability of the continuous time system.

7.3.2 Matrix exponential

R Watch the "How (and why) to raise e to the power of a matrix" video from 3Blue1Brown's *Essence of linear algebra* series for a visual introduction to the matrix exponential.

[3] Functions for which their Taylor series expansion converges to and also equals it are called analytic functions.

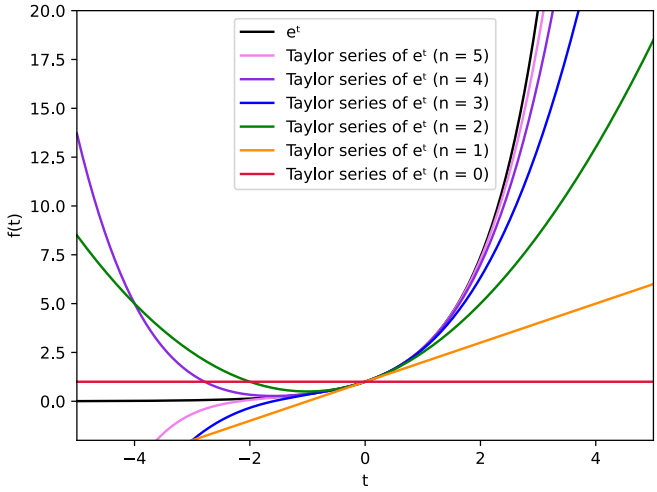


Figure 7.6: Taylor series expansions of e^t around $t = 0$ for n terms

"How (and why) to raise e to the power of a matrix" (27 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/matrix-exponents

Definition 7.3.1 — Matrix exponential. Let $\mathbf{X}$ be an $n \times n$ matrix. The exponential of $\mathbf{X}$ denoted by $e^{\mathbf{X}}$ is the $n \times n$ matrix given by the following power series.

$$e^{\mathbf{X}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{X}^k \tag{7.3}$$

where $\mathbf{X}^0$ is defined to be the identity matrix $\mathbf{I}$ with the same dimensions as $\mathbf{X}$.

To understand why the matrix exponential is used in the discretization process, consider the scalar differential equation $\dot{x} = ax$. The solution to this type of differential equation

Method	Conversion to $\mathbf{A}_d$
Zero-order hold	$e^{\mathbf{A}_c T}$
Bilinear	$\left(\mathbf{I} + \frac{1}{2}\mathbf{A}_c T\right)\left(\mathbf{I} - \frac{1}{2}\mathbf{A}_c T\right)^{-1}$
Backward Euler	$\left(\mathbf{I} - \mathbf{A}_c T\right)^{-1}$
Forward Euler	$\mathbf{I} + \mathbf{A}_c T$

Table 7.2: Discretization methods (matrix case). The zero-order hold discretization method is exact.

Method	Taylor series expansion
Zero-order hold	$\mathbf{I} + \mathbf{A}_c T + \frac{1}{2}\mathbf{A}_c^2 T^2 + \frac{1}{6}\mathbf{A}_c^3 T^3 + \ldots$
Bilinear	$\mathbf{I} + \mathbf{A}_c T + \frac{1}{2}\mathbf{A}_c^2 T^2 + \frac{1}{4}\mathbf{A}_c^3 T^3 + \ldots$
Backward Euler	$\mathbf{I} + \mathbf{A}_c T + \mathbf{A}_c^2 T^2 + \mathbf{A}_c^3 T^3 + \ldots$
Forward Euler	$\mathbf{I} + \mathbf{A}_c T$

Table 7.3: Taylor series expansions of discretization methods (matrix case). The zero-order hold discretization method is exact.

uses an exponential.

$$
\begin{aligned}
\dot{x} &= ax \\
\frac{dx}{dt} &= ax(t) \\
dx &= ax(t)\,dt \\
\frac{1}{x(t)}\,dx &= a\,dt \\
\int_0^t \frac{1}{x(t)}\,dx &= \int_0^t a\,dt \\
\ln(x(t))|_0^t &= at|_0^t \\
\ln(x(t)) - \ln(x(0)) &= at - a \cdot 0 \\
\ln(x(t)) - \ln(x_0) &= at \\
\ln\left(\frac{x(t)}{x_0}\right) &= at
\end{aligned}
$$

$$\frac{x(t)}{x_0} = e^{at}$$
$$x(t) = e^{at}x_0$$

This solution generalizes via the matrix exponential to the set of differential equations $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ we use to describe systems.[4]

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \mathbf{A}^{-1}(e^{\mathbf{A}t} - \mathbf{I})\mathbf{B}\mathbf{u}$$

where $\mathbf{x}_0$ contains the initial conditions and $\mathbf{u}$ is the constant input from time 0 to t. If the initial state is the current system state, then we can describe the system's state over time as

$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B}\mathbf{u}_k$$

or more compactly,

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k$$

where T is the time between samples $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$. Theorem 7.3.1 has more efficient ways to compute $\mathbf{A}_d$ and $\mathbf{B}_d$.

7.3.3 Definition

The model can be discretized as follows

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k + \mathbf{w}_k\\ \mathbf{y}_k &= \mathbf{C}_d\mathbf{x}_k + \mathbf{D}_d\mathbf{u}_k + \mathbf{v}_k\end{aligned}$$

with covariances

$$\begin{aligned}\mathbf{w}_k &\sim N(0, \mathbf{Q}_d)\\ \mathbf{v}_k &\sim N(0, \mathbf{R}_d)\end{aligned}$$

[4] See section D.1 for a complete derivation of the linear system zero-order hold.

Theorem 7.3.1 — Linear system zero-order hold.

$$\mathbf{A}_d = e^{\mathbf{A}_c T} \tag{7.4}$$

$$\mathbf{B}_d = \int_0^T e^{\mathbf{A}_c \tau} d\tau \mathbf{B}_c = \mathbf{A}_c^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}_c \tag{7.5}$$

$$\mathbf{C}_d = \mathbf{C}_c \tag{7.6}$$

$$\mathbf{D}_d = \mathbf{D}_c \tag{7.7}$$

$$\mathbf{Q}_d = \int_{\tau=0}^T e^{\mathbf{A}_c \tau} \mathbf{Q}_c e^{\mathbf{A}_c^\mathsf{T} \tau} d\tau \tag{7.8}$$

$$\mathbf{R}_d = \frac{1}{T}\mathbf{R}_c \tag{7.9}$$

where subscripts c and d denote matrices for the continuous or discrete systems respectively, T is the sample period of the discrete system, and $e^{\mathbf{A}_c T}$ is the matrix exponential of $\mathbf{A}_c T$.

$\mathbf{A}_d$ and $\mathbf{B}_d$ can be computed in one step as

$$e^{\begin{bmatrix} \mathbf{A}_c & \mathbf{B}_c \\ \mathbf{0} & \mathbf{0} \end{bmatrix} T} = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

and $\mathbf{Q}_d$ can be computed as

$$\Phi = e^{\begin{bmatrix} -\mathbf{A}_c & \mathbf{Q}_c \\ \mathbf{0} & \mathbf{A}_c^\mathsf{T} \end{bmatrix} T} = \begin{bmatrix} -\mathbf{A}_d & \mathbf{A}_d^{-1}\mathbf{Q}_d \\ \mathbf{0} & \mathbf{A}_d^\mathsf{T} \end{bmatrix}$$

where $\mathbf{Q}_d = \Phi_{2,2}^\mathsf{T} \Phi_{1,2}$ [10].

See appendix D.1 for derivations.

To see why $\mathbf{R}_c$ is being divided by T, consider the discrete white noise sequence $\mathbf{v}_k$ and the (non-physically realizable) continuous white noise process $\mathbf{v}$. Whereas $\mathbf{R}_{d,k} = E[\mathbf{v}_k \mathbf{v}_k^\mathsf{T}]$ is a covariance matrix, $\mathbf{R}_c(t)$ defined by $E[\mathbf{v}(t)\mathbf{v}^\mathsf{T}(\tau)] = \mathbf{R}_c(t)\delta(t-\tau)$ is a spectral density matrix (the Dirac function $\delta(t-\tau)$ has units of $1/\text{sec}$). The covariance matrix $\mathbf{R}_c(t)\delta(t-\tau)$ has infinite-valued elements. The discrete white noise sequence can be made to approximate the continuous white noise process by shrinking the pulse

lengths (T) and increasing their amplitude, such that $\mathbf{R}_d \to \frac{1}{T}\mathbf{R}_c$.

That is, in the limit as $T \to 0$, the discrete noise sequence tends to one of infinite-valued pulses of zero duration such that the area under the "impulse" autocorrelation function is $\mathbf{R}_d T$. This is equal to the area $\mathbf{R}_c$ under the continuous white noise impulse autocorrelation function.

7.4 Discrete state-space notation

Below is the discrete version of state-space notation.

Definition 7.4.1 — Discrete state-space notation.

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{7.10}$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \tag{7.11}$$

$\mathbf{A}$	system matrix	$\mathbf{x}$	state vector
$\mathbf{B}$	input matrix	$\mathbf{u}$	input vector
$\mathbf{C}$	output matrix	$\mathbf{y}$	output vector
$\mathbf{D}$	feedthrough matrix		

7.5 Closed-loop controller

With the control law $\mathbf{u}_k = \mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)$, we can derive the closed-loop state-space equations. We'll discuss where this control law comes from in subsection 7.7.

First is the state update equation. Substitute the control law into equation (7.10).

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k) \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k - \mathbf{B}\mathbf{K}\mathbf{x}_k \\
\mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k
\end{aligned} \tag{7.12}$$

Now for the output equation. Substitute the control law into equation (7.11).

$$\begin{aligned}
\mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}(\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)) \\
\mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{K}\mathbf{r}_k - \mathbf{D}\mathbf{K}\mathbf{x}_k \\
\mathbf{y}_k &= (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x}_k + \mathbf{D}\mathbf{K}\mathbf{r}_k
\end{aligned} \tag{7.13}$$

Instead of commanding the system to a state using the vector $\mathbf{u}_k$ directly, we can now specify a vector of desired states through $\mathbf{r}_k$ and the controller will choose values of $\mathbf{u}_k$ for us over time that drive the system toward the reference.

The eigenvalues of $\mathbf{A} - \mathbf{BK}$ are the poles of the closed-loop system. Therefore, the rate of convergence and stability of the closed-loop system can be changed by moving the poles via the eigenvalues of $\mathbf{A} - \mathbf{BK}$. $\mathbf{A}$ and $\mathbf{B}$ are inherent to the system, but $\mathbf{K}$ can be chosen arbitrarily by the controller designer. For equation (7.12) to reach steady-state, the eigenvalues of $\mathbf{A} - \mathbf{BK}$ must be in the left-half plane. There will be steady-state error if $\mathbf{A}\mathbf{r}_k \neq \mathbf{r}_k$.

Symbol	Name	Rows × Columns
$\mathbf{A}$	system matrix	states × states
$\mathbf{B}$	input matrix	states × inputs
$\mathbf{C}$	output matrix	outputs × states
$\mathbf{D}$	feedthrough matrix	outputs × inputs
$\mathbf{K}$	controller gain matrix	inputs × states
$\mathbf{r}$	reference vector	states × 1
$\mathbf{x}$	state vector	states × 1
$\mathbf{u}$	input vector	inputs × 1
$\mathbf{y}$	output vector	outputs × 1

Table 7.4: Controller matrix dimensions

7.6 Pole placement

This is the practice of placing the poles of a closed-loop system directly to produce a desired response. Python Control offers several pole placement algorithms for generating controller or observer gains from a set of poles.

Since all our applications will be discrete systems, we'll place poles in the discrete domain (the z-plane). The s-plane's LHP maps to the inside of a unit circle (see figure 7.7).

Pole placement should only be used if you know what you're doing. It's much easier to let LQR place the poles for you, which we'll discuss next.

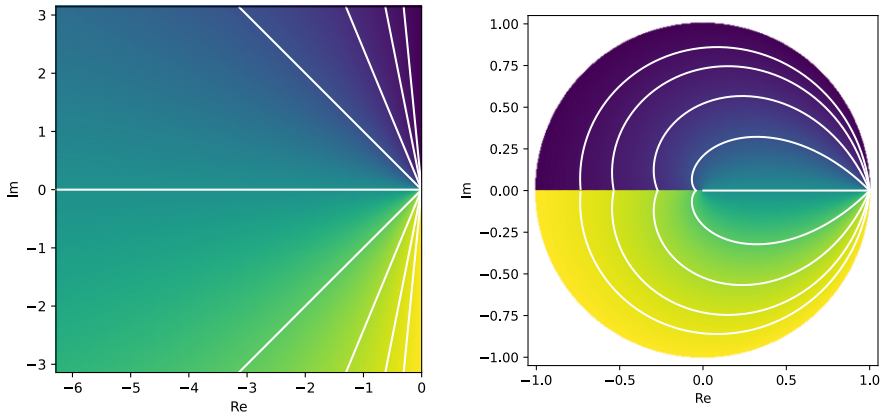


Figure 7.7: Mapping of complex plane from continuous (left) to discrete (right)

7.7 Linear-quadratic regulator

7.7.1 The intuition

We can demonstrate the basic idea behind the linear-quadratic regulator with the following flywheel model.

$$\dot{x} = ax + bu$$

where a is a negative constant representing the back-EMF of the motor, x is the angular velocity, b is a positive constant that maps the input voltage to some change in angular velocity (angular acceleration), u is the voltage applied to the motor, and $\dot{x}$ is the angular acceleration. Discretized, this equation would look like

$$x_{k+1} = a_d x + b_d u_k$$

If the angular velocity starts from zero and we apply a positive voltage, we'd see the motor spin up to some constant speed following an exponential decay, then stay at that speed. If we throw in the control law $u_k = k_p(r_k - x_k)$, we can make the system converge to a desired state r_k through proportional feedback. In what manner can we pick the constant k_p that balances getting to the target angular velocity quickly with getting there efficiently (minimal oscillations or excessive voltage)?

We can solve this problem with something called the linear-quadratic regulator. We'll

define the following cost function that includes the states and inputs:

$$J = \sum_{k=0}^{\infty}(Q(r_k - x_k)^2 + Ru_k^2)$$

We want to minimize this while obeying the constraint that the system follow our flywheel dynamics $x_{k+1} = a_d x_k + b_d u_k$.

The cost is the sum of the squares of the error and the input for all time. If the controller gain k_p we pick in the control law $u_k = k_p(r_k - x_k)$ is stable, the error $r_k - x_k$ and the input u_k will both go to zero and give us a finite cost. Q and R let us decide how much the error and input contribute to the cost (we will require that $Q \geq 0$ and $R > 0$ for reasons that will be clear shortly[5]). Penalizing error more will make the controller more aggressive, while penalizing the input more will make the controller less aggressive. We want to pick a k_p that minimizes the cost.

There's a common trick for finding the value of a variable that minimizes a function of that variable. We'll take the derivative (the slope) of the cost function with respect to the input u_k, set the derivative to zero, then solve for u_k. When the slope is zero, the function is at a minimum or maximum. Now, the cost function we picked is quadratic. All the terms are strictly positive on account of the squared variables and nonnegative weights, so our cost is strictly positive and the quadratic function is concave up. The u_k we found is therefore a minimum.

The actual process of solving for u_k is mathematically intensive and outside the scope of this explanation (appendix B references a derivation for those curious). The rest of this section will describe the more general form of the linear-quadratic regulator and how to use it.

7.7.2 The mathematical definition

Instead of placing the poles of a closed-loop system manually, the linear-quadratic regulator (LQR) places the poles for us based on acceptable relative error and control effort costs. This method of controller design uses a quadratic function for the cost-to-go defined as the sum of the error and control effort over time for the linear system

[5]Lets consider the boundary conditions on the weights Q and R. If we set Q to zero, error doesn't contribute to the cost, so the optimal solution is to not move. This minimizes the sum of the inputs over time. If we let R be zero, the input doesn't contribute to the cost, so infinite inputs are allowed as they minimize the sum of the errors over time. This isn't useful, so we require that the input be penalized with a nonzero R.

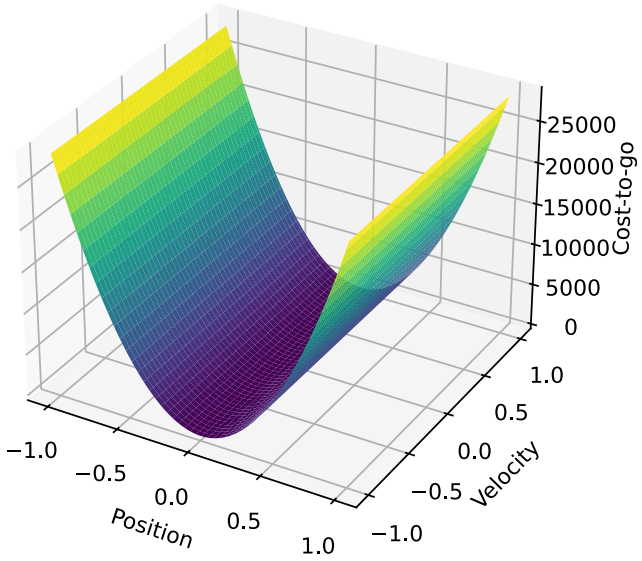


Figure 7.8: Cost-to-go for elevator model

$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$.

$$J = \sum_{k=0}^{\infty} \left(\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k\right)$$

where J represents a trade-off between state excursion and control effort with the weighting factors $\mathbf{Q}$ and $\mathbf{R}$. LQR is a control law $\mathbf{u}$ that minimizes the cost functional. Figure 7.8 shows the optimal cost-to-go for an elevator model. Pole placement, on the other hand, will have a cost-to-go above this for an arbitrary state vector (in this case, an arbitrary position-velocity pair).

The cost-to-go looks effectively constant along the velocity axis because the velocity is contributing much less to the cost-to-go than position.[6] In other words, it's much

[6]While it may not look like it at this scale, the elevator's cost-to-go along both state axes is strictly increasing away from the origin (convex upward). This means there's a global minimum at the origin, and the system is globally stable; no matter what state you start from, you're guaranteed to reach the origin with this controller.

more expensive to correct for a position error than a velocity error. This difference in cost is reflected by LQR's selected position feedback gain of $K_p = 234.041$ and selected velocity feedback gain of $K_d = 5.603$.

The minimum of LQR's cost functional is found by setting the derivative of the cost functional to zero and solving for the control law $\mathbf{u}_k$. However, matrix calculus is used instead of normal calculus to take the derivative.

The feedback control law that minimizes J is shown in theorem 7.7.1.

Theorem 7.7.1 — Linear-quadratic regulator.

$$\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k} \sum_{k=0}^{\infty} \left(\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k\right)$$
$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{7.14}$$

If the system is controllable, the optimal control policy $\mathbf{u}_k^*$ that drives all the states to zero is $-\mathbf{K}\mathbf{x}_k$. To converge to nonzero states, a reference vector $\mathbf{r}_k$ can be added to the state $\mathbf{x}_k$.

$$\mathbf{u}_k = \mathbf{K}(\mathbf{r}_k - \mathbf{x}_k) \tag{7.15}$$

This means that optimal control can be achieved with simply a set of proportional gains on all the states. To use the control law, we need knowledge of the full state of the system. That means we either have to measure all our states directly or estimate those we do not measure.

See appendix B for how $\mathbf{K}$ is calculated. If the result is finite, the controller is guaranteed to be stable and robust with a phase margin of 60 degrees [19].

R LQR design's $\mathbf{Q}$ and $\mathbf{R}$ matrices don't need discretization, but the $\mathbf{K}$ calculated for continuous time and discrete time systems will be different. The discrete time gains approach the continuous time gains as the sample period tends to zero.

7.7.3 Bryson's rule

The next obvious question is what values to choose for $\mathbf{Q}$ and $\mathbf{R}$. While this can be more of an art than a science, Bryson's rule provides a good starting point. With Bryson's rule, the diagonals of the $\mathbf{Q}$ and $\mathbf{R}$ matrices are chosen based on the maximum acceptable value for each state and actuator. The nondiagonal elements are zero. The balance between state excursion and control effort penalty ($\mathbf{Q}$ and $\mathbf{R}$ respectively)

can be adjusted using the weighting factor ρ.

$$\mathbf{Q} = \text{diag}\left(\frac{\rho}{\mathbf{x}_{max}^2}\right) \quad \mathbf{R} = \text{diag}\left(\frac{1}{\mathbf{u}_{max}^2}\right)$$

where $\mathbf{x}_{max}$ is the vector of acceptable state tolerances, $\mathbf{u}_{max}$ is the vector of acceptable input tolerances, and ρ is a tuning constant. Small values of ρ penalize control effort while large values of ρ penalize state excursions. Large values would be chosen in applications like fighter jets where performance is necessary. Spacecrafts would use small values to conserve their limited fuel supply.

7.7.4 Pole placement vs LQR

This example uses the following continuous second-order model for a CIM motor (a DC motor).

$$\mathbf{A} = \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L} & -\frac{R}{L} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix}$$

Figure 7.9 shows the response using various discrete pole placements and using LQR with the following cost matrices.

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{20^2} & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \frac{1}{12^2} \end{bmatrix}$$

With Bryson's rule, this means an angular velocity tolerance of 20 rad/s, an infinite current tolerance (in other words, we don't care what the current does), and a voltage tolerance of 12 V.

Notice with pole placement that as the current pole moves toward the origin, the control effort becomes more aggressive.

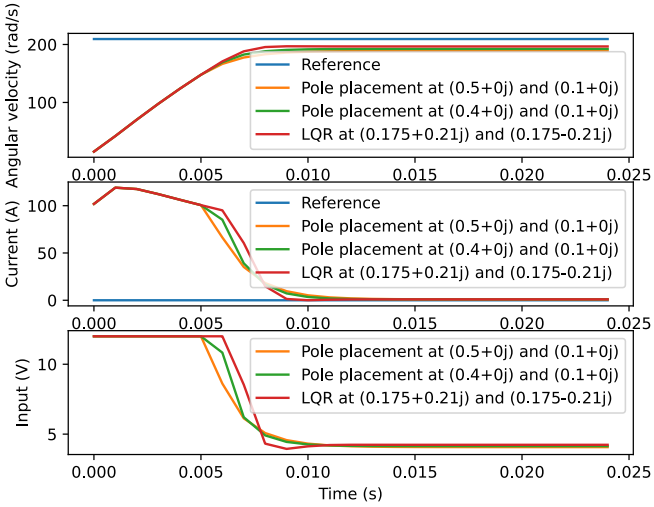


Figure 7.9: Second-order CIM motor response with pole placement and LQR

7.8 Feedforward

There are two types of feedforwards: model-based feedforward and feedforward for unmodeled dynamics. The first solves a mathematical model of the system for the inputs required to meet desired velocities and accelerations. The second compensates for unmodeled forces or behaviors directly so the feedback controller doesn't have to. Both types can facilitate simpler feedback controllers; we'll cover examples of each.

7.8.1 Plant inversion

Plant inversion is a method of model-based feedforward that solves the plant for the input that will make the plant track a desired state. This is called inversion because in a block diagram, the inverted plant feedforward and plant cancel out to produce a unity system from input to output.

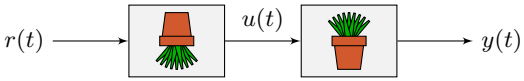


Figure 7.10: Open-loop control system with plant inversion feedforward

While it can be an effective tool, the following should be kept in mind.

1. Don't invert an unstable plant. If the expected plant doesn't match the real plant exactly, the plant inversion will still result in an unstable system. Stabilize the plant first with feedback, then inject an inversion.
2. Don't invert a nonminimum phase system. The advice for pole-zero cancellation in subsection E.2.2 applies here.

Setup

Let's start with the equation for the reference dynamics

$$\mathbf{r}_{k+1} = \mathbf{A}\mathbf{r}_k + \mathbf{B}\mathbf{u}_k$$

where $\mathbf{u}_k$ is the feedforward input. Note that this feedforward equation does not and should not take into account any feedback terms. We want to find the optimal $\mathbf{u}_k$ such that we minimize the tracking error between $\mathbf{r}_{k+1}$ and $\mathbf{r}_k$.

$$\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k = \mathbf{B}\mathbf{u}_k$$

To solve for $\mathbf{u}_k$, we need to take the inverse of the nonsquare matrix $\mathbf{B}$. This isn't possible, but we can find the pseudoinverse given some constraints on the state tracking error and control effort. To find the optimal solution for these sorts of trade-offs, one can define a cost function and attempt to minimize it. To do this, we'll first solve the expression for $\mathbf{0}$.

$$\mathbf{0} = \mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)$$

This expression will be the state tracking cost we use in the following cost function as an H_2 norm.

$$\mathbf{J} = (\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k))^\mathsf{T}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k))$$

Minimization

Given theorem 5.12.1, find the minimum of $\mathbf{J}$ by taking the partial derivative with respect to $\mathbf{u}_k$ and setting the result to $\mathbf{0}$.

$$\begin{aligned}
\frac{\partial \mathbf{J}}{\partial \mathbf{u}_k} &= 2\mathbf{B}^\mathsf{T}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) \\
\mathbf{0} &= 2\mathbf{B}^\mathsf{T}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) \\
\mathbf{0} &= 2\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{u}_k - 2\mathbf{B}^\mathsf{T}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \\
2\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{u}_k &= 2\mathbf{B}^\mathsf{T}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \\
\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{u}_k &= \mathbf{B}^\mathsf{T}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)
\end{aligned}$$

$$\mathbf{u}_k = (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)$$

$(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}$ is the left Moore-Penrose pseudoinverse of $\mathbf{B}$ denoted by $\mathbf{B}^+$.

Theorem 7.8.1 — Linear plant inversion. Given the discrete model $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$, the plant inversion feedforward is

$$\mathbf{u}_k = \mathbf{B}^+(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \tag{7.16}$$

where $\mathbf{B}^+$ is the Moore-Penrose pseudoinverse of $\mathbf{B}$, $\mathbf{r}_{k+1}$ is the reference at the next timestep, and $\mathbf{r}_k$ is the reference at the current timestep.

Discussion

Linear plant inversion in theorem 7.8.1 compensates for reference dynamics that don't follow how the model inherently behaves. If they do follow the model, the feedforward has nothing to do as the model already behaves in the desired manner. When this occurs, $\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k$ will return a zero vector.

For example, a constant reference requires a feedforward that opposes system dynamics that would change the state over time. If the system has no dynamics, then $\mathbf{A} = \mathbf{I}$ and thus

$$\begin{aligned}\mathbf{u}_k &= \mathbf{B}^+(\mathbf{r}_{k+1} - \mathbf{I}\mathbf{r}_k)\\ \mathbf{u}_k &= \mathbf{B}^+(\mathbf{r}_{k+1} - \mathbf{r}_k)\end{aligned}$$

For a constant reference, $\mathbf{r}_{k+1} = \mathbf{r}_k$.

$$\begin{aligned}\mathbf{u}_k &= \mathbf{B}^+(\mathbf{r}_k - \mathbf{r}_k)\\ \mathbf{u}_k &= \mathbf{B}^+(\mathbf{0})\\ \mathbf{u}_k &= \mathbf{0}\end{aligned}$$

so no feedforward is required to hold a system with no dynamics at a constant reference, as expected.

Figure 7.11 shows plant inversion applied to a second-order CIM motor model. Plant inversion accounts for the motor back-EMF and eliminates steady-state error.

7.8.2 Unmodeled dynamics

In addition to plant inversion, one can include feedforwards for unmodeled dynamics. Consider an elevator model which doesn't include gravity. A constant voltage offset

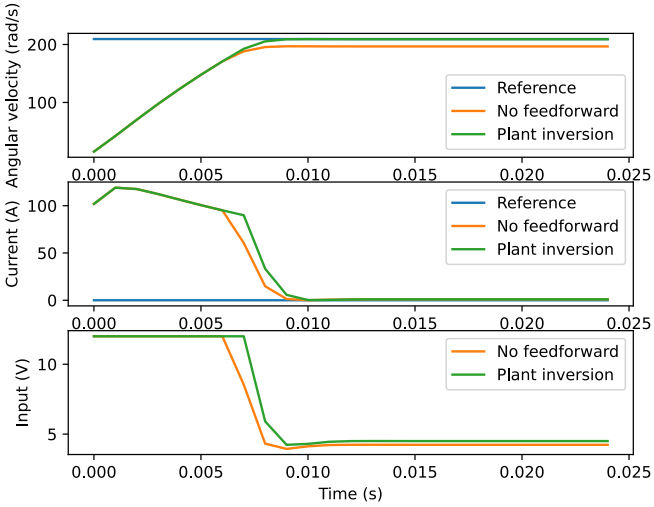


Figure 7.11: Second-order CIM motor response with plant inversion

can be used compensate for this. The feedforward takes the form of a voltage constant because voltage is proportional to force applied, and the force is acting in only one direction at all times.

$$u_k = V_{app} \tag{7.17}$$

where V_{app} is a constant. Another feedforward holds a single-jointed arm steady in the presence of gravity. It has the following form.

$$u_k = V_{app} \cos\theta \tag{7.18}$$

where V_{app} is the voltage required to keep the single-jointed arm level with the ground, and θ is the angle of the arm relative to the ground. Therefore, the force applied is greatest when the arm is parallel with the ground and zero when the arm is perpendicular to the ground (at that point, the joint supports all the weight).

Note that the elevator model could be augmented easily enough to include gravity and still be linear, but this wouldn't work for the single-jointed arm since a trigonometric function is required to model the gravitational force in the arm's rotating reference frame.[7]

[7]While the applied torque of the motor is constant throughout the arm's range of motion, the torque caused by gravity in the opposite direction varies according to the arm's angle.

7.8.3 Do feedforwards affect stability?

Feedforwards have no effect on stability because they don't use the system state. We'll demonstrate this for a plant inversion feedforward.

Let $\mathbf{u}_k = \mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)$ be a feedback controller for the discrete model $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$.

$$
\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}(\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)) \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k) \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k - \mathbf{B}\mathbf{K}\mathbf{x}_k \\
\mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k
\end{aligned}
$$

The system is stable if the eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ are within the unit circle. Now add the plant inversion feedforward controller $\mathbf{B}^+(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)$ to $\mathbf{u}_k$.

$$
\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}(\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k) + \mathbf{B}^+(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k) + \mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k \\
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k - \mathbf{B}\mathbf{K}\mathbf{x}_k + \mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k \\
\mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k + \mathbf{r}_{k+1} + (\mathbf{B}\mathbf{K} - \mathbf{A})\mathbf{r}_k \\
\mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k + \mathbf{r}_{k+1} - (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{r}_k
\end{aligned}
$$

The multiplicative term on $\mathbf{x}_k$ is still $\mathbf{A} - \mathbf{B}\mathbf{K}$, so the feedforward didn't affect stability. It still affects the system response and steady-state error though.

7.9 Numerical integration methods

Most systems don't have linear dynamics and their differential equations can't be solved analytically. Instead, we'll have to approximate their solutions with numerical integration.

7.9.1 Butcher tableaus

Butcher tableaus are a more succinct representation for explicit and implicit Runge-Kutta numerical integration methods. Here's the general structure for explicit methods.

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{2,1} & & & \\ \vdots & \vdots & \ddots & & \\ c_s & a_{s,1} & \dots & a_{s,s-1} & \\ \hline & b_1 & \dots & \dots & b_s \end{array}$$

where s is the number of stages in the method, the matrix $[a_{ij}]$ is the Runge-Kutta matrix, $b_1, \dots, b_s$ are the weights, and $c_1, \dots, c_s$ are the nodes. The top-left quadrant contains the sums of the rows in the top-right quadrant. Each column in the right half corresponds to a $\mathbf{k}$ coefficient from $\mathbf{k}_1$ to $\mathbf{k}_s$.

The family of solutions to $\dot{\mathbf{x}} = f(t, \mathbf{x})$ is given by

$$\begin{aligned} \mathbf{k}_1 &= f(t_k, \mathbf{x}_k) \\ \mathbf{k}_2 &= f(t_k + c_2 h, \mathbf{x}_k + h(a_{2,1}\mathbf{k}_1)) \\ &\vdots \\ \mathbf{k}_s &= f(t_k + c_s h, \mathbf{x}_k + h(a_{s,1}\mathbf{k}_1 + \ldots + a_{s,s-1}\mathbf{k}_{s-1})) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + h\sum_{i=1}^{s} b_i \mathbf{k}_i \end{aligned}$$

where h is the timestep duration.

7.9.2 Forward Euler method

The simplest explicit Runge-Kutta integration method is forward Euler integration. We don't recommend using it because it suffers from numerical stability issues. We'll demonstrate how to translate its Butcher tableau into equations that integrate $\dot{\mathbf{x}} = f(t, \mathbf{x})$ from 0 to h.

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

$$\begin{aligned} \mathbf{k}_1 &= f(t+0h, \mathbf{x}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + h(1\mathbf{k}_1) \end{aligned}$$

Remove zeroed out terms.

$$\begin{aligned}\mathbf{k}_1 &= f(t, \mathbf{x}_k)\\ \mathbf{x}_{k+1} &= \mathbf{x}_k + h\mathbf{k}_1\end{aligned}$$

Simplify.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + hf(t, \mathbf{x}_k)$$

In FRC, our differential equations are of the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ where $\mathbf{u}$ is held constant between timesteps. Since it's time-invariant, we can ignore the time argument of the integration method. This gives theorem 7.9.1.

Theorem 7.9.1 — Forward Euler integration. Given the differential equation $\dot{\mathbf{x}} = f(\mathbf{x}_k, \mathbf{u}_k)$, this method solves for $\mathbf{x}_{k+1}$ at h seconds in the future. $\mathbf{u}$ is assumed to be held constant between timesteps.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + hf(\mathbf{x}_k, \mathbf{u}_k)$$

0	
	1

7.9.3 Runge-Kutta fourth-order method

The most common method we'll cover is Runge-Kutta fourth-order (RK4). It's simple and accurate for most systems we'll see in FRC. We'll demonstrate how to translate its Butcher tableau into equations that integrate $\dot{\mathbf{x}} = f(t, \mathbf{x})$ from 0 to h.

$$\begin{aligned}\mathbf{k}_1 &= f(t+0h, \mathbf{x}_k)\\ \mathbf{k}_2 &= f(t+\frac{1}{2}h, \mathbf{x}_k + h(\frac{1}{2}\mathbf{k}_1))\\ \mathbf{k}_3 &= f(t+\frac{1}{2}h, \mathbf{x}_k + h(0\mathbf{k}_1 + \frac{1}{2}\mathbf{k}_2))\\ \mathbf{k}_4 &= f(t+1h, \mathbf{x}_k + h(0\mathbf{k}_1 + 0\mathbf{k}_2 + 1\mathbf{k}_3))\\ \mathbf{x}_{k+1} &= \mathbf{x}_k + h(\frac{1}{6}\mathbf{k}_1 + \frac{1}{3}\mathbf{k}_2 + \frac{1}{3}\mathbf{k}_3 + \frac{1}{6}\mathbf{k}_4)\end{aligned}$$

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Remove zeroed out terms.

$$\mathbf{k}_1 = f(t, \mathbf{x}_k)$$

$$\begin{aligned}
\mathbf{k}_2 &= f(t + \frac{1}{2}h, \mathbf{x}_k + h\frac{1}{2}\mathbf{k}_1) \\
\mathbf{k}_3 &= f(t + \frac{1}{2}h, \mathbf{x}_k + h\frac{1}{2}\mathbf{k}_2) \\
\mathbf{k}_4 &= f(t + h, \mathbf{x}_k + h\mathbf{k}_3) \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + h\left(\frac{1}{6}\mathbf{k}_1 + \frac{1}{3}\mathbf{k}_2 + \frac{1}{3}\mathbf{k}_3 + \frac{1}{6}\mathbf{k}_4\right)
\end{aligned}$$

$\frac{1}{6}$ is usually factored out of the last equation to reduce the number of floating point operations.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

In FRC, our differential equations are of the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ where $\mathbf{u}$ is held constant between timesteps. Since it's time-invariant, we can ignore the time argument of the integration method. This gives theorem 7.9.2.

Theorem 7.9.2 — Runge-Kutta fourth-order integration. Given the differential equation $\dot{\mathbf{x}} = f(\mathbf{x}_k, \mathbf{u}_k)$, this method solves for $\mathbf{x}_{k+1}$ at h seconds in the future. $\mathbf{u}$ is assumed to be held constant between timesteps.

$$\begin{aligned}
\mathbf{k}_1 &= f(\mathbf{x}_k, \mathbf{u}_k) \\
\mathbf{k}_2 &= f(\mathbf{x}_k + h\frac{1}{2}\mathbf{k}_1, \mathbf{u}_k) \\
\mathbf{k}_3 &= f(\mathbf{x}_k + h\frac{1}{2}\mathbf{k}_2, \mathbf{u}_k) \\
\mathbf{k}_4 &= f(\mathbf{x}_k + h\mathbf{k}_3, \mathbf{u}_k) \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + h\frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned}$$

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Here's a reference implementation.

```
#include <Eigen/Core>
#include <units/time.h>

/**
 * Performs 4th order Runge-Kutta integration of dx/dt = f(x, u) for dt.
 *
 * @param f  The function to integrate. It must take two arguments x and u.
```

```
 * @param x  The initial value of x.
 * @param u  The value u held constant over the integration period.
 * @param dt The time over which to integrate.
 */
template <typename F, typename T, typename U>
T RK4(F&& f, T x, U u, units::second_t dt) {
  const auto h = dt.value();

  T k1 = f(x, u);
  T k2 = f(x + h * 0.5 * k1, u);
  T k3 = f(x + h * 0.5 * k2, u);
  T k4 = f(x + h * k3, u);

  return x + h / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4);
}
```

Snippet 7.1. RK4 implementation in C++

Other methods of Runge-Kutta integration exist with various properties [1], but the one presented here is popular for its high accuracy relative to the amount of floating point operations (FLOPs) it requires.

7.9.4 Dormand-Prince method

Dormand-Prince (RKDP) is a fourth-order method with fifth-order error checking. It uses an adaptive stepsize to enforce an upper bound on the integration error.

Theorem 7.9.3 — Dormand-Prince integration. Given the differential equation $\dot{\mathbf{x}} = f(\mathbf{x}_k, \mathbf{u}_k)$, this method solves for $\mathbf{x}_{k+1}$ at h seconds in the future. $\mathbf{u}$ is assumed to be held constant between timesteps. It has the following Butcher tableau.

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

The first row of coefficients below the table divider gives the fifth-order accurate solution. The second row gives an alternative solution which, when subtracted from the first solution, gives the error estimate.

Here's a reference implementation.

```
#include <algorithm>
#include <array>
#include <cmath>

#include <Eigen/Core>
#include <units/time.h>

/**
 * Performs adaptive Dormand-Prince integration of dx/dt = f(x, u) for dt.
 *
 * @param f        The function to integrate. It must take two arguments x
     and
 *                 u.
 * @param x        The initial value of x.
 * @param u        The value u held constant over the integration period.
 * @param dt       The time over which to integrate.
 * @param maxError The maximum acceptable truncation error. Usually a small
 *                 number like 1e-6.
 */
template <typename F, typename T, typename U>
T RKDP(F&& f, T x, U u, units::second_t dt, double maxError = 1e-6) {
  // See https://en.wikipedia.org/wiki/Dormand%E2%80%93Prince_method for the
  // Butcher tableau the following arrays came from.

  constexpr int kDim = 7;

  // clang-format off
  constexpr double A[kDim - 1][kDim - 1]{
      {       1.0 / 5.0},
      {       3.0 / 40.0,         9.0 / 40.0},
      {      44.0 / 45.0,       -56.0 / 15.0,        32.0 / 9.0},
      {19372.0 / 6561.0, -25360.0 / 2187.0, 64448.0 / 6561.0, -212.0 /
          729.0},
      { 9017.0 / 3168.0,      -355.0 / 33.0, 46732.0 / 5247.0,   49.0 / 176.0,
           -5103.0 / 18656.0},
      {    35.0 / 384.0,                0.0,   500.0 / 1113.0,  125.0 / 192.0,
            -2187.0 / 6784.0, 11.0 / 84.0}};
  // clang-format on

  constexpr std::array<double, kDim> b1{
      35.0 / 384.0, 0.0, 500.0 / 1113.0, 125.0 / 192.0, -2187.0 / 6784.0,
      11.0 / 84.0,  0.0};
  constexpr std::array<double, kDim> b2{5179.0 / 57600.0,    0.0,
                                        7571.0 / 16695.0,    393.0 / 640.0,
                                        -92097.0 / 339200.0, 187.0 / 2100.0,
                                        1.0 / 40.0};

  T newX;
  double truncationError;

  double dtElapsed = 0.0;
  double h = dt.value();

  // Loop until we've gotten to our desired dt
  while (dtElapsed < dt.value()) {
    do {
      // Only allow us to advance up to the dt remaining
      h = std::min(h, dt.value() - dtElapsed);
```

```
      // clang-format off
      T k1 = f(x, u);
      T k2 = f(x + h * (A[0][0] * k1), u);
      T k3 = f(x + h * (A[1][0] * k1 + A[1][1] * k2), u);
      T k4 = f(x + h * (A[2][0] * k1 + A[2][1] * k2 + A[2][2] * k3), u);
      T k5 = f(x + h * (A[3][0] * k1 + A[3][1] * k2 + A[3][2] * k3 + A[3][3]
           * k4), u);
      T k6 = f(x + h * (A[4][0] * k1 + A[4][1] * k2 + A[4][2] * k3 + A[4][3]
           * k4 + A[4][4] * k5), u);
      // clang-format on

      // Since the final row of A and the array b1 have the same coefficients
      // and k7 has no effect on newX, we can reuse the calculation.
      newX = x + h * (A[5][0] * k1 + A[5][1] * k2 + A[5][2] * k3 +
                      A[5][3] * k4 + A[5][4] * k5 + A[5][5] * k6);
      T k7 = f(newX, u);

      truncationError = (h * ((b1[0] - b2[0]) * k1 + (b1[1] - b2[1]) * k2 +
                              (b1[2] - b2[2]) * k3 + (b1[3] - b2[3]) * k4 +
                              (b1[4] - b2[4]) * k5 + (b1[5] - b2[5]) * k6 +
                              (b1[6] - b2[6]) * k7))
                            .norm();

      h *= 0.9 * std::pow(maxError / truncationError, 1.0 / 5.0);
    } while (truncationError > maxError);

    dtElapsed += h;
    x = newX;
  }

  return x;
}
```

Snippet 7.2. RKDP implementation in C++

This page intentionally left blank


Trees by Interdisciplinary Sciences building at UCSC

8. Nonlinear control

While many tools exist for designing controllers for linear systems, all systems in reality are inherently nonlinear. We'll briefly mention some considerations for nonlinear systems.

8.1 Introduction

Recall from linear system theory that we defined systems as having the following form:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{v}$$

In this equation, $\mathbf{A}$ and $\mathbf{B}$ are constant matrices, which means they are both time-invariant and linear (all transformations on the system state are linear ones, and those transformations remain the same for all time). In nonlinear and time-variant systems, the state evolution and output are defined by arbitrary functions of the current states and inputs.

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w})$$
$$\mathbf{y} = h(\mathbf{x}, \mathbf{u}, \mathbf{v})$$

Nonlinear functions come up regularly when attempting to control the pose of a vehicle in the global coordinate frame instead of the vehicle's rotating local coordinate frame.

Converting from one to the other requires applying a rotation matrix, which consists of sine and cosine operations. These functions are nonlinear.

8.2 Linearization

One way to control nonlinear systems is to linearize the model around a reference point. Then, all the powerful tools that exist for linear controls can be applied. This is done by taking the Jacobians of f and h with respect to the state and input vectors. See section 5.12 for more on Jacobians.

$$\mathbf{A} = \frac{\partial f(\mathbf{x}, \mathbf{u}, \mathbf{w})}{\partial \mathbf{x}} \qquad \mathbf{B} = \frac{\partial f(\mathbf{x}, \mathbf{u}, \mathbf{w})}{\partial \mathbf{u}}$$
$$\mathbf{C} = \frac{\partial h(\mathbf{x}, \mathbf{u}, \mathbf{v})}{\partial \mathbf{x}} \qquad \mathbf{D} = \frac{\partial h(\mathbf{x}, \mathbf{u}, \mathbf{v})}{\partial \mathbf{u}}$$

Linearization of a nonlinear equation is a Taylor series expansion to only the first-order terms (that is, terms whose variables have exponents on the order of x^1). This is where the small angle approximations for $\sin\theta$ and $\cos\theta$ (θ and 1 respectively) come from.

Higher order partial derivatives can be added to better approximate the nonlinear dynamics. We typically only linearize around equilibrium points[1] because we are interested in how the system behaves when perturbed from equilibrium. An FAQ on this goes into more detail [5]. To be clear though, linearizing the system around the current state as the system evolves does give a closer approximation over time.

Note that linearization with static matrices (that is, with a time-invariant linear system) only works if the original system in question is feedback linearizable.

8.3 Lyapunov stability

Lyapunov stability is a fundamental concept in nonlinear control, so we're going to give a brief overview of what it is so students can research it further.

Since the state evolution in nonlinear systems is defined by a function rather than a constant matrix, the system's poles as determined by linearization move around. Nonlinear control uses Lyapunov stability to determine if nonlinear systems are stable. From a linear control theory point of view, Lyapunov stability says the system is stable if, for a given initial condition, all possible eigenvalues of $\mathbf{A}$ from that point on remain in the left-half plane. However, nonlinear control uses a different definition.

[1] Equilibrium points are points where $\dot{\mathbf{x}} = \mathbf{0}$. At these points, the system is in steady-state.

Lyapunov stability means that the system trajectory can be kept arbitrarily close to the origin by starting sufficiently close to it. Lyapunov's direct method uses a function consisting of the energy in a system or derivatives of the system's state to prove stability around an equilibrium point. This is done by showing that the function, and thus its inputs, decay to some ground state. More rigorously, the value function $V(\mathbf{x})$ must be positive definite and equal zero at the equilibrium point

$$\begin{aligned} V(\mathbf{x}) &> 0 \\ V(\mathbf{0}) &= 0 \end{aligned}$$

and its derivative $\dot{V}(\mathbf{x})$ must be negative definite.

$$\dot{V}(\mathbf{x}) = \frac{dV}{dt} = \frac{\partial V}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt} \leq 0$$

More than one Lyapunov function can prove stability, and if one function doesn't prove it, another candidate should be tried. For this reason, we refer to these functions as *Lyapunov candidate functions*.

8.3.1 Lyapunov stability for linear systems

We're going to find stability criteria for the linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ using Lyapunov theory. Let's use the following Lyapunov candidate function.

$$V(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{P}\mathbf{x} \text{ where } \mathbf{P} = \mathbf{P}^\mathsf{T} > \mathbf{0}$$

This function is positive definite by definition. Its derivative is

$$\begin{aligned} \dot{V}(\mathbf{x}) &= \dot{\mathbf{x}}^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{x}^\mathsf{T}\dot{\mathbf{P}}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{P}\dot{\mathbf{x}} \\ \dot{V}(\mathbf{x}) &= \dot{\mathbf{x}}^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{0}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{P}\dot{\mathbf{x}} \\ \dot{V}(\mathbf{x}) &= \dot{\mathbf{x}}^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{P}\dot{\mathbf{x}} \\ \dot{V}(\mathbf{x}) &= (\mathbf{A}\mathbf{x})^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{P}(\mathbf{A}\mathbf{x}) \\ \dot{V}(\mathbf{x}) &= \mathbf{x}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{P}\mathbf{A}\mathbf{x} \\ \dot{V}(\mathbf{x}) &= \mathbf{x}^\mathsf{T}(\mathbf{A}^\mathsf{T}\mathbf{P} + \mathbf{P}\mathbf{A})\mathbf{x} \end{aligned}$$

For this function to be negative definite, $\mathbf{A}^\mathsf{T}\mathbf{P} + \mathbf{P}\mathbf{A}$ must be negative definite. Since $\mathbf{P}$ is positive definite, the only way to satisfy that condition is if $\mathbf{A}$ is negative definite (i.e., $\mathbf{A}$ is stable).

8.4 Affine systems

Let $\mathbf{x} = \mathbf{x}_0 + \delta\mathbf{x}$ and $\mathbf{u} = \mathbf{u}_0 + \delta\mathbf{u}$ where $\delta\mathbf{x}$ and $\delta\mathbf{u}$ are perturbations from $(\mathbf{x}_0, \mathbf{u}_0)$. A first-order linearization of $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ around $(\mathbf{x}_0, \mathbf{u}_0)$ gives

$$\dot{\mathbf{x}} \approx f(\mathbf{x}_0, \mathbf{u}_0) + \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\mathbf{x}_0, \mathbf{u}_0} \delta\mathbf{x} + \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}_0, \mathbf{u}_0} \delta\mathbf{u}$$

$$\dot{\mathbf{x}} = f(\mathbf{x}_0, \mathbf{u}_0) + \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\mathbf{x}_0, \mathbf{u}_0} \delta\mathbf{x} + \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}_0, \mathbf{u}_0} \delta\mathbf{u}$$

An affine system is a linear system with a constant offset in the dynamics. If $(\mathbf{x}_0, \mathbf{u}_0)$ is an equilibrium point, $f(\mathbf{x}_0, \mathbf{u}_0) = \mathbf{0}$, the resulting model is linear, and LQR works as usual. If $(\mathbf{x}_0, \mathbf{u}_0)$ is, say, the current operating point rather than an equilibrium point, the easiest way to correctly apply LQR is

1. Find a control input $\mathbf{u}_0$ that makes $(\mathbf{x}_0, \mathbf{u}_0)$ an equilibrium point.
2. Obtain an LQR for the linearized system.
3. Add $\mathbf{u}_0$ to the LQR's control input.

A control-affine system is of the form $\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}$. Since it has separable control inputs, $\mathbf{u}_0$ can be derived via plant inversion as follows.

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}_0) + g(\mathbf{x}_0)\mathbf{u}_0 \\ \mathbf{0} &= f(\mathbf{x}_0) + g(\mathbf{x}_0)\mathbf{u}_0 \\ g(\mathbf{x}_0)\mathbf{u}_0 &= -f(\mathbf{x}_0) \\ \mathbf{u}_0 &= -g^{-1}(\mathbf{x}_0) f(\mathbf{x}_0) \end{aligned} \tag{8.1}$$

For the control-affine system $\dot{\mathbf{x}} = f(\mathbf{x}) + \mathbf{B}\mathbf{u}$, $\mathbf{u}_0$ would be

$$\mathbf{u}_0 = -\mathbf{B}^+ f(\mathbf{x}_0) \tag{8.2}$$

8.4.1 Feedback linearization for reference tracking

Feedback linearization lets us erase the nonlinear dynamics of a system so we can apply our own (usually linear) dynamics for reference tracking. To do this, we will perform a similar procedure as in subsection 7.8.1 and solve for $\mathbf{u}$ given the reference dynamics in $\dot{\mathbf{r}}$.

$$\begin{aligned} \dot{\mathbf{r}} &= f(\mathbf{x}) + \mathbf{B}\mathbf{u} \\ \mathbf{B}\mathbf{u} &= \dot{\mathbf{r}} - f(\mathbf{x}) \\ \mathbf{u} &= \mathbf{B}^+(\dot{\mathbf{r}} - f(\mathbf{x})) \end{aligned} \tag{8.3}$$

R To use equation (8.3) in a discrete controller, one can approximate $\dot{\mathbf{r}}$ with $\frac{\mathbf{r}_{k+1}-\mathbf{r}_k}{T}$ where T is the time period between the two references.

8.4.2 Affine system discretization

We're going to discretize the following continuous time state-space model with a zero-order hold.

$$\dot{\mathbf{x}} = \mathbf{A}_c\mathbf{x} + \mathbf{B}_c\mathbf{u} + \mathbf{c}$$

Since $\mathbf{u}$ and $\mathbf{c}$ are held constant between updates, we can treat them as the aggregated input of the linear model $\dot{\mathbf{x}} = \mathbf{A}_c\mathbf{x} + \mathbf{B}_c\mathbf{u}$ and use the zero-order hold from theorem 7.3.1.

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}_c\mathbf{x} + \mathbf{B}_c(\mathbf{u} + \mathbf{B}_c^+\mathbf{c}) \\
\mathbf{x}_{k+1} &= \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d(\mathbf{u}_k + \mathbf{B}_c^+\mathbf{c}_k) \\
\mathbf{x}_{k+1} &= \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k + \mathbf{B}_d\mathbf{B}_c^+\mathbf{c}_k
\end{aligned} \tag{8.4}$$

See theorem 7.3.1 for how to compute $\mathbf{A}_d$ and $\mathbf{B}_d$.

8.5 Pendulum

8.5.1 State-space model

Below is the model for a pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

where θ is the angle of the pendulum and l is the length of the pendulum.

Since state-space representation requires that only single derivatives be used, they should be broken up as separate states. We'll reassign $\dot{\theta}$ to be ω so the derivatives are easier to keep straight for state-space representation.

$$\dot{\omega} = -\frac{g}{l}\sin\theta$$

Now separate the states.

$$\begin{aligned}
\dot{\theta} &= \omega \\
\dot{\omega} &= -\frac{g}{l}\sin\theta
\end{aligned}$$

This makes our state vector $\begin{bmatrix}\theta & \omega\end{bmatrix}^\mathsf{T}$ and our nonlinear model the following.

$$f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \omega \\ -\frac{g}{l}\sin\theta \end{bmatrix}$$

Linearization around $\theta = 0$

To apply our tools for linear control theory, the model must be a linear combination of the states and inputs (addition and multiplication by constants). Since this model is nonlinear on account of the sine function, we should linearize it.

Linearization finds a tangent line to the nonlinear dynamics at a desired point in the state-space. The Taylor series is a way to approximate arbitrary functions with polynomials, so we can use it for linearization.

The taylor series expansion for $\sin\theta$ around $\theta = 0$ is $\theta - \frac{1}{6}\theta^3 + \frac{1}{120}\theta^5 - \ldots$. We'll take just the first-order term θ to obtain a linear function.

$$\begin{aligned}\dot{\theta} &= \omega \\ \dot{\omega} &= -\frac{g}{l}\theta\end{aligned}$$

Now write the model in state-space representation. We'll write out the system of equations with the zeroed variables included to assist with this.

$$\begin{aligned}\dot{\theta} &= \quad 0\theta + 1\omega \\ \dot{\omega} &= -\frac{g}{l}\theta + 0\omega\end{aligned}$$

Factor out θ and ω into a column vector.

$$\begin{bmatrix}\dot{\theta} \\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ -\frac{g}{l} & 0\end{bmatrix}\begin{bmatrix}\theta \\ \omega\end{bmatrix} \tag{8.5}$$

Linearization with the Jacobian

Here's the original nonlinear model in state-space representation.

$$f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix}\omega \\ -\frac{g}{l}\sin\theta\end{bmatrix}$$

If we want to linearize around an arbitrary point, we can take the Jacobian with respect to $\mathbf{x}$.

$$\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{bmatrix}0 & 1 \\ -\frac{g}{l}\cos\theta & 0\end{bmatrix}$$

For full state feedback, knowledge of all states is required. If not all states are measured directly, an estimator can be used to supplement them.

We may only be measuring θ in the pendulum example, not $\dot{\theta}$, so we'll need to estimate the latter. The $\mathbf{C}$ matrix the observer would use in this case is

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Therefore, the output vector is

$$\begin{aligned} \mathbf{y} &= \mathbf{C}\mathbf{x} \\ \mathbf{y} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} \\ \mathbf{y} &= 1\theta + 0\omega \\ \mathbf{y} &= \theta \end{aligned}$$

8.6 Holonomic drivetrains

8.6.1 Model

Holonomic drivetrains have three degrees of freedom: x, y, and heading. They are described by the following kinematics.

$$\begin{bmatrix} \dot{x} \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{x,chassis} \\ v_{y,chassis} \\ \omega_{chassis} \end{bmatrix}$$

where $v_{x,chassis}$ is the velocity ahead in the chassis frame, $v_{y,chassis}$ is the velocity to the left in the chassis frame, and $\omega_{chassis}$ is the angular velocity in the chassis frame. This can be written in state-space notation as

$$\begin{bmatrix} \dot{x} \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{x,chassis} \\ v_{y,chassis} \\ \omega_{chassis} \end{bmatrix}$$

8.6.2 Control

This control-affine model is fully actuated but nonlinear in the chassis frame. However, we can apply linear control theory to the error dynamics in the global frame instead. Note how equation (8.6)'s state vector contains the global pose and its input vector contains the global linear and angular velocities.

$$\begin{bmatrix} \dot{x} \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{x,global} \\ v_{y,global} \\ \omega_{global} \end{bmatrix} \tag{8.6}$$

$$\text{where } \begin{bmatrix} v_{x,global} \\ v_{y,global} \\ \omega_{global} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{x,chassis} \\ v_{y,chassis} \\ \omega_{chassis} \end{bmatrix} \tag{8.7}$$

We can control the model in equation (8.6) with an LQR, which will have three independent proportional controllers. Then, we can convert the global velocity commands to chassis velocity commands with equation (8.7) and convert the chassis velocity commands to wheel speed commands with inverse kinematics. LQRs on each wheel can track the wheel speed commands.

Note that the full control law is nonlinear because the kinematics contain a rotation matrix for transforming from the chassis frame to the global frame. However, the nonlinear part has been abstracted away from the tunable linear control laws.

8.6.3 Holonomic vs nonholonomic control

Drivetrains that are unable to exercise all possible degrees of freedom (e.g., moving sideways with respect to the chassis) are nonholonomic. An LQR on each degree of freedom is ideal for holonomic drivetrains, but not for nonholonomic. Section 8.7 will use the differential drive as a motivating example for various nonholonomic controllers.

R Nonholonomic controllers should not be used for holonomic drivetrains. They make different assumptions about the drivetrain dynamics that yield suboptimal results compared to holonomic controllers.

8.7 Differential drive

This drivetrain consists of two DC motors per side which are chained together on their respective sides and drive wheels which are assumed to be massless.

8.7.1 Velocity subspace state-space model

By equations (12.35) and (12.36)

$$\dot{v}_l = \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_1 v_l + C_2 V_l) + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_3 v_r + C_4 V_r)$$
$$\dot{v}_r = \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_1 v_l + C_2 V_l) + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_3 v_r + C_4 V_r)$$

Regroup the terms into states v_l and v_r and inputs V_l and V_r.

$$\dot{v}_l = \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 v_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 V_l$$

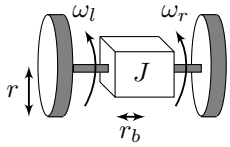


Figure 8.1: Differential drive dimensions

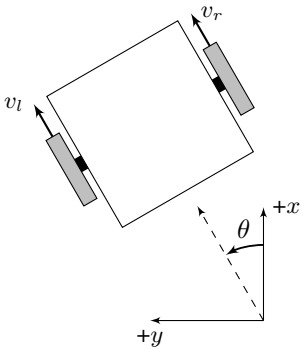


Figure 8.2: Differential drive coordinate frame

$$
\begin{aligned}
&+ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 v_r + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 V_r \\
\dot{v}_r = &\left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 v_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 V_l \\
&+ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 v_r + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 V_r \\
\dot{v}_l = &\left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 v_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 v_r \\
&+ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 V_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 V_r \\
\dot{v}_r = &\left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 v_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 v_r \\
&+ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 V_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 V_r
\end{aligned}
$$

Factor out v_l and v_r into a column vector and V_l and V_r into a column vector.

$$
\begin{aligned}
\begin{bmatrix} \dot{v}_l \\ v_r \end{bmatrix} = &\begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix} \\
&+ \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \begin{bmatrix} V_l \\ V_r \end{bmatrix}
\end{aligned}
$$

Theorem 8.7.1 — Differential drive velocity state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{bmatrix} \text{left velocity} \\ \text{right velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \tag{8.8}$$

$$\mathbf{B} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \tag{8.9}$$

where $C_1 = -\frac{G_l^2 K_t}{K_v R r^2}$, $C_2 = \frac{G_l K_t}{Rr}$, $C_3 = -\frac{G_r^2 K_t}{K_v R r^2}$, and $C_4 = \frac{G_r K_t}{Rr}$.

Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[2] creates and tests a controller for it. Figure 8.3 shows the closed-loop system response.

Given the high inertia in drivetrains, it's better to drive the reference with a motion profile instead of a step input for reproducibility.

8.7.2 Heading state-space model

We can control heading by augmenting the state with that. The change in heading is defined as

$$\dot{\theta} = \frac{v_r - v_l}{2r_b} = \frac{v_r}{2r_b} - \frac{v_l}{2r_b}$$

This gives the following linear model.

[2] https://github.com/calcmogul/frccontrol/blob/main/examples/differential_drive.py

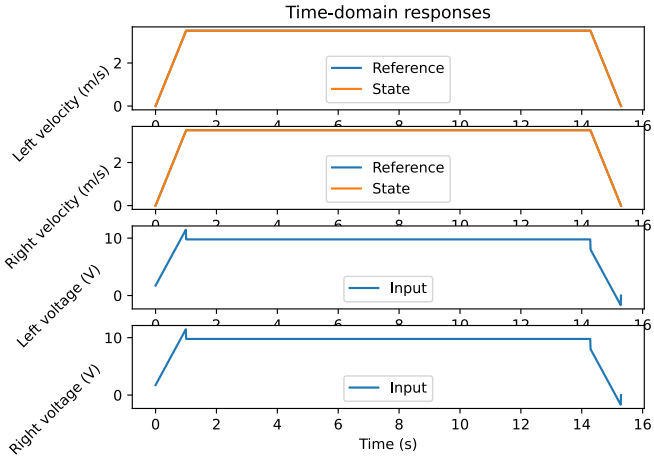


Figure 8.3: Drivetrain response

Theorem 8.7.2 — Differential drive heading state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} \theta \\ v_l \\ v_r \end{bmatrix} = \begin{bmatrix} \text{heading} \\ \text{left velocity} \\ \text{right velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\ 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \tag{8.10}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \tag{8.11}$$

where $C_1 = -\frac{G_l^2 K_t}{K_v R r^2}$, $C_2 = \frac{G_l K_t}{Rr}$, $C_3 = -\frac{G_r^2 K_t}{K_v R r^2}$, and $C_4 = \frac{G_r K_t}{Rr}$. The constants C_1 through C_4 are from the derivation in section 12.6.

The velocity states are required to make the heading controllable.

8.7.3 Linear time-varying model

We can control the drivetrain's global pose (x, y, θ) by augmenting the state with x and y. The change in global pose is defined by these three equations.

$$\begin{aligned}
\dot{x} &= \frac{v_l + v_r}{2}\cos\theta = \frac{v_r}{2}\cos\theta + \frac{v_l}{2}\cos\theta \\
\dot{y} &= \frac{v_l + v_r}{2}\sin\theta = \frac{v_r}{2}\sin\theta + \frac{v_l}{2}\sin\theta \\
\dot{\theta} &= \frac{v_r - v_l}{2r_b} = \frac{v_r}{2r_b} - \frac{v_l}{2r_b}
\end{aligned}$$

This augmented model is a nonlinear vector function where $\mathbf{x} = \begin{bmatrix} x & y & \theta & v_l & v_r \end{bmatrix}^\mathsf{T}$ and $\mathbf{u} = \begin{bmatrix} V_l & V_r \end{bmatrix}^\mathsf{T}$.

$$f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix}
\frac{v_r}{2}\cos\theta + \frac{v_l}{2}\cos\theta \\
\frac{v_r}{2}\sin\theta + \frac{v_l}{2}\sin\theta \\
\frac{v_r}{2r_b} - \frac{v_l}{2r_b} \\
\left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_1 v_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_3 v_r + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_2 V_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_4 V_r \\
\left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_1 v_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_3 v_r + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_2 V_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_4 V_r
\end{bmatrix} \tag{8.12}$$

As mentioned in chapter 8, one can approximate a nonlinear system via linearizations around points of interest in the state-space and design controllers for those linearized subspaces. If we sample linearization points progressively closer together, we converge on a control policy for the original nonlinear system. Since the linear plant being controlled varies with time, its controller is called a linear time-varying (LTV) controller.

If we use LQRs for the linearized subspaces, the nonlinear control policy will also be locally optimal. We'll be taking this approach with a differential drive. To create an LQR, we need to linearize equation (8.12).

$$\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{bmatrix}
0 & 0 & -\frac{v_l + v_r}{2}\sin\theta & \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\
0 & 0 & \frac{v_l + v_r}{2}\cos\theta & \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\
0 & 0 & 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\
0 & 0 & 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_3 \\
0 & 0 & 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right)C_3
\end{bmatrix}$$

$$\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix}$$

Therefore,

Theorem 8.7.3 — Linear time-varying differential drive state-space model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v_l \\ v_r \end{bmatrix} = \begin{bmatrix} \text{x position} \\ \text{y position} \\ \text{heading} \\ \text{left velocity} \\ \text{right velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -vs & \frac{1}{2}c & \frac{1}{2}c \\ 0 & 0 & vc & \frac{1}{2}s & \frac{1}{2}s \\ 0 & 0 & 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\ 0 & 0 & 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ 0 & 0 & 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \tag{8.13}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \tag{8.14}$$

where $v = \frac{v_l + v_r}{2}$, $c = \cos\theta$, $s = \sin\theta$, $C_1 = -\frac{G_l^2 K_t}{K_v R r^2}$, $C_2 = \frac{G_l K_t}{Rr}$, $C_3 = -\frac{G_r^2 K_t}{K_v R r^2}$, and $C_4 = \frac{G_r K_t}{Rr}$. The constants C_1 through C_4 are from the derivation in section 12.6.

We can also use this in an extended Kalman filter as is since the measurement model ($\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$) is linear.

8.7.4 Improving model accuracy

Figures 8.4 and 8.5 demonstrate the tracking behavior of the linearized differential drive controller.

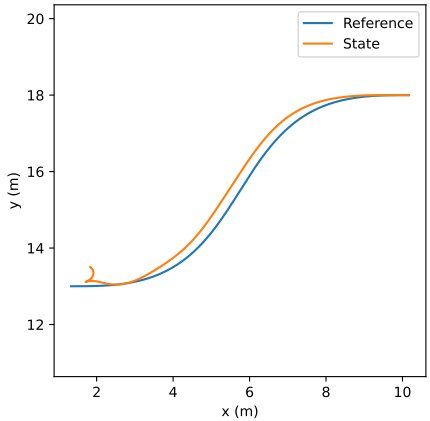


Figure 8.4: Linear time-varying differential drive controller x-y plot (first-order)

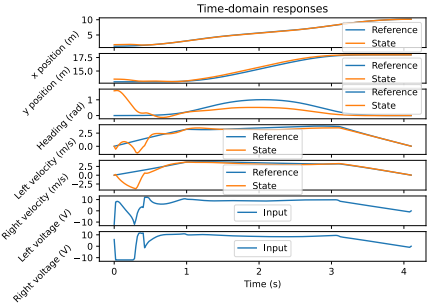


Figure 8.5: Linear time-varying differential drive controller response (first-order)

The linearized differential drive model doesn't track well because the first-order linearization of $\mathbf{A}$ doesn't capture the full heading dynamics, making the model update inaccurate. This linearization inaccuracy is evident in the Hessian matrix (second partial derivative with respect to the state vector) being nonzero.

$$\frac{\partial^2 f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}^2} = \begin{bmatrix} 0 & 0 & -\frac{v_l + v_r}{2}\cos\theta & 0 & 0 \\ 0 & 0 & -\frac{v_l + v_r}{2}\sin\theta & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The second-order Taylor series expansion of the model around $\mathbf{x}_0$ would be

$$f(\mathbf{x}, \mathbf{u}_0) \approx f(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}\frac{\partial^2 f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}^2}(\mathbf{x} - \mathbf{x}_0)^2$$

To include higher-order dynamics in the linearized differential drive model integration, we'll apply the Dormand-Prince integration method (RKDP) from theorem 7.9.3 to equation (8.12).

Figures 8.6 and 8.7 show a simulation using RKDP instead of the first-order model.

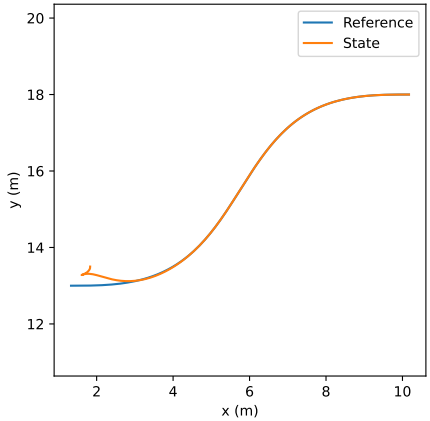


Figure 8.6: Linear time-varying differential drive controller (global reference frame formulation) x-y plot

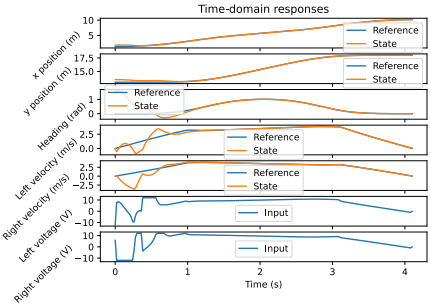


Figure 8.7: Linear time-varying differential drive controller (global reference frame formulation) response

8.7.5 Cross track error controller

Figures 8.6 and 8.7 show the tracking performance of the linearized differential drive controller for a given trajectory. The performance-effort trade-off can be tuned rather intuitively via the Q and R gains. However, if the x and y error cost are too high, the x and y components of the controller will fight each other, and it will take longer to converge to the path. This can be fixed by applying a clockwise rotation matrix to the global tracking error to transform it into the robot's coordinate frame.

$$
{}^{R}\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{G}\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix}
$$

where the the superscript R represents the robot's coordinate frame and the superscript G represents the global coordinate frame.

With this transformation, the x and y error cost in LQR penalize the error ahead of the robot and cross-track error respectively instead of global pose error. Since the cross-track error is always measured from the robot's coordinate frame, the model used to

compute the LQR should be linearized around $\theta = 0$ at all times.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -\frac{v_l+v_r}{2}\sin 0 & \frac{1}{2}\cos 0 & \frac{1}{2}\cos 0 \\ 0 & 0 & \frac{v_l+v_r}{2}\cos 0 & \frac{1}{2}\sin 0 & \frac{1}{2}\sin 0 \\ 0 & 0 & 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\ 0 & 0 & 0 & \left(\frac{1}{m}+\frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m}-\frac{r_b^2}{J}\right)C_3 \\ 0 & 0 & 0 & \left(\frac{1}{m}-\frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m}+\frac{r_b^2}{J}\right)C_3 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{v_l+v_r}{2} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\ 0 & 0 & 0 & \left(\frac{1}{m}+\frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m}-\frac{r_b^2}{J}\right)C_3 \\ 0 & 0 & 0 & \left(\frac{1}{m}-\frac{r_b^2}{J}\right)C_1 & \left(\frac{1}{m}+\frac{r_b^2}{J}\right)C_3 \end{bmatrix}$$

Theorem 8.7.4 — Linear time-varying differential drive controller. Let the differential drive dynamics be of the form $\dot{\mathbf{x}} = f(\mathbf{x}) + \mathbf{B}\mathbf{u}$ where

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v_l \\ v_r \end{bmatrix} = \begin{bmatrix} \text{x position} \\ \text{y position} \\ \text{heading} \\ \text{left velocity} \\ \text{right velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\mathbf{A} = \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\theta=0} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & v & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2r_b} & \frac{1}{2r_b} \\ 0 & 0 & 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ 0 & 0 & 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \tag{8.15}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \tag{8.16}$$

where $v = \frac{v_l + v_r}{2}$, $C_1 = -\frac{G_l^2 K_t}{K_v R r^2}$, $C_2 = \frac{G_l K_t}{R r}$, $C_3 = -\frac{G_r^2 K_t}{K_v R r^2}$, and $C_4 = \frac{G_r K_t}{R r}$. The constants C_1 through C_4 are from the derivation in section 12.6.

The linear time-varying differential drive controller is

$$\mathbf{u} = \mathbf{K} \left[\begin{array}{cc|c} \cos\theta & \sin\theta & \mathbf{0}_{2\times 3} \\ -\sin\theta & \cos\theta & \\ \hline \multicolumn{2}{c|}{\mathbf{0}_{3\times 2}} & \mathbf{I}_{3\times 3} \end{array}\right] (\mathbf{r} - \mathbf{x}) \tag{8.17}$$

At each timestep, the LQR controller gain $\mathbf{K}$ is computed for the $(\mathbf{A}, \mathbf{B})$ pair evaluated at the current state.

With the model in theorem 8.7.4, y is uncontrollable at $v = 0$ because the row corresponding to y becomes the zero vector. This means the state dynamics and inputs can no longer affect y. This is obvious given that nonholonomic drivetrains can't move sideways. Some DARE solvers throw errors in this case, but one can avoid it by linearizing

the model around a slightly nonzero velocity instead.

The controller in theorem 8.7.4 results in figures 8.8 and 8.9, which show slightly better tracking performance than the previous formulation.

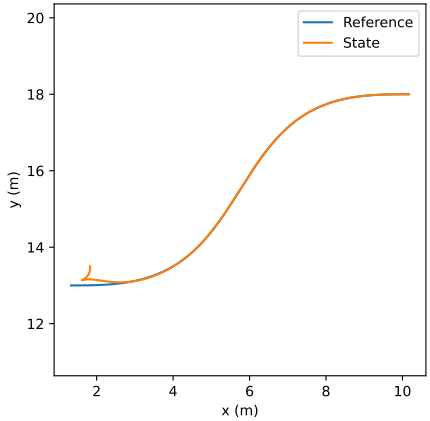


Figure 8.8: Linear time-varying differential drive controller x-y plot

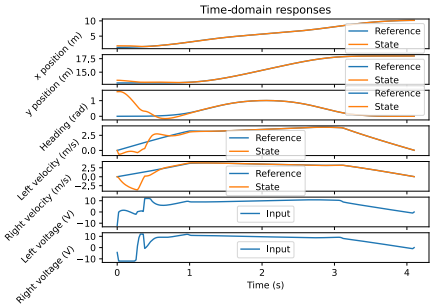


Figure 8.9: Linear time-varying differential drive controller response

8.7.6 Nonlinear observer design

Encoder position augmentation

Estimation of the global pose can be significantly improved if encoder position measurements are used instead of velocity measurements. By augmenting the plant with the line integral of each wheel's velocity over time, we can provide a mapping from model states to position measurements. We can augment the linear subspace of the model as follows.

Augment the matrix equation with position states x_l and x_r, which have the model equations $\dot{x}_l = v_l$ and $\dot{x}_r = v_r$. The matrix elements corresponding to v_l in the first equation and v_r in the second equation will be 1, and the others will be 0 since they don't appear, so $\dot{x}_l = 1v_l + 0v_r + 0x_l + 0x_r + 0V_l + 0V_r$ and $\dot{x}_r = 0v_l + 1v_r + 0x_l + 0x_r + 0V_l + 0V_r$. The existing rows will have zeroes inserted where x_l and x_r are multiplied in.

$$\begin{bmatrix} \dot{x}_l \\ \dot{x}_r \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_l \\ V_r \end{bmatrix}$$

This produces the following linear subspace over $\mathbf{x} = \begin{bmatrix} v_l & v_r & x_l & x_r \end{bmatrix}^\mathsf{T}$.

$$\mathbf{A} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 & 0 & 0 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{8.18}$$

$$\mathbf{B} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{8.19}$$

The measurement model for the complete nonlinear model is now $\mathbf{y} = \begin{bmatrix} \theta & x_l & x_r \end{bmatrix}^\mathsf{T}$ instead of $\mathbf{y} = \begin{bmatrix} \theta & v_l & v_r \end{bmatrix}^\mathsf{T}$.

U error estimation

As per subsection 6.7.2, we will now augment the model so u_{error} states are added to the control inputs.

The plant and observer augmentations should be performed before the model is discretized. After the controller gain is computed with the unaugmented discrete model, the controller may be augmented. Therefore, the plant and observer augmentations assume a continuous model and the controller augmentation assumes a discrete controller.

The three u_{error} states we'll be adding are $u_{error,l}$, $u_{error,r}$, and $u_{error,heading}$ for left voltage error, right voltage error, and heading error respectively. The left and right wheel positions are filtered encoder positions and are not adjusted for heading error. The turning angle computed from the left and right wheel positions is adjusted by the gyroscope heading. The heading u_{error} state is the heading error between what the wheel positions imply and the gyroscope measurement.

The full state is thus

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v_l \\ v_r \\ x_l \\ x_r \\ u_{error,l} \\ u_{error,r} \\ u_{error,heading} \end{bmatrix}$$

The complete nonlinear model is as follows. Let $v = \frac{v_l + v_r}{2}$. The three u_{error} states augment the linear subspace, so the nonlinear pose dynamics are the same.

$$\begin{bmatrix} \dot{x} \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v_r}{2r_b} - \frac{v_l}{2r_b} \end{bmatrix} \tag{8.20}$$

The left and right voltage error states should be mapped to the corresponding velocity states, so the system matrix should be augmented with $\mathbf{B}$.

The heading u_{error} is measuring counterclockwise encoder understeer relative to the gyroscope heading, so it should add to the left position and subtract from the right position for clockwise correction of encoder positions. That corresponds to the following input mapping vector.

$$\mathbf{B}_\theta = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

Now we'll augment the linear system matrix horizontally to accomodate the u_{error} states.

$$\begin{bmatrix} \dot{v}_l \\ v_r \\ x_l \\ x_r \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B}_\theta \end{bmatrix} \begin{bmatrix} v_l \\ v_r \\ x_l \\ x_r \\ u_{error,l} \\ u_{error,r} \\ u_{error,heading} \end{bmatrix} + \mathbf{Bu} \tag{8.21}$$

$\mathbf{A}$ and $\mathbf{B}$ are the linear subspace from equation (8.19).

The u_{error} states have no dynamics. The observer selects them to minimize the difference between the expected and actual measurements.

$$\dot{\begin{bmatrix} u_{error,l} \\ u_{error,r} \\ u_{error,heading} \end{bmatrix}} = \mathbf{0}_{3\times 1} \tag{8.22}$$

The controller is augmented as follows.

$$\mathbf{K}_{error} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & \mathbf{K}_{error} \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8.23}$$

This controller augmentation compensates for unmodeled dynamics like:

1. Understeer caused by wheel friction inherent in skid-steer robots
2. Battery voltage drop under load, which reduces the available control authority

R The process noise for the voltage error states should be how much the voltage can be expected to drop. The heading error state should be the encoder model uncertainty.

8.8 Linear time-varying unicycle controller

One can also create a linear time-varying controller with a cascaded control architecture, where the outer layer consumes a pose command and produces unicycle velocity commands, and the inner layer consumes unicycle velocity commands and produces wheel motor voltages.

The change in global pose for a unicycle is defined by the following three equations.

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned}$$

Here's the model as a vector function where $\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^\mathsf{T}$ and $\mathbf{u} = \begin{bmatrix} v & \omega \end{bmatrix}^\mathsf{T}$.

$$f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{8.24}$$

To create an LQR, we need to linearize this.

$$\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 0 & -v \sin\theta \\ 0 & 0 & v \cos\theta \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}$$

We're going to make a cross-track error controller, so we'll apply a clockwise rotation matrix to the global tracking error to transform it into the robot's coordinate frame. Since the cross-track error is always measured from the robot's coordinate frame, the model used to compute the LQR should be linearized around $\theta = 0$ at all times.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -v \sin 0 \\ 0 & 0 & v \cos 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \cos 0 & 0 \\ \sin 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Therefore,

Theorem 8.8.1 — Linear time-varying unicycle controller. Let the unicycle dynamics be $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos\theta & v\sin\theta & \omega \end{bmatrix}^\mathsf{T}$ where

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \text{x position} \\ \text{y position} \\ \text{heading} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \text{linear velocity} \\ \text{angular velocity} \end{bmatrix}$$

$$\mathbf{A} = \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\theta=0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{B} = \left.\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\theta=0} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

The linear time-varying unicycle controller is

$$\mathbf{u} = \mathbf{K} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} (\mathbf{r} - \mathbf{x}) \tag{8.25}$$

At each timestep, the LQR controller gain $\mathbf{K}$ is computed for the $(\mathbf{A}, \mathbf{B})$ pair evaluated at the current input.

With the model in theorem 8.8.1, y is uncontrollable at $v = 0$ because nonholonomic drivetrains are unable to move sideways. Some DARE solvers throw errors in this case, but one can avoid it by linearizing the model around a slightly nonzero velocity instead.

The controller in theorem 8.8.1 results in figures 8.10 and 8.11.

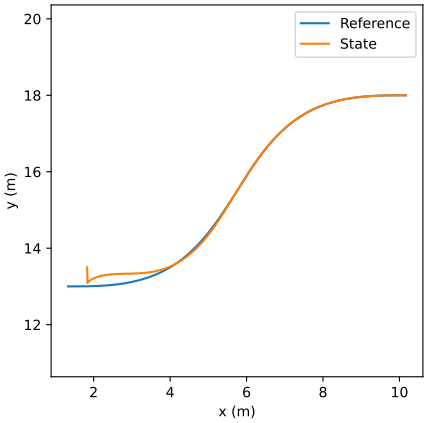


Figure 8.10: Linear time-varying unicycle controller x-y plot

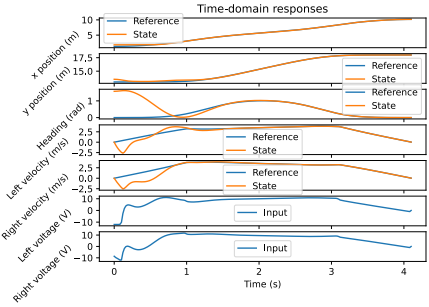


Figure 8.11: Linear time-varying unicycle controller response

8.9 Further reading

To learn more about nonlinear control, we recommend watching the underactuated robotics lectures by MIT and reading their lecture note [16].

"Underactuated Robotics"
Russ Tedrake
https://www.youtube.com/channel/UChfUOAhz7ynELF-s_1LPpWg/videos

The books *Nonlinear Dynamics and Chaos* by Steven Strogatz and *Applied Nonlinear Control* by Jean-Jacques Slotine are also good references.

II Estimation and localization

This page intentionally left blank

A sample of the Santa Cruz banana distribution [11]

9. Stochastic control theory

Stochastic control theory is a subfield of control theory that deals with the existence of uncertainty either in observations or in noise that drives the evolution of a system. We assign probability distributions to this random noise and aim to achieve a desired control task despite the presence of this uncertainty.

Stochastic optimal control is concerned with doing this with minimum cost defined by some cost functional, like we did with LQR earlier. First, we'll cover the basics of probability and how we represent linear stochastic systems in state-space representation. Then, we'll derive an optimal estimator using this knowledge, the Kalman filter, and demonstrate creative applications of the Kalman filter theory.

This will be a math-heavy introduction, so we recommend reading *Kalman and Bayesian Filters in Python* by Roger Labbe first [9].

9.1 Terminology

First, we should provide definitions for terms that have specific meanings in this field.

A causal system is one that uses only past information. A noncausal system also uses information from the future. A filter is a causal system that *filters* information through a probabilistic model to produce an estimate of a desired quantity that can't be measured directly. A smoother is a noncausal system, so it uses information from before and after the current state to produce a better estimate.

9.2 State observers

State observers are used to estimate states which cannot be measured directly. This can be due to noisy measurements or the state not being measurable (a hidden state). This information can be used for localization, which is the process of using external measurements to determine an agent's pose,[1] or orientation in the world.

One type of state estimator is LQE. "LQE" stands for "Linear-Quadratic Estimator". Similar to LQR, it places the estimator poles such that it minimizes the sum of squares of the estimation error. The Luenberger observer and Kalman filter are examples of these, where the latter chooses the pole locations optimally based on the model and measurement uncertainties.

Computer vision can also be used for localization. By extracting features from an image taken by the agent's camera, like a retroreflective target in FRC, and comparing them to known dimensions, one can determine where the agent's camera would have to be to see that image. This can be used to correct our state estimate in the same way we do with an encoder or gyroscope.

9.2.1 Luenberger observer

We'll introduce the Luenberger observer first to demonstrate the general form of a state estimator and some of their properties.

[1] An agent is a system-agnostic term for independent controlled actors like robots or aircraft.

Theorem 9.2.1 — Luenberger observer.

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \hat{\mathbf{y}}) \tag{9.1}$$
$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\mathbf{u} \tag{9.2}$$

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{9.3}$$
$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k \tag{9.4}$$

$\mathbf{A}$	system matrix	$\hat{\mathbf{x}}$	state estimate vector
$\mathbf{B}$	input matrix	$\mathbf{u}$	input vector
$\mathbf{C}$	output matrix	$\mathbf{y}$	output vector
$\mathbf{D}$	feedthrough matrix	$\hat{\mathbf{y}}$	output estimate vector
$\mathbf{L}$	estimator gain matrix		

Matrix	Rows × Columns	Matrix	Rows × Columns
$\mathbf{A}$	states × states	$\hat{\mathbf{x}}$	states × 1
$\mathbf{B}$	states × inputs	$\mathbf{u}$	inputs × 1
$\mathbf{C}$	outputs × states	$\mathbf{y}$	outputs × 1
$\mathbf{D}$	outputs × inputs	$\hat{\mathbf{y}}$	outputs × 1
$\mathbf{L}$	states × outputs		

Table 9.1: Luenberger observer matrix dimensions

Variables denoted with a hat are estimates of the corresponding variable. For example, $\hat{\mathbf{x}}$ is the estimate of the true state $\mathbf{x}$.

Notice that a Luenberger observer has an extra term in the state evolution equation. This term uses the difference between the estimated outputs and measured outputs to steer the estimated state toward the true state. $\mathbf{L}$ approaching $\mathbf{C}^+$ trusts the measurements more while $\mathbf{L}$ approaching $\mathbf{0}$ trusts the model more.

R Using an estimator forfeits the performance guarantees from earlier, but the

responses are still generally very good if the process and measurement noises are small enough. See John Doyle's paper *Guaranteed Margins for LQG Regulators* for a proof.

A Luenberger observer combines the prediction and update steps of an estimator. To run them separately, use the equations in theorem 9.2.2 instead.

Theorem 9.2.2 — Luenberger observer with separate predict/update.

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k \tag{9.5}$$

Update step

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{9.6}$$

$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_{k}^{-} \tag{9.7}$$

See appendix D.2.1 for a derivation.

Eigenvalues of closed-loop observer

The eigenvalues of the system matrix can be used to determine whether a state observer's estimate will converge to the true state.

Plugging equation (9.4) into equation (9.3) gives

$$\begin{aligned}
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - (\mathbf{C}\hat{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k)) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k - \mathbf{D}\mathbf{u}_k)
\end{aligned}$$

Plugging in $\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$ gives

$$\begin{aligned}
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}((\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k) - \mathbf{C}\hat{\mathbf{x}}_k - \mathbf{D}\mathbf{u}_k) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k - \mathbf{C}\hat{\mathbf{x}}_k - \mathbf{D}\mathbf{u}_k) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{C}\mathbf{x}_k - \mathbf{C}\hat{\mathbf{x}}_k) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}(\mathbf{x}_k - \hat{\mathbf{x}}_k)
\end{aligned}$$

Let $\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$ be the error in the estimate $\hat{\mathbf{x}}_k$.

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}\mathbf{e}_k$$

Subtracting this from $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ gives

$$\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - (\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}\mathbf{e}_k)$$

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - (\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}\mathbf{e}_k) \\
\mathbf{e}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - \mathbf{A}\hat{\mathbf{x}}_k - \mathbf{B}\mathbf{u}_k - \mathbf{L}\mathbf{C}\mathbf{e}_k \\
\mathbf{e}_{k+1} &= \mathbf{A}\mathbf{x}_k - \mathbf{A}\hat{\mathbf{x}}_k - \mathbf{L}\mathbf{C}\mathbf{e}_k \\
\mathbf{e}_{k+1} &= \mathbf{A}(\mathbf{x}_k - \hat{\mathbf{x}}_k) - \mathbf{L}\mathbf{C}\mathbf{e}_k \\
\mathbf{e}_{k+1} &= \mathbf{A}\mathbf{e}_k - \mathbf{L}\mathbf{C}\mathbf{e}_k \\
\mathbf{e}_{k+1} &= (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{e}_k
\end{aligned} \tag{9.8}$$

For equation (9.8) to have a bounded output, the eigenvalues of $\mathbf{A} - \mathbf{LC}$ must be within the unit circle. These eigenvalues represent how fast the estimator converges to the true state of the given model. A fast estimator converges quickly while a slow estimator avoids amplifying noise in the measurements used to produce a state estimate.

The effect of noise can be seen if it is modeled stochastically as

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}((\mathbf{y}_k + \nu_k) - \hat{\mathbf{y}}_k)$$

where ν_k is the measurement noise. Rearranging this equation yields

$$\begin{aligned}
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k + \nu_k) \\
\hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) + \mathbf{L}\nu_k
\end{aligned}$$

As $\mathbf{L}$ increases, the measurement noise is amplified.

9.2.2 Separation principle

The separation principle for linear stochastic systems states that the optimal controller and optimal observer for the stochastic system can be designed independently, and the combination of a stable controller and a stable observer is itself stable.

This means that designing the optimal feedback controller for the stochastic system can be decomposed into designing the optimal state observer, then feeding that into the optimal controller for the deterministic system.

Consider the following state-space model.

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\
\mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k
\end{aligned}$$

We'll use the following controller for state feedback.

$$\mathbf{u}_k = -\mathbf{K}\hat{\mathbf{x}}_k$$

With the estimation error defined as $\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$, we get the observer dynamics derived in equation (9.8).

$$\mathbf{e}_{k+1} = (\mathbf{A} - \mathbf{LC})\mathbf{e}_k$$

Also, after rearranging the error equation to be $\hat{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{e}_k$, the controller can be rewritten as

$$\mathbf{u}_k = -\mathbf{K}(\mathbf{x}_k - \mathbf{e}_k)$$

Substitute this into the model.

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k - \mathbf{BK}(\mathbf{x}_k - \mathbf{e}_k) \\ \mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{BK})\mathbf{x}_k + \mathbf{BK}\mathbf{e}_k\end{aligned}$$

Now, we can write the closed-loop dynamics as

$$\begin{bmatrix}\mathbf{x}_{k+1} \\ \mathbf{e}_{k+1}\end{bmatrix} = \begin{bmatrix}\mathbf{A} - \mathbf{BK} & \mathbf{BK} \\ \mathbf{0} & \mathbf{A} - \mathbf{LC}\end{bmatrix} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{e}_k\end{bmatrix}$$

Since the closed-loop system matrix is triangular, the eigenvalues are those of $\mathbf{A} - \mathbf{BK}$ and $\mathbf{A} - \mathbf{LC}$. Therefore, the stability of the feedback controller and observer are independent.

9.3 Introduction to probability

Now we'll begin establishing probability concepts we need to describe and manipulate stochastic systems.

9.3.1 Random variables

A random variable is a variable whose values are the outcomes of a random phenomenon, like dice rolls or noisy process measurements. As such, a random variable is defined as a function that maps the outcomes of an unpredictable process to numerical quantities. A particular output of this function is called a sample. The sample space is the set of possible values taken by the random variable.

A probability density function (PDF) is a function of the random variable whose value at a given sample (measured value) in the sample space (the range of possible measured values) is the probability of that sample occurrring. The area under the function over a range gives the probability that the sample falls within that range. Let x be a random variable, and let $p(x)$ denote the probability density function of x. The probability that

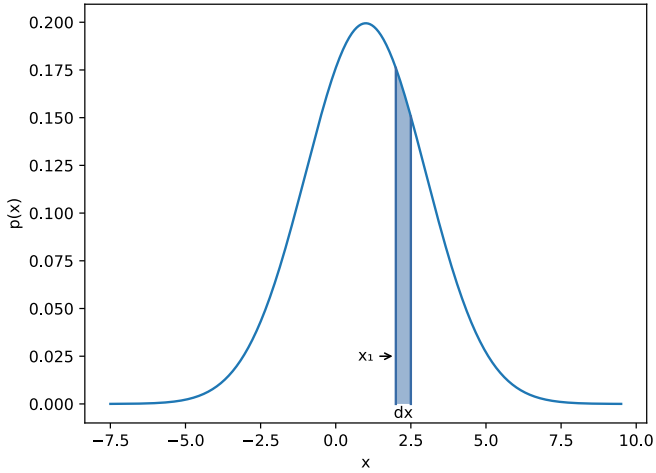


Figure 9.1: Probability density function

the value of x will be in the interval $x \in [x_1, x_1 + dx]$ is $p(x_1)\,dx$. In other words, the probability is the area under the PDF within the region $[x_1, x_1 + dx]$ (see figure 9.1).

A probability of zero means that the sample will not occur and a probability of one means that the sample will always occur. Probability density functions require that no probabilities are negative and that the sum of all probabilities is 1. If the probabilities sum to 1, that means one of those outcomes *must* happen. In other words,

$$p(x) \geq 0, \int_{-\infty}^{\infty} p(x)\,dx = 1$$

or given that the probability of a given sample is greater than or equal to zero, the sum of probabilities for all possible input values is equal to one.

9.3.2 Expected value

Expected value or expectation is a weighted average of the values the PDF can produce where the weight for each value is the probability of that value occurring. This can be written mathematically as

$$\overline{x} = E[x] = \int_{-\infty}^{\infty} x\,p(x)\,dx$$

The expectation can be applied to random functions as well as random variables.

$$E[f(x)] = \int_{-\infty}^{\infty} f(x)\, p(x)\, dx$$

The mean of a random variable is denoted by an overbar (e.g., $\overline{x}$) pronounced x-bar. The expectation of the difference between a random variable and its mean $x - \overline{x}$ converges to zero. In other words, the expectation of a random variable is its mean. Here's a proof.

$$\begin{aligned}
E[x - \overline{x}] &= \int_{-\infty}^{\infty} (x - \overline{x})\, p(x)\, dx \\
E[x - \overline{x}] &= \int_{-\infty}^{\infty} x\, p(x)\, dx - \int_{-\infty}^{\infty} \overline{x}\, p(x)\, dx \\
E[x - \overline{x}] &= \int_{-\infty}^{\infty} x\, p(x)\, dx - \overline{x} \int_{-\infty}^{\infty} p(x)\, dx \\
E[x - \overline{x}] &= \overline{x} - \overline{x} \cdot 1 \\
E[x - \overline{x}] &= 0
\end{aligned}$$

9.3.3 Variance

Informally, variance is a measure of how far the outcome of a random variable deviates from its mean. Later, we will use variance to quantify how confident we are in the estimate of a random variable; we can't know the true value of that variable without randomness, but we can give a bound on its randomness.

$$\operatorname{var}(x) = \sigma^2 = E[(x - \overline{x})^2] = \int_{-\infty}^{\infty} (x - \overline{x})^2\, p(x)\, dx$$

The standard deviation is the square root of the variance.

$$\operatorname{std}[x] = \sigma = \sqrt{\operatorname{var}(x)}$$

9.3.4 Joint probability density functions

Probability density functions can also include more than one variable. Let x and y be random variables. The joint probability density function $p(x, y)$ defines the probability $p(x, y)\, dx\, dy$, so that x and y are in the intervals $x \in [x, x + dx], y \in [y, y + dy]$. In other words, the probability is the volume under a region of the PDF manifold (see figure 9.2 for an example of a joint PDF).

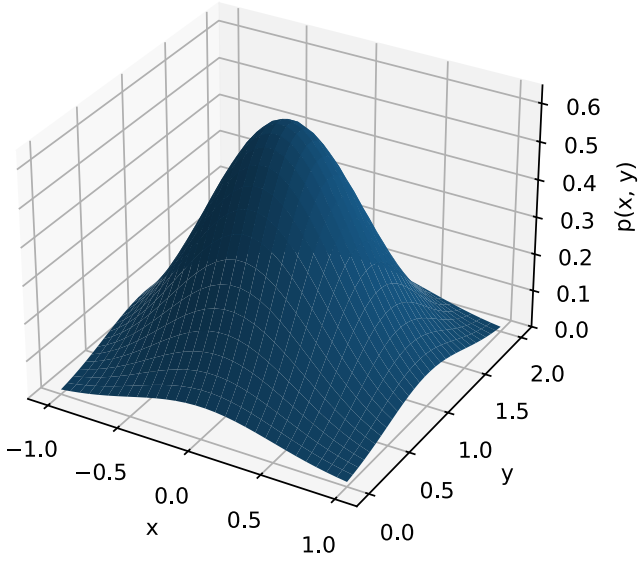


Figure 9.2: Joint probability density function

Joint probability density functions also require that no probabilities are negative and that the sum of all probabilities is 1.

$$p(x, y) \geq 0, \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x, y)\, dx\, dy = 1$$

The expected values for joint PDFs are as follows.

$$E[x] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xp(x, y)\, dx\, dy$$
$$E[y] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} yp(x, y)\, dx\, dy$$
$$E[f(x, y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)p(x, y)\, dx\, dy$$

The variance of a joint PDF measures how a variable correlates with itself (we'll cover variances with respect to other variables shortly).

$$\text{var}(x) = \Sigma_{xx} = E[(x - \overline{x})^2] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \overline{x})^2\, p(x, y)\, dx\, dy$$
$$\text{var}(y) = \Sigma_{yy} = E[(y - \overline{y})^2] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (y - \overline{y})^2\, p(x, y)\, dx\, dy$$

9.3.5 Covariance

A covariance is a measurement of how a variable correlates with another. If they vary in the same direction, the covariance increases. If they vary in opposite directions, the covariance decreases.

$$\text{cov}(x, y) = \Sigma_{xy} = E[(x - \overline{x})(y - \overline{y})] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \overline{x})(y - \overline{y})\, p(x, y)\, dx\, dy$$

9.3.6 Correlation

Two random variables are correlated if the result of one random variable affects the result of another. Correlation is defined as

$$\rho(x, y) = \frac{\Sigma_{xy}}{\sqrt{\Sigma_{xx}\Sigma_{yy}}}, |\rho(x, y)| \leq 1$$

So two variable's correlation is defined as their covariance over the geometric mean of their variances. Uncorrelated sources have a covariance of zero.

9.3.7 Independence

Two random variables are independent if the following relation is true.

$$p(x, y) = p(x)\, p(y)$$

This means that the values of x do not correlate with the values of y. That is, the outcome of one random variable does not affect another's outcome. If we assume independence,

$$\begin{aligned}
E[xy] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xy\, p(x, y)\, dx\, dy \\
E[xy] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xy\, p(x)\, p(y)\, dx\, dy \\
E[xy] &= \int_{-\infty}^{\infty} x\, p(x)\, dx \int_{-\infty}^{\infty} y\, p(y)\, dy \\
E[xy] &= E[x]E[y] \\
E[xy] &= \overline{x}\,\overline{y}
\end{aligned}$$

$$\begin{aligned}
\operatorname{cov}(x, y) &= E[(x - \overline{x})(y - \overline{y})] \\
\operatorname{cov}(x, y) &= E[(x - \overline{x})]E[(y - \overline{y})] \\
\operatorname{cov}(x, y) &= 0 \cdot 0
\end{aligned}$$

Therefore, the covariance Σ_{xy} is zero, as expected. Furthermore, $\rho(x, y) = 0$, which means they are uncorrelated.

9.3.8 Marginal probability density functions

Given two random variables x and y whose joint distribution is known, the marginal PDF $p(x)$ expresses the probability of x averaged over information about y. In other words, it's the PDF of x when y is unknown. This is calculated by integrating the joint PDF over y.

$$p(x) = \int_{-\infty}^{\infty} p(x, y)\, dy$$

9.3.9 Conditional probability density functions

Let us assume that we know the joint PDF $p(x, y)$ and the exact value for y. The conditional PDF gives the probability of x in the interval $[x, x + dx]$ for the given value y. The probability of x given y is denoted by $p(x|y)$.

If $p(x, y)$ is known, then we also know $p(x, y = y^*)$. However, note that the latter is not the conditional density $p(x|y^*)$, instead

$$C(y^*) = \int_{-\infty}^{\infty} p(x, y = y^*)\, dx$$
$$p(x|y^*) = \frac{1}{C(y^*)} p(x, y = y^*)$$

The scale factor $\frac{1}{C(y^*)}$ is used to scale the area under the PDF to 1.

9.3.10 Bayes's rule

Bayes's rule is used to determine the probability of an event based on prior knowledge of conditions related to the event.

$$p(x, y) = p(x|y)\, p(y) = p(y|x)\, p(x)$$

If x and y are independent, then $p(x|y) = p(x)$, $p(y|x) = p(y)$, and $p(x, y) = p(x)\, p(y)$.

9.3.11 Conditional expectation

The concept of expectation can also be applied to conditional PDFs. This allows us to determine what the mean of a variable is given prior knowledge of other variables.

$$E[x|y] = \int_{-\infty}^{\infty} x\, p(x|y)\, dx = f(y), E[x|y] \neq E[x]$$
$$E[y|x] = \int_{-\infty}^{\infty} y\, p(y|x)\, dy = f(x), E[y|x] \neq E[y]$$

9.3.12 Conditional variances

$$\text{var}(x|y) = E[(x - E[x|y])^2|y]$$
$$\text{var}(x|y) = \int_{-\infty}^{\infty} (x - E[x|y])^2\, p(x|y)\, dx$$

9.3.13 Random vectors

Now we will extend the probability concepts discussed so far to vectors where each element has a PDF.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

The elements of $\mathbf{x}$ are scalar variables jointly distributed with a joint density $p(x_1, \ldots, x_n)$. The expectation is

$$E[\mathbf{x}] = \overline{\mathbf{x}} = \int_{-\infty}^{\infty} \mathbf{x}\, p(\mathbf{x})\, d\mathbf{x}$$

$$E[\mathbf{x}] = \begin{bmatrix} E[x_1] \\ \vdots \\ E[x_n] \end{bmatrix}$$

$$E[x_i] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} x_i\, p(x_1, \ldots, x_n)\, dx_1 \ldots dx_n$$

$$E[f(\mathbf{x})] = \int_{-\infty}^{\infty} f(\mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}$$

9.3.14 Covariance matrix

The covariance matrix for a random vector $\mathbf{x} \in \mathbb{R}^n$ is

$$\boldsymbol{\Sigma} = \operatorname{cov}(\mathbf{x}, \mathbf{x}) = E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathsf{T}}]$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \operatorname{cov}(x_1, x_1) & \ldots & \operatorname{cov}(x_1, x_n) \\ \vdots & \ddots & \vdots \\ \operatorname{cov}(x_n, x_1) & \ldots & \operatorname{cov}(x_n, x_n) \end{bmatrix}$$

This $n \times n$ matrix is symmetric and positive semidefinite. A positive semidefinite matrix satisfies the relation that for any $\mathbf{v} \in \mathbb{R}^n$ for which $\mathbf{v} \neq 0$, $\mathbf{v}^{\mathsf{T}} \boldsymbol{\Sigma} \mathbf{v} \geq 0$. In other words, the eigenvalues of $\boldsymbol{\Sigma}$ are all greater than or equal to zero.

9.3.15 Relations for independent random vectors

First, independent vectors imply linearity from $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})\, p(\mathbf{y})$.

$$E[\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}] = \mathbf{A}E[\mathbf{x}] + \mathbf{B}E[\mathbf{y}]$$
$$E[\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}] = \mathbf{A}\overline{\mathbf{x}} + \mathbf{B}\overline{\mathbf{y}}$$

Second, independent vectors being uncorrelated means their covariance is zero.

$$\boldsymbol{\Sigma}_{\mathbf{xy}} = \operatorname{cov}(\mathbf{x}, \mathbf{y})$$
$$\boldsymbol{\Sigma}_{\mathbf{xy}} = E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{y} - \overline{\mathbf{y}})^{\mathsf{T}}]$$
$$\boldsymbol{\Sigma}_{\mathbf{xy}} = E[\mathbf{x}\mathbf{y}^{\mathsf{T}}] - E[\mathbf{x}\overline{\mathbf{y}}^{\mathsf{T}}] - E[\overline{\mathbf{x}}\mathbf{y}^{\mathsf{T}}] + E[\overline{\mathbf{x}}\,\overline{\mathbf{y}}^{\mathsf{T}}]$$

$$\begin{aligned}\boldsymbol{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^\mathsf{T}] - E[\mathbf{x}]\overline{\mathbf{y}}^\mathsf{T} - \overline{\mathbf{x}}E[\mathbf{y}^\mathsf{T}] + \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} \\ \boldsymbol{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^\mathsf{T}] - \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} - \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} + \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} \\ \boldsymbol{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^\mathsf{T}] - \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T}\end{aligned} \tag{9.9}$$

Now, compute $E[\mathbf{x}\mathbf{y}^\mathsf{T}]$.

$$E[\mathbf{x}\mathbf{y}^\mathsf{T}] = \int_X \int_Y \mathbf{x}\mathbf{y}^\mathsf{T}\, p(\mathbf{x})\, p(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y}^\mathsf{T}$$

Factor out constants from the inner integral. This includes variables which are held constant for each inner integral evaluation.

$$E[\mathbf{x}\mathbf{y}^\mathsf{T}] = \int_X p(\mathbf{x})\, \mathbf{x}\, d\mathbf{x} \int_Y p(\mathbf{y})\, \mathbf{y}^\mathsf{T}\, d\mathbf{y}^\mathsf{T}$$

Each of these integrals is just the expected value of their respective integration variable.

$$E[\mathbf{x}\mathbf{y}^\mathsf{T}] = \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} \tag{9.10}$$

Substitute equation (9.10) into equation (9.9).

$$\begin{aligned}\boldsymbol{\Sigma}_{\mathbf{xy}} &= (\overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T}) - \overline{\mathbf{x}}\overline{\mathbf{y}}^\mathsf{T} \\ \boldsymbol{\Sigma}_{\mathbf{xy}} &= 0\end{aligned}$$

Using these results, we can compute the covariance of $\mathbf{z} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}$.

$$\begin{aligned}\Sigma_z &= \operatorname{cov}(\mathbf{z}, \mathbf{z}) \\ \Sigma_z &= E[(\mathbf{z} - \overline{\mathbf{z}})(\mathbf{z} - \overline{\mathbf{z}})^\mathsf{T}] \\ \Sigma_z &= E[(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \mathbf{B}\overline{\mathbf{y}})(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \mathbf{B}\overline{\mathbf{y}})^\mathsf{T}] \\ \Sigma_z &= E[(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))^\mathsf{T}] \\ \Sigma_z &= E[(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))((\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}\mathbf{A}^\mathsf{T} + (\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}\mathbf{B}^\mathsf{T})] \\ \Sigma_z &= E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}\mathbf{A}^\mathsf{T} + \mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}\mathbf{B}^\mathsf{T} \\ &\qquad + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}\mathbf{A}^\mathsf{T} + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}\mathbf{B}^\mathsf{T}]\end{aligned}$$

Since $\mathbf{x}$ and $\mathbf{y}$ are independent, $\Sigma_{xy} = 0$ and the cross terms cancel out.

$$\Sigma_z = E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}\mathbf{A}^\mathsf{T} + 0 + 0 + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}\mathbf{B}^\mathsf{T}]$$

$$\Sigma_z = E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}\mathbf{A}^\mathsf{T}] + E[\mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}\mathbf{B}^\mathsf{T}]$$
$$\Sigma_z = \mathbf{A}E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^\mathsf{T}]\mathbf{A}^\mathsf{T} + \mathbf{B}E[(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^\mathsf{T}]\mathbf{B}^\mathsf{T}$$

Recall that $\Sigma_x = \mathrm{cov}(\mathbf{x}, \mathbf{x})$ and $\Sigma_y = \mathrm{cov}(\mathbf{y}, \mathbf{y})$.

$$\Sigma_z = \mathbf{A}\Sigma_x\mathbf{A}^\mathsf{T} + \mathbf{B}\Sigma_y\mathbf{B}^\mathsf{T}$$

9.3.16 Gaussian random variables

A Gaussian random variable has the following properties:

$$E[x] = \overline{x}$$
$$\mathrm{var}(x) = \sigma^2$$
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\overline{x})^2}{2\sigma^2}}$$

While we could use any random variable to represent a random process, we use the Gaussian random variable a lot in probability theory due to the central limit theorem.

Definition 9.3.1 — Central limit theorem. When independent random variables are added, their properly normalized sum tends toward a normal distribution (a Gaussian distribution or "bell curve").

This is the case even if the original variables themselves are not normally distributed. The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.

For example, suppose that a sample is obtained containing a large number of independent observations, and that the arithmetic mean of the observed values is computed. The central limit theorem says that the computed values of the mean will tend toward being distributed according to a normal distribution.

"But what is the Central Limit Theorem?" (31 minutes)
3Blue1Brown
https://youtu.be/zeJD6dqJ5lo

"A pretty reason why Gaussian + Gaussian = Gaussian" (13 minutes)
3Blue1Brown
https://youtu.be/d_qvLDhkg00

9.4 Linear stochastic systems

Given the following stochastic system

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k\end{aligned}$$

where $\mathbf{w}_k$ is the process noise and $\mathbf{v}_k$ is the measurement noise,

$$\begin{aligned}E[\mathbf{w}_k] &= 0 \\ E[\mathbf{w}_k\mathbf{w}_k^\mathsf{T}] &= \mathbf{Q}_k \\ E[\mathbf{v}_k] &= 0 \\ E[\mathbf{v}_k\mathbf{v}_k^\mathsf{T}] &= \mathbf{R}_k\end{aligned}$$

where $\mathbf{Q}_k$ is the process noise covariance matrix and $\mathbf{R}_k$ is the measurement noise covariance matrix. We assume the noise samples are independent, so $E[\mathbf{w}_k\mathbf{w}_j^\mathsf{T}] = 0$ and $E[\mathbf{v}_k\mathbf{v}_j^\mathsf{T}] = 0$ where $k \neq j$. Furthermore, process noise samples are independent from measurement noise samples.

We'll compute the expectation of these equations and their covariance matrices, which we'll use later for deriving the Kalman filter.

9.4.1 State vector expectation evolution

First, we will compute how the expectation of the system state evolves.

$$\begin{aligned}E[\mathbf{x}_{k+1}] &= E[\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= E[\mathbf{A}\mathbf{x}_k] + E[\mathbf{B}\mathbf{u}_k] + E[\mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= \mathbf{A}E[\mathbf{x}_k] + \mathbf{B}E[\mathbf{u}_k] + E[\mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= \mathbf{A}E[\mathbf{x}_k] + \mathbf{B}\mathbf{u}_k + 0\end{aligned}$$

$$\overline{\mathbf{x}}_{k+1} = \mathbf{A}\overline{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k$$

9.4.2 Error covariance matrix evolution

Now, we will use this to compute how the error covariance matrix $\mathbf{P}$ evolves.

$$\begin{aligned}\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k - (\mathbf{A}\overline{\mathbf{x}}_k - \mathbf{B}\mathbf{u}_k)\\ \mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1} &= \mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{w}_k\end{aligned}$$

$$E[(\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1})(\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1})^\mathsf{T}] = E[(\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{w}_k)(\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{w}_k)^\mathsf{T}]$$

$$\begin{aligned}\mathbf{P}_{k+1} &= E[(\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{w}_k)(\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{w}_k)^\mathsf{T}]\\ \mathbf{P}_{k+1} &= E[(\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k)(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}\mathbf{A}^\mathsf{T}] + E[\mathbf{A}(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{w}_k^\mathsf{T}]\\ &\quad + E[\mathbf{w}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}\mathbf{A}^\mathsf{T}] + E[\mathbf{w}_k\mathbf{w}_k^\mathsf{T}]\\ \mathbf{P}_{k+1} &= \mathbf{A}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{A}^\mathsf{T} + \mathbf{A}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{w}_k^\mathsf{T}]\\ &\quad + E[\mathbf{w}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{A}^\mathsf{T} + E[\mathbf{w}_k\mathbf{w}_k^\mathsf{T}]\\ \mathbf{P}_{k+1} &= \mathbf{A}\mathbf{P}_k\mathbf{A}^\mathsf{T} + \mathbf{A}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{w}_k^\mathsf{T}] + E[\mathbf{w}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{A}^\mathsf{T} + \mathbf{Q}_k\end{aligned}$$

Since the error and noise are independent, the cross terms are zero.

$$\begin{aligned}\mathbf{P}_{k+1} &= \mathbf{A}\mathbf{P}_k\mathbf{A}^\mathsf{T} + \mathbf{A}\underbrace{E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{w}_k^\mathsf{T}]}_{\mathbf{0}} + \underbrace{E[\mathbf{w}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]}_{\mathbf{0}}\mathbf{A}^\mathsf{T} + \mathbf{Q}_k\\ \mathbf{P}_{k+1} &= \mathbf{A}\mathbf{P}_k\mathbf{A}^\mathsf{T} + \mathbf{Q}_k\end{aligned}$$

9.4.3 Measurement vector expectation

Next, we will compute the expectation of the output $\mathbf{y}$.

$$\begin{aligned}E[\mathbf{y}_k] &= E[\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k]\\ E[\mathbf{y}_k] &= \mathbf{C}E[\mathbf{x}_k] + \mathbf{D}\mathbf{u}_k + 0\\ \overline{\mathbf{y}}_k &= \mathbf{C}\overline{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k\end{aligned}$$

9.4.4 Measurement covariance matrix

Now, we will use this to compute how the measurement covariance matrix $\mathbf{S}$ evolves.

$$\begin{aligned}\mathbf{y}_k - \overline{\mathbf{y}}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k - (\mathbf{C}\overline{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k)\\ \mathbf{y}_k - \overline{\mathbf{y}}_k &= \mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k\end{aligned}$$

$$E[(\mathbf{y}_k - \overline{\mathbf{y}}_k)(\mathbf{y}_k - \overline{\mathbf{y}}_k)^\mathsf{T}] = E[(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k)(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k)^\mathsf{T}]$$

$$\begin{aligned}
\mathbf{S}_k &= E[(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k)(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k)^\mathsf{T}] \\
\mathbf{S}_k &= E[(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{v}_k)((\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}\mathbf{C}^\mathsf{T} + \mathbf{v}_k^\mathsf{T})] \\
\mathbf{S}_k &= E[(\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k)(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}\mathbf{C}^\mathsf{T}] + E[\mathbf{C}(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{v}_k^\mathsf{T}] \\
&\quad + E[\mathbf{v}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}\mathbf{C}^\mathsf{T}] + E[\mathbf{v}_k\mathbf{v}_k^\mathsf{T}] \\
\mathbf{S}_k &= \mathbf{C}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{C}^\mathsf{T} + \mathbf{C}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{v}_k^\mathsf{T}] \\
&\quad + E[\mathbf{v}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{C}^\mathsf{T} + E[\mathbf{v}_k\mathbf{v}_k^\mathsf{T}] \\
\mathbf{S}_k &= \mathbf{C}\mathbf{P}_k\mathbf{C}^\mathsf{T} + \mathbf{C}E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{v}_k^\mathsf{T}] + E[\mathbf{v}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]\mathbf{C}^\mathsf{T} + \mathbf{R}_k
\end{aligned}$$

Since the error and noise are independent, the cross terms are zero.

$$\begin{aligned}
\mathbf{S}_k &= \mathbf{C}\mathbf{P}_k\mathbf{C}^\mathsf{T} + \mathbf{C}\underbrace{E[(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{v}_k^\mathsf{T}]}_{\mathbf{0}} + \underbrace{E[\mathbf{v}_k(\mathbf{x}_k - \overline{\mathbf{x}}_k)^\mathsf{T}]}_{\mathbf{0}}\mathbf{C}^\mathsf{T} + \mathbf{R}_k \\
\mathbf{S}_k &= \mathbf{C}\mathbf{P}_k\mathbf{C}^\mathsf{T} + \mathbf{R}_k
\end{aligned}$$

9.5 Two-sensor problem

9.5.1 Two noisy (independent) observations

Variable z_1 is the noisy measurement of the variable x. The value of the noisy measurement is the random variable defined by the probability density function $p(z_1|x)$.

Variable z_2 is the noisy measurement of the same variable x and independent of z_1. If we know the property of our sensor (measurement noise), then the probability density function of the measurement is $p(z_2|x)$.

We are interested in the probability density function of x given the measurements z_1 and z_2.

$$p(x|z_1, z_2) = \frac{p(x, z_1, z_2)}{p(z_1, z_2)} = \frac{p(z_1, z_2|x)p(x)}{p(z_1, z_2)} = \frac{p(z_1|x)p(z_2|x)p(x)}{p(z_1, z_2)}$$

where $p(z_1, z_2) = \int_X p(z_1|x)p(z_2|x)\,dx$, but z_1 and z_2 are given and we can write

$$p(x|z_1, z_2) = \frac{1}{C}p(z_1|x)p(z_2|x)p(x) \quad \text{or} \quad p(x|z_1, z_2) \sim p(z_1|x)p(z_2|x)p(x)$$

where C is a normalizing constant providing that $\int_X p(x|z_1, z_2)\,dx = 1$. The probability density $p(x|z_1, z_2)$ summarizes our complete knowledge about x.

9.5.2 Single noisy observations

Variable z_1 is the noisy measurement of the variable x. The value of the noisy measurement is the random variable defined by the probability density function $p(z_1|x)$.

The probability density function of x given the measurement z_1 is

$$p(x|z_1) = \frac{p(x, z_1)}{p(z_1)} = \frac{p(z_1|x)p(x)}{p(z_1)}$$

where $p(z_1) = \int_X p(z_1|x)\, dx$, but z_1 is given so we can write

$$p(x|z_1) = \frac{1}{C} p(z_1|x)p(x) \quad \text{or} \quad p(x|z_1) \sim p(z_1|x)p(x)$$

The probability density $p(x|z_1)$ summarizes our complete knowledge about x.

R In both the single and double measurement cases, the estimation of the variable x is a data/information fusion problem. In the single measurement case, we combine the *prior probability* with the probability resulting from the measurement.

9.5.3 Single noisy observations: a Gaussian case

$$p(z_1|x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{z_1 - x}{\sigma}\right)^2} \quad \text{and} \quad p(x) = \frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x - x_0}{\sigma_0}\right)^2}$$

z_1, x_0, σ^2, and σ_0^2 are given.

$$p(x|z_1) = \frac{p(x, z_1)}{p(z_1)}$$

$$p(x|z_1) = \frac{p(z_1|x)p(x)}{p(z_1)}$$

$$p(x|z_1) = \frac{1}{C} p(z_1|x)p(x)$$

$$p(x|z_1) = \frac{1}{C} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{z_1 - x}{\sigma}\right)^2} \frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x - x_0}{\sigma_0}\right)^2}$$

Absorb the leading coefficients of the two probability distributions into a new constant C_1.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{z_1 - x}{\sigma}\right)^2} e^{-\frac{1}{2}\left(\frac{x - x_0}{\sigma_0}\right)^2}$$

Combine the exponents.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{(z_1-x)^2}{\sigma^2} + \frac{(x-x_0)^2}{\sigma_0^2}\right)}$$

Expand the exponent into separate terms.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{z_1^2}{\sigma^2} - \frac{2z_1x}{\sigma^2} + \frac{x^2}{\sigma^2} + \frac{x^2}{\sigma_0^2} - \frac{2xx_0}{\sigma_0^2} + \frac{x_0^2}{\sigma_0^2}\right)}$$

Multiply in $\frac{\sigma^2}{\sigma^2}$ or $\frac{\sigma_0^2}{\sigma_0^2}$ as appropriate to give all terms a denominator of $\sigma^2\sigma_0^2$.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{\sigma_0^2 z_1^2}{\sigma^2\sigma_0^2} - 2\frac{\sigma_0^2 z_1}{\sigma^2\sigma_0^2}x + \frac{\sigma_0^2}{\sigma^2\sigma_0^2}x^2 + \frac{\sigma^2}{\sigma^2\sigma_0^2}x^2 - 2\frac{\sigma^2 x_0}{\sigma^2\sigma_0^2}x + \frac{\sigma^2 x_0^2}{\sigma^2\sigma_0^2}\right)}$$

Reorder terms in the exponent.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{\sigma_0^2}{\sigma^2\sigma_0^2}x^2 + \frac{\sigma^2}{\sigma^2\sigma_0^2}x^2 - 2\frac{\sigma_0^2 z_1}{\sigma^2\sigma_0^2}x - 2\frac{\sigma^2 x_0}{\sigma^2\sigma_0^2}x + \frac{\sigma_0^2 z_1^2}{\sigma^2\sigma_0^2} + \frac{\sigma^2 x_0^2}{\sigma^2\sigma_0^2}\right)}$$

Combine like terms.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\left(\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}x^2 - 2\frac{\sigma_0^2 z_1 + \sigma^2 x_0}{\sigma^2\sigma_0^2}x + \frac{\sigma_0^2 z_1^2 + \sigma^2 x_0^2}{\sigma^2\sigma_0^2}\right)}$$

Factor out $\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}$.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\left(x^2 - 2\frac{\sigma_0^2 z_1 + \sigma^2 x_0}{\sigma_0^2+\sigma^2}x + \frac{\sigma_0^2 z_1^2 + \sigma^2 x_0^2}{\sigma_0^2+\sigma^2}\right)}$$

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\left(x^2 - 2\left(\frac{\sigma_0^2}{\sigma_0^2+\sigma^2}z_1 + \frac{\sigma^2}{\sigma_0^2+\sigma^2}x_0\right)x + \frac{\sigma_0^2 z_1^2 + \sigma^2 x_0^2}{\sigma_0^2+\sigma^2}\right)}$$

$\frac{\sigma_0^2}{\sigma^2+\sigma_0^2}z_1 + \frac{\sigma^2}{\sigma_0^2+\sigma^2}x_0$ is the mean of the combined probability distribution, which we'll denote as μ.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\frac{\sigma^2+\sigma_0^2}{\sigma^2\sigma_0^2}\left(x^2 - 2\mu x + \frac{\sigma_0^2 z_1^2 + \sigma^2 x_0^2}{\sigma_0^2+\sigma^2}\right)}$$

Add in $\mu^2 - \mu^2$ to perform some factoring.

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\left(x^2 - 2\mu x + \mu^2 - \mu^2 + \frac{\sigma_0^2 z_1^2 + \sigma^2 x_0^2}{\sigma_0^2+\sigma^2}\right)}$$

$$p(x|z_1) = \frac{1}{C_1} e^{-\frac{1}{2}\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\left((x-\mu)^2-\mu^2+\frac{\sigma_0^2 z_1^2+\sigma^2 x_0^2}{\sigma_0^2+\sigma^2}\right)}$$

Pull out all constant terms in the exponent and combine them with C_1 to make a new constant C_2. We're basically doing $c_1 e^{x+a} \rightarrow c_1 e^a e^x \rightarrow c_2 e^x$.

$$p(x|z_1) = \frac{1}{C_2} e^{-\frac{1}{2}\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}(x-\mu)^2}$$

This means that if we're given an initial estimate x_0 and a measurement z_1 with associated means and variances represented by Gaussian distributions, this information can be combined into a third Gaussian distribution with its own mean value and variance. The expected value of x given z_1 is

$$E[x|z_1] = \mu = \frac{\sigma_0^2}{\sigma_0^2+\sigma^2} z_1 + \frac{\sigma^2}{\sigma_0^2+\sigma^2} x_0 \tag{9.11}$$

The variance of x given z_1 is

$$E[(x-\mu)^2|z_1] = \frac{\sigma^2\sigma_0^2}{\sigma_0^2+\sigma^2} \tag{9.12}$$

The expected value, which is also the maximum likelihood value, is the linear combination of the prior expected (maximum likelihood) value and the measurement. The expected value is a reasonable estimator of x.

$$\hat{x} = E[x|z_1] = \frac{\sigma_0^2}{\sigma_0^2+\sigma^2} z_1 + \frac{\sigma^2}{\sigma_0^2+\sigma^2} x_0 \tag{9.13}$$
$$\hat{x} = w_1 z_1 + w_2 x_0$$

Note that the weights w_1 and w_2 sum to 1. When the prior (i.e., prior knowledge of state) is uninformative (a large variance),

$$w_1 = \lim_{\sigma_0^2 \rightarrow \infty} \frac{\sigma_0^2}{\sigma_0^2+\sigma^2} = 1 \tag{9.14}$$

$$w_2 = \lim_{\sigma_0^2 \rightarrow \infty} \frac{\sigma^2}{\sigma_0^2+\sigma^2} = 0 \tag{9.15}$$

and $\hat{x} = z_1$. That is, the weight is on the observations and the estimate is equal to the measurement.

Let us assume we have a model providing an almost exact prior for x. In that case, σ_0^2 approaches 0 and

$$w_1 = \lim_{\sigma_0^2 \to 0} \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} = 0 \tag{9.16}$$

$$w_2 = \lim_{\sigma_0^2 \to 0} \frac{\sigma^2}{\sigma_0^2 + \sigma^2} = 1 \tag{9.17}$$

The Kalman filter uses this optimal fusion as the basis for its operation.

9.6 Kalman filter

So far, we've derived equations for updating the expected value and state covariance without measurements and how to incorporate measurements into an initial state optimally. Now, we'll combine these concepts to produce an estimator which minimizes the error covariance for linear systems.

9.6.1 Derivations

Given the *a posteriori* update equation $\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1})$, we want to find the value of $\mathbf{K}_{k+1}$ that minimizes the *a posteriori* estimate covariance (the error covariance) because this minimizes the estimation error.

a posteriori estimate covariance update equation

The following is the definition of the *a posteriori* estimate covariance matrix.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{+})$$

Substitute in the *a posteriori* update equation and expand the measurement equations.

$$\begin{aligned}
\mathbf{P}_{k+1}^{+} &= \operatorname{cov}(\mathbf{x}_{k+1} - (\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1}))) \\
\mathbf{P}_{k+1}^{+} &= \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1})) \\
\mathbf{P}_{k+1}^{+} &= \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\\
&\qquad (\mathbf{C}_{k+1}\mathbf{x}_{k+1} + \mathbf{D}_{k+1}\mathbf{u}_{k+1} + \mathbf{v}_{k+1}) \\
&\qquad - (\mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{D}_{k+1}\mathbf{u}_{k+1}))) \\
\mathbf{P}_{k+1}^{+} &= \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\\
&\qquad \mathbf{C}_{k+1}\mathbf{x}_{k+1} + \mathbf{D}_{k+1}\mathbf{u}_{k+1} + \mathbf{v}_{k+1} \\
&\qquad - \mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1}^{-} - \mathbf{D}_{k+1}\mathbf{u}_{k+1}))
\end{aligned}$$

Reorder terms.

$$\begin{aligned}\mathbf{P}_{k+1}^{+} = \operatorname{cov}(&\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\\ &\mathbf{C}_{k+1}\mathbf{x}_{k+1} - \mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1}^{-} \\ &+ \mathbf{D}_{k+1}\mathbf{u}_{k+1} - \mathbf{D}_{k+1}\mathbf{u}_{k+1} + \mathbf{v}_{k+1}))\end{aligned}$$

The $\mathbf{D}_{k+1}\mathbf{u}_{k+1}$ terms cancel.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\mathbf{C}_{k+1}\mathbf{x}_{k+1} - \mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{v}_{k+1}))$$

Distribute $\mathbf{K}_{k+1}$ to $\mathbf{v}_{k+1}$.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}(\mathbf{C}_{k+1}\mathbf{x}_{k+1} - \mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})$$

Factor out $\mathbf{C}_{k+1}$.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})$$

Factor out $\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}$ to the right.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})$$

Covariance is a linear operator, so it can be applied to each term separately. Covariance squares terms internally, so the negative sign on $\mathbf{K}_{k+1}\mathbf{v}_{k+1}$ is removed.

$$\mathbf{P}_{k+1}^{+} = \operatorname{cov}((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})) + \operatorname{cov}(\mathbf{K}_{k+1}\mathbf{v}_{k+1})$$

Now just evaluate the covariances.

$$\begin{aligned}\mathbf{P}_{k+1}^{+} &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\operatorname{cov}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^{\mathsf{T}} \\ &\quad + \mathbf{K}_{k+1}cov(\mathbf{v}_{k+1})\mathbf{K}_{k+1}^{\mathsf{T}} \\ \mathbf{P}_{k+1}^{+} &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^{\mathsf{T}} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}} \qquad (9.18)\end{aligned}$$

This is the *Joseph form* of the covariance update equation, which is valid for all Kalman gains.

The Joseph form is helpful in floating point implementations of the Kalman filter since it has better numerical stability than the optimal Kalman gain form discussed later.

Finding the optimal Kalman gain

The error in the *a posteriori* state estimation is $\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{+}$. We want to minimize the expected value of the square of the magnitude of this vector, which can be written as $E[(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{+})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{+})^{\mathsf{T}}]$. By definition, this expected value is the *a posteriori* error covariance $\mathbf{P}_{k+1}^{+}$. Remember that the eigenvectors of a matrix are the fundamental transformation directions of that matrix, and the eigenvalues are the magnitudes of those transformations. By minimizing the eigenvalues, we minimize the error variance (the uncertainty in the state estimate) for each state.

$\mathbf{P}_{k+1}^{+}$ is positive definite, so we know all the eigenvalues are positive. Therefore, a reasonable quantity to minimize with our choice of Kalman gain is the sum of the eigenvalues. We don't have direct access to the eigenvalues, but we can use the fact that the sum of the eigenvalues is equal to the trace of $\mathbf{P}_{k+1}^{+}$ and minimize that instead.[2]

We'll start with the equation for $\mathbf{P}_{k+1}^{+}$.

$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^{\mathsf{T}} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}}$$

We're going to expand the equation for $\mathbf{P}_{k+1}^{+}$ and collect terms. First, multiply in $\mathbf{P}_{k+1}^{-}$.

$$\mathbf{P}_{k+1}^{+} = (\mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-})(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^{\mathsf{T}} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}}$$

Tranpose each term in $\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}$. $\mathbf{I}$ is symmetric, so its transpose is dropped.

$$\mathbf{P}_{k+1}^{+} = (\mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-})(\mathbf{I} - \mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}}) + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}}$$

Multiply in $\mathbf{I} - \mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}}$.

$$\begin{aligned}\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}}) - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}}) \\ + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}}\end{aligned}$$

Expand terms.

$$\begin{aligned}\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-} \\ + \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^{\mathsf{T}}\mathbf{K}_{k+1}^{\mathsf{T}} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}}\end{aligned}$$

[2] By definition, the characteristic polynomial of an $n \times n$ matrix $\mathbf{P}$ is given by

$$p(t) = \det(t\mathbf{I} - \mathbf{P}) = t^n - \operatorname{tr}(\mathbf{P})t^{n-1} + \cdots + (-1)^n \det(\mathbf{P})$$

as well as $p(t) = (t - \lambda_1)\ldots(t - \lambda_n)$ where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $\mathbf{P}$. The coefficient for t^{n-1} in the second polynomial's expansion is $-(\lambda_1 + \cdots + \lambda_n)$. Therefore, by matching coefficients for t^{n-1}, we get $\operatorname{tr}(\mathbf{P}) = \lambda_1 + \cdots + \lambda_n$.

Factor out $\mathbf{K}_{k+1}$ and $\mathbf{K}_{k+1}^\mathsf{T}$.

$$\begin{aligned}\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-} \\ + \mathbf{K}_{k+1}(\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^\mathsf{T} + \mathbf{R}_{k+1})\mathbf{K}_{k+1}^\mathsf{T}\end{aligned}$$

$\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^\mathsf{T}+\mathbf{R}_{k+1}$ is the innovation (measurement residual) covariance at timestep $k+1$. We'll let this expression equal $\mathbf{S}_{k+1}$. We won't need it in the final theorem, but it makes the derivations after this point more concise.

$$\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T} \tag{9.19}$$

Now take the trace.

$$\begin{aligned}\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{P}_{k+1}^{-}\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) \\ + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T})\end{aligned}$$

Transpose one of the terms twice.

$$\begin{aligned}\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}((\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-\mathsf{T}})^\mathsf{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) \\ + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T})\end{aligned}$$

$\mathbf{P}_{k+1}^{-}$ is symmetric, so we can drop the transpose.

$$\begin{aligned}\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}((\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-})^\mathsf{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) \\ + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T})\end{aligned}$$

The trace of a matrix is equal to the trace of its transpose since the elements used in the trace are on the diagonal.

$$\begin{aligned}\operatorname{tr}(\mathbf{P}_{k+1}^{+}) &= \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) \\ &\quad + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T}) \\ \operatorname{tr}(\mathbf{P}_{k+1}^{+}) &= \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - 2\operatorname{tr}(\mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T})\end{aligned}$$

Given theorems 9.6.1 and 9.6.2

Theorem 9.6.1 $\frac{\partial}{\partial \mathbf{A}}\operatorname{tr}(\mathbf{A}\mathbf{B}\mathbf{A}^\mathsf{T}) = 2\mathbf{A}\mathbf{B}$ where $\mathbf{B}$ is symmetric.

Theorem 9.6.2 $\frac{\partial}{\partial \mathbf{A}} \operatorname{tr}(\mathbf{AC}) = \mathbf{C}^\mathsf{T}$

find the minimum of the trace of $\mathbf{P}_{k+1}^+$ by taking the partial derivative with respect to $\mathbf{K}$ and setting the result to $\mathbf{0}$.

$$\begin{aligned}
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^+)}{\partial \mathbf{K}} &= \mathbf{0} - 2(\mathbf{C}_{k+1}\mathbf{P}_{k+1}^-)^\mathsf{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^+)}{\partial \mathbf{K}} &= -2\mathbf{P}_{k+1}^{-\mathsf{T}}\mathbf{C}_{k+1}^\mathsf{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^+)}{\partial \mathbf{K}} &= -2\mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\mathbf{0} &= -2\mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
2\mathbf{K}_{k+1}\mathbf{S}_{k+1} &= 2\mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T} \\
\mathbf{K}_{k+1}\mathbf{S}_{k+1} &= \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T} \\
\mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{S}_{k+1}^{-1}
\end{aligned}$$

This is the optimal Kalman gain.

Simplified *a posteriori* estimate covariance update equation

If the optimal Kalman gain is used, the *a posteriori* estimate covariance matrix update equation can be simplified. First, we'll manipulate the equation for the optimal Kalman gain.

$$\begin{aligned}
\mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{S}_{k+1}^{-1} \\
\mathbf{K}_{k+1}\mathbf{S}_{k+1} &= \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T} \\
\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T} &= \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T}
\end{aligned}$$

Now we'll substitute it into equation (9.19).

$$\begin{aligned}
\mathbf{P}_{k+1}^+ &= \mathbf{P}_{k+1}^- - \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^- + \mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^\mathsf{T} \\
\mathbf{P}_{k+1}^+ &= \mathbf{P}_{k+1}^- - \mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^- + (\mathbf{P}_{k+1}^-\mathbf{C}_{k+1}^\mathsf{T}\mathbf{K}_{k+1}^\mathsf{T}) \\
\mathbf{P}_{k+1}^+ &= \mathbf{P}_{k+1}^- - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\mathbf{P}_{k+1}^-
\end{aligned}$$

Factor out $\mathbf{P}_{k+1}^-$ to the right.

$$\mathbf{P}_{k+1}^+ = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\mathbf{P}_{k+1}^-$$

9.6.2 Predict and update equations

Now that we've derived all the pieces we need, we can finally write all the equations for a Kalman filter. Theorem 9.6.3 shows the predict and update steps for a Kalman filter at the k^{th} timestep.

Intuitively, the predict step projects the error covariance forward in an increasing parabolic shape because you become less certain in your state estimate as you go longer since a measurement. In your state-space (think like 3D space but each state is an axis), the error ellipsoid grows over time.

The correct step decreases the error covariance again by injecting new information. The input has no effect on this because it has no noise associated with it. In fact, input cancels out in the Kalman filter expectation and covariance update equation derivations.

A Kalman filter chooses the Kalman gain $\mathbf{K}_{k+1}$ in the correct equation $\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1})$ such that the eigenvalues of $\mathbf{P}$ are minimized. This minimizes the error variances and thus the dimensions of the uncertainty ellipsoid over time.

If the update period is constant, the predict and correct steps are run in a loop as opposed to sporadically skipping correct steps, and the $(\mathbf{A}, \mathbf{C})$ pair is detectable, the error covariance matrix $\mathbf{P}$ willl approach a steady-state. This steady-state $\mathbf{P}$ results in a steady-state Kalman gain that can be used instead of the error covariance update equations to conserve computational resources.

Theorem 9.6.3 — Kalman filter.

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{+} + \mathbf{B}\mathbf{u}_k \tag{9.20}$$

$$\mathbf{P}_{k+1}^{-} = \mathbf{A}\mathbf{P}_{k}^{-}\mathbf{A}^{\mathsf{T}} + \mathbf{Q} \tag{9.21}$$

Update step

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}^{-}\mathbf{C}^{\mathsf{T}}(\mathbf{C}\mathbf{P}_{k+1}^{-}\mathbf{C}^{\mathsf{T}} + \mathbf{R})^{-1} \tag{9.22}$$

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-} - \mathbf{D}\mathbf{u}_{k+1}) \tag{9.23}$$

$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^{\mathsf{T}} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{\mathsf{T}} \tag{9.24}$$

$\mathbf{A}$	system matrix	$\hat{\mathbf{x}}$	state estimate vector
$\mathbf{B}$	input matrix	$\mathbf{u}$	input vector
$\mathbf{C}$	output matrix	$\mathbf{y}$	output vector
$\mathbf{D}$	feedthrough matrix	$\mathbf{Q}$	process noise covariance
$\mathbf{P}$	error covariance matrix	$\mathbf{R}$	measurement noise covariance
$\mathbf{K}$	Kalman gain matrix		

where a superscript of minus denotes *a priori* and plus denotes *a posteriori* estimate (before and after update respectively).

$\mathbf{C}$, $\mathbf{D}$, $\mathbf{Q}$, and $\mathbf{R}$ from the equations derived earlier are made constants here.

R To implement a discrete time Kalman filter from a continuous model, the model and continuous time $\mathbf{Q}$ and $\mathbf{R}$ matrices can be discretized using theorem 7.3.1.

Unknown states in a Kalman filter are generally represented by a Wiener (pronounced VEE-ner) process.[3] This process has the property that its variance increases linearly with time t.

[3] Explaining why we use the Wiener process would require going much more in depth into stochastic processes and Itô calculus, which is outside the scope of this book.

Matrix	Rows × Columns	Matrix	Rows × Columns
$\mathbf{A}$	states × states	$\hat{\mathbf{x}}$	states × 1
$\mathbf{B}$	states × inputs	$\mathbf{u}$	inputs × 1
$\mathbf{C}$	outputs × states	$\mathbf{y}$	outputs × 1
$\mathbf{D}$	outputs × inputs	$\mathbf{Q}$	states × states
$\mathbf{P}$	states × states	$\mathbf{R}$	outputs × outputs
$\mathbf{K}$	states × outputs		

Table 9.2: Kalman filter matrix dimensions

9.6.3 Setup

Equations to model

The following example system will be used to describe how to define and initialize the matrices for a Kalman filter.

A robot is between two parallel walls. It starts driving from one wall to the other at a velocity of 0.8 cm/s and uses ultrasonic sensors to provide noisy measurements of the distances to the walls in front of and behind it. To estimate the distance between the walls, we will define three states: robot position, robot velocity, and distance between the walls.

$$x_{k+1} = x_k + v_k \Delta T \tag{9.25}$$

$$v_{k+1} = v_k \tag{9.26}$$

$$x^w_{k+1} = x^w_k \tag{9.27}$$

This can be converted to the following state-space model.

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ v_k \\ x^w_k \end{bmatrix} \tag{9.28}$$

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1 \\ 0 \end{bmatrix} w_k \tag{9.29}$$

where the Gaussian random variable w_k has a mean of 0 and a variance of 1. The

observation model is

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \theta_k \tag{9.30}$$

where the covariance matrix of Gaussian measurement noise θ is a 2×2 matrix with both diagonals $10\ \text{cm}^2$.

The state vector is usually initialized using the first measurement or two. The covariance matrix entries are assigned by calculating the covariance of the expressions used when assigning the state vector. Let $k = 2$.

$$\mathbf{Q} = \begin{bmatrix} 1 \end{bmatrix} \tag{9.31}$$

$$\mathbf{R} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{9.32}$$

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{y}_{k,1} \\ (\mathbf{y}_{k,1} - \mathbf{y}_{k-1,1})/dt \\ \mathbf{y}_{k,1} + \mathbf{y}_{k,2} \end{bmatrix} \tag{9.33}$$

$$\mathbf{P} = \begin{bmatrix} 10 & 10/dt & 10 \\ 10/dt & 20/dt^2 & 10/dt \\ 10 & 10/dt & 20 \end{bmatrix} \tag{9.34}$$

Initial conditions

To fill in the $\mathbf{P}$ matrix, we calculate the covariance of each combination of state variables. The resulting value is a measure of how much those variables are correlated. Due to how the covariance calculation works out, the covariance between two variables is the sum of the variance of matching terms which aren't constants multiplied by any constants the two have. If no terms match, the variables are uncorrelated and the covariance is zero.

In $\mathbf{P}_{11}$, the terms in $\mathbf{x}_1$ correlate with itself. Therefore, $\mathbf{P}_{11}$ is $\mathbf{x}_1$'s variance, or $\mathbf{P}_{11} = 10$. For $\mathbf{P}_{21}$, One term correlates between $\mathbf{x}_1$ and $\mathbf{x}_2$, so $\mathbf{P}_{21} = \frac{10}{dt}$. The constants from each are simply multiplied together. For $\mathbf{P}_{22}$, both measurements are correlated, so the variances add together. Therefore, $\mathbf{P}_{22} = \frac{20}{dt^2}$. It continues in this fashion until the matrix is filled up. Order doesn't matter for correlation, so the matrix is symmetric.

Selection of priors

Choosing good priors is important for a well performing filter, even if little information is known. This applies to both the measurement noise and the noise model. The act of giving a state variable a large variance means you know something about the system. Namely, you aren't sure whether your initial guess is close to the true state. If you make a guess and specify a small variance, you are telling the filter that you are very confident

in your guess. If that guess is incorrect, it will take the filter a long time to move away from your guess to the true value.

Covariance selection

While one could assume no correlation between the state variables and set the covariance matrix entries to zero, this may not reflect reality. The Kalman filter is still guaranteed to converge to the steady-state covariance after an infinite time, but it will take longer than otherwise.

Noise model selection

We typically use a Gaussian distribution for the noise model because the sum of many independent random variables produces a normal distribution by the central limit theorem. Kalman filters only require that the noise is zero-mean. If the true value has an equal probability of being anywhere within a certain range, use a uniform distribution instead. Each of these communicates information regarding what you know about a system in addition to what you do not.

Process noise and measurement noise covariance selection

Recall that the process noise covariance is $\mathbf{Q}$ and the measurement noise covariance is $\mathbf{R}$. To tune the elements of these, it can be helpful to take a collection of measurements, then run the Kalman filter on them offline to evaluate its performance.

The diagonal elements of $\mathbf{R}$ are the variances of each measurement, which can be easily determined from the offline measurements. The diagonal elements of $\mathbf{Q}$ are the variances of each state. They represent how much each state is expected to deviate from the model.

Selecting $\mathbf{Q}$ is more difficult. If the data is trusted too much over the model, one risks overfitting the data. One should balance estimating any hidden states sufficiently with actually filtering out the noise.

Modeling other noise colors

The Kalman filter assumes a model with zero-mean white noise. If the model is incomplete in some way, whether it's missing dynamics or assumes an incorrect noise model, the residual $\widetilde{\mathbf{y}} = \mathbf{y} - \mathbf{C}\hat{\mathbf{x}}$ over time will have probability characteristics not indicative of white noise (e.g., it isn't zero-mean).

To handle other colors of noise in a Kalman filter, define that color of noise in terms of white noise and augment the model with it.

9.6.4 Simulation

Figure 9.3 shows the state estimates and measurements of the Kalman filter over time. Figure 9.4 shows the position estimate and variance over time. Figure 9.5 shows the

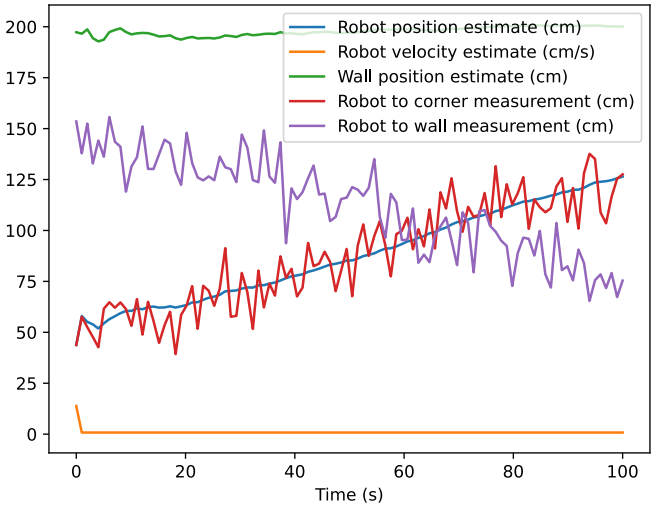


Figure 9.3: State estimates and measurements with Kalman filter

wall position estimate and variance over time. Notice how the variances decrease over time as the filter gathers more measurements. This means that the filter becomes more confident in its state estimates.

The final precisions in estimating the position of the robot and the wall are the square roots of the corresponding elements in the covariance matrix. That is, 0.5188 m and 0.4491 m respectively. They are smaller than the precision of the raw measurements, $\sqrt{10} = 3.1623$ m. As expected, combining the information from several measurements produces a better estimate than any one measurement alone.

9.6.5 Kalman filter as Luenberger observer

A Kalman filter can be represented as a Luenberger observer by letting $\mathbf{L} = \mathbf{A}\mathbf{K}_k$ (see appendix D.2). The Luenberger observer has a constant observer gain matrix $\mathbf{L}$, so the steady-state Kalman gain is used to calculate it. We will demonstrate how to find this shortly.

Kalman filter theory provides a way to place the poles of the Luenberger observer optimally in the same way we placed the poles of the controller optimally with LQR. The eigenvalues of the Kalman filter are

$$\text{eig}(\mathbf{A}(\mathbf{I} - \mathbf{K}_k\mathbf{C})) \tag{9.35}$$

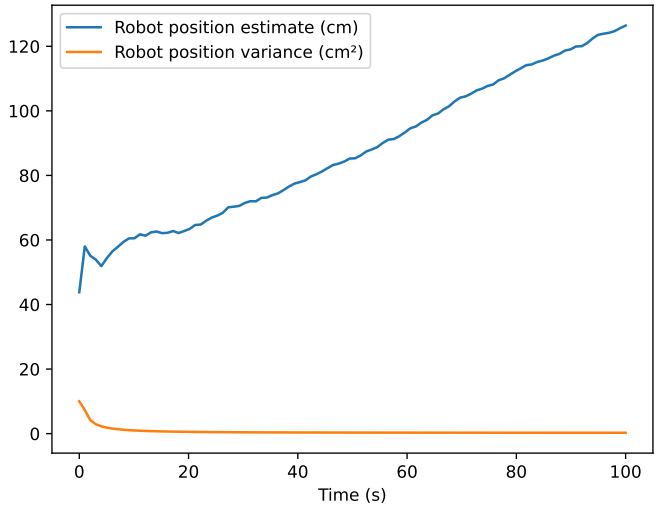


Figure 9.4: Robot position estimate and variance with Kalman filter

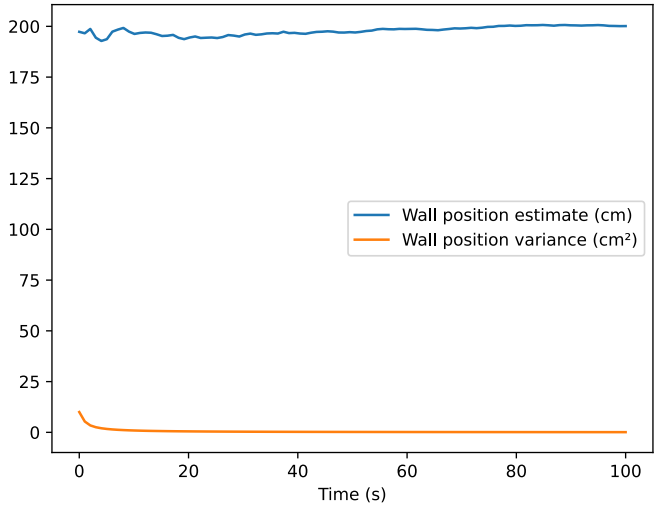


Figure 9.5: Wall position estimate and variance with Kalman filter

Steady-state Kalman gain

One may have noticed that the error covariance matrix can be updated independently of the rest of the model. The error covariance matrix tends toward a steady-state value, and this matrix can be obtained via the discrete algebraic Riccati equation. This can then be used to compute a steady-state Kalman gain.

General matrix inverses like $\mathbf{S}^{-1}$ are expensive. We want to put $\mathbf{K} = \mathbf{P}\mathbf{C}^\mathsf{T}\mathbf{S}^{-1}$ into $\mathbf{A}\mathbf{x} = \mathbf{b}$ form so we can solve it more efficiently.

$$\begin{aligned}
\mathbf{K} &= \mathbf{P}\mathbf{C}^\mathsf{T}\mathbf{S}^{-1} \\
\mathbf{K}\mathbf{S} &= \mathbf{P}\mathbf{C}^\mathsf{T} \\
(\mathbf{K}\mathbf{S})^\mathsf{T} &= (\mathbf{P}\mathbf{C}^\mathsf{T})^\mathsf{T} \\
\mathbf{S}^\mathsf{T}\mathbf{K}^\mathsf{T} &= \mathbf{C}\mathbf{P}^\mathsf{T}
\end{aligned}$$

The solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be found via $\mathbf{x} = \text{solve}(\mathbf{A}, \mathbf{b})$.

$$\begin{aligned}
\mathbf{K}^\mathsf{T} &= \text{solve}(\mathbf{S}^\mathsf{T}, \mathbf{C}\mathbf{P}^\mathsf{T}) \\
\mathbf{K} &= \text{solve}(\mathbf{S}^\mathsf{T}, \mathbf{C}\mathbf{P}^\mathsf{T})^\mathsf{T}
\end{aligned}$$

Snippet 9.1 computes the steady-state Kalman gain matrix.

```
"""Function for computing the steady-state Kalman gain matrix."""

import numpy as np
import scipy as sp


def kalmd(A, C, Q, R):
    """Solves for the discrete steady-state Kalman gain.

    Keyword arguments:
    A -- numpy.array(states x states), system matrix.
    C -- numpy.array(outputs x states), output matrix.
    Q -- numpy.array(states x states), process noise covariance matrix.
    R -- numpy.array(outputs x outputs), measurement noise covariance matrix.

    Returns:
    K -- numpy.array(outputs x states), Kalman gain matrix.
    """
    P = sp.linalg.solve_discrete_are(a=A.T, b=C.T, q=Q, r=R)
    return np.linalg.solve(C @ P @ C.T + R, C @ P.T).T
```

Snippet 9.1. Steady-state Kalman gain matrix calculation in Python

9.7 Kalman smoother

The Kalman filter uses the data up to the current time to produce an optimal estimate of the system state. If data beyond the current time is available, it can be ran through

a Kalman smoother to produce a better estimate. This is done by recording measurements, then applying the smoother to it offline.

The Kalman smoother does a forward pass on the available data, then a backward pass through the system dynamics so it takes into account the data before and after the current time. This produces state variances that are lower than that of a Kalman filter.

9.7.1 Predict and update equations

One first does a forward pass with the typical Kalman filter equations and stores the results. Then one can use the Rauch-Tung-Striebel (RTS) algorithm to do the backward pass. Theorem 9.7.1 shows the predict and and update steps for the forward and backward passes for a Kalman smoother at the k^{th} timestep.

See section 3 of https://users.aalto.fi/~ssarkka/course_k2011/pdf/handout7.pdf for a derivation of the Rauch-Tung-Striebel smoother.

Theorem 9.7.1 — Kalman smoother.

Forward predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{+} + \mathbf{B}\mathbf{u}_k \tag{9.36}$$

$$\mathbf{P}_{k+1}^{-} = \mathbf{A}\mathbf{P}_{k}^{-}\mathbf{A}^{\mathsf{T}} + \mathbf{Q} \tag{9.37}$$

Forward update step

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}^{-}\mathbf{C}^{\mathsf{T}}(\mathbf{C}\mathbf{P}_{k+1}^{-}\mathbf{C}^{\mathsf{T}} + \mathbf{R})^{-1} \tag{9.38}$$

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-} - \mathbf{D}\mathbf{u}_{k+1}) \tag{9.39}$$

$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C})\mathbf{P}_{k+1}^{-} \tag{9.40}$$

Backward update step

$$\mathbf{K}_k = \mathbf{P}_k^{+}\mathbf{A}_k^{\mathsf{T}}(\mathbf{P}_{k+1}^{-})^{-1} \tag{9.41}$$

$$\hat{\mathbf{x}}_{k|N} = \hat{\mathbf{x}}_k^{+} + \mathbf{K}_k(\hat{\mathbf{x}}_{k+1|N} - \hat{\mathbf{x}}_{k+1}^{-}) \tag{9.42}$$

$$\mathbf{P}_{k|N} = \mathbf{P}_k^{+} + \mathbf{K}_k(\mathbf{P}_{k+1|N} - \mathbf{P}_{k+1}^{-})\mathbf{K}_k^{\mathsf{T}} \tag{9.43}$$

Backward initial conditions

$$\hat{\mathbf{x}}_{N|N} = \hat{\mathbf{x}}_N^{+} \tag{9.44}$$

$$\mathbf{P}_{N|N} = \mathbf{P}_N^{+} \tag{9.45}$$

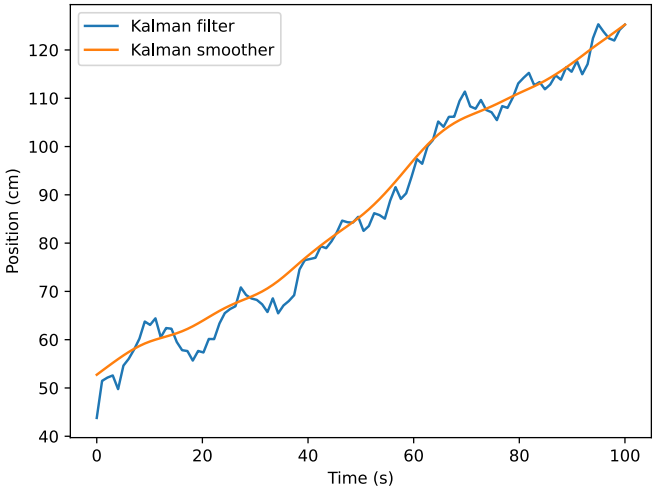


Figure 9.6: Robot position with Kalman smoother

9.7.2 Example

We will modify the robot model so that instead of a velocity of 0.8 cm/s with random noise, the velocity is modeled as a random walk from the current velocity.

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ v_k \\ x_k^w \end{bmatrix} \tag{9.46}$$

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0.1 \\ 0 \end{bmatrix} w_k \tag{9.47}$$

We will use the same observation model as before.

Using the same data from subsection 9.6.4, figures 9.6, 9.7, and 9.8 show the improved state estimates and figure 9.9 shows the improved robot position covariance with a Kalman smoother.

Notice how the wall position produced by the smoother is a constant. This is because that state has no dynamics, so the final estimate from the Kalman filter is already the best estimate.

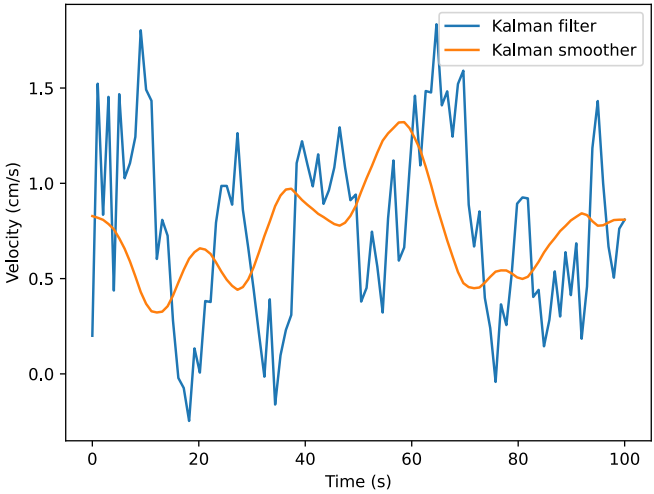


Figure 9.7: Robot velocity with Kalman smoother

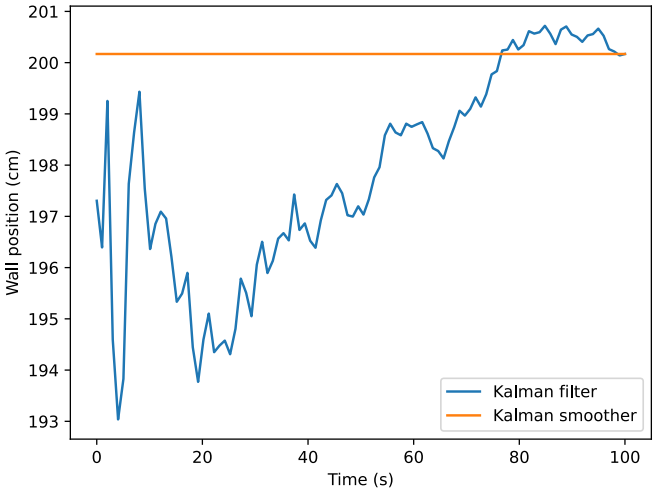


Figure 9.8: Wall position with Kalman smoother

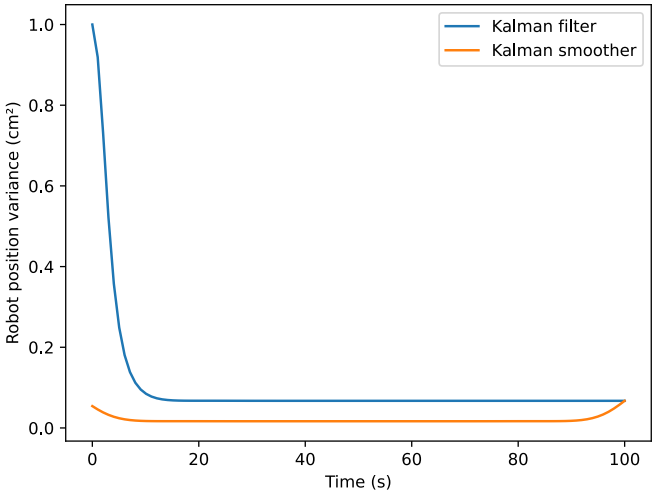


Figure 9.9: Robot position variance with Kalman smoother

See Roger Labbe's book *Kalman and Bayesian Filters in Python* for more on smoothing.[4]

[4] https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/13-Smoothing.ipynb

9.8 Extended Kalman filter

In this book, we have covered the Kalman filter, which is the optimal unbiased estimator for linear systems. One method for extending it to nonlinear systems is the extended Kalman filter.

The extended Kalman filter (EKF) linearizes the dynamics and measurement models during the prediction and correction steps respectively. Then, the linear Kalman filter equations are used to compute the error covariance matrix $\mathbf{P}$ and Kalman gain matrix $\mathbf{K}$.

Theorem 9.8.1 shows the predict and update steps for an extended Kalman filter at the k^{th} timestep.

Theorem 9.8.1 — Extended Kalman filter.

Predict step

$$\mathbf{A} = \underbrace{\left.\frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_k^+,\mathbf{u}_k}}_{\text{Linearize } f(\mathbf{x},\mathbf{u})} \tag{9.48}$$

$$\mathbf{A}_k = \underbrace{e^{\mathbf{A}T}}_{\text{Discretize } \mathbf{A}} \tag{9.49}$$

$$\hat{\mathbf{x}}_{k+1}^- = \underbrace{\text{RK4}(f, \hat{\mathbf{x}}_k^+, \mathbf{u}_k, T)}_{\text{Numerical integration}} \tag{9.50}$$

$$\mathbf{P}_{k+1}^- = \mathbf{A}_k \mathbf{P}_k^- \mathbf{A}_k^\mathsf{T} + \mathbf{Q}_k \tag{9.51}$$

Update step

$$\mathbf{C}_{k+1} = \underbrace{\left.\frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k+1}^-,\mathbf{u}_{k+1}}}_{\text{Linearize } h(\mathbf{x},\mathbf{u})} \tag{9.52}$$

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}^- \mathbf{C}_{k+1}^\mathsf{T} (\mathbf{C}_{k+1} \mathbf{P}_{k+1}^- \mathbf{C}_{k+1}^\mathsf{T} + \mathbf{R}_{k+1})^{-1} \tag{9.53}$$

$$\hat{\mathbf{x}}_{k+1}^+ = \hat{\mathbf{x}}_{k+1}^- + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - h(\hat{\mathbf{x}}_{k+1}^-, \mathbf{u}_{k+1})) \tag{9.54}$$

$$\mathbf{P}_{k+1}^+ = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})\mathbf{P}_{k+1}^- (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1})^\mathsf{T} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^\mathsf{T} \tag{9.55}$$

$f(\mathbf{x},\mathbf{u})$	continuous dynamics model	$\hat{\mathbf{x}}$	state estimate vector
$h(\mathbf{x},\mathbf{u})$	measurement model	$\mathbf{u}$	input vector
T	sample timestep duration	$\mathbf{y}$	output vector
$\mathbf{P}$	error covariance matrix	$\mathbf{Q}$	process noise covariance
$\mathbf{K}$	Kalman gain matrix	$\mathbf{R}$	measurement noise covariance

where a superscript of minus denotes *a priori* and plus denotes *a posteriori* estimate (before and after update respectively).

R To implement a discrete time extended Kalman filter from a continuous model, the dynamics model can be numerically integrated via a method from section 7.9 and the continuous time $\mathbf{Q}$ and $\mathbf{R}$ matrices can be discretized using theorem 7.3.1.

Matrix	Rows × Columns	Matrix	Rows × Columns
$\mathbf{A}$	states × states	$\hat{\mathbf{x}}$	states × 1
$\mathbf{C}$	outputs × states	$\mathbf{u}$	inputs × 1
$\mathbf{P}$	states × states	$\mathbf{y}$	outputs × 1
$\mathbf{K}$	states × outputs	$\mathbf{Q}$	states × states
		$\mathbf{R}$	outputs × outputs

Table 9.3: Extended Kalman filter matrix dimensions

9.9 Unscented Kalman filter

In this book, we have covered the Kalman filter, which is the optimal unbiased estimator for linear systems. One method for extending it to nonlinear systems is the unscented Kalman filter.

The unscented Kalman filter (UKF) propagates carefully chosen points called sigma points through the nonlinear state and measurement models to obtain estimates of the true covariances (as opposed to linearized versions of them). We recommend reading Roger Labbe's book *Kalman and Bayesian Filters in Python* for more on UKFs.[5]

The original paper on the UKF is also an option [6]. See also the equations for van der Merwe's sigma point algorithm [21]. Here's a paper on a quaternion-based Unscented Kalman filter for orientation tracking [8].

Here's an interview about the origin of the UKF with its creator [20].

9.9.1 Square-root UKF

The UKF uses a matrix square root (Cholesky decomposition) to compute the scaling for the sigma points. This operation can introduce numerical instability. The square-root UKF avoids this by propagating the square-root of the error covariance directly instead of the error covariance; the Cholesky decomposition is replaced with a QR decomposition and a Cholesky rank-1 update.

See the original paper [13] and this implementation tutorial [18].

[5] https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/10-Unscented-Kalman-Filter.ipynb

9.10 Multiple model adaptive estimation

Multiple model adaptive estimation (MMAE) runs multiple Kalman filters with different models on the same data. The Kalman filter with the lowest residual has the highest likelihood of accurately reflecting reality. This can be used to detect certain system states like an aircraft engine failing without needing to invest in costly sensors to determine this directly.

For example, say you have three Kalman filters: one for turning left, one for turning right, and one for going straight. If the control input is attempting to fly the plane straight and the Kalman filter for going left has the lowest residual, the aircraft's left engine probably failed.

See Roger Labbe's book *Kalman and Bayesian Filters in Python* for more on MMAE.[6]

[6]MMAE section of https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/14-Adaptive-Filtering.ipynb

10. Pose estimation

Pose is defined as the position and orientation of an agent (a system with a controller). The plant usually includes the pose in its state vector. We'll cover several methods for estimating an agent's pose from local measurements such as encoders and gyroscope heading.

10.1 Forward Euler integration

The simplest way to perform pose estimation via dead reckoning (that is, no direct measurements of the pose are used) is to integrate the velocity in each orthogonal direction over time. In two dimensions, one could use

$$\begin{aligned} x_{k+1} &= x_k + v_k \cos\theta_k \, T \\ y_{k+1} &= y_k + v_k \sin\theta_k \, T \\ \theta_{k+1} &= \theta_{gyro,k+1} \end{aligned}$$

where T is the sample period. This odometry approach assumes that the robot follows a straight path between samples (that is, $\omega = 0$ at all but the sample times).

10.2 Pose exponential

We can obtain a more accurate approximation of the pose by including first-order dynamics for the heading θ. To provide a rationale for the math we're about to do, we

need to cover some aspects of group theory.

10.2.1 What is a group?

In mathematics, a group is a set equipped with a binary operation (an operation with two arguments) that combines any two elements (of the set) to form a third element in such a way that four conditions called *group axioms* are satisfied: closure, associativity, identity, and invertibility.

Closure means that the result is in the same set as the arguments.

Associativity means that within an expression containing two or more occurrences in a row of the same associative operator, the order in which the operations are performed does not matter as long as the sequence of the operands is not changed. In other words, different groupings of the operations produces the same result.

Identity, or an identity element, is a special type of element of a set with respect to a binary operation on that set, which leaves any element of the set unchanged when combined with it. For example, the additive identity of the set of integers is zero, which means that any integer summed with zero produces that integer.

Invertibility means there is an element that can "undo" the effect of combination with another given element. For integers and the addition operator, the inverse element would be the negation.

10.2.2 What is a pose?

To develop what a pose is in group theory, we need to define a few key groups. SO(2) is the special orthogonal group in dimension 2. They represent a 2D rotation.

SE(2) is the special euclidean group in dimension 2. They represent a 2D rotation and a 2D translation, which we call a 2D pose. In other words, pose is an element of SE(2).

10.2.3 What is a twist?

A 2D twist is an element of the tangent space of SE(2) (like the tangential distance traveled by the robot along an arc in SE(2)). We use the "pose exponential" to map a twist (an element of the tangent space) to an element of SE(2). In other words, we map a twist to a pose.

We call it a pose exponential because it's an exponential map onto a pose. The term exponential is used because an exponential is the solution to integrating a differential equation whose derivative of a value is proportional to the value itself. For example, $\frac{dx}{dt} = ax$ has the solution $x = x_0 e^{at}$.

We use the pose exponential to take encoder measurement deltas and gyro angle deltas

(which are in the tangent space and are thus a twist) and turn them into a change in pose. This gets added to the pose from the last update.

10.2.4 Derivation

We can obtain a more accurate approximation of the pose than Euler integration by including first-order dynamics for the heading θ.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

v_x, v_y, and ω are the x and y velocities of the robot within its local coordinate frame, which will be treated as constants.

There are two coordinate frames used here: robot and global. A superscript on the left side of a matrix denotes the coordinate frame in which that matrix is represented. The robot's coordinate frame is denoted by R and the global coordinate frame is denoted by G.

In the robot frame (the tangent space)

$$\begin{aligned} {}^R dx &= {}^R v_x \, dt \\ {}^R dy &= {}^R v_y \, dt \\ {}^R d\theta &= {}^R \omega \, dt \end{aligned}$$

To transform this into the global frame SE(2), we apply a counterclockwise rotation matrix where θ changes over time.

$$\begin{aligned} {}^G \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} &= \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^R \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} dt \\ {}^G \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} &= \begin{bmatrix} \cos\omega t & -\sin\omega t & 0 \\ \sin\omega t & \cos\omega t & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^R \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} dt \end{aligned}$$

Now, integrate the matrix equation (matrices are integrated element-wise). This derivation heavily utilizes the integration method described in section 4.3.

$${}^G \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \frac{\sin\omega t}{\omega} & \frac{\cos\omega t}{\omega} & 0 \\ -\frac{\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0 \\ 0 & 0 & t \end{bmatrix} {}^R \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \Bigg|_0^t$$

$$
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\frac{\sin\omega t}{\omega} & \frac{\cos\omega t-1}{\omega} & 0\\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0\\ 0 & 0 & t\end{bmatrix} {}^{R}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix}
$$

This equation assumes a starting orientation of $\theta = 0$. For nonzero starting orientations, we can apply a counterclockwise rotation by θ.

$$
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\frac{\sin\omega t}{\omega} & \frac{\cos\omega t-1}{\omega} & 0\\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0\\ 0 & 0 & t\end{bmatrix} {}^{R}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix} \tag{10.1}
$$

Control system implementations will generally have a model update and a controller update in a given iteration. Equation (10.1) (the model update) uses the current velocity to advance the state to the next timestep (into the future). Since controllers use the current state, the controller update should be run before the model update.

If we factor out a t, we can use change in pose between updates instead of velocities.

$$
\begin{aligned}
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} &= \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\frac{\sin\omega t}{\omega} & \frac{\cos\omega t-1}{\omega} & 0\\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0\\ 0 & 0 & t\end{bmatrix} {}^{R}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix}\\
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} &= \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\frac{\sin\omega t}{\omega t} & \frac{\cos\omega t-1}{\omega t} & 0\\ \frac{1-\cos\omega t}{\omega t} & \frac{\sin\omega t}{\omega t} & 0\\ 0 & 0 & 1\end{bmatrix} {}^{R}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix} t\\
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} &= \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\frac{\sin\omega t}{\omega t} & \frac{\cos\omega t-1}{\omega t} & 0\\ \frac{1-\cos\omega t}{\omega t} & \frac{\sin\omega t}{\omega t} & 0\\ 0 & 0 & 1\end{bmatrix} {}^{R}\begin{bmatrix}v_x t\\ v_y t\\ \omega t\end{bmatrix}\\
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} &= \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} {}^{R}\begin{bmatrix}\frac{\sin\Delta\theta}{\Delta\theta} & \frac{\cos\Delta\theta-1}{\Delta\theta} & 0\\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \frac{\sin\Delta\theta}{\Delta\theta} & 0\\ 0 & 0 & 1\end{bmatrix} {}^{R}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix}
\end{aligned} \tag{10.2}
$$

The vector ${}^{R}\begin{bmatrix}\Delta x & \Delta y & \Delta\theta\end{bmatrix}^{\mathsf{T}}$ is a twist because it's an element of the tangent space (the robot's local coordinate frame).

Control system implementations will generally have a model update and a controller update in a given iteration. Equation (10.2) (the model update) uses local

distance and heading deltas between the previous and current timestep, so it advances the state to the current timestep. Since controllers use the current state, the controller update should be run after the model update.

When the robot is traveling on a straight trajectory ($\Delta\theta = 0$), some expressions within the equation above are indeterminate. We can approximate these with Taylor series expansions.

$$\begin{aligned}
\frac{\sin\Delta\theta}{\Delta\theta} &= 1 - \frac{\Delta\theta^2}{6} + \ldots &&\approx 1 - \frac{\Delta\theta^2}{6} \\
\frac{\cos\Delta\theta - 1}{\Delta\theta} &= -\frac{\Delta\theta}{2} + \frac{\Delta\theta^3}{24} - \ldots &&\approx -\frac{\Delta\theta}{2} \\
\frac{1 - \cos\Delta\theta}{\Delta\theta} &= \frac{\Delta\theta}{2} - \frac{\Delta\theta^3}{24} + \ldots &&\approx \frac{\Delta\theta}{2}
\end{aligned}$$

Theorem 10.2.1 — Pose exponential.

$$\begin{bmatrix}\Delta x \\ \Delta y \\ \Delta\theta\end{bmatrix}_G = \begin{bmatrix}\cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1\end{bmatrix}_R \begin{bmatrix}\frac{\sin\Delta\theta}{\Delta\theta} & \frac{\cos\Delta\theta - 1}{\Delta\theta} & 0 \\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \frac{\sin\Delta\theta}{\Delta\theta} & 0 \\ 0 & 0 & 1\end{bmatrix}_R \begin{bmatrix}\Delta x \\ \Delta y \\ \Delta\theta\end{bmatrix} \tag{10.3}$$

where G denotes global coordinate frame and R denotes robot's coordinate frame.

For sufficiently small $\Delta\theta$:

$$\frac{\sin\Delta\theta}{\Delta\theta} = 1 - \frac{\Delta\theta^2}{6} \qquad \frac{\cos\Delta\theta - 1}{\Delta\theta} = -\frac{\Delta\theta}{2} \qquad \frac{1 - \cos\Delta\theta}{\Delta\theta} = \frac{\Delta\theta}{2} \tag{10.4}$$

Δx	change in pose's x	Δy	change in pose's y
$\Delta\theta$	change in pose's θ	θ	starting angle in global coordinate frame

This change in pose can be added directly to the previous pose estimate to update it.

Figures 10.1 through 10.4 show the pose estimation errors of forward Euler odometry and pose exponential odometry for a feedforward S-curve trajectory (dt = 20 ms).

The highest errors for the 8.84 m by 5.0 m trajectory are 3.415 cm in x, 0.158 cm in y, and -0.644 deg in heading. The difference would be even more noticeable for paths with higher curvatures and longer durations. The error returns to near zero in this case

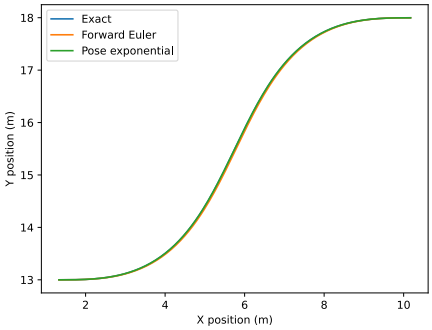


Figure 10.1: Pose estimation comparison (y vs x)

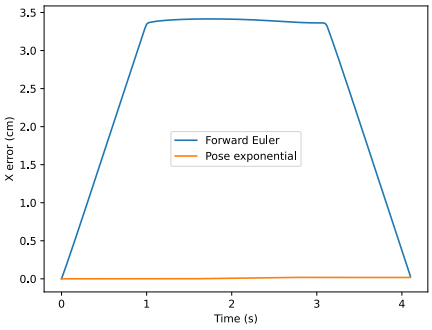


Figure 10.2: Pose estimation comparison (x error vs time)

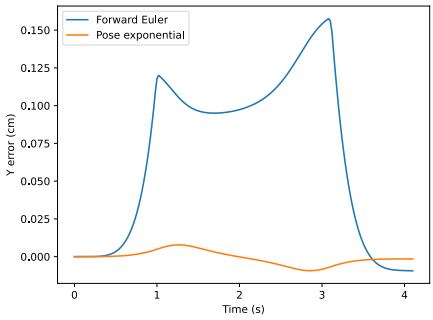


Figure 10.3: Pose estimation comparison (y error vs time)

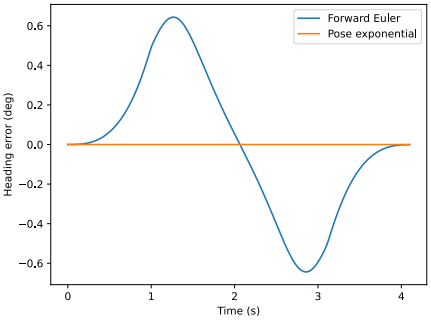


Figure 10.4: Pose estimation comparison (heading error vs time)

because the curvature is symmetric, so the second half cancels the error accrued in the first half.

Using a smaller update period somewhat mitigates the forward Euler pose estimation error. However, there are bigger sources of error like turning scrub on skid steer robots that should be dealt with before odometry numerical accuracy.

10.2.5 Lie groups

While we avoided the topic in our explanation, pose is what's known as a Lie group (a group that is also a differentiable manifold). There's a lot of mathematical and controls results developed around Lie groups, so we're mentioning the connection here in case

you want to search the Internet for more information.

10.3 Pose correction

The previous methods for pose estimation have assumed no direct pose measurements are available. At best, only heading is available. To augment any of them with corrections from full pose measurements, an Extended Kalman filter (section 9.8) or Unscented Kalman filter (section 9.9) can be used.

This page intentionally left blank

IV System modeling

This page intentionally left blank

11. Dynamics

11.1 Linear motion

$$\sum F = ma$$

where $\sum F$ is the sum of all forces applied to an object in Newtons, m is the mass of the object in kg, and a is the net acceleration of the object in $\frac{m}{s^2}$.

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2$$

where $x(t)$ is an object's position at time t, x_0 is the initial position, v_0 is the initial velocity, and a is the acceleration.

11.2 Angular motion

$$\sum \tau = I\alpha$$

where $\sum \tau$ is the sum of all torques applied to an object in Newton-meters, I is the moment of inertia of the object in $kg\text{-}m^2$ (also called the rotational mass), and α is the net angular acceleration of the object in $\frac{rad}{s^2}$.

$$\theta(t) = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2$$

where $\theta(t)$ is an object's angle at time t, θ_0 is the initial angle, ω_0 is the initial angular velocity, and α is the angular acceleration.

11.3 Vectors

Vectors are quantities with a magnitude and a direction. Vectors in three-dimensional space have a coordinate for each spatial direction x, y, and z. Let's take the vector $\vec{a} = \langle 1, 2, 3 \rangle$. $\vec{a}$ is a three-dimensional vector that describes a movement 1 unit in the x direction, 2 units in the y direction, and 3 units in the z direction.

We define $\hat{i}$, $\hat{j}$, and $\hat{k}$ as vectors that represent the fundamental movements one can make the three-dimensional space: 1 unit of movement in the x direction, 1 unit of movement y direction, and 1 unit of movement in the z direction respectively. These three vectors form a *basis* of three-dimensional space because copies of them can be added together to reach any point in three-dimensional space.

$$\hat{i} = \langle 1, 0, 0 \rangle$$
$$\hat{j} = \langle 0, 1, 0 \rangle$$
$$\hat{k} = \langle 0, 0, 1 \rangle$$

We can also write the vector $\vec{a}$ in terms of these basis vectors.

$$\vec{a} = 1\hat{i} + 2\hat{j} + 3\hat{k}$$

11.3.1 Basic vector operations

We will now show this is equivalent to the original notation through some vector mathematics. First, we'll substitute in the values for the basis vectors.

$$\vec{a} = 1\langle 1, 0, 0 \rangle + 2\langle 0, 1, 0 \rangle + 3\langle 0, 0, 1 \rangle$$

Scalars are multiplied component-wise with vectors.

$$\vec{a} = \langle 1, 0, 0 \rangle + \langle 0, 2, 0 \rangle + \langle 0, 0, 3 \rangle$$

Vectors are added by summing each of their components.

$$\vec{a} = \langle 1, 2, 3 \rangle$$

11.3.2 Cross product

The cross product is denoted by $\times$. The cross product of the basis vectors are computed as follows.

$$\hat{i} \times \hat{j} = \hat{k}$$
$$\hat{j} \times \hat{k} = \hat{i}$$
$$\hat{k} \times \hat{i} = \hat{j}$$

They proceed in a cyclic fashion through i, j, and k. If a vector is crossed with itself, it produces the zero vector (a scalar zero for each coordinate). The cross products of the basis vectors in the opposite order progress backwards and include a negative sign.

$$\hat{i} \times \hat{k} = -\hat{j}$$
$$\hat{k} \times \hat{j} = -\hat{i}$$
$$\hat{j} \times \hat{i} = -\hat{k}$$

Given vectors $\vec{u} = a\hat{i} + b\hat{j} + c\hat{k}$ and $\vec{v} = d\hat{i} + e\hat{j} + f\hat{k}$, $\vec{u} \times \vec{v}$ is computed using the distributive property.

$$\begin{aligned}
\vec{u} \times \vec{v} &= (a\hat{i} + b\hat{j} + c\hat{k}) \times (d\hat{i} + e\hat{j} + f\hat{k}) \\
\vec{u} \times \vec{v} &= ad(\hat{i} \times \hat{i}) + ae(\hat{i} \times \hat{j}) + af(\hat{i} \times \hat{k}) \\
&\quad + bd(\hat{j} \times \hat{i}) + be(\hat{j} \times \hat{j}) \\
&\quad + bf(\hat{j} \times \hat{k}) \\
&\quad + cd(\hat{k} \times \hat{i}) + ce(\hat{k} \times \hat{j}) + cf(\hat{k} \times \hat{k}) \\
\vec{u} \times \vec{v} &= ae\hat{k} + af(-\hat{j}) + bd(-\hat{k}) + bf\hat{i} + cd\hat{j} + ce(-\hat{i}) \\
\vec{u} \times \vec{v} &= ae\hat{k} - af\hat{j} - bd\hat{k} + bf\hat{i} + cd\hat{j} - ce\hat{i} \\
\vec{u} \times \vec{v} &= (bf - ce)\hat{i} + (cd - af)\hat{j} + (ae - bd)\hat{k}
\end{aligned}$$

11.4 Curvilinear motion

Curvilinear motion describes the motion of an object along a fixed curve. This motion has both linear and angular components. For derivations involving curvilinear motion, we'll assume positive x ($\hat{i}$) is forward, positive y ($\hat{j}$) is to the left, positive z ($\hat{k}$) is up, and the robot is facing in the x direction. This axes convention is known as North-West-Up (NWU), and is shown in figure 11.1.

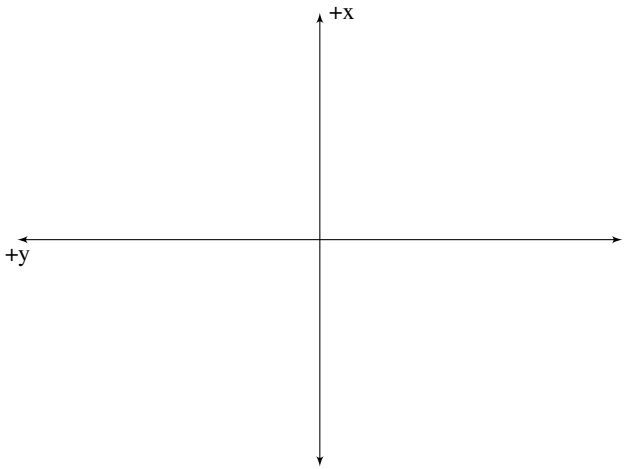


Figure 11.1: 2D projection of North-West-Up (NWU) axes convention. The positive z-axis is pointed out of the page toward the reader.

The main equation we'll need is the following.

$$\vec{v}_B = \vec{v}_A + \omega_A \times \vec{r}_{B|A}$$

where $\vec{v}_B$ is the velocity vector at point B, $\vec{v}_A$ is the velocity vector at point A, ω_A is the angular velocity vector at point A, and $\vec{r}_{B|A}$ is the distance vector from point A to point B (also described as the "distance to B relative to A").

11.5 Differential drive kinematics

A differential drive has two wheels, one on each side, separated by some distance $2r_b$. The forces they generate when moving forward are shown in figure 11.2.

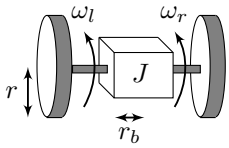


Figure 11.2: Differential drive free body diagram

11.5.1 Inverse kinematics

The mapping from v and ω to the left and right wheel velocities v_l and v_r is derived as follows. Let $\vec{v}_c$ be the velocity vector of the center of rotation, $\vec{v}_l$ be the velocity vector of the left wheel, $\vec{v}_r$ be the velocity vector of the right wheel, r_b is the distance from the center of rotation to each wheel, and ω is the counterclockwise turning rate around the center of rotation.

Once we have the vector equation representing the wheel's velocity, we'll project it onto the wheel direction vector using the dot product.

First, we'll derive v_l.

$$\begin{aligned}\vec{v}_l &= v_c\hat{i} + \omega\hat{k} \times r_b\hat{j} \\ \vec{v}_l &= v_c\hat{i} - \omega r_b\hat{i} \\ \vec{v}_l &= (v_c - \omega r_b)\hat{i}\end{aligned}$$

Now, project this vector onto the left wheel, which is pointed in the $\hat{i}$ direction.

$$v_l = (v_c - \omega r_b)\hat{i} \cdot \frac{\hat{i}}{\|\hat{i}\|}$$

The magnitude of $\hat{i}$ is 1, so the denominator cancels.

$$\begin{aligned}v_l &= (v_c - \omega r_b)\hat{i} \cdot \hat{i} \\ v_l &= v_c - \omega r_b\end{aligned} \tag{11.1}$$

Next, we'll derive v_r.

$$\begin{aligned}\vec{v}_r &= v_c\hat{i} + \omega\hat{k} \times r_b\hat{j} \\ \vec{v}_r &= v_c\hat{i} + \omega r_b\hat{i} \\ \vec{v}_r &= (v_c + \omega r_b)\hat{i}\end{aligned}$$

Now, project this vector onto the right wheel, which is pointed in the $\hat{i}$ direction.

$$v_r = (v_c + \omega r_b)\hat{i} \cdot \frac{\hat{i}}{\|\hat{i}\|}$$

The magnitude of $\hat{i}$ is 1, so the denominator cancels.

$$v_r = (v_c + \omega r_b)\hat{i} \cdot \hat{i}$$

$$v_r = v_c + \omega r_b \tag{11.2}$$

So the two inverse kinematic equations are as follows.

$$v_l = v_c - \omega r_b \tag{11.3}$$
$$v_r = v_c + \omega r_b \tag{11.4}$$

11.5.2 Forward kinematics

The mapping from the left and right wheel velocities v_l and v_r to v and ω is derived as follows.

$$\begin{aligned} v_r &= v_c + \omega r_b \\ v_c &= v_r - \omega r_b \end{aligned} \tag{11.5}$$

Substitute equation (11.5) equation for v_l.

$$\begin{aligned} v_l &= v_c - \omega r_b \\ v_l &= (v_r - \omega r_b) - \omega r_b \\ v_l &= v_r - 2\omega r_b \\ 2\omega r_b &= v_r - v_l \\ \omega &= \frac{v_r - v_l}{2r_b} \end{aligned}$$

Substitute this back into equation (11.5).

$$\begin{aligned} v_c &= v_r - \omega r_b \\ v_c &= v_r - \left(\frac{v_r - v_l}{2r_b}\right) r_b \\ v_c &= v_r - \frac{v_r - v_l}{2} \\ v_c &= v_r - \frac{v_r}{2} + \frac{v_l}{2} \\ v_c &= \frac{v_r + v_l}{2} \end{aligned}$$

So the two forward kinematic equations are as follows.

$$v_c = \frac{v_r + v_l}{2} \tag{11.6}$$
$$\omega = \frac{v_r - v_l}{2r_b} \tag{11.7}$$

11.6 Mecanum drive kinematics

A mecanum drive has four wheels, one on each corner of a rectangular chassis. The wheels have rollers offset at 45 degrees (whether it's clockwise or not varies per wheel). The forces they generate when moving forward are shown in figure 11.3.

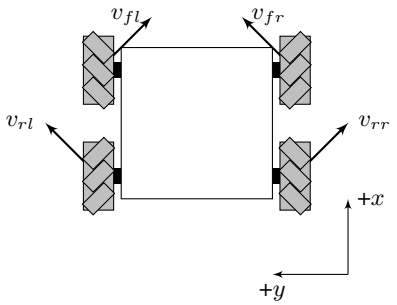


Figure 11.3: Mecanum drive free body diagram

Note that the velocity of the wheel is the same as the velocity in the diagram for the purposes of feedback control. The rollers on the wheel redirect the velocity vector.

11.6.1 Inverse kinematics

First, we'll derive the front-left wheel kinematics.

$$\begin{aligned}
\vec{v}_{fl} &= v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{fl_x}\hat{i} + r_{fl_y}\hat{j}) \\
\vec{v}_{fl} &= v_x\hat{i} + v_y\hat{j} + \omega r_{fl_x}\hat{j} - \omega r_{fl_y}\hat{i} \\
\vec{v}_{fl} &= (v_x - \omega r_{fl_y})\hat{i} + (v_y + \omega r_{fl_x})\hat{j}
\end{aligned}$$

Project the front-left wheel onto its wheel vector.

$$\begin{aligned}
v_{fl} &= ((v_x - \omega r_{fl_y})\hat{i} + (v_y + \omega r_{fl_x})\hat{j}) \cdot \frac{\hat{i} - \hat{j}}{\sqrt{2}} \\
v_{fl} &= ((v_x - \omega r_{fl_y}) - (v_y + \omega r_{fl_x}))\frac{1}{\sqrt{2}} \\
v_{fl} &= (v_x - \omega r_{fl_y} - v_y - \omega r_{fl_x})\frac{1}{\sqrt{2}} \\
v_{fl} &= (v_x - v_y - \omega r_{fl_y} - \omega r_{fl_x})\frac{1}{\sqrt{2}}
\end{aligned}$$

$$
\begin{aligned}
v_{fl} &= (v_x - v_y - \omega(r_{fl_x} + r_{fl_y}))\frac{1}{\sqrt{2}} \\
v_{fl} &= \frac{1}{\sqrt{2}}v_x - \frac{1}{\sqrt{2}}v_y - \frac{1}{\sqrt{2}}\omega(r_{fl_x} + r_{fl_y})
\end{aligned} \tag{11.8}
$$

Next, we'll derive the front-right wheel kinematics.

$$
\begin{aligned}
\vec{v}_{fr} &= v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{fr_x}\hat{i} + r_{fr_y}\hat{j}) \\
\vec{v}_{fr} &= v_x\hat{i} + v_y\hat{j} + \omega r_{fr_x}\hat{j} - \omega r_{fr_y}\hat{i} \\
\vec{v}_{fr} &= (v_x - \omega r_{fr_y})\hat{i} + (v_y + \omega r_{fr_x})\hat{j}
\end{aligned}
$$

Project the front-right wheel onto its wheel vector.

$$
\begin{aligned}
v_{fr} &= ((v_x - \omega r_{fr_y})\hat{i} + (v_y + \omega r_{fr_x})\hat{j}) \cdot (\hat{i} + \hat{j})\frac{1}{\sqrt{2}} \\
v_{fr} &= ((v_x - \omega r_{fr_y}) + (v_y + \omega r_{fr_x}))\frac{1}{\sqrt{2}} \\
v_{fr} &= (v_x - \omega r_{fr_y} + v_y + \omega r_{fr_x})\frac{1}{\sqrt{2}} \\
v_{fr} &= (v_x + v_y - \omega r_{fr_y} + \omega r_{fr_x})\frac{1}{\sqrt{2}} \\
v_{fr} &= (v_x + v_y + \omega(r_{fr_x} - r_{fr_y}))\frac{1}{\sqrt{2}} \\
v_{fr} &= \frac{1}{\sqrt{2}}v_x + \frac{1}{\sqrt{2}}v_y + \frac{1}{\sqrt{2}}\omega(r_{fr_x} - r_{fr_y})
\end{aligned} \tag{11.9}
$$

Next, we'll derive the rear-left wheel kinematics.

$$
\begin{aligned}
\vec{v}_{rl} &= v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{rl_x}\hat{i} + r_{rl_y}\hat{j}) \\
\vec{v}_{rl} &= v_x\hat{i} + v_y\hat{j} + \omega r_{rl_x}\hat{j} - \omega r_{rl_y}\hat{i} \\
\vec{v}_{rl} &= (v_x - \omega r_{rl_y})\hat{i} + (v_y + \omega r_{rl_x})\hat{j}
\end{aligned}
$$

Project the rear-left wheel onto its wheel vector.

$$
\begin{aligned}
v_{rl} &= ((v_x - \omega r_{rl_y})\hat{i} + (v_y + \omega r_{rl_x})\hat{j}) \cdot (\hat{i} + \hat{j})\frac{1}{\sqrt{2}} \\
v_{rl} &= ((v_x - \omega r_{rl_y}) + (v_y + \omega r_{rl_x}))\frac{1}{\sqrt{2}}
\end{aligned}
$$

$$
\begin{aligned}
v_{rl} &= (v_x - \omega r_{rl_y} + v_y + \omega r_{rl_x})\frac{1}{\sqrt{2}} \\
v_{rl} &= (v_x + v_y - \omega r_{rl_y} + \omega r_{rl_x})\frac{1}{\sqrt{2}} \\
v_{rl} &= (v_x + v_y + \omega(r_{rl_x} - r_{rl_y}))\frac{1}{\sqrt{2}} \\
v_{rl} &= \frac{1}{\sqrt{2}}v_x + \frac{1}{\sqrt{2}}v_y + \frac{1}{\sqrt{2}}\omega(r_{rl_x} - r_{rl_y})
\end{aligned} \tag{11.10}
$$

Next, we'll derive the rear-right wheel kinematics.

$$
\begin{aligned}
\vec{v}_{rr} &= v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{rr_x}\hat{i} + r_{rr_y}\hat{j}) \\
\vec{v}_{rr} &= v_x\hat{i} + v_y\hat{j} + \omega r_{rr_x}\hat{j} - \omega r_{rr_y}\hat{i} \\
\vec{v}_{rr} &= (v_x - \omega r_{rr_y})\hat{i} + (v_y + \omega r_{rr_x})\hat{j}
\end{aligned}
$$

Project the rear-right wheel onto its wheel vector.

$$
\begin{aligned}
v_{rr} &= ((v_x - \omega r_{rr_y})\hat{i} + (v_y + \omega r_{rr_x})\hat{j}) \cdot \frac{\hat{i} - \hat{j}}{\sqrt{2}} \\
v_{rr} &= ((v_x - \omega r_{rr_y}) - (v_y + \omega r_{rr_x}))\frac{1}{\sqrt{2}} \\
v_{rr} &= (v_x - \omega r_{rr_y} - v_y - \omega r_{rr_x})\frac{1}{\sqrt{2}} \\
v_{rr} &= (v_x - v_y - \omega r_{rr_y} - \omega r_{rr_x})\frac{1}{\sqrt{2}} \\
v_{rr} &= (v_x - v_y - \omega(r_{rr_x} + r_{rr_y}))\frac{1}{\sqrt{2}} \\
v_{rr} &= \frac{1}{\sqrt{2}}v_x - \frac{1}{\sqrt{2}}v_y - \frac{1}{\sqrt{2}}\omega(r_{rr_x} + r_{rr_y})
\end{aligned} \tag{11.11}
$$

This gives the following inverse kinematic equations.

$$
\begin{aligned}
v_{fl} &= \frac{1}{\sqrt{2}}v_x - \frac{1}{\sqrt{2}}v_y - \frac{1}{\sqrt{2}}\omega(r_{fl_x} + r_{fl_y}) \\
v_{fr} &= \frac{1}{\sqrt{2}}v_x + \frac{1}{\sqrt{2}}v_y + \frac{1}{\sqrt{2}}\omega(r_{fr_x} - r_{fr_y}) \\
v_{rl} &= \frac{1}{\sqrt{2}}v_x + \frac{1}{\sqrt{2}}v_y + \frac{1}{\sqrt{2}}\omega(r_{rl_x} - r_{rl_y})
\end{aligned}
$$

$$v_{rr} = \frac{1}{\sqrt{2}}v_x - \frac{1}{\sqrt{2}}v_y - \frac{1}{\sqrt{2}}\omega(r_{rr_x} + r_{rr_y})$$

Now we'll factor them out into matrices.

$$\begin{bmatrix} v_{fl} \\ v_{fr} \\ v_{rl} \\ v_{rr} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}(r_{fl_x} + r_{fl_y}) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}(r_{fr_x} - r_{fr_y}) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}(r_{rl_x} - r_{rl_y}) \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}(r_{rr_x} + r_{rr_y}) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

$$\begin{bmatrix} v_{fl} \\ v_{fr} \\ v_{rl} \\ v_{rr} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & -(r_{fl_x} + r_{fl_y}) \\ 1 & 1 & (r_{fr_x} - r_{fr_y}) \\ 1 & 1 & (r_{rl_x} - r_{rl_y}) \\ 1 & -1 & -(r_{rr_x} + r_{rr_y}) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{11.12}$$

11.6.2 Forward kinematics

Let $\mathbf{M}$ be the 4×3 inverse kinematics matrix above including the $\frac{1}{\sqrt{2}}$ factor. The forward kinematics are

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \mathbf{M}^+ \begin{bmatrix} v_{fl} \\ v_{fr} \\ v_{rl} \\ v_{rr} \end{bmatrix} \tag{11.13}$$

where $\mathbf{M}^+$ is the pseudoinverse of $\mathbf{M}$.

11.7 Swerve drive kinematics

A swerve drive has an arbitrary number of wheels which can rotate in place independent of the chassis. The forces they generate are shown in figure 11.4.

11.7.1 Inverse kinematics

$$\begin{aligned} \vec{v}_{wheel1} &= \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel1} \\ \vec{v}_{wheel2} &= \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel2} \\ \vec{v}_{wheel3} &= \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel3} \\ \vec{v}_{wheel4} &= \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel4} \end{aligned}$$

where $\vec{v}_{wheel}$ is the wheel velocity vector, $\vec{v}_{robot}$ is the robot velocity vector, $\vec{\omega}_{robot}$ is the robot angular velocity vector, $\vec{r}_{robot2wheel}$ is the displacement vector from the

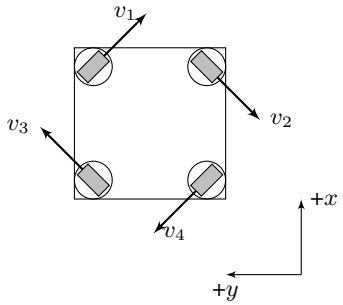


Figure 11.4: Swerve drive free body diagram

robot's center of rotation to the wheel,[1] $\vec{v}_{robot} = v_x\hat{i} + v_y\hat{j}$, and $\vec{r}_{robot2wheel} = r_x\hat{i} + r_y\hat{j}$. The number suffixes denote a specific wheel in figure 11.4.

$$\vec{v}_1 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{1x}\hat{i} + r_{1y}\hat{j})$$
$$\vec{v}_2 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{2x}\hat{i} + r_{2y}\hat{j})$$
$$\vec{v}_3 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{3x}\hat{i} + r_{3y}\hat{j})$$
$$\vec{v}_4 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{4x}\hat{i} + r_{4y}\hat{j})$$

$$\vec{v}_1 = v_x\hat{i} + v_y\hat{j} + (\omega r_{1x}\hat{j} - \omega r_{1y}\hat{i})$$
$$\vec{v}_2 = v_x\hat{i} + v_y\hat{j} + (\omega r_{2x}\hat{j} - \omega r_{2y}\hat{i})$$
$$\vec{v}_3 = v_x\hat{i} + v_y\hat{j} + (\omega r_{3x}\hat{j} - \omega r_{3y}\hat{i})$$
$$\vec{v}_4 = v_x\hat{i} + v_y\hat{j} + (\omega r_{4x}\hat{j} - \omega r_{4y}\hat{i})$$

$$\vec{v}_1 = v_x\hat{i} + v_y\hat{j} + \omega r_{1x}\hat{j} - \omega r_{1y}\hat{i}$$
$$\vec{v}_2 = v_x\hat{i} + v_y\hat{j} + \omega r_{2x}\hat{j} - \omega r_{2y}\hat{i}$$
$$\vec{v}_3 = v_x\hat{i} + v_y\hat{j} + \omega r_{3x}\hat{j} - \omega r_{3y}\hat{i}$$
$$\vec{v}_4 = v_x\hat{i} + v_y\hat{j} + \omega r_{4x}\hat{j} - \omega r_{4y}\hat{i}$$

$$\vec{v}_1 = v_x\hat{i} - \omega r_{1y}\hat{i} + v_y\hat{j} + \omega r_{1x}\hat{j}$$
$$\vec{v}_2 = v_x\hat{i} - \omega r_{2y}\hat{i} + v_y\hat{j} + \omega r_{2x}\hat{j}$$

[1]The robot's center of rotation need not coincide with the robot's geometric center.

$$\vec{v}_3 = v_x\hat{i} - \omega r_{3y}\hat{i} + v_y\hat{j} + \omega r_{3x}\hat{j}$$
$$\vec{v}_4 = v_x\hat{i} - \omega r_{4y}\hat{i} + v_y\hat{j} + \omega r_{4x}\hat{j}$$

$$\vec{v}_1 = (v_x - \omega r_{1y})\hat{i} + (v_y + \omega r_{1x})\hat{j}$$
$$\vec{v}_2 = (v_x - \omega r_{2y})\hat{i} + (v_y + \omega r_{2x})\hat{j}$$
$$\vec{v}_3 = (v_x - \omega r_{3y})\hat{i} + (v_y + \omega r_{3x})\hat{j}$$
$$\vec{v}_4 = (v_x - \omega r_{4y})\hat{i} + (v_y + \omega r_{4x})\hat{j}$$

Separate the i-hat components into independent equations.

$$v_{1x} = v_x - \omega r_{1y}$$
$$v_{2x} = v_x - \omega r_{2y}$$
$$v_{3x} = v_x - \omega r_{3y}$$
$$v_{4x} = v_x - \omega r_{4y}$$

Separate the j-hat components into independent equations.

$$v_{1y} = v_y + \omega r_{1x}$$
$$v_{2y} = v_y + \omega r_{2x}$$
$$v_{3y} = v_y + \omega r_{3x}$$
$$v_{4y} = v_y + \omega r_{4x}$$

Now we'll factor them out into matrices.

$$\begin{bmatrix} v_{1x} \\ v_{2x} \\ v_{3x} \\ v_{4x} \\ v_{1y} \\ v_{2y} \\ v_{3y} \\ v_{4y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_{1y} \\ 1 & 0 & -r_{2y} \\ 1 & 0 & -r_{3y} \\ 1 & 0 & -r_{4y} \\ 0 & 1 & r_{1x} \\ 0 & 1 & r_{2x} \\ 0 & 1 & r_{3x} \\ 0 & 1 & r_{4x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Rearrange the rows so the x and y components are in adjacent rows.

$$\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \\ v_{4x} \\ v_{4y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_{1y} \\ 0 & 1 & r_{1x} \\ 1 & 0 & -r_{2y} \\ 0 & 1 & r_{2x} \\ 1 & 0 & -r_{3y} \\ 0 & 1 & r_{3x} \\ 1 & 0 & -r_{4y} \\ 0 & 1 & r_{4x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{11.14}$$

To convert the swerve module x and y velocity components to a velocity and heading, use the Pythagorean theorem and arctangent respectively. Here's an example for module 1.

$$v_1 = \sqrt{v_{1x}^2 + v_{1y}^2} \tag{11.15}$$

$$\theta_1 = \tan^{-1}\left(\frac{v_{1y}}{v_{1x}}\right) \tag{11.16}$$

11.7.2 Forward kinematics

Let $\mathbf{M}$ be the 8×3 inverse kinematics matrix from equation (11.14). The forward kinematics are

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \mathbf{M}^+ \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \\ v_{4x} \\ v_{4y} \end{bmatrix} \tag{11.17}$$

where $\mathbf{M}^+$ is the pseudoinverse of $\mathbf{M}$.

This page intentionally left blank

Hills by northbound freeway between Santa Maria and Ventura

12. Newtonian mechanics examples

A model is a set of differential equations describing how the system behaves over time. There are two common approaches for developing them.

1. Collecting data on the physical system's behavior and performing system identification with it.
2. Using physics to derive the system's model from first principles.

This chapter covers the second approach using Newtonian mechanics.

The models derived here should cover most types of motion seen on an FRC robot. Furthermore, they can be easily tweaked to describe many types of mechanisms just by pattern-matching. There's only so many ways to hook up a mass to a motor in FRC. The flywheel model can be used for spinning mechanisms, the elevator model can be used for spinning mechanisms transformed to linear motion, and the single-jointed arm model can be used for rotating servo mechanisms (it's just the flywheel model augmented with a position state).

These models assume all motor controllers driving DC motors are set to brake mode instead of coast mode. Brake mode behaves the same as coast mode except where the applied voltage is zero. In brake mode, the motor leads are shorted together to prevent movement. In coast mode, the motor leads are an open circuit.

12.1 DC motor

We will be deriving a first-order model for a DC motor. A second-order model would include the inductance of the motor windings as well, but we're assuming the time constant of the inductor is small enough that its affect on the model behavior is negligible for FRC use cases (see subsection 6.10.7 for a demonstration of this for a real DC motor).

R For the brushless motor commutation methods currently available in FRC (trapezoidal commutation, field-oriented control), brushed and brushless DC motors have the same dynamics. However, more advanced commutation methods can break the linear back-EMF assumption of the brushed motor model.

The first-order model will only require numbers from the motor's datasheet. The second-order model would require measuring the motor inductance as well, which generally isn't in the datasheet. It can be difficult to measure accurately without the right equipment.

12.1.1 Equations of motion

The circuit for a DC motor is shown in figure 12.1.

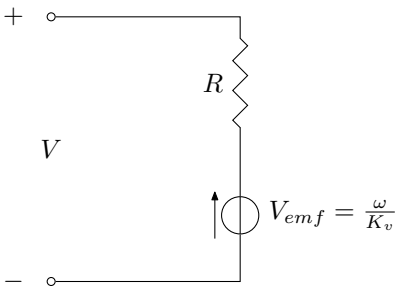


Figure 12.1: DC motor circuit

V is the voltage applied to the motor, I is the current through the motor in Amps, R is the resistance across the motor in Ohms, ω is the angular velocity of the motor in radians per second, and K_v is the angular velocity constant in radians per second per Volt. This circuit reflects the following relation.

$$V = IR + \frac{\omega}{K_v} \tag{12.1}$$

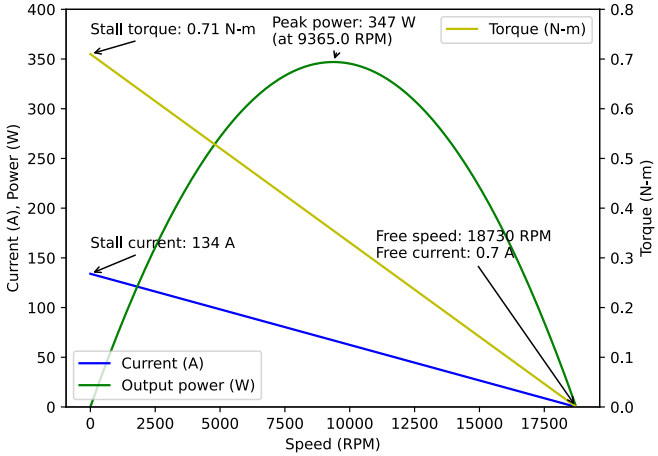


Figure 12.2: Example motor datasheet for 775pro

The mechanical relation for a DC motor is

$$\tau = K_t I \tag{12.2}$$

where τ is the torque produced by the motor in Newton-meters and K_t is the torque constant in Newton-meters per Amp. Therefore

$$I = \frac{\tau}{K_t}$$

Substitute this into equation (12.1).

$$V = \frac{\tau}{K_t} R + \frac{\omega}{K_v} \tag{12.3}$$

12.1.2 Calculating constants

A typical motor's datasheet should include graphs of the motor's measured torque and current for different angular velocities for a given voltage applied to the motor. Figure 12.2 is an example. An FRC motor's datasheet can be found on its vendor's website.

Torque constant K_t

$$\tau = K_t I$$

$$K_t = \frac{\tau}{I}$$
$$K_t = \frac{\tau_{stall}}{I_{stall}} \tag{12.4}$$

where τ_{stall} is the stall torque and I_{stall} is the stall current of the motor from its datasheet.

Resistance R

Recall equation (12.1).

$$V = IR + \frac{\omega}{K_v}$$

When the motor is stalled, $\omega = 0$.

$$V = I_{stall}R$$
$$R = \frac{V}{I_{stall}} \tag{12.5}$$

where I_{stall} is the stall current of the motor and V is the voltage applied to the motor at stall.

Angular velocity constant K_v

Recall equation (12.1).

$$V = IR + \frac{\omega}{K_v}$$
$$V - IR = \frac{\omega}{K_v}$$
$$K_v = \frac{\omega}{V - IR}$$

When the motor is spinning under no load,

$$K_v = \frac{\omega_{free}}{V - I_{free}R} \tag{12.6}$$

where ω_{free} is the angular velocity of the motor under no load (also known as the free speed), and V is the voltage applied to the motor when it's spinning at ω_{free}, and I_{free} is the current drawn by the motor under no load.

To model a mechanism with several identical motors in one gearbox, multiply the stall torque, stall current, and free current by the number of motors N. K_t and K_v will be the same because N cancels out, but R will be divided by N. This multiplies the acceleration contribution of each model term by N.

12.1.3 Current limiting

Current limiting of a DC motor reduces the maximum input voltage to avoid exceeding a current threshold. Predictive current limiting uses a projected estimate of the current, so the voltage is reduced before the current threshold is exceeded. Reactive current limiting uses an actual current measurement, so the voltage is reduced after the current threshold is exceeded.

The following pseudocode demonstrates each type of current limiting.

```
# Normal feedback control
V = K @ (r - x)

# Calculations for predictive current limiting
omega = angular_velocity_measurement
I = V / R - omega / (Kv * R)

# Calculations for reactive current limiting
I = current_measurement
omega = Kv * V - I * R * Kv  # or can be angular velocity measurement

# If predicted/actual current above max, limit current by reducing voltage
if I > I_max:
    V = I_max * R + omega / Kv
```

Snippet 12.1. Limits current of DC motor to I_{max}

12.2 Elevator

This elevator consists of a DC motor attached to a pulley that drives a mass up or down.

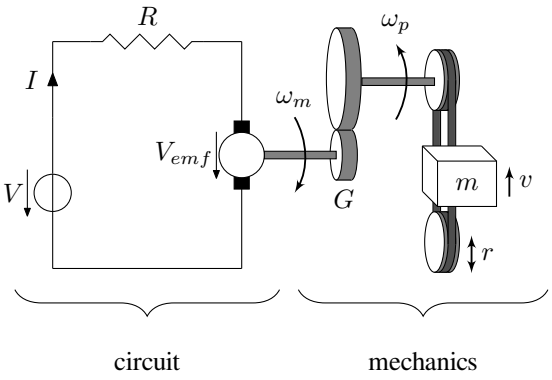


Figure 12.3: Elevator system diagram

Gear ratios are written as output over input, so G is greater than one in figure 12.3.

12.2.1 Equations of motion

We want to derive an equation for the carriage acceleration a (derivative of v) given an input voltage V, which we can integrate to get carriage velocity and position.

First, we'll find a torque to substitute into the equation for a DC motor. Based on figure 12.3

$$\tau_m G = \tau_p \tag{12.7}$$

where G is the gear ratio between the motor and the pulley and τ_p is the torque produced by the pulley.

$$rF_m = \tau_p \tag{12.8}$$

where r is the radius of the pulley. Substitute equation (12.7) into τ_m in the DC motor equation (12.3).

$$V = \frac{\frac{\tau_p}{G}}{K_t} R + \frac{\omega_m}{K_v}$$
$$V = \frac{\tau_p}{GK_t} R + \frac{\omega_m}{K_v}$$

Substitute in equation (12.8) for τ_p.

$$V = \frac{rF_m}{GK_t} R + \frac{\omega_m}{K_v} \tag{12.9}$$

The angular velocity of the motor armature ω_m is

$$\omega_m = G\omega_p \tag{12.10}$$

where ω_p is the angular velocity of the pulley. The velocity of the mass (the elevator carriage) is

$$v = r\omega_p$$
$$\omega_p = \frac{v}{r} \tag{12.11}$$

Substitute equation (12.11) into equation (12.10).

$$\omega_m = G\frac{v}{r} \tag{12.12}$$

Substitute equation (12.12) into equation (12.9) for ω_m.

$$V = \frac{rF_m}{GK_t}R + \frac{G\frac{v}{r}}{K_v}$$
$$V = \frac{RrF_m}{GK_t} + \frac{G}{rK_v}v$$

Solve for F_m.

$$\frac{RrF_m}{GK_t} = V - \frac{G}{rK_v}v$$
$$F_m = \left(V - \frac{G}{rK_v}v\right)\frac{GK_t}{Rr}$$
$$F_m = \frac{GK_t}{Rr}V - \frac{G^2K_t}{Rr^2K_v}v \tag{12.13}$$

We need to find the acceleration of the elevator carriage. Note that

$$\sum F = ma \tag{12.14}$$

where $\sum F$ is the sum of forces applied to the elevator carriage, m is the mass of the elevator carriage in kilograms, and a is the acceleration of the elevator carriage.

$$ma = F_m$$
$$ma = \left(\frac{GK_t}{Rr}V - \frac{G^2K_t}{Rr^2K_v}v\right)$$
$$a = \frac{GK_t}{Rrm}V - \frac{G^2K_t}{Rr^2mK_v}v$$
$$a = -\frac{G^2K_t}{Rr^2mK_v}v + \frac{GK_t}{Rrm}V \tag{12.15}$$

R Gravity is not part of the modeled dynamics because it complicates the state-space model and the controller will behave well enough without it.

This model will be converted to state-space notation in section 6.9.

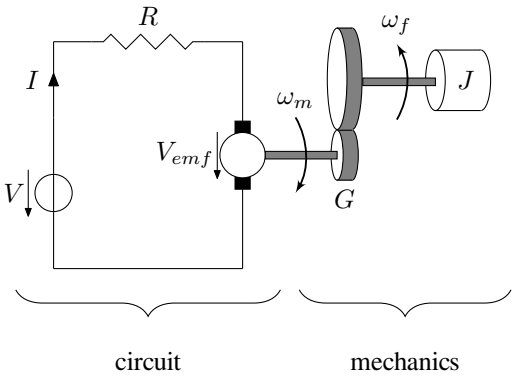


Figure 12.4: Flywheel system diagram

12.3 Flywheel

This flywheel consists of a DC motor attached to a spinning mass of non-negligible moment of inertia.

Gear ratios are written as output over input, so G is greater than one in figure 12.4.

12.3.1 Equations of motion

We want to derive an equation for the flywheel angular acceleration $\dot{\omega}_f$ given an input voltage V, which we can integrate to get flywheel angular velocity.

We will start with the equation derived earlier for a DC motor, equation (12.3).

$$V = \frac{\tau_m}{K_t} R + \frac{\omega_m}{K_v}$$

Solve for the angular acceleration. First, we'll rearrange the terms because from inspection, V is the model input, ω_m is the state, and τ_m contains the angular acceleration.

$$V = \frac{R}{K_t} \tau_m + \frac{1}{K_v} \omega_m$$

Solve for τ_m.

$$\begin{aligned} V &= \frac{R}{K_t} \tau_m + \frac{1}{K_v} \omega_m \\ \frac{R}{K_t} \tau_m &= V - \frac{1}{K_v} \omega_m \end{aligned}$$

$$\tau_m = \frac{K_t}{R}V - \frac{K_t}{K_v R}\omega_m$$

Since $\tau_m G = \tau_f$ and $\omega_m = G\omega_f$,

$$\begin{aligned}\left(\frac{\tau_f}{G}\right) &= \frac{K_t}{R}V - \frac{K_t}{K_v R}(G\omega_f)\\ \frac{\tau_f}{G} &= \frac{K_t}{R}V - \frac{GK_t}{K_v R}\omega_f\\ \tau_f &= \frac{GK_t}{R}V - \frac{G^2K_t}{K_v R}\omega_f\end{aligned} \tag{12.16}$$

The torque applied to the flywheel is defined as

$$\tau_f = J\dot{\omega}_f \tag{12.17}$$

where J is the moment of inertia of the flywheel and $\dot{\omega}_f$ is the angular acceleration. Substitute equation (12.17) into equation (12.16).

$$\begin{aligned}(J\dot{\omega}_f) &= \frac{GK_t}{R}V - \frac{G^2K_t}{K_v R}\omega_f\\ \dot{\omega}_f &= \frac{GK_t}{RJ}V - \frac{G^2K_t}{K_v RJ}\omega_f\\ \dot{\omega}_f &= -\frac{G^2K_t}{K_v RJ}\omega_f + \frac{GK_t}{RJ}V\end{aligned}$$

We'll relabel ω_f as ω for convenience.

$$\dot{\omega} = -\frac{G^2K_t}{K_v RJ}\omega + \frac{GK_t}{RJ}V \tag{12.18}$$

This model will be converted to state-space notation in section 6.10.

12.3.2 Calculating constants

Moment of inertia J

Given the simplicity of this mechanism, it may be easier to compute this value theoretically using material properties in CAD. A procedure for measuring it experimentally is presented below.

First, rearrange equation (12.18) into the form $y = mx + b$ such that J is in the numerator of m.

$$\dot{\omega} = -\frac{G^2K_t}{K_v RJ}\omega + \frac{GK_t}{RJ}V$$

$$J\dot{\omega} = -\frac{G^2 K_t}{K_v R}\omega + \frac{GK_t}{R}V$$

Multiply by $\frac{K_v R}{G^2 K_t}$ on both sides.

$$\begin{aligned}
\frac{JK_v R}{G^2 K_t}\dot{\omega} &= -\omega + \frac{GK_t}{R} \cdot \frac{K_v R}{G^2 K_t}V \\
\frac{JK_v R}{G^2 K_t}\dot{\omega} &= -\omega + \frac{K_v}{G}V \\
\omega &= -\frac{JK_v R}{G^2 K_t}\dot{\omega} + \frac{K_v}{G}V
\end{aligned} \tag{12.19}$$

The test procedure is as follows.

1. Run the flywheel forward at a constant voltage. Record the angular velocity over time.
2. Compute the angular acceleration from the angular velocity data as the difference between each sample divided by the time between them.
3. Perform a linear regression of angular velocity versus angular acceleration. The slope of this line has the form $-\frac{JK_v R}{G^2 K_t}$ as per equation (12.19).
4. Multiply the slope by $-\frac{G^2 K_t}{K_v R}$ to obtain a least squares estimate of J.

12.4 Single-jointed arm

This single-jointed arm consists of a DC motor attached to a pulley that spins a straight bar in pitch.

Gear ratios are written as output over input, so G is greater than one in figure 12.5.

12.4.1 Equations of motion

We want to derive an equation for the arm angular acceleration $\dot{\omega}_{arm}$ given an input voltage V, which we can integrate to get arm angular velocity and angle.

We will start with the equation derived earlier for a DC motor, equation (12.3).

$$V = \frac{\tau_m}{K_t}R + \frac{\omega_m}{K_v}$$

Solve for the angular acceleration. First, we'll rearrange the terms because from inspection, V is the model input, ω_m is the state, and τ_m contains the angular acceleration.

$$V = \frac{R}{K_t}\tau_m + \frac{1}{K_v}\omega_m$$

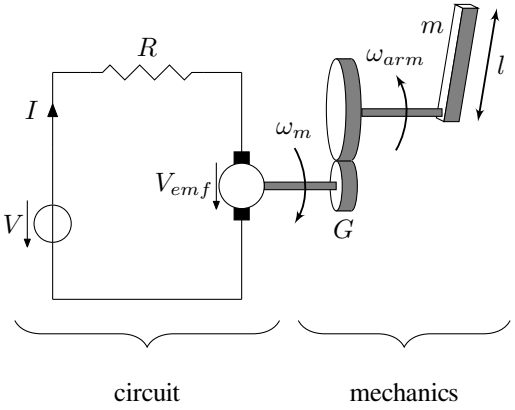


Figure 12.5: Single-jointed arm system diagram

Solve for τ_m.

$$\begin{aligned} V &= \frac{R}{K_t}\tau_m + \frac{1}{K_v}\omega_m \\ \frac{R}{K_t}\tau_m &= V - \frac{1}{K_v}\omega_m \\ \tau_m &= \frac{K_t}{R}V - \frac{K_t}{K_v R}\omega_m \end{aligned} \tag{12.20}$$

Since $\tau_m G = \tau_{arm}$ and $\omega_m = G\omega_{arm}$,

$$\begin{aligned} \left(\frac{\tau_{arm}}{G}\right) &= \frac{K_t}{R}V - \frac{K_t}{K_v R}(G\omega_{arm}) \\ \frac{\tau_{arm}}{G} &= \frac{K_t}{R}V - \frac{GK_t}{K_v R}\omega_{arm} \\ \tau_{arm} &= \frac{GK_t}{R}V - \frac{G^2K_t}{K_v R}\omega_{arm} \end{aligned} \tag{12.21}$$

The torque applied to the arm is defined as

$$\tau_{arm} = J\dot{\omega}_{arm} \tag{12.22}$$

where J is the moment of inertia of the arm and $\dot{\omega}_{arm}$ is the angular acceleration. Substitute equation (12.22) into equation (12.21).

$$(J\dot{\omega}_{arm}) = \frac{GK_t}{R}V - \frac{G^2K_t}{K_v R}\omega_{arm}$$

$$\dot{\omega}_{arm} = -\frac{G^2 K_t}{K_v R J}\omega_{arm} + \frac{G K_t}{R J}V$$

We'll relabel ω_{arm} as ω for convenience.

$$\dot{\omega} = -\frac{G^2 K_t}{K_v R J}\omega + \frac{G K_t}{R J}V \tag{12.23}$$

This model will be converted to state-space notation in section 6.11.

12.4.2 Calculating constants

Moment of inertia J

Given the simplicity of this mechanism, it may be easier to compute this value theoretically using material properties in CAD. J can also be approximated as the moment of inertia of a thin rod rotating around one end. Therefore

$$J = \frac{1}{3}ml^2 \tag{12.24}$$

where m is the mass of the arm and l is the length of the arm. Otherwise, a procedure for measuring it experimentally is presented below.

First, rearrange equation (12.23) into the form $y = mx + b$ such that J is in the numerator of m.

$$\dot{\omega} = -\frac{G^2 K_t}{K_v R J}\omega + \frac{G K_t}{R J}V$$
$$J\dot{\omega} = -\frac{G^2 K_t}{K_v R}\omega + \frac{G K_t}{R}V$$

Multiply by $\frac{K_v R}{G^2 K_t}$ on both sides.

$$\frac{J K_v R}{G^2 K_t}\dot{\omega} = -\omega + \frac{G K_t}{R} \cdot \frac{K_v R}{G^2 K_t}V$$
$$\frac{J K_v R}{G^2 K_t}\dot{\omega} = -\omega + \frac{K_v}{G}V$$
$$\omega = -\frac{J K_v R}{G^2 K_t}\dot{\omega} + \frac{K_v}{G}V \tag{12.25}$$

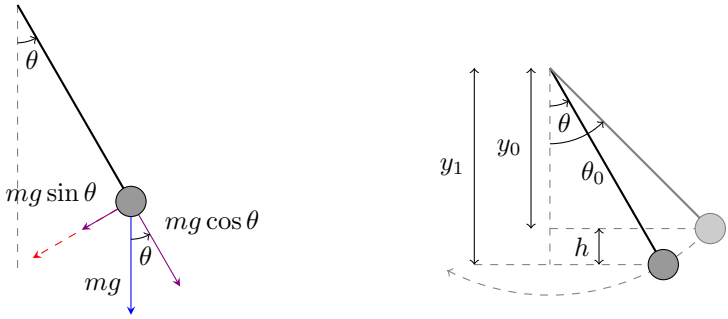
The test procedure is as follows.

1. Orient the arm such that its axis of rotation is aligned with gravity (i.e., the arm is on its side). This avoids gravity affecting the measurements.

2. Run the arm forward at a constant voltage. Record the angular velocity over time.
3. Compute the angular acceleration from the angular velocity data as the difference between each sample divided by the time between them.
4. Perform a linear regression of angular velocity versus angular acceleration. The slope of this line has the form $-\frac{JK_vR}{G^2K_t}$ as per equation (12.25).
5. Multiply the slope by $-\frac{G^2K_t}{K_vR}$ to obtain a least squares estimate of J.

12.5 Pendulum

Kinematics and dynamics are a rather large topics, so for now, we'll just focus on the basics required for working with the models in this book. We'll derive the same model, a pendulum, using three approaches: sum of forces, sum of torques, and conservation of energy.



(a) Force diagram of a pendulum

(b) Trigonometry of a pendulum

Figure 12.6: Pendulum force diagrams

12.5.1 Force derivation

Consider figure 12.6a, which shows the forces acting on a pendulum.

Note that the path of the pendulum sweeps out an arc of a circle. The angle θ is measured in radians. The blue arrow is the gravitational force acting on the bob, and the violet arrows are that same force resolved into components parallel and perpendicular to the bob's instantaneous motion. The direction of the bob's instantaneous velocity always points along the red axis, which is considered the tangential axis because its direction is always tangent to the circle. Consider Newton's second law

$$F = ma$$

where F is the sum of forces on the object, m is mass, and a is the acceleration. Because we are only concerned with changes in speed, and because the bob is forced to stay in a circular path, we apply Newton's equation to the tangential axis only. The short violet arrow represents the component of the gravitational force in the tangential axis, and trigonometry can be used to determine its magnitude. Therefore

$$\begin{aligned} -mg\sin\theta &= ma \\ a &= -g\sin\theta \end{aligned}$$

where g is the acceleration due to gravity near the surface of the earth. The negative sign on the right hand side implies that θ and a always point in opposite directions. This makes sense because when a pendulum swings further to the left, we would expect it to accelerate back toward the right.

This linear acceleration a along the red axis can be related to the change in angle θ by the arc length formulas; s is arc length and l is the length of the pendulum.

$$\begin{aligned} s &= l\theta \\ v &= \frac{ds}{dt} = l\frac{d\theta}{dt} \\ a &= \frac{d^2s}{dt^2} = l\frac{d^2\theta}{dt^2} \end{aligned} \tag{12.26}$$

Therefore,

$$\begin{aligned} l\frac{d^2\theta}{dt^2} &= -g\sin\theta \\ \frac{d^2\theta}{dt^2} &= -\frac{g}{l}\sin\theta \\ \ddot{\theta} &= -\frac{g}{l}\sin\theta \end{aligned}$$

12.5.2 Torque derivation

The equation can be obtained using two definitions for torque.

$$\tau = \mathbf{r} \times \mathbf{F}$$

First start by defining the torque on the pendulum bob using the force due to gravity.

$$\tau = \mathbf{l} \times \mathbf{F}_g$$

where $\mathbf{l}$ is the length vector of the pendulum and $\mathbf{F}_g$ is the force due to gravity.

For now just consider the magnitude of the torque on the pendulum.

$$|\tau| = -mgl \sin\theta$$

where m is the mass of the pendulum, g is the acceleration due to gravity, l is the length of the pendulum and θ is the angle between the length vector and the force due to gravity.

Next rewrite the angular momentum.

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} = m\mathbf{r} \times (\omega \times \mathbf{r})$$

Again just consider the magnitude of the angular momentum.

$$\begin{aligned} |\mathbf{L}| &= mr^2\omega \\ |\mathbf{L}| &= ml^2 \frac{d\theta}{dt} \\ \frac{d}{dt}|\mathbf{L}| &= ml^2 \frac{d^2\theta}{dt^2} \end{aligned}$$

According to $\tau = \frac{d\mathbf{L}}{dt}$, we can just compare the magnitudes.

$$\begin{aligned} -mgl \sin\theta &= ml^2 \frac{d^2\theta}{dt^2} \\ -\frac{g}{l} \sin\theta &= \frac{d^2\theta}{dt^2} \\ \ddot{\theta} &= -\frac{g}{l} \sin\theta \end{aligned}$$

which is the same result from force analysis.

12.5.3 Energy derivation

The equation can also be obtained via the conservation of mechanical energy principle: any object falling a vertical distance h would acquire kinetic energy equal to that which it lost to the fall. In other words, gravitational potential energy is converted into kinetic energy. Change in potential energy is given by

$$\Delta U = mgh$$

The change in kinetic energy (body started from rest) is given by

$$\Delta K = \frac{1}{2}mv^2$$

Since no energy is lost, the gain in one must be equal to the loss in the other

$$\frac{1}{2}mv^2 = mgh$$

The change in velocity for a given change in height can be expressed as

$$v = \sqrt{2gh}$$

Using equation (12.26), this equation can be rewritten in terms of $\frac{d\theta}{dt}$.

$$\begin{aligned} v = l\frac{d\theta}{dt} &= \sqrt{2gh} \\ \frac{d\theta}{dt} &= \frac{2gh}{l} \end{aligned} \tag{12.27}$$

where h is the vertical distance the pendulum fell. Look at figure 12.6b, which presents the trigonometry of a pendulum. If the pendulum starts its swing from some initial angle θ_0, then y_0, the vertical distance from the pivot point, is given by

$$y_0 = l\cos\theta_0$$

Similarly for y_1, we have

$$y_1 = l\cos\theta$$

Then h is the difference of the two

$$h = l(\cos\theta - \cos\theta_0)$$

Substituting this into equation (12.27) gives

$$\frac{d\theta}{dt} = \sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}$$

This equation is known as the first integral of motion. It gives the velocity in terms of the location and includes an integration constant related to the initial displacement (θ_0). We can differentiate by applying the chain rule with respect to time. Doing so gives the acceleration.

$$\begin{aligned} \frac{d}{dt}\frac{d\theta}{dt} &= \frac{d}{dt}\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)} \\ \frac{d^2\theta}{dt^2} &= \frac{1}{2}\frac{-\frac{2g}{l}\sin\theta}{\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}}\frac{d\theta}{dt} \end{aligned}$$

$$\frac{d^2\theta}{dt^2} = \frac{1}{2}\frac{-\frac{2g}{l}\sin\theta}{\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}}\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}$$
$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta$$
$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

which is the same result from force analysis.

12.6 Differential drive

This drivetrain consists of two DC motors per side which are chained together on their respective sides and drive wheels which are assumed to be massless.

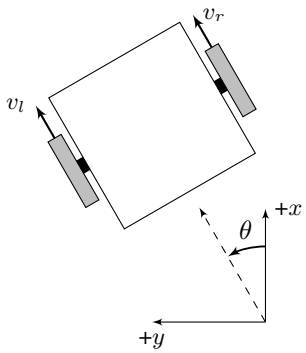


Figure 12.7: Differential drive system diagram

12.6.1 Equations of motion

We want to derive equations for the accelerations of the left and right sides of the robot $\dot{v}_l$ and $\dot{v}_r$ given left and right input voltages V_l and V_r.

From equation (12.16) of the flywheel model derivations

$$\tau = \frac{GK_t}{R}V - \frac{G^2K_t}{K_vR}\omega \tag{12.28}$$

where τ is the torque applied by one wheel of the differential drive, G is the gear ratio of the differential drive, K_t is the torque constant of the motor, R is the resistance of

the motor, and K_v is the angular velocity constant. Since $\tau = rF$ and $\omega = \frac{v}{r}$ where v is the velocity of a given drive side along the ground and r is the drive wheel radius

$$\begin{aligned}
(rF) &= \frac{GK_t}{R}V - \frac{G^2K_t}{K_vR}\left(\frac{v}{r}\right) \\
rF &= \frac{GK_t}{R}V - \frac{G^2K_t}{K_vRr}v \\
F &= \frac{GK_t}{Rr}V - \frac{G^2K_t}{K_vRr^2}v \\
F &= -\frac{G^2K_t}{K_vRr^2}v + \frac{GK_t}{Rr}V
\end{aligned}$$

Therefore, for each side of the robot,

$$\begin{aligned}
F_l &= -\frac{G_l^2K_t}{K_vRr^2}v_l + \frac{G_lK_t}{Rr}V_l \\
F_r &= -\frac{G_r^2K_t}{K_vRr^2}v_r + \frac{G_rK_t}{Rr}V_r
\end{aligned}$$

where the l and r subscripts denote the side of the robot to which each variable corresponds.

Let $C_1 = -\frac{G_l^2K_t}{K_vRr^2}$, $C_2 = \frac{G_lK_t}{Rr}$, $C_3 = -\frac{G_r^2K_t}{K_vRr^2}$, and $C_4 = \frac{G_rK_t}{Rr}$.

$$F_l = C_1v_l + C_2V_l \tag{12.29}$$
$$F_r = C_3v_r + C_4V_r \tag{12.30}$$

First, find the sum of forces.

$$\begin{aligned}
\sum F &= ma \\
F_l + F_r &= m\dot{v} \\
F_l + F_r &= m\frac{\dot{v}_l + \dot{v}_r}{2} \\
\frac{2}{m}(F_l + F_r) &= \dot{v}_l + \dot{v}_r \\
\dot{v}_l &= \frac{2}{m}(F_l + F_r) - \dot{v}_r
\end{aligned} \tag{12.31}$$

Next, find the sum of torques.

$$\sum \tau = J\dot{\omega}$$

$$\tau_l + \tau_r = J\left(\frac{\dot{v}_r - \dot{v}_l}{2r_b}\right)$$

where r_b is the radius of the differential drive.

$$\begin{aligned}
(-r_b F_l) + (r_b F_r) &= J\frac{\dot{v}_r - \dot{v}_l}{2r_b} \\
-r_b F_l + r_b F_r &= \frac{J}{2r_b}(\dot{v}_r - \dot{v}_l) \\
-F_l + F_r &= \frac{J}{2r_b^2}(\dot{v}_r - \dot{v}_l) \\
\frac{2r_b^2}{J}(-F_l + F_r) &= \dot{v}_r - \dot{v}_l \\
\dot{v}_r &= \dot{v}_l + \frac{2r_b^2}{J}(-F_l + F_r)
\end{aligned}$$

Substitute in equation (12.31) for $\dot{v}_l$ to obtain an expression for $\dot{v}_r$.

$$\begin{aligned}
\dot{v}_r &= \left(\frac{2}{m}(F_l + F_r) - \dot{v}_r\right) + \frac{2r_b^2}{J}(-F_l + F_r) \\
2\dot{v}_r &= \frac{2}{m}(F_l + F_r) + \frac{2r_b^2}{J}(-F_l + F_r) \\
\dot{v}_r &= \frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(-F_l + F_r) \qquad (12.32) \\
\dot{v}_r &= \frac{1}{m}F_l + \frac{1}{m}F_r - \frac{r_b^2}{J}F_l + \frac{r_b^2}{J}F_r \\
\dot{v}_r &= \left(\frac{1}{m} - \frac{r_b^2}{J}\right)F_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)F_r \qquad (12.33)
\end{aligned}$$

Substitute equation (12.32) back into equation (12.31) to obtain an expression for $\dot{v}_l$.

$$\begin{aligned}
\dot{v}_l &= \frac{2}{m}(F_l + F_r) - \left(\frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(-F_l + F_r)\right) \\
\dot{v}_l &= \frac{1}{m}(F_l + F_r) - \frac{r_b^2}{J}(-F_l + F_r) \\
\dot{v}_l &= \frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(F_l - F_r) \\
\dot{v}_l &= \frac{1}{m}F_l + \frac{1}{m}F_r + \frac{r_b^2}{J}F_l - \frac{r_b^2}{J}F_r
\end{aligned}$$

$$\dot{v}_l = \left(\frac{1}{m} + \frac{r_b^2}{J}\right) F_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) F_r \tag{12.34}$$

Now, plug the expressions for F_l and F_r into equation (12.33).

$$\dot{v}_r = \left(\frac{1}{m} - \frac{r_b^2}{J}\right) F_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) F_r$$
$$\dot{v}_r = \left(\frac{1}{m} - \frac{r_b^2}{J}\right) (C_1 v_l + C_2 V_l) + \left(\frac{1}{m} + \frac{r_b^2}{J}\right) (C_3 v_r + C_4 V_r) \tag{12.35}$$

Now, plug the expressions for F_l and F_r into equation (12.34).

$$\dot{v}_l = \left(\frac{1}{m} + \frac{r_b^2}{J}\right) F_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) F_r$$
$$\dot{v}_l = \left(\frac{1}{m} + \frac{r_b^2}{J}\right) (C_1 v_l + C_2 V_l) + \left(\frac{1}{m} - \frac{r_b^2}{J}\right) (C_3 v_r + C_4 V_r) \tag{12.36}$$

This model will be converted to state-space notation in section 8.7.

12.6.2 Calculating constants

Moment of inertia J

We'll use empirical measurements of linear and angular velocity to determine J. First, we'll derive the equation required to perform a linear regression using velocity test data.

$$\tau_1 = \mathbf{r} \times \mathbf{F}$$
$$\tau_1 = rma$$

where τ_1 is the torque applied by a drive motor during only linear acceleration, r is the wheel radius, m is the robot mass, and a is the linear acceleration.

$$\tau_2 = I\alpha$$

where τ_2 is the torque applied by a drive motor during only angular acceleration, I is the moment of inertia (same as J), and α is the angular acceleration. If a constant voltage is applied during both the linear and angular acceleration tests, $\tau_1 = \tau_2$. Therefore,

$$rma = I\alpha$$

Integrate with respect to time.

$$rmv + C_1 = I\omega + C_2$$

$$\begin{aligned} rmv &= I\omega + C_3 \\ v &= \frac{I}{rm}\omega + C_3 \end{aligned} \tag{12.37}$$

where v is linear velocity and ω is angular velocity. C_1, C_2, and C_3 are arbitrary constants of integration that won't be needed. The test procedure is as follows.

1. Run the drivetrain forward at a constant voltage. Record the linear velocity over time using encoders.
2. Rotate the drivetrain around its center by applying the same voltage as the linear acceleration test with the motors driving in opposite directions. Record the angular velocity over time using a gyroscope.
3. Perform a linear regression of linear velocity versus angular velocity. The slope of this line has the form $\frac{I}{rm}$ as per equation (12.37).
4. Multiply the slope by rm to obtain a least squares estimate of I.

This page intentionally left blank

Hills by northbound freeway between Santa Maria and Ventura

13. Lagrangian mechanics examples

A model is a set of differential equations describing how the system behaves over time. There are two common approaches for developing them.

1. Collecting data on the physical system's behavior and performing system identification with it.
2. Using physics to derive the system's model from first principles.

This chapter covers the second approach using Lagrangian mechanics.

We suggest reading *An introduction to Lagrangian and Hamiltonian Mechanics* by Simon J.A. Malham for the basics [12].

13.1 Single-jointed arm

See https://underactuated.mit.edu/pend.html for the dynamics of a single-jointed arm.

13.2 Double-jointed arm

See https://underactuated.mit.edu/multibody.html for a derivation of the kinematics and dynamics of a double-jointed arm via Lagrangian mechanics.

13.3 Cart-pole

See https://underactuated.mit.edu/acrobot.html#cart_pole for a derivation of the kinematics and dynamics of a cart-pole via Lagrangian mechanics.

Hills by northbound freeway between Santa Maria and Ventura

14. System identification

First, we'll cover a system identification procedure for generating a velocity system feedforward model from performance data. Then, we'll use that feedforward model to create state-space models for several common mechanisms.

14.1 Ordinary least squares

The parameter estimation in this chapter will be performed using ordinary least squares (OLS). Let there be a linear equation of the form $y = \beta_1 x_1 + \ldots + \beta_p x_p$. We want to find the constants $\beta_1, \ldots, \beta_p$ that best fit a set of observations represented by $x_{1,\ldots,n}$-y tuples. Consider the overdetermined system of n linear equations $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$ where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

each row corresponds to a datapoint, and $n > p$ (more datapoints than unknowns). OLS is a type of linear least squares method that estimates the unknown parameters $\boldsymbol{\beta}$ in a linear regression model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is the error in the observations $\mathbf{y}$. $\boldsymbol{\beta}$ is chosen by the method of least squares: minimizing the sum of the squares of the difference between $\mathbf{y}$ (observations of the dependent variable) and $\mathbf{X}\boldsymbol{\beta}$ (predictions of $\mathbf{y}$ using a linear function of the independent variable $\boldsymbol{\beta}$).

Geometrically, this is the sum of the squared distances, parallel to the y-axis, between each value in $\mathbf{y}$ and the corresponding value in $\mathbf{X}\boldsymbol{\beta}$. Smaller differences mean a better model fit.

To find the $\boldsymbol{\beta}$ that fits best, we can solve the following quadratic minimization problem

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathsf{T}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

$\arg\min_{\boldsymbol{\beta}}$ means "find the value of $\boldsymbol{\beta}$ that minimizes the following function of $\boldsymbol{\beta}$". Given corollary 5.12.3, take the partial derivative of the cost function with respect to $\boldsymbol{\beta}$ and set it equal to zero, then solve for $\hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
\mathbf{0} &= -2\mathbf{X}^{\mathsf{T}}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) \\
\mathbf{0} &= \mathbf{X}^{\mathsf{T}}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) \\
\mathbf{0} &= \mathbf{X}^{\mathsf{T}}\mathbf{y} - \mathbf{X}^{\mathsf{T}}\mathbf{X}\hat{\boldsymbol{\beta}} \\
\mathbf{X}^{\mathsf{T}}\mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{X}^{\mathsf{T}}\mathbf{y} \\
\hat{\boldsymbol{\beta}} &= (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}
\end{aligned} \tag{14.1}$$

14.1.1 Examples

While this is a linear regression, we can fit nonlinear functions by making the contents of $\mathbf{X}$ nonlinear functions of the independent variables. For example, we can find the quadratic equation $y = ax^2 + bx + c$ that best fits a set of x-y tuples. Lets assume we have a set of observations as follows.

$$\begin{aligned}
y_1 &= ax_1^2 + bx_1 + c \\
&\vdots \\
y_n &= ax_n^2 + bx_n + c
\end{aligned}$$

We want to find a, b, and c, so let's factor those out.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Plug these matrices into equation (14.1) to obtain the coefficients a, b, and c.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

14.2 1-DOF mechanism feedforward model

14.2.1 Introduction

The feedforward model we'll be using is

$$u = K_s \operatorname{sgn}(v) + K_v v + K_a a \tag{14.2}$$

where u is input voltage, v is velocity, a is acceleration, and K_s, K_v, and K_a are constants reflecting how much each variable contributes to the required voltage.

Since this is a linear multiple regression, the most obvious approach for obtaining the feedforward gains K_s, K_v, and K_a is to use ordinary least squares (OLS) on 4-tuples of recorded voltage, velocity sign, velocity, and acceleration. The problem is we can't cleanly measure acceleration directly, and numerical differentiation of velocity is noisy. Ideally, we could find the feedforward gains with samples of current velocity, current input, next velocity, and the nominal sample period instead.

14.2.2 Overview

First, we'll put equation (14.2) into the form $\frac{dx}{dt} = Ax + Bu + c$. Then, we'll discretize it to obtain an equation of the form $x_{k+1} = \alpha x_k + \beta u_k + \gamma \operatorname{sgn}(x_k)$. Then, we'll perform OLS on $x_{k+1} = \alpha x_k + \beta u_k + \gamma \operatorname{sgn}(x_k)$ using 4-tuples of current velocity, next velocity, input voltage, and velocity sign to obtain α, β, and γ. Since we solved for those in the discretization step earlier, we have a system of three equations we can solve for K_s, K_v, and K_a.

14.2.3 Derivation

First, solve $u = K_s \operatorname{sgn}(v) + K_v v + K_a a$ for a.

$$\begin{aligned} K_a a &= u - K_s \operatorname{sgn}(v) - K_v v \\ a &= \frac{1}{K_a} u - \frac{K_s}{K_a} \operatorname{sgn}(v) - \frac{K_v}{K_a} v \\ a &= -\frac{K_v}{K_a} v + \frac{1}{K_a} u - \frac{K_s}{K_a} \operatorname{sgn}(v) \end{aligned}$$

Let $x = v$, $\frac{dx}{dt} = a$, $A = -\frac{K_v}{K_a}$, $B = \frac{1}{K_a}$, and $c = -\frac{K_s}{K_a} \operatorname{sgn}(x)$.

$$\frac{dx}{dt} = Ax + Bu + c$$

Since $Bu + c$ is a constant over the time interval $[0, T)$ where T is the sample period, this equation can be integrated according to appendix D.1, which gives

$$x_{k+1} = e^{AT} x_k + A^{-1}(e^{AT} - 1)(Bu_k + c)$$

$$x_{k+1} = e^{AT}x_k + A^{-1}(e^{AT} - 1)Bu_k + A^{-1}(e^{AT} - 1)c$$

Substitute c back in.

$$x_{k+1} = e^{AT}x_k + A^{-1}\left(e^{AT} - 1\right)Bu_k + A^{-1}\left(e^{AT} - 1\right)\left(-\frac{K_s}{K_a}\operatorname{sgn}(x_k)\right)$$

$$x_{k+1} = e^{AT}x_k + A^{-1}\left(e^{AT} - 1\right)Bu_k - \frac{K_s}{K_a}A^{-1}\left(e^{AT} - 1\right)\operatorname{sgn}(x_k)$$

This equation has the form $x_{k+1} = \alpha x_k + \beta u_k + \gamma \operatorname{sgn}(x_k)$. Running OLS with $(x_{k+1}, x_k, u_k, \operatorname{sgn}(x_k))$ 4-tuples will give α, β, and γ. To obtain K_s, K_v, and K_a, solve the following system.

$$\begin{cases} \alpha = e^{AT} \\ \beta = A^{-1}\left(e^{AT} - 1\right)B \\ \gamma = -\frac{K_s}{K_a}A^{-1}\left(e^{AT} - 1\right) \end{cases}$$

Substitute α into the second and third equation.

$$\begin{cases} \alpha = e^{AT} \\ \beta = A^{-1}(\alpha - 1)B \\ \gamma = -\frac{K_s}{K_a}A^{-1}(\alpha - 1) \end{cases} \tag{14.3}$$

Divide the second equation by the third equation and solve for K_s.

$$\begin{aligned} \frac{\beta}{\gamma} &= \frac{A^{-1}(\alpha - 1)B}{-\frac{K_s}{K_a}A^{-1}(\alpha - 1)} \\ \frac{\beta}{\gamma} &= -\frac{K_a B}{K_s} \\ \frac{\beta}{\gamma} &= -\frac{K_a\left(\frac{1}{K_a}\right)}{K_s} \\ \frac{\beta}{\gamma} &= -\frac{1}{K_s} \\ K_s &= -\frac{\gamma}{\beta} \end{aligned} \tag{14.4}$$

Solve the second equation in (14.3) for K_v.

$$\beta = A^{-1}(\alpha - 1)B$$

$$\begin{aligned}
\beta &= \left(-\frac{K_v}{K_a}\right)^{-1}(\alpha - 1)\left(\frac{1}{K_a}\right) \\
\beta &= -\frac{K_a}{K_v}(\alpha - 1)\frac{1}{K_a} \\
\beta &= \frac{1}{K_v}(1 - \alpha) \\
K_v\beta &= 1 - \alpha \\
K_v &= \frac{1 - \alpha}{\beta}
\end{aligned} \tag{14.5}$$

Solve the first equation in (14.3) for K_a.

$$\begin{aligned}
\alpha &= e^{-\frac{K_v}{K_a}T} \\
\ln \alpha &= -\frac{K_v}{K_a}T \\
K_a \ln \alpha &= -K_v T \\
K_a &= -\frac{K_v T}{\ln \alpha}
\end{aligned}$$

Substitute in K_v from equation (14.5).

$$\begin{aligned}
K_a &= -\frac{\left(\frac{1-\alpha}{\beta}\right)T}{\ln \alpha} \\
K_a &= -\frac{(1 - \alpha)T}{\beta \ln \alpha} \\
K_a &= \frac{(\alpha - 1)T}{\beta \ln \alpha}
\end{aligned} \tag{14.6}$$

14.2.4 Results

The new system identification process is as follows.

1. Gather input voltage and velocity from a dynamic acceleration event
2. Given an equation of the form $x_{k+1} = \alpha x_k + \beta u_k + \gamma \operatorname{sgn}(x_k)$, perform OLS on $(x_{k+1}, x_k, u_k, \operatorname{sgn}(x_k))$ 4-tuples to obtain α, β, and γ

The feedforward gains are

$$K_s = -\frac{\gamma}{\beta} \tag{14.7}$$

$$K_v = \frac{1 - \alpha}{\beta} \tag{14.8}$$

$$K_a = \frac{(\alpha - 1)T}{\beta \ln \alpha} \qquad (14.9)$$
$$\alpha > 0, \beta \neq 0$$

where T is the sample period of the velocity data.

For K_s, K_v, and K_a to be defined, the dataset must include robot movement, and there must be a discernible relationship between a change in the input voltage and a change in the velocity (in other words, the robot must have accelerated).

14.3 1-DOF mechanism state-space model

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{x} = \text{velocity} \quad \dot{\mathbf{x}} = \text{acceleration} \quad \mathbf{u} = \text{voltage}$$

We want to derive what $\mathbf{A}$ and $\mathbf{B}$ are from the following feedforward model

$$\mathbf{u} = K_v\mathbf{x} + K_a\dot{\mathbf{x}}$$

Since this equation is a linear multiple regression, the constants K_v and K_a can be determined by applying ordinary least squares (OLS) to vectors of recorded input voltage, velocity, and acceleration data from quasistatic velocity tests and acceleration tests.

K_v is a proportional constant that describes how much voltage is required to maintain a given constant velocity by offsetting the electromagnetic resistance of the motor and any friction that increases linearly with speed (viscous drag). The relationship between speed and voltage (for a given initial acceleration) is linear for permanent-magnet DC motors in the FRC operating regime.

K_a is a proportional constant that describes how much voltage is required to induce a given acceleration in the motor shaft. As with K_v, the relationship between voltage and acceleration (for a given initial velocity) is linear.

To convert $\mathbf{u} = K_v\mathbf{x} + K_a\dot{\mathbf{x}}$ to state-space form, simply solve for $\dot{\mathbf{x}}$.

$$\begin{aligned}
\mathbf{u} &= K_v\mathbf{x} + K_a\dot{\mathbf{x}} \\
K_a\dot{\mathbf{x}} &= \mathbf{u} - K_v\mathbf{x} \\
K_a\dot{\mathbf{x}} &= -K_v\mathbf{x} + \mathbf{u} \\
\dot{\mathbf{x}} &= -\frac{K_v}{K_a}\mathbf{x} + \frac{1}{K_a}\mathbf{u}
\end{aligned}$$

By inspection, $\mathbf{A} = -\frac{K_v}{K_a}$ and $\mathbf{B} = \frac{1}{K_a}$. A model with position and velocity states would be

Theorem 14.3.1 — 1-DOF mechanism position model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix}\text{position} \\ \text{velocity}\end{bmatrix} \quad \mathbf{u} = \begin{bmatrix}\text{voltage}\end{bmatrix}$$

$$\dot{\mathbf{x}} = \begin{bmatrix}0 & 1 \\ 0 & -\frac{K_v}{K_a}\end{bmatrix}\mathbf{x} + \begin{bmatrix}0 \\ \frac{1}{K_a}\end{bmatrix}\mathbf{u} \tag{14.10}$$

The model in theorem 14.3.1 is undefined when $K_a = 0$. If one wants to design an LQR for such a system, use the model in theorem 14.3.2. As K_a tends to zero, acceleration requires no effort and velocity becomes an input for position control.

Theorem 14.3.2 — 1-DOF mechanism position model ($K_a = 0$).

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix}\text{position}\end{bmatrix} \quad \mathbf{u} = \begin{bmatrix}\text{velocity}\end{bmatrix}$$

$$\dot{\mathbf{x}} = \begin{bmatrix}0\end{bmatrix}\mathbf{x} + \begin{bmatrix}1\end{bmatrix}\mathbf{u} \tag{14.11}$$

14.4 Drivetrain left/right velocity state-space model

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix}\text{left velocity} \\ \text{right velocity}\end{bmatrix} \quad \dot{\mathbf{x}} = \begin{bmatrix}\text{left acceleration} \\ \text{right acceleration}\end{bmatrix} \quad \mathbf{u} = \begin{bmatrix}\text{left voltage} \\ \text{right voltage}\end{bmatrix}$$

We want to derive what $\mathbf{A}$ and $\mathbf{B}$ are from linear and angular feedforward models. Since the left and right dynamics are symmetric, we'll guess the model has the form

$$\mathbf{A} = \begin{bmatrix}A_1 & A_2 \\ A_2 & A_1\end{bmatrix} \quad \mathbf{B} = \begin{bmatrix}B_1 & B_2 \\ B_2 & B_1\end{bmatrix}$$

Let $K_{v,lin}$ be the linear velocity gain, $K_{a,lin}$ be the linear acceleration gain, $K_{v,ang}$ be the angular velocity gain, and $K_{a,ang}$ be the angular acceleration gain. Let $\mathbf{u} = \begin{bmatrix}K_{v,lin}v & K_{v,lin}v\end{bmatrix}^\mathsf{T}$ be the input that makes both sides of the drivetrain move at a constant velocity v. Therefore, $\mathbf{x} = \begin{bmatrix}v & v\end{bmatrix}^\mathsf{T}$ and $\dot{\mathbf{x}} = \begin{bmatrix}0 & 0\end{bmatrix}^\mathsf{T}$. Substitute these into

the state-space model.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \begin{bmatrix} K_{v,lin} v \\ K_{v,lin} v \end{bmatrix}$$

Since the column vectors contain the same element, the elements in the second row can be rearranged.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,lin} v \\ K_{v,lin} v \end{bmatrix}$$

Since the rows are linearly dependent, we can use just one of them.

$$\begin{aligned} 0 &= \begin{bmatrix} A_1 & A_2 \end{bmatrix} v + \begin{bmatrix} B_1 & B_2 \end{bmatrix} K_{v,lin} v \\ 0 &= \begin{bmatrix} v & v & K_{v,lin} v & K_{v,lin} v \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} \\ 0 &= \begin{bmatrix} 1 & 1 & K_{v,lin} & K_{v,lin} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} \end{aligned}$$

Let $\mathbf{u} = \begin{bmatrix} K_{v,lin} v + K_{a,lin} a & K_{v,lin} v + K_{a,lin} a \end{bmatrix}^\mathsf{T}$ be the input that accelerates both sides of the drivetrain by a from an initial velocity of v. Therefore, $\mathbf{x} = \begin{bmatrix} v & v \end{bmatrix}^\mathsf{T}$ and $\dot{\mathbf{x}} = \begin{bmatrix} a & a \end{bmatrix}^\mathsf{T}$. Substitute these into the state-space model.

$$\begin{bmatrix} a \\ a \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \begin{bmatrix} K_{v,lin} v + K_{a,lin} a \\ K_{v,lin} v + K_{a,lin} a \end{bmatrix}$$

Since the column vectors contain the same element, the elements in the second row can be rearranged.

$$\begin{bmatrix} a \\ a \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,lin} v + K_{a,lin} a \\ K_{v,lin} v + K_{a,lin} a \end{bmatrix}$$

Since the rows are linearly dependent, we can use just one of them.

$$\begin{aligned} a &= \begin{bmatrix} A_1 & A_2 \end{bmatrix} v + \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,lin} v + K_{a,lin} a \end{bmatrix} \\ a &= \begin{bmatrix} v & v & K_{v,lin} v + K_{a,lin} a & K_{v,lin} + K_{a,lin} a \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} \end{aligned}$$

Let $\mathbf{u} = \begin{bmatrix} -K_{v,ang}v & K_{v,ang}v \end{bmatrix}^\mathsf{T}$ be the input that rotates the drivetrain in place where each wheel has a constant velocity v. Therefore, $\mathbf{x} = \begin{bmatrix} -v & v \end{bmatrix}^\mathsf{T}$ and $\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 \end{bmatrix}^\mathsf{T}$.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} -v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \begin{bmatrix} -K_{v,ang}v \\ K_{v,ang}v \end{bmatrix}$$
$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ -A_2 & A_1 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ -B_2 & B_1 \end{bmatrix} \begin{bmatrix} K_{v,ang}v \\ K_{v,ang}v \end{bmatrix}$$
$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ A_1 & -A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ B_1 & -B_2 \end{bmatrix} \begin{bmatrix} K_{v,ang}v \\ K_{v,ang}v \end{bmatrix}$$

Since the column vectors contain the same element, the elements in the second row can be rearranged.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ -A_1 & A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ -B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,ang}v \\ K_{v,ang}v \end{bmatrix}$$

Since the rows are linearly dependent, we can use just one of them.

$$0 = \begin{bmatrix} -A_1 & A_2 \end{bmatrix} v + \begin{bmatrix} -B_1 & B_2 \end{bmatrix} K_{v,ang}v$$
$$0 = -vA_1 + vA_2 - K_{v,ang}vB_1 + K_{v,ang}vB_2$$
$$0 = \begin{bmatrix} -v & v & -K_{v,ang}v & K_{v,ang}v \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix}$$
$$0 = \begin{bmatrix} -1 & 1 & -K_{v,ang} & K_{v,ang} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix}$$

Let $\mathbf{u} = \begin{bmatrix} -K_{v,ang}v - K_{a,ang}a & K_{v,ang}v + K_{a,ang}a \end{bmatrix}^\mathsf{T}$ be the input that rotates the drivetrain in place where each wheel has an initial speed of v and accelerates by a. Therefore, $\mathbf{x} = \begin{bmatrix} -v & v \end{bmatrix}^\mathsf{T}$ and $\dot{\mathbf{x}} = \begin{bmatrix} -a & a \end{bmatrix}^\mathsf{T}$.

$$\begin{bmatrix} -a \\ a \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} -v \\ v \end{bmatrix} + \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \begin{bmatrix} -K_{v,ang}v - K_{a,ang}a \\ K_{v,ang}v + K_{a,ang}a \end{bmatrix}$$
$$\begin{bmatrix} -a \\ a \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ -A_2 & A_1 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ -B_2 & B_1 \end{bmatrix} \begin{bmatrix} K_{v,ang}v + K_{a,ang}a \\ K_{v,ang}v + K_{a,ang}a \end{bmatrix}$$

$$\begin{bmatrix} -a \\ a \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ A1 & -A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ B_1 & -B_2 \end{bmatrix} \begin{bmatrix} K_{v,ang}v + K_{a,ang}a \\ K_{v,ang}v + K_{a,ang}a \end{bmatrix}$$

Since the column vectors contain the same element, the elements in the second row can be rearranged.

$$\begin{bmatrix} -a \\ -a \end{bmatrix} = \begin{bmatrix} -A_1 & A_2 \\ -A_1 & A_2 \end{bmatrix} \begin{bmatrix} v \\ v \end{bmatrix} + \begin{bmatrix} -B_1 & B_2 \\ -B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,ang}v + K_{a,ang}a \\ K_{v,ang}v + K_{a,ang}a \end{bmatrix}$$

Since the rows are linearly dependent, we can use just one of them.

$$\begin{aligned} -a &= \begin{bmatrix} -A_1 & A_2 \end{bmatrix} v + \begin{bmatrix} -B_1 & B_2 \end{bmatrix} \begin{bmatrix} K_{v,ang}v + K_{a,ang}a \end{bmatrix} \\ -a &= -vA_1 + vA_2 - (K_{v,ang}v + K_{a,ang}a)B_1 + (K_{v,ang}v + K_{a,ang}a)B_2 \end{aligned}$$

$$-a = \begin{bmatrix} -v & v & -(K_{v,ang}v + K_{a,ang}a) & K_{v,ang}v + K_{a,ang}a \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix}$$

$$a = \begin{bmatrix} v & -v & K_{v,ang}v + K_{a,ang}a & -(K_{v,ang}v + K_{a,ang}a) \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix}$$

Now stack the rows.

$$\begin{bmatrix} 0 \\ a \\ 0 \\ a \end{bmatrix} = \begin{bmatrix} 1 & 1 & K_{v,lin} & K_{v,lin} \\ v & v & K_{v,lin}v + K_{a,lin}a & K_{v,lin}v + K_{a,lin}a \\ -1 & 1 & -K_{v,ang} & K_{v,ang} \\ v & -v & K_{v,ang}v + K_{a,ang}a & -(K_{v,ang}v + K_{a,ang}a) \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix}$$

Solve for matrix elements with Wolfram Alpha. Let $b = K_{v,lin}$, $c = K_{a,lin}$, $d = K_{v,ang}$, and $f = K_{a,ang}$.

```
inverse of {{1, 1, b, b},
            {v, v, b v + c a, b v + c a},
            {-1, 1, -d, d},
            {v, -v, d v + f a, -(d v + f a)}}
           * {{0}, {a}, {0}, {a}}
```

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2cf} \begin{bmatrix} -cd - bf \\ cd - bf \\ c + f \\ f - c \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2K_{a,lin}K_{a,ang}} \begin{bmatrix} -K_{a,lin}K_{v,ang} - K_{v,lin}K_{a,ang} \\ K_{a,lin}K_{v,ang} - K_{v,lin}K_{a,ang} \\ K_{a,lin} + K_{a,ang} \\ K_{a,ang} - K_{a,lin} \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -\frac{K_{a,lin}K_{v,ang}}{K_{a,lin}K_{a,ang}} - \frac{K_{v,lin}K_{a,ang}}{K_{a,lin}K_{a,ang}} \\ \frac{K_{a,lin}K_{v,ang}}{K_{a,lin}K_{a,ang}} - \frac{K_{v,lin}K_{a,ang}}{K_{a,lin}K_{a,ang}} \\ \frac{K_{a,lin}}{K_{a,lin}K_{a,ang}} + \frac{K_{a,ang}}{K_{a,lin}K_{a,ang}} \\ \frac{K_{a,ang}}{K_{a,lin}K_{a,ang}} - \frac{K_{a,lin}}{K_{a,lin}K_{a,ang}} \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -\frac{K_{v,ang}}{K_{a,ang}} - \frac{K_{v,lin}}{K_{a,lin}} \\ \frac{K_{v,ang}}{K_{a,ang}} - \frac{K_{v,lin}}{K_{a,lin}} \\ \frac{1}{K_{a,ang}} + \frac{1}{K_{a,lin}} \\ \frac{1}{K_{a,lin}} - \frac{1}{K_{a,ang}} \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -\left(\frac{K_{v,lin}}{K_{a,lin}} + \frac{K_{v,ang}}{K_{a,ang}}\right) \\ -\left(\frac{K_{v,lin}}{K_{a,lin}} - \frac{K_{v,ang}}{K_{a,ang}}\right) \\ \frac{1}{K_{a,lin}} + \frac{1}{K_{a,ang}} \\ \frac{1}{K_{a,lin}} - \frac{1}{K_{a,ang}} \end{bmatrix}$$

To summarize,

Theorem 14.4.1 — Drivetrain left/right velocity model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} \text{left velocity} \\ \text{right velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\dot{\mathbf{x}} = \begin{bmatrix} A_1 & A_2 \\ A_2 & A_1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \mathbf{u}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -\left(\frac{K_{v,lin}}{K_{a,lin}} + \frac{K_{v,ang}}{K_{a,ang}}\right) \\ -\left(\frac{K_{v,lin}}{K_{a,lin}} - \frac{K_{v,ang}}{K_{a,ang}}\right) \\ \frac{1}{K_{a,lin}} + \frac{1}{K_{a,ang}} \\ \frac{1}{K_{a,lin}} - \frac{1}{K_{a,ang}} \end{bmatrix} \tag{14.12}$$

$K_{v,ang}$ and $K_{a,ang}$ have units of $\frac{V}{L/T}$ and $\frac{V}{L/T^2}$ respectively where V means voltage units, L means length units, and T means time units.

If K_v and K_a are the same for both the linear and angular cases, it devolves to the one-dimensional case. This means the left and right sides are decoupled.

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -\left(\frac{K_v}{K_a} + \frac{K_v}{K_a}\right) \\ -\left(\frac{K_v}{K_a} - \frac{K_v}{K_a}\right) \\ \frac{1}{K_a} + \frac{1}{K_a} \\ \frac{1}{K_a} - \frac{1}{K_a} \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -2\frac{K_v}{K_a} \\ 0 \\ \frac{2}{K_a} \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} -\frac{K_v}{K_a} \\ 0 \\ \frac{1}{K_a} \\ 0 \end{bmatrix}$$

14.5 Drivetrain linear/angular velocity state-space model

Let the linear and angular voltage contributions be

$$u_{lin} = K_{v,lin}v + K_{a,lin}a$$
$$u_{ang} = K_{v,ang}\omega + K_{a,ang}\alpha$$

Solve for acceleration.

$$a = -\frac{K_{v,lin}}{K_{a,lin}}v + \frac{1}{K_{a,lin}}u_{lin}$$
$$\alpha = -\frac{K_{v,ang}}{K_{a,ang}}\omega + \frac{1}{K_{a,ang}}u_{lin}$$

Factor them into a matrix equation.

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{K_{v,lin}}{K_{a,lin}} & 0 \\ 0 & -\frac{K_{v,ang}}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{K_{a,lin}} & 0 \\ 0 & \frac{1}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} u_{lin} \\ u_{ang} \end{bmatrix}$$

We want to write this model in terms of left and right voltages. The linear and angular voltages have the following relations.

$$u_{left} = u_{lin} - u_{ang}$$
$$u_{right} = u_{lin} + u_{ang}$$

Factor them into a matrix equation, then solve for the linear and angular voltages.

$$\begin{bmatrix} u_{left} \\ u_{right} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_{lin} \\ u_{ang} \end{bmatrix}$$
$$\begin{bmatrix} u_{lin} \\ u_{ang} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} u_{left} \\ u_{right} \end{bmatrix}$$

Plug this into the model input.

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{K_{v,lin}}{K_{a,lin}} & 0 \\ 0 & -\frac{K_{v,ang}}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{K_{a,lin}} & 0 \\ 0 & \frac{1}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} u_{left} \\ u_{right} \end{bmatrix}$$
$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{K_{v,lin}}{K_{a,lin}} & 0 \\ 0 & -\frac{K_{v,ang}}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{0.5}{K_{a,lin}} & \frac{0.5}{K_{a,lin}} \\ -\frac{0.5}{K_{a,ang}} & \frac{0.5}{K_{a,ang}} \end{bmatrix} \begin{bmatrix} u_{left} \\ u_{right} \end{bmatrix}$$

Theorem 14.5.1 — Drivetrain linear/angular velocity model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{x} = \begin{bmatrix} \text{linear velocity} \\ \text{angular velocity} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \text{left voltage} \\ \text{right voltage} \end{bmatrix}$$

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{K_{v,lin}}{K_{a,lin}} & 0 \\ 0 & -\frac{K_{v,ang}}{K_{a,ang}} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{0.5}{K_{a,lin}} & \frac{0.5}{K_{a,lin}} \\ -\frac{0.5}{K_{a,ang}} & \frac{0.5}{K_{a,ang}} \end{bmatrix} \mathbf{u}$$

$K_{v,ang}$ and $K_{a,ang}$ have units of $\frac{V}{A/T}$ and $\frac{V}{A/T^2}$ respectively where V means voltage units, A means angle units, and T means time units.

V Motion planning

This page intentionally left blank

15. Motion profiles

If smooth, predictable motion of a system over time is desired, it's best to only change a system's reference as fast as the system is able to physically move. Motion profiles, also known as trajectories, are used for this purpose. For multi-state systems, each state is given its own trajectory. Since these states are usually position and velocity, they share different derivatives of the same profile.

15.1 1-DOF motion profiles

Trapezoid profiles (figure 15.1) and S-curve profiles (figure 15.2) are the most commonly used motion profiles in FRC for point-to-point movements with one degree of freedom (1 DOF). These profiles accelerate the system to a maximum velocity from rest, then decelerate it later such that the final acceleration velocity, are zero at the moment the system arrives at the desired location.

These profiles are given their names based on the shape of their velocity trajectory. The trapezoid profile has a velocity trajectory shaped like a trapezoid and the S-curve profile has a velocity trajectory shaped like an S-curve.

In the context of a point-to-point move, a full S-curve consists of seven distinct phases of motion. Phase I starts moving the system from rest at a linearly increasing acceleration until it reaches the maximum acceleration. In phase II, the profile accelerates at this maximum acceleration rate until it must start decreasing as it approaches the maximum velocity. This occurs in phase III when the acceleration linearly decreases

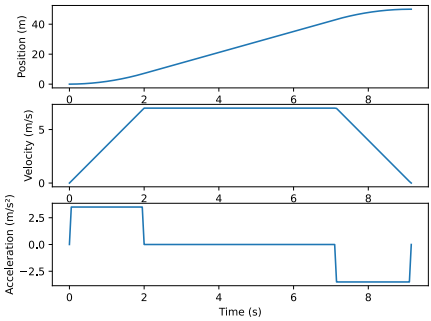


Figure 15.1: Trapezoid profile

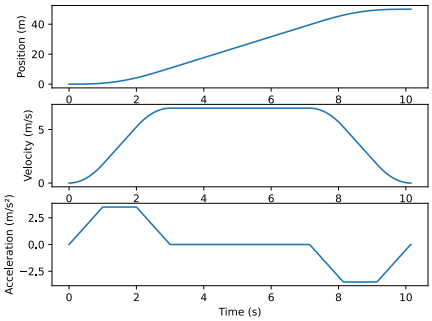


Figure 15.2: S-curve profile

until it reaches zero. In phase IV, the velocity is constant until deceleration begins, at which point the profiles decelerates in a manner symmetric to phases I, II and III.

A trapezoid profile, on the other hand, has three phases. It is a subset of an S-curve profile, having only the phases corresponding to phase II of the S-curve profile (constant acceleration), phase IV (constant velocity), and phase VI (constant deceleration). This reduced number of phases underscores the difference between these two profiles: the S-curve profile has extra motion phases which transition between periods of acceleration and periods of nonacceleration; the trapezoid profile has instantaneous transitions between these phases. This can be seen in the acceleration graphs of the corresponding velocity profiles for these two profile types.

15.1.1 Jerk

The motion characteristic that defines the change in acceleration, or transitional period, is known as "jerk". Jerk is defined as the rate of change of acceleration with time. In a trapezoid profile, the jerk (change in acceleration) is infinite at the phase transitions, while in the S-curve profile the jerk is a constant value, spreading the change in acceleration over a period of time.

From figures 15.1 and 15.2, we can see S-curve profiles are smoother than trapezoid profiles. Why, however, do the S-curve profile result in less load oscillation? For a given load, the higher the jerk, the greater the amount of unwanted vibration energy will be generated, and the broader the frequency spectrum of the vibration's energy will be.

This means that more rapid changes in acceleration induce more powerful vibrations, and more vibrational modes will be excited. Because vibrational energy is absorbed in the system mechanics, it may cause an increase in settling time or reduced accuracy if

the vibration frequency matches resonances in the mechanical system.

15.1.2 Profile selection

Since trapezoid profiles spend their time at full acceleration or full deceleration, they are, from the standpoint of profile execution, faster than S-curve profiles. However, if this "all on"/"all off" approach causes an increase in settling time, the advantage is lost. Often, only a small amount of "S" (transition between acceleration and no acceleration) can substantially reduce induced vibration. Therefore to optimize throughput, the S-curve profile must be tuned for each a given load and given desired transfer speed.

What S-curve form is right for a given system? On an application by application basis, the specific choice of the form of the S-curve will depend on the mechanical nature of the system and the desired performance specifications. For example, in medical applications which involve liquid transfers that should not be jostled, it would be appropriate to choose a profile with no phase II and VI segment at all. Instead the acceleration transitions would be spread out as far as possible, thereby maximizing smoothness.

In other applications involving high speed pick and place, overall transfer speed is most important, so a good choice might be an S-curve with transition phases (phases I, III, V, and VII) that are five to fifteen percent of phase II and VI. In this case, the S-curve profile will add a small amount of time to the overall transfer time. However, the reduced load oscillation at the end of the move considerably decreases the total effective transfer time. Trial and error using a motion measurement system is generally the best way to determine the right amount of "S" because modeling high frequency dynamics is difficult to do accurately.

Another consideration is whether that "S" segment will actually lead to smoother control of the system. If the high frequency dynamics at play are negligible, one can use the simpler trapezoid profile.

R S-curve profiles are unnecesary for FRC mechanisms. Motors in FRC effectively have first-order velocity dynamics because the inductance effects are on the order of microseconds; FRC dynamics operate on the order of milliseconds. First-order motor models can achieve the instantaneous acceleration changes of trapezoid profiles because voltage is electromotive force, which is analogous to acceleration. That is, we can instantaneously achieve any desired acceleration with a finite voltage, and we can follow any trapezoid profile perfectly with only feedforward control.

15.1.3 Profile equations

Trapezoid profile

The trapezoid profile uses the following equations.

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2$$
$$v(t) = v_0 + at$$

where $x(t)$ is the position at time t, x_0 is the initial position, v_0 is the initial velocity, and a is the acceleration at time t.

Snippet 15.1 shows a Python implementation.

```
"""Function for generating a trapezoid profile."""

import math


def generate_trapezoid_profile(max_v, time_to_max_v, dt, goal):
    """Creates a trapezoid profile with the given constraints.

    Returns:
    t_rec -- list of timestamps
    x_rec -- list of positions at each timestep
    v_rec -- list of velocities at each timestep
    a_rec -- list of accelerations at each timestep

    Keyword arguments:
    max_v -- maximum velocity of profile
    time_to_max_v -- time from rest to maximum velocity
    dt -- timestep
    goal -- final position when the profile is at rest
    """
    t_rec = [0.0]
    x_rec = [0.0]
    v_rec = [0.0]
    a_rec = [0.0]

    a = max_v / time_to_max_v
    time_at_max_v = goal / max_v - time_to_max_v

    # If profile is short
    if max_v * time_to_max_v > goal:
        time_to_max_v = math.sqrt(goal / a)
        time_from_max_v = time_to_max_v
        time_total = 2.0 * time_to_max_v
        profile_max_v = a * time_to_max_v
    else:
        time_from_max_v = time_to_max_v + time_at_max_v
        time_total = time_from_max_v + time_to_max_v
        profile_max_v = max_v

    while t_rec[-1] < time_total:
        t = t_rec[-1] + dt
        t_rec.append(t)
```

```
        if t < time_to_max_v:
            # Accelerate up
            a_rec.append(a)
            v_rec.append(a * t)
        elif t < time_from_max_v:
            # Maintain max velocity
            a_rec.append(0.0)
            v_rec.append(profile_max_v)
        elif t < time_total:
            # Accelerate down
            decel_time = t - time_from_max_v
            a_rec.append(-a)
            v_rec.append(profile_max_v - a * decel_time)
        else:
            a_rec.append(0.0)
            v_rec.append(0.0)
        x_rec.append(x_rec[-1] + v_rec[-1] * dt)
    return t_rec, x_rec, v_rec, a_rec
```

Snippet 15.1. Trapezoid profile implementation in Python

S-curve profile

The S-curve profile equations also include jerk.

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2 + \frac{1}{6}jt^3$$
$$v(t) = v_0 + at + \frac{1}{2}jt^2$$
$$a(t) = a_0 + jt$$

where j is the jerk at time t, $a(t)$ is the acceleration at time t, and a_0 is the initial acceleration.

Snippet 15.2 shows a Python implementation.

```
"""Function for generating an S-curve profile."""

import math


def generate_s_curve_profile(max_v, max_a, time_to_max_a, dt, goal):
    """Returns an s-curve profile with the given constraints.

    Returns:
    t_rec -- list of timestamps
    x_rec -- list of positions at each timestep
    v_rec -- list of velocities at each timestep
    a_rec -- list of accelerations at each timestep

    Keyword arguments:
    max_v -- maximum velocity of profile
    max_a -- maximum acceleration of profile
    time_to_max_a -- time from rest to maximum acceleration
```

```
    dt -- timestep
    goal -- final position when the profile is at rest
    """
    t_rec = [0.0]
    x_rec = [0.0]
    v_rec = [0.0]
    a_rec = [0.0]

    j = max_a / time_to_max_a
    short_profile = max_v * (time_to_max_a + max_v / max_a) > goal

    if short_profile:
        profile_max_v = max_a * (
            math.sqrt(goal / max_a - 0.75 * time_to_max_a**2) - 0.5 *
                time_to_max_a
        )
    else:
        profile_max_v = max_v

    # Find times at critical points
    t2 = profile_max_v / max_a
    t3 = t2 + time_to_max_a
    if short_profile:
        t4 = t3
    else:
        t4 = goal / profile_max_v
    t5 = t4 + time_to_max_a
    t6 = t4 + t2
    t7 = t6 + time_to_max_a
    time_total = t7

    while t_rec[-1] < time_total:
        t = t_rec[-1] + dt
        t_rec.append(t)
        if t < time_to_max_a:
            # Ramp up acceleration
            a_rec.append(j * t)
            v_rec.append(0.5 * j * t**2)
        elif t < t2:
            # Increase speed at max acceleration
            a_rec.append(max_a)
            v_rec.append(max_a * (t - 0.5 * time_to_max_a))
        elif t < t3:
            # Ramp down acceleration
            a_rec.append(max_a - j * (t - t2))
            v_rec.append(max_a * (t - 0.5 * time_to_max_a) - 0.5 * j * (t -
                t2) ** 2)
        elif t < t4:
            # Maintain max velocity
            a_rec.append(0.0)
            v_rec.append(profile_max_v)
        elif t < t5:
            # Ramp down acceleration
            a_rec.append(-j * (t - t4))
            v_rec.append(profile_max_v - 0.5 * j * (t - t4) ** 2)
        elif t < t6:
            # Decrease speed at max acceleration
            a_rec.append(-max_a)
            v_rec.append(max_a * (t2 + t5 - t - 0.5 * time_to_max_a))
        elif t < t7:
```

```
            # Ramp up acceleration
            a_rec.append(-max_a + j * (t - t6))
            v_rec.append(
                max_a * (t2 + t5 - t - 0.5 * time_to_max_a) + 0.5 * j * (t -
                    t6) ** 2
            )
        else:
            a_rec.append(0.0)
            v_rec.append(0.0)
        x_rec.append(x_rec[-1] + v_rec[-1] * dt)
    return t_rec, x_rec, v_rec, a_rec
```

Snippet 15.2. S-curve profile implementation in Python

15.1.4 Other profile types

The ultimate goal of any profile is to match the profile's motion characteristics to the desired application. Trapezoid and S-curve profiles work well when the system's torque response curve is fairly flat. In other words, when the output torque does not vary that much over the range of velocities the system will be experiencing. This is true for most servo motor systems, whether DC brushed or DC brushless.

Step motors, however, do not have flat torque/speed curves. Torque output is nonlinear, sometimes has a large drop at a location called the "mid-range instability", and generally drops off at higher velocities.

Mid-range instability occurs at the step frequency when the motor's natural resonance frequency matches the current step rate. To address mid-range instability, the most common technique is to use a nonzero starting velocity. This means that the profile instantly "jumps" to a programmed velocity upon initial acceleration, and while decelerating. While crude, this technique sometimes provides better results than a smooth ramp for zero, particularly for systems that do not use a microstepping drive technique.

To address torque drop-off at higher velocities, a parabolic profile can be used. The corresponding acceleration curve has the smallest acceleration when the velocity is highest. This is a good match for stepper motors because there is less torque available at higher speeds. However, notice that starting and ending accelerations are very high, and there is no "S" phase where the acceleration smoothly transitions to zero. If load oscillation is a problem, parabolic profiles may not work as well as an S-curve despite the fact that a standard S-curve profile is not optimized for a stepper motor from the standpoint of the torque/speed curve.

15.1.5 Further reading

FRC teams 254 and 971 gave a talk at FIRST World Championships in 2015 about motion profiles.

"Motion Planning and Control in FRC" (59 minutes)
Team 254: The Cheesy Poofs
https://youtu.be/8319J1BEHwM

15.2 2-DOF motion profiles

In FRC, point-to-point movements with two degrees of freedom (2 DOFs) are almost always within the context of drivetrains where the two degrees of freedom are the x and y axes.

A *path* is a set of (x, y) points for the drivetrain to follow. A drivetrain *trajectory* is a path that includes both the states (e.g., position and velocity) and control inputs (e.g., voltage) of the drivetrain as functions of time.

Currently, the most common form of multidimensional trajectory planning in FRC is based on polynomial splines. The splines determine the path of the trajectory, and the path is parameterized by a trapezoidal motion profile to create a trajectory. *A Dive into WPILib Trajectories* by Declan Freeman-Gleason introduces 2-DOF trajectory planning in FRC using polynomial splines [4].

Planning Motion Trajectories for Mobile Robots Using Splines by Christoph Sprunk provides a more general treatment of spline-based trajectory generation [15].


OPERS field at UCSC

16. Configuration spaces

16.1 Introduction

When a robot can interfere with itself, we need a motion profiling solution that ensures the robot doesn't destroy itself. Now that we've demonstrated various methods of motion profiling, how can we incorporate safe zones into the planning process or implement mechanism collision avoidance?

1-DOF systems are straightforward. We define the minimum and maximum position constraints for the arm, elevator, etc. and ensure we stay between them. To do so, we might use some combination of:

1. Rejecting, coercing, or clamping setpoints so that they are always within the valid range
2. Soft limits on the speed controller
3. Limit switches, and/or physical hard stops

Multi-DOF systems are more complex. In the 2-DOF case (e.g., an arm with a shoulder and a wrist, or an elevator and an arm), each DOF has some limits that are independent of the rest of the system (e.g., min and max angles), but there are also dependent constraints (e.g., if the shoulder is angled down, the wrist must be angled up to avoid hitting the ground). Examples of such constraints could include:

- Minimum/maximum angles
- Maximum extension rules

- Robot chassis
- Angles that require an infeasible amount of holding torque

One intuitive way to think about this is to envision an N-D space (2D for 2 DOFs, etc.) where the x and y axes are the positions of the degrees of freedom. A given point (x, y) can either be a valid configuration or an invalid configuration (meaning it violates one or more constraints). So, our 2D plot may have regions (half-planes, rectangles, etc.) representing constraints.

16.2 Waypoint planning

Our first example has 2 DOFs where all of the constraints are independent (minimum/-maximum limits on each axis), as shown in figure 16.1.

Motion planning and control in this example is simple; just make sure all controller references are always inside the box of valid states.

Now let's make it more interesting. Say that if the arm is near the middle of travel (pointing straight down), the elevator must be above a certain height. Otherwise, the arm intersects the robot chassis. This configuration space would look like figure 16.2.

Let's consider motion planning in the second example. Say the current state of our system is x and our goal state is r as shown in figure 16.3.

Notice that the elevator height at the current state and the goal is identical. Do we just have to move the arm? Moving this way will sweep the system into the constraint we defined for preventing self-intersection with the chassis. Instead, we need to plan a sequence of references to execute that only pass through valid states. Figure 16.4 shows an example.

a and b are intermediate waypoints we added to ensure that the whole system moves in a safe manner. Considering again our toy example, the elevator goes up, then the arm rotates underneath, then the elevator goes back down. There are many possible ways to generate this path (or other valid paths), including some sort of search (discretize the plot into a grid and find a valid path using breadth-first search or A*) or hard-coded rules to handle different regions of the constraint space.

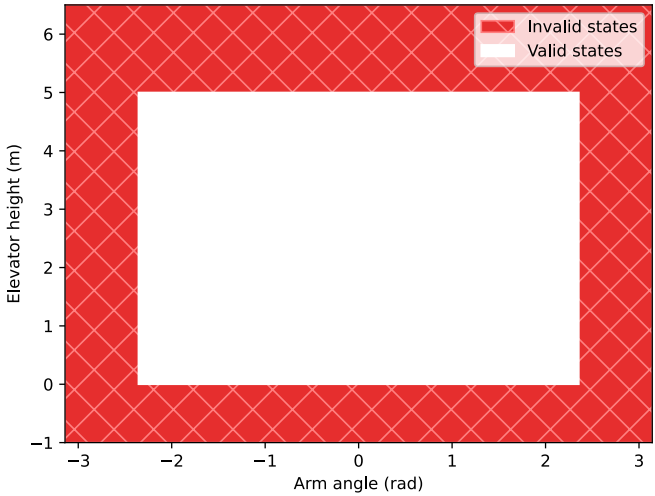


Figure 16.1: Elevator-arm configuration space with independent constraints

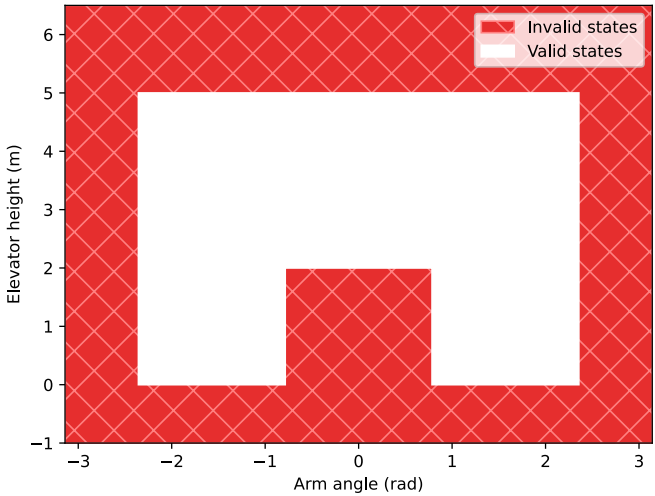


Figure 16.2: Elevator-arm configuration space requiring elevator to be above a certain height when arm is pointing down

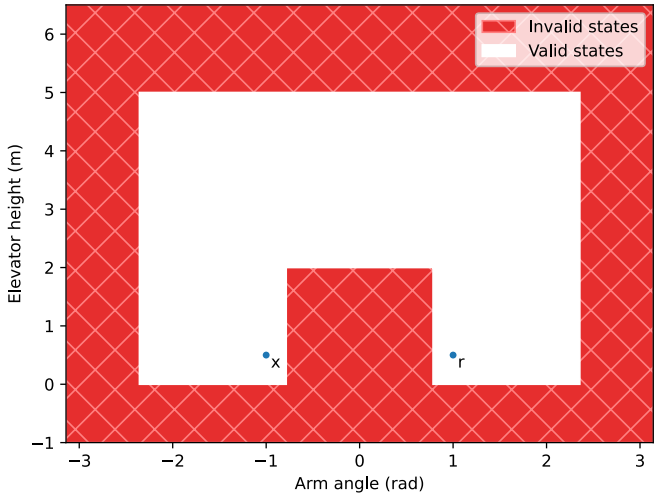


Figure 16.3: Elevator-arm configuration space with initial state x and goal state r

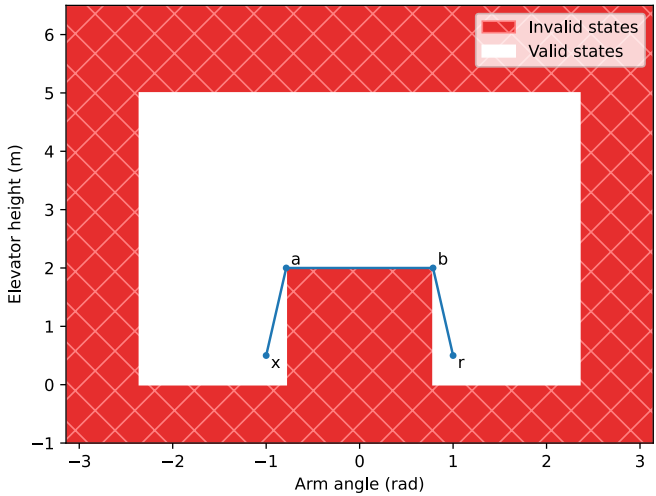


Figure 16.4: Elevator-arm configuration space with path between initial state x and final state r

16.3 Waypoint traversal

Once we have a sequence of waypoints, there are a few options for traversing them:

1. Generate trajectories for each DOF and time-synchronize them. This ensures the mechanism follows the "diagonal lines" in the path above. The upside of this approach is it can yield the fastest (time-optimal) coordinated movements possible. The downside is that the motion planning is "open loop" – if one of the DOFs lags behind its trajectory, it's possible that the system may still end up in an invalid state. In practice, we include some buffer around our references to ensure that tracking error doesn't immediately lead to violated constraints. We are making our trajectories more robust to uncertainty.
2. Perform movements sequentially. That is, move from x to a, and only start moving toward b once the path from the current system state to b is a straight line that is collision-free. This gives robustness to control error, but may result in unnecessary stops or jerky movement compared to the optimized, open-loop trajectory.

This planning approach generalizes to an arbitrary number of DOFs and is essentially how industrial 7-DOF arms are operated.

16.4 State machine validation with safe sets

Configuration spaces can also be used for state machine validation. The states of all the mechanisms, including solenoid actuation states, would be axes in a configuration space. A state machine unit test would define a valid path through the configuration space, send the desired events to the state machine, then ensure the robot took the correct path in response to each event for all iterations.

This page intentionally left blank

17. Trajectory optimization

A *trajectory* is a collection of samples defined over some time interval. A drivetrain trajectory would include its states (e.g., x position, y position, heading, and velocity) and control inputs (e.g., voltage).

Trajectory optimization finds the best choice of trajectory (for some mathematical definition of "best") by formulating and solving a constrained optimization problem.

17.1 Solving optimization problems with Sleipnir

Sleipnir is a C++ library that facilitates the specification of constrained nonlinear optimization problems with natural mathematical notation, then efficiently solves them.

Sleipnir supports problems of the form:

$$\begin{aligned} \min_x & f(x) \\ \text{subject to } & c_e(x) = 0 \\ & c_i(x) \geq 0 \end{aligned}$$

where $f(x)$ is the scalar cost function, x is the vector of decision variables (variables the solver can tweak to minimize the cost function), $c_e(x)$ is the vector-valued function whose rows are equality constraints, and $c_i(x)$ is the vector-valued function whose rows are inequality constraints. Constraints are equations or inequalities of the decision

variables that constrain what values the solver is allowed to use when searching for an optimal solution.

The nice thing about Sleipnir is users don't have to put their system in the form shown above manually; they can write it in natural mathematical form and it'll be converted for them. We'll cover some examples next.

17.1.1 Double integrator minimum time

A system with position and velocity states and an acceleration input is an example of a double integrator. We want to go from 0 m at rest to 10 m at rest in the minimum time while obeying the velocity limit $(-1, 1)$ and the acceleration limit $(-1, 1)$.

The model for our double integrator is $\ddot{x} = u$ where x is the vector $\begin{bmatrix}\text{position} & \text{velocity}\end{bmatrix}^\mathsf{T}$ and u is the acceleration. The velocity constraints are $-1 \leq x(1) \leq 1$ and the acceleration constraints are $-1 \leq u \leq 1$.

Importing required libraries

Jormungandr is the Python version of Sleipnir and can be installed via `pip install sleipnirgroup-jormungandr`.

```
from jormungandr.optimization import Problem
import numpy as np
```

Initializing a problem instance

First, we need to make a problem instance.

```
dt = 0.005  # 5 ms
N = int(T / dt)

r = 2.0

problem = Problem()
```

Creating decision variables

Next, we need to make decision variables for our state and input.

```
X = problem.decision_variable(2, N + 1)

# 1x1 input vector with N timesteps (input at last state doesn't matter)
U = problem.decision_variable(1, N)
```

By convention, we use capital letters for the variables to designate matrices.

Applying constraints

Now, we need to apply dynamics constraints between timesteps.

```
for k in range(N):
    p_k1 = X[0, k + 1]
    v_k1 = X[1, k + 1]
    p_k = X[0, k]
    v_k = X[1, k]
    a_k = U[0, k]

    problem.subject_to(p_k1 == p_k + v_k * dt + 0.5 * a_k * dt**2)
    problem.subject_to(v_k1 == v_k + a_k * dt)
```

Next, we'll apply the state and input constraints.

```
problem.subject_to(X[:, 0] == np.array([[0.0], [0.0]]))
problem.subject_to(X[:, N] == np.array([[r], [0.0]]))

# Limit velocity
problem.subject_to(-1 <= X[1, :])
problem.subject_to(X[1, :] <= 1)

# Limit acceleration
problem.subject_to(-1 <= U)
problem.subject_to(U <= 1)
```

Specifying a cost function

Next, we'll create a cost function for minimizing position error.

```
J = 0.0
for k in range(N + 1):
    J += (r - X[0, k]) ** 2
problem.minimize(J)
```

The cost function passed to Minimize() should produce a scalar output.

Solving the problem

Now we can solve the problem.

The solver will find the decision variable values that minimize the cost function while satisfying the constraints.

Accessing the solution

You can obtain the solution by querying the values of the variables like so.

```
position = X.value[0, 0]
velocity = X.value[1, 0]
acceleration = U.value(0)
```

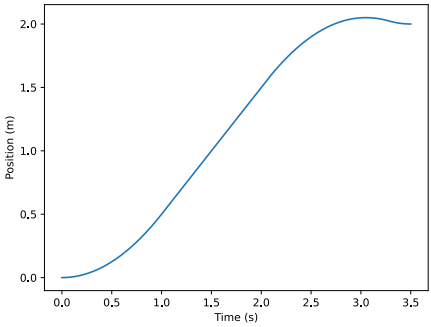


Figure 17.1: Double integrator position

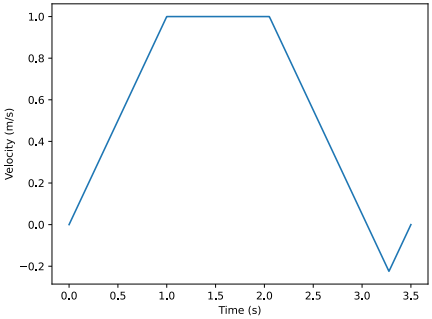


Figure 17.2: Double integrator velocity

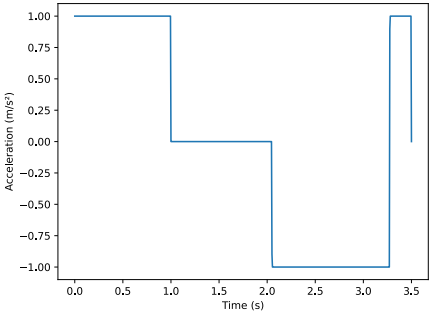


Figure 17.3: Double integrator acceleration

Other applications

In retrospect, the solution here seems obvious: if you want to reach the desired position in the minimum time, you just apply positive max input to accelerate to the max speed, coast for a while, then apply negative max input to decelerate to a stop at the desired position. Optimization problems can get more complex than this though. In fact, we can use this same framework to design optimal trajectories for a drivetrain while satisfying dynamics constraints, avoiding obstacles, and driving through points of interest.

17.1.2 Optimizing the problem formulation

Cost functions and constraints can have the following orders:

- none (i.e., there is no cost function or are no constraints)
- constant

- linear
- quadratic
- nonlinear

For nonlinear problems, the solver calculates the Hessian of the cost function and the Jacobians of the constraints at each iteration. However, problems with lower order cost functions and constraints can be solved faster. For example, the following only need to be computed once because they're constant:

- the Hessian of a quadratic or lower cost function
- the Jacobian of linear or lower constraints

A problem is constant if:

- the cost function is constant or lower
- the equality constraints are constant or lower
- the inequality constraints are constant or lower

A problem is linear if:

- the cost function is linear
- the equality constraints are linear or lower
- the inequality constraints are linear or lower

A problem is quadratic if:

- the cost function is quadratic
- the equality constraints are linear or lower
- the inequality constraints are linear or lower

All other problems are nonlinear.

17.2 Further reading

Sleipnir's authors recommend "Numerical Optimization, 2nd Ed." by Nocedal and Wright as a useful introductory resource for implementing optimization problem solvers [14]. A list of resources used to implement Sleipnir is located at https://sleipnirgroup.github.io/Sleipnir/md_docs_2resources.html.

Matthew Kelly has an approachable introduction to trajectory optimization on his website [7].

This page intentionally left blank

VI Appendices

This page intentionally left blank

A. Simplifying block diagrams

A.1 Cascaded blocks

$$Y = (P_1 P_2)X \tag{A.1}$$

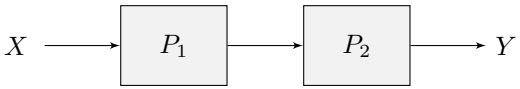


Figure A.1: Cascaded blocks

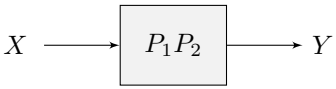


Figure A.2: Simplified cascaded blocks

A.2 Combining blocks in parallel

$$Y = P_1 X \pm P_2 X \tag{A.2}$$

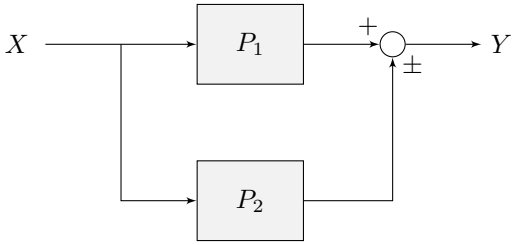


Figure A.3: Parallel blocks

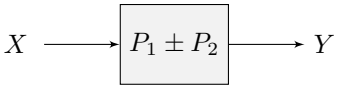


Figure A.4: Simplified parallel blocks

A.3 Removing a block from a feedforward loop

$$Y = P_1 X \pm P_2 X \tag{A.3}$$

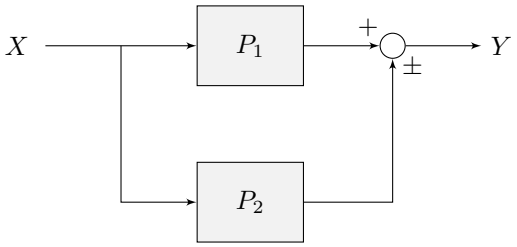


Figure A.5: Feedforward loop

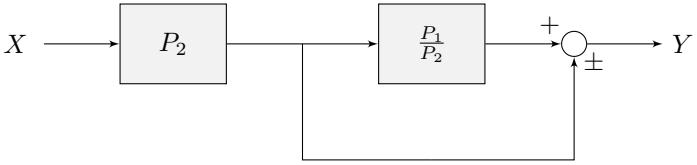


Figure A.6: Transformed feedforward loop

A.4 Eliminating a feedback loop

$$Y = P_1(X \mp P_2 Y) \tag{A.4}$$

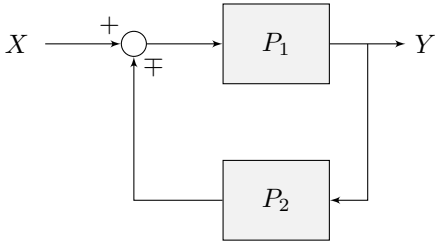


Figure A.7: Feedback loop

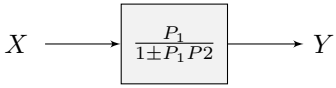


Figure A.8: Simplified feedback loop

A.5 Removing a block from a feedback loop

$$Y = P_1(X \mp P_2 Y) \tag{A.5}$$

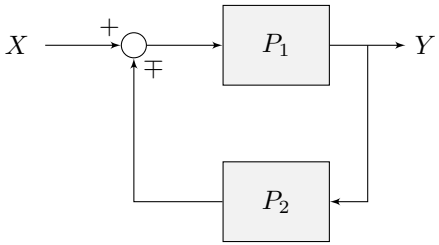


Figure A.9: Feedback loop

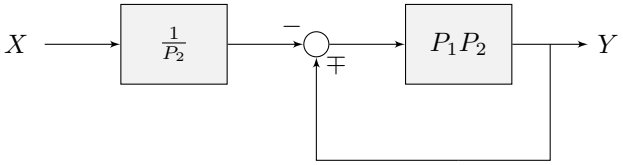


Figure A.10: Transformed feedback loop

B. Linear-quadratic regulator

This appendix will go into more detail on the linear-quadratic regulator's derivation and interesting applications.

B.1 Derivation

Let there be a discrete time linear system defined as

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{B.1}$$

with the cost functional

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{Q} & \mathbf{N} \\ \mathbf{N}^\mathsf{T} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$$

where J represents a trade-off between state excursion and control effort with the weighting factors $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{N}$. $\mathbf{Q}$ is the weight matrix for error, $\mathbf{R}$ is the weight matrix for control effort, and $\mathbf{N}$ is a cross weight matrix between error and control effort. $\mathbf{N}$ is commonly utilized when penalizing the output in addition to the state and input.

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{Q}\mathbf{x}_k + \mathbf{N}\mathbf{u}_k \\ \mathbf{N}^\mathsf{T}\mathbf{x}_k + \mathbf{R}\mathbf{u}_k \end{bmatrix}$$

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k^\mathsf{T} & \mathbf{u}_k^\mathsf{T} \end{bmatrix} \begin{bmatrix} \mathbf{Q}\mathbf{x}_k + \mathbf{N}\mathbf{u}_k \\ \mathbf{N}^\mathsf{T}\mathbf{x}_k + \mathbf{R}\mathbf{u}_k \end{bmatrix}$$

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T}(\mathbf{Q}\mathbf{x}_k + \mathbf{N}\mathbf{u}_k) + \mathbf{u}_k^\mathsf{T}(\mathbf{N}^\mathsf{T}\mathbf{x}_k + \mathbf{R}\mathbf{u}_k))$$

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{x}_k^\mathsf{T}\mathbf{N}\mathbf{u}_k + \mathbf{u}_k^\mathsf{T}\mathbf{N}^\mathsf{T}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k)$$

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{x}_k^\mathsf{T}\mathbf{N}\mathbf{u}_k + \mathbf{x}_k^\mathsf{T}\mathbf{N}\mathbf{u}_k^\mathsf{T} + \mathbf{u_k}^\mathsf{T}\mathbf{R}\mathbf{u}_k)$$

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + 2\mathbf{x}_k^\mathsf{T}\mathbf{N}\mathbf{u}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k)$$

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k + 2\mathbf{x}_k^\mathsf{T}\mathbf{N}\mathbf{u}_k)$$

The feedback control law which minimizes J subject to the constraint $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ is

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$$

where $\mathbf{K}$ is given by

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{B})^{-1}(\mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{A} + \mathbf{N}^\mathsf{T})$$

and $\mathbf{S}$ is found by solving the discrete time algebraic Riccati equation defined as

$$\mathbf{A}^\mathsf{T}\mathbf{S}\mathbf{A} - \mathbf{S} - (\mathbf{A}^\mathsf{T}\mathbf{S}\mathbf{B} + \mathbf{N})(\mathbf{R} + \mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{B})^{-1}(\mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{A} + \mathbf{N}^\mathsf{T}) + \mathbf{Q} = 0$$

or alternatively

$$\mathcal{A}^\mathsf{T}\mathbf{S}\mathcal{A} - \mathbf{S} - \mathcal{A}^\mathsf{T}\mathbf{S}\mathbf{B}(\mathbf{R} + \mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{S}\mathcal{A} + \mathbf{Q} = 0$$

with

$$\mathcal{A} = \mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{N}^\mathsf{T}$$
$$\mathcal{Q} = \mathbf{Q} - \mathbf{N}\mathbf{R}^{-1}\mathbf{N}^\mathsf{T}$$

If there is no cross-correlation between error and control effort, $\mathbf{N}$ is a zero matrix and the cost functional simplifies to

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^\mathsf{T} \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k)$$

The feedback control law which minimizes this J subject to $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ is

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$$

where $\mathbf{K}$ is given by

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{A}$$

and $\mathbf{S}$ is found by solving the discrete time algebraic Riccati equation defined as

$$\mathbf{A}^\mathsf{T}\mathbf{S}\mathbf{A} - \mathbf{S} - \mathbf{A}^\mathsf{T}\mathbf{S}\mathbf{B}(\mathbf{R} + \mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{S}\mathbf{A} + \mathbf{Q} = 0$$

Snippets B.1 and B.2 compute the infinite horizon, discrete time LQR.

```cpp
#include <Eigen/Cholesky>
#include <Eigen/Core>

#include "DARE.hpp"

template <int States, int Inputs>
Eigen::Matrix<double, Inputs, States> LQR(
    const Eigen::Matrix<double, States, States>& A,
    const Eigen::Matrix<double, States, Inputs>& B,
    const Eigen::Matrix<double, States, States>& Q,
    const Eigen::Matrix<double, Inputs, Inputs>& R,
    const Eigen::Matrix<double, States, Inputs>& N) {
  using StateMatrix = Eigen::Matrix<double, States, States>;

  auto R_llt = R.llt();
  StateMatrix A_2 = A - B * R_llt.solve(N.transpose());
  StateMatrix Q_2 = Q - N * R_llt.solve(N.transpose());

  StateMatrix S = DARE<States, Inputs>(A_2, B, Q_2, R);

  return (B.transpose() * S * B + R)
      .llt()
      .solve(B.transpose() * S * A + N.transpose());
}
```

Snippet B.1. Infinite horizon, discrete time LQR solver in C++ (see subsection 5.11.2 for DARE solver)

```python
"""Function for computing the infinite horizon LQR."""

import numpy as np
```

```
import scipy as sp


def lqr(A, B, Q, R, N):
    """Solves for the optimal discrete linear-quadratic regulator (LQR).

    Keyword arguments:
    A -- numpy.array(states x states), system matrix.
    B -- numpy.array(states x inputs), input matrix.
    Q -- numpy.array(states x states), state cost matrix.
    R -- numpy.array(inputs x inputs), control effort cost matrix.
    N -- numpy.array(states x inputs), cross weight matrix.

    Returns:
    K -- numpy.array(inputs x states), controller gain matrix.
    """
    P = sp.linalg.solve_discrete_are(a=A, b=B, q=Q, r=R, s=N)
    return np.linalg.solve(B.T @ P @ B + R, B.T @ P @ A + N.T)
```

Snippet B.2. Infinite horizon, discrete time LQR solver in Python

Other formulations of LQR for finite horizon and discrete time can be seen on Wikipedia [2].

MIT OpenCourseWare has a rigorous proof of the results shown above [17].

B.2 State feedback with output cost

LQR is normally used for state feedback on

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k\end{aligned}$$

with the cost functional

$$J = \sum_{k=0}^{\infty}(\mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k)$$

However, we may not know how to select costs for some of the states, or we don't care what certain internal states are doing. We can address this by writing the cost functional in terms of the output vector instead of the state vector. Not only can we make our output contain a subset of states, but we can use any other cost metric we can think of as long as it's representable as a linear combination of the states and inputs.[1]

[1]We'll see this later on in section B.3 when we define the cost metric as deviation from the behavior of another model.

For state feedback with an output cost, we want to minimize the following cost functional.

$$J = \sum_{k=0}^{\infty} (\mathbf{y}_k^\mathsf{T} \mathbf{Q} \mathbf{y}_k + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k)$$

Substitute in the expression for $\mathbf{y}_k$.

$$J = \sum_{k=0}^{\infty} ((\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k)^\mathsf{T} \mathbf{Q} (\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k) + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k)$$

Apply the transpose to the left-hand side of the $\mathbf{Q}$ term.

$$J = \sum_{k=0}^{\infty} ((\mathbf{x}_k^\mathsf{T} \mathbf{C}^\mathsf{T} + \mathbf{u}_k^\mathsf{T} \mathbf{D}^\mathsf{T}) \mathbf{Q} (\mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k) + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k)$$

Factor out $\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T}$ from the left side and $\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$ from the right side of each term.

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{C}^\mathsf{T} \\ \mathbf{D}^\mathsf{T} \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} + \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \left(\begin{bmatrix} \mathbf{C}^\mathsf{T} \\ \mathbf{D}^\mathsf{T} \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{C} & \mathbf{D} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Multiply in $\mathbf{Q}$.

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \left(\begin{bmatrix} \mathbf{C}^\mathsf{T}\mathbf{Q} \\ \mathbf{D}^\mathsf{T}\mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{C} & \mathbf{D} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Multiply matrices in the left term together.

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \left(\begin{bmatrix} \mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{C} & \mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{D} \\ \mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{C} & \mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{D} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Add the terms together.

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \underbrace{\mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{C}}_{\mathbf{Q}} & \underbrace{\mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{D}}_{\mathbf{N}} \\ \underbrace{\mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{C}}_{\mathbf{N}^\mathsf{T}} & \underbrace{\mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{D} + \mathbf{R}}_{\mathbf{R}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \tag{B.2}$$

Thus, state feedback with an output cost can be defined as the following optimization problem.

Theorem B.2.1 — Linear-quadratic regulator with output cost.

$$\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k} \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \underbrace{\mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{C}}_{\mathbf{Q}} & \underbrace{\mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{D}}_{\mathbf{N}} \\ \underbrace{\mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{C}}_{\mathbf{N}^\mathsf{T}} & \underbrace{\mathbf{D}^\mathsf{T}\mathbf{Q}\mathbf{D} + \mathbf{R}}_{\mathbf{R}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$$
$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{B.3}$$

The optimal control policy $\mathbf{u}_k^*$ is $\mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)$ where $\mathbf{r}_k$ is the desired state. Note that the $\mathbf{Q}$ in $\mathbf{C}^\mathsf{T}\mathbf{Q}\mathbf{C}$ is outputs $\times$ outputs instead of states $\times$ states. $\mathbf{K}$ can be computed via the typical LQR equations based on the algebraic Ricatti equation.

If the output is just the state vector, then $\mathbf{C} = \mathbf{I}$, $\mathbf{D} = \mathbf{0}$, and the cost functional simplifies to that of LQR with a state cost.

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$$

B.3 Implicit model following

If we want to design a feedback controller that erases the dynamics of our system and makes it behave like some other system, we can use *implicit model following*. This is used on the Blackhawk helicopter at NASA Ames research center when they want to make it fly like experimental aircraft (within the limits of the helicopter's actuators, of course).

B.3.1 Following reference system matrix

Let the original system dynamics be

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k \end{aligned}$$

and the desired system dynamics be $\mathbf{z}_{k+1} = \mathbf{A}_{ref}\mathbf{z}_k$.

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{C}\mathbf{x}_{k+1} \\ \mathbf{y}_{k+1} &= \mathbf{C}(\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k) \\ \mathbf{y}_{k+1} &= \mathbf{C}\mathbf{A}\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k \end{aligned}$$

We want to minimize the following cost functional.

$$J = \sum_{k=0}^{\infty} \left((\mathbf{y}_{k+1} - \mathbf{z}_{k+1})^{\mathsf{T}} \mathbf{Q} (\mathbf{y}_{k+1} - \mathbf{z}_{k+1}) + \mathbf{u}_k^{\mathsf{T}} \mathbf{R} \mathbf{u}_k \right)$$

We'll be measuring the desired system's state, so let $\mathbf{y} = \mathbf{z}$.

$$\begin{aligned} \mathbf{z}_{k+1} &= \mathbf{A}_{ref} \mathbf{y}_k \\ \mathbf{z}_{k+1} &= \mathbf{A}_{ref} \mathbf{C} \mathbf{x}_k \end{aligned}$$

Therefore,

$$\begin{aligned} \mathbf{y}_{k+1} - \mathbf{z}_{k+1} &= \mathbf{C}\mathbf{A}\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k - (\mathbf{A}_{ref}\mathbf{C}\mathbf{x}_k) \\ \mathbf{y}_{k+1} - \mathbf{z}_{k+1} &= (\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k \end{aligned}$$

Substitute this into the cost functional.

$$\begin{aligned} J &= \sum_{k=0}^{\infty} \left((\mathbf{y}_{k+1} - \mathbf{z}_{k+1})^{\mathsf{T}} \mathbf{Q} (\mathbf{y}_{k+1} - \mathbf{z}_{k+1}) + \mathbf{u}_k^{\mathsf{T}} \mathbf{R} \mathbf{u}_k \right) \\ J &= \sum_{k=0}^{\infty} \big((((\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k)^{\mathsf{T}} \mathbf{Q} ((\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k) \\ &\quad + \mathbf{u}_k^{\mathsf{T}} \mathbf{R} \mathbf{u}_k \big) \end{aligned}$$

Apply the transpose to the left-hand side of the $\mathbf{Q}$ term.

$$\begin{aligned} J = \sum_{k=0}^{\infty} \big(&(\mathbf{x}_k^{\mathsf{T}} (\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}} + \mathbf{u}_k^{\mathsf{T}} (\mathbf{C}\mathbf{B})^{\mathsf{T}}) \mathbf{Q} ((\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k) \\ &+ \mathbf{u}_k^{\mathsf{T}} \mathbf{R} \mathbf{u}_k \big) \end{aligned}$$

Factor out $\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}}$ from the left side and $\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$ from the right side of each term.

$$\begin{aligned} J = \sum_{k=0}^{\infty} \Bigg(&\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} (\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}} \\ (\mathbf{C}\mathbf{B})^{\mathsf{T}} \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C} & \mathbf{C}\mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \Bigg) \end{aligned}$$

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} (\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}} \\ (\mathbf{CB})^{\mathsf{T}} \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{CA} - \mathbf{A}_{ref}\mathbf{C} & \mathbf{CB} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Multiply in $\mathbf{Q}$.

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} (\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}}\mathbf{Q} \\ (\mathbf{CB})^{\mathsf{T}}\mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{CA} - \mathbf{A}_{ref}\mathbf{C} & \mathbf{CB} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Multiply matrices in the left term together.

$$J = \sum_{k=0}^{\infty} \left(\begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} (\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}}\mathbf{Q}(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C}) & (\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}}\mathbf{Q}(\mathbf{CB}) \\ (\mathbf{CB})^{\mathsf{T}}\mathbf{Q}(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C}) & (\mathbf{CB})^{\mathsf{T}}\mathbf{Q}(\mathbf{CB}) \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \right) \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right)$$

Add the terms together.

$$J = \sum_{k=0}^{\infty} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \underbrace{(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}}\mathbf{Q}(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})}_{\mathbf{Q}} & \underbrace{(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})^{\mathsf{T}}\mathbf{Q}(\mathbf{CB})}_{\mathbf{N}} \\ \underbrace{(\mathbf{CB})^{\mathsf{T}}\mathbf{Q}(\mathbf{CA} - \mathbf{A}_{ref}\mathbf{C})}_{\mathbf{N}^{\mathsf{T}}} & \underbrace{(\mathbf{CB})^{\mathsf{T}}\mathbf{Q}(\mathbf{CB}) + \mathbf{R}}_{\mathbf{R}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \tag{B.4}$$

Thus, implicit model following can be defined as the following optimization problem.

Theorem B.3.1 — Implicit model following.

$$\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k} \sum_{k=0}^{\infty} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}^\mathsf{T} \begin{bmatrix}\mathbf{Q}_{imf} & \mathbf{N}_{imf} \\ \mathbf{N}_{imf}^\mathsf{T} & \mathbf{R}_{imf}\end{bmatrix} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}$$
$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{B.5}$$

where

$$\begin{aligned}\mathbf{Q}_{imf} &= (\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})^\mathsf{T}\mathbf{Q}(\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C}) \\ \mathbf{N}_{imf} &= (\mathbf{C}\mathbf{A} - \mathbf{A}_{ref}\mathbf{C})^\mathsf{T}\mathbf{Q}(\mathbf{C}\mathbf{B}) \\ \mathbf{R}_{imf} &= (\mathbf{C}\mathbf{B})^\mathsf{T}\mathbf{Q}(\mathbf{C}\mathbf{B}) + \mathbf{R}\end{aligned}$$

The optimal control policy $\mathbf{u}_k^*$ is $-\mathbf{K}\mathbf{x}_k$. $\mathbf{K}$ can be computed via the typical LQR equations based on the algebraic Ricatti equation.

The control law $\mathbf{u}_{imf,k} = -\mathbf{K}\mathbf{x}_k$ makes $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_{imf,k}$ match the open-loop response of $\mathbf{z}_{k+1} = \mathbf{A}_{ref}\mathbf{z}_k$.

If the original and desired system have the same states, then $\mathbf{C} = \mathbf{I}$ and the cost functional simplifies to

$$J = \sum_{k=0}^{\infty} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}^\mathsf{T} \begin{bmatrix}\underbrace{(\mathbf{A} - \mathbf{A}_{ref})^\mathsf{T}\mathbf{Q}(\mathbf{A} - \mathbf{A}_{ref})}_{\mathbf{Q}} & \underbrace{(\mathbf{A} - \mathbf{A}_{ref})^\mathsf{T}\mathbf{Q}\mathbf{B}}_{\mathbf{N}} \\ \underbrace{\mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{A} - \mathbf{A}_{ref})}_{\mathbf{N}^\mathsf{T}} & \underbrace{\mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B} + \mathbf{R}}_{\mathbf{R}}\end{bmatrix} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix} \tag{B.6}$$

B.3.2 Following reference input matrix

The feedback control law above makes the open-loop system behave like $\mathbf{A}_{ref}$, but the input dynamics are still that of the original system. Here's how to make the input dynamics behave like $\mathbf{B}_{ref}$. We want to find the $\mathbf{u}_{imf,k}$ that makes the real model follow the reference model.

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_{imf,k} \\ \mathbf{z}_{k+1} &= \mathbf{A}_{ref}\mathbf{z}_k + \mathbf{B}_{ref}\mathbf{u}_k\end{aligned}$$

Let $\mathbf{x} = \mathbf{z}$.

$$\mathbf{x}_{k+1} = \mathbf{z}_{k+1}$$

$$
\begin{aligned}
\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_{imf,k} &= \mathbf{A}_{ref}\mathbf{x}_k + \mathbf{B}_{ref}\mathbf{u}_k \\
\mathbf{B}\mathbf{u}_{imf,k} &= \mathbf{A}_{ref}\mathbf{x}_k - \mathbf{A}\mathbf{x}_k + \mathbf{B}_{ref}\mathbf{u}_k \\
\mathbf{B}\mathbf{u}_{imf,k} &= (\mathbf{A}_{ref} - \mathbf{A})\mathbf{x}_k + \mathbf{B}_{ref}\mathbf{u}_k \\
\mathbf{u}_{imf,k} &= \mathbf{B}^{+}((\mathbf{A}_{ref} - \mathbf{A})\mathbf{x}_k + \mathbf{B}_{ref}\mathbf{u}_k) \\
\mathbf{u}_{imf,k} &= -\mathbf{B}^{+}(\mathbf{A} - \mathbf{A}_{ref})\mathbf{x}_k + \mathbf{B}^{+}\mathbf{B}_{ref}\mathbf{u}_k
\end{aligned}
$$

The first term makes the open-loop poles match that of the reference model, and the second term makes the input behave like that of the reference model.

B.4 Time delay compensation

Linear-Quadratic regulator controller gains tend to be aggressive. If sensor measurements are time-delayed too long, the LQR may be unstable (see figure B.1). However, if we know the amount of delay, we can compute the control based on where the system will be after the time delay.

We can compensate for the time delay if we know the control law we're applying in future timesteps ($\mathbf{u} = -\mathbf{K}\mathbf{x}$) and the duration of the time delay. To get the true state at the current time for control purposes, we project our delayed state forward by the time delay using our model and the aforementioned control law. Figure B.2 shows improved control with the predicted state.[2]

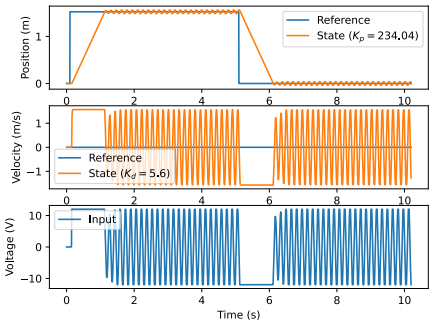


Figure B.1: Elevator response at 5 ms sample period with 50 ms of output lag

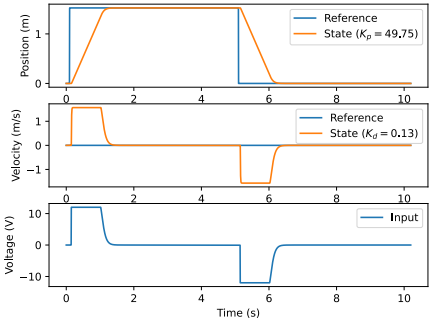


Figure B.2: Elevator response at 5 ms sample period with 50 ms of output lag (delay-compensated)

[2] Input delay and output delay have the same effect on the system, so the time delay can be simulated with either an input delay buffer or a measurement delay buffer.

For steady-state controller gains, this method of delay compensation seems to work better for second-order systems than first-order systems. Figures B.3 and B.5 show time delay for a drivetrain velocity system and flywheel system respectively. Figures B.4 and B.6 show that compensating the controller gain significantly reduces the feedback gain. For systems with fast dynamics and a long time delay, the delay-compensated controller has an almost open-loop response because only feedforward has a significant effect; this has poor disturbance rejection. Fixing the source of the time delay is always preferred for these systems.

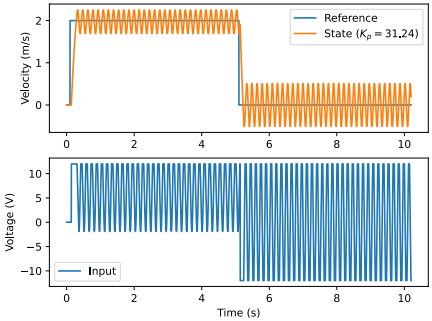


Figure B.3: Drivetrain velocity response at 1 ms sample period with 40 ms of output lag

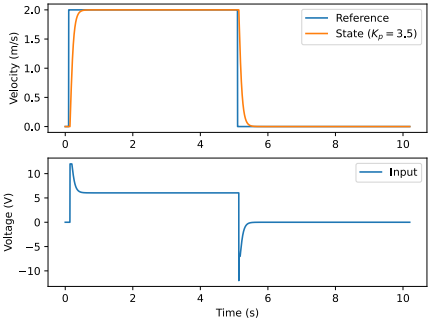


Figure B.4: Drivetrain velocity response at 1 ms sample period with 40 ms of output lag (delay-compensated)

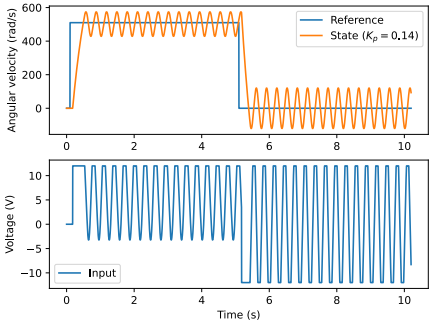


Figure B.5: Flywheel response at 1 ms sample period with 80 ms of output lag

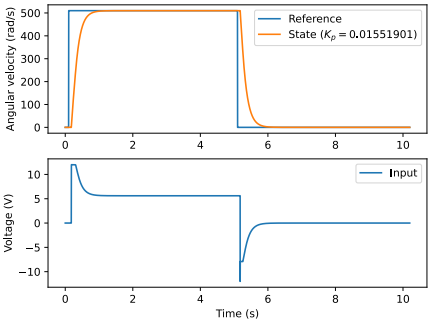


Figure B.6: Flywheel response at 1 ms sample period with 80 ms of output lag (delay-compensated)

Since we are computing the control based on future states and the state exponentially

converges to zero over time, the control action we apply at the current timestep also converges to zero for longer time delays. During startup, the inputs we use to predict the future state are zero because there's initially no input history. This means the initial inputs are larger to give the system a kick in the right direction. As the input delay buffer fills up, the controller gain converges to a smaller steady-state value. If one uses the steady-state controller gain during startup, the transient response may be slow.

All figures shown here use the steady-state control law (the second case in equation (B.12)).

We'll show how to derive this controller gain compensation for continuous and discrete systems.

B.4.1 Continuous case

The undelayed continuous linear system is defined as $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$ with the controller $\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$. Let L be the amount of time delay in seconds. We can avoid the time delay if we compute the control based on the plant L seconds in the future.

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t + L)$$

We need to find $\mathbf{x}(t + L)$ given $\mathbf{x}(t)$. Since we know $\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$ will be applied over the time interval $[t, t + L)$, substitute it into the continuous model.

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}(-\mathbf{K}\mathbf{x}(t)) \\
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x}(t) - \mathbf{B}\mathbf{K}\mathbf{x}(t) \\
\dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}(t)
\end{aligned}$$

We now have a differential equation for the closed-loop system dynamics. Take the matrix exponential from the current time t to L in the future to obtain $\mathbf{x}(t + L)$.

$$\mathbf{x}(t + L) = e^{(\mathbf{A}-\mathbf{B}\mathbf{K})L}\mathbf{x}(t) \tag{B.7}$$

This works for $t \geq L$, but if $t < L$, we have no input history for the time interval $[t, L)$. If we assume the inputs for $[t, L)$ are zero, the state prediction for that interval is

$$\mathbf{x}(L) = e^{\mathbf{A}(L-t)}\mathbf{x}(t)$$

The time interval $[0, t)$ has nonzero inputs since it's in the past and was using the normal control law.

$$\mathbf{x}(t + L) = e^{(\mathbf{A}-\mathbf{B}\mathbf{K})t}\mathbf{x}(L)$$

$$\mathbf{x}(t+L) = e^{(\mathbf{A}-\mathbf{BK})t}e^{\mathbf{A}(L-t)}\mathbf{x}(t) \tag{B.8}$$

Therefore, equations (B.7) and (B.8) give the latency-compensated control law for all $t \geq 0$.

$$\begin{aligned}\mathbf{u}(t) &= -\mathbf{K}\mathbf{x}(t+L)\\ \mathbf{u}(t) &= \begin{cases} -\mathbf{K}e^{(\mathbf{A}-\mathbf{BK})t}e^{\mathbf{A}(L-t)}\mathbf{x}(t) & \text{if } 0 \leq t < L\\ -\mathbf{K}e^{(\mathbf{A}-\mathbf{BK})L}\mathbf{x}(t) & \text{if } t \geq L \end{cases}\end{aligned} \tag{B.9}$$

B.4.2 Discrete case

The undelayed discrete linear system is defined as $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ with the controller $\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$. Let L be the amount of time delay in seconds. We can avoid the time delay if we compute the control based on the plant L seconds in the future.

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_{k+L/T}$$

We need to find $\mathbf{x}_{k+L/T}$ given $\mathbf{x}_k$. Since we know $\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$ will be applied for the timesteps k through $k + \frac{L}{T}$, substitute it into the discrete model.

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k\\ \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}(-\mathbf{K}\mathbf{x}_k)\\ \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k - \mathbf{B}\mathbf{K}\mathbf{x}_k\\ \mathbf{x}_{k+1} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k\end{aligned}$$

Let T be the duration between timesteps in seconds and L be the amount of time delay in seconds. $\frac{L}{T}$ gives the number of timesteps represented by L.

$$\mathbf{x}_{k+L/T} = (\mathbf{A} - \mathbf{B}\mathbf{K})^{\frac{L}{T}}\mathbf{x}_k \tag{B.10}$$

This works for $kT \geq L$ where kT is the current time, but if $kT < L$, we have no input history for the time interval $[kT, L)$. If we assume the inputs for $[kT, L)$ are zero, the state prediction for that interval is

$$\begin{aligned}\mathbf{x}_{L/T} &= \mathbf{A}^{\frac{L-kT}{T}}\mathbf{x}_k\\ \mathbf{x}_{L/T} &= \mathbf{A}^{\frac{L}{T}-k}\mathbf{x}_k\end{aligned}$$

The time interval $[0, kT)$ has nonzero inputs since it's in the past and was using the normal control law.

$$\begin{aligned}\mathbf{x}_{k+L/T} &= (\mathbf{A} - \mathbf{BK})^k \mathbf{x}_{L/T} \\ \mathbf{x}_{k+L/T} &= (\mathbf{A} - \mathbf{BK})^k \mathbf{A}^{\frac{L}{T}-k} \mathbf{x}_k\end{aligned} \tag{B.11}$$

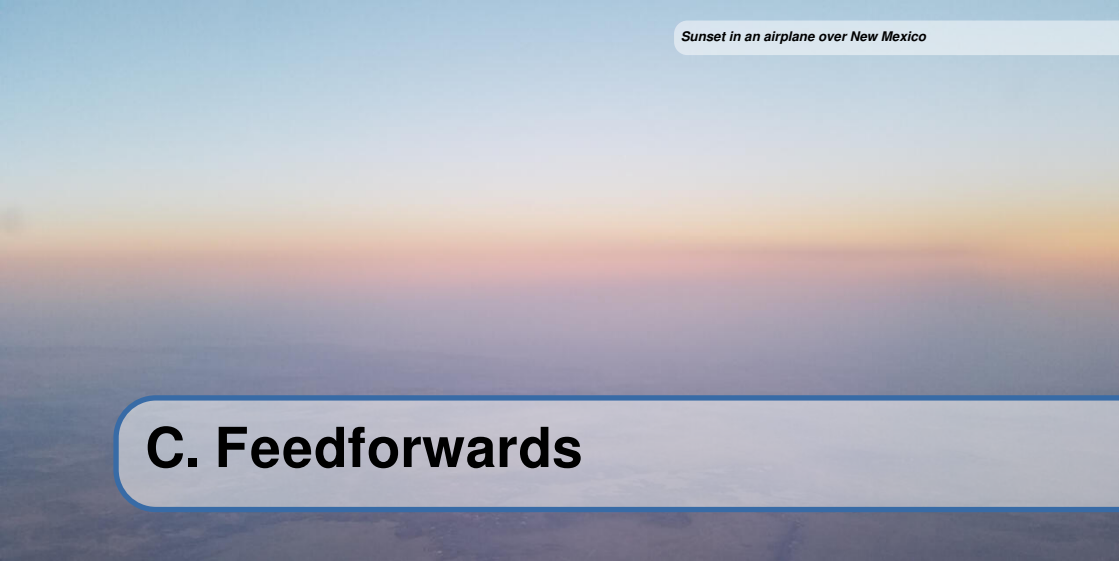
Therefore, equations (B.10) and (B.11) give the latency-compensated control law for all $t \geq 0$.

$$\begin{aligned}\mathbf{u}_k &= -\mathbf{K}\mathbf{x}_{k+L/T} \\ \mathbf{u}_k &= \begin{cases} -\mathbf{K}(\mathbf{A} - \mathbf{BK})^k \mathbf{A}^{\frac{L}{T}-k} \mathbf{x}_k & \text{if } 0 \leq k < \frac{L}{T} \\ -\mathbf{K}(\mathbf{A} - \mathbf{BK})^{\frac{L}{T}} \mathbf{x}_k & \text{if } k \geq \frac{L}{T} \end{cases}\end{aligned} \tag{B.12}$$

If the delay L isn't a multiple of the sample period T in equation (B.12), we have to evaluate fractional matrix powers, which can be tricky. Eigen (a C++ library) supports fractional powers with the `pow()` member function provided by <unsupported/Eigen/MatrixFunctions>. SciPy (a Python library) supports fractional powers with the free function scipy.linalg.fractional_matrix_power(). If the language you're using doesn't provide such a function, you can try the following approach instead.

Let there be a matrix $\mathbf{M}$ raised to a fractional power n. If $\mathbf{M}$ is diagonalizable, we can obtain an exact answer for $\mathbf{M}^n$ by decomposing $\mathbf{M}$ into $\mathbf{PDP}^{-1}$ where $\mathbf{D}$ is a diagonal matrix, computing $\mathbf{D}^n$ as each diagonal element raised to n, then recomposing $\mathbf{M}^n$ as $\mathbf{PD}^n\mathbf{P}^{-1}$.

If a matrix raised to a fractional power in equation (B.12) isn't diagonalizable, we have to approximate by rounding $\frac{L}{T}$ to the nearest integer. This approximation gets worse as $L \bmod T$ approaches $\frac{T}{2}$.



C. Feedforwards

This appendix will show the derivations for alternate feedforward formulations.

C.1 QR-weighted linear plant inversion

C.1.1 Setup

Let's start with the equation for the reference dynamics

$$\mathbf{r}_{k+1} = \mathbf{A}\mathbf{r}_k + \mathbf{B}\mathbf{u}_k$$

where $\mathbf{u}_k$ is the feedforward input. Note that this feedforward equation does not and should not take into account any feedback terms. We want to find the optimal $\mathbf{u}_k$ such that we minimize the tracking error between $\mathbf{r}_{k+1}$ and $\mathbf{r}_k$.

$$\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k = \mathbf{B}\mathbf{u}_k$$

To solve for $\mathbf{u}_k$, we need to take the inverse of the nonsquare matrix $\mathbf{B}$. This isn't possible, but we can find the pseudoinverse given some constraints on the state tracking error and control effort. To find the optimal solution for these sorts of trade-offs, one can define a cost function and attempt to minimize it. To do this, we'll first solve the expression for $\mathbf{0}$.

$$\mathbf{0} = \mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)$$

This expression will be the state tracking cost we use in our cost function.

Our cost function will use an H_2 norm with $\mathbf{Q}$ as the state cost matrix with dimensionality $states \times states$ and $\mathbf{R}$ as the control input cost matrix with dimensionality $inputs \times inputs$.

$$\mathbf{J} = (\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k))^\mathsf{T}\mathbf{Q}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k$$

C.1.2 Minimization

Given theorem 5.12.1 and corollary 5.12.3, find the minimum of $\mathbf{J}$ by taking the partial derivative with respect to $\mathbf{u}_k$ and setting the result to $\mathbf{0}$.

$$\begin{aligned}
\frac{\partial \mathbf{J}}{\partial \mathbf{u}_k} &= 2\mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) + 2\mathbf{R}\mathbf{u}_k \\
\mathbf{0} &= 2\mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) + 2\mathbf{R}\mathbf{u}_k \\
\mathbf{0} &= \mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{B}\mathbf{u}_k - (\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)) + \mathbf{R}\mathbf{u}_k \\
\mathbf{0} &= \mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B}\mathbf{u}_k - \mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) + \mathbf{R}\mathbf{u}_k \\
\mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B}\mathbf{u}_k + \mathbf{R}\mathbf{u}_k &= \mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \\
(\mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B} + \mathbf{R})\mathbf{u}_k &= \mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \\
\mathbf{u}_k &= (\mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^\mathsf{T}\mathbf{Q}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)
\end{aligned}$$

Theorem C.1.1 — QR-weighted linear plant inversion. Given the discrete model $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$, the plant inversion feedforward is

$$\mathbf{u}_k = \mathbf{K}_{ff}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k)$$

where $\mathbf{K}_{ff} = (\mathbf{B}^\mathsf{T}\mathbf{Q}\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^\mathsf{T}\mathbf{Q}$, $\mathbf{r}_{k+1}$ is the reference at the next timestep, and $\mathbf{r}_k$ is the reference at the current timestep.

Figure C.1 shows plant inversion applied to a second-order CIM motor model.

Plant inversion isn't as effective with both $\mathbf{Q}$ and $\mathbf{R}$ cost because the $\mathbf{R}$ matrix penalized control effort. The reference tracking with no cost matrices is much better.

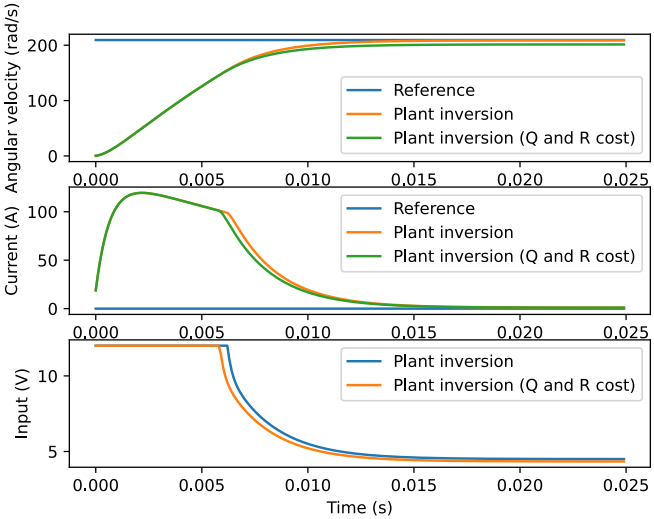


Figure C.1: Second-order CIM motor response with plant inversion

C.2 Steady-state feedforward

Steady-state feedforwards apply the control effort required to keep a system at the reference if it is no longer moving (i.e., the system is at steady-state). The first steady-state feedforward converts desired outputs to desired states.

$$\mathbf{x}_c = \mathbf{N}_x \mathbf{y}_c$$

$\mathbf{N}_x$ converts desired outputs $\mathbf{y}_c$ to desired states $\mathbf{x}_c$ (also known as $\mathbf{r}$). For steady-state, that is

$$\mathbf{x}_{ss} = \mathbf{N}_x \mathbf{y}_{ss} \tag{C.1}$$

The second steady-state feedforward converts the desired outputs $\mathbf{y}$ to the control input required at steady-state.

$$\mathbf{u}_c = \mathbf{N}_u \mathbf{y}_c$$

$\mathbf{N}_u$ converts the desired outputs $\mathbf{y}$ to the control input $\mathbf{u}$ required at steady-state. For steady-state, that is

$$\mathbf{u}_{ss} = \mathbf{N}_u \mathbf{y}_{ss} \tag{C.2}$$

C.2.1 Continuous case

To find the control input required at steady-state, set equation (6.3) to zero.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= \mathbf{A}\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= \mathbf{A}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{B}\mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{D}\mathbf{N}_u\mathbf{y}_{ss}\end{aligned}$$

$$\begin{aligned}\begin{bmatrix}\mathbf{0} \\ \mathbf{y}_{ss}\end{bmatrix} &= \begin{bmatrix}\mathbf{A}\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix}\mathbf{y}_{ss} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A}\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} &= \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^+\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix}\end{aligned}$$

where $^+$ denotes the Moore-Penrose pseudoinverse.

C.2.2 Discrete case

Now, we'll do the same thing for the discrete system. To find the control input required at steady-state, set $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ to zero.

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k\end{aligned}$$

$$\begin{aligned}\mathbf{x}_{ss} &= \mathbf{A}\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= (\mathbf{A} - \mathbf{I})\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= (\mathbf{A} - \mathbf{I})\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{B}\mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{D}\mathbf{N}_u\mathbf{y}_{ss}\end{aligned}$$

$$\begin{aligned}\begin{bmatrix}\mathbf{0} \\ \mathbf{y}_{ss}\end{bmatrix} &= \begin{bmatrix}(\mathbf{A} - \mathbf{I})\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix}\mathbf{y}_{ss} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}(\mathbf{A} - \mathbf{I})\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} &= \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^+\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix}\end{aligned}$$

where $^+$ denotes the Moore-Penrose pseudoinverse.

C.2.3 Deriving steady-state input

Now, we'll find an expression that uses $\mathbf{N}_x$ and $\mathbf{N}_u$ to convert the reference $\mathbf{r}$ to a control input feedforward $\mathbf{u}_{ff}$. Let's start with equation (C.1).

$$\begin{aligned}\mathbf{x}_{ss} &= \mathbf{N}_x\mathbf{y}_{ss} \\ \mathbf{N}_x^+\mathbf{x}_{ss} &= \mathbf{y}_{ss}\end{aligned}$$

Now substitute this into equation (C.2).

$$\begin{aligned}\mathbf{u}_{ss} &= \mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{u}_{ss} &= \mathbf{N}_u(\mathbf{N}_x^+\mathbf{x}_{ss}) \\ \mathbf{u}_{ss} &= \mathbf{N}_u\mathbf{N}_x^+\mathbf{x}_{ss}\end{aligned}$$

$\mathbf{u}_{ss}$ and $\mathbf{x}_{ss}$ are also known as $\mathbf{u}_{ff}$ and $\mathbf{r}$ respectively.

$$\mathbf{u}_{ff} = \mathbf{N}_u\mathbf{N}_x^+\mathbf{r}$$

So all together, we get theorem C.2.1.

Theorem C.2.1 — Steady-state feedforward.

Continuous:

$$\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^+ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} \tag{C.3}$$

$$\mathbf{u}_{ff} = \mathbf{N}_u\mathbf{N}_x^+\mathbf{r} \tag{C.4}$$

Discrete:

$$\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^+ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} \tag{C.5}$$

$$\mathbf{u}_{ff,k} = \mathbf{N}_u\mathbf{N}_x^+\mathbf{r}_k \tag{C.6}$$

In the augmented matrix, $\mathbf{B}$ should contain one column corresponding to an actuator and $\mathbf{C}$ should contain one row whose output will be driven by that actuator. More than one actuator or output can be included in the computation at once, but the result won't be the same as if they were computed independently and summed afterward.

After computing the feedforward for each actuator-output pair, the respective collections of $\mathbf{N}_x$ and $\mathbf{N}_u$ matrices can summed to produce the combined feedforward.

If the augmented matrix in theorem C.2.1 is square (number of inputs = number of outputs), the normal matrix inverse can be used instead.

Case study: second-order CIM motor model

Each feedforward implementation has advantages. The steady-state feedforward allows using specific actuators to maintain the reference at steady-state. Plant inversion doesn't support this, but can be used for reference tracking as well with the same tuning parameters as LQR design. Figure C.2 shows both types of feedforwards applied to a second-order CIM motor model.

Plant inversion isn't as effective in figure C.2 because the $\mathbf{R}$ matrix penalized control effort. If the $\mathbf{R}$ cost matrix is removed from the plant inversion calculation, the reference tracking is much better (see figure C.3).

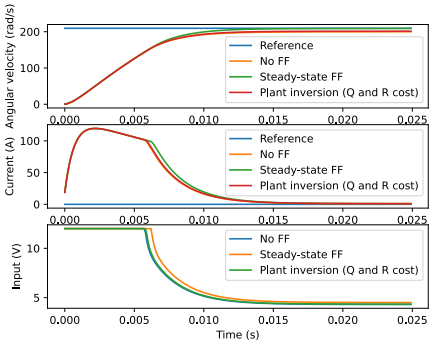


Figure C.2: Second-order CIM motor response with various feedforwards

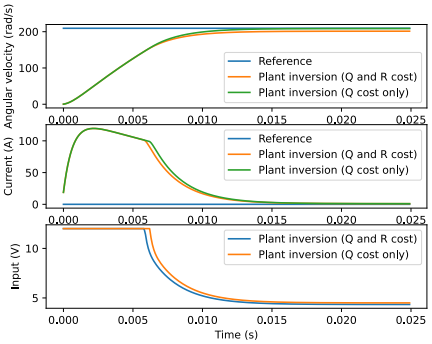
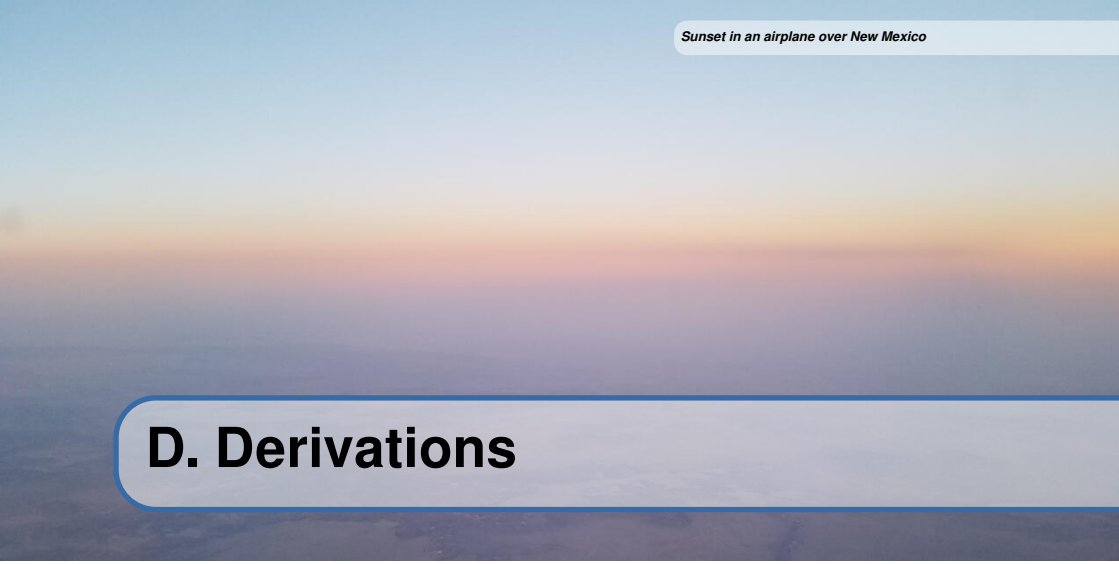


Figure C.3: Second-order CIM motor response with plant inversions

This page intentionally left blank



D. Derivations

D.1 Linear system zero-order hold

Starting with the continuous model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

by premultiplying the model by $e^{-\mathbf{A}t}$, we get

$$e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) = e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t) + e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}(t)$$
$$e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) - e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t) = e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}(t)$$

The derivative of the matrix exponential is

$$\frac{d}{dt}e^{\mathbf{A}t} = \mathbf{A}e^{\mathbf{A}t} = e^{\mathbf{A}t}\mathbf{A}$$

so we recognize the previous equation as

$$\frac{d}{dt}\left(e^{-\mathbf{A}t}\mathbf{x}(t)\right) = e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}(t)$$

By integrating this equation, we get

$$e^{-\mathbf{A}t}\mathbf{x}(t) - e^{0}\mathbf{x}(0) = \int_0^t e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{u}(\tau)\,d\tau$$

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau$$

which is an analytical solution to the continuous model. Now we want to discretize it.

$$\begin{aligned}
\mathbf{x}_k &\stackrel{def}{=} \mathbf{x}(kT) \\
\mathbf{x}_k &= e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
&\quad + \int_{kT}^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
&\quad + \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\underbrace{\left(e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau\right)}_{\mathbf{x}_k} \\
&\quad + \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau
\end{aligned}$$

We assume that $\mathbf{u}$ is constant during each timestep, so it can be pulled out of the integral.

$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \left(\int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\,d\tau\right)\mathbf{B}\mathbf{u}_k$$

The second term can be simplified by substituting it with the function $v(\tau) = kT + T - \tau$. Note that $d\tau = -dv$.

$$\begin{aligned}
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k - \left(\int_{v(kT)}^{v((k+1)T)} e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k - \left(\int_T^0 e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k
\end{aligned}$$

$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \left(\int_0^T e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k$$
$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}e^{\mathbf{A}v}|_0^T\mathbf{B}\mathbf{u}_k$$
$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}(e^{\mathbf{A}T} - e^{\mathbf{A}0})\mathbf{B}\mathbf{u}_k$$
$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B}\mathbf{u}_k$$

which is an exact solution to the discretization problem.

D.2 Kalman filter as Luenberger observer

A Luenberger observer is defined as

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_k^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{D.1}$$
$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_k^{-} \tag{D.2}$$

where a superscript of minus denotes *a priori* and plus denotes *a posteriori* estimate. Combining equation (D.1) and equation (D.2) gives

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_k^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^{-}) \tag{D.3}$$

The following is a Kalman filter that considers the current update step and the next predict step together rather than the current predict step and current update step.

Update step

$$\mathbf{K}_k = \mathbf{P}_k^{-}\mathbf{C}^{\mathsf{T}}(\mathbf{C}\mathbf{P}_k^{-}\mathbf{C}^{\mathsf{T}} + \mathbf{R})^{-1} \tag{D.4}$$
$$\hat{\mathbf{x}}_k^{+} = \hat{\mathbf{x}}_k^{-} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^{-}) \tag{D.5}$$
$$\mathbf{P}_k^{+} = (\mathbf{I} - \mathbf{K}_k\mathbf{C})\mathbf{P}_k^{-} \tag{D.6}$$

Predict step

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_k^{+} + \mathbf{B}\mathbf{u}_k \tag{D.7}$$
$$\mathbf{P}_{k+1}^{-} = \mathbf{A}\mathbf{P}_k^{+}\mathbf{A}^{\mathsf{T}} + \mathbf{Q} \tag{D.8}$$

Substitute equation (D.5) into equation (D.7).

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}(\hat{\mathbf{x}}_k^{-} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^{-})) + \mathbf{B}\mathbf{u}_k$$
$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\mathbf{x}_k^{-} + \mathbf{A}\mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^{-}) + \mathbf{B}\mathbf{u}_k$$

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{A}\mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k}^{-})$$

Let $\mathbf{L} = \mathbf{A}\mathbf{K}_k$.

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k}^{-}) \tag{D.9}$$

which matches equation (D.3). Therefore, the eigenvalues of the Kalman filter observer can be obtained by

$$\begin{aligned} &\text{eig}(\mathbf{A} - \mathbf{L}\mathbf{C}) \\ &\text{eig}(\mathbf{A} - (\mathbf{A}\mathbf{K}_k)(\mathbf{C})) \\ &\text{eig}(\mathbf{A}(\mathbf{I} - \mathbf{K}_k\mathbf{C})) \end{aligned} \tag{D.10}$$

D.2.1 Luenberger observer with separate prediction and update

To run a Luenberger observer with separate prediction and update steps, substitute the relationship between the Luenberger observer and Kalman filter matrices derived above into the Kalman filter equations.

Appendix D.2 shows that $\mathbf{L} = \mathbf{A}\mathbf{K}_k$. Since $\mathbf{L}$ and $\mathbf{A}$ are constant, one must assume $\mathbf{K}_k$ has reached steady-state. Then, $\mathbf{K} = \mathbf{A}^{-1}\mathbf{L}$. Substitute this into the Kalman filter update equation.

$$\begin{aligned} \hat{\mathbf{x}}_{k+1}^{+} &= \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-}) \\ \hat{\mathbf{x}}_{k+1}^{+} &= \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-}) \end{aligned}$$

Substitute in equation (9.4).

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1})$$

The predict step is the same as the Kalman filter's. Therefore, a Luenberger observer run with prediction and update steps is written as follows.

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k \tag{D.11}$$

Update step

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1}) \tag{D.12}$$

$$\hat{\mathbf{y}}_{k+1} = \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-} \tag{D.13}$$

E. Classical control theory

This appendix is provided for those who are curious about a lower-level interpretation of control systems. It describes what a transfer function is and shows how they can be used to analyze dynamical systems. Emphasis is placed on the geometric intuition of this analysis rather than the frequency domain math. Many tools exclusive to classical control theory (root locus, Bode plots, Nyquist plots, etc.) aren't useful for or relevant to the examples presented in the main chapters, so they would only complicate the learning process.

With classical control theory's geometric interpretation, one can perform stability and robustness analyses and design reasonable controllers for systems on the back of a napkin. However, computing power is much more plentiful nowadays; we should take advantage of the modern tools this enables to better express the controls designer's intent (for example, via optimal control criteria rather than frequency response characteristics).

Classical control theory should only be used when the controls designer cares about directly shaping the frequency response of a system. Electrical engineers do this a lot, but all the other university controls students have been forced to learn it too regardless of its utility in other disciplines.

E.1 Classical vs modern control theory

State-space notation provides a more convenient and compact way to model and analyze systems with multiple inputs and outputs. For a system with p inputs and q outputs, we would have to write $q \times p$ transfer functions to represent it. Not only is the resulting algebra unwieldy, but it only works for linear systems. Including nonzero initial conditions complicates the algebra even more. State-space representation uses the time domain instead of the Laplace domain, so it can model nonlinear systems[1] and trivially supports nonzero initial conditions.

If modern control theory is so great and classical control theory isn't needed to use it, why learn classical control theory at all? We teach classical control theory because it provides a framework within which to understand results from the mathematical machinery of modern control as well as vocabulary with which to communicate that understanding. For example, faster poles (poles moved to the left in the s-plane) mean faster decay, and oscillation means there is at least one pair of complex conjugate poles. Not only can you describe what happened succinctly, but you know why it happened from a theoretical perspective.

This book uses LQR and modern control over, say, loop shaping with Bode and Nyquist plots because we have accurate dynamical models to leverage, and LQR allows directly expressing what the author is concerned with optimizing: state excursion relative to control effort. Applying lead and lag compensators, while effective for robust controller design, doesn't provide the same expressive power.

E.2 Transfer functions

We will briefly discuss what transfer functions are, how the locations of poles and zeroes affect system response and stability, and how controllers affect pole locations.

E.2.1 Laplace transform

For an introduction to Laplace transforms and the geometric intuition behind transfer functions, we recommend the YouTube video "What does the Laplace Transform really tell us?" by Zach Star.

A more mathematical introduction is presented later in chapter E.3.

E.2.2 Parts of a transfer function

A transfer function maps an input coordinate to an output coordinate in the Laplace domain. These can be obtained by applying the Laplace transform to a differential

[1]This book primarily focuses on analysis and control of linear systems. See chapter 8 for more on nonlinear control.

"What does the Laplace Transform really tell us?" (21 minutes)
Zach Star
https://youtu.be/n2y7n6jw5d0

equation and rearranging the terms to obtain a ratio of the output variable to the input variable. Equation (E.1) is an example of a transfer function.

$$H(s) = \frac{\overbrace{(s-9+9i)(s-9-9i)}^{zeroes}}{\underbrace{s(s+10)}_{poles}} \tag{E.1}$$

Poles and zeroes

The roots of factors in the numerator of a transfer function are called *zeroes* because they make the transfer function approach zero. Likewise, the roots of factors in the denominator of a transfer function are called *poles* because they make the transfer function approach infinity; on a 3D graph, these look like the poles of a circus tent (see figure E.1).

When the factors of the denominator are broken apart using partial fraction expansion into something like $\frac{A}{s+a} + \frac{B}{s+b}$, the constants A and B are called residues, which determine how much each pole contributes to the system response.

The factors representing poles are each the Laplace transform of a decaying exponential.[2] That means the time domain responses of systems comprise decaying exponentials (e.g., $y = e^{-t}$).

R Imaginary poles and zeroes always come in complex conjugate pairs (e.g., $-2+3i$, $-2-3i$).

The locations of the closed-loop poles in the complex plane determine the stability of the system. Each pole represents a frequency mode of the system, and their location determines how much of each response is induced for a given input frequency. Figure

[2]We are handwaving Laplace transform derivations because they are complicated and neither relevant nor useful.

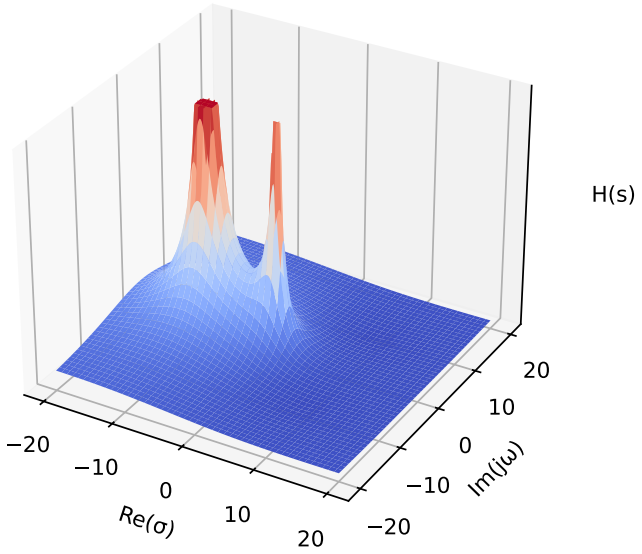


Figure E.1: Equation (E.1) plotted in 3D

E.2 shows the impulse responses in the time domain for transfer functions with various pole locations. They all have an initial condition of 1.

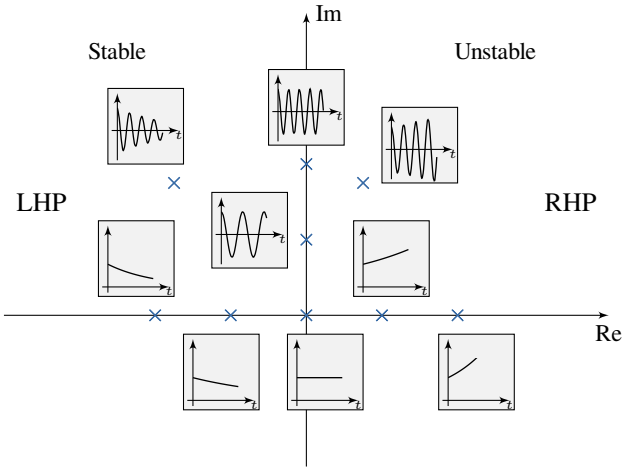


Figure E.2: Continuous impulse response vs pole location

Poles in the left half-plane (LHP) are stable; the system's output may oscillate but it converges to steady-state. Poles on the imaginary axis are marginally stable; the system's output oscillates at a constant amplitude forever. Poles in the right half-plane (RHP) are unstable; the system's output grows without bound.

Nonminimum phase zeroes

While poles in the RHP are unstable, the same is not true for zeroes. They can be characterized by the system initially moving in the wrong direction before heading toward the reference. Since the poles always move toward the zeroes, zeroes impose a "speed limit" on the system response because it takes a finite amount of time to move the wrong direction, then change directions.

One example of this type of system is bicycle steering. Try riding a bicycle without holding the handle bars, then poke the right handle; the bicycle turns right. Furthermore, if one is holding the handlebars and wants to turn left, rotating the handlebars counterclockwise will make the bicycle fall toward the right. The rider has to lean into the turn and overpower the nonminimum phase dynamics to go the desired direction.

Another example is a Segway. To move forward by some distance, the Segway must first roll backward to rotate the Segway forward. Once the Segway starts falling in

that direction, it begins rolling forward to avoid falling over until it reaches the target distance. At that point, the Segway increases its forward speed to pitch backward and slow itself down. To come to a stop, the Segway rolls backward again to level itself out.

Pole-zero cancellation

Pole-zero cancellation occurs when a pole and zero are located at the same place in the s-plane. This effectively eliminates the contribution of each to the system dynamics. By placing poles and zeroes at various locations (this is done by placing transfer functions in series), we can eliminate undesired system dynamics. While this may appear to be a useful design tool at first, there are major caveats. Most of these are due to model uncertainty resulting in poles which aren't in the locations the controls designer expected.

Notch filters are typically used to dampen a specific range of frequencies in the system response. If its band is made too narrow, it can still leave the undesirable dynamics, but now you can no longer measure them in the response. They are still happening, but they are what's called *unobservable*.

Never pole-zero cancel unstable or nonminimum phase dynamics. If the model doesn't quite reflect reality, an attempted pole cancellation by placing a nonminimum phase zero results in the pole still moving to the zero placed next to it. You have the same dynamics as before, but the pole is also stuck where it is no matter how much feedback gain is applied. For an attempted nonminimum phase zero cancellation, you have effectively placed an unstable pole that's unobservable. This means the system will be going unstable and blowing up, but you won't be able to detect this and react to it.

Keep in mind when making design decisions that the model likely isn't perfect. The whole point of feedback control is to be robust to this kind of uncertainty.

E.2.3 Transfer functions in feedback

For controllers to regulate a system or track a reference, they must be placed in positive or negative feedback with the plant (whether to use positive or negative depends on the plant in question). Stable feedback loops attempt to make the output equal the reference.

Derivation

Given the feedback network in figure E.3, find an expression for $Y(s)$.

$$\begin{aligned} Y(s) &= Z(s)G(s) \\ Z(s) &= X(s) - Y(s)H(s) \\ X(s) &= Z(s) + Y(s)H(s) \\ X(s) &= Z(s) + Z(s)G(s)H(s) \end{aligned}$$

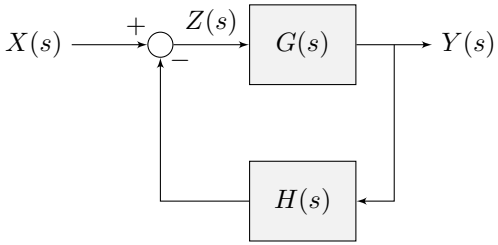


Figure E.3: Closed-loop block diagram

$$\frac{Y(s)}{X(s)} = \frac{Z(s)G(s)}{Z(s) + Z(s)G(s)H(s)}$$
$$\frac{Y(s)}{X(s)} = \frac{G(s)}{1 + G(s)H(s)} \tag{E.2}$$

A more general form is

$$\frac{Y(s)}{X(s)} = \frac{G(s)}{1 \mp G(s)H(s)} \tag{E.3}$$

where positive feedback uses the top sign and negative feedback uses the bottom sign.

Control system with feedback

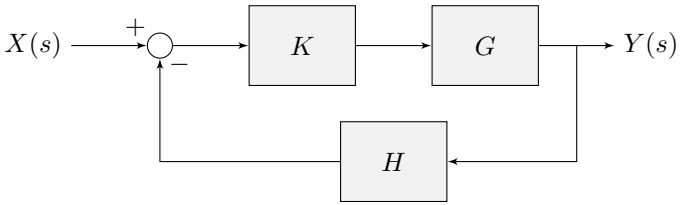


Figure E.4: Feedback controller block diagram

$X(s)$	input	H	measurement transfer function
K	controller gain	$Y(s)$	output
G	plant transfer function		

Following equation (E.3), the transfer function of figure E.4, a control system diagram with negative feedback, from input to output is

$$G_{cl}(s) = \frac{Y(s)}{X(s)} = \frac{KG}{1 + KGH} \tag{E.4}$$

The numerator is the open-loop gain and the denominator is one plus the gain around the feedback loop, which may include parts of the open-loop gain. As another example, the transfer function from the input to the error is

$$G_{cl}(s) = \frac{E(s)}{X(s)} = \frac{1}{1 + KGH} \tag{E.5}$$

The roots of the denominator of $G_{cl}(s)$ are different from those of the open-loop transfer function $KG(s)$. These are called the closed-loop poles.

DC motor transfer function

If poles are much farther left in the LHP than the typical system dynamics exhibit, they can be considered negligible. Every system has some form of unmodeled high frequency, nonlinear dynamics, but they can be safely ignored depending on the operating regime.

To demonstrate this, consider the transfer function for a second-order DC motor (a CIM motor) from voltage to velocity.

$$G(s) = \frac{K}{(Js + b)(Ls + R) + K^2}$$

where $J = 3.2284 \times 10^{-6}$ $kg\text{-}m^2$, $b = 3.5077 \times 10^{-6}$ $N\text{-}m\text{-}s$, $K_e = K_t = 0.0181\,V/rad/s$, $R = 0.0902\,\Omega$, and $L = 230 \times 10^{-6}\,H$.

This system is second-order because it has two poles; one corresponds to velocity and the other corresponds to current. To make position the output instead, we'll multiply by $\frac{1}{s}$ because position is the integral of velocity.[3]

$$G(s) = \frac{K}{s((Js + b)(Ls + R) + K^2)}$$

Compare the step response of this system (figure E.5) with the step response of this system with L set to zero (figure E.6). For small values of K, both systems are stable and have nearly indistinguishable step responses due to the exceedingly small contribution from the fast pole. The high frequency dynamics only cause instability for large values of K that induce fast system responses. In other words, the system responses of the second-order model and its first-order approximation are similar for low frequency operating regimes.

[3]This also has the effect of augmenting the system with a position state, making it a third-order system.

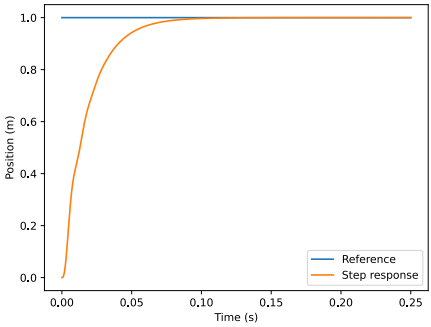


Figure E.5: Step response of second-order DC motor plant augmented with position ($L = 230$ µH)

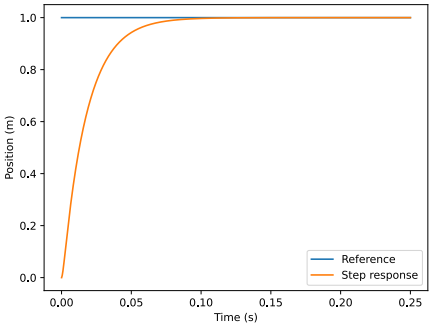


Figure E.6: Step response of first-order DC motor plant augmented with position ($L = 0$ µH)

Why can't unstable poles close to the origin be ignored in the same way? The response of high frequency stable poles decays rapidly. Unstable poles, on the other hand, represent unstable dynamics which cause the system output to grow to infinity. Regardless of how slow these unstable dynamics are, they will eventually dominate the response.

E.3 Laplace domain analysis

This chapter uses Laplace transforms and transfer functions to analyze properties of control systems like steady-state error.

These case studies cover various aspects of PID control using the algebraic approach of transfer functions. For this, we'll be using equation (E.6), the transfer function for a PID controller.

$$K(s) = K_p + \frac{K_i}{s} + K_d s \tag{E.6}$$

First, we will define mathematically what Laplace transforms and transfer functions are, which is rooted in the concept of orthogonal projections.

E.3.1 Projections

Consider a two-dimensional Euclidean space $\mathbb{R}^2$ shown in figure E.7 (each $\mathbb{R}$ is a dimension whose domain is the set of real numbers, so $\mathbb{R}^2$ is the standard x-y plane).

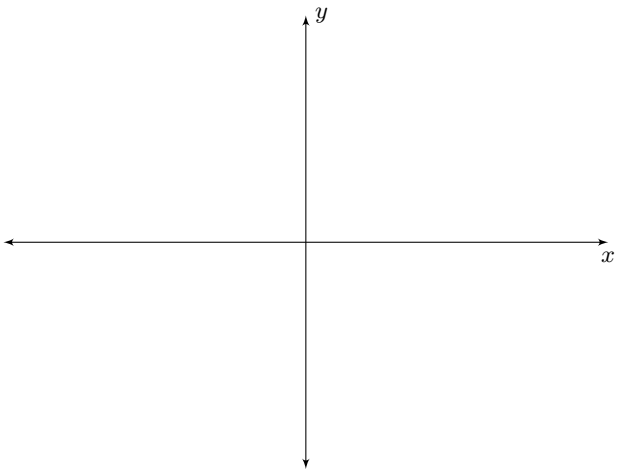


Figure E.7: Euclidean space $\mathbb{R}^2$

Ordinarily, we notate points in this plane by their components in the set of basis vectors $\{\hat{i}, \hat{j}\}$, where $\hat{i}$ (pronounced i-hat) is the unit vector in the positive x direction and $\hat{j}$ is the unit vector in the positive y direction. Figure E.8 shows an example vector $\mathbf{v}$ in this basis.

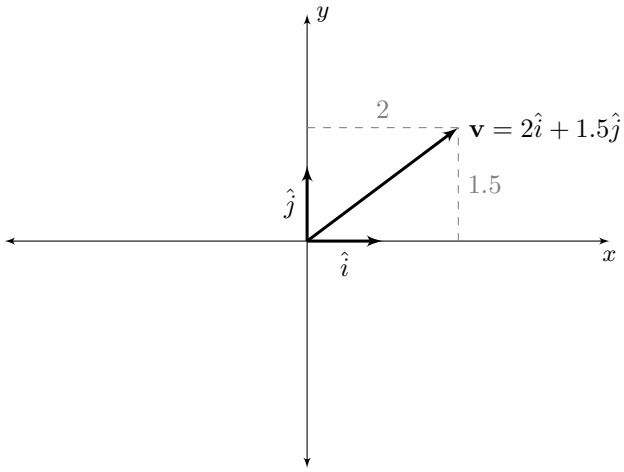


Figure E.8: $\mathbf{v}$ with basis set $\{\hat{i}, \hat{j}\}$

How do we find the coordinates of $\mathbf{v}$ in this basis mathematically? As long as the basis is *orthogonal* (i.e., the basis vectors are at right angles to each other), we simply take the *orthogonal projection* of $\mathbf{v}$ onto $\hat{i}$ and $\hat{j}$. Intuitively, this means finding "the amount of $\mathbf{v}$ that points in the direction of $\hat{i}$ or $\hat{j}$". Note that a set of orthogonal vectors have a dot product of zero with respect to each other.

More formally, we can calculate projections with the dot product - the projection of $\mathbf{v}$ onto any other vector $\mathbf{w}$ is as follows.

$$\text{proj}_{\mathbf{w}}\mathbf{v} = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{w}|}$$

Since $\hat{i}$ and $\hat{j}$ are *unit vectors*, their magnitudes are 1 so the coordinates of $\mathbf{v}$ are $\mathbf{v} \cdot \hat{i}$ and $\mathbf{v} \cdot \hat{j}$.

We can use this same process to find the coordinates of $\mathbf{v}$ in *any* orthogonal basis. For example, imagine the basis $\{\hat{i} + \hat{j}, \hat{i} - \hat{j}\}$ - the coordinates in this basis are given by $\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}}$ and $\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}}$. Let's "unwrap" the formula for dot product and look a bit more closely.

$$\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} = \frac{1}{\sqrt{2}} \sum_{i=0}^{n} \mathbf{v}_i (\hat{i} + \hat{j})_i$$

where the subscript i denotes which component of each vector and n is the total number

of components. To change coordinates, we expanded both $\mathbf{v}$ and $\hat{i} + \hat{j}$ in a basis, multiplied their components, and added them up. Here's a more concrete example. Let $\mathbf{v} = 2\hat{i} + 1.5\hat{j}$ from figure E.8. First, we'll project $\mathbf{v}$ onto the $\hat{i} + \hat{j}$ basis vector.

$$\begin{aligned}
\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} &= \frac{1}{\sqrt{2}}(2\hat{i} \cdot \hat{i} + 1.5\hat{j} \cdot \hat{j}) \\
\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} &= \frac{1}{\sqrt{2}}(2 + 1.5) \\
\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} &= \frac{3.5}{\sqrt{2}} \\
\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} &= \frac{3.5\sqrt{2}}{2} \\
\frac{\mathbf{v} \cdot (\hat{i} + \hat{j})}{\sqrt{2}} &= 1.75\sqrt{2}
\end{aligned}$$

Next, we'll project $\mathbf{v}$ onto the $\hat{i} - \hat{j}$ basis vector.

$$\begin{aligned}
\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}} &= \frac{1}{\sqrt{2}}(2\hat{i} \cdot \hat{i} - 1.5\hat{j} \cdot \hat{j}) \\
\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}} &= \frac{1}{\sqrt{2}}(2 - 1.5) \\
\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}} &= \frac{0.5}{\sqrt{2}} \\
\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}} &= \frac{0.5\sqrt{2}}{2} \\
\frac{\mathbf{v} \cdot (\hat{i} - \hat{j})}{\sqrt{2}} &= 0.25\sqrt{2}
\end{aligned}$$

Figure E.9 shows this result geometrically with respect to the basis $\{\hat{i} + \hat{j}, \hat{i} - \hat{j}\}$.

The previous example was only a change of coordinates in a finite-dimensional vector space. However, as we will see, the core idea does not change much when we move to more complicated structures. Observe the formula for the Fourier transform.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} \, dx \text{ where } \xi \in \mathbb{R}$$

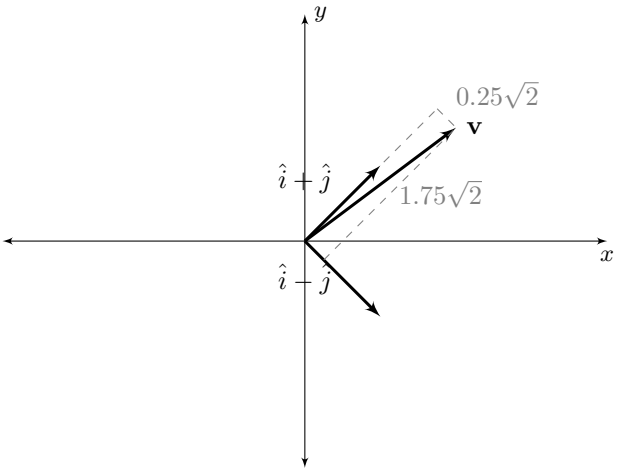


Figure E.9: $\mathbf{v}$ with basis $\{\hat{i}+\hat{j}, \hat{i}-\hat{j}\}$

This is fundamentally the same formula we had before. $f(x)$ has taken the place of v_n, $e^{-2\pi i x\xi}$ has taken the place of $(\hat{i}+\hat{j})_i$, and the sum over i has turned into an integral over dx, but the underlying concept is the same. To change coordinates in a *function space*, we simply take the orthogonal projection onto our new basis *functions*. In the case of the Fourier transform, the function basis is the family of functions of the form $f(x) = e^{-2\pi i x\xi}$ for $\xi \in \mathbb{R}$. Since these functions are oscillatory at a frequency determined by ξ, we can think of this as a "frequency basis".

R Watch the "Abstract vector spaces" video from 3Blue1Brown's *Essence of linear algebra* series for a more geometric introduction to using functions as a basis.

"Abstract vector spaces" (17 minutes)
3Blue1Brown
https://www.3blue1brown.com/lessons/abstract-vector-spaces

Now, the Laplace transform is somewhat more complicated - as it turns out, the Fourier

basis is orthogonal, so the analogy to the simpler vector space holds almost-precisely. The Laplace transform is *not* orthogonal, so we can't interpret it *strictly* as a change of coordinates in the traditional sense. However, the intuition is the same: we are taking the orthogonal projection of our original function onto the functions of our new basis set.

$$F(s) = \int_0^\infty f(t)e^{-st}\,dt, \text{ where } s \in \mathbb{C}$$

Here, it becomes obvious that the Laplace transform is a *generalization* of the Fourier transform in that the basis family is strictly larger (we have allowed the "frequency" parameter to take *complex* values, as opposed to merely *real* values). As a result, the Laplace basis contains functions that grow and decay, while the Fourier basis does not.

E.3.2 Fourier transform

The Fourier transform decomposes a function of time into its component frequencies. Each of these frequencies is part of what's called a *basis*. These basis waveforms can be multiplied by their respective contribution amount and summed to produce the original signal (this weighted sum is called a linear combination). In other words, the Fourier transform provides a way for us to determine, given some signal, what frequencies can we add together and in what amounts to produce the original signal.

Think of an Fmajor4 chord which has the notes F_4 (349.23 Hz), A_4 (440 Hz), and C_4 (261.63 Hz). The waveform over time looks like figure E.10.

Notice how this complex waveform can be represented just by three frequencies. They show up as Dirac delta functions[4] in the frequency domain with the area underneath them equal to their contribution (see figure E.11).

E.3.3 Laplace transform

The Laplace domain is a generalization of the frequency domain that has the frequency ($j\omega$) on the imaginary y-axis and a real number on the x-axis, yielding a two-dimensional coordinate system. We represent coordinates in this space as a complex number $s = \sigma + j\omega$. The real part σ corresponds to the x-axis and the imaginary part $j\omega$ corresponds to the y-axis (see figure E.12).

To extend our analogy of each coordinate being represented by some basis, we now have the y coordinate representing the oscillation frequency of the system response (the frequency domain) and also the x coordinate representing the speed at which that

[4]The Dirac delta function is zero everywhere except at the origin. The nonzero region has an infinitesimal width and has a height such that the area within that region is 1.

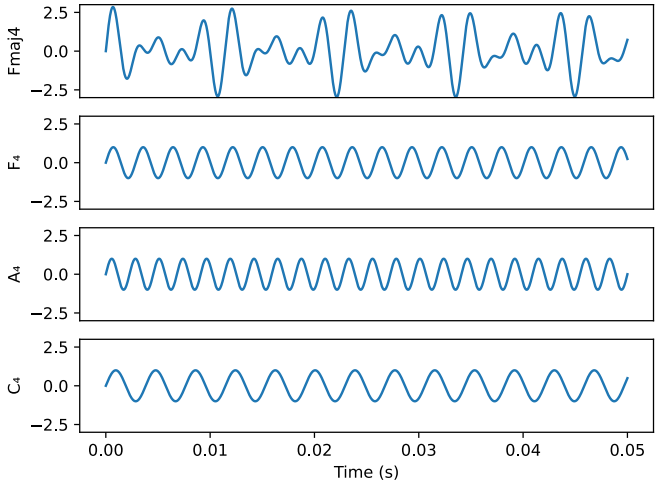


Figure E.10: Frequency decomposition of Fmajor4 chord

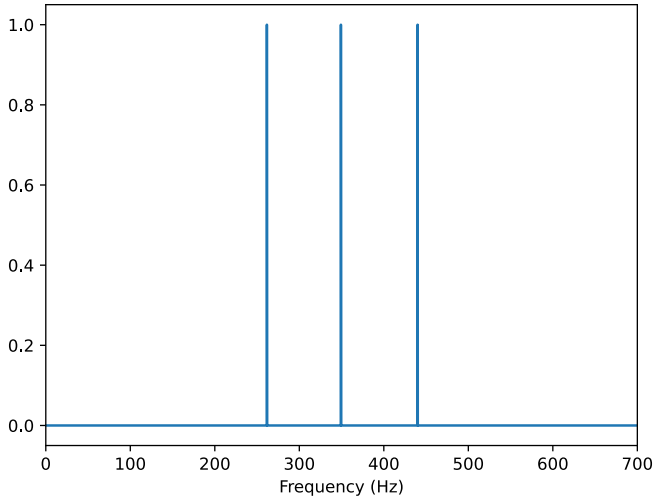


Figure E.11: Fourier transform of Fmajor4 chord

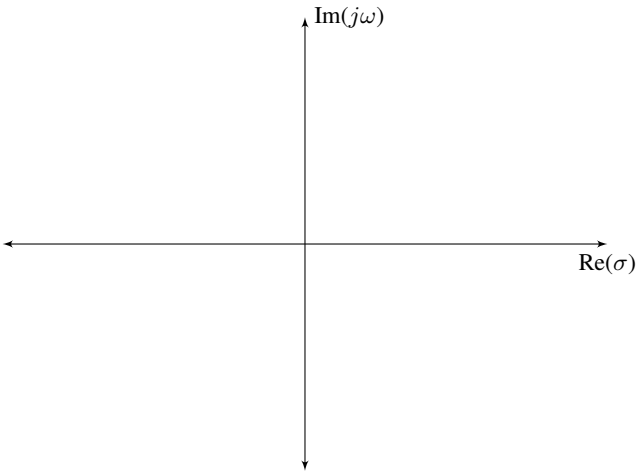


Figure E.12: Laplace domain

oscillation decays and the system converges to zero (i.e., a decaying exponential). Figure E.2 shows this for various points.

If we move the component frequencies in the Fmajor4 chord example parallel to the real axis to $\sigma = -25$, the resulting time domain response attenuates according to the decaying exponential e^{-25t} (see figure E.13).

Note that this explanation as a basis isn't exact because the Laplace basis isn't orthogonal (that is, the x and y coordinates affect each other and have cross-talk). In the frequency domain, we had a basis of sine waves that we represented as delta functions in the frequency domain. Each frequency contribution was independent of the others. In the Laplace domain, this is not the case; a pure exponential is $\frac{1}{s-a}$ (a rational function where a is a real number) instead of a delta function. This function is nonzero at points that aren't actually frequencies present in the time domain. Figure E.14 demonstrates this, which shows the Laplace transform of the Fmajor4 chord plotted in 3D.

Notice how the values of the function around each component frequency decrease according to $\frac{1}{\sqrt{x^2+y^2}}$ in the x and y directions (in just the x direction, it would be $\frac{1}{x}$).

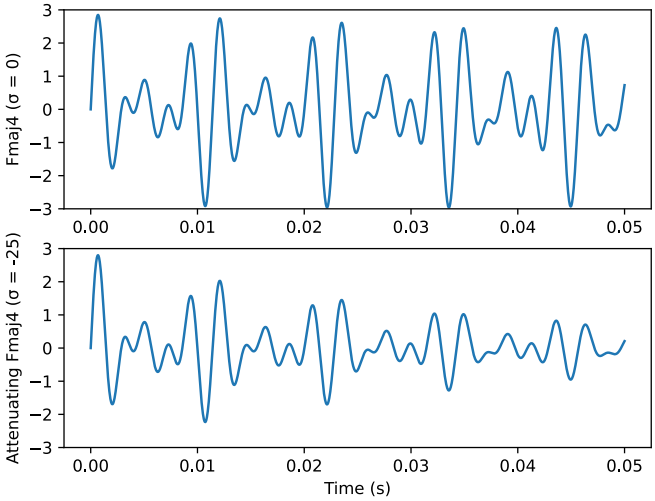


Figure E.13: Fmajor4 chord at $\sigma = 0$ and $\sigma = -25$

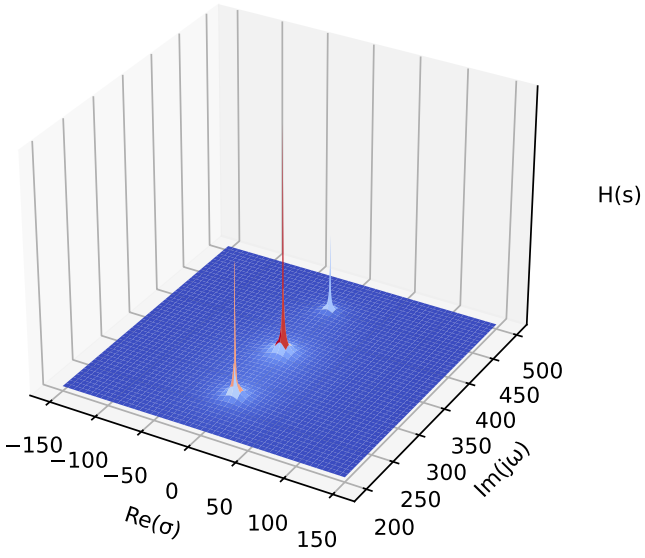


Figure E.14: Laplace transform of Fmajor4 chord plotted in 3D

E.3.4 Laplace transform definition

The Laplace transform of a function $f(t)$ is defined as

$$\mathcal{L}\{f(t)\} = F(s) = \int_0^\infty f(t)e^{-st}\,dt$$

We won't be computing any Laplace transforms by hand using this formula (everyone in the real world looks these up in a table anyway). Common Laplace transforms (assuming zero initial conditions) are shown in table E.1. Of particular note are the Laplace transforms for the derivative, unit step,[5] and exponential decay. We can see that a derivative is equivalent to multiplying by s, and an integral is equivalent to multiplying by $\frac{1}{s}$.

	Time domain	Laplace domain
Linearity	$a\,f(t) + b\,g(t)$	$a\,F(s) + b\,G(s)$
Convolution	$(f * g)(t)$	$F(s)\,G(s)$
Derivative	$f'(t)$	$s\,F(s)$
n^{th} derivative	$f^{(n)}(t)$	$s^n\,F(s)$
Unit step	$u(t)$	$\frac{1}{s}$
Ramp	$t\,u(t)$	$\frac{1}{s^2}$
Exponential decay	$e^{-\alpha t}u(t)$	$\frac{1}{s+\alpha}$

Table E.1: Common Laplace transforms and Laplace transform properties with zero initial conditions

E.3.5 Steady-state error

To demonstrate the problem of steady-state error, we will use a DC motor controlled by a velocity PID controller. A DC motor has a transfer function from voltage (V) to angular velocity ($\dot{\theta}$) of

$$G(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \tag{E.7}$$

First, we'll try controlling it with a P controller defined as

$$K(s) = K_p$$

[5] The unit step $u(t)$ is defined as 0 for $t < 0$ and 1 for $t \geq 0$.

When these are in unity feedback, the transfer function from the input voltage to the error is

$$\begin{aligned}
\frac{E(s)}{V(s)} &= \frac{1}{1 + K(s)G(s)} \\
E(s) &= \frac{1}{1 + K(s)G(s)} V(s) \\
E(s) &= \frac{1}{1 + (K_p)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} V(s) \\
E(s) &= \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} V(s)
\end{aligned}$$

The steady-state of a transfer function can be found via

$$\lim_{s \to 0} sH(s) \tag{E.8}$$

since steady-state has an input frequency of zero.

$$\begin{aligned}
e_{ss} &= \lim_{s \to 0} sE(s) \\
e_{ss} &= \lim_{s \to 0} s \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} V(s) \\
e_{ss} &= \lim_{s \to 0} s \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} \frac{1}{s} \\
e_{ss} &= \lim_{s \to 0} \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} \\
e_{ss} &= \frac{1}{1 + \frac{K_p K}{(J(0)+b)(L(0)+R)+K^2}} \\
e_{ss} &= \frac{1}{1 + \frac{K_p K}{bR+K^2}}
\end{aligned} \tag{E.9}$$

Notice that the steady-state error is nonzero. To fix this, an integrator must be included in the controller.

$$K(s) = K_p + \frac{K_i}{s}$$

The same steady-state calculations are performed as before with the new controller.

$$\frac{E(s)}{V(s)} = \frac{1}{1 + K(s)G(s)}$$

$$E(s) = \frac{1}{1 + K(s)G(s)}V(s)$$

$$E(s) = \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)}\left(\frac{1}{s}\right)$$

$$e_{ss} = \lim_{s \to 0} s\frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)}\left(\frac{1}{s}\right)$$

$$e_{ss} = \lim_{s \to 0} \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)}$$

$$e_{ss} = \lim_{s \to 0} \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)}\frac{s}{s}$$

$$e_{ss} = \lim_{s \to 0} \frac{s}{s + (K_p s + K_i)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)}$$

$$e_{ss} = \frac{0}{0 + (K_p(0) + K_i)\left(\frac{K}{(J(0)+b)(L(0)+R)+K^2}\right)}$$

$$e_{ss} = \frac{0}{K_i \frac{K}{bR+K^2}}$$

The denominator is nonzero, so $e_{ss} = 0$. Therefore, an integrator is required to eliminate steady-state error in all cases for this model.

It should be noted that e_{ss} in equation (E.9) approaches zero for $K_p = \infty$. This is known as a bang-bang controller. In practice, an infinite switching frequency cannot be achieved, but it may be close enough for some performance specifications.

E.3.6 Do flywheels need PD control?

PID controllers typically control voltage to a motor in FRC independent of the equations of motion of that motor. For position PID control, large values of K_p can lead to overshoot and K_d is commonly used to reduce overshoots. Let's consider a flywheel controlled with a standard PID controller. Why wouldn't K_d provide damping for velocity overshoots in this case?

PID control is designed to control second-order and first-order systems well. It can be used to control a lot of things, but struggles when given higher order systems. It has three degrees of freedom. Two are used to place the two poles of the system, and the third is used to remove steady-state error. With higher order systems like a one input,

seven state system, there aren't enough degrees of freedom to place the system's poles in desired locations. This will result in poor control.

The math for PID doesn't assume voltage, a motor, etc. It defines an output based on derivatives and integrals of its input. We happen to use it for motors because it actually works pretty well for it because motors are second-order systems.

The following math will be in continuous time, but the same ideas apply to discrete time. This is all assuming a velocity controller.

Our simple motor model hooked up to a mass is

$$V = IR + \frac{\omega}{K_v} \tag{E.10}$$

$$\tau = IK_t \tag{E.11}$$

$$\tau = J\frac{d\omega}{dt} \tag{E.12}$$

For an explanation of where these equations come from, read section 12.1.

First, we'll solve for $\frac{d\omega}{dt}$ in terms of V.

Substitute equation (E.11) into equation (E.10).

$$V = IR + \frac{\omega}{K_v}$$
$$V = \left(\frac{\tau}{K_t}\right)R + \frac{\omega}{K_v}$$

Substitute in equation (E.12).

$$V = \frac{\left(J\frac{d\omega}{dt}\right)}{K_t}R + \frac{\omega}{K_v}$$

Solve for $\frac{d\omega}{dt}$.

$$V = \frac{J\frac{d\omega}{dt}}{K_t}R + \frac{\omega}{K_v}$$
$$V - \frac{\omega}{K_v} = \frac{J\frac{d\omega}{dt}}{K_t}R$$
$$\frac{d\omega}{dt} = \frac{K_t}{JR}\left(V - \frac{\omega}{K_v}\right)$$
$$\frac{d\omega}{dt} = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}V$$

Now take the Laplace transform. Because the Laplace transform is a linear operator, we can take the Laplace transform of each term individually. Based on table E.1, $\frac{d\omega}{dt}$ becomes $s\omega$ and $\omega(t)$ and $V(t)$ become $\omega(s)$ and $V(s)$ respectively (the parenthetical notation has been dropped for clarity).

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}V \tag{E.13}$$

Solve for the transfer function $H(s) = \frac{\omega}{V}$.

$$\begin{aligned} s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}V \\ \left(s + \frac{K_t}{JRK_v}\right)\omega &= \frac{K_t}{JR}V \\ \frac{\omega}{V} &= \frac{\frac{K_t}{JR}}{s + \frac{K_t}{JRK_v}} \end{aligned}$$

That gives us a pole at $-\frac{K_t}{JRK_v}$, which is actually stable. Notice that there is only one pole.

First, we'll use a simple P controller.

$$V = K_p(\omega_{goal} - \omega)$$

Substitute this controller into equation (E.13).

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}K_p(\omega_{goal} - \omega)$$

Solve for the transfer function $H(s) = \frac{\omega}{\omega_{goal}}$.

$$\begin{aligned} s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}(\omega_{goal} - \omega) \\ s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}\omega_{goal} - \frac{K_tK_p}{JR}\omega \\ \left(s + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}\right)\omega &= \frac{K_tK_p}{JR}\omega_{goal} \\ \frac{\omega}{\omega_{goal}} &= \frac{\frac{K_tK_p}{JR}}{\left(s + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}\right)} \end{aligned}$$

This has a pole at $-\left(\frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)$. Assuming that that quantity is negative (i.e., we are stable), that pole corresponds to a time constant of $\frac{1}{\frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}}$.

As can be seen above, a flywheel has a single pole. It therefore only needs a single pole controller to place that pole anywhere on the real axis.

This analysis assumes that the motor is well coupled to the mass and that the time constant of the inductor is small enough that it doesn't factor into the motor equations. The latter is a pretty good assumption for a CIM motor (see figures E.15 and E.16). If more mass is added to the motor armature, the response timescales increase and the inductance matters even less.

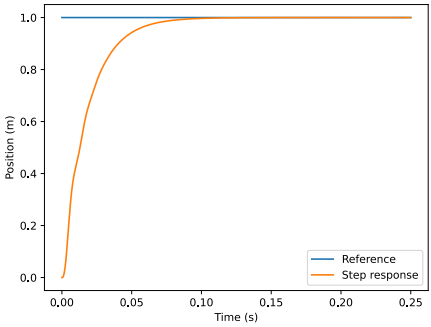


Figure E.15: Step response of second-order DC motor plant augmented with position ($L = 230$ µH)

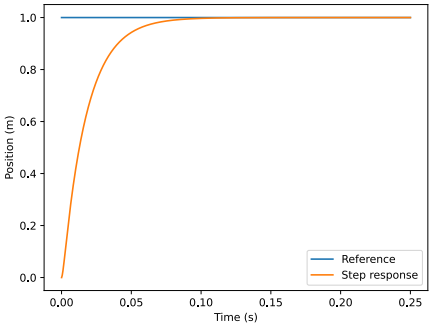


Figure E.16: Step response of first-order DC motor plant augmented with position ($L = 0$ µH)

Next, we'll try a PD loop. (This will use a perfect derivative, but anyone following along closely already knows that we can't really take a derivative here, so the math will need to be updated at some point. We could switch to discrete time and pick a differentiation method, or pick some other way of modeling the derivative.)

$$V = K_p(\omega_{goal} - \omega) + K_d s(\omega_{goal} - \omega)$$

Substitute this controller into equation (E.13).

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}\left(K_p(\omega_{goal} - \omega) + K_d s(\omega_{goal} - \omega)\right)$$

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}(\omega_{goal} - \omega) + \frac{K_tK_ds}{JR}(\omega_{goal} - \omega)$$
$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}\omega_{goal} - \frac{K_tK_p}{JR}\omega + \frac{K_tK_ds}{JR}\omega_{goal} - \frac{K_tK_ds}{JR}\omega$$

Collect the common terms on separate sides and refactor.

$$s\omega + \frac{K_tK_ds}{JR}\omega + \frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}\omega = \frac{K_tK_p}{JR}\omega_{goal} + \frac{K_tK_ds}{JR}\omega_{goal}$$
$$\left(s + \frac{K_tK_ds}{JR} + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}\right)\omega = \left(\frac{K_tK_p}{JR} + \frac{K_tK_ds}{JR}\right)\omega_{goal}$$
$$\left(s\left(1 + \frac{K_tK_d}{JR}\right) + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}\right)\omega = \frac{K_t}{JR}\left(K_p + K_ds\right)\omega_{goal}$$

Solve for $\frac{\omega}{\omega_{goal}}$.

$$\frac{\omega}{\omega_{goal}} = \frac{\frac{K_t}{JR}\left(K_p + K_ds\right)}{s\left(1 + \frac{K_tK_d}{JR}\right) + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}}$$

So, we added a zero at $-\frac{K_p}{K_d}$ and moved our pole to $-\frac{\frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}}{1 + \frac{K_tK_d}{JR}}$. This isn't progress. We've added more complexity to our system and, practically speaking, gotten nothing good in return. Zeroes should be avoided if at all possible because they amplify unwanted high frequency modes of the system and are noisier the faster the system is sampled. At least this is a stable zero, but it's still undesirable.

In summary, derivative doesn't help on an ideal flywheel. K_d may compensate for unmodeled dynamics such as accelerating projectiles slowing the flywheel down, but that effect may also increase recovery time; K_d drives the acceleration to zero in the undesired case of negative acceleration as well as well as the actually desired case of positive acceleration.

Subsection 6.7.2 covers a superior compensation method that avoids zeroes in the controller, doesn't act against the desired control action, and facilitates better tracking.

E.3.7 Gain margin and phase margin

One generally needs to learn about Bode plots and Nyquist plots to truly understand gain and phase margin and their origins, but those plots are large topics unto themselves. Since we won't be using either of them for controller design, we'll just cover what gain and phase margin are in a general sense and how they are used.

Gain margin and phase margin are two metrics for measuring a system's relative stability. Gain and phase margin are the amounts by which the closed-loop gain and phase can be varied respectively before the system becomes unstable. In a sense, they are safety margins for when unmodeled dynamics affect the system response.

For a more thorough explanation of gain and phase margin, watch Brian Douglas's video on them.

"Gain and Phase Margins Explained!" (14 minutes)
Brian Douglas
https://youtu.be/ThoA4amCAX4

He has other videos too on classical control methods like Bode and Nyquist plots that we recommend.

E.4 s-plane to z-plane

Transfer functions are converted to impulse responses using the Z-transform. The s-plane's LHP maps to the inside of a unit circle in the z-plane. Table E.2 contains a few common points and figure E.17 shows the mapping visually.

s-plane	z-plane
$(0, 0)$	$(1, 0)$
imaginary axis	edge of unit circle
$(-\infty, 0)$	$(0, 0)$

Table E.2: Mapping from s-plane to z-plane

E.4.1 Discrete system stability

Eigenvalues of a system that are within the unit circle are stable. To demonstrate this, consider the discrete system $x_{k+1} = ax_k$ where a is a complex number. $|a| < 1$ will make x_{k+1} converge to zero.

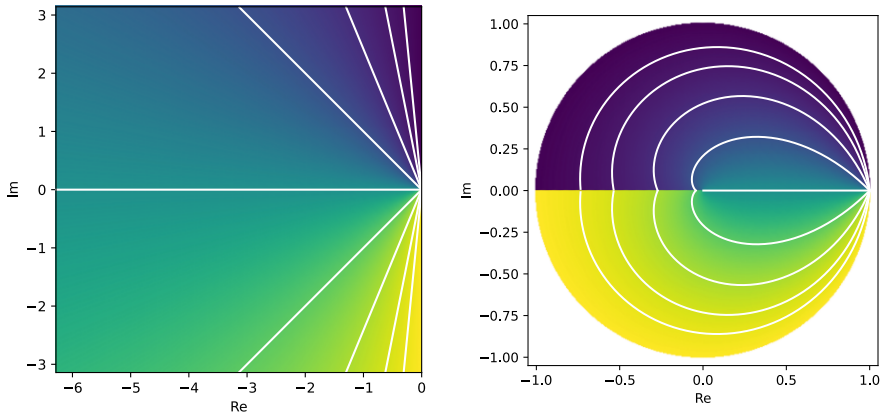


Figure E.17: Mapping of complex plane from s-plane (left) to z-plane (right)

E.4.2 Discrete system behavior

Figure E.18 shows the impulse responses in the time domain for systems with various pole locations in the complex plane (real numbers on the x-axis and imaginary numbers on the y-axis). Each response has an initial condition of 1.

As ω increases in $s = j\omega$, a pole in the z-plane moves around the perimeter of the unit circle. Once it hits $\frac{\omega_s}{2}$ (half the sampling frequency) at $(-1, 0)$, the pole wraps around. This is due to poles faster than the sample frequency folding down to below the sample frequency (that is, higher frequency signals *alias* to lower frequency ones).

Placing the poles at $(0, 0)$ produces a *deadbeat controller*. An N^{th}-order deadbeat controller decays to the reference in N timesteps. While this sounds great, there are other considerations like control effort, robustness, and noise immunity.

Poles in the left half-plane cause jagged outputs because the frequency of the system dynamics is above the Nyquist frequency (twice the sample frequency). The discretized signal doesn't have enough samples to reconstruct the continuous system's dynamics. See figures E.19 and E.20 for examples.

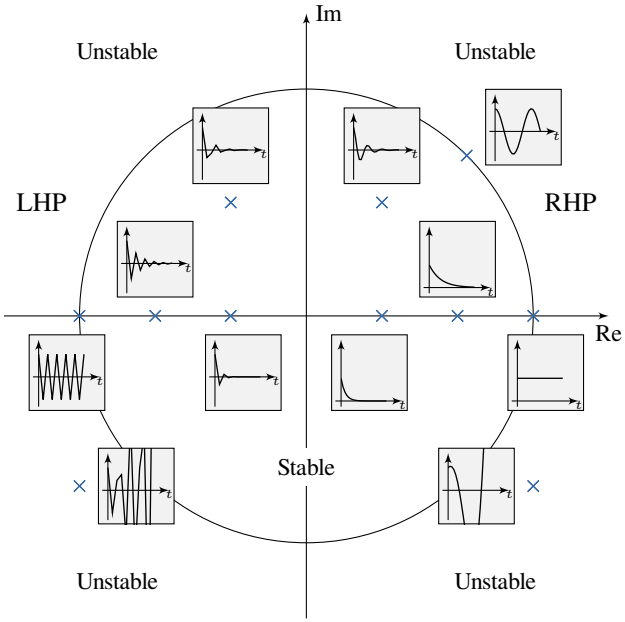


Figure E.18: Discrete impulse response vs pole location

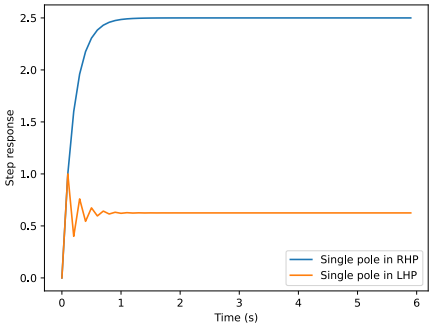


Figure E.19: Single poles in various locations in z-plane

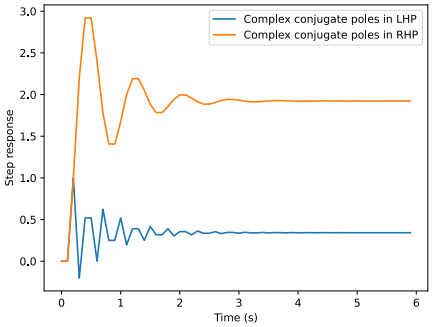


Figure E.20: Complex conjugate poles in various locations in z-plane

E.5 Discretization methods

Discretization is done using a zero-order hold. That is, the input is only updated at discrete intervals and it's held constant between samples (see figure E.21). The exact method of applying this uses the matrix exponential, but this can be computationally expensive. Instead, approximations such as the following are used.

1. Forward Euler method. This is defined as $y_{n+1} = y_n + f(t_n, y_n)\Delta t$.
2. Backward Euler method. This is defined as $y_{n+1} = y_n + f(t_{n+1}, y_{n+1})\Delta t$.
3. Bilinear transform. The first-order bilinear approximation is $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$.

where the function $f(t_n, y_n)$ is the slope of y at n and T is the sample period for the discrete system. Each of these methods is essentially finding the area underneath a curve. The forward and backward Euler methods use rectangles to approximate that area while the bilinear transform uses trapezoids (see figures E.22 and E.23). Since these are approximations, there is distortion between the real discrete system's poles and the approximate poles. This is in addition to the phase loss introduced by discretizing at a given sample rate in the first place. For fast-changing systems, this distortion can quickly lead to instability.

Figures E.24, E.25, and E.26 show simulations of the same controller for different sampling methods and sample rates, which have varying levels of fidelity to the real system.

Forward Euler is numerically unstable for low sample rates. The bilinear transform is a significant improvement due to it being a second-order approximation, but zero-order hold performs best due to the matrix exponential including much higher orders.

Table E.3 compares the Taylor series expansions of several common discretization methods (these are found using polynomial division). The bilinear transform does best with accuracy trailing off after the third-order term. Forward Euler has no second-order or higher terms, so it undershoots. Backward Euler has twice the second-order term and overshoots the remaining higher order terms as well.

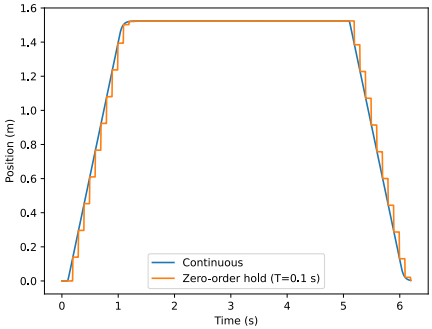


Figure E.21: Zero-order hold of a system response

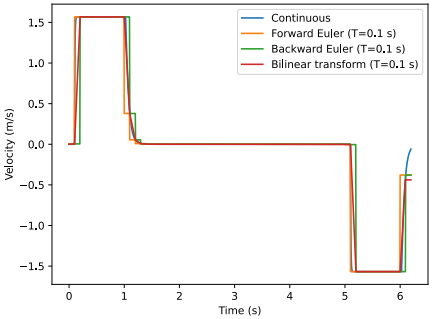


Figure E.22: Discretization methods applied to velocity data

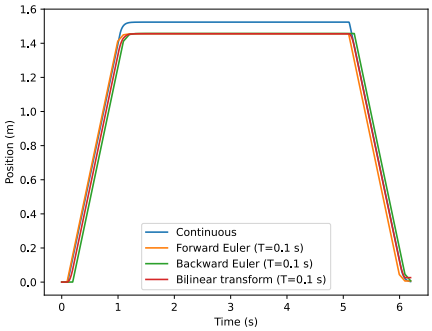


Figure E.23: Position plot of discretization methods applied to velocity data

Method	Conversion to z	Taylor series expansion
Zero-order hold	e^{Ts}	$1 + Ts + \frac{1}{2}T^2s^2 + \frac{1}{6}T^3s^3 + \ldots$
Bilinear	$\frac{1+\frac{1}{2}Ts}{1-\frac{1}{2}Ts}$	$1 + Ts + \frac{1}{2}T^2s^2 + \frac{1}{4}T^3s^3 + \ldots$
Forward Euler	$1 + Ts$	$1 + Ts$
Backward Euler	$\frac{1}{1-Ts}$	$1 + Ts + T^2s^2 + T^3s^3 + \ldots$

Table E.3: Taylor series expansions of discretization methods (scalar case). The zero-order hold discretization method is exact.

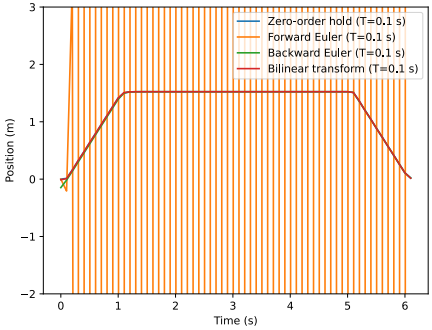


Figure E.24: Sampling methods for system simulation with $T = 0.1$ s

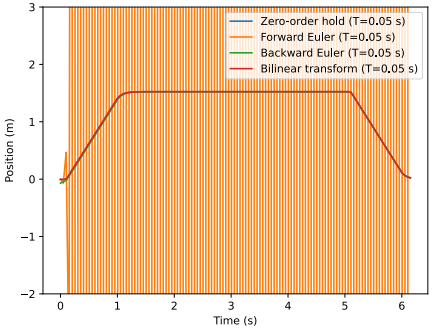


Figure E.25: Sampling methods for system simulation with $T = 0.05$ s

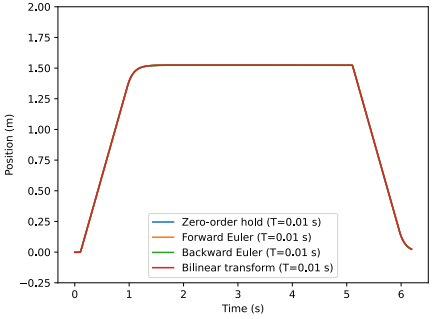


Figure E.26: Sampling methods for system simulation with $T = 0.01$ s

E.6 Phase loss

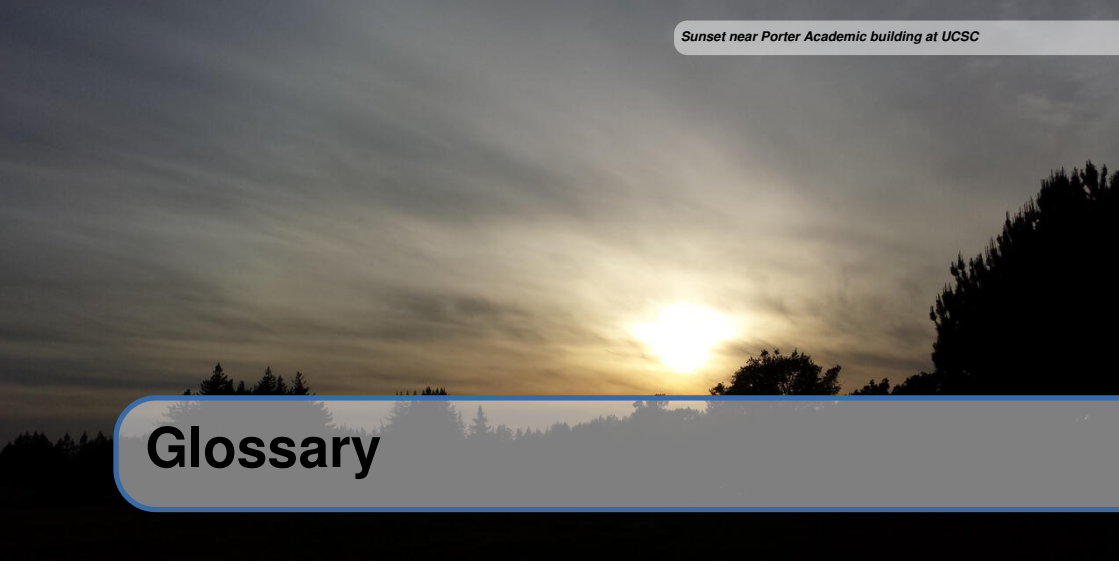
Implementing a discrete control system is easier than implementing a continuous one, but discretization has drawbacks. A microcontroller updates the system input in discrete intervals of duration T; it's held constant between updates. This introduces an average sample delay of $\frac{T}{2}$ that leads to phase loss in the controller. Phase loss is the reduction of phase margin that occurs in digital implementations of feedback controllers from sampling the continuous system at discrete time intervals. As the sample rate of the controller decreases, the phase margin decreases according to $-\frac{T}{2}\omega$ where T is the sample period and ω is the frequency of the system dynamics. Instability occurs if the phase margin of the system reaches zero. Large amounts of phase loss can make a stable controller in the continuous domain become unstable in the discrete domain. Here are a few ways to combat this.

- Run the controller with a high sample rate.
- Designing the controller in the analog domain with enough phase margin to compensate for any phase loss that occurs as part of discretization.
- Convert the plant to the digital domain and design the controller completely in the digital domain.

E.7 System identification with Bode plots

It's straightforward to perform system identification by feeding a system sine waves of increasing frequency, recording the amplitude of the output oscillations, then collating that information into a Bode plot. This plot can be used to create a transfer function, or lead and lag compensators can be applied directly based on the Bode plot.

This page intentionally left blank


Sunset near Porter Academic building at UCSC

Glossary

agent An independent actor being controlled through autonomy or human-in-the-loop (e.g., a robot, aircraft, etc.).

control effort A term describing how much force, pressure, etc. an actuator is exerting.
control input The input of a plant used for the purpose of controlling it.
control law Also known as control policy, is a mathematical formula used by the controller to determine the input u that is sent to the plant. This control law is designed to drive the system from its current state to some other desired state.
control system Monitors and controls the behavior of a system.
controller Applies an input to a plant to bring about a desired system state by driving the difference between a reference signal and the output to zero.

discretization The process by which a continuous (e.g., analog) system or controller design is converted to discrete (e.g., digital).
disturbance An external force acting on a system that isn't included in the system's model.
disturbance rejection The quality of a feedback control system to compensate for external forces to reach a desired reference.

error Reference minus an output or state.

feedback controller Used in positive or negative feedback with a plant to bring about

a desired system state by driving the difference between a reference signal and the output to zero.

feedback gain The gain from the output to an earlier point in a control system diagram.

feedforward controller A controller that injects information about the system's dynamics (like a model does) or the desired movement. The feedforward handles parts of the control actions we already know must be applied to make a system track a reference, then the feedback controller compensates for what we do not or cannot know about the system's behavior at runtime.

gain A proportional value that shows the relationship between the magnitude of an input signal to the magnitude of an output signal at steady-state.

gain margin See section E.3.7 on gain and phase margin.

impulse response The response of a system to the Dirac delta function.

input An input to the plant (hence the name) that can be used to change the plant's state.

linearization A method by which a nonlinear system's dynamics are approximated by a linear system.

localization The process of using measurements of the environment to determine an agent's pose.

model A set of mathematical equations that reflects some aspect of a physical system's behavior.

noise immunity The quality of a system to have its performance or stability unaffected by noise in the outputs (see also: robustness).

observer In control theory, a system that estimates the internal state of a given real system from measurements of the input and output of the real system.

open-loop gain The gain directly from the input to the output, ignoring loops.

output Measurements from sensors.

output-based control Controls the system's state via the outputs.

overshoot The amount by which a system's state surpasses the reference after rising toward it.

phase margin See section E.3.7 on gain and phase margin.

plant The system or collection of actuators being controlled.

pose The position and orientation of an agent in the world, which is represented by all or part of the agent's state.

process variable The term used to describe the output of a plant in the context of PID control.

realization In control theory, this is an implementation of a given input-output behavior as a state-space model.

reference The desired state. This value is used as the reference point for a controller's error calculation.

regulator A controller that attempts to minimize the error from a constant reference in the presence of disturbances.

rise time The time a system takes to initially reach the reference after applying a step input.

robustness The quality of a feedback control system to remain stable in response to disturbances and uncertainty.

setpoint The term used to describe the reference of a PID controller.

settling time The time a system takes to settle at the reference after a step input is applied.

state A characteristic of a system (e.g., velocity) that can be used to determine the system's future behavior.

state feedback Uses state instead of output in feedback.

steady-state error Error after system reaches equilibrium.

step input A system input that is 0 for $t < 0$ and a constant greater than 0 for $t \geq 0$. A step input that is 1 for $t \geq 0$ is called a unit step input.

step response The response of a system to a step input.

stochastic process A process whose model is partially or completely defined by random variables.

system A term encompassing a plant and its interaction with a controller and observer, which are treated as a single entity. Mathematically speaking, a system maps inputs to outputs through a linear combination of states.

system response The behavior of a system over time for a given input.

time-invariant The system's fundamental response does not change over time.

tracking In control theory, the process of making the output of a control system follow the reference.

unity feedback A feedback network in a control system diagram with a feedback gain of 1.

This page intentionally left blank

Bibliography

Online

[1] Wikimedia Commons. *Runge-Kutta methods: Explicit Runge-Kutta methods*. URL: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods#Explicit_Runge.E2.80.93Kutta_methods (visited on 06/18/2020) (cited on page 101).

[2] Wikipedia Commons. *Linear-quadratic regulator*. URL: https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator (visited on 03/24/2018) (cited on page 266).

[3] Wikipedia Commons. *Matrix calculus identities*. URL: https://en.wikipedia.org/wiki/Matrix_calculus#Identities (visited on 08/03/2021) (cited on page 44).

[4] Declan Freeman-Gleason. *A Dive into WPILib Trajectories*. 2020. URL: https://pietroglyph.github.io/trajectory-presentation/ (cited on page 244).

[5] Sean Humbert. *Why do we have to linearize around an equilibrium point?* URL: https://www.cds.caltech.edu/%7Emurray/courses/cds101/fa02/faq/02-10-09_linearization.html (visited on 07/12/2018) (cited on page 106).

[6] Simon J. Julier and Jeffrey K. Uhlmann. *A New Extension of the Kalman Filter to Nonlinear Systems*. URL: https://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf (visited on 09/29/2019) (cited on page 171).

[7] Matthew Kelly. *An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation*. URL: https://www.matthewpeterkelly.com/research/MatthewKelly_IntroTrajectoryOptimization_SIAM_Review_2017.pdf (visited on 08/27/2019) (cited on page 255).

[8] Edgar Kraft. *A Quaternion-based Unscented Kalman Filter for Orientation Tracking*. URL: https://kodlab.seas.upenn.edu/uploads/Arun/UKFpaper.pdf (visited on 07/11/2018) (cited on page 171).

[9] Roger Labbe. *Kalman and Bayesian Filters in Python*. URL: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/ (visited on 04/29/2020) (cited on page 131).

[10] Charles F. Van Loan. *Computing Integrals Involving the Matrix Exponential*. URL: https://file.tavsys.net/control/papers/Computing%20Integrals%20Involving%20the%20Matrix%20Exponential%2C%20Van%20Loan.pdf (visited on 12/20/2021) (cited on page 85).

[11] Andrew W. Long et al. *The Banana Distribution is Gaussian: A Localization Study with Exponential Coordinates*. 2008. URL: https://rpk.lcsr.jhu.edu/wp-content/uploads/2014/08/p34_Long12_The-Banana-Distribution.pdf (cited on page 131).

[12] Simon J.A. Malham. *An introduction to Lagrangian and Hamiltonian Mechanics*. URL: https://www.macs.hw.ac.uk/~simonm/mechanics.pdf (visited on 08/23/2016) (cited on page 219).

[13] Rudolph van der Merwe and Eric A. Wan. *The Square-Root Unscented Kalman Filter for State and Parameter Estimation*. URL: https://www.researchgate.net/publication/3908304_The_Square-Root_Unscented_Kalman_Filter_for_State_and_Parameter-Estimation (visited on 04/25/2022) (cited on page 171).

[14] J. Nocedal and S. Wright. *Numerical Optimization*. 2008. URL: https://www.math.uci.edu/~qnie/Publications/NumericalOptimization.pdf (cited on page 255).

[15] Christoph Sprunk. *Planning Motion Trajectories for Mobile Robots Using Splines*. 2008. URL: http://www2.informatik.uni-freiburg.de/~lau/students/Sprunk2008.pdf (cited on page 244).

[16] Russ Tedrake. *Underactuated Robotics*. 2019. URL: https://underactuated.mit.edu (cited on page 128).

[17] Russ Tedrake. *Chapter 9. Linear Quadratic Regulators*. URL: https://underactuated.mit.edu/lqr.html (visited on 03/22/2020) (cited on page 266).

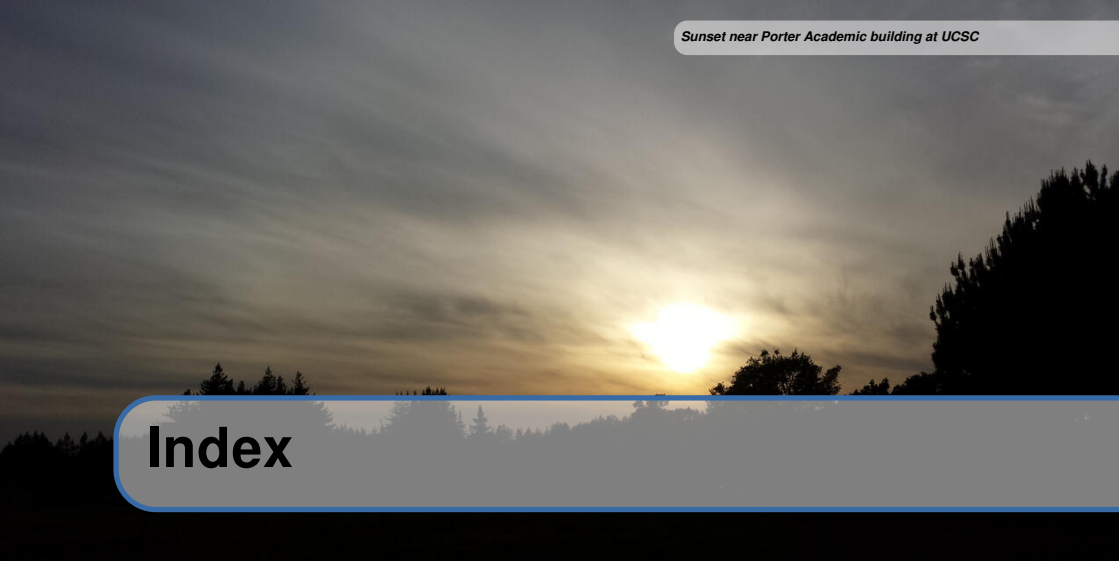
[18] Gabriel A. Terejanu. *Unscented Kalman Filter Tutorial*. URL: https://web.archive.org/web/20220000000000*/https://cse.sc.edu/~terejanu/files/tutorialUKF.pdf (visited on 09/28/2022) (cited on page 171).

[19] Michael Triantafyllou. *lec19.pdf | Maneuvering and Control of Surface and Underwater Vehicles (13.49)*. URL: https://ocw.mit.edu/courses/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/resources/lec19/ (visited on 04/02/2022) (cited on page 91).

[20] Jeffrey Uhlmann. *First-Hand:The Unscented Transform*. URL: https://ethw.org/First-Hand:The_Unscented_Transform (visited on 07/27/2021) (cited on page 171).

[21] Eric A. Wan and Rudolph van der Merwe. *The Unscented Kalman Filter for Nonlinear Estimation*. URL: https://groups.seas.harvard.edu/courses/cs281/papers/unscented.pdf (visited on 06/26/2018) (cited on page 171).

This page intentionally left blank



Index

This page intentionally left blank

When I was a high school student on FIRST Robotics Competition (FRC) team 3512, I had to learn control theory from scattered internet sources that either weren't rigorous enough or assumed too much prior knowledge. After I took graduate-level control theory courses from University of California, Santa Cruz for my bachelor's degree, I realized that the concepts weren't difficult when presented well, but the information wasn't broadly accessible outside academia.

I wanted to fix the information disparity so more people could appreciate the beauty and elegance I saw in control theory. This book streamlines the learning process to make that possible.

I wrote the initial draft of this book as a final project for an undergraduate technical writing class I took at UCSC in Spring 2017 (CMPE 185). It was a 13-page IEEE-formatted paper intended as a reference manual and guide to state-space control that summarized the three graduate controls classes I had taken that year. I kept working on it the following year to flesh it out, and it eventually became long enough to make into a proper book. I've been adding to it ever since as I learn new things.

I contextualized the material within FRC because it's always been a significant part of my life, and it's a useful application sandbox. I maintain implementations of many of this book's tools in the FRC standard library (WPILib).

– Tyler Veness