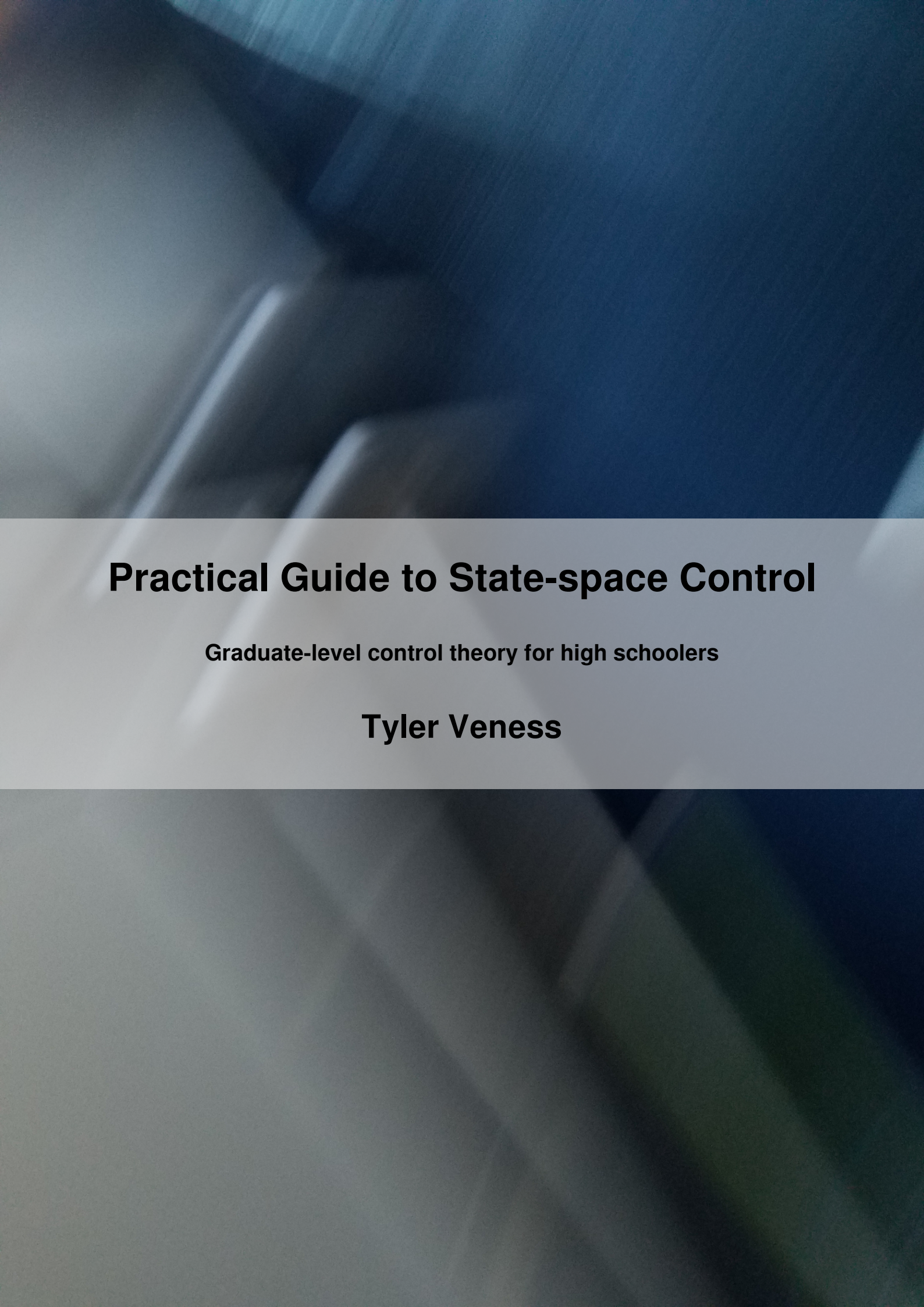


Practical Guide to State-space Control

Graduate-level control theory for high schoolers

Tyler Veness

Practical Guide to State-space Control

Graduate-level control theory for high schoolers

Tyler Veness

HTTPS://GITHUB.COM/CALCMOGUL/STATE-SPACE-GUIDE

Generated on December 25, 2018 from commit 5fb11ec. The latest version can be downloaded from https://file.tavsys.net/control/state-space-guide.pdf.


Tree next to Oakes Circle at UCSC

Contents

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

Preface

Motivation

I am the software mentor for a FIRST Robotics Competition (FRC) team. My responsibilities for that include teaching programming, software engineering practices, and applications of control theory. The curriculum I developed as of the spring of 2017 (located at `https://csweb.frc3512.com/ci/`) taught rookies enough to be minimally competitive, but it provided no formal avenues of growth for veteran students.

Also, out of a six week build season, the software team usually only got a few days with the completed robot due to poor build schedule management. This led to two problems. First, two days is only enough time to verify basic software functionality, not test and tune feedback controllers. Second, this was also the first time the robot's electromechanical systems have been tested after integration, so any issues that arose consumed valuable software integration time while the team traced the problem to a mechanical, electrical, or software cause.

This book expands my curriculum to cover control theory topics I learned in my graduate-level engineering classes at University of California, Santa Cruz. It introduces state-space controllers and serves as a practical guide for formulating and implementing them. Since state-space control utilizes a system model, both problems mentioned earlier can be addressed. Basic software functionality can be tested against it and feedback controllers can be tuned automatically based on system constraints. This allows software teams to test their work much earlier in the build season in a controlled environment as well as save time during feedback controller design, implementation, and testing.

I originally started writing this book because I felt that the topic wasn't difficult, but the information for it wasn't easily accessible. When I was a high school robotics team member, I had to learn everything from scratch through various internet sources and eventually from college courses as part of my bachelor's degree. Control theory has a certain beauty to it, and I want more people to appreciate it the way I do. Therefore, my goal is to make the learning process quicker and easier for the team members who come after me by collating all the relevant information.

Intended audience

This guide is intended to make an advanced engineering topic approachable so it can be applied by those who aren't experts in control theory. My intended audience is high school students who are members of a FIRST Robotics Competition team. As such, they will likely have passing knowledge of PID control and have basic proficiency in programming. This guide will expand their incomplete understanding of control theory to include the fundamentals of classical control theory, enough linear algebra to understand the notation and mechanics of modern control, and finally how to apply modern control to design challenges they regularly see in their FRC robots from year to year.

Acknowledgements

I would like to thank my controls engineering instructors Dejan Milutinović and Gabriel Elkaim of University of California, Santa Cruz. They taught their classes from a pragmatic perspective focused on application and intuition that I appreciated. I would also like to thank Dejan Milutinović for introducing me to the field of control theory and both of my instructors for teaching me what it means to be a controls engineer.

Thanks to Austin Schuh from FRC team 971 for providing the final continuous state-space models used in the examples section.

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 The 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, "Kinematics and dynamics," introduces the basic calculus and physics concepts required to derive the models used in the later chapters. It walks through the derivations for several common FRC subsystems.

Part II, "Classical control theory," introduces the basics of control theory, teaches the fundamentals of PID controller design, 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.

Part III, "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 II 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 I are converted to state-space representation, implemented, and tested with a digital controller.

Part III 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 IV, "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 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. Motion profiles are introduced.

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 The 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.

0.3.1 The role of classical control theory

The sections on classical control theory are only included to develop geometric intuition for the mathematical machinery of modern control theory. Many tools exclusive to classical control theory (root locus, Bode plots, Nyquist plots, etc.) aren't useful for or relevant to the examples presented, so they serve only to complicate the learning process.

Classical control theory is interesting in that one can perform stability and robustness analyses and design reasonable controllers for systems on the back of a napkin. It's also useful for controlling systems which don't have a model. One can generate a Bode plot of a system by feeding in sine waves of increasing frequency and recording the amplitude of the output oscillations. This data can be used to create a transfer function or lead and lag compensators can be applied directly based on the Bode plot. However, computing power is much more plentiful nowadays; we should take advantage of this in the design phase and use the more modern tools that enables when it makes sense.

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.

0.3.2 An integrated approach to nonlinear control theory

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 The mindset of an egoless engineer

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.

This page intentionally left blank

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

1. Introduction to control theory

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 (see section 0.4 for more on engineering).

1.1 Nomenclature

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

The system or collection of actuators being controlled by a control system is called the plant. Controllers which don't include information measured from the plant's output are called open-loop controllers. Controllers which incorporate information fed back from the plant's output are called closed-loop controllers or feedback controllers.

Table 1.1 describes how the terms input and output apply to plants versus controllers and what letters are commonly associated with each when working with them. Namely, that the terms input and output are defined with respect to the plant, not the controller.

	Plant	**Controller**
Inputs	$u(t)$	$r(t)$, $y(t)$
Outputs	$y(t)$	$u(t)$

Table 1.1: Plant versus controller nomenclature

Kinematics and dynamics

This page intentionally left blank

2. Calculus

This book uses derivatives and integrals occasionally to represent small changes in values over small changes in time and the infintessimal sum of values over time respectively. If you are interested in more information after reading this chapter, 3Blue1Brown does a fantastic job of introducing them in his *Essence of calculus* video series [12]. We recommend watching videos 1 through 3 and 7 through 11 from that playlist for a solid foundation. The Taylor series (presented in video 11) will be used in chapter 10.

2.1 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 2.1: Notation for derivatives of $f(x)$

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

2.1.1 Power rule

$$f(x) = x^n$$

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

2.1.2 Product rule

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

$$\begin{aligned} h(x) &= f(x)g(x) \\ h'(x) &= f'(x)g(x) + f(x)g'(x) \end{aligned}$$

2.1.3 Chain rule

This is for taking the derivative of nested expressions.

$$\begin{aligned} h(x) &= f(g(x)) \\ h'(x) &= f'(g(x)) \cdot g'(x) \end{aligned}$$

For example

$$\begin{aligned} h(x) &= (3x + 2)^5 \\ h'(x) &= 5\,(3x + 2)^4 \cdot (3) \\ h'(x) &= 15\,(3x + 2)^4 \end{aligned}$$

2.2 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 2.2.

$$\begin{aligned} &\int e^{at}\,dt \\ &\frac{1}{a}e^{at} + C \end{aligned}$$

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.

$$\begin{aligned} &\int_0^t e^{at}\,dt \\ &\left(\frac{1}{a}e^{at} + C\right)\Big|_0^t \\ &\left(\frac{1}{a}e^{at} + C\right) - \left(\frac{1}{a}e^{a\cdot 0} + C\right) \end{aligned}$$

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

When we do this, the constant cancels out.

2.2.1 Change of variables

Change of variables substitutes an expression with a single variable to make the calculation more straightforward. 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.

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

2.3 Tables

2.3.1 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 2.2: Common derivatives and integrals

3. Dynamics

3.1 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.

3.2 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.

3.3 Curvilinear motion

3.3.1 Two-wheeled vehicle kinematics

The mapping from v and ω to the left and right wheel velocities v_L and v_R are 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. For this derivation, we'll assume the robot is facing in the $\hat{j}$ direction.

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").

First, we'll derive v_l.

$$\begin{aligned}
\vec{v}_l &= v_c\hat{j} + \omega\hat{k} \times -r_b\hat{i} \\
\vec{v}_l &= v_c\hat{j} - \omega r_b\hat{j} \\
\vec{v}_l &= (v_c - \omega r_b)\hat{j} \\
|\vec{v}_l| &= v_c - \omega r_b \\
v_l &= v_c - \omega r_b
\end{aligned} \tag{3.1}$$

Next, we'll derive v_r.

$$\begin{aligned}
\vec{v}_r &= v_c\hat{j} + \omega\hat{k} \times r_b\hat{i} \\
\vec{v}_r &= v_c\hat{j} + \omega r_b\hat{j} \\
\vec{v}_r &= (v_c + \omega r_b)\hat{j} \\
|\vec{v}_r| &= v_c + \omega r_b \\
v_r &= v_c + \omega r_b
\end{aligned} \tag{3.2}$$

4. Model 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.

We'll use the second approach in this book.

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 brushed 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.

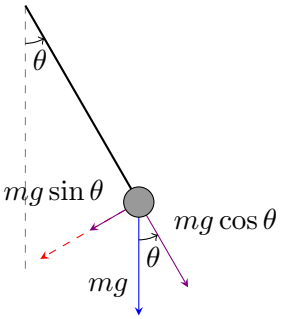
4.1 Pendulum

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.

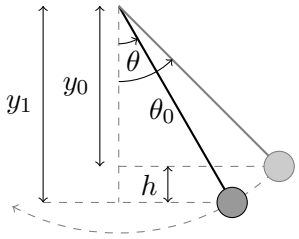
4.1.1 Force derivation

Consider figure 4.1a, 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



(a) Force diagram of a pendulum



(b) Trigonometry of a pendulum

Figure 4.1: Pendulum force diagrams

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.

$$\begin{aligned} s &= l\theta \\ v &= \frac{ds}{dt} = l\frac{d\theta}{dt} \\ a &= \frac{d^2s}{dt^2} = l\frac{d^2\theta}{dt^2} \end{aligned} \tag{4.1}$$

Therefore

$$l\frac{d^2\theta}{dt^2} = -g\sin\theta$$

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta$$
$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

4.1.2 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.

$$|\mathbf{L}| = mr^2\omega$$
$$|\mathbf{L}| = ml^2\frac{d\theta}{dt}$$
$$\frac{d}{dt}|\mathbf{L}| = ml^2\frac{d^2\theta}{dt^2}$$

According to $\tau = \frac{d\mathbf{L}}{dt}$, we can just compare the magnitudes.

$$-mgl\sin\theta = ml^2\frac{d^2\theta}{dt^2}$$
$$-\frac{g}{l}\sin\theta = \frac{d^2\theta}{dt^2}$$
$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

which is the same result from force analysis.

4.1.3 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 (4.1), this equation can be rewritten in terms of $\frac{d\theta}{dt}$.

$$\begin{aligned} v = l\frac{d\theta}{dt} &= \sqrt{2gh} \\ \frac{d\theta}{dt} &= \frac{2gh}{l} \end{aligned} \tag{4.2}$$

where h is the vertical distance the pendulum fell. Look at figure 4.1b, 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 (4.2) 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.

$$\begin{aligned}
\frac{d}{dt}\frac{d\theta}{dt} &= \frac{d}{dt}\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)} \\
\frac{d^2\theta}{dt^2} &= \frac{1}{2}\frac{-\frac{2g}{l}\sin\theta}{\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}}\frac{d\theta}{dt} \\
\frac{d^2\theta}{dt^2} &= \frac{1}{2}\frac{-\frac{2g}{l}\sin\theta}{\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}}\sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)} \\
\frac{d^2\theta}{dt^2} &= -\frac{g}{l}\sin\theta \\
\ddot{\theta} &= -\frac{g}{l}\sin\theta
\end{aligned}$$

which is the same result from force analysis.

4.2 DC brushed motor

We will be deriving a first-order model for a DC brushed 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 section 7.4 for a demonstration of this for a real DC brushed motor).

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.

4.2.1 Equations of motion

The circuit for a DC brushed motor is shown in figure 4.2.

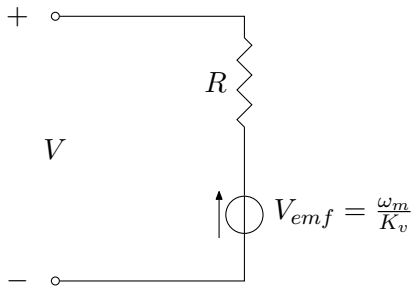


Figure 4.2: DC brushed motor circuit

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_m 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.

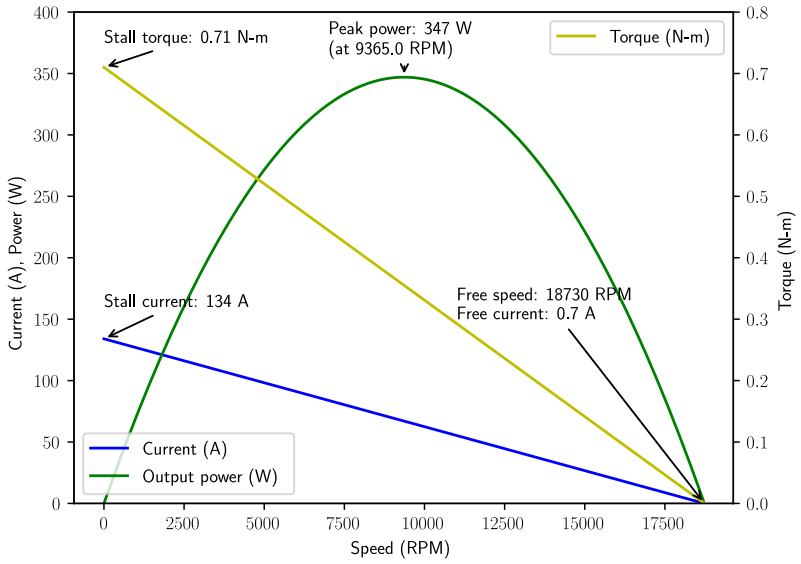


Figure 4.3: Example motor datasheet for 775pro

$$V = IR + \frac{\omega_m}{K_v} \tag{4.3}$$

The mechanical relation for a DC brushed motor is

$$\tau_m = K_t I \tag{4.4}$$

where τ_m is the torque produced by the motor in Newton-meters and K_t is the torque constant in Newton-meters per Amp. Therefore

$$I = \frac{\tau_m}{K_t}$$

Substitute this into equation (4.3).

$$V = \frac{\tau_m}{K_t} R + \frac{\omega_m}{K_v} \tag{4.5}$$

4.2.2 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 4.3 is an example. Data for the most common motors in FRC can be found at `https://motors.vex.com`.

To find K_t

$$\tau_m = K_t I$$
$$K_t = \frac{\tau_m}{I}$$

$$K_t = \frac{\tau_{m,stall}}{I_{stall}} \tag{4.6}$$

where $\tau_{m,stall}$ is the stall torque and I_{stall} is the stall current of the motor from its datasheet.

To find R, recall equation (4.3).

$$V = IR + \frac{\omega_m}{K_v}$$

When the motor is stalled, $\omega_m = 0$.

$$\begin{aligned} V &= I_{stall}R \\ R &= \frac{V}{I_{stall}} \end{aligned} \tag{4.7}$$

where I_{stall} is the stall current of the motor and V is the voltage applied to the motor at stall.

To find K_v, again recall equation (4.3).

$$\begin{aligned} V &= IR + \frac{\omega_m}{K_v} \\ V - IR &= \frac{\omega_m}{K_v} \\ K_v &= \frac{\omega_m}{V - IR} \end{aligned}$$

When the motor is spinning under no load

$$K_v = \frac{\omega_{m,free}}{V - I_{free}R} \tag{4.8}$$

where $\omega_{m,free}$ 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 $\omega_{m,free}$, and I_{free} is the current drawn by the motor under no load.

If several identical motors are being used in one gearbox for a mechanism, multiply the stall torque by the number of motors.

4.3 Elevator

4.3.1 Equations of motion

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

Gear ratios are written as output over input, so G is greater than one in figure 4.4.

Based on figure 4.4

$$\tau_m G = \tau_p \tag{4.9}$$

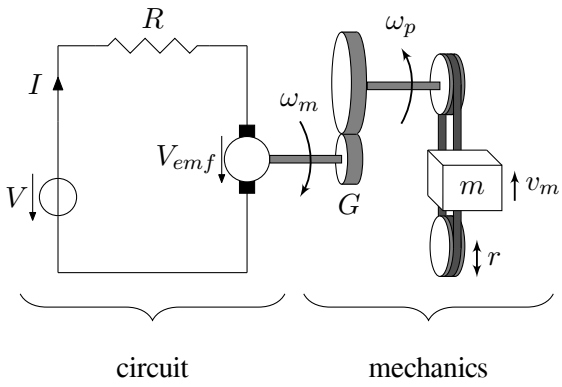


Figure 4.4: Elevator system diagram

where G is the gear ratio between the motor and the pulley and τ_p is the torque produced by the pulley.

$$rF_m = \tau_p \tag{4.10}$$

where r is the radius of the pulley. Substitute equation (4.9) into equation (4.5).

$$V = \frac{\frac{\tau_p}{G}}{K_t} R + \frac{\omega_m}{K_v}$$
$$V = \frac{\tau_p}{GK_t} R + \frac{\omega_m}{K_v}$$

Substitute in equation (4.10).

$$V = \frac{rF_m}{GK_t} R + \frac{\omega_m}{K_v} \tag{4.11}$$

The angular velocity of the motor armature ω_m is

$$\omega_m = G\omega_p \tag{4.12}$$

where ω_p is the angular velocity of the pulley. The velocity of the mass (the elevator carriage) is

$$v_m = r\omega_p$$

$$\omega_p = \frac{v_m}{r} \tag{4.13}$$

Substitute equation (4.13) into equation (4.12).

$$\omega_m = G\frac{v_m}{r} \tag{4.14}$$

Substitute equation (4.14) into equation (4.11).

$$V = \frac{rF_m}{GK_t}R + \frac{G\frac{v_m}{r}}{K_v}$$
$$V = \frac{RrF_m}{GK_t} + \frac{G}{rK_v}v_m$$

Solve for F_m.

$$\frac{RrF_m}{GK_t} = V - \frac{G}{rK_v}v_m$$
$$F_m = \left(V - \frac{G}{rK_v}v_m\right)\frac{GK_t}{Rr}$$
$$F_m = \frac{GK_t}{Rr}V - \frac{G^2K_t}{Rr^2K_v}v_m \tag{4.15}$$

$$\sum F = ma_m \tag{4.16}$$

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_m is the acceleration of the elevator carriage.

$$ma_m = F_m$$

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.

$$ma_m = \left(\frac{GK_t}{Rr}V - \frac{G^2K_t}{Rr^2K_v}v_m\right)$$
$$a_m = \frac{GK_t}{Rrm}V - \frac{G^2K_t}{Rr^2mK_v}v_m \tag{4.17}$$

4.4 Flywheel

4.4.1 Equations of motion

This flywheel consists of a DC brushed 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 4.5.

We will start with the equation derived earlier for a DC brushed motor, equation (4.5).

$$V = \frac{\tau_m}{K_t}R + \frac{\omega_m}{K_v}$$

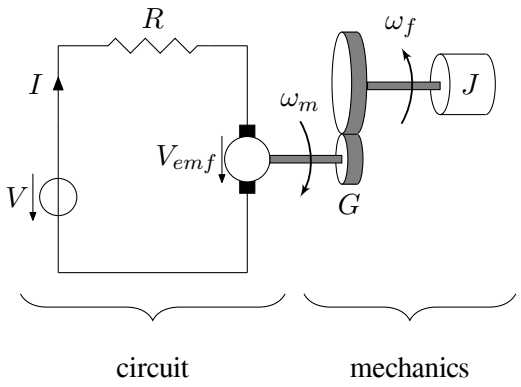


Figure 4.5: Flywheel system diagram

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.

$$\begin{aligned} V &= \frac{R}{K_t}\tau_m + \frac{1}{K_v}\omega_m \\ \frac{R}{K_t}\tau_m &= V - \frac{1}{K_v}\omega_m \\ \tau_m &= \frac{K_t}{R}V - \frac{K_t}{K_v R}\omega_m \end{aligned}$$

Since $\tau_m G = \tau_f$ and $\omega_m = G\omega_f$

$$\begin{aligned} \left(\frac{\tau_f}{G}\right) &= \frac{K_t}{R}V - \frac{K_t}{K_v R}(G\omega_f) \\ \frac{\tau_f}{G} &= \frac{K_t}{R}V - \frac{GK_t}{K_v R}\omega_f \\ \tau_f &= \frac{GK_t}{R}V - \frac{G^2 K_t}{K_v R}\omega_f \end{aligned} \tag{4.18}$$

The torque applied to the flywheel is defined as

$$\tau_f = J\dot{\omega}_f \tag{4.19}$$

where J is the moment of inertia of the flywheel and $\dot{\omega}_f$ is the angular acceleration. Substitute equation (4.19) into equation (4.18).

$$(J\dot{\omega}_f) = \frac{GK_t}{R}V - \frac{G^2 K_t}{K_v R}\omega_f$$

$$\dot{\omega}_f = \frac{GK_t}{RJ}V - \frac{G^2K_t}{K_vRJ}\omega_f \tag{4.20}$$

4.5 Drivetrain

4.5.1 Equations of motion

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

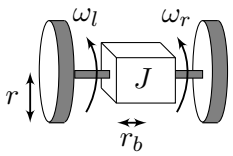


Figure 4.6: Drivetrain system diagram

From equation (4.18) of the flywheel model derivations

$$\tau = \frac{GK_t}{R}V - \frac{G^2K_t}{K_vR}\omega \tag{4.21}$$

where τ is the torque applied by one wheel of the drivetrain, G is the gear ratio of the drivetrain, 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 drivetrain side along the ground and r is the drivetrain 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 $C_4 = \frac{G_rK_t}{Rr}$.

$$F_l = C_1v_l + C_2V_l \tag{4.22}$$

$$F_r = C_3 v_r + C_4 V_r \tag{4.23}$$

First, find the sum of forces.

$$\begin{aligned}
\sum F &= ma \\
F_l + F_r &= m\dot{v} \\
F_l + F_r &= m\frac{\dot{v_l} + \dot{v_r}}{2} \\
\frac{2}{m}(F_l + F_r) &= \dot{v_l} + \dot{v_r} \\
\dot{v_l} &= \frac{2}{m}(F_l + F_r) - \dot{v_r}
\end{aligned} \tag{4.24}$$

Next, find the sum of torques.

$$\begin{aligned}
\sum \tau &= J\dot{\omega} \\
\tau_l + \tau_r &= J\left(\frac{\dot{v_r} - \dot{v_l}}{2r_b}\right)
\end{aligned}$$

where r_b is the radius of the drivetrain.

$$\begin{aligned}
(-r_b F_l) + (r_b F_r) &= J\frac{\dot{v_r} - \dot{v_l}}{2r_b} \\
-r_b F_l + r_b F_r &= \frac{J}{2r_b}(\dot{v_r} - \dot{v_l}) \\
-F_l + F_r &= \frac{J}{2r_b^2}(\dot{v_r} - \dot{v_l}) \\
\frac{2r_b^2}{J}(-F_l + F_r) &= \dot{v_r} - \dot{v_l} \\
\dot{v_r} &= \dot{v_l} + \frac{2r_b^2}{J}(-F_l + F_r)
\end{aligned}$$

Substitute in equation (4.24) to obtain an expression for $\dot{v_r}$.

$$\begin{aligned}
\dot{v_r} &= \left(\frac{2}{m}(F_l + F_r) - \dot{v_r}\right) + \frac{2r_b^2}{J}(-F_l + F_r) \\
2\dot{v_r} &= \frac{2}{m}(F_l + F_r) + \frac{2r_b^2}{J}(-F_l + F_r) \\
\dot{v_r} &= \frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(-F_l + F_r) \qquad (4.25) \\
\dot{v_r} &= \frac{1}{m}F_l + \frac{1}{m}F_r - \frac{r_b^2}{J}F_l + \frac{r_b^2}{J}F_r \\
\dot{v_r} &= \left(\frac{1}{m} - \frac{r_b^2}{J}\right)F_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)F_r \qquad (4.26)
\end{aligned}$$

Substitute equation (4.25) back into equation (4.24) to obtain an expression for $\dot{v}_l$.

$$\begin{aligned}
\dot{v}_l &= \frac{2}{m}(F_l + F_r) - \left(\frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(-F_l + F_r)\right) \\
\dot{v}_l &= \frac{1}{m}(F_l + F_r) - \frac{r_b^2}{J}(-F_l + F_r) \\
\dot{v}_l &= \frac{1}{m}(F_l + F_r) + \frac{r_b^2}{J}(F_l - F_r) \\
\dot{v}_l &= \frac{1}{m}F_l + \frac{1}{m}F_r + \frac{r_b^2}{J}F_l - \frac{r_b^2}{J}F_r \\
\dot{v}_l &= \left(\frac{1}{m} + \frac{r_b^2}{J}\right)F_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)F_r
\end{aligned} \tag{4.27}$$

Now, plug the expressions for F_l and F_r into equation (4.26).

$$\begin{aligned}
\dot{v}_r &= \left(\frac{1}{m} - \frac{r_b^2}{J}\right)F_l + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)F_r \\
\dot{v}_r &= \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_1 v_l + C_2 V_l) + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_3 v_r + C_4 V_r)
\end{aligned} \tag{4.28}$$

Now, plug the expressions for F_l and F_r into equation (4.27).

$$\begin{aligned}
\dot{v}_l &= \left(\frac{1}{m} + \frac{r_b^2}{J}\right)F_l + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)F_r \\
\dot{v}_l &= \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_1 v_l + C_2 V_l) + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_3 v_r + C_4 V_r)
\end{aligned} \tag{4.29}$$

4.6 Single-jointed arm

4.6.1 Equations of motion

This single-jointed arm consists of a DC brushed 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 4.7.

We will start with the equation derived earlier for a DC brushed motor, equation (4.5).

$$V = \frac{\tau_m}{K_t}R + \frac{\omega_m}{K_v}$$

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.

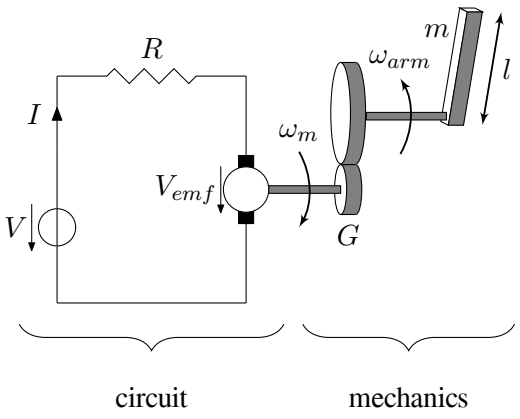


Figure 4.7: Single-jointed arm system diagram

$$
\begin{aligned}
V &= \frac{R}{K_t}\tau_m + \frac{1}{K_v}\omega_m \\
\frac{R}{K_t}\tau_m &= V - \frac{1}{K_v}\omega_m \\
\tau_m &= \frac{K_t}{R}V - \frac{K_t}{K_v R}\omega_m
\end{aligned}
$$

Since $\tau_m G = \tau_{arm}$ and $\omega_m = G\omega_{arm}$

$$
\begin{aligned}
\left(\frac{\tau_{arm}}{G}\right) &= \frac{K_t}{R}V - \frac{K_t}{K_v R}(G\omega_f) \\
\frac{\tau_{arm}}{G} &= \frac{K_t}{R}V - \frac{GK_t}{K_v R}\omega_{arm} \\
\tau_{arm} &= \frac{GK_t}{R}V - \frac{G^2 K_t}{K_v R}\omega_{arm}
\end{aligned} \tag{4.30}
$$

The angular velocity of the arm is defined as

$$\tau_{arm} = J\dot{\omega}_{arm} \tag{4.31}$$

where J is the moment of inertia of the arm and $\dot{\omega}_{arm}$ is the angular acceleration. Substitute equation (4.31) into equation (4.30).

$$
\begin{aligned}
(J\dot{\omega}_{arm}) &= \frac{GK_t}{R}V - \frac{G^2 K_t}{K_v R}\omega_{arm} \\
\dot{\omega}_{arm} &= \frac{GK_t}{RJ}V - \frac{G^2 K_t}{K_v RJ}\omega_{arm}
\end{aligned} \tag{4.32}
$$

J can be approximated as the moment of inertia of a thin rod rotating around one end. Therefore

$$J = \frac{1}{3}ml^2 \tag{4.33}$$

where m is the mass of the arm and l is the length of the arm.

4.7 Rotating claw

4.7.1 Equations of motion

This claw consists of independent upper and lower jaw pieces each driven by its own DC brushed motor.

This page intentionally left blank

II Classical control theory

This page intentionally left blank

5. Control system basics

5.1 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 5.1 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.

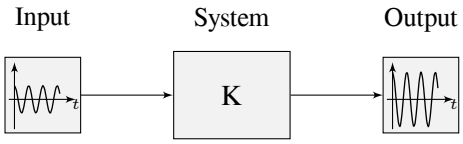


Figure 5.1: Demonstration of system with a gain of $K = 2$

5.2 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 5.2 is an example of one.

The open-loop gain is the total gain from the sum node at the input to the output branch. The feedback gain is the total gain from the output back to the input sum node. The circle's output is the sum of its inputs.

Figure 5.3 is a block diagram with more formal notation in a feedback configuration.

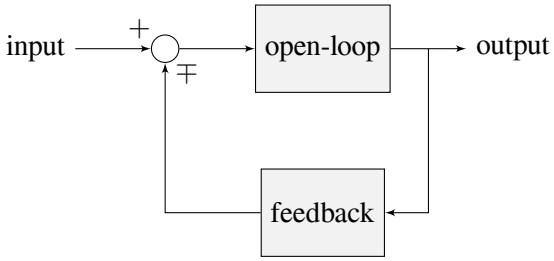


Figure 5.2: Block diagram with nomenclature

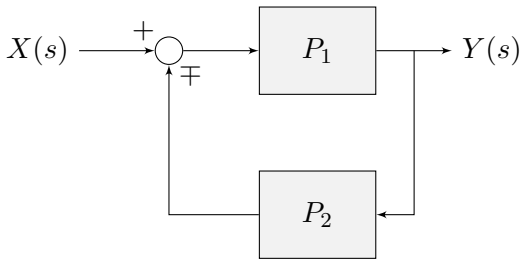


Figure 5.3: Feedback block diagram

Theorem 5.2.1 — Closed-loop gain for a feedback loop.

$$\frac{Y(s)}{X(s)} = \frac{P_1}{1 \mp P_1 P_2} \tag{5.1}$$

See appendix D.1 for a derivation.

5.3 Why feedback control?

Let's say we are controlling a DC brushed 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.

6. PID controllers

6.1 The definition

Negative feedback loops drive the difference between the reference and output to zero. Figure 6.1 shows a block diagram for a system controlled by a PID controller.

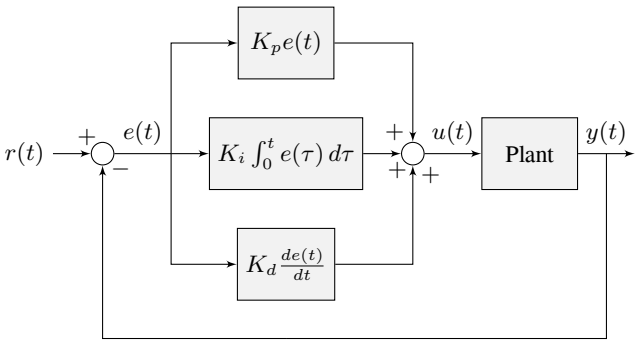


Figure 6.1: PID controller diagram

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

PID control has three gains acting on the error.

6.1.1 Proportional gain

The *Proportional* gain compensates for the current error. This gain acts like a "software-defined spring". 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 (0 in this case). For P control of a DC motor toward zero, F is proportional to the voltage applied, k is the proportional constant from the P term, and x is the measured position. In other words, the "force" with which the proportional gain pulls the system's output toward the reference is proportional to the error.

6.1.2 Integral gain

The *Integral* gain compensates for past error (i.e., steady-state error). It integrates the error over time and adds the current total times K_i to the control input. When the system is close to the reference in steady-state, the proportional term is too small to pull the output to the reference and the derivative term is zero. The integral gain is commonly used to address this problem.

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.

6.1.3 Derivative gain

The *Derivative* gain compensates for future error by slowing the controller down if the error is changing over time. This gain acts like a "software-defined damper". These are commonly seen on door closers, and their damping force increases linearly with velocity.

6.2 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 6.2.

For the step responses in figure 6.2, 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 fastest rise time without overshooting the reference.

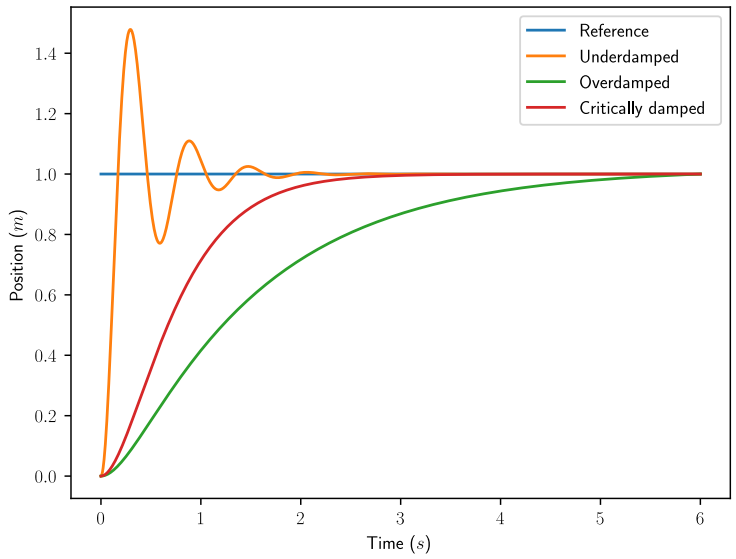


Figure 6.2: PID controller response types

6.3 Types of PID controllers

PID controller inputs that are different orders of derivatives, such as position and velocity, affect the system response differently. Below is the standard form for a position PID controller.

Definition 6.3.1 — Position PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de}{dt} \tag{6.1}$$

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

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.

If the position controller formulation is measuring and controlling velocity instead, the error $e(t)$ becomes $\frac{de}{dt}$. Substituting this into equation (6.1) yields

$$\begin{aligned} u(t) &= K_p \frac{de}{dt} + K_i \int_0^t \frac{de}{d\tau}\, d\tau + K_d \frac{d^2e}{dt^2} \\ u(t) &= K_p \frac{de}{dt} + K_i e(t) + K_d \frac{d^2e}{dt^2} \end{aligned} \tag{6.2}$$

So the velocity $\frac{de}{dt}$ is integrated by the integral term. By the fundamental theorem of calculus, the derivative and integral cancel because they are inverse operations. This produces just the error $e(t)$, so the K_i term has the effect of proportional control. Furthermore, the K_p term in equation (6.2) behaves like a derivative term for velocity PID control due to the $\frac{de}{dt}$. Letting $e_v = \frac{de}{dt}$, this means the velocity controller is analogous to theorem 6.3.1.

Note this isn't strictly true because the fundamental theorem of calculus requires that $e(t)$ is continuous, but when we actually implement the controller, $e(t)$ is discrete. Therefore, the result here is only correct up to the accuracy of the iterative integration method used. We'll discuss approximations like these in section 10.3 and how they affect controller behavior.

Theorem 6.3.1 — Velocity PID controller.

$$u(t) = K_p e_v(t) + K_i \int_0^t e_v(\tau)\, d\tau \tag{6.3}$$

where $e_v(t)$ is the velocity error at the current time t, τ is the integration variable, K_p is now the derivative gain, and K_i is now the proportional gain.

Integral control for the velocity is analogous to the throttle pedal on a car. One must hold the throttle pedal (the control input) at a nonzero value to keep the car traveling at the reference velocity.

R *Note:* For a real velocity control implementation, a steady-state feedforward and proportional control are preferred to integral control.

Read `https://en.wikipedia.org/wiki/PID_controller` for more information on PID control theory.

6.4 General control theory perspective

PID control defines setpoint as the desired position and process variable as the measured position. Control theory has more general terms for these: reference and output respectively.

The derivative term is commonly used to "slow down" the system if it's already heading toward the reference. We will explore what K_p and K_d are really doing for a two-state system (position and velocity) and why K_d acts that way.

First, we will rearrange the equation for a PD controller.

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

where u_k is the control input at timestep k and e_k is the error at timestep k. e_k is defined as $e_k = r_k - x_k$ where r_k is the reference and x_k is the current state at timestep k.

$$\begin{aligned}
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - x_k) - (r_{k-1} - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - x_k - r_{k-1} + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{r_k - r_{k-1} - x_k + x_{k-1}}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \frac{(r_k - r_{k-1}) - (x_k - x_{k-1})}{dt} \\
u_k &= K_p(r_k - x_k) + K_d \left(\frac{r_k - r_{k-1}}{dt} - \frac{x_k - x_{k-1}}{dt}\right)
\end{aligned}$$

Notice how $\frac{r_k - r_{k-1}}{dt}$ is the velocity of the reference. By the same reasoning, $\frac{x_k - x_{k-1}}{dt}$ is the system's velocity at a given timestep. That means the K_d term of the PD controller is driving the estimated velocity to the reference velocity. If the reference is constant, that means the K_d term is trying to drive the velocity of the system to zero.

However, K_p and K_d are controlling the same actuator, and their effects conflict with each other; K_p is trying to make the system move while K_d is trying to stop it. If K_p is larger than K_d, one is in effect slowing down the response of the controller during transients with the hope of decreasing overshoot and settling time. If one makes K_d much larger than K_p, K_d overpowers K_p to bring the system to a stop. However, when the velocity is low enough, K_p is stronger and starts accelerating the system again. This oscillatory behavior in the velocity repeats as the system moves toward the reference.

6.5 Manual tuning

These steps apply for both position and velocity PID controllers.

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

If the controller settles at an output above or below the reference, increase K_i such that the controller reaches the reference in a reasonable amount of time. However, a steady-state feedforward is preferred to 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 reference, 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.

6.6 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 and can drive up to two states to references by using both the proportional and integral terms. We'll revisit this in subsection 7.6.2.

This page intentionally left blank

Grass clearing by Interdisciplinary Sciences building and Thimann Labs at UCSC

7. Laplace domain analysis

This chapter briefly discusses what transfer functions are, how the locations of poles and zeroes affects system response and stability, and how controllers affect pole locations. The case studies cover various aspects of PID control using the algebraic approach of transfer functions.

7.1 The 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 7.1.

Notice how this complex waveform can be represented just by three frequencies. They show up as Dirac delta functions[1] in the frequency domain with the area underneath them equal to their contribution (see figure 7.2).

Since Euler's identity states that $e^{i\theta} = \cos\theta + i\sin\theta$, we can represent frequencies as complex numbers on a number line (we will use $e^{j\omega} = \cos\omega + j\sin\omega$ for notational consistency going forward). For example, $400\,Hz$ would be e^{j400}. The frequency domain just uses the complex exponent $400j$.

[1] 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.

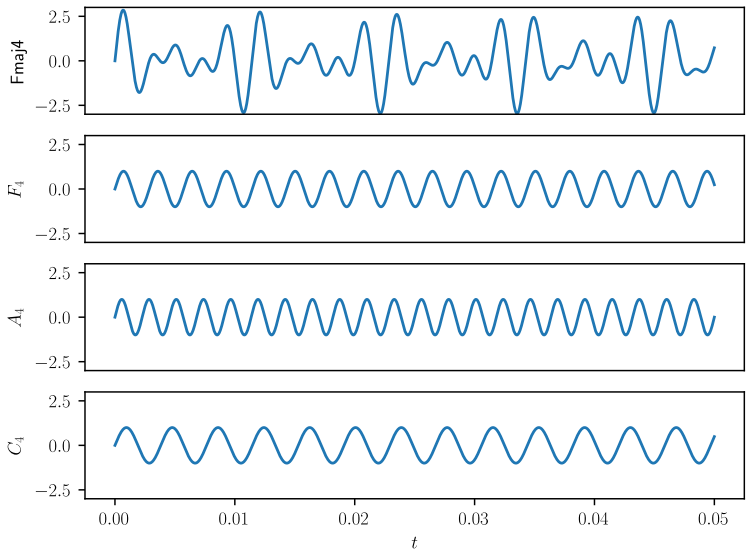


Figure 7.1: Frequency decomposition of Fmajor4 chord

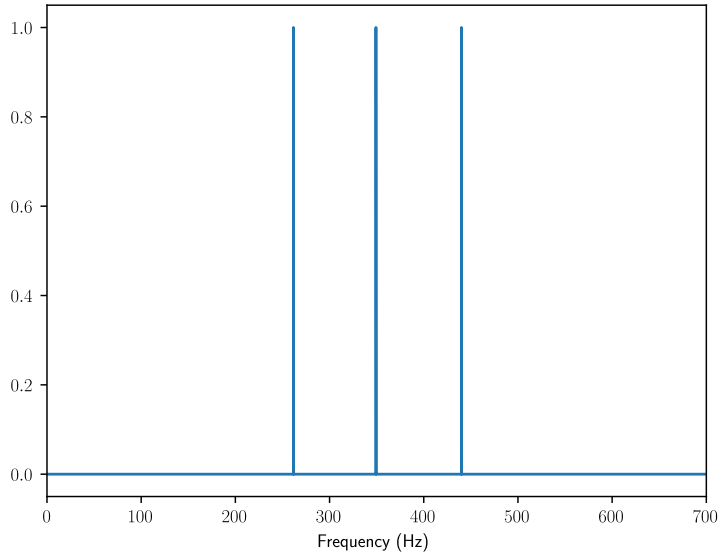


Figure 7.2: Fourier transform of Fmajor4 chord

7.2 The 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 cooresponds to the x-axis and the imaginary part $j\omega$ cooresponds to the y-axis (see figure 7.3).

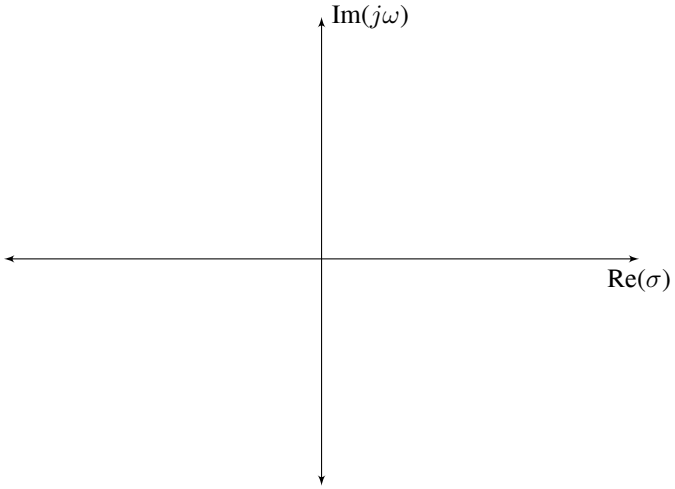


Figure 7.3: Laplace domain

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 oscillation decays and the system converges to zero (i.e., a decaying exponential). Figure 7.7 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 7.4).

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 7.5 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 $\frac{1}{x}$).

7.2.1 The 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)

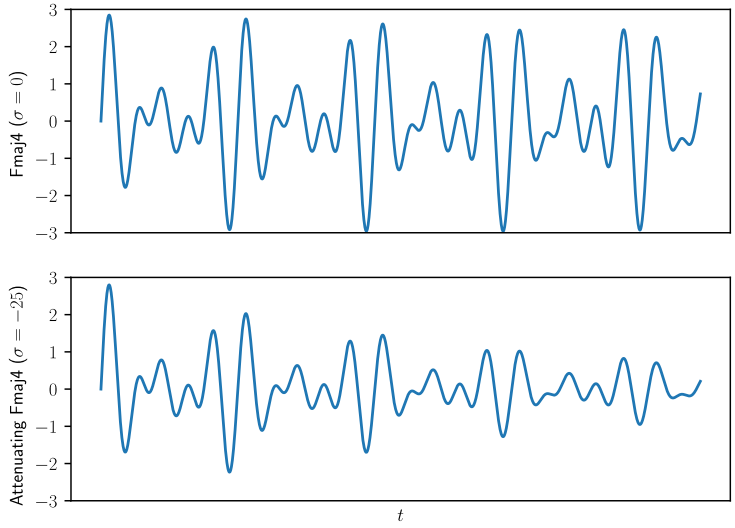


Figure 7.4: Fmajor4 chord at $\sigma = 0$ and $\sigma = -25$

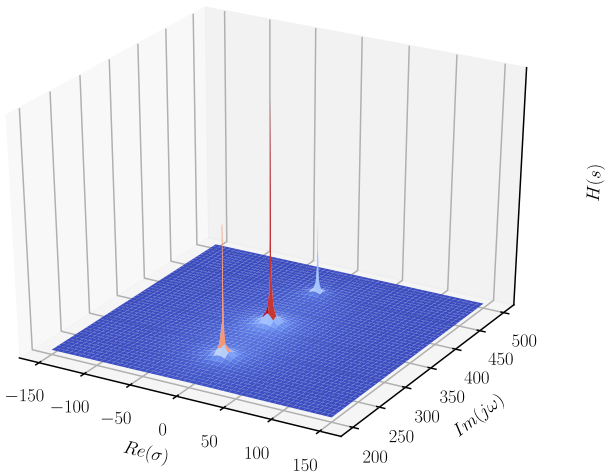


Figure 7.5: Laplace transform of Fmajor4 chord plotted in 3D

are shown in table 7.1. Of particular note are the Laplace transforms for the derivative, unit step[2], and exponential decay. We can see that a derivative is equivalent to multiplying by s, and an integral is equivalent to multiplying by $\frac{1}{s}$. We'll discuss the decaying exponential in subsection 7.3.1.

	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 7.1: Common Laplace transforms and Laplace transform properties with zero initial conditions

7.3 Transfer functions

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 equation and rearranging the terms to obtain a ratio of the output variable to the input variable. Equation (7.1) is an example of a transfer function.

$$H(s) = \frac{\overbrace{(s - 9 + 9i)(s - 9 - 9i)}^{zeroes}}{\underbrace{s(s + 10)}_{poles}} \tag{7.1}$$

7.3.1 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 7.6).

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.

You may notice that the factors representing poles look a lot like the Laplace transform for a decaying exponential. This is why the time domain responses of systems are typically decaying exponentials. One differential equation which can produce this kind of response is $\dot{x} = ax$ where $\dot{x}$ is the derivative of x and $a < 0$.

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

[2]The unit step $u(t)$ is defined as 0 for $t < 0$ and 1 for $t \geq 0$.

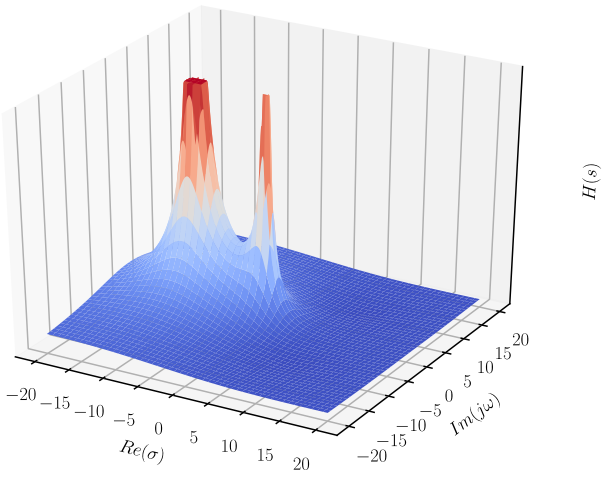


Figure 7.6: Equation 7.1 plotted in 3D

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 7.7 shows the impulse responses in the time domain for transfer functions with various pole locations. They all have an initial condition of 1.

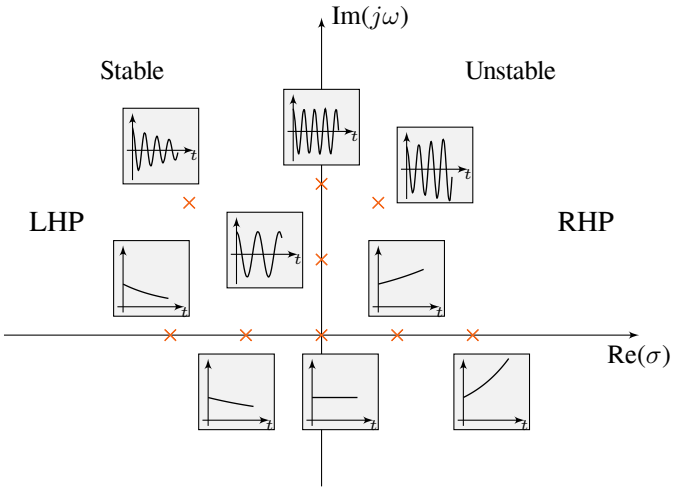


Figure 7.7: Impulse response vs pole location

When a system is stable, its output may oscillate but it converges to steady-state. When a system is marginally stable, its output oscillates at a constant amplitude forever. When a system is unstable, its output grows without bound.

7.3.2 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

Location	Stability
Left Half-plane (LHP)	Stable
Imaginary axis	Marginally stable
Right Half-plane (RHP)	Unstable

Table 7.2: Pole location and stability

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 dymamics 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.

7.3.3 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.

7.3.4 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.

The transfer function of figure 7.8, a control system diagram with feedback, from input to output is

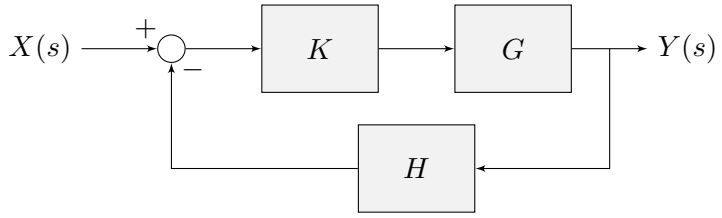


Figure 7.8: Feedback controller block diagram

$X(s)$	input	H	measurement transfer function
K	controller gain	$Y(s)$	output
G	plant transfer function		

$$G_{cl}(s) = \frac{Y(s)}{X(s)} = \frac{KG}{1 + KGH} \tag{7.2}$$

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 (see appendix D.1 for a derivation). As another example, the transfer function from the input to the error is

$$G_{cl}(s) = \frac{E(s)}{X(s)} = \frac{1}{1 + KGH} \tag{7.3}$$

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.

7.4 Root locus

In closed-loop, the poles can be moved around by adjusting the controller gain, but the zeroes stay put. The root locus shows where the poles will go as the gain for a P controller is increased and tells us for what range of gains the controller will be stable. As the controller gain is increased, poles can move toward negative infinity (figure 7.9), move toward each other then split toward asymptotes (figure 7.10), or move toward zeroes (figure 7.11). The system in figure 7.11 becomes unstable as the gain is increased.

We won't be using root locus plots for any of our control systems analysis later, but it does help provide an intuition for what controllers actually do to a system.

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 brushed motor from voltage to position

$$G(s) = \frac{K}{s((Js + b)(Ls + R) + K^2)}$$

where $J = 3.2284 \times 10^{-6}\ kg\text{-}m^2$, $b = 3.5077 \times 10^{-6}\ N\text{-}m\text{-}s$, $K_e = K_t = 0.0274\,V/rad/s$, $R = 4\,\Omega$, and $L = 2.75 \times 10^{-6}\,H$.

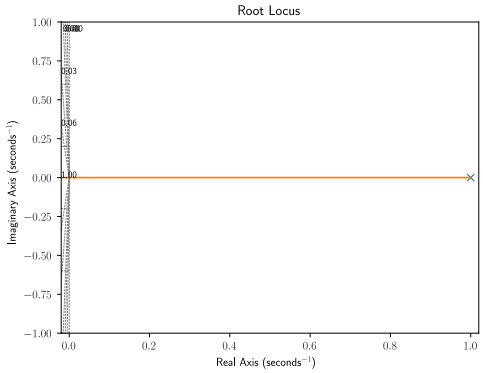


Figure 7.9: Root locus showing pole moving toward negative infinity

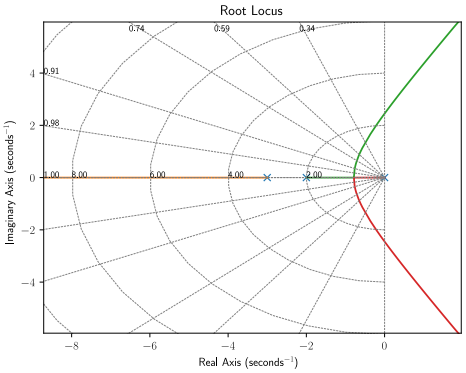


Figure 7.10: Root locus showing poles moving toward asymptotes

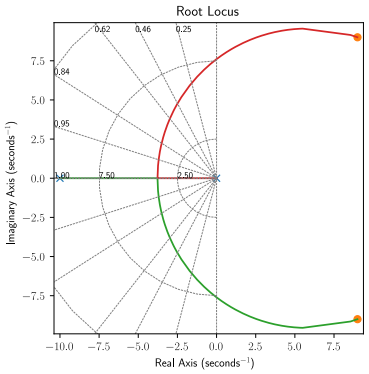


Figure 7.11: Root locus of equation (7.1) showing poles moving toward zeroes.

This plant has the root locus shown in figure 7.12. In proportional feedback, the plant is unstable for large values of K. However, if we remove the unstable pole by setting L in the transfer function to zero, we get the root locus in figure 7.13. 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 (see figures 7.14 and 7.15). 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.

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.

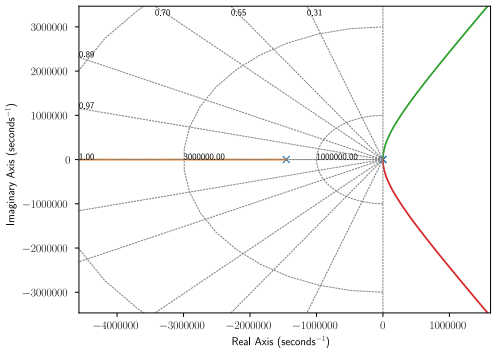


Figure 7.12: Root locus of second-order DC brushed motor plant

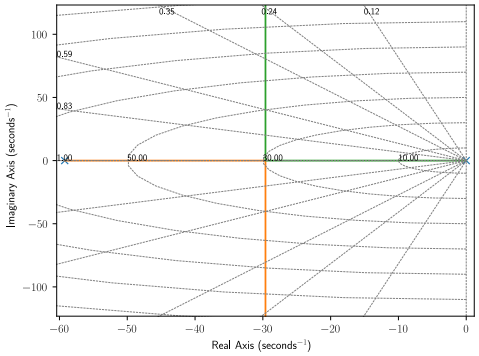


Figure 7.13: Root locus of first-order DC brushed motor plant

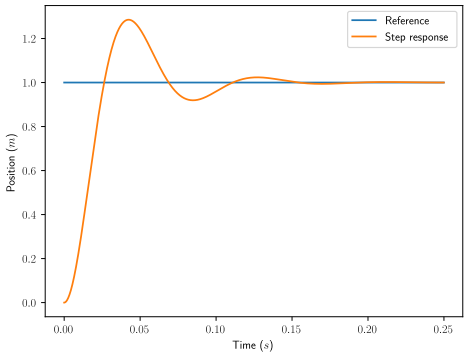


Figure 7.14: Step response of second-order DC brushed motor plant

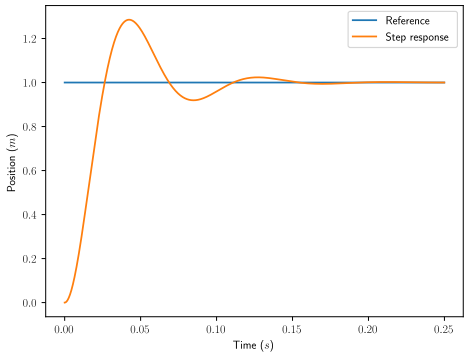


Figure 7.15: Step response of first-order DC brushed motor plant

7.5 Actuator saturation

Recall that a 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 controller to apply. When the actuator limits are reached, the controller acts as if the gain has been temporarily reduced.

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.

7.6 Case studies of Laplace domain analysis

We'll be using equation (7.4), the transfer function for a PID controller, in the case studies.

$$K(s) = K_p + \frac{K_i}{s} + K_d s \tag{7.4}$$

Remember, multiplication by $\frac{1}{s}$ corresponds to an integral in the Laplace domain and multiplication by s corresponds to a derivative.

7.6.1 Steady-state error

To demonstrate the problem of steady-state error, we will use a DC brushed motor controlled by a velocity PID controller. A DC brushed motor has a transfer function from voltage (V) to angular velocity ($\dot{\theta}$) of

$$G(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2} \tag{7.5}$$

First, we'll try controlling it with a P controller defined as

$$K(s) = K_p$$

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

$$\lim_{s \to 0} sH(s) \tag{7.6}$$

since steady-state has an input freuency of zero.

$$e_{ss} = \lim_{s \to 0} sE(s)$$

$$
\begin{aligned}
e_{ss} &= \lim_{s \to 0} s \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} V(s) \\
e_{ss} &= \lim_{s \to 0} s \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} \frac{1}{s} \\
e_{ss} &= \lim_{s \to 0} \frac{1}{1 + \frac{K_p K}{(Js+b)(Ls+R)+K^2}} \\
e_{ss} &= \frac{1}{1 + \frac{K_p K}{(J(0)+b)(L(0)+R)+K^2}} \\
e_{ss} &= \frac{1}{1 + \frac{K_p K}{bR+K^2}}
\end{aligned} \tag{7.7}
$$

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.

$$
\begin{aligned}
\frac{E(s)}{V(s)} &= \frac{1}{1 + K(s)G(s)} \\
E(s) &= \frac{1}{1 + K(s)G(s)} V(s) \\
E(s) &= \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} \left(\frac{1}{s}\right) \\
e_{ss} &= \lim_{s \to 0} s \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} \left(\frac{1}{s}\right) \\
e_{ss} &= \lim_{s \to 0} \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} \\
e_{ss} &= \lim_{s \to 0} \frac{1}{1 + \left(K_p + \frac{K_i}{s}\right)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} \frac{s}{s} \\
e_{ss} &= \lim_{s \to 0} \frac{s}{s + (K_p s + K_i)\left(\frac{K}{(Js+b)(Ls+R)+K^2}\right)} \\
e_{ss} &= \frac{0}{0 + (K_p(0) + K_i)\left(\frac{K}{(J(0)+b)(L(0)+R)+K^2}\right)} \\
e_{ss} &= \frac{0}{K_i \frac{K}{bR+K^2}}
\end{aligned}
$$

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 (7.7) 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.

7.6.2 Flywheel PID 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

$$V = IR + \frac{\omega}{K_v} \tag{7.8}$$

$$\tau = IK_t \tag{7.9}$$

$$\tau = J\frac{d\omega}{dt} \tag{7.10}$$

For an explanation of where these equations come from, read section 4.2.

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

Substitute equation (7.9) into equation (7.8).

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

Substitute in equation (7.10).

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

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

$$
\begin{aligned}
V &= \frac{J\frac{d\omega}{dt}}{K_t}R + \frac{\omega}{K_v} \\
V - \frac{\omega}{K_v} &= \frac{J\frac{d\omega}{dt}}{K_t}R \\
\frac{d\omega}{dt} &= \frac{K_t}{JR}\left(V - \frac{\omega}{K_v}\right) \\
\frac{d\omega}{dt} &= -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}V
\end{aligned}
$$

Now take the Laplace transform.

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}V \tag{7.11}$$

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 loop.

$$V = K_p(\omega_{goal} - \omega)$$

Substitute this controller into equation (7.11).

$$s\omega = -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}K_p(\omega_{goal} - \omega)$$

Solve for the transfer function $H(s) = \frac{\omega}{\omega_{goal}}$.

$$
\begin{aligned}
s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}(\omega_{goal} - \omega) \\
s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_tK_p}{JR}\omega_{goal} - \frac{K_tK_p}{JR}\omega \\
\left(s + \frac{K_t}{JRK_v} + \frac{K_tK_p}{JR}\right)\omega &= \frac{K_tK_p}{JR}\omega_{goal}
\end{aligned}
$$

$$\frac{\omega}{\omega_{goal}} = \frac{\frac{K_t K_p}{JR}}{\left(s + \frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)}$$

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 all of its poles anywhere.

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. In Austin Schuh's experience with 971's robots, these are pretty good assumptions.

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 (7.11).

$$\begin{aligned}
s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_t}{JR}\left(K_p(\omega_{goal} - \omega) + K_d s(\omega_{goal} - \omega)\right) \\
s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_t K_p}{JR}(\omega_{goal} - \omega) + \frac{K_t K_d s}{JR}(\omega_{goal} - \omega) \\
s\omega &= -\frac{K_t}{JRK_v}\omega + \frac{K_t K_p}{JR}\omega_{goal} - \frac{K_t K_p}{JR}\omega + \frac{K_t K_d s}{JR}\omega_{goal} - \frac{K_t K_d s}{JR}\omega
\end{aligned}$$

Collect the common terms on separate sides and refactor.

$$\begin{aligned}
s\omega + \frac{K_t K_d s}{JR}\omega + \frac{K_t}{JRK_v}\omega + \frac{K_t K_p}{JR}\omega &= \frac{K_t K_p}{JR}\omega_{goal} + \frac{K_t K_d s}{JR}\omega_{goal} \\
\left(s\left(1 + \frac{K_t K_d}{JR}\right) + \frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)\omega &= \frac{K_t}{JR}\left(K_p + K_d s\right)\omega_{goal} \\
\frac{\omega}{\omega_{goal}} &= \frac{\frac{K_t}{JR}\left(K_p + K_d s\right)}{\left(s\left(1 + \frac{K_t K_d}{JR}\right) + \frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}\right)}
\end{aligned}$$

So, we added a zero at $-\frac{K_p}{K_d}$ and moved our pole to $-\frac{\frac{K_t}{JRK_v} + \frac{K_t K_p}{JR}}{1 + \frac{K_t K_d}{JR}}$. This isn't progress. We've added more complexity to our system and, practically speaking, gotten nothing good out of it. Zeroes should

be avoided if at all possible because they amplify unwanted high frequency modes of the system. At least this is a stable zero, but it's still undesirable.

In summary, derivative doesn't help on a flywheel. K_d may help if the real system isn't ideal, but we don't suggest relying on that.

7.7 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 these plots for controller design, we'll just cover what gain and phase margin are in a general sense and how they are used. We'll be discussing how discretization affects phase margin in section 10.1.

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 [14]. He has other videos too on classical control methods like Bode and Nyquist plots that we recommend.

II Modern control theory

This page intentionally left blank

Hills by northbound freeway between Santa Maria and Ventura

8. 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. First, watch 3Blue1Brown's preview video for the *Essence of linear algebra* video series (5 minutes) [4]. The goal here is to provide an intuitive, geometric understanding of linear algebra as a method of linear transformations.

While only a subset of the material from the videos will be presented here that we think is relevant to this book, we highly suggest watching the whole series [3].

R The following sections are essentially transcripts of the video content for those who don't want to watch an hour of YouTube videos. However, we suggest watching the videos instead because animations are better at conveying the geometric intuition involved than text.

8.1 Vectors

8.1.1 What is a vector?

The fundamental building block for linear algebra is the vector. Broadly speaking, there are three distinct but related ideas about vectors: the physics student perspective, the computer science student perspective, and the mathmetician's perspective.

The physics student perspective is that vectors are arrows pointing in space. A given vector is defined by its length and direction, but as long as those two facts are the same, you can move it around and it's still the same vector. Vectors in the flat plane are two-dimensional, and those sitting in broader space that we live in are three-dimensional.

The computer science perspective is that vectors are ordered lists of numbers. For example, let's say you were doing some analytics about house prices and the only features you cared about where square footage and price. You might model each house with a pair of numbers where the first indicates square footage and the second indicates price. Notice the order matters here. In the lingo, you'd be

modeling houses as two-dimensional vectors where, in this context, vector is a synonym for list, and what makes it two-dimensional is that the length of that list is two.

The mathematician, on the other hand, seeks to generalize both these views by saying that a vector can be anything where there's a sensible notion of adding two vectors and multiplying a vector by a number (operations that we'll talk about later on). The details of this view are rather abstract and won't be needed for this book as we'll favor a more concrete setting. We bring it up here because it hints at the fact that the ideas of vector addition and multiplication by numbers will play an important role throughout linear algebra.

8.1.2 Geometric interpretation of vectors

Before we talk about vector addition and multiplication by numbers, let's settle on a specific thought to have in mind when we say the word "vector". Given the geometric focus that we're intending here, whenever we introduce a new topic involving vectors, we want you to first think about an arrow, and specifically, think about that arrow inside a coordinate system, like the x-y plane with its tail sitting at the origin. This is slightly different from the physics student perspective where vectors can freely sit anywhere they want in space. In linear algebra, it's almost always the case that your vector will be rooted at the origin. Then, once you understand a new concept in the concept of arrows in space, we'll translate it over to the "list of numbers" point of view, which we can do by considering the coordinates of the vector.

While you may already be familiar with this coordinate system, it's worth walking through explicitly since this is where all of the important back-and-forth happens between the two perspectives of linear algebra. Focusing your attention on two dimensions for the moment, you have a horizontal line called the x-axis and a vertical line called the y-axis. The point at which they intersect is called the origin, which you should think of as the center of space and the root of all vectors. After choosing an arbitrary length to represent one, you make tick marks on each axis to represent this distance. The coordinates of a vector is a pair of numbers that essentially gives instructions for getting from the tail of that vector at the origin to its tip. The first number is how far to move along the x-axis (positive numbers indicating rightward motion and negative numbers indicating leftward motion), and the second number is how far to move parallel to the y-axis after that (positive numbers indicating upward motion and negative numbers indicating downward motion). To distinguish vectors from points, the convention is to write this pair of numbers vertically with square brackets around them. For example:

$$\begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

Every pair of numbers represents one and only one vector, and every vector is associated with one and only one pair of numbers. In three dimensions, there is a third axis called the z-axis which is perpendicular to both the x and y axes. In this case, each vector is associated with an ordered triplet of numbers. The first is how far to move along the x-axis, the second is how far to move parallel to the y-axis, and the third is how far to then move parallel to this new z-axis. For example:

$$\begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

Every triplet of numbers represents one unique vector in space, and every vector in space represents exactly one triplet of numbers.

8.1.3 Vector addition

Back to vector addition and multiplication by numbers. Afterall, every topic in linear algebra is going to center around these two operations. Luckily, each one is straightforward to define. Let's say we have two vectors, one pointing up and a little to the right, and the other one pointing right and down a bit. To add these two vectors, move the second one so that its tail sits at the tip of the first one. Then, if you draw a new vector from the tail of the first one to where the tip of the second one now sits, that new vector is their sum.

This definition of addition, by the way, is one of the only times in linear algebra where we let vectors stray away from the origin, but why is this a reasonable thing to do? Why this definition of addition and not some other one? Each vector represents a sort of movement, a step with a certain distance and direction in space. If you take a step along the first vector, then take a step in the direction and distance described by the second vector, the overall effect is just the same as if you moved along the sum of those two vectors to start with.

You could think about this as an extension of how we think about adding numbers on a number line. One way that we teach students to think about this, say with $2+5$, is to think of moving two steps to the right, followed by another 5 steps to the right. The overall effect is the same as if you just took 7 steps to the right. In fact, let's see how vector addition looks numerically. The first vector here has coordinates $(1, 2)$ and the second has coordinates $(3, -1)$.

$$\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}3\\-1\end{bmatrix}$$

When you take the vector sum using this tip-to-tail method, you can think of a four-step path from the origin to the tip of the second vector: "walk 1 to the right, then 2 up, then 3 to the right, then 1 down." Reorganizing these steps so that you first do all of the rightward motion, then do all of the vertical motion, you can read it as saying, "first move $1+3$ to the right, then move $2+(-1)$ up," so the new vector has coordinates $1+3$ and $2+(-1)$.

$$\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}3\\-1\end{bmatrix} = \begin{bmatrix}1+3\\2+(-1)\end{bmatrix}$$

In general, vector addition in this list-of-numbers conception looks like matching up their terms, and adding each one together.

$$\begin{bmatrix}x_1\\y_1\end{bmatrix} + \begin{bmatrix}x_2\\y_2\end{bmatrix} = \begin{bmatrix}x_1+x_2\\y_1+y_2\end{bmatrix}$$

8.1.4 Scalar-vector multiplication

The other fundamental vector operation is multiplication by a number. Now this is best understood just by looking at a few examples. If you take the number 2, and multiply it by a given vector, you stretch out that vector so that it's two times as long as when you started. If you multiply that vector by, say, $\frac{1}{3}$, you compress it down so that it's $\frac{1}{3}$ of the original length. When you multiply it by a negative number, like -1.8, then the vector is first flipped around, then stretched out by that factor of 1.8.

This process of stretching, compressing, or reversing the direction of a vector is called "scaling", and whenver a number like 2 or $\frac{1}{3}$ or -1.8 acting like this–scaling some vector–we call it a "scalar". In fact, throughout linear algebra, one of the main things numbers do is scale vectors, so it's common to

use the word "scalar" interchangeably with the word "number". Numerically, stretching out a vector by a factor of, say, 2, corresponds to multiplying each of its components by that factor, 2.

$$2 \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$$

So in the conception of vectors as lists of numbers, multiplying a given vector by a scalar means multiplying each one of those components by that scalar.

$$2 \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

R See the corresponding *Essence of linear algebra* video for a more visual presentation (5 minutes) [11].

8.2 Linear combinations, span, and basis vectors

Vector coordinates were probably already familiar to you, but there's another interesting way to think about these coordinates which is central to linear algebra. When given a pair of numbers that's meant to describe a vector, like $(3, 2)$, we want you to think about each coordinate as a scalar, meaning, think about how each one stretches or compresses vectors.

8.2.1 Basis vectors

In the xy-coordinate system, there are two special vectors: the one pointing to the right with length 1, commonly called "i-hat", or the unit vector in the x-direction ($\hat{i}$), and the one pointing straight up, with length 1, commonly called "j-hat", or the unit vector in the y-direction ($\hat{j}$).

Now think of the x-coordinate of our vector as a scalar that scales $\hat{i}$, stretching it by a factor of 3, and the y-coordinate as a scalar that scales $\hat{j}$, flipping it and stretching it by a factor of 2. In this sense, the vectors that these coordinates describe is the sum of two scaled vectors $(3)\hat{i} + (-2)\hat{j}$. This idea of adding together two scaled vectors is a surprisingly important concept. Those two vectors, $\hat{i}$ and $\hat{j}$, have a special name, by the way. Together they're called the *basis* of a coordinate system ($\hat{i}$ and $\hat{j}$ are the "basis vectors" of the xy-coordinate system). When you think about coordinates as scalars, the basis vectors are what those scalars actually scale.

By framing our coordinate system in terms of these two special basis vectors, it raises an interesting and subtle point: we could have chosen different basis vectors and had a completely reasonable, new coordinate system system. For example, take some vector pointing up and to the right, along with some other vector pointing down and to the right, in some way. Take a moment to think about all the different vectors that you can get by choosing two scalars, using each one to scale one of the vectors, then adding together what you get. Which two-dimensional vectors can you reach by altering the choices of scalars? The answer is that you can reach every possible two-dimensional vector. A new pair of basis vectors like this still gives us a valid way to go back and forth between pairs of numbers and two-dimensional vectors, but the association is definitely different from the one that you get using the more standard basis of $\hat{i}$ and $\hat{j}$.

8.2.2 Linear combination

Any time we describe vectors numerically, it depends on an implicit choice of what basis vectors we're using. So any time that you're scaling two vectors and adding them like this, it's called a *linear*

combination of those two vectors. Below is a linear combination of vectors $\vec{v}$ and $\vec{w}$ with scalars a and b.

$$a\vec{v} + b\vec{w}$$

Where does this word "linear" come from? Why does this have anything to do with lines? This isn't the etymology, but if you fix one of those scalars and let the other one change its value freely, the tip of the resulting vector draws a straight line.

8.2.3 Span

Now, if you let both scalars range freely and consider every possible resultant vector, there are three things that can happen. For most pairs of vectors, you'll be able to reach every possible point in the plane; every two-dimensional vector is within your grasp. However, in the unlucky case where your two original vectors happen to line up, the tip of the resulting vector is limited to just a single line passing through the origin. The vectors could also both be zero, in which case the resultant vector is just at the origin.

The set of all possible vectors that you can reach with a linear combination of a given pair of vectors is called the *span* of those two vectors. So, restating what we just discussed in this lingo, the span of most pairs of 2D vectors is all vectors in 2D space, but when they line up, their span is all vectors whose tip sits on a certain line.

Remember how we said that linear algebra revolves around vector addition and scalar multiplication? The span of two vectors is a way of asking, "What are all the possible vectors one can reach using only these two fundamental operations, vector addition and scalar multiplication?"

Thinking about a whole collection of vectors sitting on a line gets crowded, and even more so to think about all two-dimensional vectors at once filling up the plane. When dealing with collections of vectors like this, it's common to represent each one with just a point in space where the tip of the vector was. This way, if you want to think about every possible vector whose tip sits on a certain line, just think about the line itself.

Likewise, to think about all possible two-dimensional vectors at once, conceptualize each one as the point where its tip sits. In effect, you're thinking about the infinite, flat sheet of two-dimensional space itself, leaving the arrows out of it.

In general, if you're thinking about a vector on its own, think of it as an arrow, and if you're dealing with a collection of vectors, it's convenient to think of them all as points. Therefore, for our span example, the span of most pairs of vectors ends up being the entire infinite sheet of two-dimensional space, but if they line up, their span is just a line.

The idea of span gets more interesting if we start thinking about vectors in three-dimensional space. For example, given two vectors in 3D space that are not pointing in the same direction, what does it mean to take their span? Their span is the collection of all possible linear combinations of those two vectors, meaning all possible vectors you get by scaling each vector in some way, then adding them together.

You can imagine turning two different knobs to change the two scalars defining the linear combination, adding the scaled vectors and following the tip of the resulting vector. That tip will trace out a flat sheet cutting through the origin of three-dimensional space. This flat sheet is the span of the two vectors. More precisely, the span of the two vectors is the set of all possible vectors whose tips sit on that flat sheet.

So what happens if we add a third vector and consider the span of all three? A linear combination of three vectors is defined similarly as it is for two; you'll choose three different scalars, scale each of those vectors, then add them all together. The linear combination of $\vec{v}$, $\vec{w}$, and $\vec{u}$ looks like

$$a\vec{v} + b\vec{w} + c\vec{u}$$

where a, b, and c are allowed to vary. Again, the span of these vectors is the set of all possible linear combinations.

Two different things could happen here. If your third vector happens to be sitting on the span of the first two, then the span doesn't change; you're trapped on that same flat sheet. In other words, adding a scaled version of that third vector to the linear combination doesn't give you access to any new vectors. However, if you just randomly choose a third vector, it's almost certainly not sitting on the span of those first two. Then, since it's pointing in a separate direction, it unlocks access to every possible three-dimensional vector. As you scale that new third vector, it moves around that span sheet of the first two, sweeping it through all of space.

Another way to think about it is that you're making full use of the three, freely-changing scalars that you have at your disposal to acces the full three dimensions of space.

8.2.4 Linear dependence and independence

In the case where the third vector was already sitting on the span of the first two, or the case where two vectors happen to line up, we want some terminology to describe the fact that at least one of these vectors is redundant–not adding anything to our span. When there are multiple vectors and one could be removed without reducing the span, the relevant terminology is to say that they are *linearly dependent*.

In other words, one of the vectors can be expressed as a linear combination of the others since it's already in the span of the others.

$$\vec{u} = a\vec{v} + b\vec{w} \text{ for some values of } a \text{ and } b$$

On the other hand, if each vector really does add another dimension to the span, they're said to be *linearly independent*.

$$\vec{w} \neq a\vec{v} \text{ for all values of } a$$

Now with all that terminology, and hopefully some good mental images to go with it, the technical definition of a basis of a space is as follows.

Definition 8.2.1 — Basis of a vector space. The *basis* of a vector space is a set of *linearly independent* vectors that *span* the full space.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (10 minutes) [6].

8.3 Linear transformations and matrices

This section focuses on what linear transformations look like in the case of two dimensions and how they relate to the idea of a matrix-vector multiplication.

$$\begin{bmatrix} 1 & -3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} = \begin{bmatrix} (1)(5) + (-3)(7) \\ (2)(5) + (4)(7) \end{bmatrix}$$

In particular, we want to show you a way to think about matrix-vector multiplication that doesn't rely on memorization of the procedure shown above.

8.3.1 What is a linear transformation?

To start, let's just parse this term "linear transformation". "Transformation" is essentially another name for "function". It's something that takes in inputs and returns an output for each one. Specifically in the context of linear algebra, we consider transformations that take in some vector and spit out another vector.

$$\begin{bmatrix} 5 \\ 7 \end{bmatrix} \longrightarrow L(\vec{v}) \longrightarrow \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

Vector input Vector output

So why use the word "transformation" instead of "function" if they mean the same thing? It's to be suggestive of a certain way to visualize this input-output relation. You see, a great way to understand functions of vectors is to use movement. If a transformation takes some input vector to some output vector, we imagine that input vector moving over to the output vector. Then to understand the transformation as a whole, we might imagine watching every possible input vector move over to its corresponding output vector. it gets really crowded to think about all the vectors all at once, where each one is an arrow. Therefore, as we mentioned in the previous section, it's useful to conceptualize each vector as a single point where its tip sits rather than an arrow. To think about a transformation taking every possible input vector to some output vector, we watch every point in space moving to some other point.

The effect of various transformations moving around all of the points in space gives the feeling of compressing and morphing space itself. As you can imagine though, arbitrary transformations can look complicated. Luckily, linear algebra limits itself to a special type of transformation, ones that are easier to understand, called "linear" transformations. Visually speaking, a transformation is linear if it has two properties: all straight lines must remain as such, and the origin must remain fixed in place. In general, you should think of linear transformations as keeping grid lines parallel and evenly spaced.

8.3.2 Describing transformations numerically

Some transformations are simple to think about, like rotations about the origin. Others are more difficult to describe with words. So how could one describe these transformations numerically? If you were, say, programming some animations to make a video teaching the topic, what formula could you give the computer so that if you give it the coordinates of a vector, it would return the coordinates of where that vectors lands?

$$\begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \longrightarrow ???? \longrightarrow \begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix}$$

You only need to record where the two basis vectors, $\hat{i}$ and $\hat{j}$, each land, and everything else will follow from that. For example, consider the vector v with coordinates $(-1, 2)$, meaning $\vec{v} = -1\hat{i} + 2\hat{j}$. If we play some transformation and follow where all three of these vectors go, the property that grid lines remain parallel and evenly spaced has a really important consequence: the place where $\vec{v}$ lands will be -1 times the vector where $\hat{i}$ landed plus 2 times the vector where $\hat{j}$ landed. In other words, it started off as a certain linear combination of $\hat{i}$ and $\hat{j}$ and it ends up as that same linear combination of where those two vectors landed. This means you can deduce where $\vec{v}$ must go based only on where $\hat{i}$ and $\hat{j}$ each land. For this transformation, $\hat{i}$ lands on the coordinates $(1, -2)$ and $\hat{j}$ lands on the x-axis at the coordinates $(3, 0)$.

$$\text{Transformed } \vec{v} = -1(\text{Transformed } \hat{i}) + 2(\text{Transformed } \hat{j})$$
$$\text{Transformed } \vec{v} = -1 \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 2 \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Adding that all together, you can deduce that $\vec{v}$ has to land on the vector $(5, 2)$.

$$\text{Transformed } \vec{v} = \begin{bmatrix} -1(1) + 2(3) \\ -1(-2) + 2(0) \end{bmatrix}$$
$$\text{Transformed } \vec{v} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

This is a good point to pause and ponder, because it's pretty important. This gives us a technique to deduce where any vectors land, so long as we have a record of where $\hat{i}$ and $\hat{j}$ each land, without needing to watch the transformation itself.

Given a vector with more general coordinates x and y, it will land on x times the vector where $\hat{i}$ lands $(1, -2)$, plus y times the vector where $\hat{j}$ lands $(3, 0)$. Carrying out that sum, you see that it lands at $(1x + 3y, -2x + 0y)$.

$$\hat{i} \to \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad \hat{j} \to \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \to x \begin{bmatrix} 1 \\ -2 \end{bmatrix} + y \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 1x + 3y \\ -2x + 0y \end{bmatrix}$$

Given any vector, this formula will describe where that vector lands.

What all of this is saying is that a two dimensional linear transformation is completely described by just four numbers: the two coordinates for where $\hat{i}$ lands and the two coordinates for where $\hat{j}$ lands. It's common to package these coordinates into a two-by-two grid of numbers, called a two-by-two matrix, where you can interpret the columns as the two special vectors where $\hat{i}$ and $\hat{j}$ each land. If $\hat{i}$ lands on the vector $(3, -2)$ and $\hat{j}$ lands on the vector $(2, 1)$, this two-by-two matrix would be

$$\begin{bmatrix} 3 & 2 \\ -2 & 1 \end{bmatrix}$$

If you're given a two-by-two matrix describing a linear transformation and some specific vector, say $(5, 7)$, and you want to know where that linear transformation takes that vector, you can multiply the coordinates of the vector by the corresponding columns of the matrix, then add together the result.

$$\begin{bmatrix} 3 & 2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} = 5 \begin{bmatrix} 3 \\ -2 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

This corresponds with the idea of adding the scaled versions of our new basis vectors.

Let's see what this looks like in the most general case where your matrix has entries a, b, c, d.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Remember, this matrix is just a way of packaging the information needed to describe a linear transformation. Always remember to interpret that first column, (a, c), as the place where the first basis vector lands and that second column, (b, d), as the place where the second basis vector lands.

When we apply this transformation to some vector (x, y), the result will be x times (a, c) plus y times (b, d). Together, this gives a vector $(ax + by, cx + dy)$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

You could even define this as matrix-vector multiplication when you put the matrix on the left of the vector like it's a function. Then, you could make high schoolers memorize this, without showing them the crucial part that makes it feel intuitive (yes, that was sarcasm). Isn't it more fun to think about these columns as the transformed versions of your basis vectors and to think about the result as the appropriate linear combination of those vectors?

8.3.3 Examples of linear transformations

Let's practice describing a few linear transformations with matrices. For example, if we rotate all of space $90°$ counterclockwise then $\hat{i}$ lands on the coordinates $(0, 1)$ and $\hat{j}$ lands on the coordinates $(-1, 0)$. So the matrix we end up with has the columns $(0, 1)$, $(-1, 0)$.

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

To ascertain what happens to any vector after a $90°$ rotation, you could just multiply its coordinates by this matrix.

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Here's a fun transformation with a special name, called a "shear". In it, $\hat{i}$ remains fixed so the first column of the matrix is $(1, 0)$, but $\hat{j}$ moves over to the coordinates $(1, 1)$ which become the second column of the matrix.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

And, at the risk of being redundant here, figuring out how shear transforms a given vector comes down to multiplying this matrix by that vector.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Let's say we want to go the other way around, starting with a matrix, say with columns $(1, 2)$ and $(3, 1)$, and we want to deduce what its transformation looks like. Pause and take a moment to see if you can imagine it.

$$\begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

One way to do this is to first move $\hat{i}$ to $(1, 2)$. Then, move $\hat{j}$ to $(3, 1)$, always moving the rest of space in such a way that that keeps grid lines parallel and evenly spaced.

Suppose that the vectors that $\hat{i}$ and $\hat{j}$ land on are linearly dependent as in the following matrix (that is, it has linearly dependent columns).

$$\begin{bmatrix} 2 & -2 \\ 1 & -1 \end{bmatrix}$$

If you recall from last section, this means that one vector is a scaled version of the other, so that linear transformation compresses all of 2D space onto the line where those two vectors sit. This is also known as the one-dimensional span of those two linearly dependent vectors.

To sum up, linear transformations are a way to move around space such that the grid lines remain parallel and evenly spaced and such that the origin remains fixed. Delightfully, these transformations can be described using only a handful of numbers: the coordinates of where each basis vector lands. Matrices give us a language to describe these transformations where the columns represent those coordinates and matrix-vector multiplication is just a way to compute what that transformation does to a given vector. The important take-away here is that every time you see a matrix, you can interpret it as a certain transformation of space. Once you really digest this idea, you're in a great position to understand linear algebra deeply. Almost all of the topics coming up, from matrix multiplication to determinants, eigenvalues, etc. will become easier to understand once you start thinking about matrices as transformations of space.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (11 minutes) [7].

8.4 Matrix multiplication as composition

Often, you find yourself wanting to describe the effect of applying one transformation and then another. For example, you may want to describe what happens when you first rotate the plane $90°$ counterclockwise then apply a shear. The overall effect here, from start to finish, is another linear transformation distinct from the rotation and the shear. This new linear transformation is commonly called the "composition" of the two separate transformations we applied, and like any linear transformation, it can be described with a matrix all its own by following $\hat{i}$ and $\hat{j}$. In this example, the ultimate landing spot for $\hat{i}$ after both transformations is $(1, 1)$, so that's the first column of the matrix. Likewise, $\hat{j}$ ultimately ends up at the location $(-1, 0)$, so we make that the second column of the matrix.

$$\begin{bmatrix} 1 & -1 \\ 1 & -0 \end{bmatrix}$$

This new matrix captures the overall effect of applying a rotation then a shear but as one single action rather than two successive ones.

Here's one way to think about that new matrix: if you were to feed some vector through the rotation then the shear, the long way to compute where it ends up is to, first, multiply it on the left by the rotation matrix; then, take whatever you get and multiply that on the left by the shear matrix.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \right)$$

This is, numerically speaking, what it means to apply a rotation then a shear to a given vector, but the result should be the same as just applying this new composition matrix we found to that same vector. This applies to any vector because this new matrix is supposed to capture the same overall effect as the rotation-then-shear action.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \right) = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Based on how things are written down here, it's reasonable to call this new matrix the "product" of the original two matrices.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$$

We can think about how to compute that product more generally in just a moment, but it's way too easy to get lost in the forest of numbers. Always remember that multiplying two matrices like this has the geometric meaning of applying one transformation then another.

One oddity here is that we are reading the transformations from right to left; you first apply the transformation represented by the matrix on the right, then you apply the transformation represented by the matrix on the left. This stems from function notation, since we write functions on the left of variables, so every time you compose two functions, you always have to read it right to left.

Let's look at another example. Take the matrix with columns $(1, 1)$ and $(-2, 0)$.

$$M_1 = \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix}$$

Next, take the matrix with columns $(0, 1)$ and $(2, 0)$.

$$M_2 = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$$

The total effect of applying M_1 then M_2 gives us a new transformation, so let's find its matrix. First, we need to determine where $\hat{i}$ goes. After applying M_1, the new coordinates of $\hat{i}$, by definition, are given by that first column of M_1, namely, $(1, 1)$. To see what happens after applying M_2, multiply the matrix for M_2 by that vector $(1, 1)$. Working it out the way described in the last section, you'll get the vector $(2, 1)$.

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

This will be the first column of the composition matrix. Likewise, to follow $\hat{j}$, the second column of M_1 tells us that it first lands on $(-2, 0)$. Then, when we apply M_2 to that vector, you can work out the matrix-vector product to get $(0, -2)$.

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = -2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0 \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \end{bmatrix}$$

This will be the second column of the composition matrix.

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & -2 \end{bmatrix}$$

8.4.1 General matrix multiplication

Let's go through that same process again, but this time, we'll use variable entries in each matrix, just to show that the same line of reasoning works for any matrices. This is more symbol-heavy, but it should be pretty satisfying for anyone who has previously been taught matrix multiplication the more rote way.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

To follow where $\hat{i}$ goes, start by looking at the first column of the matrix on the right, since this is where $\hat{i}$ initially lands. Multiplying that column by the matrix on the left is how you can tell where the intermediate version of $\hat{i}$ ends up after applying the second transformation.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e \\ g \end{bmatrix} = e \begin{bmatrix} a \\ c \end{bmatrix} + g \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ae + bg \\ ce + dg \end{bmatrix}$$

So the first column of the composition matrix will always equal the left matrix times the first column of the right matrix. Likewise, $\hat{j}$ will always initially land on the second column of the right matrix,

so multiplying by this second column will give its final location, and hence, that's the second column of the composition matrix.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} f \\ h \end{bmatrix} = f \begin{bmatrix} a \\ c \end{bmatrix} + h \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} af + bh \\ cf + dh \end{bmatrix}$$

So the complete composition matrix is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

Notice there's a lot of symbols here, and it's common to be taught this formula as something to memorize along with a certain algorithmic process to help remember it. Before memorizing that process, you should get in the habit of thinking about what matrix multiplication really represents: applying one transformation after another. This will give you a much better conceptual framework that makes the properties of matrix multiplication much easier to understand.

8.4.2 Matrix multiplication associativity

For example, here's a question: does it matter what order we put the two matrices in when we multiply them? Let's think through a simple example. Take a shear which fixes $\hat{i}$ and moves $\hat{j}$ over to the right, and a $90°$ rotation. If you first do the shear then rotate, we can see that $\hat{i}$ ends up at $(0, 1)$ and $\hat{j}$ ends up at $(-1, 1)$. Both are generally pointing close together. If you first rotate then do the shear, $\hat{i}$ ends up over at $(1, 1)$ and $\hat{j}$ is off on a different direction at $(-1, 0)$ and they're pointing farther apart. The overall effect here is clearly different, so evidently, order totally does matter. Notice by thinking in terms of transformations, that's the kind of thing that you can do in your head by visualizing. No matrix multiplication necessary.

Let's consider trying to prove that matrix multiplication is associative. This means that if you have three matrices A, B, and C, and you multiply them all together, it shouldn't matter if you first compute A times B then multiply the result by C, or if you first multiply B times C then multiply that result by A on the left. In other words, it doesn't matter where you put the parenthesis.

$$(AB)C \stackrel{?}{=} A(BC)$$

If you try to work through this numerically, it's horrible, and unenligthening for that matter. However, when you think about matrix multiplication as applying one transformation after another, this property is just trivial. Can you see why? What it's saying is that if you first apply C then B, then A, it's the same as applying C, then B then A. There's nothing to prove, you're just applying the same three things one after the other all in the same order. This might feel like cheating, but it's not. This is a valid proof that matrix multiplication is associative, and even better than that, it's a good explanation for why that property should be true.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (10 minutes) [8].

8.5 The determinant

So, moving forward, we will be assuming you have a visual understanding of linear transformations and how they're represented with matrices.

8.5.1 Scaling areas

If you think about a couple linear transformations, you might notice how some of them seem to stretch space out while others compress it. It's useful for understanding these transformations to measure exactly how much it stretches or compresses things (more specifically, to measure the factor by which areas are scaled). For example, look at the matrix with columns $(3, 0)$, and $(0, 2)$.

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

It scales $\hat{i}$ by a factor of 3 and scales $\hat{j}$ by a factor of 2. Now, if we focus our attention on the one-by-one square whose bottom sits on $\hat{i}$ and whose left side sits on $\hat{j}$, after the transformation, this turns into a 2 by 3 rectangle. Since this region started out with area 1 and ended up with area 6, we can say the linear transformation has scaled its area by a factor of 6. Compare that to a shear whose matrix has columns $(1, 0)$ and $(1, 1)$ meaning $\hat{i}$ stays in place and $\hat{j}$ moves over to $(1, 1)$.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

That same unit square determined by $\hat{i}$ and $\hat{j}$ gets slanted and turned into a parallelogram, but the area of that parallelogram is still 1 since its base and height each continue to each have length 1. Even though this transformation pushes things about, it seems to leave areas unchanged (at least in the case of that one uint square).

Actually though, if you know how much the area of that one single unit square changes, you can know how the area of any possible region in space changes. First off, notice that whatever happens to one square in the grid has to happen to any other square in the grid no matter the size. This follows from the fact that grid lines remain parallel and evenly spaced. Then, any shape that's not a grid square can be approximated by grid squares pretty well with arbitrarily good approximations if you use small enough grid squares. So, since the areas of all those tiny grid squares are being scaled by some single amount, the area of the shape as a whole will also be scaled by that same single amount.

8.5.2 Exploring the determinant

This special scaling factor, the factor by which a linear transformation changes any area, is called the *determinant* of that transformation. We'll show how to compute the determinant of a transformation using its matrix later on, but understanding what it represents is much more important than the computation. For example, the determinant of a transformation would be 3 if that transformation increases the area of the region by a factor of 3.

$$det\left(\begin{bmatrix} 0 & 2 \\ -1.5 & 1 \end{bmatrix}\right) = 3$$

The determinant of a matrix is commonly denoted by vertical bars instead of square brackets.

$$\begin{vmatrix} 0 & 2 \\ -1.5 & 1 \end{vmatrix} = 3$$

The determinant of a transformation would be $\frac{1}{2}$ if it compresses all areas by a factor of $\frac{1}{2}$.

$$\begin{vmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{vmatrix} = 0.5$$

The determinant of a 2D transformation is zero if it compresses all of space onto a line or even onto a single point since then, the area of any region would become zero.

$$\begin{vmatrix} 4 & 2 \\ 2 & 1 \end{vmatrix} = 0$$

That last example proved to be pretty important. It means checking if the determinant of a given matrix is zero will give a way of computing whether the transformation associated with that matrix compresses everything into a smaller dimension.

This analogy so far isn't quite right. The full concept of a determinant allows for negative values.

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = -2$$

What would scaling an area by a negative amount even mean? This has to do with the idea of orientation. A 2D transformation with a negative determinant essentially flips space over. Any transformations that do this are said to "invert the orientation of space". Another way to think about it is in terms of $\hat{i}$ and $\hat{j}$. In their starting positions, $\hat{j}$ is to the left of $\hat{i}$. If, after a transformation, $\hat{j}$ is now on the right side of $\hat{i}$, the orientation of space has been inverted. Whenever this happens, the determinant will be negative. The absolute value of the determinant still tells you the factor by which areas have been scaled.

For example, the matrix with columns $(1, 1)$ and $(2, -1)$ encodes a transformation that has determinant -3.

$$\begin{vmatrix} 1 & 2 \\ 1 & -1 \end{vmatrix} = -3$$

This means that space gets flipped over and areas are scaled by a factor of 3.

Why would this idea of a negative area scaling factor be a natural way to describe orientation-flipping? Think about the series of transformations you get by slowly letting $\hat{i}$ rotate closer and closer to $\hat{j}$. As $\hat{i}$ gets closer, all the areas in space are getting compressed more and more meaning the determinant approaches zero. Once $\hat{i}$ lines up perfectly with $\hat{j}$, the determinant is zero. Then, if $\hat{i}$ continues, doesn't it feel natural for the determinant to keep decreasing into negative numbers?

8.5.3 The determinant in 3D

That's the understanding of determinants in two dimensions. What should it mean for three dimensions? The determinant of a 3×3 matrix tells you how much volumes get scaled. A determinant of zero would mean that all of space is compressed onto something with zero volume meaning either a flat plane, a line, or in the most extreme case, a single point. This means that the columns of the matrix are linearly dependent.

What should negative determinants mean for three dimensions? One way to describe orientation in 3D is with the right-hand rule. Point the forefinger of your right hand in the direction if $\hat{i}$, stick out your middle finger in the direction of $\hat{j}$, and notice how when you point your thumb up, it is in the

direction of $\hat{k}$. If you can still do that after the transformation, orientation has not changed and the determinant is positive. Otherwise, if after the transformation it only makes sense to do that with your left hand, orientation has been flipped and the determinant is negative.

8.5.4 Computing the determinant

How do you actually compute the determinant? For a 2×2 matrix with entries a, b, c, d, the formula is as follows.

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Here's part of an intution for where this formula comes from. Let's say that the terms b and c were both zero. Then, the term a tells you how much $\hat{i}$ is stretched in the x-direction and the term d tells you how much $\hat{j}$ is stretched in the y-direction. Since those other terms are zero, it should make sense that ad gives the area of the rectangle that the unit square turns into. Even if only one of b or c are zero, you'll have a parallelogram with a base of a and a height d, so the area should still be ad. Loosely speaking, if both b and c are nonzero, then that bc term tells you how much this parallelogram is stretched or compressed in the diagonal direction.

If you feel like computing determinants by hand is something that you need to know (you won't for this book), the only way to get it down is to just practice it with a few. This is all triply true for 3D determinants. There is a formula, and if you feel like that's something you need to know, you should practice with a few matrices.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

We don't think those computations fall within the essence of linear algebra, but understanding what the determinant represents falls within that essence.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (10 minutes) [10].

8.6 Inverse matrices, column space, and null space

As you can probably tell by now, the bulk of this chapter is on understanding matrix and vector operations through that more visual lens of linear transformations. This section is no exception, describing the concepts of inverse matrices, columns space, rank, and null space through that lens. A fair warning though: we're not going to talk about the methods for actually computing these things, and some would argue that that's pretty important. There are a lot of very good resources for learning those methods outside of this chapter. Keywords: "Gaussian elimination" and "row echelon form". Most of the value that we actually have to add here is on the intuition half. Plus, in practice, we usually use software to compute these things for us anyway.

8.6.1 Linear systems of equations

First, a few words on the usefulness of linear algebra. By now, you already have a hint for how it's used in describing the manipulation of space, which is useful for computer graphics and robotics. However, one of the main reasons that linear algebra is more broadly applicable, and required for

just about any technical discipline, is that it lets us solve certain systems of equations. When we say "system of equations", we mean there is a list of variables, things you don't know, and a list of equations relating them. For example,

$$\begin{aligned} 6x - 3y + 2z &= 7 \\ x + 2y + 5z &= 0 \\ 2x - 8y - z &= -2 \end{aligned}$$

is a system of equations with the unknowns x, y, and z.

In a lot of situations, those equations can get very complicated, but, if you're lucky, they might take on a certain special form. Within each equation, the only thing happening to each variable is that it's scaled by some constant, and the only thing happening to each of those scaled variables is that they're added to each other, so no exponents or fancy functions, or multiplying two variables together.

The typical way to organize this sort of special system of equations is to throw all the variables on the left and put any lingering constants on the right. It's also nice to vertically line up the common variables, and to do that, you might need to throw in some zero coefficients whenever the variable doesn't show up in one of the equations.

$$\begin{aligned} 2x + 5y + 3z &= -3 \\ 4x + 0y + 8z &= 0 \\ 1x + 3y + 0z &= 2 \end{aligned}$$

This is called a "linear system of equations". You might notice that this looks a lot like matrix-vector multiplication. In fact, you can package all of the equations together into a single vector equation, where you have the matrix containing all the constant coefficients, a vector containing all the constant coefficients, and a vector containing all the variables. Their matrix-vector product equals some different constant vector.

$$\begin{bmatrix} 2 & 5 & 3 \\ 4 & 0 & 8 \\ 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 2 \end{bmatrix}$$

Let's name that constant matrix $\mathbf{A}$, denote the vector holding the variables with $\mathbf{x}$, and call the constant vector on the right-hand side $\mathbf{v}$. This is more than just a notational trick to get our system of equations written on one line. It sheds light on a pretty cool geometric interpretation for the problem.

$$\mathbf{A}\mathbf{x} = \mathbf{v}$$

The matrix $\mathbf{A}$ corresponds with some linear transformation, so solving $\mathbf{A}\mathbf{x} = \mathbf{v}$ means we're looking for a vector $\mathbf{x}$ which, after applying the transformation $\mathbf{A}$, lands on $\mathbf{v}$.

Think about what's happening here for a moment. You can hold in your head this really complicated idea of multiple variables all intermingling with each other just by thinking about compressing or morphing space and trying to determine which vector lands on another.

To start simple, let's say you have a system with two equations and two unknowns. This means the matrix $\mathbf{A}$ is a 2×2 matrix, and $\mathbf{v}$ and $\mathbf{x}$ are each two-dimensional vectors.

$$\begin{aligned} 2x + 2y &= -4 \\ 1x + 3y &= -1 \\ \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} -4 \\ -1 \end{bmatrix} \end{aligned}$$

8.6.2 Inverse

How we think about the solutions to this equation depends on whether the transformation associated with $\mathbf{A}$ compresses all of space into a lower dimension, like a line or a point, or if it leaves everything spanning the full two dimensions where it started. In the language of the last section, we subdivide into the case where $\mathbf{A}$ has zero determinant and the case where $\mathbf{A}$ has nonzero determinant.

Let's start with the most likely case where the determinant is nonzero, meaning space does not get compressed into a zero area region. In this case, there will always be one and only one vector that lands on $\mathbf{v}$, and you can find it by playing the transformation in reverse. Following where $\mathbf{v}$ goes as we undo the transformation, you'll find the vector $\mathbf{x}$ such that $\mathbf{A}$ times $\mathbf{x}$ equals $\mathbf{v}$.

When you play the transformation in reverse, it actually corresponds to a separate linear transformation, commonly called the "inverse of $\mathbf{A}$" denoted $\mathbf{A}^{-1}$.

$$\mathbf{A}^{-1} = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}^{-1}$$

For example, if $\mathbf{A}$ was a counterclockwise rotation by $90°$, then the inverse of $\mathbf{A}$ would be a clockwise rotation by $90°$.

$$\mathbf{A} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \mathbf{A}^{-1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

If $\mathbf{A}$ was a rightward shear that pushes $\hat{j}$ one unit to the right, the inverse of $\mathbf{A}$ would be a leftward shear that pushes $\hat{j}$ one unit to the left.

In general, $\mathbf{A}^{-1}$ is the unique transformation with the property that if you first apply $\mathbf{A}$, then follow it with the transformation $\mathbf{A}^{-1}$, you end up back where you started. Applying one transformation after another is captured algebraically with matrix multiplication, so the core property of this transformation $\mathbf{A}^{-1}$ is that $\mathbf{A}^{-1}\mathbf{A}$ equals the matrix that corresponds to doing nothing.

The transformation that does nothing is called the "identity transformation". It leaves $\hat{i}$ and $\hat{j}$ each where they are, unmoved, so its columns are $(1, 0)$ and $(0, 1)$.

$$\mathbf{A}^{-1}\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Once you find this inverse, which in practice, you do with a computer, you can solve your equation by multipling this inverse matrix by $\mathbf{v}$.

$$\begin{aligned}\mathbf{A}\mathbf{x} &= \mathbf{v}\\ \mathbf{A}^{-1}\mathbf{A}\mathbf{x} &= \mathbf{A}^{-1}\mathbf{v}\\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{v}\end{aligned}$$

Again, what this means geometrically is that you're playing the transformation in reverse and following $\mathbf{v}$. This nonzero determinant case, which for a random choice of matrix is by far the most likely one, corresponds with the idea that if you have two unknowns and two equations, it's almost certainly the case that there's a single, unique solution.

This idea also makes sense in higher dimensions when the number of equations equals the number of unknowns. Again, the system of equations can be translated to the geometric interpretation where you have some transformation, $\mathbf{A}$, some vector $\mathbf{v}$, and you're looking for the vector $\mathbf{x}$ that lands on $\mathbf{v}$. As long as the transformation $\mathbf{A}$ doesn't compress all of space into a lower dimension, meaning, its determinant is nonzero, there will be an inverse transformation, $\mathbf{A}^{-1}$, with the property that if you first do $\mathbf{A}$, then you do $\mathbf{A}^{-1}$, it's the same as doing nothing. To solve your equation, you just have to multiply that reverse transformation matrix by the vector $\mathbf{v}$.

When the determinant is zero and the transformation associated with this system of equations compresses space into a smaller dimension, there is no inverse. You cannot uncompress a line to turn it into a plane. At least, that's not something that a function can do. That would require transforming each individual vector into a whole line full of vectors, but functions can only take a single input to a single output.

Similarly, for three equations and three unknowns, there will be no inverse if the corresponding transformation compresses 3D space onto the plane, or even if it compresses it onto a line, or a point. Those all correspond to a determinant of zero since any region is compressed into something with zero volume.

It's still possible that a solution exists even when there is no inverse. It's just that when your transformation compresses space onto, say, a line, you have to be lucky enough that the vector $\mathbf{v}$ exists somewhere on that line.

8.6.3 Rank and column space

You might notice that some of these zero determinant cases feel a lot more restrictive than others. Given a 3×3 matrix, for example, it seems a lot harder for a solution to exist when it compresses space onto a line compared to when it compresses space onto a plane even though both of those have zero determinant. We have some language that's more specific than just saying "zero determinant". When the output of a transformation is a line, meaning it's one-dimensional, we say the transformation has a *rank* of one. If all the vectors land on some two-dimensional plane, we say the transformation has a rank of two. The word "rank" means the number of dimensions in the output of a transformation.

For instance, in the case of 2×2 matrices, the highest possible rank is 2. It means the basis vectors continue to span the full two dimensions of space, and the determinant is nonzero. For 3×3 matrices, rank 2 means that we've collapsed, but not as much as we would have collapsed for a rank 1 situation. If a 3D transformation has a nonzero determinant and its output fills all of 3D space, it has a rank of 3.

This set of all possible outputs for your matrix, whether it's a line, a plane, 3D space, whatever, is called the *column space* of your matrix. You can probably guess where that name comes from. The columns of your matrix tell you where the basis vectors land, and the span of those transformed basis

vectors gives you all possible outputs. In other words, the column space is the span of the columns of your matrix, so a more precise definition of rank would be that it's the number of dimensions in the column space. When this rank is as high as it can be, meaning it equals the number of columns, we call the matrix "full rank".

8.6.4 Null space

Notice, the zero vector will always be included in the column space since linear transformations must keep the origin fixed in place. For a full rank transformation, the only vector that lands at the origin is the zero vector itself, but for matrices that aren't full rank, which compress to a smaller dimension, you can have a whole bunch of vectors that land on zero. If a 2D transformation compresses space onto a line, for example, there is a separate line in a different direction full of vectors that get compressed onto the origin. If a 3D transformation compresses space onto a plane, there's also a full line of vectors that land on the origin. If a 3D transformation compresses all the space onto a line, then there's a whole plane full of vectors that land on the origin.

This set of vectors that lands on the origin is called the *null space* or the *kernel* of your matrix. It's the space of all vectors that become null in the sense that they land on the zero vector. In terms of the linear system of equations $\mathbf{Ax} = \mathbf{v}$, when $\mathbf{v}$ happens to be the zero vector, the null space gives you all the possible solutions to the equation.

8.6.5 Closing remarks

That's a high-level overview of how to think about linear systems of equations geometrically. Each system has some kind of linear transformation associated with it, and when that transformation has an inverse, you can use that inverse to solve your system. Otherwise, the idea of column space lets us understand when a solution even exists, and the idea of a null space helps us understand what the set of all possible solutions can look like.

Again, there's a lot not covered here, most notably how to compute these things. We also had to limit the scope to examples where the number of equations equals the number of unknowns. The goal here is not to try to teach everything: it's that you come away with a strong intuition for inverse matrices, column space, and null space, and that those intuitions make any future learning that you do more fruitful.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (12 minutes) [5].

8.7 Nonsquare matrices as transformations between dimensions

When we've talked about linear transformations so far, we've only really talked about transformations from 2D vectors to other 2D vectors, represented with 2×2 matrices; or from 3D vectors to other 3D vectors, represented with 3×3 matrices. What about nonsquare matrices? We'll take a moment to discuss what those mean geometrically.

By now, you have most of the background you need to start pondering a question like this on your own, but we'll start talking through it, just to give a little mental momentum.

It's perfectly reasonable to talk about transformations between dimensions, such as one that takes 2D vectors to 3D vectors. Again, what makes one of these linear is that grid lines remain parallel and evenly spaced, and that the origin maps to the origin.

Encoding one of these transformations with a matrix the same as what we've done before. You look at where each basis vector lands and write the coordinates of the landing spots as the coordinates of the landing spots as the columns of a matrix. For example, the following is a transformation that takes $\hat{i}$ to the coordinates $(2, -1, -2)$ and $\hat{j}$ to the coordinates $(0, 1, 1)$.

$$\begin{bmatrix} 2 & 0 \\ -1 & 1 \\ -2 & 1 \end{bmatrix}$$

Notice, this means the matrix encoding our transformation has three rows and two columns, which, to use standard terminology, makes it a 3×2 matrix. In the language of last section, the column space of this matrix, the place where all the vectors land, is a 2D plane slicing through the origin of 3D space. The matrix is still full rank since the number of dimensions in this column space is the same as the number of dimensions of the input space.

If you see a 3×2 matrix out in the wild, you can know that it has the geometric interpretation of mapping two dimensions to three dimensions since the two columns indicate that the input space has two basis vectors, and the three rows indicate that the landing spots for each of those basis vectors is described with three separate coordinates.

For a 2×3 matrix, the three columns indicate a starting space that has three basis vectors, so it starts in three dimensions; and the two rows indicate that the landing spot for each of those three basis vectors is described with only two coordinates, so they must be landing in two dimensions. It's a transformation from 3D space onto the 2D plane.

You could also have a transformation from two dimensions to one dimension. One-dimensional space is really just the number line, so a transformation like this takes in 2D vectors and returns numbers. Thinking about gridlines remaining parallel and evenly spaced is messy due to all the compression happening here, so in this case, the visual understanding for what linearity means is that if you have a line of evenly spaced dots, it would remain evenly spaced once they're mapped onto the number line.

One of these transformations is encoded with a 1×2 matrix, each of whose two columns has just a single entry. The two columns represent where the basis vectors land, and each one of those columns requires just one number, the number that that basis vector landed on.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (4 minutes) [9].

8.8 Eigenvectors and eigenvalues

8.8.1 What is an eigenvector?

To start, consider some linear transformation in two dimensions that moves the basis vector $\hat{i}$ to the coordinates $(3, 0)$ and $\hat{j}$ to $(1, 2)$, so it's represented with a matrix whose columns are $(3, 0)$ and $(1, 2)$.

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

Focus in on what it does to one particular vector and think about the span of that vector, the line passing through its origin and its tip. Most vectors are going to get knocked off their span during

the transformation, but some special vectors do remain on their own span meaning the effect that the matrix has on such a vector is just to stretch it or compress it like a scalar.

For this specific example, the basis vector $\hat{i}$ is one such special vector. The span of $\hat{i}$ is the x-axis, and from the first column of the matrix, we can see that $\hat{i}$ moves over to three times itself still on that x-axis. What's more, due to the way linear transformations work, any other vector on the x-axis is also just stretched by a factor of 3, and hence, remains on its own span.

A slightly sneakier vector that remains on its own span during this transformation is $(-1, 1)$. It ends up getting stretched by a factor of 2. Again, linearity is going to imply that any other vector on the diagonal line spanned by this vector is just going to get stretched out by a factor of 2.

For this transformation, those are all the vectors with this special property of staying on their span. Those on the x-axis get stretched out by a factor of 3 and those on the diagonal line get stretched out by a factor of 2. Any other vector is going to get rotated somewhat during the transformation and knocked off the line that it spans. As you might have guessed by now, these special vectors are called the *eigenvectors* of the transformation, and each eigenvector has associated with it an *eigenvalue*, which is just the factor by which it's stretched or compressed during the transformation.

Of course, there's nothing special about stretching vs compressing or the fact that these eigenvalues happen to be positive. In another example, you could have an eigenvector with eigenvalue $-\frac{1}{2}$, meaning that the vector gets flipped and compressed by a factor of $\frac{1}{2}$.

$$\begin{bmatrix} 0.5 & -1 \\ -1 & 0.5 \end{bmatrix}$$

The important part here is that it stays on the line that it spans out without getting rotated off of it.

8.8.2 Eigenvectors in 3D rotation

For a glimpse of why this might be a useful thing to think about, consider some three-dimensional rotation. If you can find an eigenvector for that rotation, a vector that remains on its own span, you have found the axis of rotation. It's much easier to think about a 3D rotation in terms of some axis of rotation and an angle by which it's rotating rather than thinking about the full 3×3 matrix associated with that transformation. In this case, by the way, the corresponding eigenvalue would have to be 1 since rotations never stretch or compress anything, so the length of the vector would remain the same.

8.8.3 Finding eigenvalues

The following pattern shows up a lot in linear algebra. With any linear transformation described by a matrix, you could understand what it's doing by reading off the columns of this matrix as the landing spots for basis vectors, but often a better way to get at the heart of what the linear transformation actually does, less dependent on your particular coordinate system, is to find the eigenvectors and eigenvalues.

I won't cover the full details on methods for computing eigenvectors and eigenvalues here, but I'll try to give an overview of the computational ideas that are most important for a conceptual understanding. Symbolically, an eigenvector look like the following

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

$\mathbf{A}$ is the matrix representing some transformation, $\mathbf{v}$ is the eigenvector, and λ is a number, namely the corresponding eigenvalue. This expression is saying that the matrix-vector product $\mathbf{Av}$ gives the same result as just scaling the eigenvector $\mathbf{v}$ by some value λ. Finding the eigenvectors and their eigenvalues of the matrix $\mathbf{A}$ involves finding the values of $\mathbf{v}$ and λ that make this expression true. It's awkward to work with at first because that left-hand side represents matrix-vector multiplication, but the right-hand side is scalar-vector multiplication. Let's rewrite the right-hand side as some kind of matrix-vector multiplication using a matrix which has the effect of scaling any vector by a factor of λ. The columns of such a matrix will represent what happens to each basis vector, and each basis vector is simply multiplied by λ, so this matrix will have the number λ down the diagonal and zeroes everywhere else.

$$\begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}$$

The common way to write this is to factor out λ and write it as $\lambda\mathbf{I}$ where $\mathbf{I}$ is the identity matrix with ones down the diagonal.

$$\mathbf{Av} = (\lambda\mathbf{I})\mathbf{v}$$

With both sides looking like matrix-vector multiplication, we can subtract off that right-hand side and factor out $\mathbf{v}$.

$$\begin{aligned} \mathbf{Av} - (\lambda\mathbf{I})\mathbf{v} &= \mathbf{0} \\ (\mathbf{A} - \lambda\mathbf{I})\mathbf{v} &= \mathbf{0} \end{aligned}$$

We now have a new matrix $\mathbf{A} - \lambda\mathbf{I}$, and we're looking for a vector $\mathbf{v}$ such that this new matrix times $\mathbf{v}$ gives the zero vector. This will always be true if $\mathbf{v}$ itself is the zero vector, but that's boring. We want a nonzero eigenvector. The only way it's possible for the product of a matrix with a nonzero vector to become zero is if the transformation associated with that matrix compresses space into a lower dimension. That compression corresponds to a zero determinant for the matrix.

$$det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

To be concrete, let's say your matrix $\mathbf{A}$ has columns $(2, 1)$ and $(2, 3)$, and think about subtracting off a variable amount λ.

$$det\left(\begin{bmatrix} 2-\lambda & 2 \\ 1 & 3-\lambda \end{bmatrix}\right) = 0$$

The goal is to find a value of λ that will make this determinant zero meaning the tweaked transformation compresses space into a lower dimension. In this case, that value is $\lambda = 1$. Of course, if we had chosen some other matrix, the eigenvalue might not necessarily be 1.

This is kind of a lot, but let's unravel what this is saying. When $\lambda = 1$, the matrix $\mathbf{A} - \lambda\mathbf{I}$ compresses space onto a line. That means there's a nonzero vector $\mathbf{v}$ such that $(\mathbf{A} - \lambda\mathbf{I})\mathbf{v}$ equals the zero vector.

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$$

Remember, we care about that because it means $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$, which you can read off as saying that the vector $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ staying on its own span during the transformation $\mathbf{A}$. For the following example

$$\begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix} \mathbf{v} = 1\mathbf{v}$$

the corresponding eigenvalue is 1, so $\mathbf{v}$ would actually just stay fixed in place.

To summarize our line of reasoning:

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \lambda\mathbf{v} \\ \mathbf{A}\mathbf{v} - \lambda\mathbf{I}\mathbf{v} &= \mathbf{0} \\ (\mathbf{A} - \lambda\mathbf{I})\mathbf{v} &= \mathbf{0} \\ det(\mathbf{A} - \lambda\mathbf{I}) &= \mathbf{0} \end{aligned}$$

To see this in action, let's visit the example from the start with a matrix whose columns are $(3, 0)$ and $(1, 2)$. To determine if a value λ is an eigenvalue, subtract it from the diagonals of this matrix and compute the determinant.

$$\begin{aligned} \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} & \\ \begin{bmatrix} 3-\lambda & 1 \\ 0 & 2-\lambda \end{bmatrix} & \\ det\left(\begin{bmatrix} 3-\lambda & 1 \\ 0 & 2-\lambda \end{bmatrix}\right) &= (3-\lambda)(2-\lambda) - 1 \cdot 0 \\ &= (3-\lambda)(2-\lambda) \end{aligned}$$

We get a certain quadratic polynomial in λ. Since λ can only be an eigenvalue if this determinant happens to be zero, you can conclude that the only possible eigenvalues are $\lambda = 2$ and $\lambda = 3$.

To determine what the eigenvectors are that actually have one of these eigenvalues, say $\lambda = 2$, plug in that value of λ to the matrix and then solve for which vectors this diagonally altered matrix sends to zero.

$$\begin{bmatrix} 3-2 & 1 \\ 0 & 2-2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If you computed this the way you would any other linear system, you'd see that the solutions are all the vectors on the diagonal line spanned by $(-1, 1)$. This corresponds to the fact that the unaltered matrix has the effect of stretching all those vectors by a factor of 2.

8.8.4 Transformations with no eigenvectors

A 2D transformation doesn't have to have eigenvectors. For example, consider a rotation by 90°.

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

This doesn't have any eigenvectors since it rotates every vector off its own span. If you actually tried computing the eigenvalues of a rotation like this, notice what happens.

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -\lambda & -1 \\ 1 & -\lambda \end{bmatrix}$$

$$\begin{aligned} det\left(\begin{bmatrix} -\lambda & -1 \\ 1 & -\lambda \end{bmatrix}\right) &= (-\lambda)(-\lambda) - (-1)(1) \\ &= \lambda^2 + 1 = 0 \end{aligned}$$

The only roots of that polynomial are the imaginary numbers i and $-i$. The fact that there are no real number solutions indicates that there are no eigenvectors.

R Interestingly though, the fact that multiplication by i in the complex plane looks like a 90° rotation is related to the fact that i is an eigenvalue of this transformation of 2D real vectors. The specifics of this are out of scope, but note that eigenvalues which are complex numbers generally correspond to some kind of rotation in the transformation.

8.8.5 Repeated eigenvalues

Another interesting example is a shear which fixes $\hat{i}$ in place and moves $\hat{j}$ over by 1.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

All the vectors on the x-axis are eigenvectors with eigenvalue 1. In fact, these are the only eigenvectors.

$$\begin{aligned} det\left(\begin{bmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix}\right) &= (1-\lambda)(1-\lambda) = 0 \\ (1-\lambda)^2 &= 0 \end{aligned}$$

The only root of this expression is $\lambda = 1$.

8.8.6 Transformations with larger eigenvector spans

Keep in mind it's also possible to have just one eigenvalue but with more than just a line full of eigenvectors. A simple example is a matrix that scales everything by 2.

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The only eigenvalue is 2, but every vector in the plane gets to be an eigenvector with that eigenvalue.

R See the corresponding *Essence of linear algebra* video for a more visual presentation (17 minutes) [2].

8.9 Miscellaneous notation

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.

The matrix $\mathbf{I}$ is known as the identity matrix, which is a square matrix with ones along its diagonal and zeroes elsewhere. For example

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix denoted by $\mathbf{0}_{m \times n}$ is a matrix filled with zeroes with m rows and n columns.

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

The † in $\mathbf{B}^\dagger$ denotes the Moore-Penrose pseudoinverse given by $\mathbf{B}^\dagger = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$. The pseudoinverse is used when the matrix is nonsquare and thus not invertible to produce a close approximation of an inverse in the least squares sense.

Night sky above Dufour Street in Santa Cruz, CA

9. State-space controllers

R Chapters from here on use the `frccontrol` Python package to demonstrate the concepts discussed and perform the complex math required. See appendix B for how to install it.

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.

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

9.2 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 brushed 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 brushed 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 brushed 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.

9.3 State-space notation

9.3.1 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 addition to the state, inputs and outputs are represented as vectors. Since the mapping from the current states and inputs to the change in state is a system of equations, it's natural to write it in matrix form.

9.3.2 Benefits over classical control

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

$q \times p$ Laplace transforms 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.

Students are still taught classical control first because it provides a framework within which to understand the results we get from the fancy mathematical machinery of modern control.

9.3.3 The equations

Below are the continuous and discrete versions of state-space notation.

Definition 9.3.1 — State-space notation.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{9.1}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{9.2}$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{9.3}$$
$$\mathbf{y}_{k+1} = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \tag{9.4}$$

$\mathbf{A}$	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 9.1: State-space matrix dimensions

In the continuous case, 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 $\mathbf{y}$.

9.4 Controllability

State controllability implies that it is possible – by admissible inputs – to steer the states from any initial value to any final value within some finite time window.

[1] This book focuses on analysis and control of linear systems. See chapter 12 for more on nonlinear control.

Theorem 9.4.1 — Controllability. A continuous time-invariant linear state-space model is controllable if and only if

$$\text{rank}\begin{pmatrix}\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}\end{pmatrix} = n \tag{9.5}$$

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

The matrix in equation (9.5) 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 $\mathbb{C}$ is defined as $\frac{\sigma_{max}(\mathbb{C})}{\sigma_{min}(\mathbb{C})}$ where σ_{max} is the maximum singular value[2] 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.

9.5 Observability

Observability is a measure for how well internal states of a system can be inferred by knowledge of its external outputs. The observability and controllability of a system are mathematical duals (i.e., as controllability proves that an input is available that brings any initial state to any desired final state, observability proves that knowing enough output values provides enough information to predict the initial state of the system).

Theorem 9.5.1 — Observability. A continuous time-invariant linear state-space model is observable if and only if

$$\text{rank}\begin{pmatrix}\begin{bmatrix}C \\ CA \\ \vdots \\ CA^{n-1}\end{bmatrix}\end{pmatrix} = n \tag{9.6}$$

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

The matrix in equation (9.6) being rank-deficient means the outputs do not contain contributions from every state. That is, not all states are mapped to a linear commbination in the output. Therefore, the outputs alone are insufficient to estimate all the states.

The condition number of the observability matrix $\mathbb{O}$ is defined as $\frac{\sigma_{max}(\mathbb{O})}{\sigma_{min}(\mathbb{O})}$ where σ_{max} is the maximum singular value[2] 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.

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

9.6 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 9.8.

First is the state update equation. Substitute the control law into equation (9.1).

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{K}(\mathbf{r} - \mathbf{x}) \\ \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r} - \mathbf{B}\mathbf{K}\mathbf{x} \\ \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r}\end{aligned} \tag{9.7}$$

Now for the output equation. Substitute the control law into equation (9.2).

$$\begin{aligned}\mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}(\mathbf{K}(\mathbf{r} - \mathbf{x})) \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{K}\mathbf{r} - \mathbf{D}\mathbf{K}\mathbf{x} \\ \mathbf{y} &= (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x} + \mathbf{D}\mathbf{K}\mathbf{r}\end{aligned} \tag{9.8}$$

Now, we'll do the same for the discrete system. We'd like to know whether the system defined by equation (9.3) operating with the control law $\mathbf{u}_k = \mathbf{K}(\mathbf{r}_k - \mathbf{x}_k)$ converges to the reference $\mathbf{r}_k$.

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

Theorem 9.6.1 — Closed-loop state-space controller.

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{r} \tag{9.9}$$
$$\mathbf{y} = (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x} + \mathbf{D}\mathbf{K}\mathbf{r} \tag{9.10}$$

$$\mathbf{x}_{k+1} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{r}_k \tag{9.11}$$
$$\mathbf{y}_k = (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x}_k + \mathbf{D}\mathbf{K}\mathbf{r}_k \tag{9.12}$$

$\mathbf{A}$	system matrix	$\mathbf{K}$	controller gain matrix
$\mathbf{B}$	input matrix	$\mathbf{x}$	state vector
$\mathbf{C}$	output matrix	$\mathbf{r}$	reference vector
$\mathbf{D}$	feedthrough matrix	$\mathbf{y}$	output vector

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 to make the system

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	$\mathbf{r}$	states × 1
$\mathbf{K}$	inputs × states		

Table 9.2: Controller matrix dimensions

converge to the reference. For equation (9.9) to reach steady-state, the eigenvalues of $\mathbf{A} - \mathbf{BK}$ must be in the left-half plane. For equation (9.11) to have a bounded output, the eigenvalues of $\mathbf{A} - \mathbf{BK}$ must be within the unit circle.

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.

9.7 Pole placement

This is the practice of placing the poles of a closed-loop system directly to produce a desired response. This can be done manually for state feedback controllers with controllable canonical form (see section C.1). This can also be done manually for state observers with observable canonical form (see section C.2).

In general, 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, then use those as a starting point for pole placement.

9.8 LQR

Instead of placing the poles of a closed-loop system manually, LQR design places the poles for us based on acceptable error and control effort constraints. "LQR" stands for "Linear-Quadratic Regulator". 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 $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$.

$$J = \int_0^\infty \left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right) dt$$

where J represents a tradeoff between state excursion and control effort with the weighting factors $\mathbf{Q}$ and $\mathbf{R}$. LQR finds a control law $\mathbf{u}$ that minimizes the cost function. $\mathbf{Q}$ and $\mathbf{R}$ slide the cost along a Pareto boundary between state tracking and control effort (see figure 9.1). Pareto optimality for this problem means that an improvement in state tracking cannot be obtained without using more control effort to do so. Also, a reduction in control effort cannot be obtained without sacrificing state tracking performance. Pole placement, on the other hand, will have a cost anywhere on, above, or to the right of the Pareto boundary (no cost can be inside the boundary).

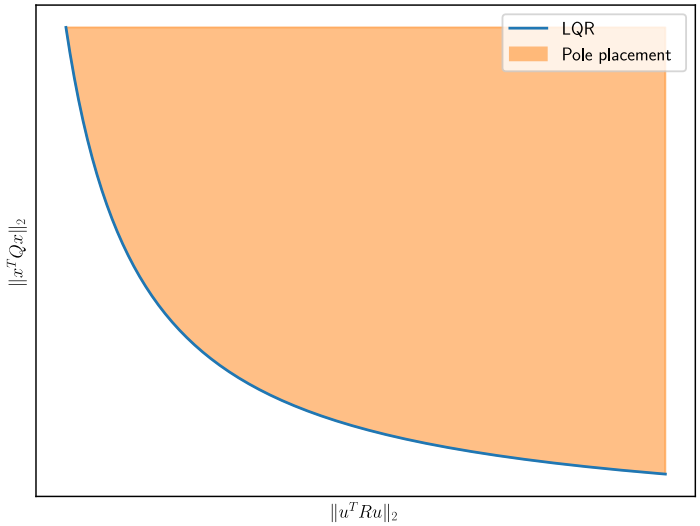


Figure 9.1: Pareto boundary for LQR

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

The feedback control law that minimizes J, which we'll call the "optimal control law", is shown in theorem 9.8.1.

Theorem 9.8.1 — Optimal control law.

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{9.13}$$

This means that optimal control can be achieved with simply a set of proportional gains on all the states. This control law will make all states converge to zero assuming the system is controllable. To converge to nonzero states, a reference vector $\mathbf{r}$ can be added to the state $\mathbf{x}$.

Theorem 9.8.2 — Optimal control law with nonzero reference.

$$\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x}) \tag{9.14}$$

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 D.2 for how $\mathbf{K}$ is calculated in Python. If the result is finite, the controller is guaranteed to be stable and robust with a phase margin of 60 degrees [20].

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.

9.8.1 Bryson's rule

The next obvious question is what values to choose for $\mathbf{Q}$ and $\mathbf{R}$. 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 $\mathbf{Q}$ and $\mathbf{R}$ can be slid along the Pareto boundary using a weighting factor ρ.

$$J = \int_0^\infty \left(\rho \left[\left(\frac{x_1}{x_{1,max}} \right)^2 + \ldots + \left(\frac{x_n}{x_{n,max}} \right)^2 \right] + \left[\left(\frac{u_1}{u_{1,max}} \right)^2 + \ldots + \left(\frac{u_n}{u_{n,max}} \right)^2 \right] \right) dt$$

$$\mathbf{Q} = \begin{bmatrix} \frac{\rho}{x_{1,max}^2} & 0 & \ldots & 0 \\ 0 & \frac{\rho}{x_{2,max}^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \frac{\rho}{x_{n,max}^2} \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \frac{1}{u_{1,max}^2} & 0 & \ldots & 0 \\ 0 & \frac{1}{u_{2,max}^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \frac{1}{u_{n,max}^2} \end{bmatrix}$$

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.

9.9 Case studies of controller design methods

This example uses the following second-order model for a CIM motor (a DC brushed 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 9.2 shows the response using poles placed at $(0.1, 0)$ and $(0.9, 0)$ and LQR with the following cost matrices.

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{20^2} & 0 \\ 0 & \frac{1}{40^2} \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \frac{1}{12^2} \end{bmatrix}$$

LQR selected poles at $(0.593, 0)$ and $(0.955, 0)$. Notice with pole placement that as the current pole moves left, the control effort becomes more aggressive.

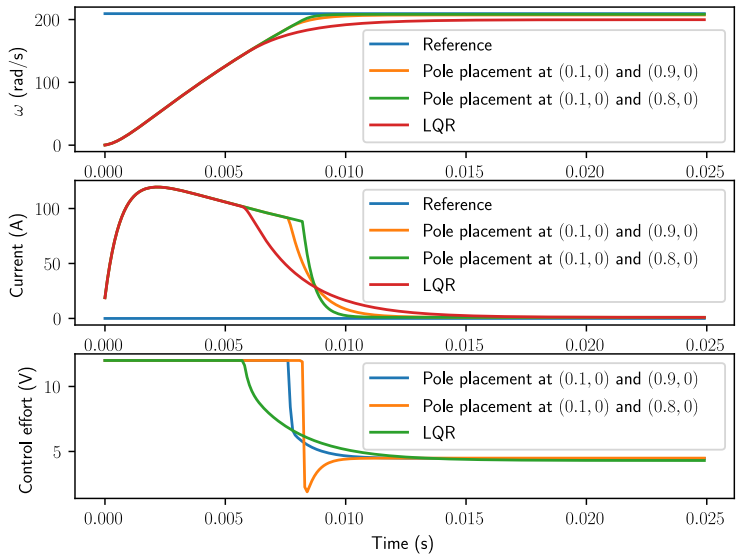


Figure 9.2: Second-order CIM motor response with pole placement and LQR

9.10 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.

9.10.1 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.

9.10.2 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.

9.10.3 Observer augmentation

Observer augmentation is closely related to plant augmentation. In addition to adding entries to the observer matrix $\mathbf{L}$, 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.

9.10.4 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 $\mathbf{y} = \mathbf{Cx} + \mathbf{Du}$, 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 $\mathbf{u} = -\mathbf{Kx}$, augmenting $\mathbf{C}$ with $-\mathbf{K}$ makes the observer estimate the control input $\mathbf{u}$ applied.

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

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.

9.10.5 Examples

Snippet 9.1 shows how one packs together the following augmented matrix in Python.

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

```
#!/usr/bin/env python3

import numpy as np


def main():
    J = 7.7500e-05
    b = 8.9100e-05
    Kt = 0.0184
    Ke = 0.0211
    R = 0.0916
    L = 5.9000e-05

    # fmt: off
    A = np.array([[-b / J, Kt / J],
                  [-Ke / L, -R / L]])
    B = np.array([[0],
                  [1 / L]])
    C = np.array([[1, 0]])
    D = np.array([[0]])
    # fmt: on

    print("A =")
    print(A)
    print("B =")
    print(B)
```

```
    print("C =")
    print(C)
    print("D =")
    print(D)

    tmp = np.concatenate(
        (
            np.concatenate((A - np.eye(A.shape[0]), B), axis=1),
            np.concatenate((C, D), axis=1),
        ),
        axis=0,
    )

    print("[A, B; C, D] =")
    print(tmp)


if __name__ == "__main__":
    main()
```

Snippet 9.1. Matrix augmentation example

Section 9.12 demonstrates model augmentation for different types of integral control.

9.11 Feedforwards

Feedforwards are used to inject information about either the system's dynamics (like a model does) or the intended movement into a controller. Feedforward is generally used to handle parts of the control actions we already know must be applied to make a system track a reference, then let the feedback controller correct for at runtime what we do not or cannot know about the system. We will present two ways of implementing feedforward for state feedback.

9.11.1 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

$$\mathbf{x}_{ss} = \mathbf{N}_x \mathbf{y}_{ss} \tag{9.15}$$

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

$$\mathbf{u}_{ss} = \mathbf{N}_u \mathbf{y}_{ss} \tag{9.16}$$

Continuous case

To find the control input required at steady-state, set equation (9.1) to zero.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= \mathbf{A}\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= \mathbf{A}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{B}\mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{D}\mathbf{N}_u\mathbf{y}_{ss}\end{aligned}$$

$$\begin{aligned}\begin{bmatrix}\mathbf{0} \\ \mathbf{y}_{ss}\end{bmatrix} &= \begin{bmatrix}\mathbf{A}\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix} \mathbf{y}_{ss} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A}\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix} \begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} &= \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{\dagger} \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix}\end{aligned}$$

where † is the Moore-Penrose pseudoinverse.

Discrete case

Now, we'll do the same thing for the discrete system. To find the control input required at steady-state, set equation (9.3) to zero.

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

$$\begin{aligned}\mathbf{x}_{ss} &= \mathbf{A}\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= (\mathbf{A} - \mathbf{I})\mathbf{x}_{ss} + \mathbf{B}\mathbf{u}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{x}_{ss} + \mathbf{D}\mathbf{u}_{ss}\end{aligned}$$

$$\begin{aligned}\mathbf{0} &= (\mathbf{A} - \mathbf{I})\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{B}\mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{y}_{ss} &= \mathbf{C}\mathbf{N}_x\mathbf{y}_{ss} + \mathbf{D}\mathbf{N}_u\mathbf{y}_{ss}\end{aligned}$$

$$\begin{aligned}\begin{bmatrix}\mathbf{0} \\ \mathbf{y}_{ss}\end{bmatrix} &= \begin{bmatrix}(\mathbf{A} - \mathbf{I})\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix}\mathbf{y}_{ss} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}(\mathbf{A} - \mathbf{I})\mathbf{N}_x + \mathbf{B}\mathbf{N}_u \\ \mathbf{C}\mathbf{N}_x + \mathbf{D}\mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} &= \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} \\ \begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} &= \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{\dagger}\begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix}\end{aligned}$$

where † is the Moore-Penrose pseudoinverse.

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 (9.15).

$$\begin{aligned}\mathbf{x}_{ss} &= \mathbf{N}_x\mathbf{y}_{ss} \\ \mathbf{N}_x^{\dagger}\mathbf{x}_{ss} &= \mathbf{y}_{ss}\end{aligned}$$

Now substitute this into equation (9.16).

$$\begin{aligned}\mathbf{u}_{ss} &= \mathbf{N}_u\mathbf{y}_{ss} \\ \mathbf{u}_{ss} &= \mathbf{N}_u(\mathbf{N}_x^{\dagger}\mathbf{x}_{ss}) \\ \mathbf{u}_{ss} &= \mathbf{N}_u\mathbf{N}_x^{\dagger}\mathbf{x}_{ss}\end{aligned}$$

$\mathbf{u}_{ss}$ and $\mathbf{x}_{ss}$ are also known as $\mathbf{u}_{ff}$ and $\mathbf{r}$ respectively.

$$\mathbf{u}_{ff} = \mathbf{N}_u\mathbf{N}_x^{\dagger}\mathbf{r}$$

So all together, we get theorem 9.11.1.

Theorem 9.11.1 — Steady-state feedforward.

Continuous:

$$\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^\dagger \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} \tag{9.17}$$

Discrete:

$$\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^\dagger \begin{bmatrix}\mathbf{0} \\ \mathbf{1}\end{bmatrix} \tag{9.18}$$

$$\mathbf{u}_{ff} = \mathbf{N}_u \mathbf{N}_x^\dagger \mathbf{r} \tag{9.19}$$

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 9.11.1 is square (number of inputs = number of outputs), the normal matrix inverse can be used instead.

9.11.2 Two-state feedforward

Let's start with the equation for the reference dynamics

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

where $\mathbf{u}_{ff}$ 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 $\mathbf{u}_{ff}$ such that we minimize the tracking error between $\mathbf{r}_{k+1}$ and $\mathbf{r}_k$.

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

To solve for $\mathbf{u}_{ff}$, 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 $\mathbf{0}$.

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

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 $inputs \times inputs$.

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

R $\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k$ will only return a nonzero vector if the reference isn't following the system dynamics. If it is, the feedback controller already compensates for it. This feedforward compensates for any unmodeled dynamics reflected in how the reference is changing (or not changing). In the case of a constant reference, the feedforward opposes any system dynamics that would change the state over time.

The following theorems will be needed to find the minimum of $\mathbf{J}$.

Theorem 9.11.2 $\frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is symmetric.

Theorem 9.11.3 $\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^T\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e})}{\partial \mathbf{x}} = \mathbf{A}^T\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e}) + \mathbf{D}^T\mathbf{C}^T(\mathbf{A}\mathbf{x}+\mathbf{b})$

Corollary 9.11.4 $\frac{\partial (\mathbf{A}\mathbf{x}+\mathbf{b})^T\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})}{\partial \mathbf{x}} = 2\mathbf{A}^T\mathbf{C}(\mathbf{A}\mathbf{x}+\mathbf{b})$ where $\mathbf{C}$ is symmetric.

Proof:

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

$\mathbf{C}$ is symmetric, so

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

Given theorem 9.11.2 and corollary 9.11.4, find the minimum of $\mathbf{J}$ by taking the partial derivative with respect to $\mathbf{u}_{ff}$ and setting the result to $\mathbf{0}$.

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

Theorem 9.11.5 — Two-state feedforward.

$$\mathbf{u}_{ff} = \mathbf{K}_{ff}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \tag{9.20}$$
$$\text{where } \mathbf{K}_{ff} = (\mathbf{B}^T\mathbf{Q}\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^T\mathbf{Q} \tag{9.21}$$

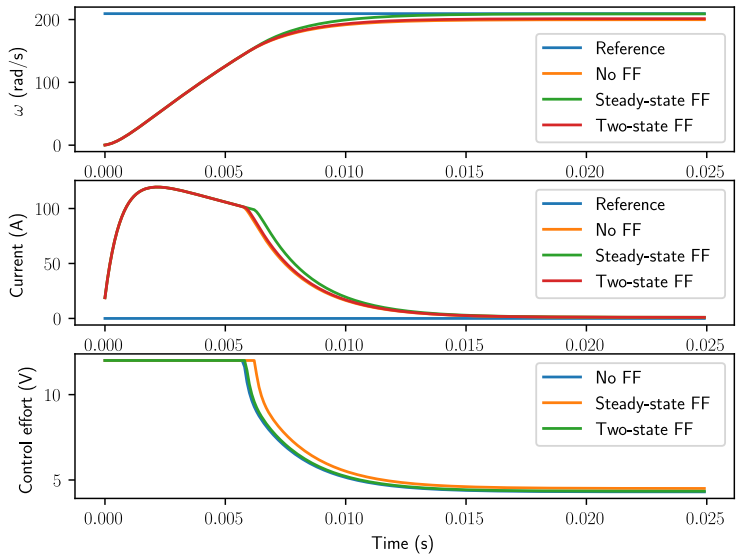


Figure 9.3: Second-order CIM motor response with various feedforwards

If control effort is considered inexpensive, $\mathbf{R} \ll \mathbf{Q}$ and $\mathbf{u}_{ff}$ approaches corollary 9.11.6.

Corollary 9.11.6 — Two-state feedforward with inexpensive control effort.

$$\mathbf{u}_{ff} = \mathbf{K}_{ff}(\mathbf{r}_{k+1} - \mathbf{A}\mathbf{r}_k) \tag{9.22}$$

$$\text{where } \mathbf{K}_{ff} = (\mathbf{B}^T\mathbf{Q}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{Q} \tag{9.23}$$

R If the cost matrix $\mathbf{Q}$ isn't included in the cost function (that is, $\mathbf{Q}$ is set to the identity matrix), $\mathbf{K}_{ff}$ becomes the Moore-Penrose pseudoinverse of $\mathbf{B}$ given by $\mathbf{B}^\dagger = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$.

9.11.3 Case studies of feedforwards

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. The two-state feedforward doesn't support this, but can be used for reference tracking as well with the same tuning parameters as LQR design. Figure 9.3 shows both types of feedforwards applied to a second-order CIM motor model.

The two-state feedforward isn't as effective in figure 9.3 because the $\mathbf{R}$ matrix penalized control effort. If the $\mathbf{R}$ cost matrix is removed from the two-state feedforward calculation, the reference tracking is much better (see figure 9.4).

9.11.4 Feedforwards for unmodeled dynamics

While the feedforwards previously mentioned only handle modeled dynamics, one can also include feedforwards for unmodeled dynamics separately. Consider an elevator model which doesn't include gravity. A constant voltage offset 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.

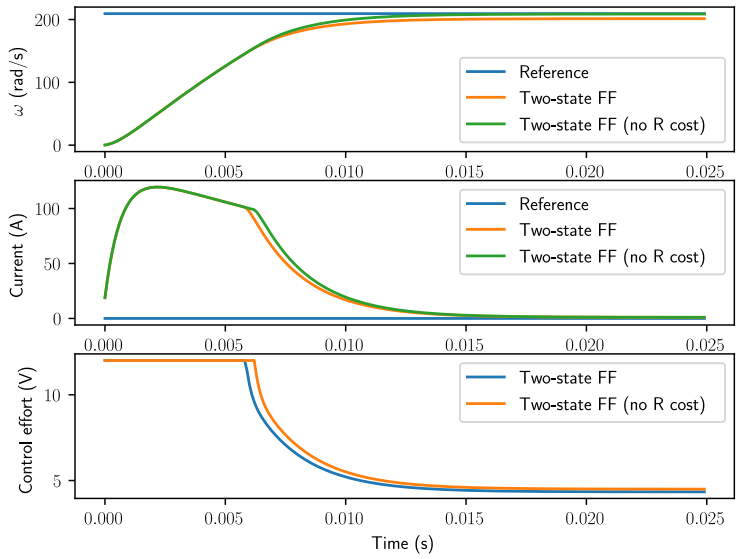


Figure 9.4: Second-order CIM motor response with two-state feedforwards

$$u_{ff} = V_{app} \tag{9.24}$$

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

$$u_{ff} = V_{app} \cos\theta \tag{9.25}$$

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[3].

9.12 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.

There are two drawbacks to this method. First, there is integral windup on a unit step input. That is, the integrator accumulates even if the system is tracking the model correctly. The second is demonstrated by an example from Jared Russell of FRC team 254. Say there is a position/velocity trajectory for some plant to follow. Without integral control, one can calculate a desired $\mathbf{Kx}$ to use as the control input. As a result of using both desired position and velocity, reference tracking is good.

[3] 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.

With integral control added, the reference is always the desired position, but there is no way to tell the controller the desired velocity.

Consider carefully whether integral control is necessary. One can get relatively close without integral control, and integral adds all the issues listed above. Below, it is assumed that the controls designer has determined that integral control will be worth the inconvenience.

There are three methods FRC team 971 has used over the years:

1. Augment the plant as described earlier. For an arm, one would add an "integral of position" state.
2. Add an integrator to the output of the controller, then estimate the control effort being applied. 971 has called this Delta U control. The upside is that it doesn't have the windup issue described above; the integrator only acts if the system isn't behaving like the model, which was the original intent. The downside is working with it is very confusing.
3. Estimate an "error" in the observer and compensate for it. This quantity is the difference between what was applied and what was observed to happen. To use it, you simply add it to your control input and it will converge.

We'll present the first and third methods since the third is strictly better than the second.

9.12.1 Plant augmentation

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

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

The plant is augmented as

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

The controller is augmented as

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

9.12.2 U error estimation

Let u_{error} be the difference between the input actually applied to a system and the desired input. The u_{error} term is then added to the system as follows.

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

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

For a multiple-output system, this would be

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

where $\mathbf{B}_{error}$ is the 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 $\mathbf{u}$.

The plant is augmented as

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

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.

This page intentionally left blank

10. Digital 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 set of differential equations like

$$\dot{x} = x - 3 \tag{10.1}$$

to a set of difference equations like

$$x_{k+1} = x_k + (x_k - 3)\Delta T \tag{10.2}$$

where the difference equation is run with some update period denoted by T, by ΔT, or sometimes sloppily by dt.

While higher order terms of a differential equation are derivatives of the state variable (e.g., $\ddot{x}$ in relation to equation (10.1)), 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 (10.2)).

10.1 Phase loss

However, discretization has drawbacks. Since a microcontroller performs discrete steps, there is a sample delay that introduces 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.

10.2 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 10.1 contains a few common points and figure 10.1 shows the mapping visually.

s-plane	z-plane
$(0, 0)$	$(1, 0)$
imaginary axis	edge of unit circle
$(-\infty, 0)$	$(0, 0)$

Table 10.1: Mapping from s-plane to z-plane

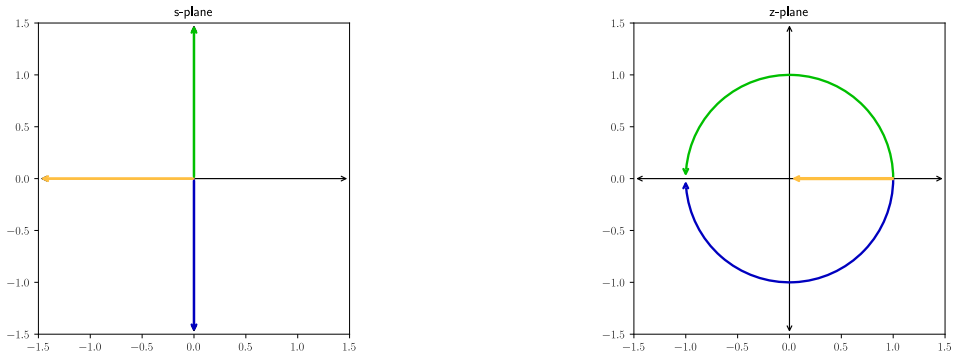


Figure 10.1: Mapping of axes from s-plane (left) to z-plane (right)

10.2.1 z-plane stability

Eigenvalues of a system that are within the unit circle are stable, but why is that? Let's consider a scalar equation $x_{k+1} = ax_k$. $a < 1$ makes x_{k+1} converge to zero. The same applies to a complex number like $z = x + yi$ for $x_{k+1} = zx_k$. If the magnitude of the complex number z is less than one, x_{k+1} will converge to zero. Values with a magnitude of 1 oscillate forever because x_{k+1} never decays.

10.2.2 z-plane behavior

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).

You may notice that poles can be placed at $(0, 0)$ in the z-plane. This is known as 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. These will be discussed in more detail with LQR and LQE.

If poles from $(1, 0)$ to $(0, 0)$ on the x-axis approach infinity, then what do poles from $(-1, 0)$ to $(0, 0)$ represent? Them being faster than infinity doesn't make sense. Poles in this location exhibit oscillatory behavior similar to complex conjugate pairs. See figures 10.2 and 10.3. The jaggedness of these signals is due to the frequency of the system dynamics being above the Nyquist frequency. The discretized signal doesn't have enough samples to reconstruct the continuous system's dynamics.

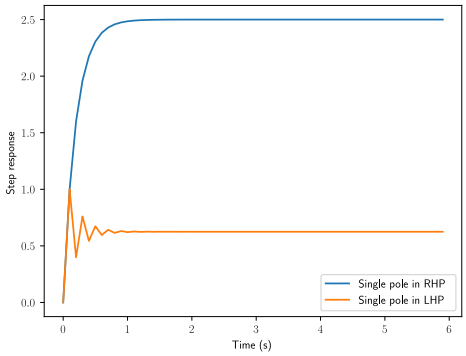


Figure 10.2: Single poles in various locations in z-plane

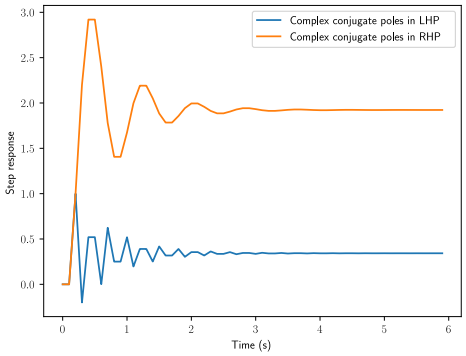


Figure 10.3: Complex conjugate poles in various locations in z-plane

10.2.3 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 $44.1k\,Hz$. The maximum frequency a typical human can hear is about $20\,kHz$, so the Nyquist frequency is $40\,kHz$. ($44.1kHz$ 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[1]. Figure 10.4 provides a demonstration of aliasing.

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

[1]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.

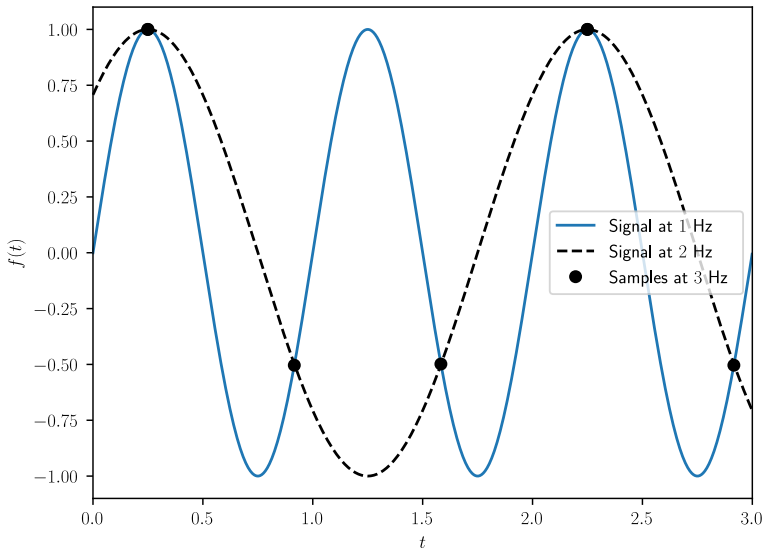


Figure 10.4: The samples of two sine waves can be identical when at least one of them is at a frequency above half the sample rate. In this case, the 2 Hz sine wave is above the Nyquist frequency 1.5 Hz.

10.3 Discretization methods

Discretization is done using a zero-order hold. That is, the system state is only updated at discrete intervals and it's held constant between samples (see figure 10.5). The exact method of applying this uses the matrix exponential, but this can be computationaly 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 10.6 and 10.7). 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.

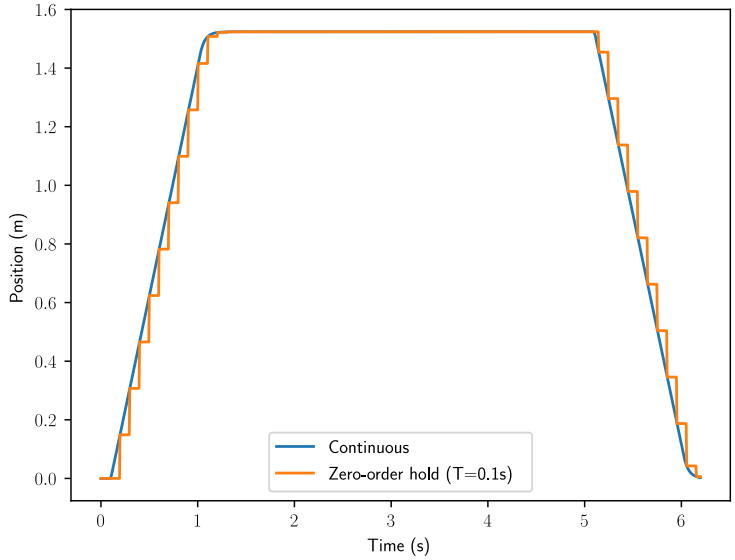


Figure 10.5: Zero-order hold of a system response

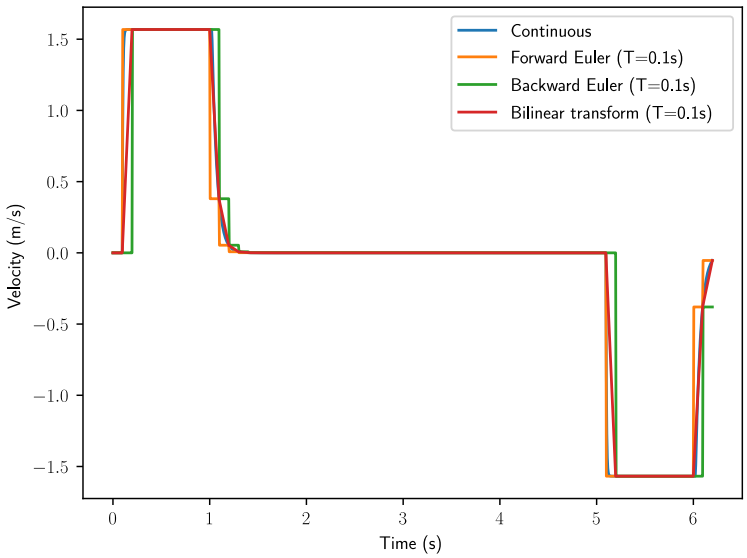


Figure 10.6: Discretization methods applied to velocity data

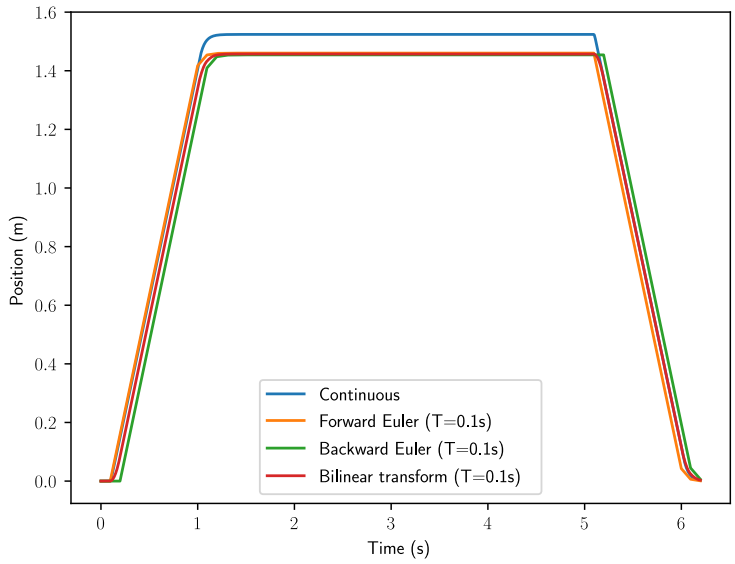


Figure 10.7: Position plot of discretization methods applied to velocity data

10.4 Effects of discretization on controller performance

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 250ms, you obviously want to run the controller with a period less than 250ms. 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 1kHz doesn't make sense for this because the plant input the controller computes won't change much, if at all, in 1ms.

Figures 10.8, 10.9, and 10.10 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 (we'll cover the matrix exponential in the next section).

Table 10.2 compares the Taylor series expansions of the discretization methods presented so far (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

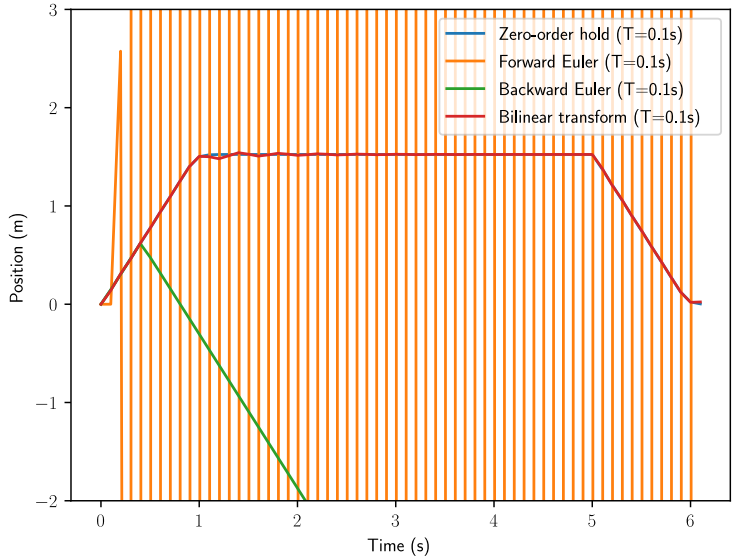


Figure 10.8: Sampling methods for system simulation with $T = 0.1s$

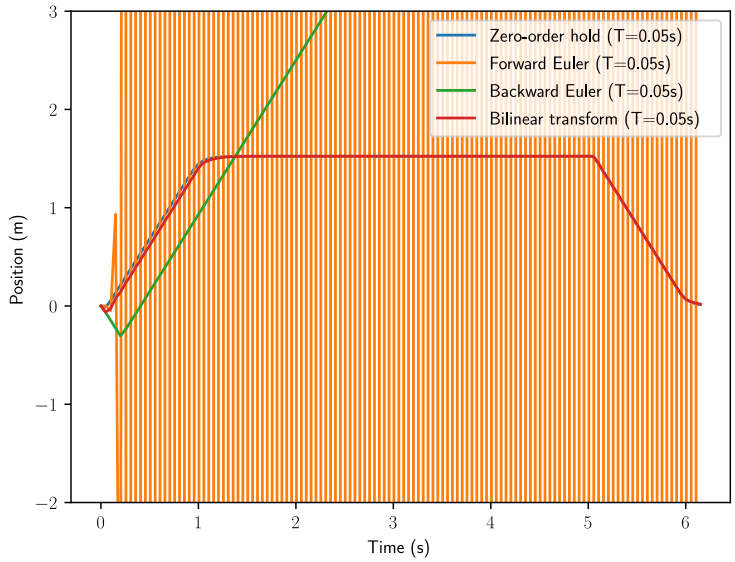


Figure 10.9: Sampling methods for system simulation with $T = 0.05s$

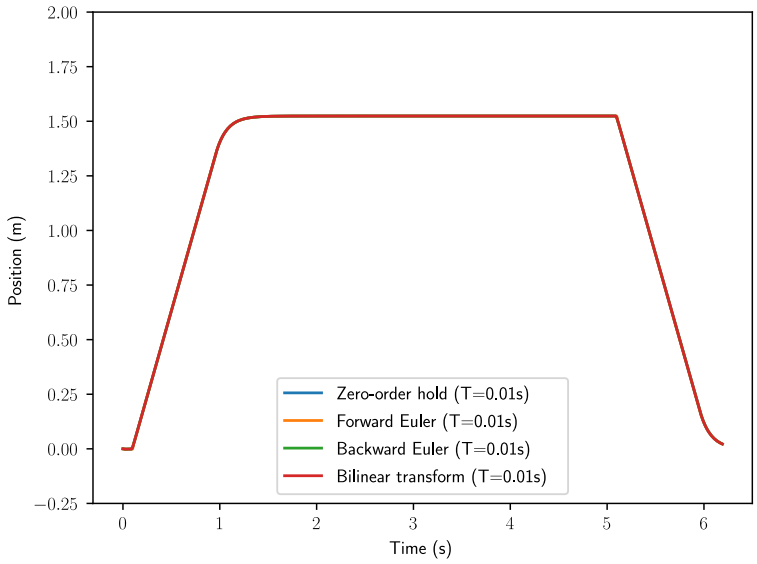


Figure 10.10: Sampling methods for system simulation with $T = 0.01s$

well.

Method	Conversion	Taylor series expansion
Zero-order hold	$z = e^{Ts}$	$z = 1 + Ts + \frac{1}{2}T^2s^2 + \frac{1}{6}T^3s^3 + \dots$
Bilinear	$z = \frac{1+\frac{1}{2}Ts}{1-\frac{1}{2}Ts}$	$z = 1 + Ts + \frac{1}{2}T^2s^2 + \frac{1}{4}T^3s^3 + \dots$
Forward Euler	$z = 1 + Ts$	$z = 1 + Ts$
Reverse Euler	$z = \frac{1}{1-Ts}$	$z = 1 + Ts + T^2s^2 + T^3s^3 + \dots$

Table 10.2: Taylor series expansions of discretization methods (scalar case). The zero-order hold discretization method is exact.

10.5 The matrix exponential

The matrix exponential (and system discretization in general) is typically solved with a computer. Python Control's `StateSpace.sample()` with the "zoh" method (the default) does this.

Definition 10.5.1 — 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.

$$e^{\mathbf{X}} = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{X}^k \tag{10.3}$$

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 set of differential equations $\dot{\mathbf{x}} = \mathbf{Ax}$ we use to describe systems (systems also have a $\mathbf{Bu}$ term, but we'll

ignore it for clarity). The solution to this type of differential equation uses an exponential. Since we are using matrices and vectors here, we use the matrix exponential.

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$$

where $\mathbf{x}_0$ contains the initial conditions. If the initial state is the current system state, then we can describe the system's state over time as

$$\mathbf{x}_{k+1} = e^{\mathbf{A}T}\mathbf{x}_k$$

where T is the time between samples $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$.

10.6 The Taylor series

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 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[2]. Figure 10.11 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 (10.3) 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 10.3 compares the Taylor series expansions of the discretization methods for the matrix case. These use a more complex formula which we won't present here.

Method	Conversion	Taylor series expansion
Zero-order hold	$\mathbf{\Phi} = e^{\mathbf{A}T}$	$\mathbf{\Phi} = \mathbf{I} + \mathbf{A}T + \frac{1}{2}\mathbf{A}^2T^2 + \frac{1}{6}\mathbf{A}^3T^3 + \dots$
Bilinear	$\mathbf{\Phi} = \left(\mathbf{I} + \frac{1}{2}\mathbf{A}T\right)\left(\mathbf{I} - \frac{1}{2}\mathbf{A}T\right)^{-1}$	$\mathbf{\Phi} = \mathbf{I} + \mathbf{A}T + \frac{1}{2}\mathbf{A}^2T^2 + \frac{1}{4}\mathbf{A}^3T^3 + \dots$
Forward Euler	$\mathbf{\Phi} = \mathbf{I} + \mathbf{A}T$	$\mathbf{\Phi} = \mathbf{I} + \mathbf{A}T$
Reverse Euler	$\mathbf{\Phi} = (\mathbf{I} - \mathbf{A}T)^{-1}$	$\mathbf{\Phi} = \mathbf{I} + \mathbf{A}T + \mathbf{A}^2T^2 + \mathbf{A}^3T^3 + \dots$

Table 10.3: Taylor series expansions of discretization methods (matrix case). The zero-order hold discretization method is exact.

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

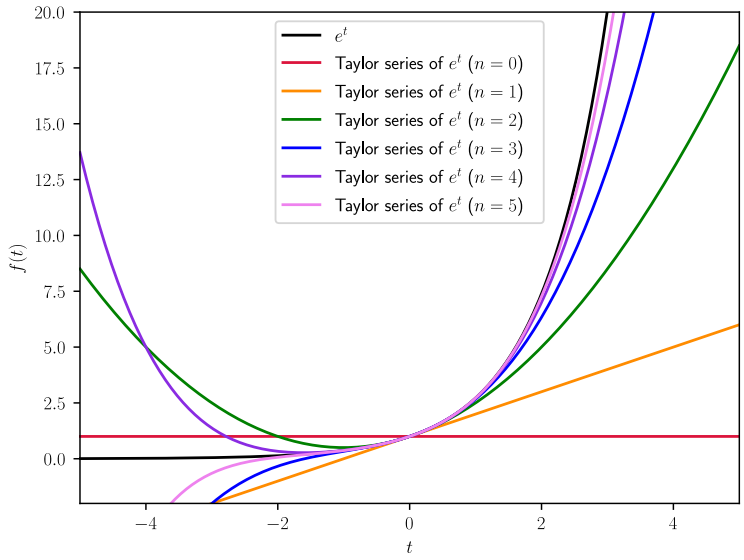


Figure 10.11: Taylor series expansions of e^t around $t = 0$ for n terms

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

10.7 Zero-order hold for state-space

Given 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.

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

The model can be discretized as follows

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

with covariances

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

Theorem 10.7.1 — Zero-order hold for state-space.

$$\mathbf{A}_d = e^{\mathbf{A}_c T} \tag{10.4}$$
$$\mathbf{B}_d = \int_0^T e^{\mathbf{A}_c \tau} d\tau \mathbf{B}_c = \mathbf{A}_c^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}_c \tag{10.5}$$
$$\mathbf{C}_d = \mathbf{C}_c \tag{10.6}$$
$$\mathbf{D}_d = \mathbf{D}_c \tag{10.7}$$
$$\mathbf{Q}_d = \int_{\tau=0}^T e^{\mathbf{A}_c \tau} \mathbf{Q}_c e^{\mathbf{A}_c^T \tau} d\tau \tag{10.8}$$
$$\mathbf{R}_d = \frac{1}{T}\mathbf{R}_c \tag{10.9}$$

where a subscript of d denotes discrete, a subscript of c denotes the continuous version of the corresponding matrix, T is the sample period for the discrete system, and $e^{\mathbf{A}_c T}$ is the matrix exponential of $\mathbf{A}_c$.

See appendix D.3 for derivations.

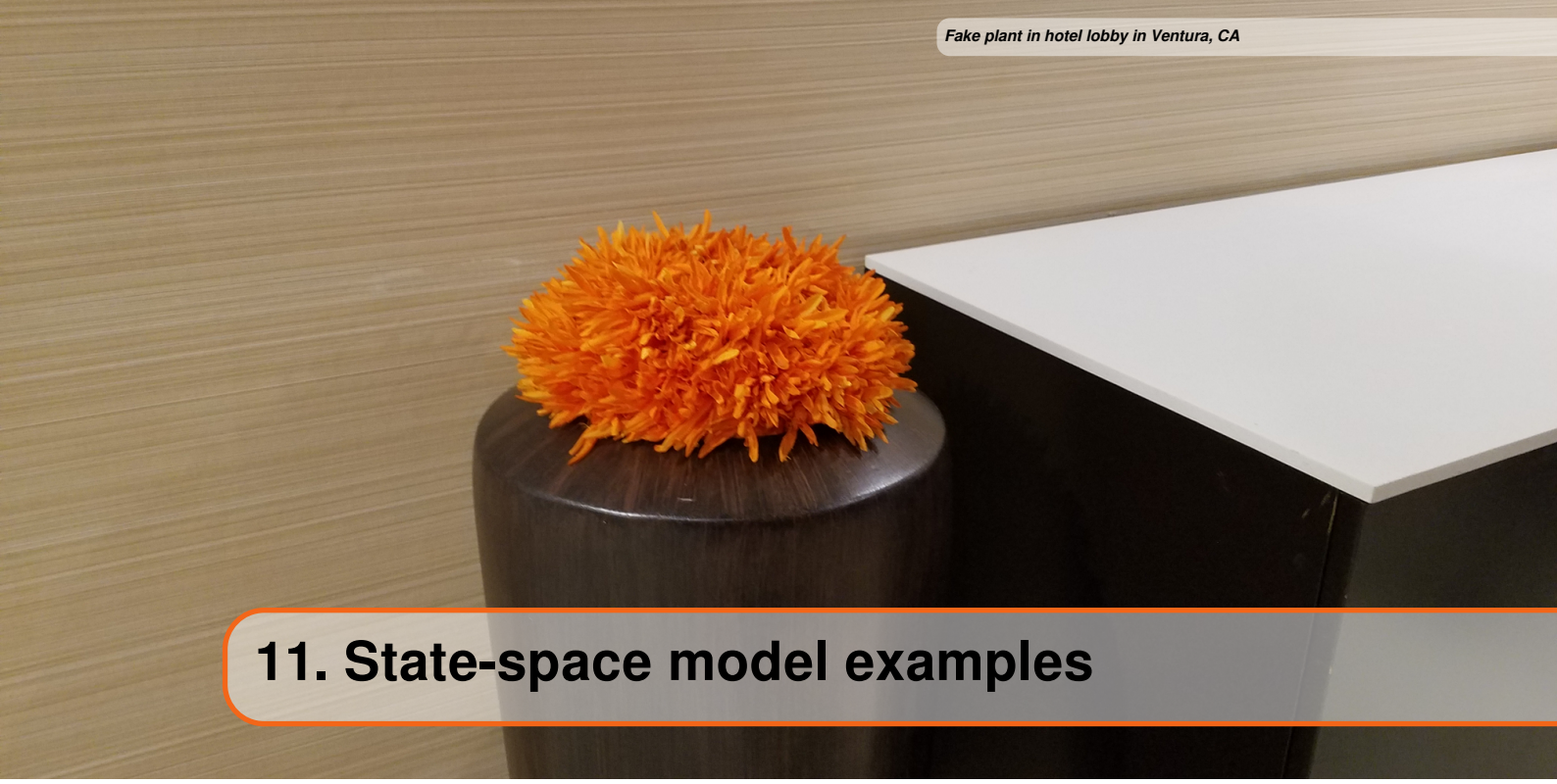
$\mathbf{Q}_d$ is computed as

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

and $\mathbf{Q}_d$ is the lower-right quadrant multiplied by the upper-right quadrant [18]. To compute $\mathbf{A}_d$ and $\mathbf{B}_d$ in one step, one can utilize the following property.

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

This page intentionally left blank


Fake plant in hotel lobby in Ventura, CA

11. State-space model examples

Up to now, we've just been teaching what tools are available. Now, we'll go into specifics on how to apply them and provide advice on certain applications.

The code shown in each example can be obtained from frccontrol's Git repository at `https://github.com/calcmogul/frccontrol/tree/master/examples`. See appendix B for setup instructions.

11.1 Pendulum

11.1.1 Write model in state-space representation

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.

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

Since this model is nonlinear, we should linearize it. We will use the small angle approximation ($\sin\theta = \theta$ for small values of θ).

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

Now write the model in state-space representation.

$$\begin{bmatrix}\dot{\theta} \\ \omega\end{bmatrix} = \begin{bmatrix}0 & 1 \\ -\frac{g}{l} & 0\end{bmatrix}\begin{bmatrix}\theta \\ \omega\end{bmatrix} \tag{11.1}$$

11.1.2 Add estimator for unmeasured states

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.

For example, 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

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

11.1.3 Implement controller

Use Bryson's rule when making the performance vs control effort trade-off. Optimizing for performance will get you to the reference as fast as possible while optimizing control effort will get you to the reference in the most "fuel-efficient" way possible. The latter, for example, would potentially avoid voltage drops from motor usage on robots with a limited power supply, but the result would be slower to reach the reference.

11.1.4 Simulate model/controller

This can be done in any platform supporting numerical computation. Common choices are MATLAB, v-REP, or Python. Tweak the LQR gains as necessary.

If you're comfortable enough with it, you can use the controller designed by LQR as a starting point and tweak the pole locations after that with pole placement to produce the desired response.

Simulating a closed-loop system

Recall equation (9.9) where a closed-loop system is written as $\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{BKr}$. In the open-loop system, our control input $\mathbf{u}$ was a vector of system inputs. In the closed-loop system, our control input $\mathbf{u}$ is now a vector of reference states $\mathbf{r}$. To use this form in simulation, the corresponding state-space matrices, which we'll denote with apostrophes, would be

$$\mathbf{A}' = \mathbf{A} - \mathbf{BK}$$
$$\mathbf{B}' = \mathbf{BK}$$
$$\mathbf{C}' = \mathbf{C} - \mathbf{DK}$$
$$\mathbf{D}' = \mathbf{DK}$$

11.1.5 Verify pole locations

Check the pole locations as a sanity check and to potentially gain an intuition for the chosen pole locations.

11.1.6 Unit test

Write unit tests to test the model performance and robustness with different initial conditions and references. For C++, we recommend Google Test.

11.1.7 Test on real system

Try the controller on a real system with low maximum control inputs for safety. The control inputs can be increased after verifying the sensors function and mechanisms move the correct direction.

11.2 Elevator

11.2.1 Continuous state-space model

The position and velocity of the elevator can be written as

$$\dot{x}_m = v_m \tag{11.2}$$
$$\dot{v}_m = a_m \tag{11.3}$$

where by equation (4.17),

$$a_m = \frac{GK_t}{Rrm}V - \frac{G^2K_t}{Rr^2mK_v}v_m$$

Substitute this into equation (11.3).

$$\dot{v}_m = \frac{GK_t}{Rrm}V - \frac{G^2K_t}{Rr^2mK_v}v_m$$
$$\dot{v}_m = -\frac{G^2K_t}{Rr^2mK_v}v_m + \frac{GK_t}{Rrm}V \tag{11.4}$$

Theorem 11.2.1 — Elevator state-space model.

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

$$\mathbf{x} = \begin{bmatrix} x \\ v_m \end{bmatrix} \quad \mathbf{y} = x \quad \mathbf{u} = V$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{G^2K_t}{Rr^2mK_v} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{GK_t}{Rrm} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \mathbf{D} = 0 \tag{11.5}$$

11.2.2 Model augmentation

As per subsection 9.12.2, we will now augment the model so a u_{error} term 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.

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

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0}_{1\times 2} & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{11.6}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{11.7}$$

This will compensate for unmodeled dynamics such as gravity. However, using a constant feedforward to counteract gravity is preferred over this method.

11.2.3 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[1] creates and tests a controller for it.

R Python Control currently doesn't support finding the transmission zeroes of MIMO systems with a different number of inputs than outputs, so `control.pzmap()` and `frccontrol.System.plot_pzmaps()` fail with an error if Slycot isn't installed.

Figure 11.1 shows the pole-zero maps for the open-loop system, closed-loop system, and observer. Figure 11.2 shows the system response with them.

Figure 11.2 shows the system response.

11.2.4 Implementation

The script linked above also generates two files: ElevatorCoeffs.h and ElevatorCoeffs.cpp. These can be used with the WPILib StateSpacePlant, StateSpaceController, and StateSpaceObserver classes in C++ and Java. A C++ implementation of this elevator controller is available online[2].

R Instead of implementing u_{error} estimation to compensate for gravity, one can apply a constant voltage feedforward since input voltage is proportional to force and gravity is a constant force.

[1] `https://github.com/calcmogul/frccontrol/blob/master/examples/elevator.py`

[2] `https://github.com/calcmogul/allwpilib/tree/state-space/wpilibcExamples/src/main/cpp/examples/StateSpaceElevator`

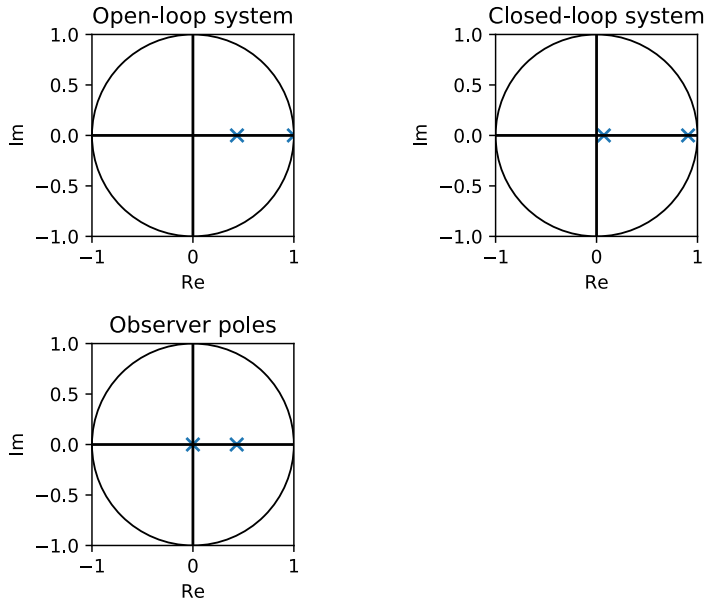


Figure 11.1: Elevator pole-zero maps

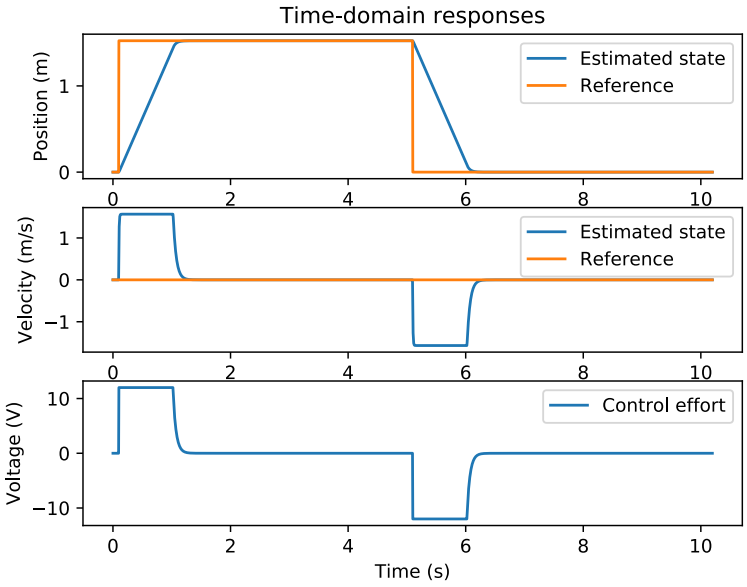


Figure 11.2: Elevator response

11.3 Flywheel

11.3.1 Continuous state-space model

By equation (4.20)

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

Theorem 11.3.1 — Flywheel state-space model.

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

$$\mathbf{x} = \omega_f \quad \mathbf{y} = \omega_f \quad \mathbf{u} = V$$
$$\mathbf{A} = -\frac{G^2 K_t}{K_v R J} \quad \mathbf{B} = \frac{G K_t}{R J} \quad \mathbf{C} = 1 \quad \mathbf{D} = 0 \tag{11.8}$$

11.3.2 Model augmentation

As per subsection 9.12.2, we will now augment the model so a u_{error} term 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.

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

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ 0 & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{11.9}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{11.10}$$

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

11.3.3 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[3] creates and tests a controller for it.

Figure 11.3 shows the pole-zero maps for the open-loop system, closed-loop system, and observer. Figure 11.4 shows the system response with them.

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

[3] `https://github.com/calcmogul/frccontrol/blob/master/examples/flywheel.py`

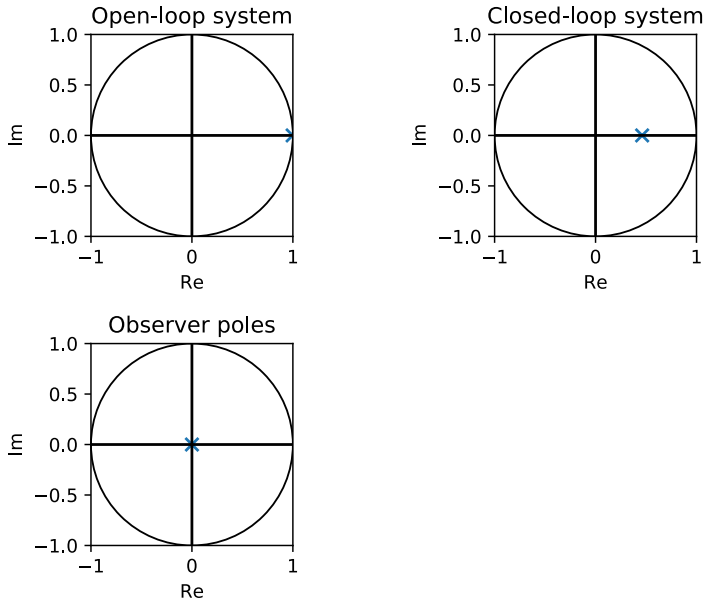


Figure 11.3: Flywheel pole-zero maps

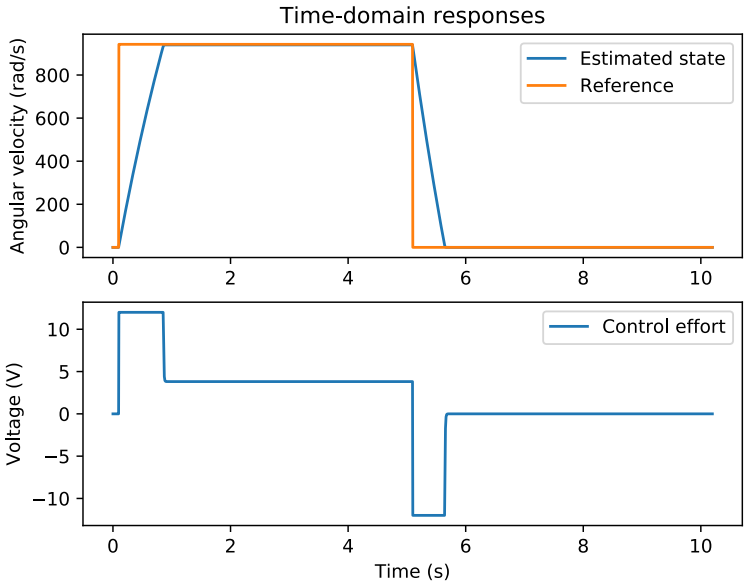


Figure 11.4: Flywheel response

11.3.4 Implementation

The script linked above also generates two files: FlywheelCoeffs.h and FlywheelCoeffs.cpp. These can be used with the WPILib StateSpacePlant, StateSpaceController, and StateSpaceObserver classes in C++ and Java. A C++ implementation of this flywheel controller is available online[4].

11.4 Drivetrain

11.4.1 Continuous state-space model

The position and velocity of each drivetrain side can be written as

$$\dot{x}_l = v_l \tag{11.11}$$
$$\dot{v}_l = \dot{v}_l \tag{11.12}$$
$$\dot{x}_r = v_r \tag{11.13}$$
$$\dot{v}_r = \dot{v}_r \tag{11.14}$$

By equations (4.28) and (4.29)

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

Theorem 11.4.1 — Drivetrain state-space model.

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

$$\mathbf{x} = \begin{bmatrix} x_l \\ v_l \\ x_r \\ v_r \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} x_l \\ x_r \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ 0 & 0 & 0 & 1 \\ 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ 0 & 0 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{D} = \mathbf{0}_{2\times 2} \tag{11.15}$$

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

[4] https://github.com/calcmogul/allwpilib/tree/state-space/wpilibcExamples/src/main/cpp/examples/StateSpaceFlywheel

11.4.2 Model augmentation

As per subsection 9.12.2, we will now augment the model so u_{error} terms 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.

For this augmented model, the left and right wheel positions are filtered encoder positions and are not adjusted for heading error. The turning velocity computed from the left and right velocities is adjusted by the gyroscope angular velocity. The angular velocity u_{error} term is the angular velocity error between the wheel speeds and the gyroscope measurement.

$$\mathbf{x}_{aug} = \begin{bmatrix} \mathbf{x} \\ u_{error,l} \\ u_{error,r} \\ u_{error,angle} \end{bmatrix} \quad \mathbf{y}_{aug} = \begin{bmatrix} \mathbf{y} \\ \omega \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix}$$

where ω is the angular rate of the robot's center of mass measured by a gyroscope.

$$\mathbf{B}_\omega = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix} \quad \mathbf{C}_\omega = \begin{bmatrix} 0 & -\frac{1}{2r_b} & 0 & \frac{1}{2r_b} \end{bmatrix}$$

$$\begin{aligned} \mathbf{A}_{aug} &= \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B}_\omega \\ \mathbf{0}_{3\times4} & \mathbf{0}_{3\times2} & \mathbf{0}_{3\times1} \end{bmatrix} & \mathbf{B}_{aug} &= \begin{bmatrix} \mathbf{B} \\ \mathbf{0}_{3\times2} \end{bmatrix} \\ \mathbf{C}_{aug} &= \begin{bmatrix} \mathbf{C} & \mathbf{0}_{2\times3} \\ \mathbf{C}_\omega & \mathbf{0}_{1\times3} \end{bmatrix} & \mathbf{D}_{aug} &= \begin{bmatrix} \mathbf{D} \\ \mathbf{0}_{1\times2} \end{bmatrix} \end{aligned} \tag{11.16}$$

The augmentation of $\mathbf{A}$ with $\mathbf{B}$ maps the left and right wheel velocity u_{error} terms to their respective states. The augmentation of $\mathbf{A}$ with $\mathbf{B}_\omega$ maps the angular velocity u_{error} term to corresponding changes in the left and right wheel positions.

$\mathbf{C}_\omega$ maps the left and right wheel velocities to an angular velocity estimate that the observer can compare against the gyroscope measurement.

$$\mathbf{K}_{error} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & \mathbf{K}_{error} \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{11.17}$$

This will compensate for unmodeled dynamics such as understeer due to the wheel friction inherent in skid-steer robots.

R The process noise for the angular velocity u_{error} term should be the encoder model uncertainty. The augmented measurement noise term is obviously for the gyroscope measurement.

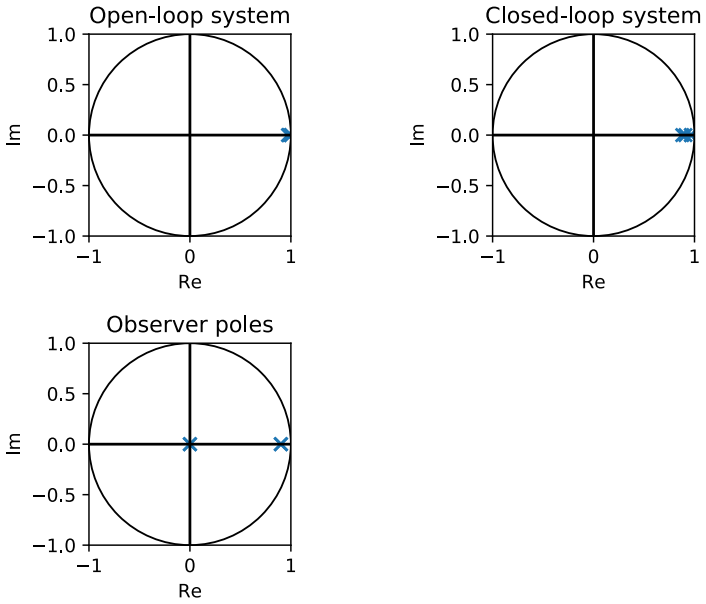


Figure 11.5: Drivetrain pole-zero maps

11.4.3 Simulation

Python Control will be used to discretize the model and simulate it. One of the frccontrol examples[5] creates and tests a controller for it.

R Python Control currently doesn't support finding the transmission zeroes of MIMO systems with a different number of inputs than outputs, so `control.pzmap()` and `frccontrol.System.plot_pzmaps()` fail with an error if Slycot isn't installed.

Figure 11.5 shows the pole-zero maps for the open-loop system, closed-loop system, and observer. Figure 11.6 shows the system response with them.

Figure 11.6 shows the 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.

11.4.4 Implementation

The script linked above also generates two files: DrivetrainCoeffs.h and DrivetrainCoeffs.cpp. These can be used with the WPILib StateSpacePlant, StateSpaceController, and StateSpaceObserver classes in C++ and Java. A C++ implementation of this drivetrain controller is available online[6].

[5] `https://github.com/calcmogul/frccontrol/blob/master/examples/drivetrain.py`

[6] `https://github.com/calcmogul/allwpilib/tree/state-space/wpilibcExamples/src/main/cpp/examples/StateSpaceDrivetrain`

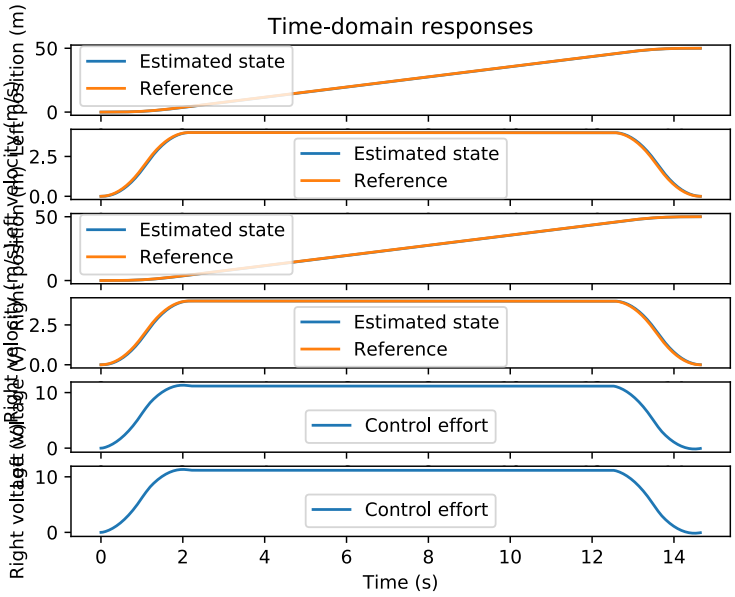


Figure 11.6: Drivetrain response

11.5 Single-jointed arm

11.5.1 Continuous state-space model

The position and velocity of the elevator can be written as

$$\dot{\theta}_{arm} = \omega_{arm} \tag{11.18}$$
$$\dot{\omega}_{arm} = \dot{\omega}_{arm} \tag{11.19}$$

By equation (4.32)

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

Theorem 11.5.1 — Single-jointed arm state-space model.

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

$$\mathbf{x} = \begin{bmatrix} \theta_{arm} \\ \omega_{arm} \end{bmatrix} \quad \mathbf{y} = \theta_{arm} \quad \mathbf{u} = V$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{G^2 K_t}{K_v R J} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{G K_t}{RJ} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \mathbf{D} = 0 \tag{11.20}$$

11.5.2 Model augmentation

As per subsection 9.12.2, we will now augment the model so a u_{error} term 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.

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

$$\mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0}_{1\times 2} & 0 \end{bmatrix} \quad \mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} \quad \mathbf{C}_{aug} = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \quad \mathbf{D}_{aug} = \mathbf{D} \tag{11.21}$$

$$\mathbf{K}_{aug} = \begin{bmatrix} \mathbf{K} & 1 \end{bmatrix} \quad \mathbf{r}_{aug} = \begin{bmatrix} \mathbf{r} \\ 0 \end{bmatrix} \tag{11.22}$$

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.

11.5.3 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.

> **R** Python Control currently doesn't support finding the transmission zeroes of MIMO systems with a different number of inputs than outputs, so `control.pzmap()` and `frccontrol.System.plot_pzmaps()` fail with an error if Slycot isn't installed.

Figure 11.7 shows the pole-zero maps for the open-loop system, closed-loop system, and observer. Figure 11.8 shows the system response with them.

Figure 11.8 shows the system response.

11.5.4 Implementation

The script linked above also generates two files: SingleJointedArmCoeffs.h and SingleJointedArmCoeffs.cpp. These can be used with the WPILib StateSpacePlant, StateSpaceController, and StateSpaceObserver classes in C++ and Java. A C++ implementation of this single-jointed arm controller is available online[8].

[7] `https://github.com/calcmogul/frccontrol/blob/master/examples/single_jointed_arm.py`

[8] `https://github.com/calcmogul/allwpilib/tree/state-space/wpilibcExamples/src/main/cpp/examples/StateSpaceSingleJointedArm`

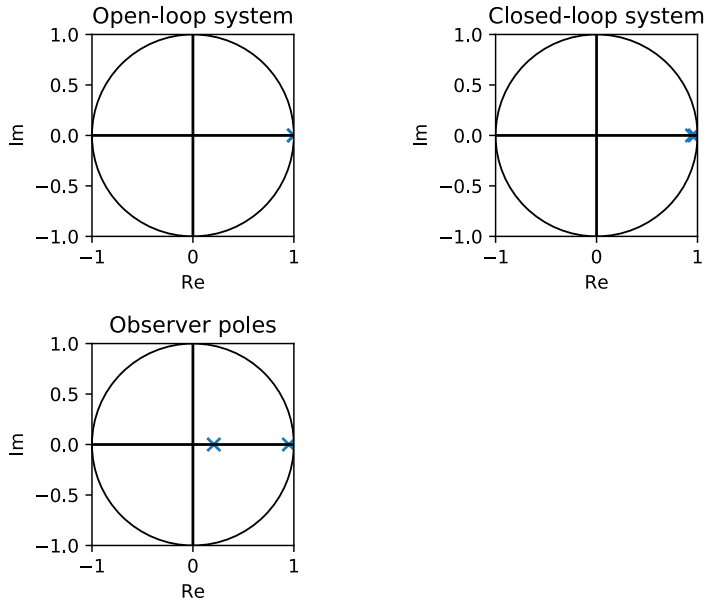


Figure 11.7: Single-jointed arm pole-zero maps

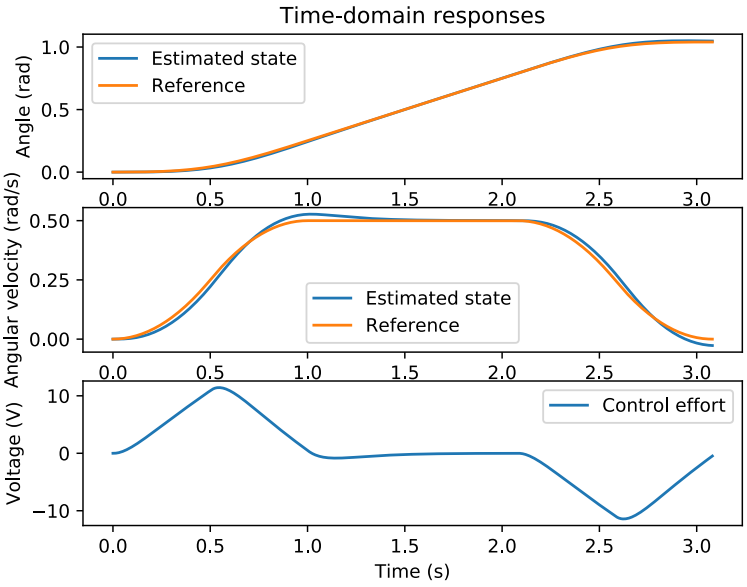


Figure 11.8: Single-jointed arm response

11.6 Rotating claw

11.6.1 Continuous state-space model

11.6.2 Simulation


Trees by Interdisciplinary Sciences building at UCSC

12. 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.

12.1 Introduction

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

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

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.

12.2 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 partial derivative of the functions.

$$\mathbf{A} = \frac{\partial f(\mathbf{x},\mathbf{0},\mathbf{0})}{\partial \mathbf{x}} \quad \mathbf{B} = \frac{\partial f(\mathbf{0},\mathbf{u},\mathbf{0})}{\partial \mathbf{u}} \quad \mathbf{C} = \frac{\partial h(\mathbf{x},\mathbf{0},\mathbf{0})}{\partial \mathbf{x}} \quad \mathbf{D} = \frac{\partial h(\mathbf{0},\mathbf{u},\mathbf{0})}{\partial \mathbf{u}}$$

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 [15]. 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.

12.3 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.

Essentially, 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 is concerned with finding a function representing the energy in a system to prove that the system is stable around an equilibrium point. This can be used to prove a system's open-loop stability as well as its closed-loop stability in the presence of a controller. Typically, these functions include the energy of the system or the derivatives of the system state, which are then used to show the system decays to some ground state.

Lyapunov functions are merely sufficient to prove stability (as opposed to a specific Lyapunov function being necessary). If one function doesn't prove it, another candidate should be tried. For this reason, we refer to these functions as *Lyapunov candidate functions*.

12.4 Control law for nonholonomic wheeled vehicle

Why would we need a nonlinear control law in addition to the linear ones we have used so far? If we use the original approach with an LQR controller for left and right position and velocity states, the controller only deals with the local pose. If the robot deviates from the path, there is no way for

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

the controller to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like an LQR controller back in by adjusting the references of the LQR controller.

12.4.1 Ramsete controller

The paper *Control of Wheeled Mobile Robots: An Experimental Overview* describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and θ; and a desired pose consisting of x_d, y_d, and θ_d [19]. We'll call it the Ramsete nonlinear control law because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SErvizi e le TEcnologie").

Velocity and turning rate command derivation

The state tracking error $\mathbf{e}$ in the vehicle's coordinate frame is defined as

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix}$$

where e_1 is the tracking error in x, e_2 is the tracking error in y, and e_3 is the tracking error in θ. The 3×3 matrix is a rotation matrix that transforms the error in the pose (represented by $x_d - x$, $y_d - y$, and $\theta_d - \theta$) from the global coordinate frame into the vehicle's coordinate frame.

Multiplying the matrices out gives

$$\begin{aligned} e_1 &= (x_d - x)\cos\theta + (y_d - y)\sin\theta \\ e_2 &= -(x_d - x)\sin\theta + (y_d - y)\cos\theta \\ e_3 &= \theta_d - \theta \end{aligned}$$

$$\begin{aligned} e_1 &= (x_d - x)\cos\theta + (y_d - y)\sin\theta && (12.1) \\ e_2 &= (y_d - y)\cos\theta - (x_d - x)\sin\theta && (12.2) \\ e_3 &= \theta_d - \theta && (12.3) \end{aligned}$$

We will use the following control laws u_1 and u_2 for velocity and turning rate respectively.

$$\begin{aligned} u_1 &= -k_1 e_1 \\ u_2 &= -k_2 v_d \operatorname{sinc}(e_3) e_2 - k_3 e_3 \end{aligned} \tag{12.4}$$

where k_1, k_2, and k_3 are time-varying gains and $\operatorname{sinc}(e_3)$ is defined as $\frac{\sin e_3}{e_3}$.

Substitute equations (12.1), (12.2), and (12.3) into the control laws.

$$u_1 = -k_1((x_d - x)\cos\theta + (y_d - y)\sin\theta$$

$$u_2 = -k_2 v_d \operatorname{sinc}(\theta_d - \theta)((y_d - y)\cos\theta - (x_d - x)\sin\theta) - k_3(\theta_d - \theta)$$

Our velocity and turning rate commands for the vehicle will use the following nonlinear transformation of the inputs.

$$v = v_d \cos e_3 - u_1$$
$$\omega = \omega_d - u_2$$

Substituting the control laws u_1 and u_2 into these equations gives

$$v = v_d \cos(\theta_d - \theta) - (-k_1((x_d - x)\cos\theta + (y_d - y)\sin\theta)$$
$$v = v_d \cos(\theta_d - \theta) + k_1((x_d - x)\cos\theta + (y_d - y)\sin\theta)$$

$$\omega = \omega_d - (-k_2 v_d \operatorname{sinc}(\theta_d - \theta)((y_d - y)\cos\theta - (x_d - x)\sin\theta) - k_3(\theta_d - \theta))$$
$$\omega = \omega_d + k_2 v_d \operatorname{sinc}(\theta_d - \theta)((y_d - y)\cos\theta - (x_d - x)\sin\theta) + k_3(\theta_d - \theta)$$

Theorem 12.4.1 Assuming that v_d and ω_d are bounded with bounded derivatives, and that $v_d(t) \to 0$ or $\omega_d(t) \to 0$ when $t \to \infty$, the control laws in equation (12.4) globally asymptotically stabilize the origin $\mathbf{e} = 0$.

Proof:

To prove convergence, the paper previously mentioned uses the following Lyapunov function.

$$V = \frac{k_2}{2}(e_1^2 + e_2^2) + \frac{e_3^2}{2}$$

where k_2 is a tuning constant, e_1 is the tracking error in x, e_2 is the tracking error in y, and e_3 is the tracking error in θ.

The time derivative along the solutions of the closed-loop system is nonincreasing since

$$\dot{V} = -k_1 k_2 e_1^2 - k_3 e_3^2 \leq 0$$

Thus, $\|e(t)\|$ is bounded, $\dot{V}(t)$ is uniformly continuous, and $V(t)$ tends to some limit value. Using the Barbalat lemma, $\dot{V}(t)$ tends to zero [19].

Nonlinear controller equations

Let $k_2 = b$.

Theorem 12.4.2 — Ramsete nonlinear control law.

$$v = v_d \cos(\theta_d - \theta) + k_1(v_d, \omega_d)((x_d - x)\cos\theta + (y_d - y)\sin\theta) \tag{12.5}$$

$$\omega = \omega_d + b v_d \operatorname{sinc}(\theta_d - \theta)((y_d - y)\cos\theta - (x_d - x)\sin\theta) + k_3(v_d, \omega_d)(\theta_d - \theta) \tag{12.6}$$

$$k_1(v_d(t), \omega_d(t)) = k_3(v_d(t), \omega_d(t)) = 2\zeta\sqrt{\omega_d^2(t) + b v_d^2(t)} \tag{12.7}$$

$$\operatorname{sinc}(\theta_d - \theta) = \frac{\sin(\theta_d - \theta)}{\theta_d - \theta} \tag{12.8}$$

v	velocity command	v_d	desired velocity
ω	turning rate command	ω_d	desired turning rate
x	actual x position in global coordinate frame	x_d	desired x position
y	actual y position in global coordinate frame	y_d	desired y position
θ	actual angle in global coordinate frame	θ_d	desired angle

b and ζ are tuning parameters where $b > 0$ and $\zeta \in (0, 1)$. Larger values of b make convergence more aggressive (like a proportional term), and larger values of ζ provide more damping.

v and ω should be the references for a reference tracker for the drivetrain. This can be a linear controller. x, y, and θ are obtained via a nonlinear pose estimator (see section 15.1 for how to implement one).

12.4.2 Linear reference tracker

We need a velocity reference tracker for the nonlinear controller's commands. Starting from equation (11.15), we'll derive two models for this purpose and compare their responses with a straight profile and as part of a nonlinear trajectory follower.

Left/right velocity reference tracker

$$\mathbf{x} = \begin{bmatrix} x_l \\ v_l \\ x_r \\ v_r \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} x_l \\ x_r \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ 0 & 0 & 0 & 1 \\ 0 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & 0 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ 0 & 0 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{D} = \mathbf{0}_{2\times 2}$$

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

To obtain a left/right velocity reference tracker, we can just remove the position states from the model.

Theorem 12.4.3 — Left/right velocity reference tracker.

$$\mathbf{x} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_3 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_1 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_4 \\ \left(\frac{1}{m} - \frac{r_b^2}{J}\right) C_2 & \left(\frac{1}{m} + \frac{r_b^2}{J}\right) C_4 \end{bmatrix} \tag{12.9}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{D} = \mathbf{0}_{2\times 2}$$

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 $C_4 = \frac{G_r K_t}{Rr}$.

See `https://github.com/calcmogul/state-space-guide/blob/master/code/ramsete_decoupled.py` for an implementation.

Linear/angular velocity reference tracker

Since the Ramsete controller produces velocity and turning rate commands, it would be more convenient if the states are velocity and turning rate instead of left and right wheel velocity. We'll create a second model that has velocity and angular velocity states and see how well it performs.

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

Substitute in equations (3.1) and (3.2).

$$\dot{v}_c + \dot{\omega} r_b = \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_1(v_c - \omega r_b) + C_2 V_l) + \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_3(v_c + \omega r_b) + C_4 V_r) \tag{12.10}$$

$$\dot{v}_c - \dot{\omega} r_b = \left(\frac{1}{m} - \frac{r_b^2}{J}\right)(C_1(v_c - \omega r_b) + C_2 V_l) + \left(\frac{1}{m} + \frac{r_b^2}{J}\right)(C_3(v_c + \omega r_b) + C_4 V_r) \tag{12.11}$$

Now, we need to solve for $\dot{v}_c$ and $\dot{\omega}$. First, we'll add equation (12.10) to equation (12.11).

$$\begin{aligned} 2\dot{v}_c &= \frac{2}{m}(C_1(v_c - \omega r_b) + C_2 V_l) + \frac{2}{m}(C_3(v_c + \omega r_b) + C_4 V_r) \\ \dot{v}_c &= \frac{1}{m}(C_1(v_c - \omega r_b) + C_2 V_l) + \frac{1}{m}(C_3(v_c + \omega r_b) + C_4 V_r) \\ \dot{v}_c &= \frac{1}{m}(C_1 + C_3)v_c + \frac{1}{m}(-C_1 + C_3)\omega r_b + \frac{1}{m}C_2 V_l + \frac{1}{m}C_4 V_r \\ \dot{v}_c &= \frac{1}{m}(C_1 + C_3)v_c + \frac{r_b}{m}(-C_1 + C_3)\omega + \frac{1}{m}C_2 V_l + \frac{1}{m}C_4 V_r \end{aligned}$$

Next, we'll subtract equation (12.11) from equation (12.10).

$$
\begin{aligned}
2\dot{\omega} r_b &= \frac{2r_b^2}{J}\left(C_1(v_c - \omega r_b) + C_2 V_l\right) - \frac{2r_b^2}{J}\left(C_3(v_c + \omega r_b) + C_4 V_r\right) \\
\dot{\omega} &= \frac{r_b}{J}\left(C_1(v_c - \omega r_b) + C_2 V_l\right) - \frac{r_b}{J}\left(C_3(v_c + \omega r_b) + C_4 V_r\right) \\
\dot{\omega} &= \frac{r_b}{J}(C_1 - C_3)v_c + \frac{r_b}{J}(-C_1 - C_3)\omega r_b + \frac{r_b}{J}C_2 V_l - \frac{r_b}{J}C_4 V_r \\
\dot{\omega} &= \frac{r_b}{J}(C_1 - C_3)v_c + \frac{r_b^2}{J}(-C_1 - C_3)\omega + \frac{r_b}{J}C_2 V_l - \frac{r_b}{J}C_4 V_r
\end{aligned}
$$

Now, just convert the two equations to state-space notation.

Theorem 12.4.4 — Linear/angular velocity reference tracker.

$$
\mathbf{x} = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} V_l \\ V_r \end{bmatrix}
$$

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} \frac{1}{m}(C_1 + C_3) & \frac{r_b}{m}(-C_1 + C_3) \\ \frac{r_b}{J}(C_1 - C_3) & \frac{r_b^2}{J}(-C_1 - C_3) \end{bmatrix} & \mathbf{B} &= \begin{bmatrix} \frac{1}{m}C_2 & \frac{1}{m}C_4 \\ \frac{r_b}{J}C_2 & -\frac{r_b}{J}C_4 \end{bmatrix} \\
\mathbf{C} &= \begin{bmatrix} 1 & -r_b \\ 1 & r_b \end{bmatrix} & \mathbf{D} &= \mathbf{0}_{2\times 2}
\end{aligned}
\tag{12.12}
$$

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

See `https://github.com/calcmogul/state-space-guide/blob/master/code/ramsete_coupled.py` for an implementation.

Reference tracker comparison

Figures 12.1 and 12.2 shows how well each reference tracker performs.

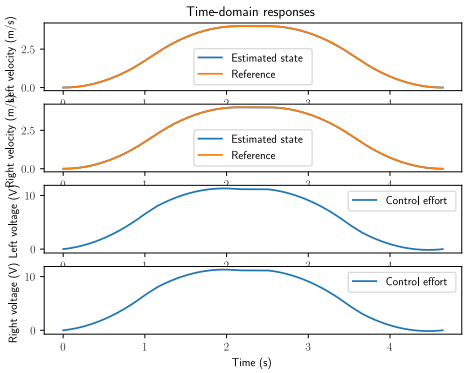


Figure 12.1: Left/right velocity reference tracker response to a motion profile

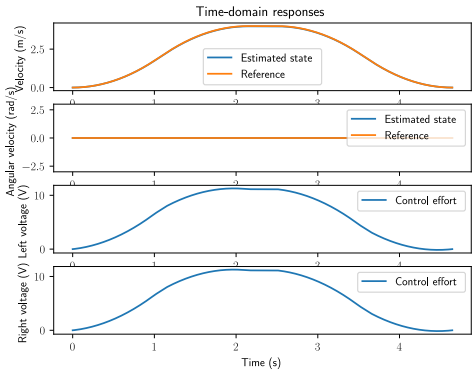


Figure 12.2: Linear/angular velocity reference tracker response to a motion profile

For a simple s-curve motion profile, they behave similarly.

Figure 12.3 demonstrates the Ramsete controller with the left/right velocity reference tracker for a typical FRC drivetrain with the reference tracking behavior shown in figure 12.4 and figure 12.5

demonstrates the Ramsete controller with the velocity / angular velocity reference tracker for a typical FRC drivetrain with the reference tracking behavior shown in figure 12.6.

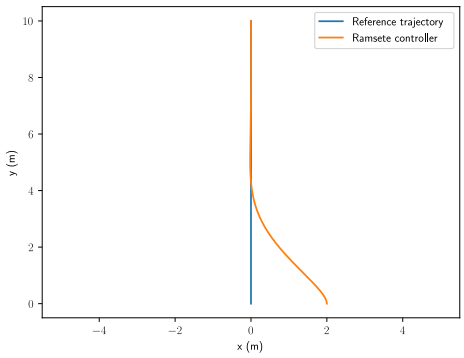


Figure 12.3: Ramsete controller response with left/right velocity reference tracker ($b = 2$, $\zeta = 0.7$)

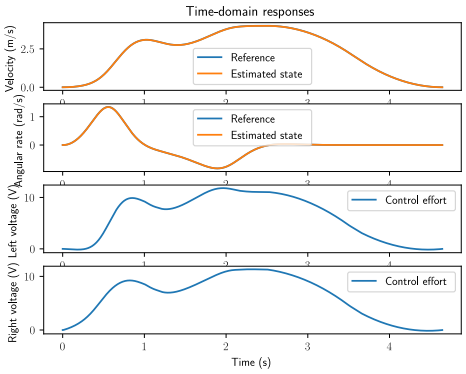


Figure 12.4: Ramsete controller's left/right velocity reference tracker response

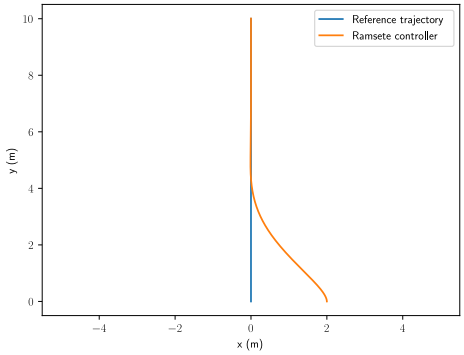


Figure 12.5: Ramsete controller response with velocity / angular velocity reference tracker ($b = 2$, $\zeta = 0.7$)

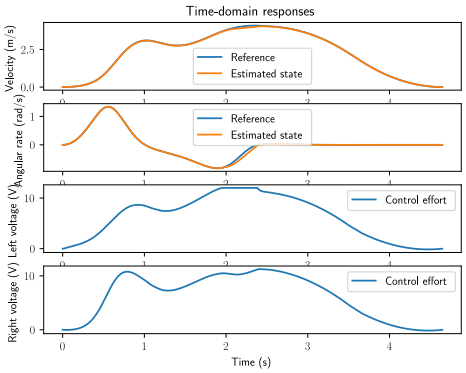


Figure 12.6: Ramsete controller's velocity / angular velocity reference tracker response

The Ramsete controller behaves relatively the same in each case, but the left/right velocity reference tracker tracks the references produced by the Ramsete controller better and with smaller control efforts overall. This is likely due to the second controller having coupled dynamics. That is, the control inputs don't act independently on the system states. In the coupled controller, v and ω require opposing control actions to reach their respective references, which results in them fighting each other. This hypothesis is supported by the condition number of the coupled controller's controllability matrix being larger (2.692 for coupled and 1.314 for decoupled).

Therefore, theorem 12.4.3 is a superior reference tracker to theorem 12.4.4 due to its decoupled dynamics, even though conversions are required between it and the Ramsete controller's commands.

12.5 Further reading

For learning more about nonlinear control, we recommend reading the book *Applied Nonlinear Control* by Jean-Jacques Slotine. For a more complex type of nonlinear control, read *A guiding vector field algorithm for path following control of nonholonomic mobile robots* [16].

IV Estimation and localization

This page intentionally left blank

Road next to Stevenson Academic building at UCSC

13. State-space observers

State-space 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 error. The Luenberger observer and Kalman filter are examples of these.

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.

13.1 Luenberger observer

[1] An agent is a system-agnostic term for independent controlled actors like robots or aircraft.

Theorem 13.1.1 — Luenberger observer.

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \hat{\mathbf{y}}) \tag{13.1}$$
$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\mathbf{u} \tag{13.2}$$

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{13.3}$$
$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k \tag{13.4}$$

$\mathbf{A}$	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 13.1: 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. Large values of $\mathbf{L}$ trust the measurements more while small values trust 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 13.1.2 instead.

Theorem 13.1.2 — Luenberger observer with separate predict/update.

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k \tag{13.5}$$

Update step

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{13.6}$$

$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_{k}^{-} \tag{13.7}$$

See appendix D.4.1 for a derivation.

13.1.1 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 (13.4) into equation (13.3) 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 equation (9.4) 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 $E_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$ 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 equation (9.3) gives

$$\begin{aligned}
\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - (\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}\mathbf{E}_k) \\
\mathbf{E}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - (\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\mathbf{C}\mathbf{E}_k) \\
\mathbf{E}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k - \mathbf{A}\hat{\mathbf{x}}_k - \mathbf{B}\mathbf{u}_k - \mathbf{L}\mathbf{C}\mathbf{E}_k \\
\mathbf{E}_{k+1} &= \mathbf{A}\mathbf{x}_k - \mathbf{A}\hat{\mathbf{x}}_k - \mathbf{L}\mathbf{C}\mathbf{E}_k \\
\mathbf{E}_{k+1} &= \mathbf{A}(\mathbf{x}_k - \hat{\mathbf{x}}_k) - \mathbf{L}\mathbf{C}\mathbf{E}_k \\
\mathbf{E}_{k+1} &= \mathbf{A}\mathbf{E}_k - \mathbf{L}\mathbf{C}\mathbf{E}_k \\
\mathbf{E}_{k+1} &= (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{E}_k
\end{aligned} \tag{13.8}$$

For equation (13.8) 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.

As stated before, the controller and estimator are dual problems. Controller gains can be found assuming perfect estimator (i.e., perfect knowledge of all states). Estimator gains can be found assuming an accurate model and a controller with perfect tracking.

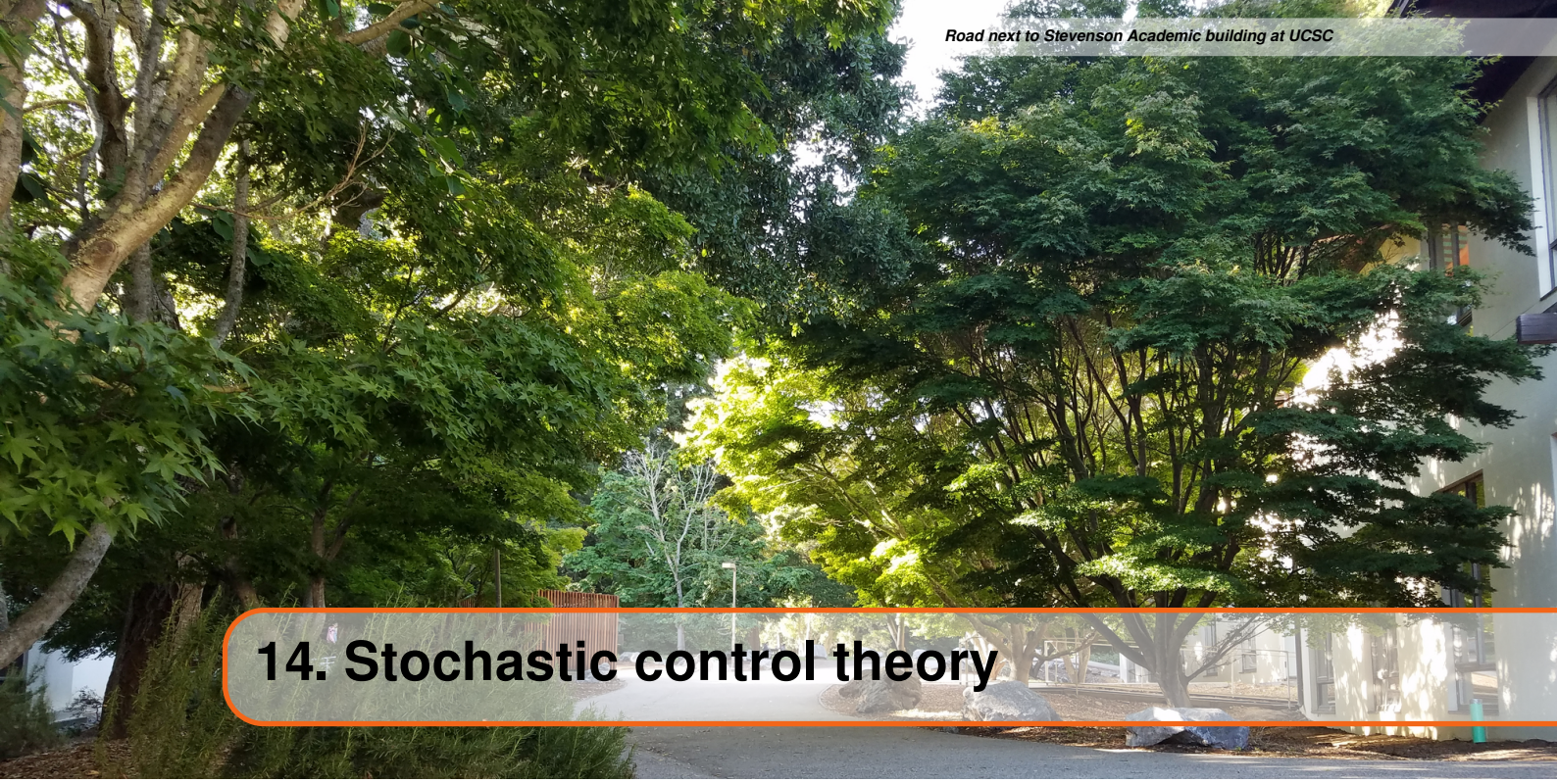
The effect of noise can be seen if it is modeled stochastically as

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}((\mathbf{y}_k + \nu_k) - \hat{\mathbf{y}}_k)$$

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.



Road next to Stevenson Academic building at UCSC

14. Stochastic control theory

Stochastic control theory is a subfield of control theory that deals with the existence of uncertainty either in observations or in the 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 noise.

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.

14.1 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.

14.2 Introduction to probability

14.2.1 Random variables

A random variable is a variable whose values are the outcomes of a random phenomenon. 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 whose value at any given sample in the sample space is the probability of the value of the random variable equaling that sample. The area under the

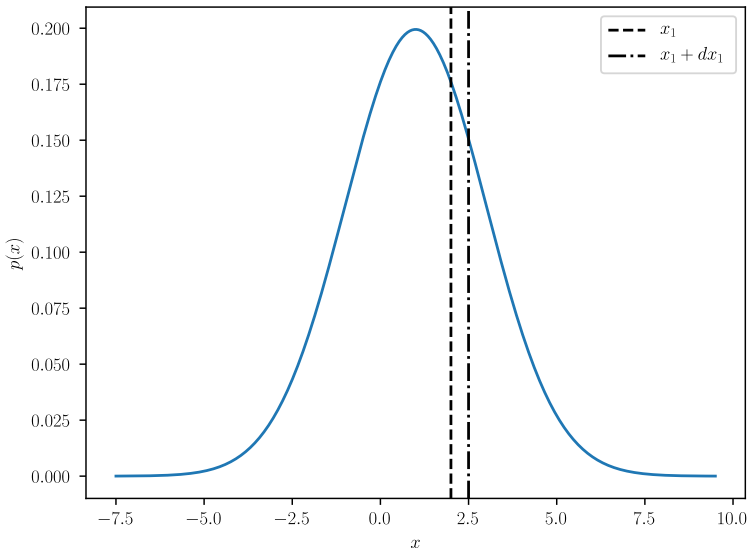


Figure 14.1: Probability density function

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 the value of x will be in the interval $x \in [x_1, x_1 + dx]$ is $p(x_1)\,dx_1$ (see figure 14.1).

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.

$$p(x) \geq 0, \int_{-\infty}^{\infty} p(x)\,dx = 1$$

14.2.2 Expected value

Expected value or expectation is a weighted average of the values the PDF can produce where the weight for each is the corresponding 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., $\overline{x}$). The expectation of the difference between a random variable and its mean converges to zero. In other words, the expectation of a random variable is its mean.

$$
\begin{aligned}
E[x - \overline{x}] &= \int_{-\infty}^{\infty} (x - \overline{x})\, p(x)\, dx \\
E[x - \overline{x}] &= \int_{-\infty}^{\infty} x\, p(x)\, dx - \int_{-\infty}^{\infty} \overline{x}\, p(x)\, dx \\
E[x - \overline{x}] &= \int_{-\infty}^{\infty} x\, p(x)\, dx - \overline{x} \int_{-\infty}^{\infty} p(x)\, dx \\
E[x - \overline{x}] &= \overline{x} - \overline{x} \cdot 1 \\
E[x - \overline{x}] &= 0
\end{aligned}
$$

14.2.3 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. The standard deviation is the square root of the variance.

$$
\begin{aligned}
var(x) = \sigma^2 = E[(x - \overline{x})^2] &= \int_{-\infty}^{\infty} (x - \overline{x})^2\, p(x)\, dx \\
std[x] = \sigma &= \sqrt{var(x)}
\end{aligned}
$$

14.2.4 Joint probability density functions

Probability density functions can also include more than one variable. Let x and y are 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 $x \in [x, x + dx], y \in [y, y + dy]$ (see figure 14.2 for an example of a joint PDF).

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.

$$
\begin{aligned}
E[x] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x\, dx\, dy \\
E[y] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} y\, dx\, dy \\
E[f(x, y)] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, dx\, dy
\end{aligned}
$$

The variance of a joint PDF measures how a variable correlates with itself.

$$
var(x) = \Sigma_{xx} = E[(x - \overline{x})^2] = \int_{-} \infty^{\infty} \int_{-\infty}^{\infty} (x - \overline{y})^2\, p(x, y)\, dx\, dy
$$

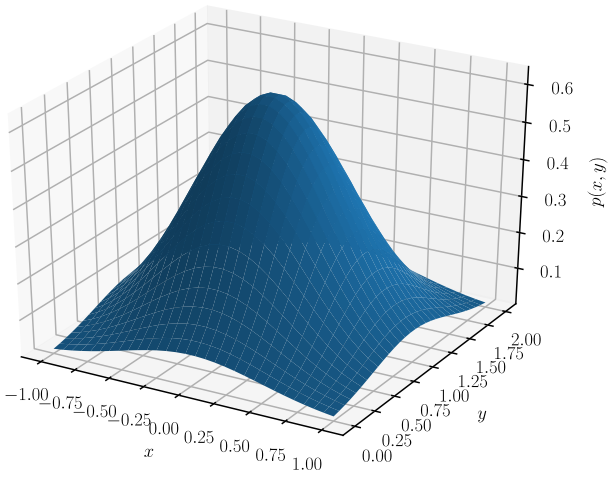


Figure 14.2: Joint probability density function

$$var(y) = \Sigma_{yy} = E[(y - \overline{y})^2] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (y - \overline{y})^2 \, p(x, y) \, dx \, dy$$

14.2.5 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.

$$cov(x, y) = \Sigma_{xy} = E[(x - \overline{x})(y - \overline{y})] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \overline{y})(y - \overline{y}) \, p(x, y) \, dx \, dy$$

14.2.6 Correlation

Correlation is defined as

$$\rho(x, y) = \frac{\Sigma_{xy}}{\sqrt{\Sigma_{xx} \Sigma_{yy}}}, |\rho(x, y)| \leq 1$$

14.2.7 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,

$$E[xy] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xy \, p(x, y) \, dx \, dy$$

$$
\begin{aligned}
E[xy] &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy\, p(x)\, p(y)\, dx\, dy \\
E[xy] &= \int_{-\infty}^{\infty} x\, p(x)\, dx \int_{-\infty}^{\infty} y\, p(y)\, dy \\
E[xy] &= E[x]E[y] \\
E[xy] &= \overline{x}\,\overline{y}
\end{aligned}
$$

$$
\begin{aligned}
cov(x, y) &= E[(x - \overline{x})(y - \overline{y})] \\
cov(x, y) &= E[(x - \overline{x})]E[(y - \overline{y})] \\
cov(x, y) &= 0 \cdot 0
\end{aligned}
$$

Therefore, the covariance Σ_{xy} is zero. Furthermore, $\rho(x, y) = 0$.

14.2.8 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.

$$
p(x) = \int_{-\infty}^{\infty} p(x, y)\, dy
$$

14.2.9 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.

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

$$
\begin{aligned}
C(y^*) &= \int_{-\infty}^{\infty} p(x, y = y^*)\, dx \\
p(x|y^*) &= \frac{1}{C(y^*)} p(x, y = y^*)
\end{aligned}
$$

The scale factor $\frac{1}{C(y^*)}$ is used to scale the area under the PDF to 1.

14.2.10 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 $p(x, y) = p(x)\, p(y)$.

14.2.11 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.

$$E[x|y] = \int_{-\infty}^{\infty} x\, p(x|y)\, dx = f(y), E[x|y] \neq E[x]$$
$$E[y|x] = \int_{-\infty}^{\infty} y\, p(y|x)\, dy = f(x), E[y|x] \neq E[y]$$

14.2.12 Conditional variances

$$var(x|y) = E[(x - E[x|y])^2|y]$$
$$var(x|y) = \int_{-\infty}^{\infty} (x - E[x|y])^2\, p(x|y)\, dx$$

14.2.13 Random vectors

Now we will extend the probability concepts discussed so far to vectors where each element has a PDF.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

The elements of $\mathbf{x}$ are scalar variables jointly distributed with a joint density $p(x_1, x_2, \dots, x_n)$. The expectation is

$$E[\mathbf{x}] = \overline{\mathbf{x}} = \int_{-\infty}^{\infty} \mathbf{x}\, p(\mathbf{x})\, d\mathbf{x}$$
$$E[\mathbf{x}] = \begin{bmatrix} E[x_1] \\ E[x_2] \\ \vdots \\ E[x_n] \end{bmatrix}$$
$$E[x_i] = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} x_i\, p(x_1, x_2, \dots, x_n)\, dx_1 \dots dx_n$$
$$E[f(\mathbf{x})] = \int_{-\infty}^{\infty} f(\mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}$$

14.2.14 Covariance matrix

The covariance matrix for a random vector $\mathbf{x} \in \mathbb{R}^n$ is

$$\mathbf{\Sigma} = cov(\mathbf{x}, \mathbf{x}) = E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T]$$

$$\mathbf{\Sigma} = \begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) & \dots & cov(x_1, x_n) \\ cov(x_2, x_1) & cov(x_1, x_2) & \dots & cov(x_1, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ cov(x_n, x_1) & cov(x_n, x_2) & \dots & cov(x_n, x_n) \end{bmatrix}$$

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 $\mathbf{v} \neq 0$, $\mathbf{v}^T \mathbf{\Sigma} \mathbf{v} \geq 0$. In other words, the eigenvalues of $\mathbf{\Sigma}$ are all greater than or equal to zero.

14.2.15 Relations for independent random vectors

First, independent vectors imply linearity from $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})\, p(\mathbf{y})$.

$$\begin{aligned} E[\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}] &= \mathbf{A}E[\mathbf{x}] + \mathbf{B}E[\mathbf{y}] \\ E[\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}] &= \mathbf{A}\overline{\mathbf{x}} + \mathbf{B}\overline{\mathbf{y}} \end{aligned}$$

Second, independent vectors being uncorrelated means their covariance is zero.

$$\begin{aligned} \mathbf{\Sigma}_{\mathbf{xy}} &= cov(\mathbf{x}, \mathbf{y}) \\ \mathbf{\Sigma}_{\mathbf{xy}} &= E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{y} - \overline{\mathbf{y}})^T] \\ \mathbf{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^T] - E[\mathbf{x}\overline{\mathbf{y}}^T] - E[\overline{\mathbf{x}}\mathbf{y}^T] + E[\overline{\mathbf{x}}\overline{\mathbf{y}}^T] \\ \mathbf{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^T] - E[\mathbf{x}]\overline{\mathbf{y}}^T - \overline{\mathbf{x}}E[\mathbf{y}^T] + \overline{\mathbf{x}}\overline{\mathbf{y}}^T \\ \mathbf{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^T] - \overline{\mathbf{x}}\overline{\mathbf{y}}^T - \overline{\mathbf{x}}\overline{\mathbf{y}}^T + \overline{\mathbf{x}}\overline{\mathbf{y}}^T \\ \mathbf{\Sigma}_{\mathbf{xy}} &= E[\mathbf{x}\mathbf{y}^T] - \overline{\mathbf{x}}\overline{\mathbf{y}}^T \end{aligned} \tag{14.1}$$

Now, compute $E[\mathbf{x}\mathbf{y}^T]$.

$$\begin{aligned} E[\mathbf{x}\mathbf{y}^T] &= \int_X \int_Y \mathbf{x}\mathbf{y}^T\, p(\mathbf{x})\, p(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y}^T \\ E[\mathbf{x}\mathbf{y}^T] &= \int_X p(\mathbf{x})\, \mathbf{x}\, d\mathbf{x} \int_Y p(\mathbf{y})\, \mathbf{y}^T\, d\mathbf{y}^T \\ E[\mathbf{x}\mathbf{y}^T] &= \overline{\mathbf{x}}\overline{\mathbf{y}}^T \end{aligned} \tag{14.2}$$

Substitute equation (14.2) into equation (14.1).

$$\begin{aligned} \mathbf{\Sigma}_{\mathbf{xy}} &= (\overline{\mathbf{x}}\overline{\mathbf{y}}^T) - \overline{\mathbf{x}}\overline{\mathbf{y}}^T \\ \mathbf{\Sigma}_{\mathbf{xy}} &= 0 \end{aligned}$$

Using these results, we can compute the covariance of $\mathbf{z} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}$ where $\Sigma_{xy} = 0$, $\Sigma_x = cov(\mathbf{x}, \mathbf{x})$, and $\Sigma_y = cov(\mathbf{y}, \mathbf{y})$.

$$
\begin{aligned}
\Sigma_z &= cov(\mathbf{z}, \mathbf{z}) \\
\Sigma_z &= E[(\mathbf{z} - \overline{\mathbf{z}})(\mathbf{z} - \overline{\mathbf{z}})^T] \\
\Sigma_z &= E[(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \mathbf{B}\overline{\mathbf{y}})(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} - \mathbf{A}\overline{\mathbf{x}} - \mathbf{B}\overline{\mathbf{y}})^T] \\
\Sigma_z &= E[(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))^T] \\
\Sigma_z &= E[(\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}}) + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}}))((\mathbf{x} - \overline{\mathbf{x}})^T\mathbf{A}^T + (\mathbf{y} - \overline{\mathbf{y}})^T\mathbf{B}^T)] \\
\Sigma_z &= E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T\mathbf{A}^T + \mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{y} - \overline{\mathbf{y}})^T\mathbf{B}^T + \\
&\quad \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{x} - \overline{\mathbf{x}})^T\mathbf{A}^T + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^T\mathbf{B}^T]
\end{aligned}
$$

Since $\mathbf{x}$ and $\mathbf{y}$ are independent,

$$
\begin{aligned}
\Sigma_z &= E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T\mathbf{A}^T + 0 + 0 + \mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^T\mathbf{B}^T] \\
\Sigma_z &= E[\mathbf{A}(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T\mathbf{A}^T] + E[\mathbf{B}(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^T\mathbf{B}^T] \\
\Sigma_z &= \mathbf{A}E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T]\mathbf{A}^T + \mathbf{B}E[(\mathbf{y} - \overline{\mathbf{y}})(\mathbf{y} - \overline{\mathbf{y}})^T]\mathbf{B}^T \\
\Sigma_z &= \mathbf{A}\Sigma_x\mathbf{A}^T + \mathbf{B}\Sigma_y\mathbf{B}^T
\end{aligned}
$$

14.2.16 Gaussian random variables

A Gaussian random variable has the following properties:

$$
\begin{aligned}
E[x] &= \overline{x} \\
var(x) &= \sigma^2 \\
p(x) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\overline{x})^2}{2\sigma^2}}
\end{aligned}
$$

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 14.2.1 — 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.

14.3 Linear stochastic systems

Given the following stochastic system

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{\Phi}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{\Gamma}\mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k\end{aligned}$$

where $\mathbf{w}_k$ is the process noise and $\mathbf{v}_k$ is the measurement noise,

$$\begin{aligned}E[\mathbf{w}_k] &= 0 \\ E[\mathbf{w}_k\mathbf{w}_k^T] &= \mathbf{Q}_k \\ E[\mathbf{v}_k] &= 0 \\ E[\mathbf{v}_k\mathbf{v}_k^T] &= \mathbf{R}_k\end{aligned}$$

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 $E[\mathbf{w}_k\mathbf{w}_j^T] = 0$ and $E[\mathbf{v}_k\mathbf{v}_k^T] = 0$ 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.

14.3.1 State vector expectation evolution

First, we will compute how the expectation of the system state evolves.

$$\begin{aligned}E[\mathbf{x}_{k+1}] &= E[\mathbf{\Phi}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{\Gamma}\mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= E[\mathbf{\Phi}\mathbf{x}_k] + E[\mathbf{B}\mathbf{u}_k] + E[\mathbf{\Gamma}\mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= \mathbf{\Phi}E[\mathbf{x}_k] + \mathbf{B}E[\mathbf{u}_k] + \mathbf{\Gamma}E[\mathbf{w}_k] \\ E[\mathbf{x}_{k+1}] &= \mathbf{\Phi}E[\mathbf{x}_k] + \mathbf{B}\mathbf{u}_k + 0 \\ \overline{\mathbf{x}}_{k+1} &= \mathbf{\Phi}\overline{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k\end{aligned}$$

14.3.2 State covariance matrix evolution

Now, we will use this to compute how the state covariance matrix $\mathbf{P}$ evolves.

$$\begin{aligned}\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1} &= \mathbf{\Phi}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{\Gamma}\mathbf{w}_k - (\mathbf{\Phi}\overline{\mathbf{x}}_k - \mathbf{B}\mathbf{u}_k) \\ \mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1} &= \mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{\Gamma}\mathbf{w}_k\end{aligned}$$

$$E[(\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1})(\mathbf{x}_{k+1} - \overline{\mathbf{x}}_{k+1})^T] = E[(\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{\Gamma}\mathbf{w}_k)(\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{\Gamma}\mathbf{w}_k)^T]$$

$$\begin{aligned}\mathbf{P}_{k+1} &= E[(\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{\Gamma}\mathbf{w}_k)(\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k) + \mathbf{\Gamma}\mathbf{w}_k)^T] \\ \mathbf{P}_{k+1} &= E[(\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k)(\mathbf{x}_k - \overline{\mathbf{x}}_k)^T\mathbf{\Phi}^T] + E[\mathbf{\Phi}(\mathbf{x}_k - \overline{\mathbf{x}}_k)\mathbf{w}_k^T\mathbf{\Gamma}^T] +\end{aligned}$$

$$
\begin{aligned}
& E[\mathbf{\Gamma}\mathbf{w}_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T\mathbf{\Phi}^T] + E[\mathbf{\Gamma}\mathbf{w}_k\mathbf{w}_k^T\mathbf{\Gamma}^T] \\
\mathbf{P}_{k+1} = & \mathbf{\Phi}E[(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]\mathbf{\Phi}^T + \mathbf{\Phi}E[(\mathbf{x}_k - \bar{\mathbf{x}}_k)\mathbf{w}_k^T]\mathbf{\Gamma}^T + \\
& \mathbf{\Gamma}E[\mathbf{w}_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]\mathbf{\Phi}^T + \mathbf{\Gamma}E[\mathbf{w}_k\mathbf{w}_k^T]\mathbf{\Gamma}^T \\
\mathbf{P}_{k+1} = & \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{\Phi}E[(\mathbf{x}_k - \bar{\mathbf{x}}_k)\mathbf{w}_k^T]\mathbf{\Gamma}^T + \\
& \mathbf{\Gamma}E[\mathbf{w}_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]\mathbf{\Phi}^T + \mathbf{\Gamma}\mathbf{Q}_k\mathbf{\Gamma}_k^T \\
\mathbf{P}_{k+1} = & \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + 0 + 0 + \mathbf{\Gamma}\mathbf{Q}_k\mathbf{\Gamma}^T \\
\mathbf{P}_{k+1} = & \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{\Gamma}\mathbf{Q}_k\mathbf{\Gamma}^T
\end{aligned}
$$

14.3.3 Measurement vector expectation

Next, we will compute the expectation of the output $\mathbf{y}$.

$$
\begin{aligned}
E[\mathbf{y}_k] &= E[\mathbf{H}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k] \\
E[\mathbf{y}_k] &= \mathbf{H}E[\mathbf{x}_k] + \mathbf{D}\mathbf{u}_k + 0 \\
\bar{\mathbf{y}}_k &= \mathbf{H}\bar{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k
\end{aligned}
$$

14.3.4 Measurement covariance matrix

Now, we will use this to compute how the measurement covariance matrix $\mathbf{S}$ evolves.

$$
\begin{aligned}
\mathbf{y}_k - \bar{\mathbf{y}}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k - (\mathbf{H}\bar{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k) \\
\mathbf{y}_k - \bar{\mathbf{y}}_k &= \mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k) + \mathbf{v}_k
\end{aligned}
$$

$$
\begin{aligned}
E[(\mathbf{y}_k - \bar{\mathbf{y}}_k)(\mathbf{y}_k - \bar{\mathbf{y}}_k)^T] &= E[(\mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k) + \mathbf{v}_k)(\mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k) + \mathbf{v}_k)^T] \\
\mathbf{S}_k &= E[(\mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k) + \mathbf{v}_k)(\mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k) + \mathbf{v}_k)^T] \\
\mathbf{S}_k &= E[(\mathbf{H}(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T\mathbf{H}^T] + E[\mathbf{v}_k\mathbf{v}_k^T] \\
\mathbf{S}_k &= \mathbf{H}E[((\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]\mathbf{H}^T + \mathbf{R}_k \\
\mathbf{S}_k &= \mathbf{H}\mathbf{P}_k\mathbf{H}^T + \mathbf{R}_k
\end{aligned}
$$

14.4 Two-sensor problem

We'll skip the probability derivations here, but given two data points with associated variances represented by Gaussian distribution, the information can be optimally combined into a third Gaussian distribution with a mean value and variance. The expected value of x given a measurement z_1 is

$$
E[x|z_1] = \mu = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2}z_1 + \frac{\sigma^2}{\sigma_0^2 + \sigma^2}x_0 \tag{14.3}
$$

The variance of x given z_1 is

$$
E[(x - \mu)^2|z_1] = \frac{\sigma^2\sigma_0^2}{\sigma_0^2 + \sigma^2} \tag{14.4}
$$

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.

$$\hat{x} = E[x|z_1] = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} z_1 + \frac{\sigma^2}{\sigma_0^2 + \sigma^2} x_0 \tag{14.5}$$
$$\hat{x} = w_1 z_1 + w_2 x_0$$

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)

$$w_1 = \lim_{\sigma_0^2 \to 0} \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} = 0 \tag{14.6}$$
$$w_2 = \lim_{\sigma_0^2 \to 0} \frac{\sigma^2}{\sigma_0^2 + \sigma^2} = 1 \tag{14.7}$$

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

$$w_1 = \lim_{\sigma_0^2 \to 0} \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} = 1 \tag{14.8}$$
$$w_2 = \lim_{\sigma_0^2 \to 0} \frac{\sigma^2}{\sigma_0^2 + \sigma^2} = 0 \tag{14.9}$$

The Kalman filter uses this optimal fusion as the basis for its operation.

14.5 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.

14.5.1 Derivations

Given the posteriori update equation $\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{H}_{k+1}\hat{\mathbf{x}}_{k+1}^{-})$, we want to find the value of $\mathbf{K}$ that minimizes the error covariance, because doing this minimizes the estimation error.

a posteriori estimate covariance matrix

$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{+})$$
$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - (\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1}^{-})))$$
$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - (\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{H}_{k+1}\mathbf{x}_{k+1} + \mathbf{v}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1}^{-})))$$
$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - (\hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{x}_{k+1} + \mathbf{K}_{k+1}\mathbf{v}_{k+1} - \mathbf{K}_{k+1}\mathbf{H}\hat{\mathbf{x}}_{k+1}^{-}))$$

$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{x}_{k+1} - \mathbf{K}_{k+1}\mathbf{v}_{k+1} + \mathbf{K}_{k+1}\mathbf{H}\hat{\mathbf{x}}_{k+1}^{-})$$
$$\mathbf{P}_{k+1}^{+} = cov(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})$$
$$\mathbf{P}_{k+1}^{+} = cov((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})$$
$$\mathbf{P}_{k+1}^{+} = cov((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})) + cov(\mathbf{K}_{k+1}\mathbf{v}_{k+1})$$
$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})cov(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})^{T} + \mathbf{K}_{k+1}cov(\mathbf{v}_{k+1})\mathbf{K}_{k+1}^{T}$$
$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})^{T} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}$$

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. This is equivalent to minimizing the trace of the a posteriori estimate covariance matrix $\mathbf{P}_{k+1}^{-}$.

First, we'll expand the equation for $\mathbf{P}_{k+1}^{-}$ and collect terms.

$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})^{T} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}$$
$$\mathbf{P}_{k+1}^{+} = (\mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-})(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})^{T} + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}$$
$$\mathbf{P}_{k+1}^{+} = (\mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-})(\mathbf{I}^{T} - \mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T}) + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}$$
$$\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-}(\mathbf{I}^{T} - \mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T}) - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}(\mathbf{I}^{T} - \mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T}) + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}$$
$$\begin{aligned}\mathbf{P}_{k+1}^{+} = {} & \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} + \\ & \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^{T}\end{aligned}$$
$$\begin{aligned}\mathbf{P}_{k+1}^{+} = {} & \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} + \\ & \mathbf{K}_{k+1}(\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T} + \mathbf{R}_{k+1})\mathbf{K}_{k+1}^{T}\end{aligned}$$
$$\mathbf{P}_{k+1}^{+} = \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T} \tag{14.10}$$

Now we'll take the trace.

$$\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T})$$

Transpose one of the terms twice.

$$\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}((\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-T})^{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T})$$

$\mathbf{P}_{k+1}^{-}$ is symmetric, so we can drop the transpose.

$$\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}((\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-})^{T}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T})$$

The trace of a matrix is equal to the trace of its transpose since the elements used in the trace are on the diagonal.

$$\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) - \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T})$$
$$\operatorname{tr}(\mathbf{P}_{k+1}^{+}) = \operatorname{tr}(\mathbf{P}_{k+1}^{-}) - 2\operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-}) + \operatorname{tr}(\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T})$$

Given theorems 14.5.1 and 14.5.2

Theorem 14.5.1 $\frac{\partial}{\partial \mathbf{A}} \operatorname{tr}(\mathbf{A}\mathbf{B}\mathbf{A}^{T}) = 2\mathbf{A}\mathbf{B}$ where $\mathbf{B}$ is symmetric.

Theorem 14.5.2 $\frac{\partial}{\partial \mathbf{A}} \operatorname{tr}(\mathbf{A}\mathbf{C}) = \mathbf{C}^{T}$

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 $\mathbf{0}$.

$$\begin{aligned}
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^{+})}{\partial \mathbf{K}} &= \mathbf{0} - 2(\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-})^{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^{+})}{\partial \mathbf{K}} &= -2\mathbf{P}_{k+1}^{-T}\mathbf{H}_{k+1}^{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1}^{+})}{\partial \mathbf{K}} &= -2\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
\mathbf{0} &= -2\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T} + 2\mathbf{K}_{k+1}\mathbf{S}_{k+1} \\
2\mathbf{K}_{k+1}\mathbf{S}_{k+1} &= 2\mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T} \\
\mathbf{K}_{k+1}\mathbf{S}_{k+1} &= \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T} \\
\mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{S}_{k+1}^{-1}
\end{aligned}$$

This is the optimal Kalman gain.

Simplified *a priori* estimate covariance matrix

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.

$$\begin{aligned}
\mathbf{K}_{k+1} &= \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{S}_{k+1}^{-1} \\
\mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T} &= \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T}
\end{aligned}$$

Now we'll substitute it into equation (14.10).

$$\begin{aligned}
\mathbf{P}_{k+1}^{+} &= \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{S}_{k+1}\mathbf{K}_{k+1}^{T} \\
\mathbf{P}_{k+1}^{+} &= \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} + \mathbf{P}_{k+1}^{-}\mathbf{H}_{k+1}^{T}\mathbf{K}_{k+1}^{T} \\
\mathbf{P}_{k+1}^{+} &= \mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1}^{-} \\
\mathbf{P}_{k+1}^{+} &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}_{k+1}^{-}
\end{aligned}$$

14.5.2 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 14.5.3 shows the predict and update steps for a Kalman filter at the k^{th} timestep.

Theorem 14.5.3 — Kalman filter.

Predict step

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{\Phi}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k \tag{14.11}$$

$$\mathbf{P}_{k+1}^{-} = \mathbf{\Phi}\mathbf{P}_k^{-}\mathbf{\Phi}^T + \mathbf{\Gamma}\mathbf{Q}\mathbf{\Gamma}^T \tag{14.12}$$

Update step

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1}^{-}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1}^{-}\mathbf{H}^T + \mathbf{R})^{-1} \tag{14.13}$$

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1}^{-}) \tag{14.14}$$

$$\mathbf{P}_{k+1}^{+} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1}^{-} \tag{14.15}$$

$\mathbf{\Phi}$	system matrix	$\hat{\mathbf{x}}$	state estimate vector
$\mathbf{B}$	input matrix	$\mathbf{u}$	input vector
$\mathbf{H}$	measurement matrix	$\mathbf{y}$	output vector
$\mathbf{P}$	error covariance matrix	$\mathbf{Q}$	process noise covariance matrix
$\mathbf{K}$	Kalman gain matrix	$\mathbf{R}$	measurement noise covariance matrix
$\mathbf{\Gamma}$	process noise intensity vector		

where a superscript of minus denotes *a priori* and plus denotes *a posteriori* estimate (before and after update respectively).

$\mathbf{H}$, $\mathbf{Q}$, and $\mathbf{R}$ from the equations derived earlier are made constants here. $\mathbf{\Phi}$ is replaced with $\mathbf{A}$ for continuous systems.

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 10.7.1.

Matrix	Rows × Columns	Matrix	Rows × Columns
$\mathbf{\Phi}$	states × states	$\hat{\mathbf{x}}$	states × 1
$\mathbf{B}$	states × inputs	$\mathbf{u}$	inputs × 1
$\mathbf{H}$	outputs × states	$\mathbf{y}$	outputs × 1
$\mathbf{P}$	states × states	$\mathbf{Q}$	states × states
$\mathbf{K}$	states × outputs	$\mathbf{R}$	outputs × outputs
$\mathbf{\Gamma}$	states × 1		

Table 14.1: Kalman filter matrix dimensions

Unknown states in a Kalman filter are generally represented by a Wiener (pronounced VEE-ner)

process[1]. This process has the property that its variance increases linearly with time t.

14.5.3 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 $0.8cm/s$ 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.

$$x_{k+1} = x_k + v_k \Delta T \tag{14.16}$$
$$v_{k+1} = v_k \tag{14.17}$$
$$x^w_{k+1} = x^w_k \tag{14.18}$$

This can be converted to the following state-space model.

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ v_k \\ x^w_k \end{bmatrix} \tag{14.19}$$

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1 \\ 0 \end{bmatrix} w_k \tag{14.20}$$

where the Gaussian random variable w_k has a mean of 0 and a variance of 1. The observation model is

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \theta_k \tag{14.21}$$

where the covariance matrix of Gaussian measurement noise θ is a 2×2 matrix with both diagonals $10cm^2$.

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 $k = 2$.

$$\mathbf{Q} = \begin{bmatrix} 1 \end{bmatrix} \tag{14.22}$$
$$\mathbf{R} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{14.23}$$
$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{y}_{k,1} \\ (\mathbf{y}_{k,1} - \mathbf{y}_{k-1,1})/dt \\ \mathbf{y}_{k,1} + \mathbf{y}_{k,2} \end{bmatrix} \tag{14.24}$$

[1] 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.

$$\mathbf{P} = \begin{bmatrix} 10 & 10/dt & 10 \\ 10/dt & 20/dt^2 & 10/dt \\ 10 & 10/dt & 20 \end{bmatrix} \tag{14.25}$$

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 guarenteed 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 $\widetilde{\mathbf{y}} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}$

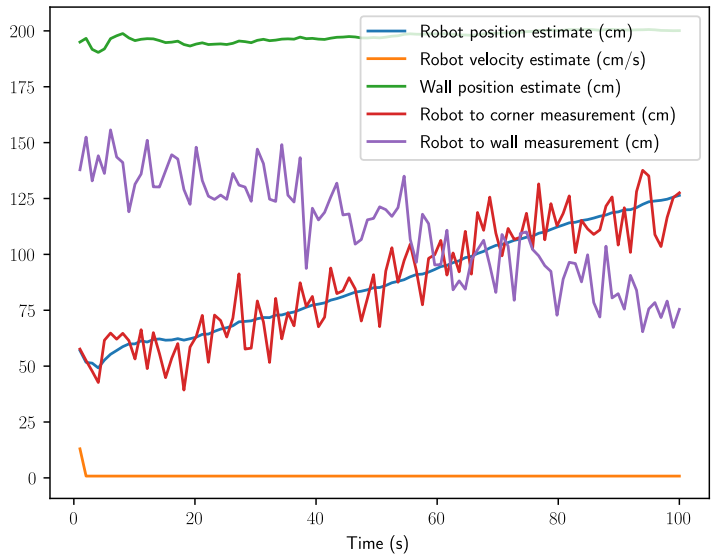


Figure 14.3: State estimates and measurements with Kalman filter

over time will be something other than white noise.

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.

14.5.4 Simulation

Figure 14.3 shows the state estimates and measurements of the Kalman filter over time. Figure 14.4 shows the position estimate and variance over time. Figure 14.5 shows the 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.

14.5.5 Kalman filter as Luenberger observer

A Kalman filter can be represented as a Luenberger observer by letting $\mathbf{C} = \mathbf{H}$ and $\mathbf{L} = \mathbf{A}\mathbf{K}_k$ (see appendix D.4). 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

$$\text{eig}(\mathbf{A}(\mathbf{I} - \mathbf{K}_k\mathbf{H})) \tag{14.26}$$

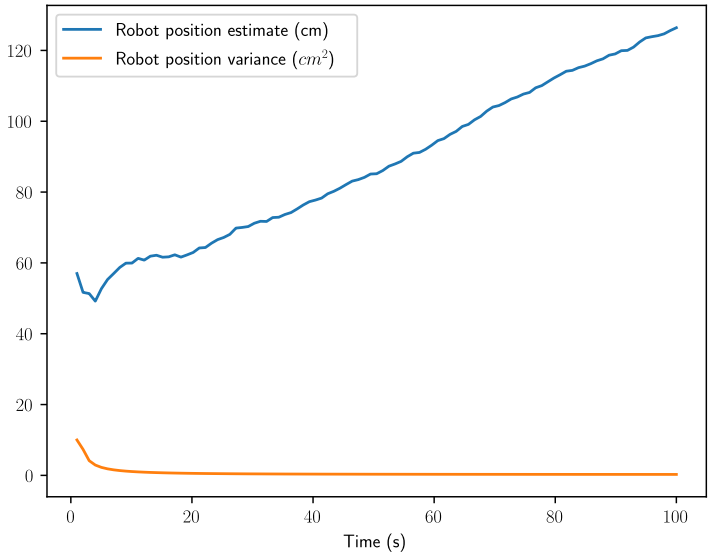


Figure 14.4: Robot position estimate and variance with Kalman filter

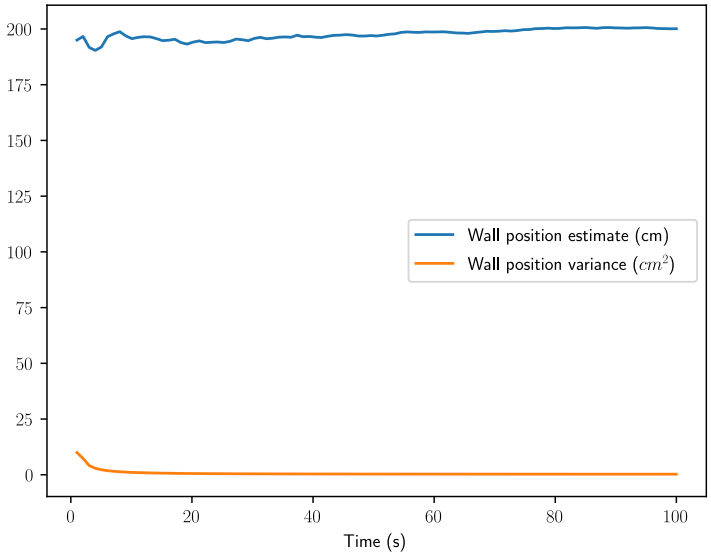


Figure 14.5: Wall position estimate and variance with Kalman filter

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 Ricatti equation. This can then be used to compute a steady-state Kalman gain.

Snippet 14.1 computes the steady-state matrices for a Kalman filter.

```
import control as cnt
import numpy as np
import scipy as sp


def kalmd(sys, Q, R):
    """Solves for the steady state kalman gain and error covariance matrices.

    Keyword arguments:
    sys -- discrete state-space model
    Q -- process noise covariance matrix
    R -- measurement noise covariance matrix

    Returns:
    Kalman gain, error covariance matrix.
    """
    m = sys.A.shape[0]

    observability_rank = np.linalg.matrix_rank(cnt.obsv(sys.A, sys.C))
    if observability_rank != m:
        print(
            "Warning: Observability of %d != %d, unobservable state"
            % (observability_rank, m)
        )

    # Compute the steady state covariance matrix
    P_prior = sp.linalg.solve_discrete_are(a=sys.A.T, b=sys.C.T, q=Q, r=R)
    S = sys.C * P_prior * sys.C.T + R
    K = P_prior * sys.C.T * np.linalg.inv(S)
    P = (np.eye(m) - K * sys.C) * P_prior

    return K, P
```

Snippet 14.1. Steady-state Kalman gain and error covariance matrices calculation in Python

14.6 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.

We will modify the robot model so that instead of a velocity of $0.8cm/s$ with random noise, the velocity is modeled as a random walk from the current velocity.

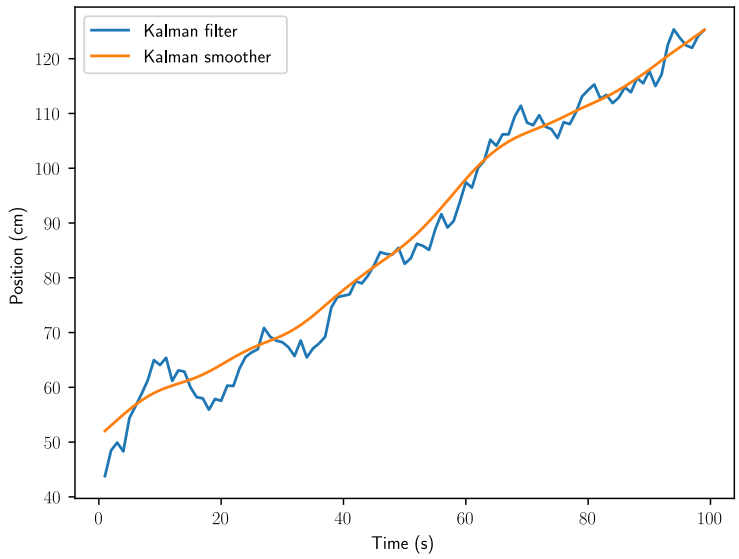


Figure 14.6: Robot position with Kalman smoother

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ v_k \\ x_k^w \end{bmatrix} \tag{14.27}$$

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0.1 \\ 0 \end{bmatrix} w_k \tag{14.28}$$

We will use the same observation model as before.

Using the same data from subsection 14.5.4, figures 14.6, 14.7, and 14.8 show the improved state estimates and figure 14.9 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.

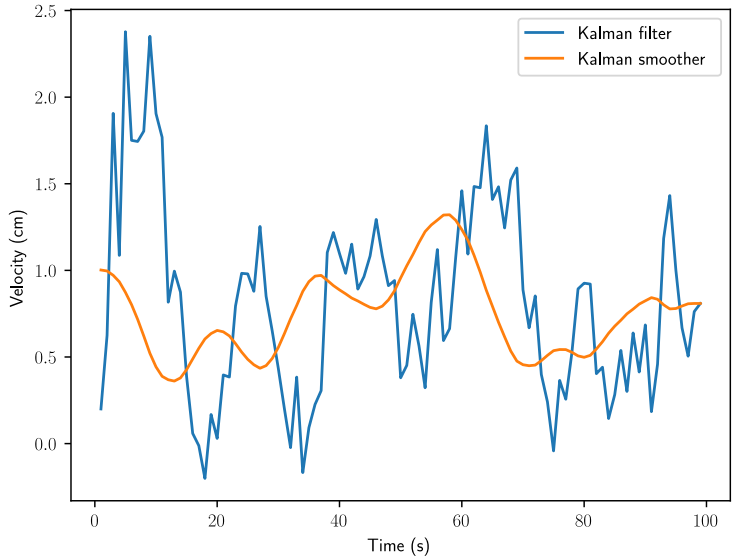


Figure 14.7: Robot velocity with Kalman smoother

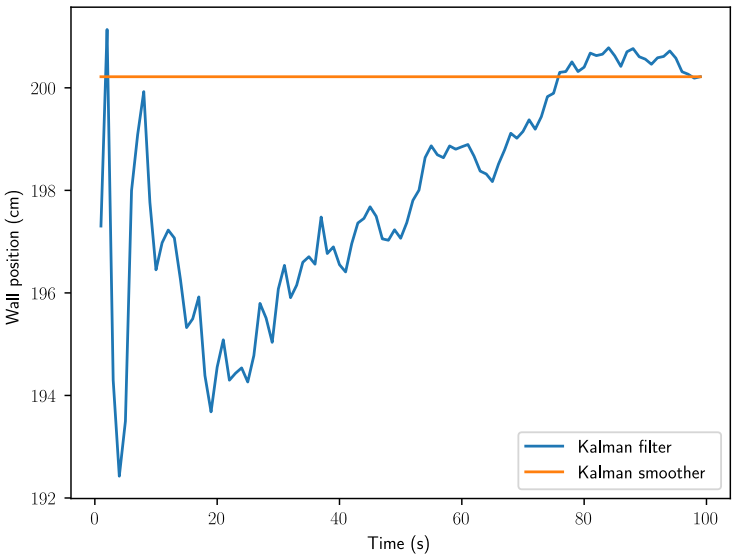


Figure 14.8: Wall position with Kalman smoother

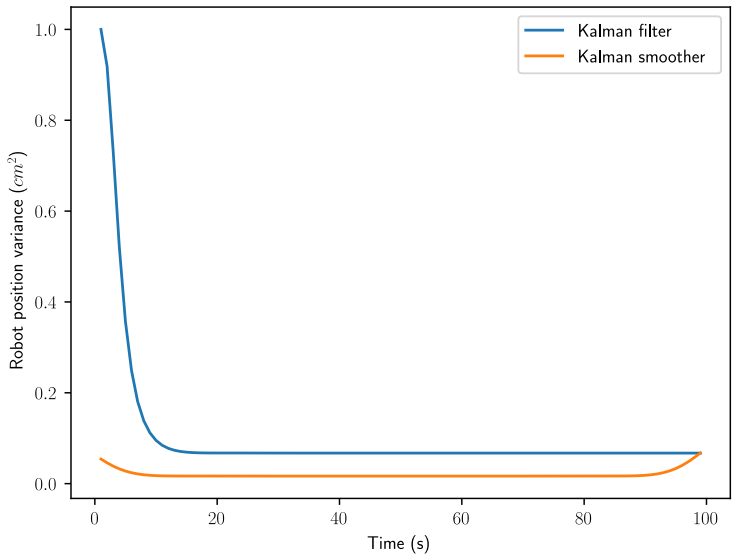


Figure 14.9: Robot position variance with Kalman smoother

14.7 MMAE

MMAE stands for 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.

14.8 Nonlinear observers

In this book, we have covered the Kalman filter, which is the optimal unbiased estimator for linear systems. It isn't optimal for nonlinear systems, but several extensions to it have been developed to make it more accurate.

14.8.1 Extended Kalman filter

This method linearizes the matrices used during the prediction step. In addition to the $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ matrices above, the process noise intensity vector $\mathbf{\Gamma}$ is linearized as follows:

$$\mathbf{\Gamma} = \frac{\partial f(\mathbf{x}, \mathbf{0}, \mathbf{0})}{\partial \mathbf{w}}$$

where $\mathbf{w}$ is the process noise included in the stochastic model.

From there, the continuous Kalman filter equations are used like normal to compute the error covariance matrix $\mathbf{P}$ and Kalman gain matrix. The state estimate update can still use the function $h(\mathbf{x})$ for accuracy.

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - h(\hat{\mathbf{x}}_{k+1}^{-}))$$

14.8.2 Unscented Kalman filter

This method linearizes around carefully chosen points to minimize the modeling error. There's a lot of detail to cover, so we recommend just reading a paper on it [22].

Here's a paper on a quaternion-based Unscented Kalman filter for orientation tracking [17].

This page intentionally left blank

15. Pose estimation

15.1 Nonlinear pose estimation

The simplest way to perform pose estimation is to integrate the velocity in each orthogonal direction over time. In two dimensions, one could use

$$x_{k+1} = x_k + v_k \cos\theta_k \, T$$
$$y_{k+1} = y_k + v_k \sin\theta_k \, T$$

where T is the sample period. x_d, y_d, and θ_d are obtained from a desired time-based trajectory.

This odometry approach assumes that the robot follows a straight path between samples (that is, $\omega = 0$). We can obtain a more accurate approximation 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. These values in a column vector are called a "twist".

R 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

$$
\begin{aligned}
{}^{R}dx &= {}^{R}v_x \, dt \\
{}^{R}dy &= {}^{R}v_y \, dt \\
{}^{R}d\theta &= {}^{R}\omega \, dt
\end{aligned}
$$

To transform this into the global frame, 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 2.2.1.

$$
\begin{aligned}
{}^{G}\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} &= \begin{bmatrix} \frac{\sin\omega t}{\omega} & \frac{\cos\omega t}{\omega} & 0 \\ -\frac{\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0 \\ 0 & 0 & t \end{bmatrix} {}^{R}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \Bigg|_0^t \\
{}^{G}\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} &= \begin{bmatrix} \frac{\sin\omega t}{\omega} & \frac{\cos\omega t - 1}{\omega} & 0 \\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0 \\ 0 & 0 & t \end{bmatrix} {}^{R}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}
\end{aligned}
$$

This equation assumes a starting orientation of $\theta = 0$. For nonzero starting orientations, we can apply a counterclockwise rotation by θ.

$$
{}^{G}\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sin\omega t}{\omega} & \frac{\cos\omega t - 1}{\omega} & 0 \\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0 \\ 0 & 0 & t \end{bmatrix} {}^{R}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}
$$

If we factor out a t, we can use change in pose between updates instead of velocities.

$$
\begin{aligned}
{}^{G}\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sin\omega t}{\omega} & \frac{\cos\omega t - 1}{\omega} & 0 \\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0 \\ 0 & 0 & t \end{bmatrix} {}^{R}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \\
{}^{G}\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sin\omega t}{\omega t} & \frac{\cos\omega t - 1}{\omega t} & 0 \\ \frac{1-\cos\omega t}{\omega t} & \frac{\sin\omega t}{\omega t} & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{R}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} t
\end{aligned}
$$

$$
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{\sin\omega t}{\omega t} & \frac{\cos\omega t-1}{\omega t} & 0\\ \frac{1-\cos\omega t}{\omega t} & \frac{\sin\omega t}{\omega t} & 0\\ 0 & 0 & 1\end{bmatrix}{}^{R}\begin{bmatrix}v_x t\\ v_y t\\ \omega t\end{bmatrix}
$$

$$
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}{}^{R}\begin{bmatrix}\frac{\sin\Delta\theta}{\Delta\theta} & \frac{\cos\Delta\theta-1}{\Delta\theta} & 0\\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \frac{\sin\Delta\theta}{\Delta\theta} & 0\\ 0 & 0 & 1\end{bmatrix}{}^{R}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix}
$$

The previous version used the current velocity and projected the model forward to the next timestep (into the future). As such, the prediction must be done after any controller calculations are performed. With the second version, the locally measured pose deltas can only be measured using past samples, so the model update must be performed before any controller calculations to advance the model to the current timestep.

When the robot is traveling on a straight trajectory ($\omega = 0$), some expressions within the equation above are indeterminate. We can approximate these with Taylor series expansions.

$$
\begin{aligned}
\frac{\sin\omega t}{\omega} &= t - \frac{t^3\omega^2}{6} + \dots\\
\frac{\sin\omega t}{\omega} &\sim t - \frac{t^3\omega^2}{6}\\
\frac{\cos\omega t - 1}{\omega} &= -\frac{t^2\omega}{2} + \frac{t^4\omega^3}{4} - \dots\\
\frac{\cos\omega t - 1}{\omega} &\sim -\frac{t^2\omega}{2}\\
\frac{1-\cos\omega t}{\omega} &= \frac{t^2\omega}{2} - \frac{t^4\omega^3}{4} + \dots\\
\frac{1-\cos\omega t}{\omega} &\sim \frac{t^2\omega}{2}
\end{aligned}
$$

If we let $\omega = 0$, we should get the standard kinematic equations like $x = vt$ with a rotation applied to them.

$$
\begin{aligned}
\frac{\sin\omega t}{\omega} &\sim t - \frac{t^3\cdot 0^2}{6} = t\\
\frac{\cos\omega t - 1}{\omega} &\sim -\frac{t^2\cdot 0}{2} = 0\\
\frac{1-\cos\omega t}{\omega} &\sim \frac{t^2\cdot 0}{2} = 0
\end{aligned}
$$

Now substitute these into equation (15.1).

$$
{}^{G}\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}t & 0 & 0\\ 0 & t & 0\\ 0 & 0 & t\end{bmatrix}{}^{R}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix}
$$

$$^G\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}v_x t\\ v_y t\\ \omega t\end{bmatrix}$$

As expected, the equations simplify to the first order case with a rotation matrix applied to the velocities in the robot's local coordinate frame.

Differential drive robots have $v_y = 0$ since they only move in the direction of the current heading, which is along the x-axis. Therefore,

$$^G\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}v_x t\\ 0\\ \omega t\end{bmatrix}$$

$$^G\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}v_x t\cos\theta\\ v_y t\sin\theta\\ \omega t\end{bmatrix}$$

Theorem 15.1.1 — Nonlinear pose estimator with constant curvature.

$$^G\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{\sin\omega t}{\omega} & \frac{\cos\omega t-1}{\omega} & 0\\ \frac{1-\cos\omega t}{\omega} & \frac{\sin\omega t}{\omega} & 0\\ 0 & 0 & t\end{bmatrix}^R\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix} \tag{15.1}$$

or

$$^G\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}^R\begin{bmatrix}\frac{\sin\Delta\theta}{\Delta\theta} & \frac{\cos\Delta\theta-1}{\Delta\theta} & 0\\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \frac{\sin\Delta\theta}{\Delta\theta} & 0\\ 0 & 0 & 1\end{bmatrix}^R\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta\theta\end{bmatrix} \tag{15.2}$$

where G denotes global coordinate frame and R denotes robot's coordinate frame.

For sufficiently small ω:

$$\frac{\sin\omega t}{\omega} = t - \frac{t^3\omega^2}{6} \qquad \frac{\cos\omega t-1}{\omega} = -\frac{t^2\omega}{2} \qquad \frac{1-\cos\omega t}{\omega} = \frac{t^2\omega}{2} \tag{15.3}$$

$$\frac{\sin\omega t}{\omega t} = 1 - \frac{t^2\omega^2}{6} \qquad \frac{\cos\omega t-1}{\omega t} = -\frac{t\omega}{2} \qquad \frac{1-\cos\omega t}{\omega t} = \frac{t\omega}{2} \tag{15.4}$$

Δx	change in pose's x	v_x	velocity along x-axis
Δy	change in pose's y	v_y	velocity along y-axis
$\Delta\theta$	change in pose's θ	ω	angular velocity
t	Time since last pose update	θ	starting angle in global coordinate frame

This change in pose can be added directly to the previous pose estimate to update it.

Figure 15.1 shows the error in the global pose coordinates over time for the simpler odometry method compared to the method using twists (uses the Ramsete controller from subsection 12.4.1).

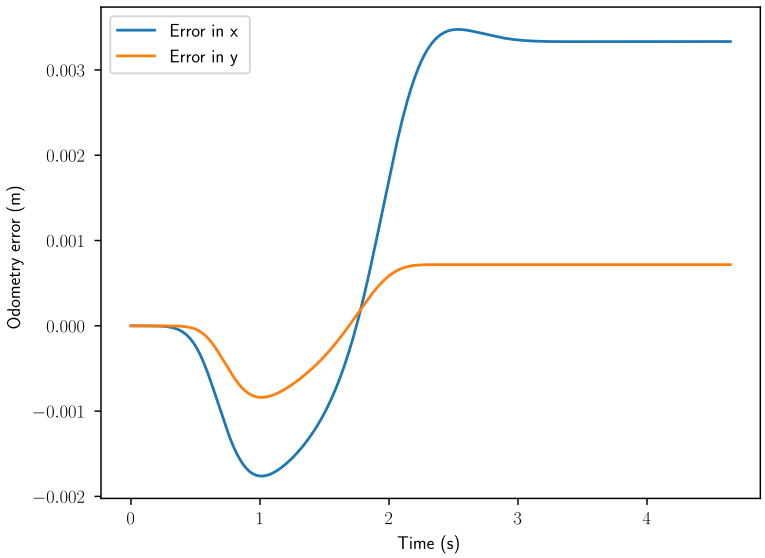


Figure 15.1: Odometry error compared to method using twists

The highest error is $0.0035m$, or roughly an eighth inch, in x over a $10m$ path starting with a $2m$ displacement. This is negligible when considering other possible sources of error like turning scrub on skid steer robots. Perhaps the difference is more noticeable for paths with higher curvatures and longer duration.

This page intentionally left blank

V Motion planning

This page intentionally left blank

16. 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.

For one degree of freedom (1 DOF) point-to-point movements in FRC, the most commonly used profiles are the trapezoidal profile (figure 16.1) and the S-curve profile (figure 16.2). 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.

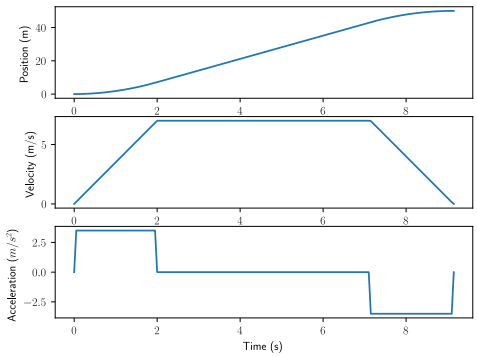


Figure 16.1: Trapezoidal profile

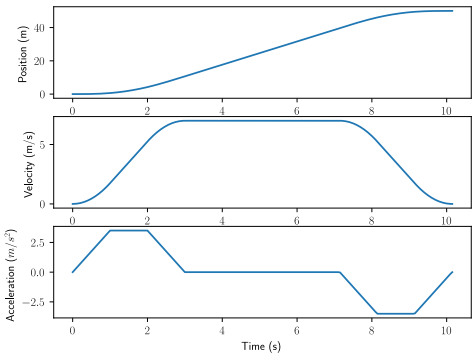


Figure 16.2: S-curve profile

These profiles are given their names based on the shape of their velocity trajectory. The trapezoidal 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 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 trapezoidal 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 trapezoidal 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.

16.1 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 trapezoidal 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 16.1 and 16.2, we can see S-curve profiles are smoother than trapezoidal 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.

16.2 Profile selection

Since trapezoidal 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 modelling 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 trapezoidal profile.

16.3 Profile equations

The trapezoidal profile uses the following equations.

$$\begin{aligned} x(t) &= x_0 + v_0 t + \frac{1}{2} a t^2 \\ v(t) &= v_0 + at \end{aligned}$$

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. The S-curve profile equations also include jerk.

$$\begin{aligned} x(t) &= x_0 + v_0 t + \frac{1}{2} a t^2 + \frac{1}{6} j t^3 \\ v(t) &= v_0 + at + \frac{1}{2} j t^2 \\ a(t) &= a_0 + jt \end{aligned}$$

where j is the jerk at time t, $a(t)$ is the acceleration at time t, and a_0 is the initial acceleration.

More derivations are required to determine when to start and stop the different profile phases. The derivations for a trapezoid profile are in appendix D.5 and the derivations for an S-curve profile are in appendix D.6.

16.4 Other profile types

The ultimate goal of any profile is to match the profile's motion characteristics to the desired application. Trapezoidal 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

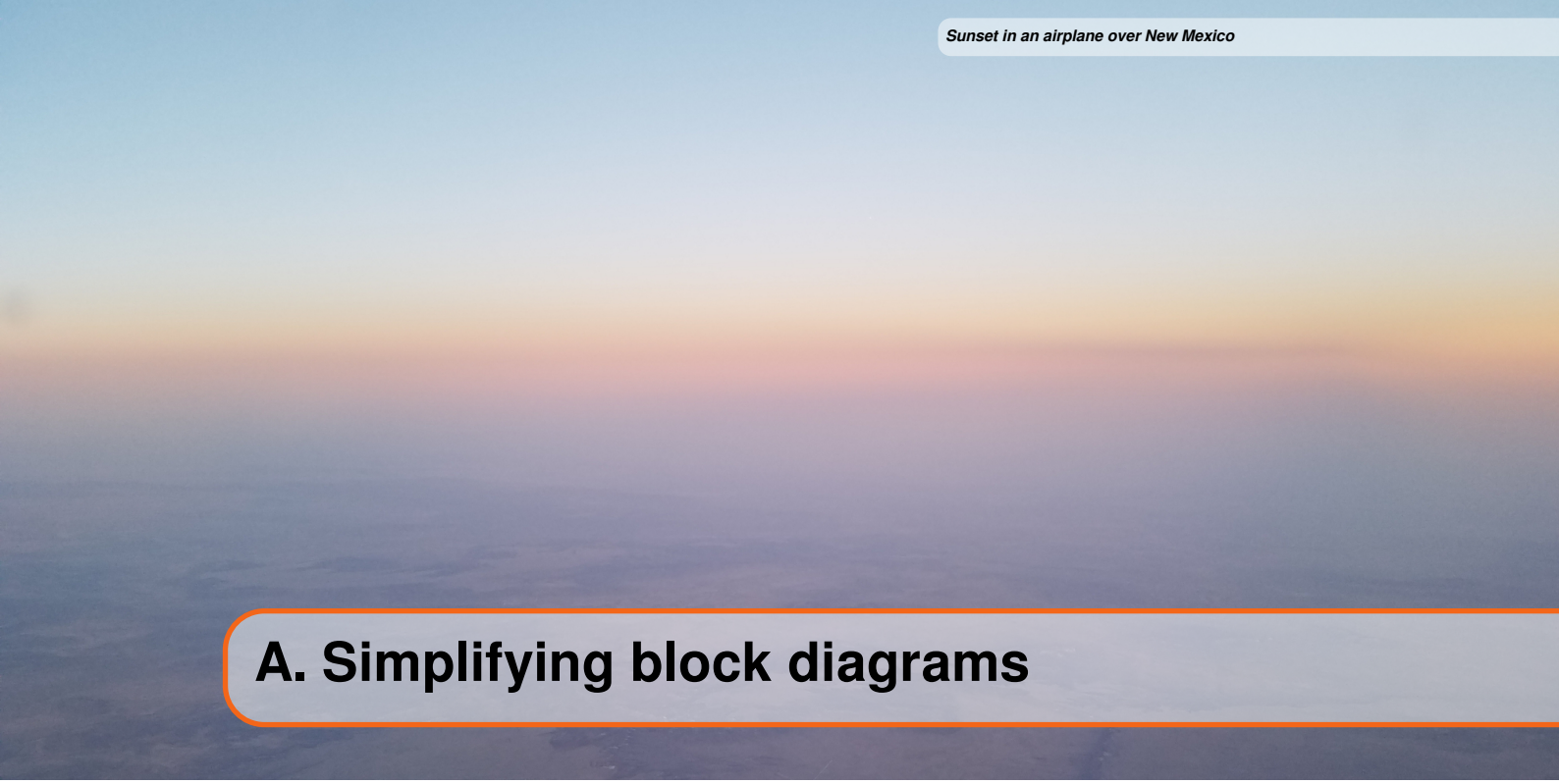
stepper motors because there is less torque available at higher speeds. However, notice that starting and ending accelerations are very high, and there is no "S" phase where the acceleration smoothly transitions to zero. If load oscillation is a problem, parabolic profiles may not work as well as an 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.

16.5 Further reading

FRC teams 254 and 971 gave a talk at FIRST World Championships in 2015 about motion profiles [1].

VI Appendices

This page intentionally left blank



A. Simplifying block diagrams

A.1 Cascaded blocks

$$Y = (P_1 P_2)X \tag{A.1}$$

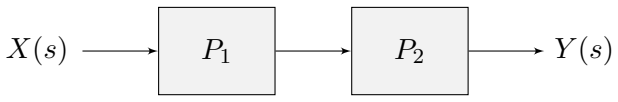


Figure A.1: Cascaded blocks

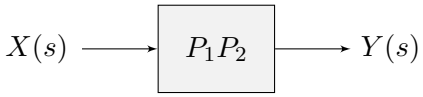


Figure A.2: Simplified cascaded blocks

A.2 Combining blocks in parallel

$$Y = P_1 X \pm P_2 X \tag{A.2}$$

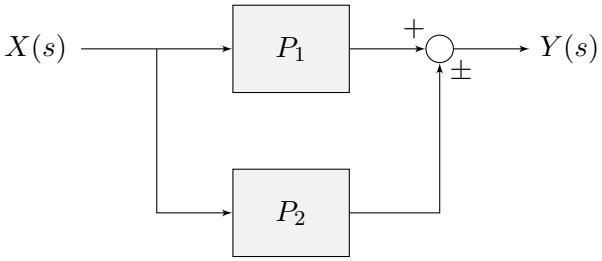


Figure A.3: Parallel blocks

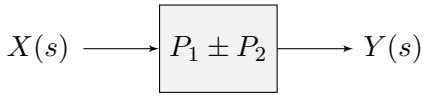


Figure A.4: Simplified parallel blocks

A.3 Removing a block from a feedforward loop

$$Y = P_1 X \pm P_2 X \tag{A.3}$$

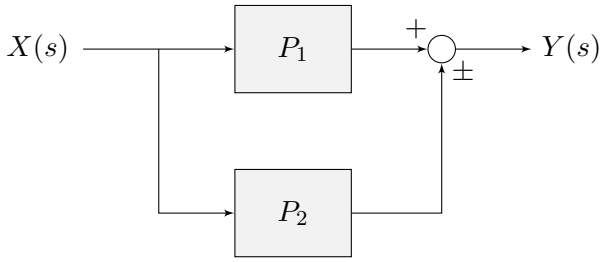


Figure A.5: Feedforward loop

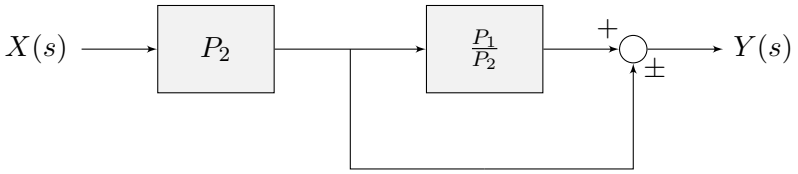


Figure A.6: Transformed feedforward loop

A.4 Eliminating a feedback loop

$$Y = P_1(X \mp P_2 Y) \tag{A.4}$$

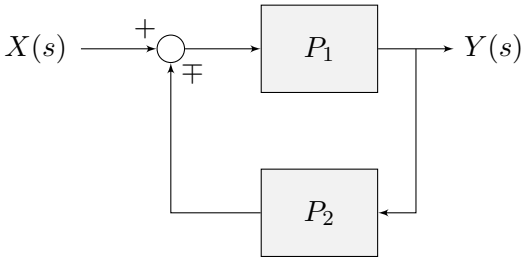


Figure A.7: Feedback loop

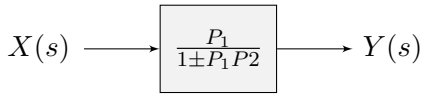


Figure A.8: Simplified feedback loop

A.5 Removing a block from a feedback loop

$$Y = P_1(X \mp P_2 Y) \tag{A.5}$$

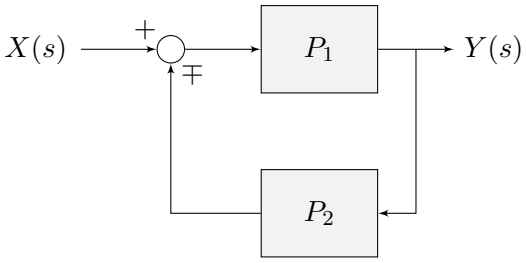


Figure A.9: Feedback loop

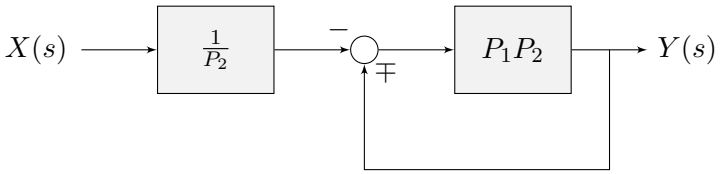
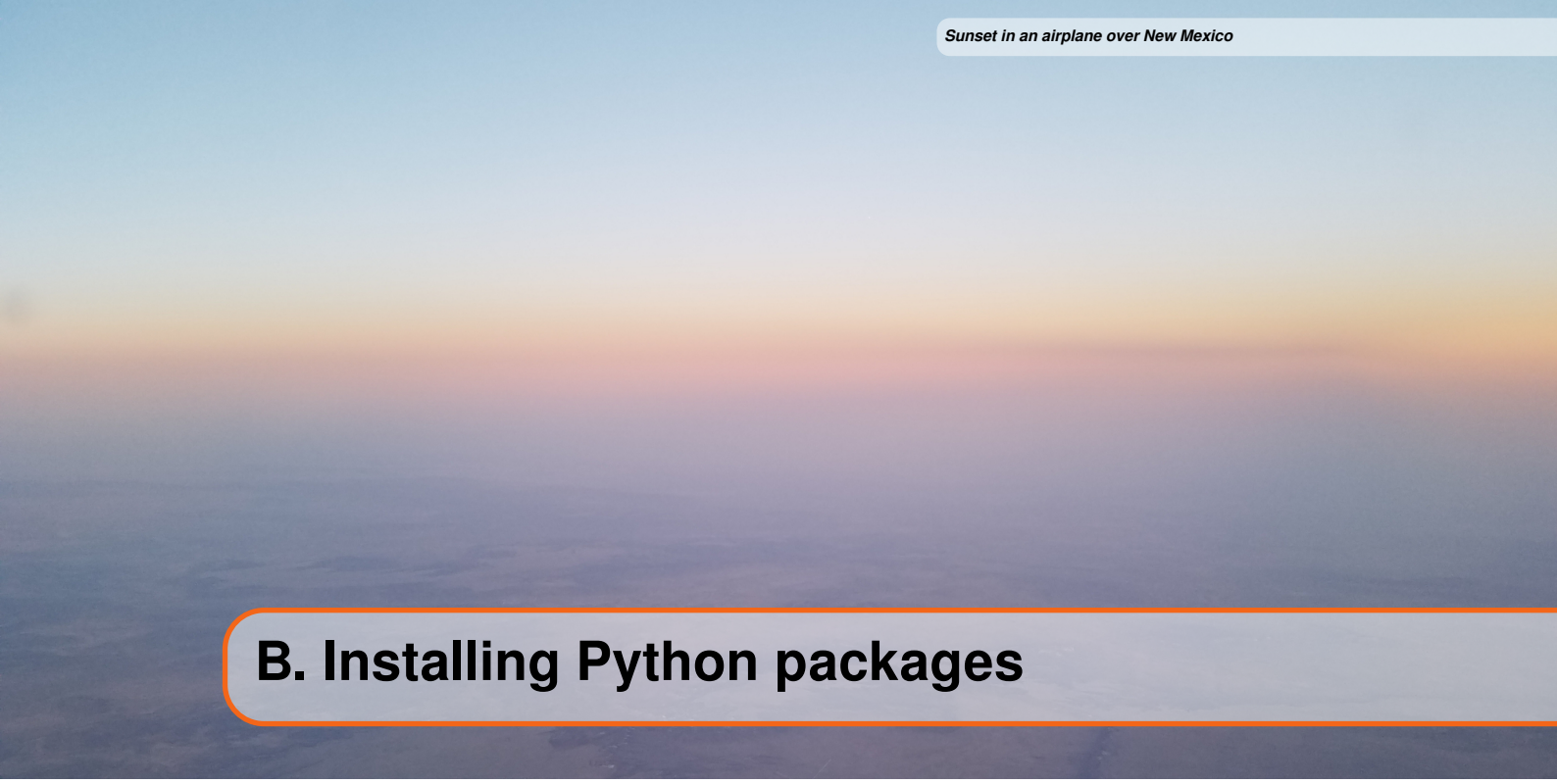


Figure A.10: Transformed feedback loop

This page intentionally left blank



B. Installing Python packages

B.1 Windows instructions

To install Python, download the installer for Python 3.5 or higher from `https://www.python.org/downloads/` and run it.

To install Python packages, run `py -3 -m pip install pkg` via cmd.exe or Powershell where `pkg` should be the name of the package. Packages can be upgraded with `py -3 -m pip install --user --upgrade pkg`.

B.2 Linux instructions

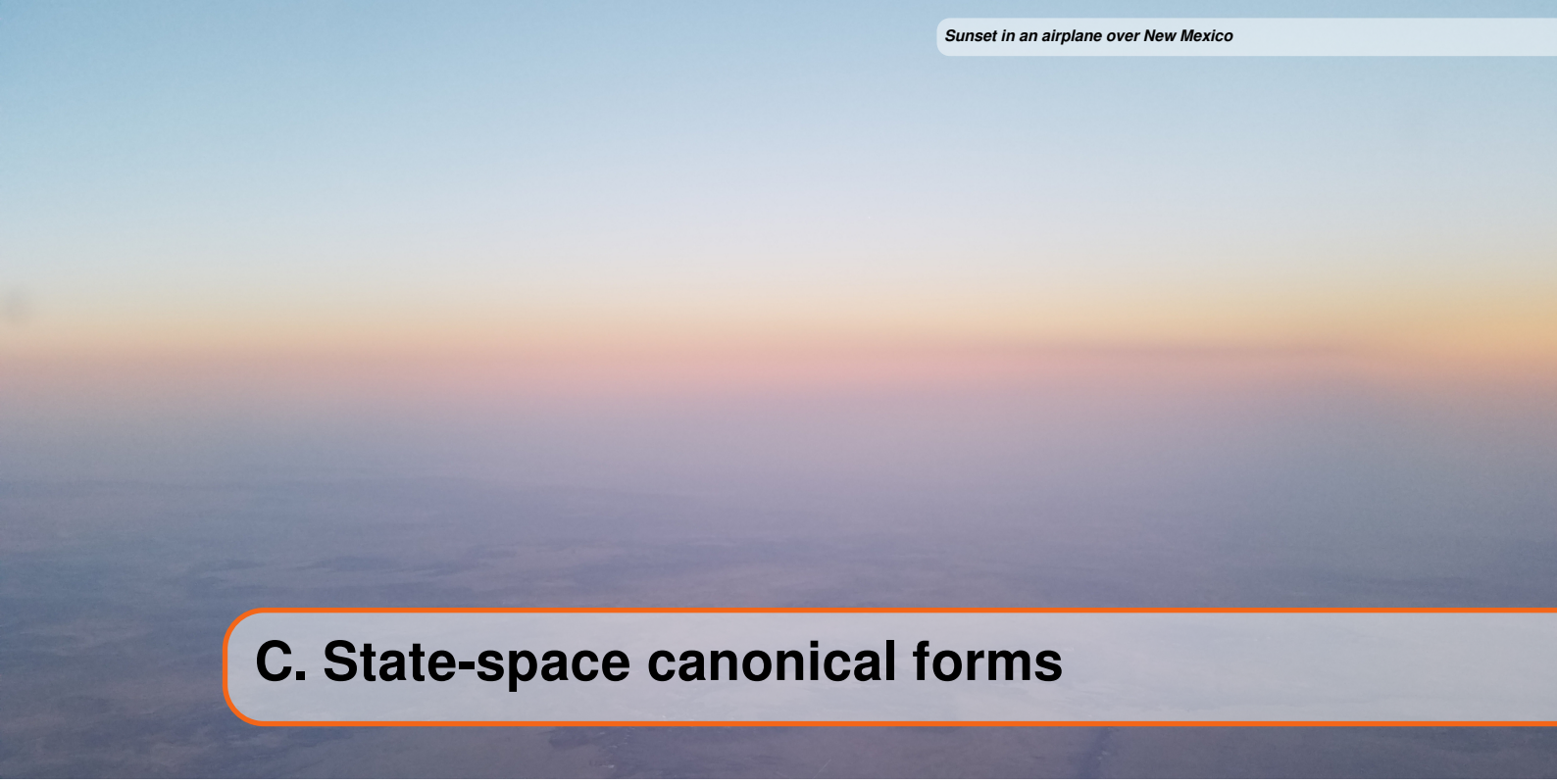
To install Python, install the appropriate packages from table B.1 using your system's package manager.

Debian/Ubuntu	Arch Linux
python3	python
python3-pip	python-pip

Table B.1: Required system packages

To install Python packages, run `pip3 install --user pkg` where `pkg` should be the name of the package. Using `--user` makes installation not require root privileges. Packages can be upgraded with `pip3 install --user --upgrade pkg`.

This page intentionally left blank



C. State-space canonical forms

There are two canonical forms used to represent state-space models: controllable canonical form and observable canonical form. They are used to guarantee controllability and observability of a system respectively, which are mathematical duals of each other. That is, the controller and estimator (state observer) are complementary problems.

C.1 Controllable canonical form

Given a system of the form

$$G(s) = \frac{n_1 s^3 + n_2 s^2 + n_3 s + n_4}{s^4 + d_1 s^3 + d_2 s^2 + d_3 s + d_4} \tag{C.1}$$

the canonical realization of it that satisfies equation (9.5) is

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -d_4 & -d_3 & -d_2 & -d_1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{u}(t) \tag{C.2}$$

$$\mathbf{y}(t) = \begin{bmatrix} n_4 & n_3 & n_2 & n_1 \end{bmatrix} \mathbf{x}(t) \tag{C.3}$$

C.2 Observable canonical form

The canonical realization of the system in equation (C.1) that satisfies equation (9.6) is

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 0 & 0 & -d_4 \\ 1 & 0 & 0 & -d_3 \\ 0 & 1 & 0 & -d_2 \\ 0 & 0 & 1 & -d_1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} n_4 \\ n_3 \\ n_2 \\ n_1 \end{bmatrix} \mathbf{u}(t) \tag{C.4}$$

$$\mathbf{y}(t) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}(t) \tag{C.5}$$

D. Derivations

D.1 Transfer function in feedback

Given the feedback network in figure D.1, find an expression for $Y(s)$.

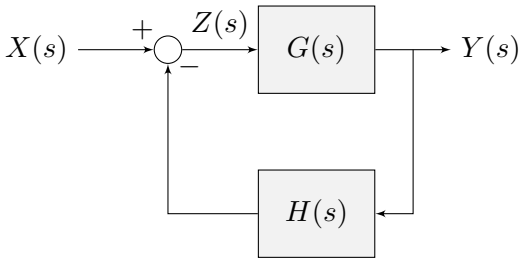


Figure D.1: Closed-loop block diagram

$$
\begin{aligned}
Y(s) &= Z(s)G(s) \\
Z(s) &= X(s) - Y(s)H(s) \\
X(s) &= Z(s) + Y(s)H(s) \\
X(s) &= Z(s) + Z(s)G(s)H(s) \\
\frac{Y(s)}{X(s)} &= \frac{Z(s)G(s)}{Z(s) + Z(s)G(s)H(s)} \\
\frac{Y(s)}{X(s)} &= \frac{G(s)}{1 + G(s)H(s)}
\end{aligned}
\tag{D.1}
$$

A more general form is

$$
\frac{Y(s)}{X(s)} = \frac{G(s)}{1 \mp G(s)H(s)} \tag{D.2}
$$

where positive feedback uses the top sign and negative feedback uses the bottom sign.

D.2 Optimal control law

For a continuous time linear system described by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{D.3}$$

with the cost function

$$J = \int_0^\infty \left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right) dt$$

where J represents a tradeoff between state excursion and control effort with the weighting factors $\mathbf{Q}$ and $\mathbf{R}$, the feedback control law which minimizes J is

$$\mathbf{u} = -\mathbf{K}\mathbf{x}$$

where $\mathbf{K}$ is given by

$$\mathbf{K} = \mathbf{R}^{-1}\left(\mathbf{B}^T\mathbf{P} + \mathbf{N}^T\right)$$

and $\mathbf{P}$ is found by solving the continuous time algebraic Riccati equation defined as

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - (\mathbf{P}\mathbf{B} + \mathbf{N})\,\mathbf{R}^{-1}\left(\mathbf{B}^T\mathbf{P} + \mathbf{N}^T\right) + \mathbf{Q} = 0$$

or alternatively

$$\mathcal{A}^T\mathbf{P} + \mathbf{P}\mathcal{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathcal{Q} = 0$$

with

$$\mathcal{A} = \mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{N}^T$$
$$\mathcal{Q} = \mathbf{Q} - \mathbf{N}\mathbf{R}^{-1}\mathbf{N}^T$$

Snippet D.1 computes the optimal infinite horizon, discrete time LQR controller.

```
import control as cnt
import numpy as np
import scipy as sp


def dlqr(sys, Q, R):
    """Solves for the optimal discrete-time LQR controller.

    x(n+1) = A * x(n) + B * u(n)
    J = sum(0, inf, x.T * Q * x + u.T * R * u)
```

```
    Keyword arguments:
    A -- numpy.array(states x states), The A matrix.
    B -- numpy.array(inputs x states), The B matrix.
    Q -- numpy.array(states x states), The state cost matrix.
    R -- numpy.array(inputs x inputs), The control effort cost matrix.

    Returns:
    numpy.array(states x inputs), K
    """
    m = sys.A.shape[0]

    controllability_rank = np.linalg.matrix_rank(cnt.ctrb(sys.A, sys.B))
    if controllability_rank != m:
        print(
            "Warning: Controllability of %d != %d, uncontrollable state"
            % (controllability_rank, m)
        )

    # P = A.T * P * A - (A.T * P * B) * np.linalg.inv(R + B.T * P * B) *
    #     (B.T * P.T * A) + Q
    P = sp.linalg.solve_discrete_are(a=sys.A, b=sys.B, q=Q, r=R)

    F = np.linalg.inv(R + sys.B.T * P * sys.B) * sys.B.T * P * sys.A
    return F
```

Snippet D.1. Infinite horizon, discrete time LQR computation in Python

Other formulations of LQR for finite horizon and discrete time can be seen on Wikipedia [13].

MIT OpenCourseWare has a rigorous proof of the results shown above [21].

D.3 Zero-order hold for state-space

Starting with the continuous model

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

we know that the matrix exponential is

$$\frac{d}{dt}e^{\mathbf{A}t} = \mathbf{A}e^{\mathbf{A}t} = e^{\mathbf{A}t}\mathbf{A}$$

and by premultiplying the model we get

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

which we recognize 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.

$$\begin{aligned}
\mathbf{x}_k &\stackrel{def}{=} \mathbf{x}(kT) \\
\mathbf{x}_k &= e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau + \int_{kT}^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau + \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\underbrace{\left(e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau\right)}_{\mathbf{x}_k} + \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau
\end{aligned}$$

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 $d\tau = -dv$.

$$\begin{aligned}
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k - \left(\int_{v(kT)}^{v((k+1)T)} e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k - \left(\int_{T}^{0} e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k + \left(\int_{0}^{T} e^{\mathbf{A}v}\,dv\right)\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}e^{\mathbf{A}v}|_0^T\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}(e^{\mathbf{A}T} - e^{\mathbf{A}0})\mathbf{B}\mathbf{u}_k \\
\mathbf{x}_{k+1} &= e^{\mathbf{A}T}\mathbf{x}_k + \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B}\mathbf{u}_k
\end{aligned}$$

which is an exact solution to the discretization problem.

D.4 Kalman filter as Luenberger observer

A Luenberger observer is defined as

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{D.4}$$

$$\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_{k}^{-} \tag{D.5}$$

where a superscript of minus denotes *a priori* and plus denotes *a posteriori* estimate. Combining equation (D.4) and equation (D.5) gives

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k}^{-}) \tag{D.6}$$

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

$$\mathbf{K}_k = \mathbf{P}_{k}^{-}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k}^{-}\mathbf{H}^T + \mathbf{R})^{-1} \tag{D.7}$$

$$\hat{\mathbf{x}}_{k}^{+} = \hat{\mathbf{x}}_{k}^{-} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_{k}^{-}) \tag{D.8}$$

$$\mathbf{P}_{k}^{+} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k}^{-} \tag{D.9}$$

Predict step

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{+} + \mathbf{B}\mathbf{u}_k \tag{D.10}$$

$$\mathbf{P}_{k+1}^{-} = \mathbf{A}\mathbf{P}_{k}^{+}\mathbf{A}^T + \mathbf{\Gamma}\mathbf{Q}\mathbf{\Gamma}^T \tag{D.11}$$

Substitute equation (D.8) into equation (D.10).

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

Let $\mathbf{C} = \mathbf{H}$ and $\mathbf{L} = \mathbf{A}\mathbf{K}_k$.

$$\hat{\mathbf{x}}_{k+1}^{+} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k + \mathbf{L}(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k}^{-}) \tag{D.12}$$

which matches equation (D.6). Therefore, the eigenvalues of the Kalman filter observer can be obtained by

$$\begin{aligned}
&\text{eig}(\mathbf{A} - \mathbf{L}\mathbf{C}) \\
&\text{eig}(\mathbf{A} - (\mathbf{A}\mathbf{K}_k)(\mathbf{H})) \\
&\text{eig}(\mathbf{A}(\mathbf{I} - \mathbf{K}_k\mathbf{H}))
\end{aligned} \tag{D.13}$$

D.4.1 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.4 shows that $\mathbf{C} = \mathbf{H}$ and $\mathbf{L} = \mathbf{A}\mathbf{K}_k$. 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 and $\mathbf{C} = \mathbf{H}$ into the Kalman filter update equation.

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}(\mathbf{y}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1}^{-})$$
$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-})$$

Substitute in equation (13.4).

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1})$$

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

$$\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{A}\hat{\mathbf{x}}_{k}^{-} + \mathbf{B}\mathbf{u}_k \tag{D.14}$$

Update step

$$\hat{\mathbf{x}}_{k+1}^{+} = \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{A}^{-1}\mathbf{L}(\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1}) \tag{D.15}$$
$$\hat{\mathbf{y}}_{k+1} = \mathbf{C}\hat{\mathbf{x}}_{k+1}^{-} \tag{D.16}$$

D.5 Trapezoidal motion profile

D.6 S-curve motion profile


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 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.

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 gain The gain from the output to an earlier point in a control system diagram.

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 7.7 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 provides an estimate of 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 7.7 on gain and phase margin.
plant The system or collection of actuators being controlled.
pose The 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 PID controller.

realization In control theory, this is an implementation of a given input-output behavior as a state-space model.
reference The desired state.
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 1 for $t \geq 0$.
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 is 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.

Bibliography

Online

[13] Wikipedia Commons. *Linear-quadratic regulator*. URL: https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator (visited on 03/24/2018) (cited on page 193).

[15] 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 132).

[16] Kapitanyuk, Proskurnikov, and Cao. *A guiding vector field algorithm for path following control of nonholonomic mobile robots*. URL: https://arxiv.org/pdf/1610.04391.pdf (visited on 08/09/2018) (cited on page 138).

[17] 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 167).

[18] Charles F. Van Loan. *Computing Integrals Involving the Matrix Exponential*. URL: https://www.cs.cornell.edu/cv/ResearchPDF/computing.integrals.involving.Matrix.Exp.pdf (visited on 06/21/2018) (cited on page 115).

[19] Luca, Oriolo, and Vendittelli. *Control of Wheeled Mobile Robots: An Experimental Overview*. URL: https://www.dis.uniroma1.it/~labrob/pub/papers/Ramsete01.pdf (visited on 08/09/2018) (cited on pages 133, 134).

[20] MIT OpenCourseWare. *Linear Quadratic Regulator*. URL: https://ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec19.pdf (visited on 06/26/2018) (cited on page 91).

[21] Russ Tedrake. *Chapter 9. Linear Quadratic Regulators*. URL: http://underactuated.csail.mit.edu/underactuated.html?chapter=lqr (visited on 07/08/2018) (cited on page 193).

[22] Eric A. Wan and Rudolph van der Merwe. *The Unscented Kalman Filter for Nonlinear Estimation*. URL: https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf (visited on 06/26/2018) (cited on page 167).

Misc

[1] FRC team 254. *Motion Planning and Control in FRC*. 2015. URL: https://www.youtube.com/watch?v=8319J1BEHwM (cited on page 180).

[2] 3Blue1Brown. *Eigenvectors and eigenvalues*. 2016. URL: https://youtu.be/PFDu9oVAE-g (cited on page 84).

[3] 3Blue1Brown. *Essence of linear algebra*. 2016. URL: https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab (cited on page 59).

[4] 3Blue1Brown. *Essence of linear algebra preview*. 2016. URL: https://youtu.be/kjBOesZCoqc (cited on page 59).

[5] 3Blue1Brown. *Inverse matrices, column space, and null space*. 2016. URL: https://youtu.be/uQhTuRlWMxw (cited on page 78).

[6] 3Blue1Brown. *Linear combinations, span, and basis vectors*. 2016. URL: https://youtu.be/k7RM-ot2NWY (cited on page 64).

[7] 3Blue1Brown. *Linear transformations and matrices*. 2016. URL: https://youtu.be/kYB8IZa5AuE (cited on page 68).

[8] 3Blue1Brown. *Linear transformations and matrices*. 2016. URL: https://youtu.be/XkY2DOUCWMU (cited on page 71).

[9] 3Blue1Brown. *Nonsquare matrices as transformations between dimensions*. 2016. URL: https://youtu.be/v8VSDg_WQlA (cited on page 79).

[10] 3Blue1Brown. *The determinant*. 2016. URL: https://youtu.be/Ip3X9LOh2dk (cited on page 74).

[11] 3Blue1Brown. *Vectors, what even are they?* 2016. URL: https://youtu.be/fNk_zzaMoSs (cited on page 62).

[12] 3Blue1Brown. *Essence of calculus*. 2017. URL: https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr (cited on page 9).

[14] Brian Douglas. *Gain and Phase Margins Explained!* 2015. URL: https://youtu.be/ThoA4amCAX4 (cited on page 56).

Index