# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

## Table of Contents

# Preface

This book is about nonlinear dynamics and control, with a focus on mechanical systems. I've spent my career thinking about how to make robots move robustly, but also with speed, efficiency, and grace. I believe that this is best achieved through a tight coupling between mechanical design, passive dynamics, and nonlinear control synthesis. These notes contain selected material from dynamical systems theory, as well as linear and nonlinear control. But the dynamics of our robots quickly get too complex for us to handle with a pencil-and-paper approach. As a result, the primary focus of these notes is on computational approaches to control design, especially using optimization.

When I started teaching this class, and writing these notes, the computational approach to control was far from mainstream in robotics. I had just finished my Ph.D. focused on reinforcement learning (applied to a bipedal robot), and was working on optimization-based motion planning. I remember sitting at a robotics conference dinner as a young faculty, surrounded by people I admired, talking about optimization. One of the senior faculty said "Russ: the people that talk like you aren't the people that get real robots to work." Wow, have things changed. Now almost every advanced robot is using optimization or learning in the planning/control system.

Today, the conversations about reinforcement learning (RL) are loud and passionate enough to drown out almost every other conversation in the room. Ironically, now I am the older professor and I find myself still believing in RL, but not with the complete faith of my youth. There is so much one can understand about the structure of the equations that govern our mechanical systems; algorithms which don't make use of that structure are missing obvious opportunities for data efficiency and robustness. The dream is to make the learning algorithms discover this structure on

their own. My goal for this course, however, is to help *you* discover this structure, and to learn how to use this structure to develop stronger algorithms and to guide your scientific endeavors into learning-based control.

I'll go even further. I'm willing to bet that our views of intelligence in 10-20 years will look less like feedforward networks with a training mode and a test mode, and more like a *system* with dynamics that ebb and flow in a beautiful dance with the dynamics of the environment. These systems will move more flexibly between perception, forward prediction / sequential decision making, storing and retrieving long-term memories, and taking action. A fascinating question is whether it will be important for these systems to be embodied (e.g. in a robot) in order to explore the world at the timescales of classical mechanics that we learn and evolve with. It certainly makes for a wonderful playground.

Although the material in the book comes from many sources, the presentation is targeted very specifically at a handful of robotics problems. Concepts are introduced only when and if they can help progress the capabilities we are trying to develop. Many of the disciplines that I am drawing from are traditionally very rigorous, to the point where the basic ideas can be hard to penetrate for someone that is new to the field. I've made a conscious effort in these notes to keep a very informal, conversational tone even when introducing these rigorous topics, and to reference the most powerful theorems but only to prove them when that proof would add particular insights without distracting from the mainstream presentation. I hope that the result is a broad but reasonably self-contained and readable manuscript that will be of use to any enthusiastic roboticist.

## Organization

The material in these notes is organized into a few main parts. "Model Systems" introduces a series of increasingly complex dynamical systems and overviews some of the relevant results from the literature for each system. "Nonlinear Planning and Control" introduces quite general computational algorithms for reasoning about those dynamical systems, with optimization theory playing a central role. Many of these algorithms treat the dynamical system as known and deterministic until the last chapters in this part which introduce stochasticity and robustness. "Estimation and Learning" follows this up with techniques from statistics and machine learning which capitalize on this viewpoint to introduce additional algorithms which can operate with less assumptions on knowing the model or having perfect sensors. The book closes with an "Appendix" that provides slightly more introduction (and references) for the main topics used in the course.

The order of the chapters was chosen to make the book valuable as a reference. When teaching the course, however, I take a spiral trajectory through the material, introducing robot dynamics and control problems one at a time, and introducing only the techniques that are required to solve that particular problem.

## Software

All of the examples and algorithms in this book, plus many more, are now available as a part of our open-source software project: DRAKE. DRAKE is a C++ project, but in this text we will use Drake's Python bindings. I encourage super-users or readers who want to dig deeper to explore the C++ code as well (and to contribute back).

Please see the appendix for specific instructions for using DRAKE along with these notes.

First chapter

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Table of contents Next Chapter

# Chapter 1
# Fully-actuated vs. Underactuated Systems

Open in Colab

Robots today move far too conservatively, and accomplish only a fraction of the tasks and achieve a fraction of the performance that they are mechanically capable of. In many cases, we are still fundamentally limited by control technology which matured on rigid robotic arms in structured factory environments. The study of underactuated robotics focuses on building control systems which use the natural dynamics of the machines in an attempt to achieve extraordinary performance in terms of speed, efficiency, or robustness.

## 1.1 Motivation

Let's start with some examples, and some videos.

### 1.1.1 Honda's ASIMO vs. passive dynamic walkers

The world of robotics changed when, in late 1996, Honda Motor Co. announced that they had been working for nearly 15 years (behind closed doors) on walking robot technology. Their designs have continued to evolve, resulting in a humanoid robot they call ASIMO (Advanced Step in Innovative MObility). For nearly 20 years, Honda's robots were widely considered to represent the state of the art in walking robots, although there are now many robots with designs and performance very similar to ASIMO's. We will dedicate effort to understanding a few of the details of ASIMO when we discuss algorithms for walking... for now I just want you to become familiar with the look and feel of ASIMO's movements [watch the asimo video below now].

Figure 1.1 - Honda's ASIMO (from http://world.honda.com/ASIMO/video/)

I hope that your first reaction is to be incredibly impressed with the quality and versatility of ASIMO's movements. Now take a second look. Although the motions

are very smooth, there is something a little unnatural about ASIMO's gait. It feels a little like an astronaut encumbered by a heavy space suit. In fact this is a reasonable analogy... ASIMO is walking a bit like somebody that is unfamiliar with his/her dynamics. Its control system is using high-gain feedback, and therefore considerable joint torque, to cancel out the natural dynamics of the machine and strictly follow a desired trajectory. This control approach comes with a stiff penalty. ASIMO uses roughly 20 times the energy (scaled) that a human uses to walk on the flat (measured by cost of transport)[1]. Also, control stabilization in this approach only works in a relatively small portion of the state space (when the stance foot is flat on the ground), so ASIMO can't move nearly as quickly as a human, and cannot walk on unmodelled or uneven terrain.

Figure 1.2 - A 3D passive dynamic walker by Steve Collins and Andy Ruina[2].

For contrast, let's now consider a very different type of walking robot, called a passive dynamic walker (PDW). This "robot" has no motors, no controllers, no computer, but is still capable of walking stably down a small ramp, powered only by gravity [watch videos above now]. Most people will agree that the passive gait of this machine is more natural than ASIMO's; it is certainly more efficient. Passive walking machines have a long history - there are patents for passively walking toys dating back to the mid 1800's. We will discuss, in detail, what people know about the dynamics of these machines and what has been accomplished experimentally. This most impressive passive dynamic walker to date was built by Steve Collins in Andy Ruina's lab at Cornell[2].

Passive walkers demonstrate that the high-gain, dynamics-cancelling feedback approach taken on ASIMO is not a necessary one. In fact, the dynamics of walking is beautiful, and should be exploited - not cancelled out.

The world is just starting to see what this vision could look like. This video from Boston Dynamics is one of my favorites of all time:

Figure 1.3 - Boston Dynamics' Atlas robot does a backflip. Make sure you've seen the dancing video, too.

This result is a marvel of engineering (the mechanical design alone is amazing...). In this class, we'll teach you the computational tools required to make robots perform this way. We'll also try to reason about how robust these types of maneuvers are and

can be. Don't worry, if you do not have a super lightweight, super capable, and super durable humanoid, then a simulation will be provided for you.

### 1.1.2 Birds vs. modern aircraft

The story is surprisingly similar in a very different type of machine. Modern airplanes are extremely effective for steady-level flight in still air. Propellers produce thrust very efficiently, and today's cambered airfoils are highly optimized for speed and/or efficiency. It would be easy to convince yourself that we have nothing left to learn from birds. But, like ASIMO, these machines are mostly confined to a very conservative, low angle-of-attack flight regime where the aerodynamics on the wing are well understood. Birds routinely execute maneuvers outside of this flight envelope (for instance, when they are landing on a perch), and are considerably more effective than our best aircraft at exploiting energy (eg, wind) in the air.

As a consequence, birds are extremely efficient flying machines; some are capable of migrating thousands of kilometers with incredibly small fuel supplies. The wandering albatross can fly for hours, or even days, without flapping its wings - these birds exploit the shear layer formed by the wind over the ocean surface in a technique called dynamic soaring. Remarkably, the metabolic cost of flying for these birds is indistinguishable from the baseline metabolic cost[3], suggesting that they can travel incredible distances (upwind or downwind) powered almost completely by gradients in the wind. Other birds achieve efficiency through similarly rich interactions with the air - including formation flying, thermal soaring, and ridge soaring. Small birds and large insects, such as butterflies and locusts, use "gust soaring" to migrate hundreds or even thousands of kilometers carried primarily by the wind.

Birds are also incredibly maneuverable. The roll rate of a highly acrobatic aircraft (e.g, the A-4 Skyhawk) is approximately 720 deg/sec[4]; a barn swallow has a roll rate in excess of 5000 deg/sec[4]. Bats can be flying at full-speed in one direction, and completely reverse direction while maintaining forward speed, all in just over 2 wing-beats and in a distance less than half the wingspan[5]. Although quantitative flow visualization data from maneuvering flight is scarce, a dominant theory is that the ability of these animals to produce sudden, large forces for maneuverability can be attributed to unsteady aerodynamics, e.g., the animal creates a large suction vortex to rapidly change direction[6]. These astonishing capabilities are called upon routinely in maneuvers like flared perching, prey-catching, and high speed flying through forests and caves. Even at high speeds and high turn rates, these animals are capable of incredible agility - bats sometimes capture prey on their wings, Peregrine falcons can pull 25 G's out of a 240 mph dive to catch a sparrow in mid-flight[7], and even the small birds outside our building can be seen diving through a chain-link fence to grab a bite of food.

Although many impressive statistics about avian flight have been recorded, our understanding is partially limited by experimental accessibility - it is quite difficult to carefully measure birds (and the surrounding airflow) during their most impressive maneuvers without disturbing them. The dynamics of a swimming fish are closely related, and can be more convenient to study. Dolphins have been known to swim gracefully through the waves alongside ships moving at 20 knots[6]. Smaller fish, such as the bluegill sunfish, are known to possess an escape response in which they propel themselves to full speed from rest in less than a body length; flow visualizations indeed confirm that this is accomplished by creating a large suction vortex along the side of the body[8] - similar to how bats change direction in less than a body length. There are even observations of a dead fish swimming upstream by pulling energy out of the wake of a cylinder; this passive propulsion is presumably part of the technique used by rainbow trout to swim upstream at mating season[9].

### 1.1.3 Manipulation

Despite a long history of success in industrial applications, and the huge potential for consumer applications, we still don't have robot arms that can perform any

meaningful tasks in the home. Admittedly, the perception problem (using sensors to detect/localize objects and understand the scene) for home robotics is incredibly difficult. But even if we were given a perfect perception system, our robots are still a long way from performing basic object manipulation tasks with the dexterity and versatility of a human.

Most robots that perform object manipulation today use a stereotypical pipeline. First, we enumerate a handful of contact locations on the hand (these points, and only these points, are allowed to contact the world). Then, given a localized object in the environment, we plan a collision-free trajectory for the arm that will move the hand into a "pre-grasp" location. At this point the robot closes it's eyes (figuratively) and closes the hand, hoping that the pre-grasp location was good enough that the object will be successfully grasped using e.g. only current feedback in the fingers to know when to stop closing. "Underactuated hands" make this approach more successful, but the entire approach really only works well for enveloping grasps.

The enveloping grasps approach may actually be sufficient for a number of simple pick-and-place tasks, but it is a very poor representation of how humans do manipulation. When humans manipulate objects, the contact interactions with the object and the world are very rich -- we often use pieces of the environment as fixtures to reduce uncertainty, we commonly *exploit* slipping behaviors (e.g. for picking things up, or reorienting it in the hand), and our brains don't throw NaNs if we use the entire surface of our arms to e.g. manipulate a large object.

In the last few years, I've began to focus my own research to problems in the manipulation domain. In this space, the interaction between dynamics and perception is incredibly rich. As a result, I've started an entirely separate set of notes (and a second course) on manipulation.

By the way, in most cases, if the robots fail to make contact at the anticipated contact times/locations, bad things can happen. The results are hilarious and depressing at the same time. (Let's fix that!)

### 1.1.4 The common theme

Classical control techniques for robotics are based on the idea that feedback can be used to override the dynamics of our machines. These examples suggest that to achieve outstanding dynamic performance (efficiency, agility, and robustness) from our robots, we need to understand how to design control systems which take advantage of the dynamics, not cancel them out. That is the topic of this course.

Surprisingly, many formal control ideas do not support the idea of "exploiting" the dynamics. Optimal control formulations (which we will study in depth) allow it in principle, but optimal control of nonlinear systems is still a relatively ad hoc discipline. Sometimes I joke that in order to convince a control theorist to consider the dynamics, you have to do something drastic, like taking away her control authority - remove a motor, or enforce a torque-limit. These issues have created a formal class of systems, the underactuated systems, for which people have begun to more carefully consider the dynamics of their machines in the context of control.

## 1.2 DEFINITIONS

According to Newton, the dynamics of mechanical systems are second order ($F = ma$ ). Their state is given by a vector of positions, $\mathbf{q}$ (also known as the configuration vector), and a vector of velocities, $\dot{\mathbf{q}}$, and (possibly) time. The general form for a second-order control dynamical system is:

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}, t),$$

where $\mathbf{u}$ is the control vector.

**Definition 1.1 (Underactuated Control Differential Equations)** A second-order control differential equation described by the equations

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}, t) \tag{1}$$

is *fully actuated* in state $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$ and time $t$ if the resulting map $\mathbf{f}$ is surjective: for every $\ddot{\mathbf{q}}$ there exists a $\mathbf{u}$ which produces the desired response. Otherwise it is *underactuated* (in $\mathbf{x}$ at time $t$).

This definition can also be extended to discrete-time systems and/or differential inclusions.

As we will see, the dynamics for many of the robots that we care about turn out to be affine in commanded torque, so let's consider a slightly constrained form:

$$\ddot{\mathbf{q}} = \mathbf{f}_1(\mathbf{q}, \dot{\mathbf{q}}, t) + \mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)\mathbf{u}. \tag{2}$$

For a control dynamical system described by equation (2), if we have

$$\text{rank}\,[\mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)] < \dim\,[\mathbf{q}], \tag{3}$$

then the system is underactuated. Be careful, though -- sometimes we will write equations that look like (2) but tack on additional constraints like $|\mathbf{u}| \leq 1$; as we will discuss below, input limits and other constraints can also make a system underactuated.

Notice that whether or not a control system is underactuated may depend on the state of the system or even on time, although for most systems (including all of the systems in this book) underactuation is a global property of the system. We will refer to a system as underactuated if it is underactuated in *all* states and times. In practice, we often refer informally to systems as fully actuated as long as they are fully actuated in *most* states (e.g., a "fully-actuated" system might still have joint limits or lose rank at a kinematic singularity). Admittedly, this permits the existence of a gray area, where it might feel awkward to describe the *system* as either fully- or underactuated (we should instead only describe its states); even powerful robot arms on factory floors do have actuator limits, but we can typically design controllers for them as though they were fully actuated. The primary interest of this text is in systems for which the underactuation is useful/necessary for developing a control strategy.

**Example 1.1 (Robot Manipulators)**

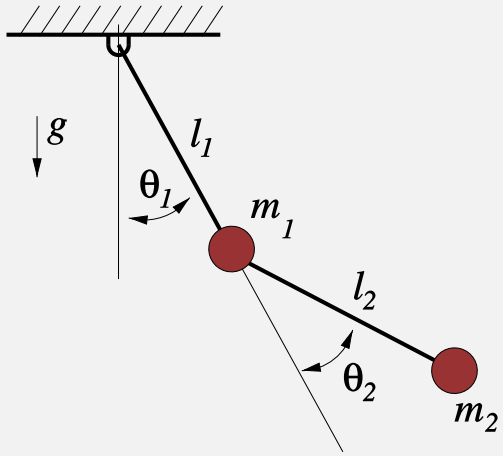



Figure 1.4 - Simple double pendulum. Click here for an animation.

Consider the simple robot manipulator illustrated above. As described in the Appendix, the equations of motion for this system are quite simple to derive, and take the form of the standard "manipulator equations":

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}.$$

It is well known that the inertia matrix, $\mathbf{M}(\mathbf{q})$ is (always) uniformly symmetric and positive definite, and is therefore invertible. Putting the system into the form of equation 2 yields:

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\left[\tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}\right].$$

Because $\mathbf{M}^{-1}(\mathbf{q})$ is always full rank, we find that a system described by the manipulator equations is fully-actuated if and only if $\mathbf{B}$ is full row rank. For this particular example, $\mathbf{q} = [\theta_1, \theta_2]^T$ and $\mathbf{u} = [\tau_1, \tau_2]^T$ (motor torques at the joints), and $\mathbf{B} = \mathbf{I}_{2\times 2}$. The system is fully actuated.

**Python Example**

I personally learn best when I can experiment and get some physical intuition. Most chapters in these notes have an associated Jupyter notebook that can run on Google's Colab; this chapter's notebook makes it easy for you to see this system in action.


Try it out! You'll see how to simulate the double pendulum, and even how to inspect the dynamics symbolically.

Note: You can also run the code on your own machines (see the Appendix for details).

While the basic double pendulum is fully actuated, imagine the somewhat bizarre case that we have a motor to provide torque at the elbow, but no motor at the shoulder. In this case, we have $\mathbf{u} = \tau_2$, and $\mathbf{B}(\mathbf{q}) = [0, 1]^T$. This system is clearly underactuated. While it may sound like a contrived example, it turns out that it is almost exactly the dynamics we will use to study as our simplest model of walking later in the class.

The matrix $\mathbf{f}_2$ in equation 2 always has $\dim[\mathbf{q}]$ rows, and $\dim[\mathbf{u}]$ columns. Therefore, as in the example, one of the most common cases for underactuation, which trivially implies that $\mathbf{f}_2$ is not full row rank, is $\dim[\mathbf{u}] < \dim[\mathbf{q}]$. This is the case when a robot has joints with no motors. But this is not the only case. The human body, for instance, has an incredible number of actuators (muscles), and in many cases has multiple muscles per joint; despite having more actuators than position variables, when I jump through the air, there is no combination of muscle inputs that can change the ballistic trajectory of my center of mass (barring aerodynamic effects). My control system is underactuated.

A quick note about notation. When describing the dynamics of rigid-body systems in this class, I will use $\mathbf{q}$ for configurations (positions), $\dot{\mathbf{q}}$ for velocities, and use $\mathbf{x}$ for the full state ($\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$). There is an important limitation to this convention (3D angular velocity should not be represented as the derivative of 3D pose) described in the Appendix, but it will keep the notes cleaner. Unless otherwise noted, vectors are always treated as column vectors. Vectors and matrices are bold (scalars are not).

## 1.3 Feedback Equivalence

Fully-actuated systems are dramatically easier to control than underactuated systems. The key observation is that, for fully-actuated systems with known dynamics (e.g., $\mathbf{f}_1$ and $\mathbf{f}_2$ are known for a second-order control-affine system), it is possible to use feedback to effectively change an arbitrary control problem into the problem of controlling a trivial linear system.

When $\mathbf{f}_2$ is full row rank, it is invertible. Consider the potentially nonlinear feedback control:

$$\mathbf{u} = \pi(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{f}_2^{-1}(\mathbf{q}, \dot{\mathbf{q}}, t) \left[\mathbf{u}' - \mathbf{f}_1(\mathbf{q}, \dot{\mathbf{q}}, t)\right],$$

where $\mathbf{u}'$ is the new control input (an input to your controller). Applying this feedback controller to equation 2 results in the linear, decoupled, second-order system:

$$\ddot{\mathbf{q}} = \mathbf{u}'.$$

In other words, if $\mathbf{f}_1$ and $\mathbf{f}_2$ are known and $\mathbf{f}_2$ is invertible, then we say that the system is "feedback equivalent" to $\ddot{\mathbf{q}} = \mathbf{u}'$. There are a number of strong results which generalize this idea to the case where $\mathbf{f}_1$ and $\mathbf{f}_2$ are estimated, rather than known (e.g, [10]).

If $\mathbf{f}_2$ is not square, for instance you have multiple actuators per joint, then this inverse may not be unique.

## Example 1.2 (Feedback Cancellation on the Double Pendulum)

Let's say that we would like our simple double pendulum to act like a simple single pendulum (with damping), whose dynamics are given by:

$$\ddot{\theta}_1 = -\frac{g}{l}\sin\theta_1 - b\dot{\theta}_1$$
$$\ddot{\theta}_2 = 0.$$

This is easily achieved using

$$\mathbf{u} = \mathbf{B}^{-1}\left[\mathbf{C}\dot{\mathbf{q}} - \tau_g + \mathbf{M}\begin{bmatrix} -\frac{g}{l}s_1 - b\dot{q}_1 \\ 0 \end{bmatrix}\right].$$

Note that our chosen dynamics do not actually stabilize $\theta_2$ - this detail was left out for clarity, but would be necessary for any real implementation.

Since we are embedding a nonlinear dynamics (not a linear one), we refer to this as "feedback cancellation", or "dynamic inversion". This idea reveals why I say control is easy -- for the special case of a fully-actuated deterministic system with known dynamics. For example, it would have been just as easy for me to invert gravity. Observe that the control derivations here would not have been any more difficult if the robot had 100 joints.

You can run these examples in the notebook:


As always, I highly recommend that you take a few minutes to read through the source code.

Fully-actuated systems are feedback equivalent to $\ddot{\mathbf{q}} = \mathbf{u}$, whereas *underactuated systems are not feedback equivalent to* $\ddot{\mathbf{q}} = \mathbf{u}$. Therefore, unlike fully-actuated systems, the control designer has no choice but to reason about the more complex dynamics of the plant in the control design. When these dynamics are nonlinear, this can dramatically complicate feedback controller design.

A related concept is feedback linearization. The feedback-cancellation controller in the example above is an example of feedback linearization -- using feedback to convert a nonlinear system into a controllable linear system. Asking whether or not a system is "feedback linearizable" is not the same as asking whether it is underactuated; even a controllable linear system can be underactuated, as we will discuss soon.

## 1.4 Input and State Constraints

Although the dynamic constraints due to missing actuators certainly embody the spirit of this course, many of the systems we care about could be subject to other dynamic constraints as well. For example, the actuators on our machines may only be mechanically capable of producing some limited amount of torque, or there may be a physical obstacle in the free space with which we cannot permit our robot to come into contact with.

**Definition 1.2 (Input and State Constraints)** A dynamical system described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$ may be subject to one or more constraints described by $\phi(\mathbf{x}, \mathbf{u}, t) \geq 0$.

In practice it can be useful to separate out constraints which depend only on the input, e.g. $\phi(\mathbf{u}) \geq 0$, such as actuator limits, as they can often be easier to handle than state constraints. An obstacle in the environment might manifest itself as one or more constraints that depend only on position, e.g. $\phi(\mathbf{q}) \geq 0$.

By our generalized definition of underactuation, we can see that input constraints can certainly cause a system to be underactuated. State (only) constraints are more subtle -- in general these actually reduce the dimensionality of the state space, therefore requiring less dimensions of actuation to achieve "full" control, but we only reap the benefits if we are able to perform the control design in the "minimal coordinates" (which is often difficult).

**Example 1.3 (Input limits)**
Consider the constrained second-order linear system

$$\ddot{x} = u, \quad |u| \leq 1.$$

By our definition, this system is underactuated. For example, there is no $u$ which can produce the acceleration $\ddot{x} = 2$.

Input and state constraints can complicate control design in similar ways to having an insufficient number of actuators, (i.e., further limiting the set of the feasible trajectories), and often require similar tools to find a control solution.

### 1.4.1 Nonholonomic constraints

You might have heard of the term "nonholonomic system" (see e.g. [11]), and be thinking about how nonholonomy relates to underactuation. Briefly, a nonholonomic constraint is a constraint of the form $\phi(\mathbf{q}, \dot{\mathbf{q}}, t) = 0$, which cannot be integrated into a constraint of the form $\phi(\mathbf{q}, t) = 0$ (a holonomic constraint). A nonholonomic constraint does not restrain the possible configurations of the system, but rather the manner in which those configurations can be reached. While a holonomic constraint reduces the number of degrees of freedom of a system by one, a nonholonomic constraint does not. An automobile or traditional wheeled robot provides a canonical example:

**Example 1.4 (Wheeled robot)**
Consider a simple model of a wheeled robot whose configuration is described by its Cartesian position $x, y$ and its orientation, $\theta$, so $\mathbf{q} = [x, y, \theta]^T$. The system is subject to a differential constraint that prevents side-slip,

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$v = \sqrt{\dot{x}^2 + \dot{y}^2}$$

or equivalently,

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0.$$

This constraint cannot be integrated into a constraint on configuration—the car can get to any configuration $(x, y, \theta)$, it just can't move directly sideways—so this is a nonholonomic constraint.

Contrast the wheeled robot example with a robot on train tracks. The train tracks correspond to a holonomic constraint: the track constraint can be written directly in terms of the configuration $\mathbf{q}$ of the system, without using the velocity $\dot{\mathbf{q}}$. Even though the track constraint could also be written as a differential constraint on the velocity, it would be possible to integrate this constraint to obtain a constraint on configuration. The track restrains the possible configurations of the system.

A nonholonomic constraint like the no-side-slip constraint on the wheeled vehicle certainly results in an underactuated system. The converse is not necessarily true—the double pendulum system which is missing an actuator is underactuated but would not typically be called a nonholonomic system. Note that the Lagrangian equations of motion are a constraint of the form

$$\phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) = 0,$$

so do not qualify as a nonholonomic constraint.

## 1.5 Underactuated robotics

The control of underactuated systems is an open and interesting problem in controls. Although there are a number of special cases where underactuated systems have been controlled, there are relatively few general principles. Now here's the rub... most of the interesting problems in robotics are underactuated:

- Legged robots are underactuated. Consider a legged machine with $N$ internal joints and $N$ actuators. If the robot is not bolted to the ground, then the degrees of freedom of the system include both the internal joints and the six degrees of freedom which define the position and orientation of the robot in space. Since $\mathbf{u} \in \mathbb{R}^N$ and $\mathbf{q} \in \mathbb{R}^{N+6}$, equation (3) is satisfied.
- (Most) Swimming and flying robots are underactuated. The story is the same here as for legged machines. Each control surface adds one actuator and one DOF. And this is already a simplification, as the true state of the system should really include the (infinite-dimensional) state of the flow.
- Robot manipulation is (often) underactuated. Consider a fully-actuated robotic arm. When this arm is manipulating an object with degrees of freedom (even a brick has six), it can become underactuated. If force closure is achieved, and maintained, then we can think of the system as fully-actuated, because the degrees of freedom of the object are constrained to match the degrees of freedom of the hand. That is, of course, unless the manipulated object has extra DOFs (for example, any object that is deformable).

Even control systems for fully-actuated and otherwise unconstrained systems can be improved using the lessons from underactuated systems, particularly if there is a need to increase the efficiency of their motions or reduce the complexity of their designs.

## 1.6 Goals for the course

This course is based on the observation that there are new computational tools from optimization theory, control theory, motion planning, and even machine learning which can be used to design feedback control for underactuated systems. The goal of this class is to develop these tools in order to design robots that are more dynamic and more agile than the current state-of-the-art.

The target audience for the class includes both computer science and mechanical/ aero students pursuing research in robotics. Although I assume a comfort with linear algebra, ODEs, and Python, the course notes aim to provide most of the material and references required for the course.

I have a confession: I actually think that the material we'll cover in these notes is valuable far beyond robotics. I think that systems theory provides a powerful language for organizing computation in exceedingly complex systems -- especially when one is trying to program and/or analyze systems with continuous variables in a feedback loop (which happens throughout computer science and engineering, by the way). I hope you find these tools to be broadly useful, even if you don't have a humanoid robot capable of performing a backflip at your immediate disposal.

## 1.7 Exercises

### Exercise 1.1 (Atlas Backflip)

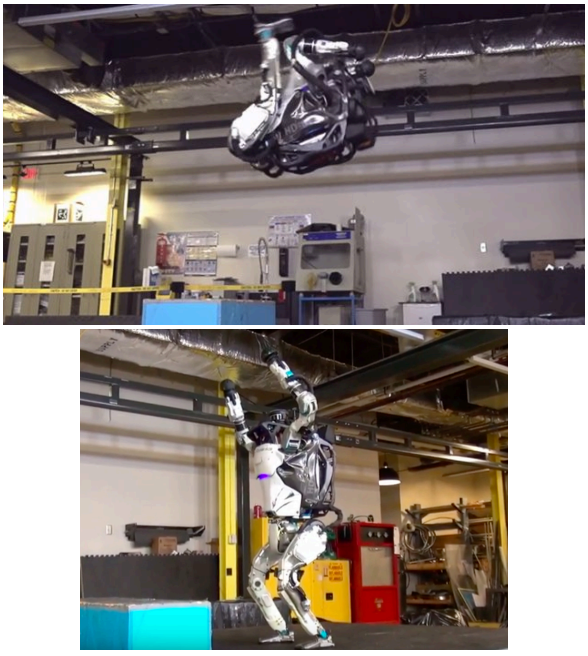


Figure 1.5 - Atlas doing a backflip and Atlas standing.

At the beginning of this chapter you have seen the Atlas humanoid doing a backflip. Now consider the robot in the two states captured in the figure above. Assuming that Atlas' actuators can produce unbounded torques $\mathbf{u}$, establish whether or not each of the following statements is true. Briefly justify your answer.

a. The state of the humanoid can be represented by the angles and the angular velocities of all its joints.
b. While doing the backflip (state in the left figure), the humanoid is fully actuated.
c. While standing (state in the right figure), the humanoid is fully actuated.

## Exercise 1.2 (Trajectory Tracking in State Space)

Take a robot whose dynamics is governed by equation 2, and assume it to be fully actuated in all states $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$ at all times $t$.

a. For any twice-differentiable desired trajectory $\mathbf{q}_{\text{des}}(t)$, is it always possible to find a control signal $\mathbf{u}(t)$ such that $\mathbf{q}(t) = \mathbf{q}_{\text{des}}(t)$ for all $t \geq 0$, provided that $\mathbf{q}(0) = \mathbf{q}_{\text{des}}(0)$ and $\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_{\text{des}}(0)$?

b. Now consider the simplest fully-actuated robot: the double integrator. The dynamics of this system reads $m\ddot{q} = u$, and you can think of it as a cart of mass $m$ moving on a straight rail, controlled by a force $u$. The figure below depicts its phase portrait when $u = 0$. Is it possible to find a control signal $u(t)$ that drives the double integrator from the initial state $\mathbf{x}(0) = [2, 0.5]^T$ to the origin along a straight line (blue trajectory)? Does the answer change if we set $\mathbf{x}(0)$ to be $[2, -0.5]^T$ (red trajectory)?

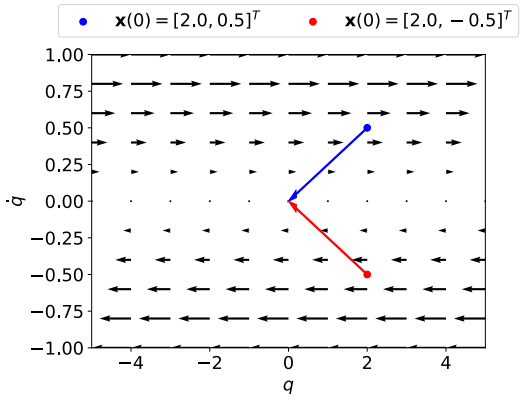



Figure 1.6 - Phase portrait of the double integrator.

c. The dynamics 2 are $n = \dim[\mathbf{q}]$ *second-order* differential equations. However, it is always possible (and we'll frequently do it) to describe these equations in terms of $2n$ *first-order* differential equations $\dot{\mathbf{x}} = f(\mathbf{x}, t)$. To this end, we simply define

$$f(\mathbf{x}, t) = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{f}_1(\mathbf{q}, \dot{\mathbf{q}}, t) + \mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)\mathbf{u} \end{bmatrix}.$$

For any twice-differentiable trajectory $\mathbf{x}_{\text{des}}(t)$, is it always possible to find a control $\mathbf{u}(t)$ such that $\mathbf{x}(t) = \mathbf{x}_{\text{des}}(t)$ for all $t \geq 0$, provided that $\mathbf{x}(0) = \mathbf{x}_{\text{des}}(0)$?

## Exercise 1.3 (Task-Space Control of the Double Pendulum)

In the example above, we have seen that the double pendulum with one motor per joint is a fully-actuated system. Here we consider a variation of it: instead of controlling the robot with the actuators at the shoulder and the elbow, we directly apply a force on the mass $m_2$ (tip of the second link). Let $\mathbf{u} = [u_1, u_2]^T$ be this force, with $u_1$ horizontal component (positive to the right) and $u_2$ vertical component (positive upwards). This modification leaves the equations of motion derived in the appendix example almost unchanged; the only difference is that the matrix $\mathbf{B}$ is now a function of $\mathbf{q}$. Namely, using the notation from the appendix,

$$\mathbf{B}(\mathbf{q}) = \begin{bmatrix} l_1 c_1 + l_2 c_{1+2} & l_1 s_1 + l_2 s_{1+2} \\ l_2 c_{1+2} & l_2 s_{1+2} \end{bmatrix}.$$

Is the double pendulum with the new control input still fully-actuated in all states? If not, identify the states in which it is underactuated.

## Exercise 1.4 (Underactuation of the Planar Quadrotor)

Later in the course we will study the dynamics of a quadrotor quite in depth, for the moment just look at the structure of the resulting equations of motion from the planar quadrotor section. The quadrotor is constrained to move in the vertical plane, with the gravity pointing downwards. The configuration vector $\mathbf{q} = [x, y, \theta]^T$ collects the position of the center of mass and the pitch angle. The control input is the thrust $\mathbf{u} = [u_1, u_2]^T$ produced by the two rotors. The input $\mathbf{u}$ can assume both signs and has no bounds.

a. Identify the set of states $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$ in which the system is underactuated.
b. For *all* the states in which the system is underactuated, identify an acceleration $\ddot{\mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}})$ that cannot be instantaneously achieved. Provide a rigorous proof of your claim by using the equations of motion: plug the candidate accelerations in the dynamics, and try to come up with contradictions such as $mg = 0$.

## References


1. Steven H. Collins and Andy Ruina and Russ Tedrake and Martijn Wisse, "Efficient bipedal robots based on passive-dynamic walkers", *Science*, vol. 307, pp. 1082-1085, February 18, 2005. [ link ]

2. Steven H. Collins and Martijn Wisse and Andy Ruina, "A Three-Dimensional Passive-Dynamic Walking Robot with Two Legs and Knees", *International Journal of Robotics Research*, vol. 20, no. 7, pp. 607-615, July, 2001.

3. J.P.Y. Arnould and D.R. Briggs and J.P. Croxall and P.A. Prince and A.G. Wood, "The foraging behaviour and energetics of wandering albatrosses brooding chicks", *Antarctic Science*, vol. 8, no. 3, pp. 229-236, 1996.

4. Wei Shyy and Yongsheng Lian and Jian Teng and Dragos Viieru and Hao Liu, "Aerodynamics of Low Reynolds Number Flyers",Cambridge University Press , 2008.

5. Xiaodong Tian and Jose Iriarte-Diaz and Kevin Middleton and Ricardo Galvao and Emily Israeli and Abigail Roemer and Allyce Sullivan and Arnold Song and Sharon Swartz and Kenneth Breuer, "Direct measurements of the kinematics and dynamics of bat flight", *Bioinspiration \& Biomimetics*, vol. 1, pp. S10-S18, 2006.

6. Michael S. Triantafyllou and George S. Triantafyllou, "An efficient swimming machine", *Scientific American*, vol. 272, no. 3, pp. 64, March, 1995.

7. Vance A. Tucker, "Gliding Flight: Speed and Acceleration of Ideal Falcons During Diving and Pull Out", *Journal of Experimental Biology*, vol. 201, pp. 403-414, Nov, 1998.

8. Eric D. Tytell and George V. Lauder, "Hydrodynamics of the escape response in bluegill sunfish, Lepomis macrochirus", *The Journal of Experimental Biology*, vol. 211, pp. 3359-3369, 2008.

9. D.N. Beal and F.S. Hover and M.S. Triantafyllou and J.C. Liao and G. V. Lauder, "Passive propulsion in vortex wakes", *Journal of Fluid Mechanics*, vol. 549, pp. 385â€“402, 2006.

10. Jean-Jacques E. Slotine and Weiping Li, "Applied Nonlinear Control",Prentice Hall , October, 1990.

11. Anthony Bloch and P. Crouch and J. Baillieul and J. Marsden, "Nonholonomic Mechanics and Control",Springer , April 8, 2003.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 2
# The Simple Pendulum

Open in Colab

## 2.1 INTRODUCTION

Our goals for this chapter are modest: we'd like to understand the dynamics of a pendulum.

Why a pendulum? In part, because the dynamics of a majority of our multi-link robotics manipulators are simply the dynamics of a large number of coupled pendula. Also, the dynamics of a single pendulum are rich enough to introduce most of the concepts from nonlinear dynamics that we will use in this text, but tractable enough for us to (mostly) understand in the next few pages.

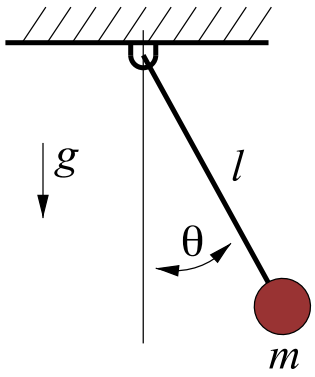



Figure 2.1 - The simple pendulum

The Lagrangian derivation of the equations of motion (as described in the appendix) of the simple pendulum yields:

$$ml^2\ddot{\theta}(t) + mgl\sin\theta(t) = Q.$$

We'll consider the case where the generalized force, $Q$, models a damping torque (from friction) plus a control torque input, $u(t)$:

$$Q = -b\dot{\theta}(t) + u(t).$$

## 2.2 NONLINEAR DYNAMICS WITH A CONSTANT TORQUE

Let us first consider the dynamics of the pendulum if it is driven in a particular simple way: a torque which does not vary with time:

$$ml^2\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = u_0. \tag{1}$$

.

**Example 2.1 (Simple Pendulum in Python)**

You can experiment with this system in DRAKE using


These are relatively simple differential equations, so if I give you $\theta(0)$ and $\dot{\theta}(0)$, then you should be able to integrate them to obtain $\theta(t)$... right? Although it is possible, integrating even the simplest case ($b = u = 0$) involves elliptic integrals of the first kind; there is relatively little intuition to be gained here.

This is in stark contrast to the case of linear systems, where much of our understanding comes from being able to explicitly integrate the equations. For instance, for a simple linear system we have

$$\dot{q} = aq \quad \rightarrow \quad q(t) = q(0)e^{at},$$

and we can immediately understand that the long-term behavior of the system is a (stable) decaying exponential if $a < 0$, an (unstable) growing exponential if $a > 0$, and that the system does nothing if $a = 0$. Here we are with certainly one of the simplest nonlinear systems we can imagine, and we can't even solve this system?

All is not lost. If what we care about is the long-term behavior of the system, then there are a number of techniques we can apply. In this chapter, we will start by investigating graphical solution methods. These methods are described beautifully in a book by Steve Strogatz[1].

### 2.2.1 The overdamped pendulum

Let's start by studying a special case -- intuitively when $b\dot{\theta} \gg ml^2\ddot{\theta}$ -- which via dimensional analysis (using the natural frequency $\sqrt{\frac{g}{l}}$ to match units) occurs when $b\sqrt{\frac{l}{g}} \gg ml^2$. This is the case of heavy damping, for instance if the pendulum was moving in molasses. In this case, the damping term dominates the acceleration term, and we have:

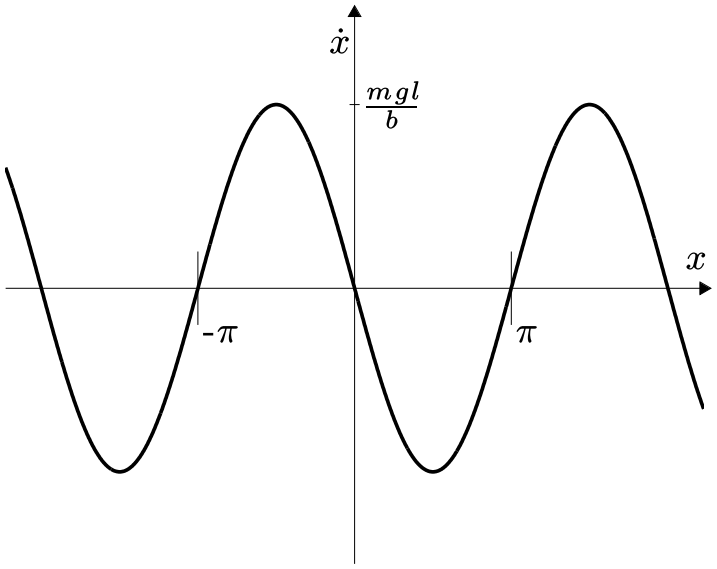
$$ml^2\ddot{\theta} + b\dot{\theta} \approx b\dot{\theta} = u_0 - mgl\sin\theta.$$

In other words, in the case of heavy damping, the system looks approximately first order. This is a general property of heavily-damped systems, such as fluids at very low Reynolds number.

I'd like to ignore one detail for a moment: the fact that $\theta$ wraps around on itself every $2\pi$. To be clear, let's write the system without the wrap-around as:

$$b\dot{x} = u_0 - mgl\sin x. \tag{2}$$

Our goal is to understand the long-term behavior of this system: to find $x(\infty)$ given $x(0)$. Let's start by plotting $\dot{x}$ vs $x$ for the case when $u_0 = 0$:




The first thing to notice is that the system has a number of *fixed points* or *steady states*, which occur whenever $\dot{x} = 0$. In this simple example, the zero-crossings are $x^* = \{\dots, -\pi, 0, \pi, 2\pi, \dots\}$. When the system is in one of these states, it will never leave that state. If the initial conditions are at a fixed point, we know that $x(\infty)$ will be at the same fixed point.

Next let's investigate the behavior of the system in the local vicinity of the fixed points. Examining the fixed point at $x^* = \pi$, if the system starts just to the right of the fixed point, then $\dot{x}$ is positive, so the system will move away from the fixed point. If it starts to the left, then $\dot{x}$ is negative, and the system will move away in the opposite direction. We'll call fixed-points which have this property *unstable*. If we look at the fixed point at $x^* = 0$, then the story is different: trajectories starting to the right or to the left will move back towards the fixed point. We will call this fixed point *locally stable*. More specifically, we'll distinguish between multiple types of stability (where $\epsilon$ is used to denote an arbitrary *small* scalar quantity):

- Locally **stable** in the sense of Lyapunov (i.s.L.). A fixed point, $x^*$ is locally stable i.s.L. if for every $\epsilon > 0$, I can produce a $\delta > 0$ such that if $\|x(0) - x^*\| < \delta$ then $\forall t \ \|x(t) - x^*\| < \epsilon$. In words, this means that for any ball of size $\epsilon$ around the fixed point, I can create a ball of size $\delta$ which guarantees that if the system is started inside the $\delta$ ball then it will remain inside the $\epsilon$ ball for all of time.
- Locally **attractive**. A fixed point is locally attractive if $x(0) = x^* + \epsilon$ implies that $\lim_{t\to\infty} x(t) = x^*$.
- Locally **asymptotically stable**. A fixed point is locally asymptotically stable if it is locally stable i.s.L. and locally attractive.
- Locally **exponentially stable**. A fixed point is locally exponentially stable if $x(0) = x^* + \epsilon$ implies that $\|x(t) - x^*\| < Ce^{-\alpha t}$, for some positive constants $C$ and $\alpha$.
- **Unstable**. A fixed point is unstable if it is not locally stable i.s.L.

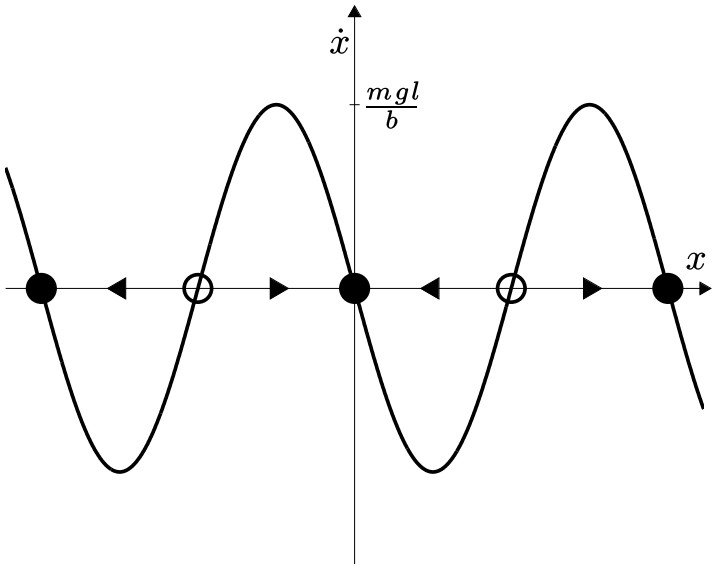
An initial condition near a fixed point that is stable in the sense of Lyapunov may never reach the fixed point (but it won't diverge), near an asymptotically stable fixed point will reach the fixed point as $t \to \infty$, and near an exponentially stable fixed point will reach the fixed point with a bounded rate. An exponentially stable fixed point is also an asymptotically stable fixed point, but the converse is not true. Attractivity does not actually imply Lyapunov stability†, which is why we requires i.s.L. specifically for the definition of asymptotic stability. Systems which are stable i.s.L. but not asymptotically stable are easy to construct (e.g. $\dot{x} = 0$). Interestingly, it is also

† we can't see that in one dimension so will have to hold that

possible to have nonlinear systems that converge (or diverge) in finite-time; a so-called *finite-time stability*; we will see examples of this later in the book, but it is a difficult topic to penetrate with graphical analysis. Rigorous nonlinear system analysis is rich with subtleties and surprises. Moreover, these differences actually matter -- the code that we will write to stabilize the systems will be subtly different depending on what type of stability we want, and it can make or break the success of our methods.

Our graph of $\dot{x}$ vs. $x$ can be used to convince ourselves of i.s.L. and asymptotic stability by visually inspecting $\dot{x}$ in the vicinity of a fixed point. Even exponential stability can be inferred if we can find a negatively-sloped line passing through the equilibrium point which separates the graph of $f(x)$ from the horizontal axis, since it implies that the nonlinear system will converge at least as fast as the linear system represented by the straight line. I will graphically illustrate unstable fixed points with open circles and stable fixed points (i.s.L.) with filled circles.

Next, we need to consider what happens to initial conditions which begin farther from the fixed points. If we think of the dynamics of the system as a flow on the $x$ -axis, then we know that anytime $\dot{x} > 0$, the flow is moving to the right, and $\dot{x} < 0$, the flow is moving to the left. If we further annotate our graph with arrows indicating the direction of the flow, then the entire (long-term) system behavior becomes clear:




For instance, we can see that any initial condition $x(0) \in (-\pi, \pi)$ will result in $\lim_{t\to\infty} x(t) = 0$. This region is called the *basin of attraction* of the fixed point at $x^* = 0$ . Basins of attraction of two fixed points cannot overlap, and the manifold separating two basins of attraction is called the *separatrix*. Here the unstable fixed points, at $x^* = \{\ldots, -\pi, \pi, 3\pi, \ldots\}$ form the separatrix between the basins of attraction of the stable fixed points.

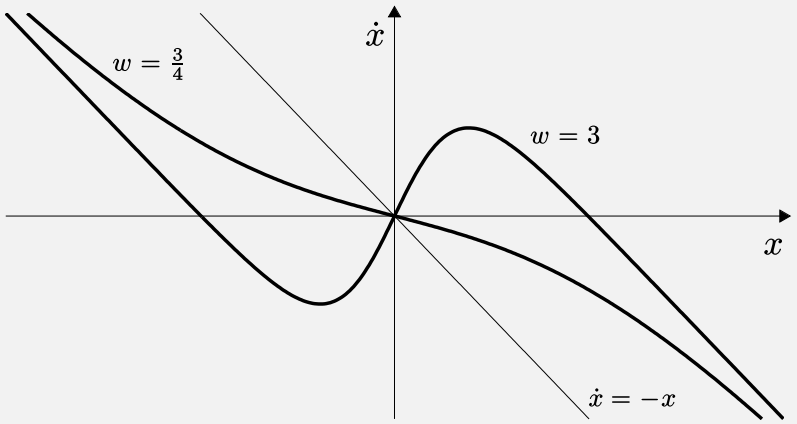
As these plots demonstrate, the behavior of a first-order one dimensional system on a line is relatively constrained. The system will either monotonically approach a fixed-point or monotonically move toward $\pm\infty$. There are no other possibilities. Oscillations, for example, are impossible. Graphical analysis is a fantastic analysis tool for many first-order nonlinear systems (not just pendula); as illustrated by the following example:

## Example 2.2 (Nonlinear autapse)

Consider the following system:

$$\dot{x} + x = \tanh(wx), \tag{3}$$

which is plotted below for two values of $w$. It's convenient to note that $\tanh(z) \approx z$ for small $z$. For $w \leq 1$ the system has only a single fixed point. For $w > 1$ the system has three fixed points : two stable and one unstable.




These equations are not arbitrary - they are actually a model for one of the simplest neural networks, and one of the simplest model of persistent memory[2]. In the equation $x$ models the firing rate of a single neuron, which has a feedback connection to itself. $\tanh$ is the activation (sigmoidal) function of the neuron, and $w$ is the weight of the synaptic feedback.

Experiment with it for yourself:

Open in Colab

As a bonus, you'll also find a the equations of an LSTM unit that you can also experiment with. See if you can figure it out!

One last piece of terminology. In the neuron example, and in many dynamical systems, the dynamics were parameterized; in this case by a single parameter, $w$. As we varied $w$, the fixed points of the system moved around. In fact, if we increase $w$ through $w = 1$, something dramatic happens - the system goes from having one fixed point to having three fixed points. This is called a *bifurcation*. This particular bifurcation is called a pitchfork bifurcation. We often draw bifurcation diagrams which plot the fixed points of the system as a function of the parameters, with solid lines indicating stable fixed points and dashed lines indicating unstable fixed points, as seen in the figure:

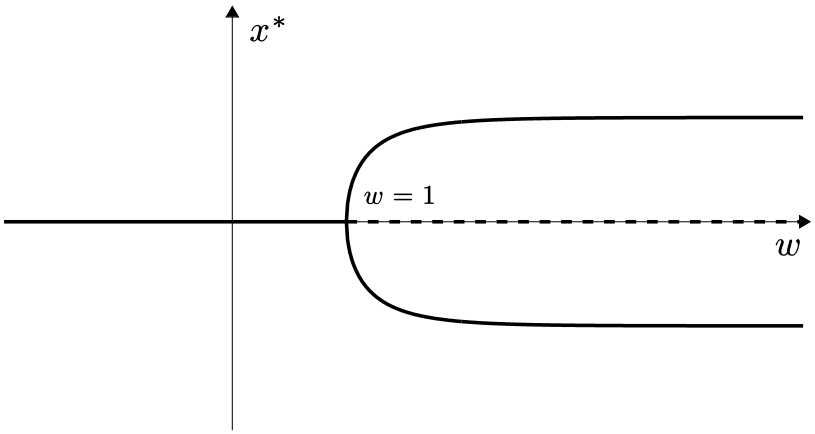



Figure 2.5 - Bifurcation diagram of the nonlinear autapse.

Our pendulum equations also have a (saddle-node) bifurcation when we change the

constant torque input, $u_0$. Finally, let's return to the original equations in $\theta$, instead of in $x$. Only one point to make: because of the wrap-around, this system will *appear* to have oscillations. In fact, the graphical analysis reveals that the pendulum will turn forever whenever $|u_0| > mgl$, but now you understand that this is not an oscillation, but an instability with $\theta \to \pm\infty$.

You can find another beautiful example of these concepts (fixed points, basins of attraction, bifurcations) from recurrent neural networks in the exercise below on Hopfield networks.

## 2.2.2 The undamped pendulum with zero torque

Consider again the system

$$ml^2\ddot{\theta} = u_0 - mgl\sin\theta - b\dot{\theta},$$

this time with $b = 0$. This time the system dynamics are truly second-order. We can always think of any second-order system as (coupled) first-order system with twice as many variables. Consider a general, autonomous (not dependent on time), second-order system,

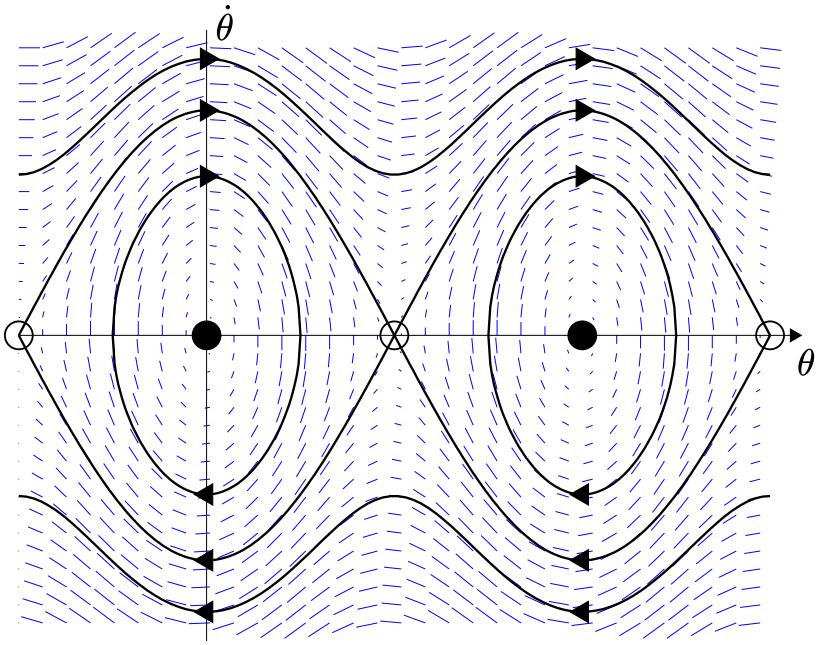
$$\ddot{q} = f(q, \dot{q}, u).$$

This system is equivalent to the two-dimensional first-order system

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2, u),\end{aligned}$$

where $x_1 = q$ and $x_2 = \dot{q}$. Therefore, the graphical depiction of this system is not a line, but a vector field where the vectors $[\dot{x}_1, \dot{x}_2]^T$ are plotted over the domain $(x_1, x_2)$. This vector field is known as the *phase portrait* of the system.

In this section we restrict ourselves to the simplest case when $u_0 = 0$. Let's sketch the phase portrait. First sketch along the $\theta$-axis. The $x$-component of the vector field here is zero, the $y$-component is $-\frac{g}{l}\sin\theta$. As expected, we have fixed points at $\pm\pi, \ldots$ Now sketch the rest of the vector field. Can you tell me which fixed points are stable? Some of them are stable i.s.L., none are asymptotically stable.




## Orbit calculations

You might wonder how we drew the black contour lines in the figure above. We could have obtained them by simulating the system numerically, but those lines can be easily obtained in closed-form. Directly integrating the equations of motion is difficult, but at least for the case when $u_0 = 0$, we have some additional physical insight for this problem that we can take advantage of. The kinetic energy, $T$, and potential energy, $U$, of the pendulum are given by

$$T = \frac{1}{2} I \dot{\theta}^2, \quad U = -mgl\cos(\theta),$$

where $I = ml^2$ and the total energy is $E(\theta, \dot{\theta}) = T(\dot{\theta}) + U(\theta)$. The undamped pendulum is a conservative system: total energy is a constant over system trajectories. Using conservation of energy, we have:

$$\begin{aligned} E(\theta(t), \dot{\theta}(t)) &= E(\theta(0), \dot{\theta}(0)) = E_0 \\ \frac{1}{2} I \dot{\theta}^2(t) - mgl\cos(\theta(t)) &= E_0 \\ \dot{\theta}(t) &= \pm\sqrt{\frac{2}{I}\left[E_0 + mgl\cos\left(\theta(t)\right)\right]} \end{aligned}$$

Using this, if you tell me $\theta$ I can determine one of two possible values for $\dot{\theta}$, and the solution has all of the richness of the black contour lines from the plot. This equation has a real solution when $\cos(\theta) > \cos(\theta_{max})$, where

$$\theta_{max} = \begin{cases} \cos^{-1}\left(-\frac{E_0}{mgl}\right), & E_0 < mgl \\ \pi, & \text{otherwise.} \end{cases}$$

Of course this is just the intuitive notion that the pendulum will not swing above the height where the total energy equals the potential energy. As an exercise, you can verify that differentiating this equation with respect to time indeed results in the equations of motion.

The particular orbit defined by $E = mgl$ is special -- this is the orbit that visits the (unstable) equilibrium at the upright. It is known as the homoclinic orbit.

### 2.2.3 The undamped pendulum with a constant torque

Now what happens if we add a constant torque? If you visualize the bifurcation diagram, you'll see that the fixed points come together, towards $q = \frac{\pi}{2}, \frac{5\pi}{2}, \ldots$, until they disappear. One fixed-point is unstable, and one is stable.

Before we continue, let me now give you the promised example of a system that is not stable i.s.L., but which attracts all trajectories as time goes to infinity. We can accomplish this with a very pendulum-like example (written here in polar coordinates):

**Example 2.3 (Unstable equilibrium point that attracts all trajectories)**

Consider the system

$$\begin{aligned} \dot{r} &= r(1 - r), \\ \dot{\theta} &= \sin^2\left(\frac{\theta}{2}\right). \end{aligned}$$

This system has two equilibrium points, one at $r^* = 0, \theta^* = 0$, and the other at $r^* = 1, \theta^* = 0$. The fixed point at zero is clearly unstable. The fixed point with $r^* = 1$ attracts all other trajectories, but it is not stable by any of our definitions.

Take a minute to draw the vector field of this (you can draw each coordinate independently, if it helps) to make sure you understand. Note that to wrap-around rotation is convenient but not essential -- we could have written the same dynamical system in cartesian coordinates without wrapping. And if this still feels too arbitrary, we will see it happen in practice when we introduce the energy-shaping swing-up controller for the pendulum in the next chapter.

### 2.2.4 The damped pendulum

Now let's add damping back. You can still add torque to move the fixed points (in the same way).

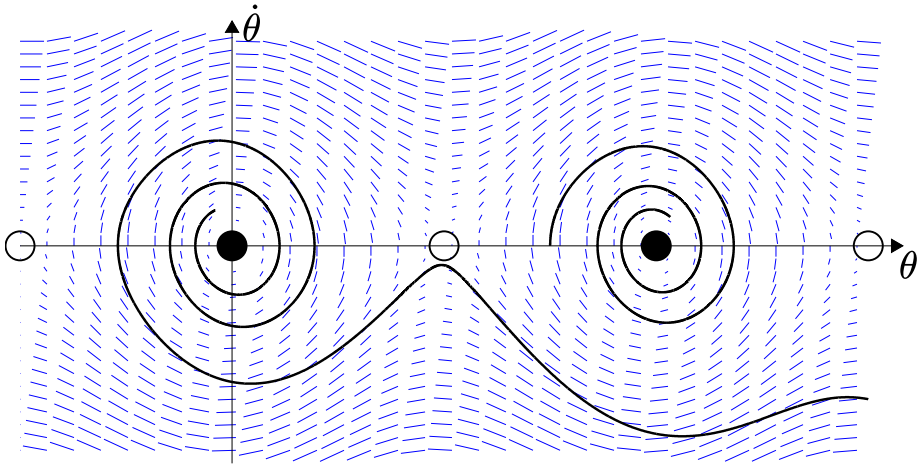



Figure 2.7 - Phase diagram for the damped pendulum

With damping, the downright fixed point of the pendulum now becomes asymptotically stable (in addition to stable i.s.L.). Is it also exponentially stable? How can we tell? One technique is to linearize the system at the fixed point. A smooth, time-invariant, nonlinear system that is locally exponentially stable *must* have a stable linearization; we'll discuss linearization more in the next chapter.

Here's a thought exercise. If $u$ is no longer a constant, but a function $\pi(\theta, \dot{\theta})$, then how would you choose $\pi$ to stabilize the vertical position. Feedback cancellation is the trivial solution, for example:

$$u = \pi(\theta, \dot{\theta}) = 2mgl \sin\theta.$$

But these plots we've been making tell a different story. How would you shape the natural dynamics - at each point pick a $u$ from the stack of phase plots - to stabilize the vertical fixed point *with minimal torque effort*? This is exactly the way that I would like you to think about control system design. And we'll give you your first solution techniques -- using dynamic programming -- in the next lecture.

## 2.3 THE TORQUE-LIMITED SIMPLE PENDULUM

The simple pendulum is fully actuated. Given enough torque, we can produce any number of control solutions to stabilize the originally unstable fixed point at the top (such as designing a feedback controller to effectively invert gravity).

The problem begins to get interesting (a.k.a. becomes underactuated) if we impose a torque-limit constraint, $|u| \leq u_{max}$. Looking at the phase portraits again, you can now visualize the control problem. Via feedback, you are allowed to change the direction of the vector field at each point, but only by a fixed amount. Clearly, if the maximum torque is small (smaller than $mgl$), then there are some states which cannot be driven directly to the goal, but must pump up energy to reach the goal. Furthermore, if the

torque-limit is too severe and the system has damping, then it may be impossible to swing up to the top. The existence of a solution, and number of pumps required to reach the top, is a non-trivial function of the initial conditions and the torque-limits.

Although this system is very simple, obtaining a solution with general-purpose algorithms requires much of the same reasoning necessary for controlling much more complex underactuated systems; this problem will be a work-horse for us as we introduce new algorithms throughout this book.

### 2.3.1 Energy-shaping control

In the specific case of the pendulum, we can give a satisfying hand-designed nonlinear controller based on our intuition of pumping energy into the system. We have already observed that the total energy of the pendulum is given by

$$E = \frac{1}{2}ml^2\dot{\theta}^2 - mgl\cos\theta.$$

To understand how to control the energy, let us examine how that energy changes with time:

$$\begin{aligned}\dot{E} =& ml^2\dot{\theta}\ddot{\theta} + \dot{\theta}mgl\sin\theta \\ =& \dot{\theta}\left[u - mgl\sin\theta\right] + \dot{\theta}mgl\sin\theta \\ =& u\dot{\theta}.\end{aligned}$$

In words, adding energy to the system is simple - apply torque in the same direction as $\dot{\theta}$. To remove energy, apply torque in the opposite direction (e.g., damping).

To swing up the pendulum, even with torque limits, let us use this observation to drive the system to its homoclinic orbit, and then let the dynamics of the pendulum carry us to the upright equilibrium. Recall that the homoclinic orbit has energy $mgl$ -- let's call this our *desired* energy:

$$E^d = mgl.$$

Furthermore, let's define the difference between our current energy and this desired energy as $\tilde{E} = E - E^d$, and note that we still have

$$\dot{\tilde{E}} = \dot{E} = u\dot{\theta}.$$

Now consider the feedback controller of the form

$$u = -k\dot{\theta}\tilde{E}, \quad k > 0.$$

I admit that this looks a bit strange; it was chosen for the simple reason that it turns the resulting "error dynamics" into something simple:

$$\dot{\tilde{E}} = -k\dot{\theta}^2\tilde{E}.$$

Think about the graphical analysis of this system if you were to draw $\dot{\tilde{E}}$ vs. $\tilde{E}$ for any fixed $\dot{\theta}$ -- it's a straight line separated from the horizontal axis which would imply an exponential convergence: $\tilde{E} \to 0$. This is true for any $\dot{\theta}$, except for $\dot{\theta} = 0$ (so it will not actually swing us up from the downright fixed point... but if you nudge the system just a bit, then it will start pumping energy and will swing all of the way up). The essential property is that when $E > E^d$, we should remove energy from the system (damping) and when $E < E^d$, we should add energy (negative damping). Even if the control actions are bounded, the convergence is easily preserved.

This is a nonlinear controller that will push all system trajectories to the unstable equilibrium. But does it make the unstable equilibrium locally stable? With only this controller, the fixed point is *attractive, but is not stable* -- just like our example above. Small perturbations may cause the system to drive all of the way around the

circle in order to once again return to the unstable equilibrium. For this reason, to actually balance the system, we'll have to switch to a different controller once we get near the fixed point (an idea that we'll discuss more in the next chapter).

### Example 2.4 (Energy Shaping for the Pendulum)

Take a minute to play around with the energy-shaping controller for swinging up the pendulum

Open in Colab

Make sure that you take a minute to look at the code which runs during these examples. Note the somewhat arbitrary threshold for switching to the balancing controller. We'll give a much more satisfying answer for that in the chapter on Lyapunov methods.

There are a few things that are extremely nice about this controller. Not only is it simple, but it is actually incredibly robust to errors we might have in our estimate of the model parameters, $m$, $l$, and $g$. In fact, the only place that the model enters our control equation is in the calculation of $\tilde{E} = \frac{1}{2}ml^2\dot{\theta}^2 - mgl(1+\cos\theta)$, and the only property of this estimate that impacts stability is the location of the orbit when $\tilde{E} = 0$, which is $\frac{1}{2}l\dot{\theta}^2 = g(\cos\theta + 1)$. This doesn't depend at all on our estimate of the mass, and only linearly on our estimates of the length and gravity (and if one cannot measure the length of the pendulum accurately, then a proper measuring device would make an excellent investment). This is somewhat amazing; we will develop many optimization-based algorithms capable of swinging up the pendulum, but it will take a lot of work to find one that is as insensitive to the model parameters.

If there is damping in the original system, of course we can cancel this out, too, using $u = -k\dot{\theta}\tilde{E} + b\dot{\theta}$. And once again, the controller is quite robust if we have errors in the estimated damping ratio.

## 2.4 Exercises

### Exercise 2.1 (Graphical Analysis)

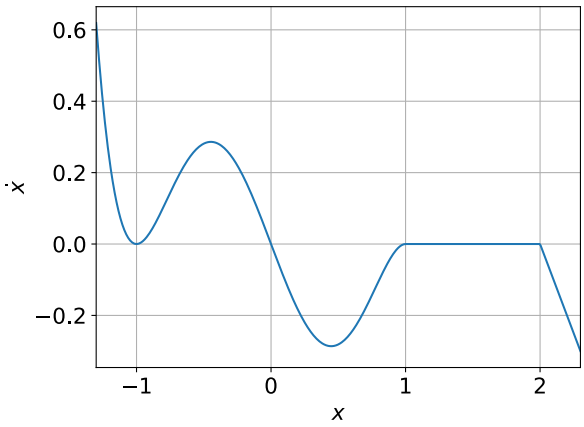



Figure 2.8 - First-order systems with multiple equilibria.

Consider the first-order system

$$\dot{x} = \begin{cases} -x^5 + 2x^3 - x & \text{if} \quad x \leq 1, \\ 0 & \text{if} \quad 1 < x \leq 2, \\ -x + 2 & \text{if} \quad x > 2, \end{cases}$$

whose dynamics is represented in the figure above. Notice that, together with $x^* = -1$ and $x^* = 0$, all the points in the closed interval $[1, 2]$ are equilibria for this system. For each equilibrium point, determine whether it is unstable, stable i.s.L., asymptotically stable, or exponentially stable.

## Exercise 2.2 (Basin of Attraction and Bifurcation Diagram)

Consider the first-order system with polynomial dynamics

$$\dot{x} = f(x) = -x^3 + 4x^2 + 11x - 30.$$

a. Sketch the graph of the function $f(x)$, and identify all the equilibrium points. (Feel free to plot this with a tool of your choice to check your work.)
b. For each equilibrium point, determine whether it is stable (i.s.L.) or unstable. For each one of the stable equilibria, identify the basin of attraction.
c. Consider an additive term $w$ in the system dynamics: $\dot{x} = f(x) + w$. As $w$ ranges from $0$ to $\infty$, determine how many stable and unstable equilibrium points the system has. Support your answer with a sketch of the bifurcation diagram for nonnegative values of $w$.

## Exercise 2.3 (Enumerate Unstable Equilibria)

We are given a dynamical system $\dot{x} = f(x)$ with a scalar state $x$. The dynamics $f(x)$ is unknown, but we are given two pieces of information:

- the function $f(x)$ is continuous;
- the system has exactly three stable (i.s.L.) equilibrium points.

Identify the minimum $n_{\min}$ and the maximum $n_{\max}$ number of unstable equilibria that the system can have. Support your answer with the sketch of two functions $f_{\min}(x)$ and $f_{\max}(x)$ that verify the requirements above and have, respectively, $n_{\min}$ and $n_{\max}$ unstable equilibria.

## Exercise 2.4 (Attractivity vs Stability)

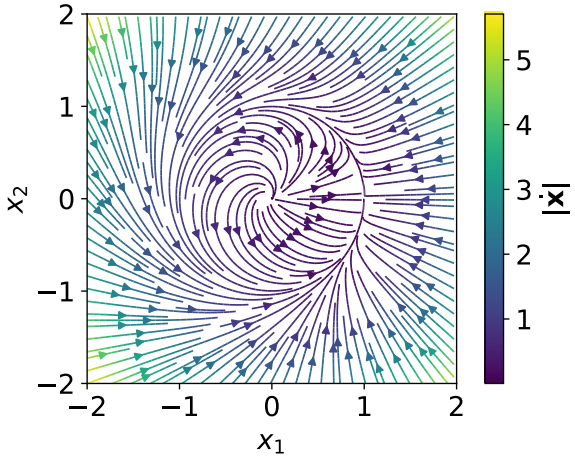



Figure 2.9 - Phase diagram of a two-dimensional system.
This figure shows the phase portrait of the dynamical system

$$\dot{x}_1 = x_1(1 - |\mathbf{x}|) + x_2 \frac{x_1 - |\mathbf{x}|}{2|\mathbf{x}|},$$
$$\dot{x}_2 = x_2(1 - |\mathbf{x}|) - x_1 \frac{x_1 - |\mathbf{x}|}{2|\mathbf{x}|},$$

where $|\mathbf{x}| = \sqrt{x_1^2 + x_2^2}$. To help you with the analysis of this system, we set up this python notebook. Take your time to understand the code in it, and then answer the following questions.

a. Find all the equilibrium points of this system (no need to look outside the portion of state space depicted above). Use the notebook to double check your answer: simulate the evolution of the system setting the equilibrium points you identified as initial conditions.
b. Determine whether the equilibria you identified at the previous point are attractive and/or stable i.s.L. Explain briefly your reasoning, and feel free to include some plots generated with the notebook in your written answer.
c. This dynamical system is a (very) close relative of one of the systems we analyzed in this chapter. Can you guess which one is it? What is the connection between the two? Extra points: support your claim with a mathematical derivation.

## Exercise 2.5 (Pendulum with Vibrating Base)

Consider an actuated pendulum whose base (pivot of the rod) is forced to oscillate horizontally according to the harmonic law $h\sin(\omega t)$, where $h$ denotes the amplitude of the oscillation and $\omega$ the frequency. The equation of motion for this system is

$$ml^2\ddot{\theta} + mgl\sin\theta = ml\omega^2 h\sin(\omega t)\cos\theta + u.$$

(If this equation seems obscure to you, try to derive it using the Lagrangian approach; but be careful, the kinetic energy $T(\theta, \dot{\theta}, t)$ of this system depends explicitly on time.) Our goal is to design a time-dependent control law $u = \pi(\theta, \dot{\theta}, t)$ that makes the pendulum spin at constant velocity $\dot{\theta} = 1$.

a. Identify a desired closed-loop dynamics $\ddot{\theta} = f(\dot{\theta})$, whose unique

equilibrium point is stable and is located in $\dot{\theta} = 1$.

b. Use feedback cancellation to design a control law $u = \pi(\theta, \dot{\theta}, t)$ that makes the closed-loop dynamics coincide with the one chosen at the previous point.

c. In this python notebook, we set up a simulation environment to test your control law. Try to understand the structure of the code: this workflow is quite general, and it could be a good starting point for your future Drake projects. Implement your control law in the dedicated cell, and check your work using the animation and the plots at the end of the notebook.

## Exercise 2.6 (Hopfield Networks)

Consider the following dynamical system

$$\dot{x} = A^T \operatorname{softmax}(\beta A x) - x \tag{4}$$

where $A \in \mathbb{R}^{m \times n}$ and $\beta \in \mathbb{R}$ are a constant matrix and a scalar, respectively, and the softmax function is given by $\operatorname{softmax}(z)_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$.

a. Draw the phase portrait of the system with constants $\beta = 5$ and $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}$.

b. Identify (approximately by looking at the phase portrait) equilibriums of the above system. What are the stable and unstable equilibriums?

c. How are the stable equilibriums related to the matrix $A$?

d. Using this python notebook, implement an image recovery algorithm using Hopfield network. Modify the dynamical system above to memorize a subset of MNIST data set. Using a corrupted image as the initial condition, simulate the dynamical system to restore images.

## Exercise 2.7 (Graphical Analysis of a Neural ODE)

We've recently seen a trend in using neural networks to describe our dynamics function. The differential equation represented by the network can be very complex, and hard to analyze rigorously. However, in the special case of one-dimensional systems we can use graphical analysis to analyze our system. For this exercise, you'll take a look at trying to learn the dynamics of the damped pendulum from data. This exercise has a coding component where you'll learn the basics of training a neural network in pytorch. There is then a written component where you'll perform a graphical analysis of the dynamics function represented by your trained network.

a. In this python notebook, learn the dynamics of the damped pendulum from data.

b. Within the range $\theta \in [-5.0, 5.0]$, which values of $(\theta, \dot{\theta})$ represent fixed points? Which fixed points are stable and which are unstable?

c. What is one reason why our model might not fit the ground truth data exactly?

d. Does our model give us a reasonable approximation of the dynamics for all $(\theta, \dot{\theta})$ pairs? Why or why not?

## References


1. Steven H. Strogatz, "Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering",Perseus Books , 1994.

2. H. Sebastian Seung and Daniel D. Lee and Ben Y. Reis and David W. Tank, "The autapse: a simple illustration of short-term analog memory storage by tuned synaptic feedback", *Journal of Computational Neuroscience*, vol. 9, pp. 171-85, 2000.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 3
# Acrobots, Cart-Poles and Quadrotors

Open in Colab

A great deal of work in the control of underactuated systems has been done in the context of low-dimensional model systems. These model systems capture the essence of the problem without introducing all of the complexity that is often involved in more real-world examples. In this chapter we will focus on two of the most well-known and well-studied model systems--the Acrobot and the Cart-Pole. After we have developed some tools, we will see that they can be applied directly to other model systems; we will give a number of examples using Quadrotors. All of these systems are trivially underactuated, having less actuators than degrees of freedom.

## 3.1 The Acrobot

The Acrobot is a planar two-link robotic arm in the vertical plane (working against gravity), with an actuator at the elbow, but no actuator at the shoulder. It was first described in detail in [1]. The companion system, with an actuator at the shoulder but not at the elbow, is known as the Pendubot[2]. The Acrobot is so named because of its resemblance to a gymnast (or acrobat) on a parallel bar, who controls his/her motion predominantly by effort at the waist (and not effort at the wrist). The most common control task studied for the Acrobot is the swing-up task, in which the system must use the elbow (or waist) torque to move the system into a vertical configuration then balance.

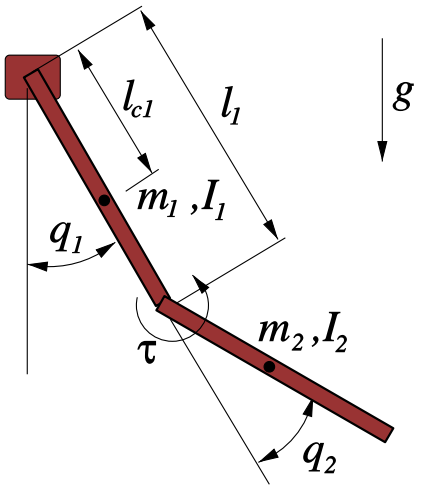



Figure 3.1 - The Acrobot. Click here to see a physical Acrobot swing up and balance.

The Acrobot is representative of the primary challenge in underactuated robots. In order to swing up and balance the entire system, the controller must reason about and exploit the state-dependent coupling between the actuated degree of freedom and the unactuated degree of freedom. It is also an important system because, as we will see, it closely resembles one of the simplest models of a walking robot.

### 3.1.1 Equations of motion

Figure 3.1 illustrates the model parameters used in our analysis. $\theta_1$ is the shoulder joint angle, $\theta_2$ is the elbow (relative) joint angle, and we will use $\mathbf{q} = [\theta_1, \theta_2]^T$, $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$. The zero configuration is with both links pointed directly down. The moments of inertia, $I_1, I_2$ are taken about the pivots. The task is to stabilize the unstable fixed point $\mathbf{x} = [\pi, 0, 0, 0]^T$.

We will derive the equations of motion for the Acrobot using the method of Lagrange. The locations of the center of mass of each link, $\mathbf{p}_{c1}, \mathbf{p}_{c2}$, are given by the kinematics:

$$\mathbf{p}_{c1} = \begin{bmatrix} l_{c1} s_1 \\ -l_{c1} c_1 \end{bmatrix}, \quad \mathbf{p}_{c2} = \begin{bmatrix} l_1 s_1 + l_{c2} s_{1+2} \\ -l_1 c_1 - l_{c2} c_{1+2} \end{bmatrix}, \tag{1}$$

where $s_1$ is shorthand for $\sin(\theta_1)$, $c_{1+2}$ is shorthand for $\cos(\theta_1 + \theta_2)$, etc. The energy is given by:

$$T = T_1 + T_2, \quad T_1 = \frac{1}{2} I_1 \dot{q}_1^2 \tag{2}$$

$$T_2 = \frac{1}{2}(m_2 l_1^2 + I_2 + 2 m_2 l_1 l_{c2} c_2) \dot{q}_1^2 + \frac{1}{2} I_2 \dot{q}_2^2 + (I_2 + m_2 l_1 l_{c2} c_2) \dot{q}_1 \dot{q}_2 \tag{3}$$

$$U = -m_1 g l_{c1} c_1 - m_2 g (l_1 c_1 + l_{c2} c_{1+2}) \tag{4}$$

Entering these quantities into the Lagrangian yields the equations of motion:

$$(I_1 + I_2 + m_2 l_1^2 + 2 m_2 l_1 l_{c2} c_2) \ddot{q}_1 + (I_2 + m_2 l_1 l_{c2} c_2) \ddot{q}_2 - 2 m_2 l_1 l_{c2} s_2 \dot{q}_1 \dot{q}_2 \tag{5}$$

$$-m_2 l_1 l_{c2} s_2 \dot{q}_2^2 + m_1 g l_{c1} s_1 + m_2 g (l_1 s_1 + l_{c2} s_{1+2}) = 0 \tag{6}$$

$$(I_2 + m_2 l_1 l_{c2} c_2) \ddot{q}_1 + I_2 \ddot{q}_2 + m_2 l_1 l_{c2} s_2 \dot{q}_1^2 + m_2 g l_{c2} s_{1+2} = \tau \tag{7}$$

In standard, manipulator equation form:

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u},$$

using $\mathbf{q} = [\theta_1, \theta_2]^T$, $\mathbf{u} = \tau$ we have:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2 m_2 l_1 l_{c2} c_2 & I_2 + m_2 l_1 l_{c2} c_2 \\ I_2 + m_2 l_1 l_{c2} c_2 & I_2 \end{bmatrix}, \tag{8}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -2 m_2 l_1 l_{c2} s_2 \dot{q}_2 & -m_2 l_1 l_{c2} s_2 \dot{q}_2 \\ m_2 l_1 l_{c2} s_2 \dot{q}_1 & 0 \end{bmatrix}, \tag{9}$$

$$\tau_g(\mathbf{q}) = \begin{bmatrix} -m_1 g l_{c1} s_1 - m_2 g (l_1 s_1 + l_{c2} s_{1+2}) \\ -m_2 g l_{c2} s_{1+2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{10}$$

**Example 3.1 (Dynamics of the Acrobot)**

You can experiment with the Acrobot dynamics in DRAKE using, e.g.


## 3.2 THE CART-POLE SYSTEM

The other model system that we will investigate here is the cart-pole system, in which the task is to balance a simple pendulum around its unstable equilibrium, using only horizontal forces on the cart. Balancing the cart-pole system is used in many introductory courses in control, including 6.003 at MIT, because it can be accomplished with simple linear control (e.g. pole placement) techniques. In this chapter we will consider the full swing-up and balance control problem, which requires a full nonlinear control treatment.

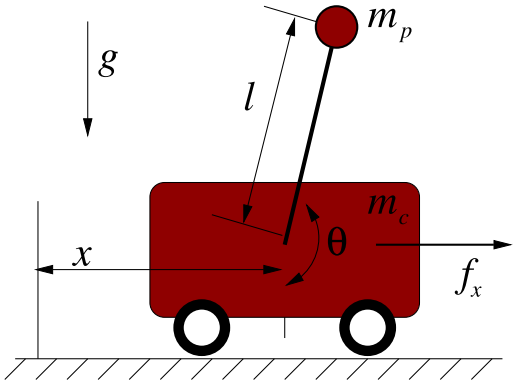



Figure 3.2 - The Cart-Pole system. Click here to see a real robot.

The figure shows our parameterization of the system. $x$ is the horizontal position of the cart, $\theta$ is the counter-clockwise angle of the pendulum (zero is hanging straight down). We will use $\mathbf{q} = [x, \theta]^T$, and $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$. The task is to stabilize the unstable fixed point at $\mathbf{x} = [0, \pi, 0, 0]^T$.

### 3.2.1 Equations of motion

The kinematics of the system are given by

$$\mathbf{x}_1 = \begin{bmatrix} x \\ 0 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} x + l \sin\theta \\ -l \cos\theta \end{bmatrix}. \tag{11}$$

The energy is given by

$$T = \frac{1}{2}(m_c + m_p)\dot{x}^2 + m_p \dot{x}\dot{\theta} l \cos\theta + \frac{1}{2} m_p l^2 \dot{\theta}^2 \tag{12}$$

$$U = -m_p g l \cos\theta. \tag{13}$$

The Lagrangian yields the equations of motion:

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2 \sin\theta = f_x \tag{14}$$

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p g l \sin\theta = 0 \tag{15}$$

In standard, manipulator equation form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u},$$

using $\mathbf{q} = [x, \theta]^T$, $\mathbf{u} = f_x$, we have:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_c + m_p & m_p l\cos\theta \\ m_p l\cos\theta & m_p l^2 \end{bmatrix}, \quad \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \begin{bmatrix} 0 & -m_p l\dot{\theta}\sin\theta \\ 0 & 0 \end{bmatrix},$$

$$\tau_g(\mathbf{q}) = \begin{bmatrix} 0 \\ -m_p g l \sin\theta \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In this case, it is particularly easy to solve directly for the accelerations:

$$\ddot{x} = \frac{1}{m_c + m_p \sin^2\theta}\left[f_x + m_p \sin\theta(l\dot{\theta}^2 + g\cos\theta)\right] \tag{16}$$

$$\ddot{\theta} = \frac{1}{l(m_c + m_p\sin^2\theta)}\left[-f_x\cos\theta - m_p l\dot{\theta}^2\cos\theta\sin\theta - (m_c + m_p)g\sin\theta\right] \tag{17}$$

In some of the analysis that follows, we will study the form of the equations of motion, ignoring the details, by arbitrarily setting all constants to 1:

$$2\ddot{x} + \ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta = f_x \tag{18}$$

$$\ddot{x}\cos\theta + \ddot{\theta} + \sin\theta = 0. \tag{19}$$

**Example 3.2 (Dynamics of the Cart-Pole System)**

You can experiment with the Cart-Pole dynamics in DRAKE using, e.g.


## 3.3 QUADROTORS

Quadrotors have become immensely popular over the last few years -- advances in outrunner motors from the hobby community made them powerful, light-weight, and inexpensive! They are strong enough to carry an interesting payload (e.g. of sensors for mapping / photographing the environment), but dynamic enough to move relatively quickly. The most interesting dynamics start showing up at higher speeds, when the propellors start achieving lift from the airflow across them due to horizontal motion, or when they are close enough to the ground to experience significant ground-effect, but we won't try to capture those effects (yet) here.

When the quadrotor revolution started to happen, I predicted that it would be followed quickly by a realization that fixed-wing vehicles are better for most applications. Propellors are almost optimally efficient for producing thrust -- making quadrotors very efficient for hovering -- but to be efficient in forward flight you probably want to have an airfoil. Wings are a really good idea! But I was wrong -- quadrotors have completely dominated fixed-wings for commercial UAVs. Perhaps it's only because they are easier to control? Maybe there is still hope...

### 3.3.1 The Planar Quadrotor

We can get started with an extremely simple model of a quadrotor that is restricted to live in the plane. In that case, it actually only needs two propellors, but calling it a "birotor" doesn't have the same ring to it. The equations of motion are almost trivial, since it is only a single rigid body, and certainly fit into our standard manipulator equations:

$$m\ddot{x} = -(u_1 + u_2)\sin\theta, \tag{20}$$
$$m\ddot{y} = (u_1 + u_2)\cos\theta - mg, \tag{21}$$
$$I\ddot{\theta} = r(u_1 - u_2) \tag{22}$$

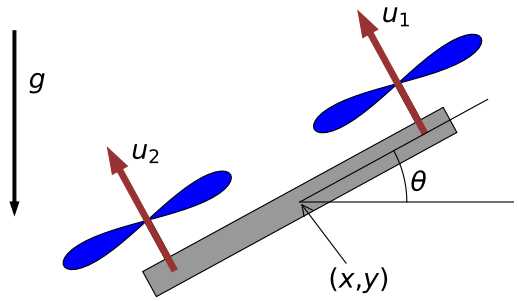



Figure 3.3 - The Planar Quadrotor System (which we also refer to in the code as "Quadrotor2D", to keep it next to "Quadrotor" in the directory listing). The model parameters are mass, $m$, moment of inertia, $I$, and the distance from the center to the base of the propellor, $r$.

### 3.3.2 The Full 3D Quadrotor

See `drake/examples/Quadrotor`. Will fill in the text description/derivation here soon! The most interesting part is that the model needs to include the moment produced by the rotating props, otherwise the system linearized about the hovering configuration is actually not controllable.

## 3.4 BALANCING

For both the Acrobot and the Cart-Pole systems, we will begin by designing a linear controller which can balance the system when it begins in the vicinity of the unstable fixed point. To accomplish this, we will linearize the nonlinear equations about the fixed point, examine the controllability of this linear system, then using linear quadratic regulator (LQR) theory to design our feedback controller.

### 3.4.1 Linearizing the manipulator equations

Although the equations of motion of both of these model systems are relatively tractable, the forward dynamics still involve quite a few nonlinear terms that must be considered in any linearization. Let's consider the general problem of linearizing a system described by the manipulator equations.

We can perform linearization around a fixed point, $(\mathbf{x}^*, \mathbf{u}^*)$, using a Taylor expansion:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*} (\mathbf{x} - \mathbf{x}^*) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right]_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*} (\mathbf{u} - \mathbf{u}^*) \tag{23}$$

Let us consider the specific problem of linearizing the manipulator equations around

a (stable or unstable) fixed point. In this case, $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*)$ is zero, and we are left with the standard linear state-space form:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{M}^{-1}(\mathbf{q})\left[\tau_g(\mathbf{q}) + \mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}\right] \end{bmatrix}, \tag{24}$$

$$\approx \mathbf{A}_{lin}(\mathbf{x} - \mathbf{x}^*) + \mathbf{B}_{lin}(\mathbf{u} - \mathbf{u}^*), \tag{25}$$

where $\mathbf{A}_{lin}$, and $\mathbf{B}_{lin}$ are constant matrices. Let us define $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}^*, \bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}^*$, and write

$$\dot{\bar{\mathbf{x}}} = \mathbf{A}_{lin}\bar{\mathbf{x}} + \mathbf{B}_{lin}\bar{\mathbf{u}}.$$

Evaluation of the Taylor expansion around a fixed point yields the following, very simple equations, given in block form by:

$$\mathbf{A}_{lin} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{M}^{-1}\frac{\partial \tau_g}{\partial \mathbf{q}} + \sum_j \mathbf{M}^{-1}\frac{\partial \mathbf{B}_j}{\partial \mathbf{q}} u_j & \mathbf{0} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*} \tag{26}$$

$$\mathbf{B}_{lin} = \begin{bmatrix} \mathbf{0} \\ \mathbf{M}^{-1}\mathbf{B} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*} \tag{27}$$

where $\mathbf{B}_j$ is the $j$th column of $\mathbf{B}$. Note that the term involving $\frac{\partial \mathbf{M}^{-1}}{\partial q_i}$ disappears because $\tau_g + \mathbf{B}\mathbf{u} - \mathbf{C}\dot{\mathbf{q}}$ must be zero at the fixed point. All of the $\mathbf{C}\dot{\mathbf{q}}$ derivatives drop out, too, because $\dot{\mathbf{q}}^* = 0$, and any terms with $\mathbf{C}$ drop out as well, since centripetal and centrifugal forces are zero at when velocity is zero. In many cases, including both the Acrobot and Cart-Pole systems (but not the Quadrotors), the matrix $\mathbf{B}(\mathbf{q})$ is a constant, so the $\frac{\partial \mathbf{B}}{\partial \mathbf{q}}$ terms also drop out.

## Example 3.3 (Linearization of the Acrobot)

Linearizing around the (unstable) upright point, we have:

$$\left[\frac{\partial \tau_\mathbf{g}}{\partial \mathbf{q}}\right]_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} g(m_1 l_{c1} + m_2 l_1 + m_2 l_{c2}) & m_2 g l_{c2} \\ m_2 g l_{c2} & m_2 g l_{c2} \end{bmatrix} \tag{28}$$

The linear dynamics follow directly from these equations and the manipulator form of the Acrobot equations.

## Example 3.4 (Linearization of the Cart-Pole System)

Linearizing around the unstable fixed point in this system, we have:

$$\left[\frac{\partial \tau_g}{\partial \mathbf{q}}\right]_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} 0 & 0 \\ 0 & m_p g l \end{bmatrix} \tag{29}$$

Again, the linear dynamics follow simply.

Studying the properties of the linearized system can tell us some things about the (local) properties of the nonlinear system. For instance, having a strictly stable linearization implies local exponential stability of the nonlinear system [3] (Theorem 4.15). It's worth noting that having an unstable linearization also implies that the system is locally unstable, but if the linearization is marginally stable then one cannot conclude whether the nonlinear system is asymptotically stable, stable i.s.L., or unstable (only that it is not exponentially stable)[4].

## 3.4.2 Controllability of linear systems

**Definition 3.1 (Controllability)** A control system of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

is called *controllable* if it is possible to construct an unconstrained input signal, $\mathbf{u}(t)$, $t \in [0, t_f]$, which will move the system from any initial state to any final state in a finite interval of time, $0 < t < t_f$ [5].

For the linear system

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

we can integrate this linear system in closed form, so it is possible to derive the exact conditions of controllability. In particular, for linear systems it is sufficient to demonstrate that there exists a control input which drives any initial condition to the origin.

### The special case of non-repeated eigenvalues

Let us first examine a special case, which falls short as a general tool but may be more useful for understanding the intuition of controllability. Let's perform an eigenvalue analysis of the system matrix $\mathbf{A}$, so that:

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i,$$

where $\lambda_i$ is the $i$th eigenvalue, and $\mathbf{v}_i$ is the corresponding (right) eigenvector. There will be $n$ eigenvalues for the $n \times n$ matrix $\mathbf{A}$. Collecting the (column) eigenvectors into the matrix $\mathbf{V}$ and the eigenvalues into a diagonal matrix $\mathbf{\Lambda}$, we have

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Lambda}.$$

Here comes our primary assumption: let us assume that each of these $n$ eigenvalues takes on a distinct value (no repeats). With this assumption, it can be shown that the eigenvectors $\mathbf{v}_i$ form a linearly independent basis set, and therefore $\mathbf{V}^{-1}$ is well-defined.

We can continue our eigenmodal analysis of the linear system by defining the modal coordinates, $\mathbf{r}$, with:

$$\mathbf{x} = \mathbf{V}\mathbf{r}, \quad \text{or} \quad \mathbf{r} = \mathbf{V}^{-1}\mathbf{x}.$$

In modal coordinates, the dynamics of the linear system are given by

$$\dot{\mathbf{r}} = \mathbf{V}^{-1}\mathbf{A}\mathbf{V}\mathbf{r} + \mathbf{V}^{-1}\mathbf{B}\mathbf{u} = \mathbf{\Lambda}\mathbf{r} + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}.$$

This illustrates the power of modal analysis; in modal coordinates, the dynamics diagonalize yielding independent linear equations:

$$\dot{r}_i = \lambda_i r_i + \sum_j \beta_{ij} u_j, \quad \beta = \mathbf{V}^{-1}\mathbf{B}.$$

Now the concept of controllability becomes clear. Input $j$ can influence the dynamics in modal coordinate $i$ if and only if $\beta_{ij} \neq 0$. In the special case of non-repeated eigenvalues, having control over each individual eigenmode is sufficient to (in finite time) regulate all of the eigenmodes[5]. Therefore, we say that the system is controllable if and only if

$$\forall i, \exists j \text{ such that } \beta_{ij} \neq 0.$$

**A general solution** Included only for completeness. Click to expand the details.

A more general solution to the controllability issue, which removes our assumption about the eigenvalues, can be obtained by examining the time-domain solution of the linear equations. The solution of this system is

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)d\tau.$$

Without loss of generality, lets consider the that the final state of the system is zero. Then we have:

$$\mathbf{x}(0) = -\int_0^{t_f} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{u}(\tau)d\tau.$$

You might be wondering what we mean by $e^{\mathbf{A}t}$; a scalar raised to the power of a matrix..? Recall that $e^z$ is actually defined by a convergent infinite sum:

$$e^z = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \ldots.$$

The notation $e^{\mathbf{A}t}$ uses the same definition:

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{1}{2}(\mathbf{A}t)^2 + \frac{1}{6}(\mathbf{A}t)^3 + \ldots.$$

Not surprisingly, this has many special forms. For instance, if $\mathbf{A}$ is diagonalizable, $e^{\mathbf{A}t} = \mathbf{V}e^{\mathbf{\Lambda}t}\mathbf{V}^{-1}$, where $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ is the eigenvalue decomposition of $\mathbf{A}$ [6]. The particular form we will use here is

$$e^{-\mathbf{A}\tau} = \sum_{k=0}^{n-1} \alpha_k(\tau)\mathbf{A}^k.$$

This is a particularly surprising form, because the infinite sum above is represented by this finite sum; the derivation uses Sylvester's Theorem [5], [7]. Then we have,

$$\begin{aligned}
\mathbf{x}(0) &= -\sum_{k=0}^{n-1} \mathbf{A}^k\mathbf{B}\int_0^{t_f} \alpha_k(\tau)\mathbf{u}(\tau)d\tau \\
&= -\sum_{k=0}^{n-1} \mathbf{A}^k\mathbf{B}w_k, \text{ where } \mathbf{w}_k = \int_0^{t_f} \alpha_k(\tau)\mathbf{u}(\tau)d\tau \\
&= -\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}_{n\times(nm)} \begin{bmatrix}\mathbf{w}_0 \\ \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_{n-1}\end{bmatrix}
\end{aligned}$$

The matrix containing the vectors $\mathbf{B}$, $\mathbf{AB}$, ... $\mathbf{A}^{n-1}\mathbf{B}$ is called the controllability matrix. In order for the system to be controllable, for every initial condition $\mathbf{x}(0)$ we must be able to find the corresponding vector $\mathbf{w}$. This is only possible when the rows of the controllability matrix are linearly independent. Therefore, the condition of controllability is that this controllability matrix is full rank. In DRAKE you can obtain the controllability matrix for a linear system using `ControllabilityMatrix`, or check the rank condition directly using `IsControllable`.

Note that a linear feedback to change the eigenvalues of the eigenmodes is not sufficient to accomplish our goal of getting to the goal in finite time. In fact, the open-loop control to reach the goal is easily obtained with a final-value LQR problem, and (for $\mathbf{R} = \mathbf{I}$) is actually a simple function of the controllability Grammian[7].

## Controllability vs. underactuated

Analysis of the controllability of both the Acrobot and Cart-Pole systems reveals that the linearized dynamics about the upright are, in fact, controllable. This implies that the linearized system, if started away from the zero state, can be returned to the zero state in finite time. This is potentially surprising - after all the systems are underactuated. For example, it is interesting and surprising that the Acrobot can balance itself in the upright position without having a shoulder motor.

The controllability of these model systems demonstrates an extremely important, point: *An underactuated system is not necessarily an uncontrollable system.* Underactuated systems cannot follow arbitrary trajectories, but that does not imply that they cannot arrive at arbitrary points in state space. However, the trajectory required to place the system into a particular state may be arbitrarily complex.

The controllability analysis presented here is for linear time-invariant (LTI) systems. A comparable analysis exists for linear time-varying (LTV) systems. We will even see extensions to nonlinear systems; although it will often be referred to by the synonym of "reachability" analysis.

### 3.4.3 LQR feedback

Controllability tells us that a trajectory to the fixed point exists, but does not tell us which one we should take or what control inputs cause it to occur. Why not? There are potentially infinitely many solutions. We have to pick one.

The tools for controller design in linear systems are very advanced. In particular, as we describe in the linear optimal control chapter, one can easily design an optimal feedback controller for a regulation task like balancing, so long as we are willing to linearize the system around the operating point and define optimality in terms of a quadratic cost function:

$$J(\mathbf{x}_0) = \int_0^\infty \left[\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t)\right]dt, \quad \mathbf{x}(0) = \mathbf{x}_0, \mathbf{Q} = \mathbf{Q}^T > 0, \mathbf{R} = \mathbf{R}^T > 0.$$

The linear feedback matrix $\mathbf{K}$ used as

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t),$$

is the so-called optimal linear quadratic regulator (LQR). Even without understanding the detailed derivation, we can quickly become practitioners of LQR. DRAKE provides a function,

```
K = LinearQuadraticRegulator(A, B, Q, R)
```

for linear systems. It also provides a version

```
controller = LinearQuadraticRegulator(system, context, Q, R)
```

that will linearize the system for you around an equilibrium and return the controller (in the original coordinates). Therefore, to use LQR, one simply needs to define the symmetric positive-definite cost matrices, $\mathbf{Q}$ and $\mathbf{R}$. In their most common form, $\mathbf{Q}$ and $\mathbf{R}$ are positive diagonal matrices, where the entries $Q_{ii}$ penalize the relative errors in state variable $x_i$ compared to the other state variables, and the entries $R_{ii}$ penalize actions in $u_i$.

**Example 3.5 (LQR for the Acrobot and Cart-Pole)**

Take a minute to play around with the LQR controller for the Acrobot and the Cart-Pole

Open in Colab

Make sure that you take a minute to look at the code which runs during these examples. Can you set the **Q** and **R** matrices differently, to improve the performance?

Simulation of the closed-loop response with LQR feedback shows that the task is indeed completed - and in an impressive manner. Oftentimes the state of the system has to move violently away from the origin in order to ultimately reach the origin. Further inspection reveals the (linearized) closed-loop dynamics are in fact non-minimum phase (acrobot has 3 right-half zeros, cart-pole has 1).

### Example 3.6 (LQR for Quadrotors)

LQR works essentially out of the box for Quadrotors, if linearized around a nominal fixed point (where the non-zero thrust from the propellers is balancing gravity).

Open in Colab

or Click here for the animation.

Note that LQR, although it is optimal for the linearized system, is not necessarily the best linear control solution for maximizing basin of attraction of the fixed-point. The theory of *robust control*(e.g., [8]), which explicitly takes into account the differences between the linearized model and the nonlinear model, will produce controllers which outperform our LQR solution in this regard.

## 3.5 Partial feedback linearization

In the introductory chapters, we made the point that the underactuated systems are not feedback equivalent to $\ddot{q} = \mathbf{u}$. Although we cannot always simplify the full dynamics of the system, it is still possible to linearize a portion of the system dynamics. The technique is called *partial feedback linearization*.

Consider the cart-pole example. The dynamics of the cart are affected by the motions of the pendulum. If we know the model, then it seems quite reasonable to think that we could create a feedback controller which would push the cart in exactly the way necessary to counter-act the dynamic contributions from the pendulum - thereby linearizing the cart dynamics. What we will see, which is potentially more surprising, is that we can also use a feedback controller for the cart to feedback linearize the dynamics of the passive pendulum joint.

We'll use the term *collocated* partial feedback linearization to describe a controller which linearizes the dynamics of the actuated joints. What's more surprising is that it is often possible to achieve *non-collocated* partial feedback linearization - a controller which linearizes the dynamics of the unactuated joints. The treatment presented here follows from [9].

### 3.5.1 PFL for the Cart-Pole System

#### Collocated

Starting from the equations 18 and 19, we have

$$\ddot{\theta} = -\ddot{x}c - s$$
$$\ddot{x}(2 - c^2) - sc - \dot{\theta}^2 s = f_x$$

Therefore, applying the feedback control

$$f_x = (2 - c^2)\ddot{x}^d - sc - \dot{\theta}^2 s \tag{30}$$

results in

$$\begin{aligned}\ddot{x} =& \ddot{x}^d \\ \ddot{\theta} =& -\ddot{x}^d c - s,\end{aligned}$$

which are valid globally.

Look carefully at the resulting equations. Of course, it says that we can impose whatever accelerations we like on the cart. But even the resulting equations of the pole happen to take a nice form here: they have been reduced to the equations of the simple pendulum (without a cart), where the torque input is now given instead by $\ddot{x}c$. It's as if we have a simple pendulum with torque control, except our command is modulated by a $\cos\theta$ term, and this $\cos\theta$ term is fundamental -- it's true that our control authority goes to zero when the pole is horizontal, because no amount of force on the cart in that configuration can act like a torque on the pole.

### Non-collocated

Starting again from equations 18 and 19, we have

$$\begin{aligned}\ddot{x} &= -\frac{\ddot{\theta} + s}{c} \\ \ddot{\theta}(c - \frac{2}{c}) - 2\tan\theta - \dot{\theta}^2 s &= f_x\end{aligned}$$

Applying the feedback control

$$f_x = (c - \frac{2}{c})\ddot{\theta}^d - 2\tan\theta - \dot{\theta}^2 s \tag{31}$$

results in

$$\begin{aligned}\ddot{\theta} =& \ddot{\theta}^d \\ \ddot{x} =& -\frac{1}{c}\ddot{\theta}^d - \tan\theta.\end{aligned}$$

Note that this expression is only valid when $\cos\theta \neq 0$. Once again, we know that the force cannot create a torque when the pole is perfectly horizontal. In fact, the controller we have written will "blow-up" -- requesting infinite force at $\cos\theta = 0$; so make sure you saturate the command before you implement it on hardware (or even in simulation). Although it may come as a small consolation, at least we have that $(c - \frac{2}{c})$ never goes to zero; in fact you can check for yourself that $|c - \frac{2}{c}| \geq 1$.

## 3.5.2 General form

For systems that are trivially underactuated (torques on some joints, no torques on other joints), we can, without loss of generality, reorganize the joint coordinates in any underactuated system described by the manipulator equations into the form:

$$\mathbf{M}_{11}\ddot{\mathbf{q}}_1 + \mathbf{M}_{12}\ddot{\mathbf{q}}_2 = \tau_1, \tag{32}$$
$$\mathbf{M}_{21}\ddot{\mathbf{q}}_1 + \mathbf{M}_{22}\ddot{\mathbf{q}}_2 = \tau_2 + \mathbf{u}, \tag{33}$$

with $\mathbf{q} \in \mathbb{R}^n$, $\mathbf{q}_1 \in \mathbb{R}^l$, $\mathbf{q}_2 \in \mathbb{R}^m$, $l = n - m$. $\mathbf{q}_1$ represents all of the passive joints, and $\mathbf{q}_2$ represents all of the actuated joints, and the $\tau = \tau_g - \mathbf{C}\dot{\mathbf{q}}$ terms capture all of the Coriolis and gravitational terms, and

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix}\mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22}\end{bmatrix}.$$

Fortunately, because $\mathbf{M}$ is uniformly (e.g. for all $\mathbf{q}$) positive definite, $\mathbf{M}_{11}$ and $\mathbf{M}_{22}$ are also positive definite, by the Schur complement condition for positive definiteness.

## Collocated linearization

Performing the same substitutions into the full manipulator equations, we get:

$$\ddot{\mathbf{q}}_1 = \mathbf{M}_{11}^{-1}\left[\tau_1 - \mathbf{M}_{12}\ddot{\mathbf{q}}_2\right] \tag{34}$$

$$(\mathbf{M}_{22} - \mathbf{M}_{21}\mathbf{M}_{11}^{-1}\mathbf{M}_{12})\ddot{\mathbf{q}}_2 - \tau_2 + \mathbf{M}_{21}\mathbf{M}_{11}^{-1}\tau_1 = \mathbf{u} \tag{35}$$

It can be easily shown that the matrix $(\mathbf{M}_{22} - \mathbf{M}_{21}\mathbf{M}_{11}^{-1}\mathbf{M}_{12})$ is invertible[9]; we can see from inspection that it is symmetric. PFL follows naturally, and is valid globally.

## Non-collocated linearization

$$\ddot{\mathbf{q}}_2 = \mathbf{M}_{12}^{+}\left[\tau_1 - \mathbf{M}_{11}\ddot{\mathbf{q}}_1\right] \tag{36}$$

$$(\mathbf{M}_{21} - \mathbf{M}_{22}\mathbf{M}_{12}^{+}\mathbf{M}_{11})\ddot{\mathbf{q}}_1 - \tau_2 + \mathbf{M}_{22}\mathbf{M}_{12}^{+}\tau_1 = \mathbf{u} \tag{37}$$

where $\mathbf{M}_{12}^{+}$ is a Moore-Penrose pseudo-inverse. This inverse provides a unique solution when the rank of $\mathbf{M}_{12}$ equals $l$, the number of passive degrees of freedom in the system (it cannot be more, since the matrix only has $l$ rows). The rank condition in this context is sometimes called the property of "Strong Inertial Coupling". It is state dependent; in the cart-pole example above $\mathbf{M}_{12} = \cos\theta$ and drops rank exactly when $\cos\theta = 0$. A system has Global Strong Inertial Coupling if it exhibits Strong Inertial Coupling in every state.

## Task-space partial feedback linearization

In general, we can define some combination of active and passive joints that we would like to control. This combination is sometimes called a "task space". Consider an output function of the form,

$$\mathbf{y} = \mathbf{h}(\mathbf{q}),$$

with $\mathbf{y} \in \mathbb{R}^p$, which defines the task space. Define $\mathbf{H}_1 = \frac{\partial \mathbf{h}}{\partial \mathbf{q}_1}$, $\mathbf{H}_2 = \frac{\partial \mathbf{h}}{\partial \mathbf{q}_2}$, $\mathbf{H} = [\mathbf{H}_1, \mathbf{H}_2]$.

### Theorem 3.1 - Task Space PFL

If the actuated joints are commanded so that

$$\ddot{\mathbf{q}}_2 = \bar{\mathbf{H}}^{+}\left[\ddot{\mathbf{y}}^d - \dot{\mathbf{H}}\dot{\mathbf{q}} - \mathbf{H}_1\mathbf{M}_{11}^{-1}\tau_1\right], \tag{38}$$

where $\bar{\mathbf{H}} = \mathbf{H}_2 - \mathbf{H}_1\mathbf{M}_{11}^{-1}\mathbf{M}_{12}$. and $\bar{\mathbf{H}}^{+}$ is the right Moore-Penrose pseudo-inverse,

$$\bar{\mathbf{H}}^{+} = \bar{\mathbf{H}}^T(\bar{\mathbf{H}}\bar{\mathbf{H}}^T)^{-1},$$

then we have

$$\ddot{\mathbf{y}} = \ddot{\mathbf{y}}^d. \tag{39}$$

subject to

$$\operatorname{rank}\left(\bar{\mathbf{H}}\right) = p, \tag{40}$$

**Proof Sketch.** Differentiating the output function we have

$$\dot{\mathbf{y}} = \mathbf{H}\dot{\mathbf{q}}$$

$$\ddot{\mathbf{y}} = \dot{\mathbf{H}}\dot{\mathbf{q}} + \mathbf{H}_1\ddot{\mathbf{q}}_1 + \mathbf{H}_2\ddot{\mathbf{q}}_2.$$

Solving 32 for the dynamics of the unactuated joints we have:

$$\ddot{\mathbf{q}}_1 = \mathbf{M}_{11}^{-1}(\tau_1 - \mathbf{M}_{12}\ddot{\mathbf{q}}_2) \tag{41}$$

Substituting, we have

$$\begin{aligned}\ddot{\mathbf{y}} =& \dot{\mathbf{H}}\dot{\mathbf{q}} + \mathbf{H}_1\mathbf{M}_{11}^{-1}(\tau_1 - \mathbf{M}_{12}\ddot{\mathbf{q}}_2) + \mathbf{H}_2\ddot{\mathbf{q}}_2 && (42)\\ =& \dot{\mathbf{H}}\dot{\mathbf{q}} + \bar{\mathbf{H}}\ddot{\mathbf{q}}_2 + \mathbf{H}_1\mathbf{M}_{11}^{-1}\tau_1 && (43)\\ =& \ddot{\mathbf{y}}^d && (44)\end{aligned}$$

Note that the last line required the rank condition (40) on $\bar{\mathbf{H}}$ to ensure that the rows of $\bar{\mathbf{H}}$ are linearly independent, allowing $\bar{\mathbf{H}}\bar{\mathbf{H}}^+ = \mathbf{I}$.

In order to execute a task space trajectory one could command

$$\ddot{\mathbf{y}}^d = \ddot{\mathbf{y}}^d + \mathbf{K}_d(\dot{\mathbf{y}}^d - \dot{\mathbf{y}}) + \mathbf{K}_p(\mathbf{y}^d - \mathbf{y}).$$

Assuming the internal dynamics are stable, this yields converging error dynamics when $\mathbf{K}_p, \mathbf{K}_d > 0$[4], which implies $y(t) \to y^d(t)$. For a position control robot, the acceleration command of (38) suffices. Alternatively, a torque command follows by substituting (38) and (41) into (33).

### Example 3.7 (End-point trajectory following with the Cart-Pole system)

Consider the task of trying to track a desired kinematic trajectory with the endpoint of pendulum in the cart-pole system. With one actuator and kinematic constraints, we might be hard-pressed to track a trajectory in both the horizontal and vertical coordinates. But we can at least try to track a trajectory in the vertical position of the end-effector.

Using the task-space PFL derivation, we have:

$$\begin{aligned} y &= h(\mathbf{q}) = -l\cos\theta \\ \dot{y} &= l\dot{\theta}\sin\theta \end{aligned}$$

If we define a desired trajectory:

$$y^d(t) = \frac{l}{2} + \frac{l}{4}\sin(t),$$

then the task-space controller is easily implemented using the derivation above.

The task space derivation above provides a convenient generalization of the partial feedback linearization (PFL) [9], which encompasses both the collocated and non-collocated results. If we choose $\mathbf{y} = \mathbf{q}_2$ (collocated), then we have

$$\mathbf{H}_1 = 0, \mathbf{H}_2 = \mathbf{I}, \dot{\mathbf{H}} = 0, \bar{\mathbf{H}} = \mathbf{I}, \bar{\mathbf{H}}^+ = \mathbf{I}.$$

From this, the command in (38) reduces to $\ddot{\mathbf{q}}_2 = \ddot{\mathbf{q}}_2^d$. The actuator command is then

$$\mathbf{u} = \mathbf{M}_{21}\mathbf{M}_{11}^{-1}(\tau_1 - \mathbf{M}_{12}\ddot{\mathbf{q}}_2^d) + \mathbf{M}_{22}\ddot{\mathbf{q}}_2^d - \tau_2,$$

and the rank condition (40) is always met.

If we choose $\mathbf{y} = \mathbf{q}_1$ (non-collocated), we have

$$\mathbf{H}_1 = \mathbf{I}, \mathbf{H}_2 = 0, \dot{\mathbf{H}} = 0, \bar{\mathbf{H}} = -\mathbf{M}_{11}^{-1}\mathbf{M}_{12}.$$

The rank condition (40) requires that $\text{rank}(\mathbf{M}_{12}) = l$, in which case we can write $\bar{\mathbf{H}}^+ = -\mathbf{M}_{12}^+\mathbf{M}_{11}$, reducing the rank condition to precisely the "Strong Inertial Coupling" condition described in [9]. Now the command in (38) reduces to

$$\ddot{\mathbf{q}}_2 = \mathbf{M}_{12}^+ \left[\tau_1 - \mathbf{M}_{11}\ddot{\mathbf{q}}_1^d\right] \tag{45}$$

The actuator command is found by substituting (45) into (33), yielding the same results as in [9].

## 3.6 SWING-UP CONTROL

### 3.6.1 Energy shaping

Recall in the last chapter, we used energy shaping to swing up the simple pendulum. This idea turns out to be a bit more general than just for the simple pendulum. As we will see, we can use similar concepts of "energy shaping" to produce swing-up controllers for the acrobot and cart-pole systems. It's important to note that it only takes one actuator to change the total energy of a system.

Although a large variety of swing-up controllers have been proposed for these model systems (c.f. [10], [11], [12], [13], [14], [15], [1], [16]), the energy shaping controllers tend to be the most natural to derive and perhaps the most well-known.

### 3.6.2 Cart-Pole

Let's try to apply the energy-shaping ideas to the cart-pole system. The basic idea, from [17], is to use collocated PFL to simplify the dynamics, use energy shaping to regulate the pendulum to its homoclinic orbit, then to add a few terms to make sure that the cart stays near the origin. This is a bit surprising... if we want to control the pendulum, shouldn't we use the non-collocated version? Actually, we ultimately want to control both the cart and the pole, and we will build on the collocated version both because it avoids inverting the $\cos\theta$ term that can go to zero and because (when all parameters are set to 1) we were left with a particularly simple form of the equations:

$$\ddot{x} = u \tag{46}$$

$$\ddot{\theta} = -uc - s. - \frac{1}{l}(uc + gs) \tag{47}$$

The first equation is clearly simple, but even the second equation is exactly the equations of a *decoupled* pendulum, just with a slightly odd control input that is modulated by $\cos\theta$.

Let us regulate the energy of this decoupled pendulum using energy shaping. The energy of the pendulum (a unit mass, unit length, simple pendulum in unit gravity) is given by:

$$E(\mathbf{x}) = \frac{1}{2}\dot{\theta}^2 - \cos\theta.$$

The desired energy, equivalent to the energy at the desired fixed-point, is

$$E^d = 1.$$

Again defining $\tilde{E}(\mathbf{x}) = E(\mathbf{x}) - E^d$, we now observe that

$$\begin{aligned}\dot{\tilde{E}}(\mathbf{x}) =& \dot{E}(\mathbf{x}) = \dot{\theta}\ddot{\theta} + \dot{\theta}s \\ =& \dot{\theta}[-uc - s] + \dot{\theta}s \\ =& -u\dot{\theta}\cos\theta.\end{aligned}$$

Therefore, if we design a controller of the form

$$u = k\dot{\theta}\cos\theta\tilde{E}, \quad k > 0$$

the result is

$$\dot{\tilde{E}} = -k\dot{\theta}^2\cos^2\theta\tilde{E}.$$

This guarantees that $|\tilde{E}|$ is non-increasing, but isn't quite enough to guarantee that it will go to zero. For example, if $\theta = \dot{\theta} = 0$, the system will never move. However, if we have that

$$\int_0^t \dot{\theta}^2(t')\cos^2\theta(t')dt' \to \infty, \quad \text{as } t \to \infty,$$

then we have $\tilde{E}(t) \to 0$. This condition, a version of the LaSalle's theorem that we will develop in our notes on Lyapunov methods, is satisfied for all but the trivial constant trajectories at fixed points.

Now we return to the full system dynamics (which includes the cart). [17] and [18] use the simple pendulum energy controller with an addition PD controller designed to regulate the cart:

$$\ddot{x}^d = k_E\dot{\theta}\cos\theta\tilde{E} - k_p x - k_d\dot{x}.$$

[17] provides a proof of convergence for this controller with some nominal parameters. [10] provides a slightly different controller derived directly from a Lyapunov argument.

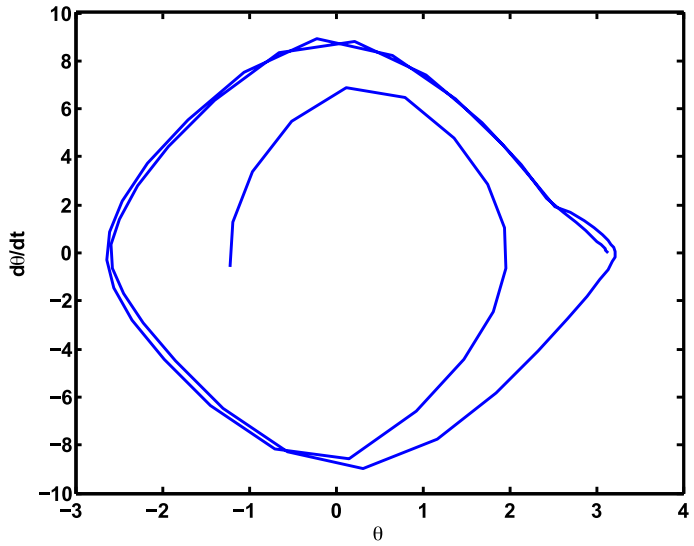



Figure 3.4 - Cart-Pole Swingup: Example phase plot of the pendulum subsystem using energy shaping control. The controller drives the system to the homoclinic orbit, then switches to an LQR balancing controller near the top.

## 3.6.3 Acrobot

Swing-up control for the acrobot can be accomplished in much the same way. [13] - pump energy. Clean and simple. No proof. Slightly modified version (uses arctan instead of sat) in [19]. Clearest presentation in [18].

Use collocated PFL. ($\ddot{q}_2 = \ddot{q}_2^d$).

$$E(\mathbf{x}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}\dot{\mathbf{q}} + U(\mathbf{x}).$$

$$E_d = U(\mathbf{x}^*).$$

$$\bar{u} = \dot{q}_1 \tilde{E}.$$

$$\ddot{q}_2^d = -k_1 q_2 - k_2 \dot{q}_2 + k_3 \bar{u},$$

Extra PD terms prevent proof of asymptotic convergence to homoclinic orbit. Proof of another energy-based controller in [12].

### 3.6.4 Discussion

The energy shaping controller for swing-up presented here is a pretty faithful representative from the field of nonlinear underactuated control. Typically these control derivations require some clever tricks for simplifying or canceling out terms in the nonlinear equations, then some clever Lyapunov function to prove stability. In these cases, PFL was used to simplify the equations, and therefore the controller design.

We will see another nice example of changing coordinates in the nonlinear equations to make the problem easier when we study "differential flatness" for trajectory optimization. This approach is perhaps most famous these days for very dynamic trajectory planning with quadrotors.

These controllers are important, representative, and relevant. But clever tricks with nonlinear equations seem to be fundamentally limited. Most of the rest of the material presented in this book will emphasize more general computational approaches to formulating and solving these and other control problems.

## 3.7 Other model systems

The acrobot and cart-pole systems are just two of the model systems used heavily in underactuated control research. Other examples include:

- Pendubot
- Inertia wheel pendulum
- Furuta pendulum (horizontal rotation and vertical pendulum)
- Hovercraft

## 3.8 Exercises

### Exercise 3.1 (Cart-Pole: Linearization and Balancing)

For this exercise you will work exclusively in this notebook. You will be asked to complete the following steps.

a. Derive the state-space dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ of the cart-pole system.
b. Linearize the dynamics from point (a) around the unstable equilibrium point.
c. Analyze the linearization error for different values of the state $\mathbf{x}$ and the control $\mathbf{u}$.
d. Among the states from point (c), identify which ones are stabilized to the origin by the LQR controller.

### Exercise 3.2 (Cart-Pole: Double Pendulum and URDF)

For this exercise you will work in this notebook. You will complete the following

steps.

a. Construct the cart-pole pendulum URDF structure for the single pendulum cart-pole..
b. Extend the single pendulum cart-pole to a double pendulum cart-pole by modifying the URDF model of the robot and testing LQR's ability to control it.

## REFERENCES


1. Richard M. Murray and John Hauser, "A case Study in Approximate Linearization: The Acrobot Example", *Memorandum No. UCB/ERL (unknown)*, April, 1991.

2. M. W. Spong, "Underactuated Mechanical Systems", *Control Problems in Robotics and Automation*, 1997.

3. Mark Spong, "The Swingup Control Problem for the Acrobot", *IEEE Control Systems Magazine*, vol. 15, no. 1, pp. 49-55, February, 1995.

4. Hassan K. Khalil, "Nonlinear Systems",Prentice Hall , December, 2001.

5. Jean-Jacques E. Slotine and Weiping Li, "Applied Nonlinear Control",Prentice Hall , October, 1990.

6. Katsuhiko Ogata, "Modern Control Engineering",Prentice Hall Incorporated , August, 1996.

7. Gilbert Strang, "Introduction to Linear Algebra",Wellesley-Cambridge Press , October, 1998.

8. Chi-Tsong Chen, "Linear System Theory and Design",Oxford University Press , Sept 10, 1998.

9. Kemin Zhou and John C. Doyle, "Essentials of Robust Control",Prentice Hall , 1997.

10. Mark Spong, "Partial feedback linearization of underactuated mechanical systems", *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, vol. 1, pp. 314-321, September, 1994.

11. Isabelle Fantoni and Rogelio Lozano, "Non-linear Control for Underactuated Mechanical Systems",Springer-Verlag , 2002.

12. Araki and N.; Okada and M.; Konishi and Y.; Ishigaki and H.;, "Parameter identification and swing-up control of an Acrobot system", *International Conference on International Technology*, 2005.

13. Xin Xin and M. Kaneda, "New analytical results of the energy based swinging up control of the Acrobot", *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC)*, vol. 1, pp. 704 - 709, Dec, 2004.

14. Mark W. Spong, "Swing up control of the Acrobot", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2356-2361, 1994.

15. Arun D. Mahindrakar and Ravi N. Banavar, "A swing-up of the acrobot based on a simple pendulum strategy", *International Journal of Control*, vol. 78, no. 6, pp. 424-429, April, 2005.

16. M.D. Berkemeier and R.S. Fearing, "Tracking fast inverted trajectories of the underactuated Acrobot", *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 4, pp. 740-750, Aug, 1999.

17. Xu-Zhi Lai and Jin-Hua She and Simon X. Yang and Min Wu, "Stability Analysis and Control Law Design for Acrobots", *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May, 2006.

18. Chung Choo Chung and John Hauser, "Nonlinear Control of a Swinging Pendulum", *Automatica*, vol. 31, no. 6, pp. 851-862, June, 1995.

19. Mark W. Spong, "Energy Based Control of a Class of Underactuated Mechanical Systems", *Proceedings of the 1996 IFAC World Congress*, 1996.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 4
# Simple Models of Walking and Running

Open in Colab

Practical legged locomotion is one of the fundamental problems in robotics; we've seen amazing progress over the last few years, but there are still some major unsolved problems. Much of the recent progress is due to improvements in hardware -- a legged robot must carry all of its sensors, actuators and power and traditionally this meant underpowered motors that act as far-from-ideal force/torque sources -- Boston Dynamics and other companies have made incredible progress here. The control systems implemented on these systems, though, are still more heuristic than you might think. They also require dramatically higher bandwidth and lower latency that the human motor control system and still perform worse in challenging environments.

The control of walking robots is definitely underactuated: assuming we cannot pull on the ground (and barring any aerodynamic effects!), then no matter how powerful my actuators are, there is nothing that we can do to accelerate the center of mass of the robot towards the ground faster than gravity. But beyond the underactuated systems we have studied so far, the control of walking robots faces an additional complexity: **controlling through contact**. The fact that we can get non-zero contact forces if and only if two bodies are in contact introduces a fundamentally discrete/combinatorial element into the problem†.

Contact is essential for many aspects of robotics (manipulation is my other favorite example); it's sad to see so many robots going through life avoiding contact at any cost. Controlling contact means that your robot is capable of performing physical work on the environment; isn't that the point of robotics, afterall?

I like to start our discussion of contact with walking robots for a few important reasons. As you'll see in this chapter, there are a number of elegant "simple models" of walking that capture much of the essence of the problem with minimal complexity. But there is another, more subtle, reason: I want to continue to use the language of stability. At this point we've seen both finite-horizon and infinite-horizon problem formulations -- when we can say something about the behavior of a system in the limit as time goes to infinity, it's almost always cleaner than a finite-time analysis (because time drops away). Walking provides a very clean way to use the language

†I've had some interesting arguments with folks on this poi because it's possible t write contact models that smooth the discontinuity, and/or t model systems that have neglible collision events. But replacing discontinuity in the vector field with a stif but smooth transition does not remove the need to decide wheth

of stability in studying behavior of systems that are sometimes in contact but sometimes out of contact in the limiting case; we just need to extend our notions from stability of a fixed point to the stability of a (periodic) cycle.

## 4.1 Limit Cycles

A limit cycle is an periodic orbit that is a limit set of the dynamical system; they can be stable (the limit set of neighboring trajectories as time goes to infinity) or unstable (the limit set of neighboring trajectories as time goes to negative infinity). One of the simplest models of limit cycle behavior is the Van der Pol oscillator. Let's examine that first...

### Example 4.1 (Van der Pol Oscillator)

The Van der Pol oscillator is described by a second-order differential equation in one variable:

$$\ddot{q} + \mu(q^2 - 1)\dot{q} + q = 0, \quad \mu > 0$$

One can think of this system as almost a simple spring-mass-damper system, except that it has nonlinear damping. In particular, the velocity term dissipates energy when $|q| > 1$, and adds energy when $|q| < 1$. Therefore, it is not terribly surprising to see that the system settles into a stable oscillation from almost any initial conditions (the exception is the state $q = 0, \dot{q} = 0$). This can be seen nicely in the phase portrait below.

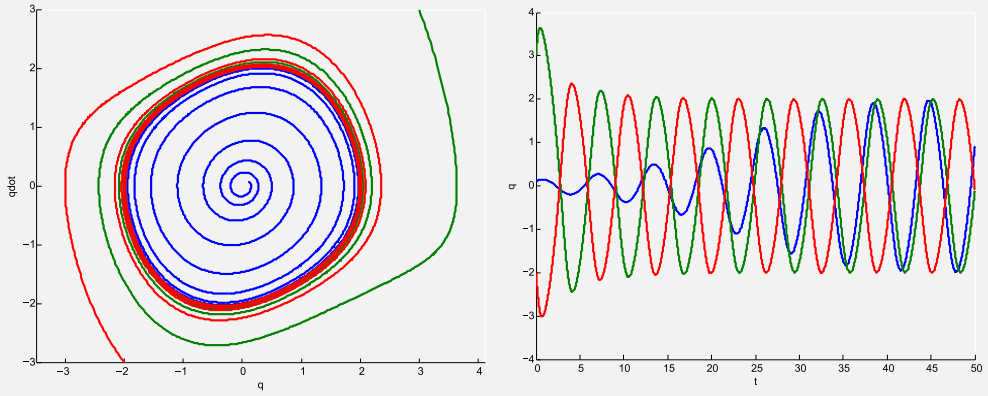



Figure 4.1 - System trajectories of the Van der Pol oscillator with $\mu = .2$. (Left) phase portrait. (Right) time domain.

However, if we examine system trajectories in the time domain (see the right panel of the figure above), then we can see that our traditional notion for stability of a trajectory, $\lim_{t\to\infty} |\mathbf{x}(t) - \mathbf{x}^*(t)| = 0$, is not satisfied here for any $\mathbf{x}^*(t)$. Although the phase portrait clearly reveals that all trajectories converge to the same orbit, the time domain plot reveals that these trajectories do not necessarily synchronize in time.

This example shows that stability of a trajectory (in time) is not the definition we want. Instead we will define the stability of an orbit, $\mathbf{x}^*(t)$, using

$$\min_{\tau} ||\mathbf{x}(t) - \mathbf{x}^*(\tau)||.$$

Here the quantity of interest is the distance between our trajectory and the *closest point* on the orbit. Depending on exactly how this distance evolves with time, we can define *orbital stability*: for every small $\epsilon > 0$, we can find a $\delta > 0$ such that $\min_\tau ||\mathbf{x}(t_0) - \mathbf{x}^*(\tau)|| < \delta$ implies $\forall t, \min_\tau ||\mathbf{x}(t) - \mathbf{x}^*(\tau)|| < \epsilon$). The definitions for

asymptotic orbital stability, exponential orbital stability, finite-time orbital stability follow naturally, as does the definition of an unstable orbit. In the case of limit cycles, this $\mathbf{x}^*(t)$ is a periodic solution with $\mathbf{x}^*(t + t_{period}) = \mathbf{x}^*(t)$.

As we begin to extend our notions of stability, allow me to make a few quick connections to the discussion of stability in the chapter on Lyapunov analysis. It's interesting to know that if we can find an invariant set in state space which does not contain any fixed points, then this set must contain a closed orbit (this is the Poincaré-Bendixson Theorem) [1]. It's also interesting to note that gradient potential fields (and Lyapunov functions with $\dot{V}(\mathbf{x}) \prec 0$) cannot have a closed orbit [1], and consequently we will also have to slightly modify the Lyapunov analysis we have introduced so far to handle limit cycles. We'll discuss this in some details in an upcoming chapter.

### 4.1.1 Poincaré Maps

The first tool one should know for analyzing the stability of a limit cycle, and one that will serve us quite well for the examples in this chapter, is the method of Poincaré. Let's consider a dynamical system, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^n$. Define an $n-1$ dimensional *surface of section*, $S$. We will also require that $S$ is transverse to the flow (i.e., all trajectories starting on $S$ flow through $S$, not parallel to it). The Poincaré map (or return map) is a mapping from $S$ to itself:

$$\mathbf{x}_p[n+1] = \mathbf{P}(\mathbf{x}_p[n]),$$

where $\mathbf{x}_p[n]$ is the state of the system at the $n$th crossing of the surface of section. Note that we will use the notation $\mathbf{x}_p$ to distinguish the state of the discrete-time system from the continuous-time system; they are related by $\mathbf{x}_p[n] = \mathbf{x}(t_c[n])$, where $t_c[n]$ is the time of the $n$th crossing of $S$.

For our purposes in this chapter, it will be useful for us to allow the Poincaré maps to also include fixed points of the original continuous-time system, and to define the return time to be zero. These points do not satisfy the transversality condition, and require some care numerically, but they do not compromise our analysis.

**Example 4.2 (Return map for the Van der Pol Oscillator)**

Since the state space of this system is two-dimensional, a return map for the system must have dimension one. For instance, we can take $S$ to be the line segment where $q = 0$, $\dot{q} \geq 0$. It is easy to see that all trajectories that start on S are transverse to it.

One dimensional maps, like one dimensional flows, are amenable to graphical analysis.

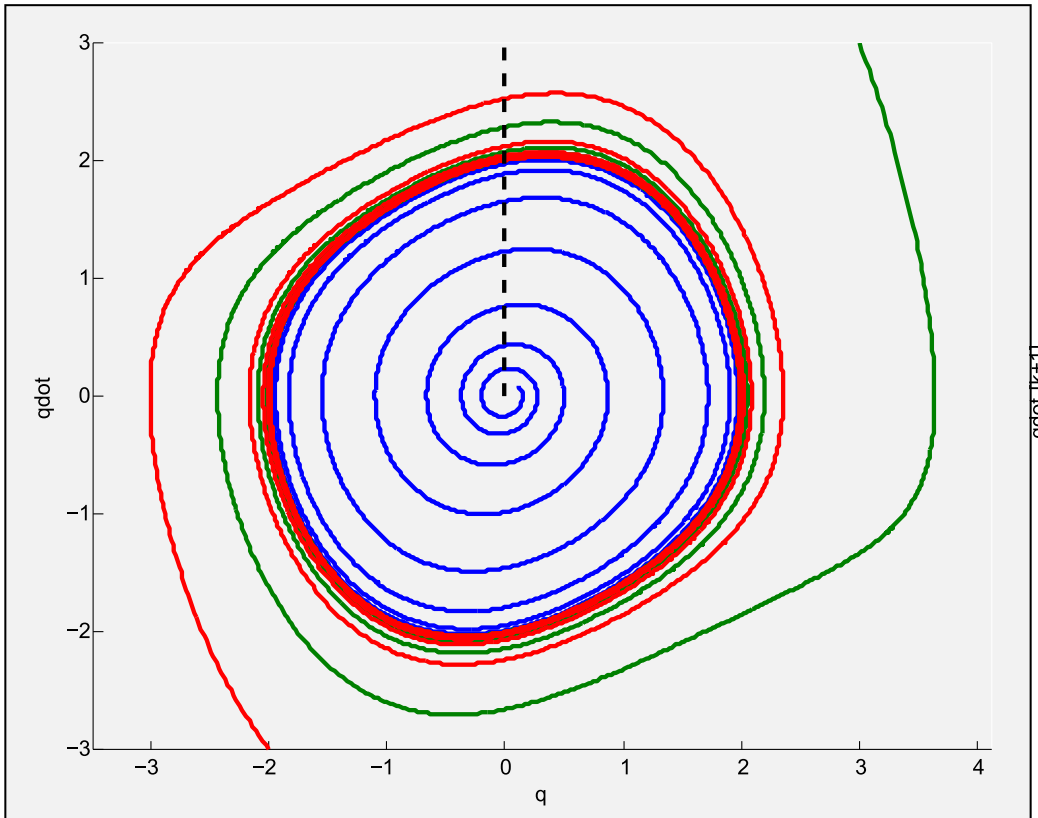



Figure 4.2 - (Left) Phase plot with the surface of section, $S$, drawn with a black dashed line. (Right) The resulting Poincaré first-return map (blue), and the line of slope one (red).

If we can demonstrate that $\mathbf{P}(\mathbf{x}_p)$ exists for all $\mathbf{x}_p \in S$, (all trajectories that leave $S$ eventually return to $S$), then something remarkable happens: we can reduce the stability analysis for a limit cycle in the original coordinates into the stability analysis of a fixed point on the discrete map. If $\mathbf{x}_p^*$ is a stable (or unstable) fixed point of the map $\mathbf{P}$, then there exists a unique periodic orbit $\mathbf{x}^*(t)$ which passes through $\mathbf{x}_p^*$ that is a (stable, or unstable) limit cycle. If we are able to upper-bound the time it takes for trajectories leaving $S$ to return, then it is also possible to infer asympotitic orbital or even exponential orbital stability.

In practice it is often difficult or impossible to find $\mathbf{P}$ analytically, but it can be sampled quite reasonably numerically. Once a fixed point of $\mathbf{P}$ is obtained (by finding a solution to $\mathbf{x}_p^* = \mathbf{P}(\mathbf{x}_p^*)$), we can infer local limit cycle stability with an eigenvalue analysis. If this eigen-analysis is performed in the original coordinates (as opposed to the $n-1$ dimensional coordinates of $S$), then there will always be a at least one zero eigenvalue - corresponding to perturbations along the limit cycle which do not change the state of first return. The limit cycle is considered locally exponentially stable if all remaining eigenvalues, $\lambda_i$, have magnitude less than one, $|\lambda_i| < 1$.

It is sometimes possible to infer more global stability properties of the return map by examining $\mathbf{P}$. We will study computational methods in the later chapter. [2] describes some less computational stability properties known for *unimodal* maps which are useful for the simple systems we study in this chapter.

A particularly graphical method for understanding the dynamics of a one-dimensional iterated map is the "staircase method": Sketch the Poincaré map -- a plot of $x_p[n]$ vs $x_p[n+1]$ -- and also the line of slope one. In order to "iterate" the map, take an initial condition on the plot as a point $(x_p[k], x_p[k])$, and move vertically on the plot to the point $(x_p[k], x_p[k+1] = f(x_p[k]))$. Now move horizontally to the next point $(x_p[k+1], x_p[k+1])$ and repeat.

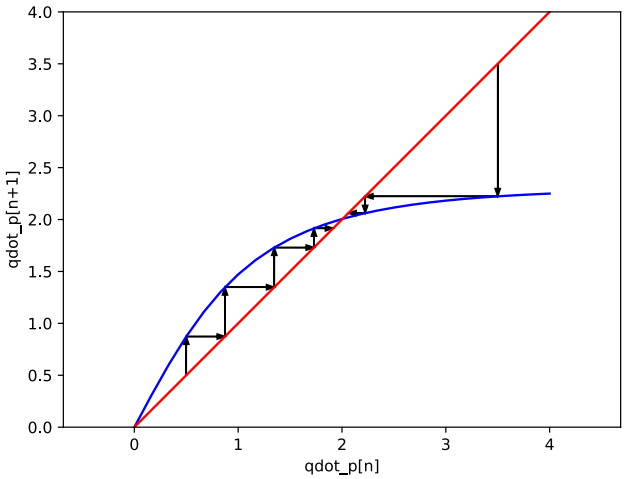



Figure 4.3 - The "staircase" on the return map of the Van der Pol Oscillator with $\mu = 0.2$, starting from two initial conditions ($\dot{q}_p[0] = .5$ and $\dot{q}_p[0] = 3.5$).

We can quickly recognize the fixed points as the points where the return map crosses this line. We can infer local exponential stability if the absolute value of the slope of return map at the fixed point is less than one ($|\lambda| < 1$). The figure above illustrates the stability of the Van der Pol limit cycle; it also illustrates the (one-sided) instability of the fixed point at the origin. Understanding the regions of attraction of fixed points on an iterated map (or any discrete-time system) graphically requires a little more care, though, than the continuous-time flows we examined before. The discrete maps can jump a finite distance on each step, and will in fact oscillate back and forth from either side of the fixed point when $\lambda < 0$.

## 4.2 Simple Models of Walking

Much of fundamental work in the dynamics of legged robots was done originally in the context of studying *passive-dynamic walkers* (often just "passive walker"). We've already seen my favorite example of a passive walker, in the first chapter, but here it is again:

Figure 4.4 - A 3D passive dynamic walker by Steve Collins and Andy Ruina[3].

This amazing robot has no motors and no controllers... it walks down a tiny ramp and is powered entirely by gravity. We can build up our understanding of this robot in a series of steps. As we will see, what is remarkable is that even though the system is passive, we believe that the periodic gait you see in the video is actually a stable limit cycle!

Well before the first passive walkers, one of the earliest models of walking was proposed by McMahon[4], who argued that humans use a mostly ballistic (passive) gait. He observed that muscle activity (recorded via EMG) in the "stance leg" is relatively high, but muscle activity in the "swing leg" is very low, except for at the very beginning and end of swing. He proposed a "ballistic walker" model -- a three-link pendulum in the sagittal plane, with one link from the ankle to the hip representing a straight "stance leg", the second link from the hip back down to the "swing leg" knee, and the final link from the knee down to the swing foot -- and argued that this model could largely reproduce the kinematics of gait. This model of "walking by vaulting" is considered overly simplistic today, as we now understand much better the role of compliance in the stance leg during walking. His model was also restricted to the continuous portion of gait, not the actual dynamics of taking a step. But it was the start. Nearly a decade later, McGeer[5] followed up with a series

of similarly inspired walking machines, which he coined "passive-dynamic walkers".

### 4.2.1 The Rimless Wheel

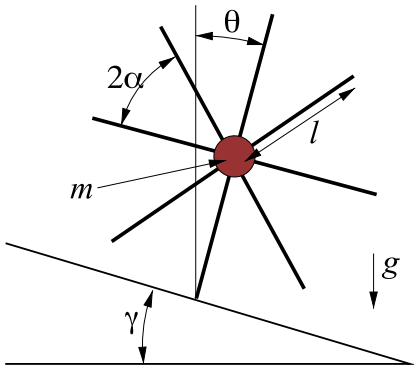



Figure 4.5 - The rimless wheel. The orientation of the stance leg, $\theta$, is measured clockwise from the vertical axis.

Perhaps the simplest possible model of a legged robot, introduced as the conceptual framework for passive-dynamic walking by McGeer[5], is the rimless wheel. This system has rigid legs and only a point mass at the hip as illustrated in the figure above. To further simplify the analysis, we make the following modeling assumptions:

- Collisions with ground are inelastic and impulsive (only angular momentum is conserved around the point of collision).
- The stance foot acts as a pin joint and does not slip.
- The transfer of support at the time of contact is instantaneous (no double support phase).
- $0 \leq \gamma < \frac{\pi}{2}$, $0 < \alpha < \frac{\pi}{2}$, $l > 0$.

Note that the coordinate system used here is slightly different than for the simple pendulum ($\theta = 0$ is at the top, and the sign of $\theta$ has changed).

The most comprehensive analysis of the rimless wheel was done by [6].

## Stance Dynamics

The dynamics of the system when one leg is on the ground are given by

$$\ddot{\theta} = \frac{g}{l}\sin(\theta).$$

If we assume that the system is started in a configuration directly after a transfer of support ($\theta(0^+) = \gamma - \alpha$), then forward walking occurs when the system has an initial velocity, $\dot{\theta}(0^+) > \omega_1$, where

$$\omega_1 = \sqrt{2\frac{g}{l}[1 - \cos(\gamma - \alpha)]}.$$

$\omega_1$ is the threshold at which the system has enough kinetic energy to vault the mass over the stance leg and take a step. This threshold is zero for $\gamma = \alpha$ and does not exist for $\gamma > \alpha$ (because when $\gamma > \alpha$ the mass is always ahead of the stance foot, and the standing fixed point disappears). The next foot touches down when $\theta(t) = \gamma + \alpha$, at which point the conversion of potential energy into kinetic energy yields the velocity

$$\dot{\theta}(t^-) = \sqrt{\dot{\theta}^2(0^+) + 4\frac{g}{l}\sin\alpha\sin\gamma}.$$

$t^-$ denotes the time immediately before the collision.

## Foot Collision

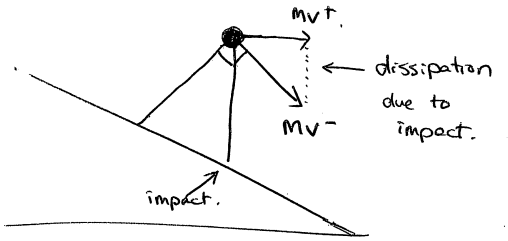



Figure 4.6 - Angular momentum is conserved around the point of impact

The angular momentum around the point of collision at time $t$ just before the next foot collides with the ground is

$$L(t^-) = -ml^2\dot{\theta}(t^-)\cos(2\alpha).$$

The angular momentum at the same point immediately after the collision is

$$L(t^+) = -ml^2\dot{\theta}(t^+).$$

Assuming angular momentum is conserved, this collision causes an instantaneous loss of velocity:

$$\dot{\theta}(t^+) = \dot{\theta}(t^-)\cos(2\alpha).$$

## Forward simulation

Given the stance dynamics, the collision detection logic ($\theta = \gamma \pm \alpha$), and the collision update, we can numerically simulate this simple model. Doing so reveals something remarkable... the wheel starts rolling, but then one of two things happens: it either runs out of energy and stands still, or it quickly falls into a stable periodic solution. Convergence to this stable periodic solution appears to happen from a huge range of initial conditions. Try it yourself.

### Example 4.3 (Numerical simulation of the rimless wheel)

Open in Colab

I've setup the notebook to make it easy for you to try a handful of interesting initial conditions. And even made an interactive widget for you to watch the simulation phase portrait as you change those initial conditions. Give it a try! (I recommend using Binder instead of Colab so you get the interactive features)

The rimless wheel model actually uses a number of the more nuanced features of DRAKE in order to simulate the hybrid system accurately (as do many of the examples in this chapter). It actually registers the collision events of the system -- the simulator uses zero-crossing detection to ensure that the time/state of collision is obtained accurately, and then applies the reset map.

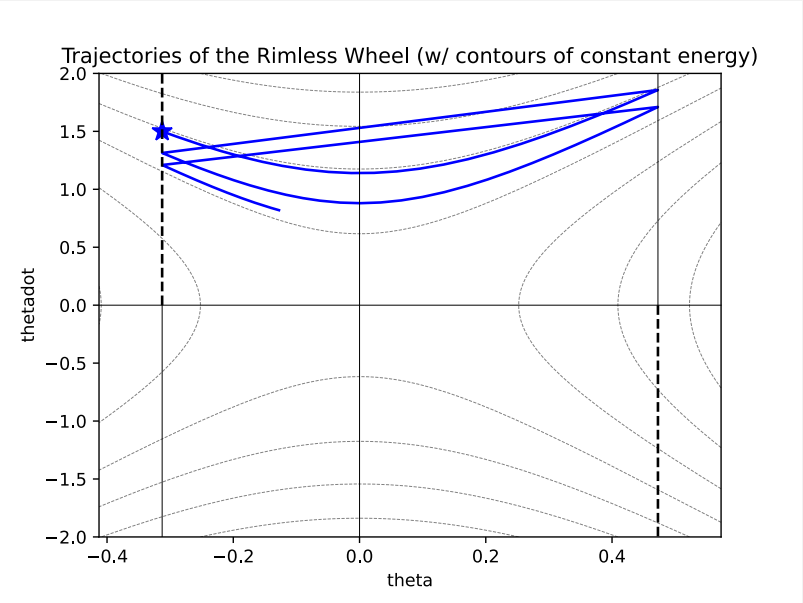



Figure 4.7 - Phase portrait of the rimless wheel ($m = 1$, $l = 1$, $g = 9.8$, $\alpha = \pi/8$, $\gamma = 0.08$).

One of the fantastic things about the rimless wheel is that the dynamics are exactly the undamped simple pendulum that we've thought so much about. So you will recognize the phase portraits of the system -- they are centered around the unstable fixed point of the simple pendulum.

## Poincaré Map

We can now derive the angular velocity at the beginning of each stance phase as a function of the angular velocity of the previous stance phase. First, we will handle the case where $\gamma \leq \alpha$ and $\dot{\theta}_n^+ > \omega_1$. The "step-to-step return map", factoring losses from a single collision, the resulting map is:

$$\dot{\theta}_{n+1}^+ = \cos(2\alpha)\sqrt{(\dot{\theta}_n{}^+)^2 + 4\frac{g}{l}\sin\alpha\sin\gamma}.$$

where the $\dot{\theta}^+$ indicates the velocity just *after* the energy loss at impact has occurred.

Using the same analysis for the remaining cases, we can complete the return map. The threshold for taking a step in the opposite direction is

$$\omega_2 = -\sqrt{2\frac{g}{l}[1 - \cos(\alpha + \gamma)]}.$$

For $\omega_2 < \dot{\theta}_n^+ < \omega_1$, we have

$$\dot{\theta}_{n+1}^+ = -\dot{\theta}_n^+ \cos(2\alpha).$$

Finally, for $\dot{\theta}_n^+ < \omega_2$, we have

$$\dot{\theta}_{n+1}^+ = -\cos(2\alpha)\sqrt{(\dot{\theta}_n^+)^2 - 4\frac{g}{l}\sin\alpha\sin\gamma}.$$

Altogether, we have (for $\gamma \leq \alpha$)

$$\dot{\theta}_{n+1}^{+} = \begin{cases} \cos(2\alpha)\sqrt{(\dot{\theta}_n^+)^2 + 4\frac{g}{l}\sin\alpha\sin\gamma}, & \omega_1 < \dot{\theta}_n^+ \\ -\dot{\theta}_n^+\cos(2\alpha), & \omega_2 < \dot{\theta}_n^+ < \omega_1 \\ -\cos(2\alpha)\sqrt{(\dot{\theta}_n^+)^2 - 4\frac{g}{l}\sin\alpha\sin\gamma}, & \dot{\theta}_n^+ < w_2 \end{cases}.$$

Notice that the return map is undefined for $\dot{\theta}_n = \{\omega_1, \omega_2\}$, because from these configurations, the wheel will end up in the (unstable) equilibrium point where $\theta = 0$ and $\dot{\theta} = 0$, and will therefore never return to the map.

This return map blends smoothly into the case where $\gamma > \alpha$. In this regime,

$$\dot{\theta}_{n+1}^{+} = \begin{cases} \cos(2\alpha)\sqrt{(\dot{\theta}_n^+)^2 + 4\frac{g}{l}\sin\alpha\sin\gamma}, & 0 \le \dot{\theta}_n^+ \\ -\dot{\theta}_n^+\cos(2\alpha), & \omega_2 < \dot{\theta}_n^+ < 0 \\ -\cos(2\alpha)\sqrt{(\dot{\theta}_n^+)^2 - 4\frac{g}{l}\sin\alpha\sin\gamma}, & \dot{\theta}_n^+ \le w_2 \end{cases}.$$

Notice that the formerly undefined points at $\{\omega_1, \omega_2\}$ are now well-defined transitions with $\omega_1 = 0$, because it is kinematically impossible to have the wheel statically balancing on a single leg.

## Fixed Points and Stability

For a fixed point, we require that $\dot{\theta}_{n+1}^{+} = \dot{\theta}_n^{+} = \omega^*$. Our system has two possible fixed points, depending on the parameters:

$$\omega_{stand}^* = 0, \quad \omega_{roll}^* = \cot(2\alpha)\sqrt{4\frac{g}{l}\sin\alpha\sin\gamma}.$$

The limit cycle plotted above illustrates a state-space trajectory in the vicinity of the rolling fixed point. $\omega_{stand}^*$ is a fixed point whenever $\gamma < \alpha$. $\omega_{roll}^*$ is a fixed point whenever $\omega_{roll}^* > \omega_1$. It is interesting to view these bifurcations in terms of $\gamma$. For small $\gamma$, $\omega_{stand}$ is the only fixed point, because energy lost from collisions with the ground is not compensated for by gravity. As we increase $\gamma$, we obtain a stable rolling solution, where the collisions with the ground exactly balance the conversion of gravitational potential to kinetic energy. As we increase $\gamma$ further to $\gamma > \alpha$, it becomes impossible for the center of mass of the wheel to be inside the support polygon, making standing an unstable configuration.

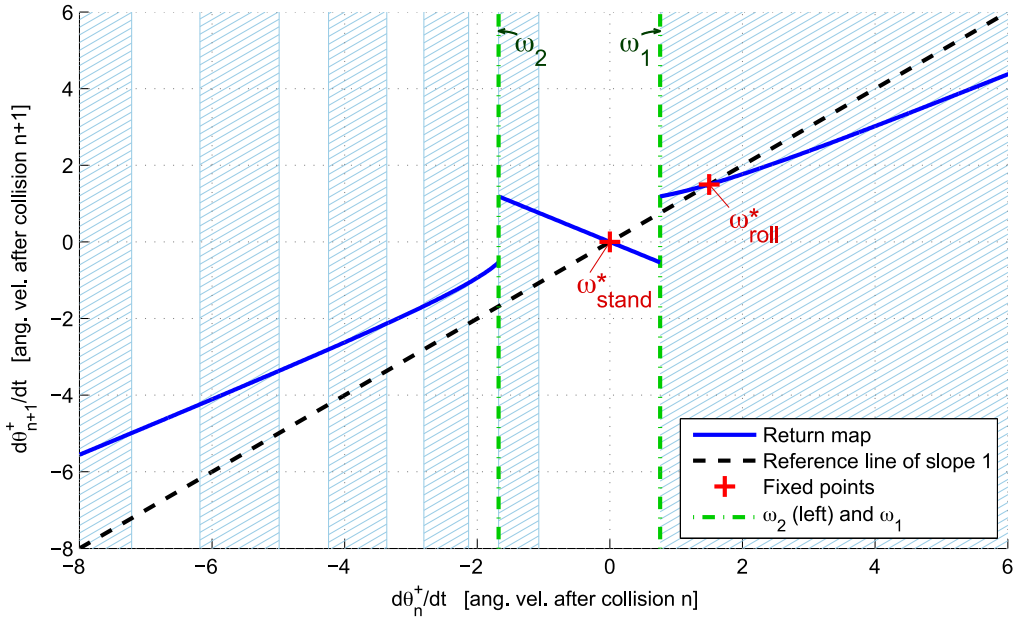


Figure 4.8 - Limit cycle trajectory of the rimless wheel ( $m = 1, l = 1, g = 9.8, \alpha = \pi/8, \gamma = 0.15$). All hatched regions converge to the rolling fixed point, $\omega^*_{roll}$; the white regions converge to zero velocity ($\omega^*_{stand}$).

## Stability of standing still

I opened this chapter with the idea that the natural notion of stability for a walking system is periodic stability, and I'll stand by it. But we can also examine the stability of a fixed-point (of the original coordinates; no Poincaré this time) for a system that has contact mechanics. For a legged robot, a fixed point means standing still. This can actually be a little subtle in our hybrid models: in the rimless wheel, standing still is the limiting case where we have infinitely frequent collisions. [Details coming soon.]

## 4.2.2 The Compass Gait

The rimless wheel models only the dynamics of the stance leg, and simply assumes that there will always be a swing leg in position at the time of collision. To remove this assumption, we take away all but two of the spokes, and place a pin joint at the hip. To model the dynamics of swing, we add point masses to each of the legs. I've derived the equations of motion for the system assuming that we have a torque actuator at the hip - resulting in swing dynamics equivalent to an Acrobot (although in a different coordinate frame) - but let's examine the passive dynamics of the system first ($\mathbf{u} = 0$).

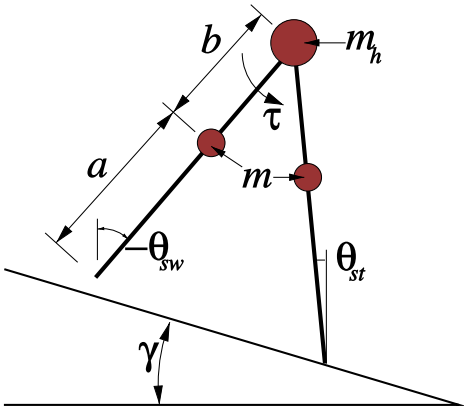



Figure 4.9 - The compass gait

In addition to the modeling assumptions used for the rimless wheel, we also assume that the swing leg retracts in order to clear the ground without disturbing the position of the mass of that leg. This model, known as the compass gait, is well studied in the literature using numerical methods [7, 8, 9], but relatively little is known about it analytically.

The state of this robot can be described by 4 variables: $\theta_{st}, \theta_{sw}, \dot{\theta}_{st}$, and $\dot{\theta}_{sw}$. The abbreviation $st$ is shorthand for the stance leg and $sw$ for the swing leg. Using $\mathbf{q} = [\theta_{st}, \theta_{sw}]^T$ and $\mathbf{u} = \tau$, we can write the dynamics as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(q) + \mathbf{B}\mathbf{u},$$

with

$$\mathbf{M} = \begin{bmatrix} (m_h + m)l^2 + ma^2 & -mlb\cos(\theta_{st} - \theta_{sw}) \\ -mlb\cos(\theta_{st} - \theta_{sw}) & mb^2 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & -mlb\sin(\theta_{st} - \theta_{sw})\dot{\theta}_{sw} \\ mlb\sin(\theta_{st} - \theta_{sw})\dot{\theta}_{st} & 0 \end{bmatrix}$$

$$\tau_g(q) = \begin{bmatrix} (m_h l + ma + ml)g\sin(\theta_{st}) \\ -mbg\sin(\theta_{sw}) \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

and $l = a + b$.

The foot collision is an instantaneous change of velocity caused by a impulsive force at the foot that brings the foot to rest. The update to the velocities can be calculated using the derivation for impulsive collisions derived in the appendix. To use it, we proceed with the following steps:

- Add a "floating base" to the system by adding a free (x,y) joint at the pre-impact stance toe, $\mathbf{q}_{fb} = [x, y, \theta_{st}, \theta_{sw}]^T$.
- Calculate the mass matrix for this new system, call it $\mathbf{M}_{fb}$.
- Write the position of the (pre-impact) swing toe as a function $\phi(\mathbf{q}_{fb})$. Define the Jacobian of this function: $\mathbf{J} = \frac{\partial \phi}{\partial \mathbf{q}_{fb}}$.
- The post-impact velocity that ensures that $\dot{\phi} = 0$ after the collision is given by

  $$\dot{\mathbf{q}}^+ = \left[\mathbf{I} - \mathbf{M}_{fb}^{-1}\mathbf{J}^T[\mathbf{J}\mathbf{M}_{fb}^{-1}\mathbf{J}^T]^{-1}\mathbf{J}\right]\dot{\mathbf{q}}^-,$$

  noting that $\dot{x}^- = \dot{y}^- = 0$, and that you need only read out the last two elements of $\dot{\mathbf{q}}^+$. The velocity of the post-impact stance foot will be zero by definition, and the new velocity of the pre-impact stance foot can be completely determined from the minimal coordinates.
- Switch the stance and swing leg positions and velocities.

**Example 4.4 (Numerical simulation of the compass gait)**

You can simulate the passive compass gait using:


Try playing with the initial conditions. You'll find this system is *much* more sensitive to initial conditions than the rimless wheel. It actually took some work to find the initial conditions that make it walk stably.

Numerical integration of these equations reveals a stable limit cycle, plotted below. Notice that when the left leg is in stance (top part of the plot), the trajectory quite resembles the familiar pendulum dynamics of the rimless wheel. But this time, when the leg impacts, it takes a long arc around as the swing leg before returning to stance. The impacts are also clearly visible as discontinuous changes (decreases) in velocity. The dependence of this limit cycle on the system parameters has been studied extensively in [7].

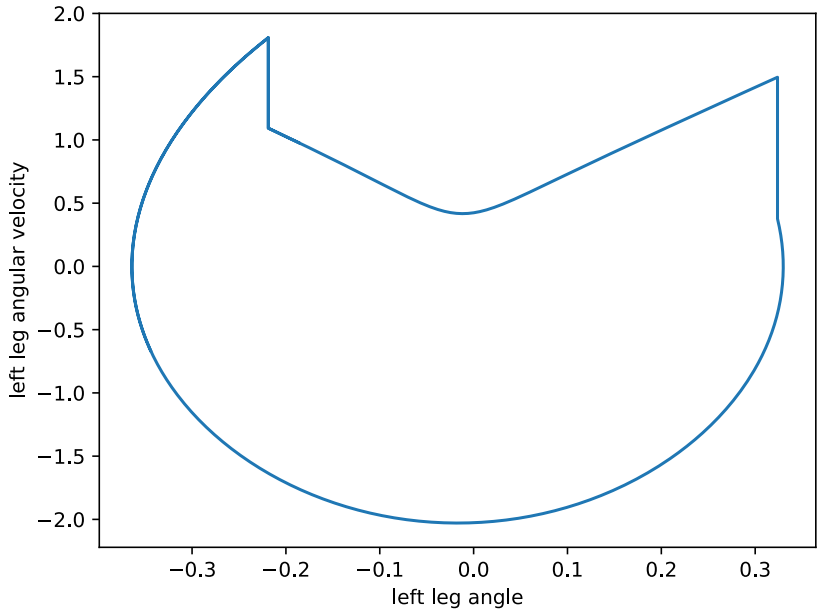



Figure 4.10 - Passive limit cycle trajectory of the compass gait. ($m = 5$kg, $m_h = 10$ kg, $a = b = 0.5$m, $\phi = 0.0525$rad. $\mathbf{x}(0) = [0, 0, 0.4, -2.0]^T$). Drawn is the phase portait of the left leg angle, which is recovered from $\theta_{st}$ and $\theta_{sw}$ in the simulation with some simple book-keeping.

The basin of attraction of the stable limit cycle is a narrow band of states surrounding the steady state trajectory. Although the simplicity of this model makes it attractive for conceptual explorations, this lack of stability makes it difficult to implement on a physical device.

### 4.2.3 The Kneed Walker

To achieve a more anthropomorphic gait, as well as to acquire better foot clearance and ability to walk on rough terrain, we want to model a walker that includes a knee[10]. The mathematical modeling would be simpler if we used a single link for the stance leg, but we'll go ahead and use two links for each leg (and each with a point mass) because this robot can actually be built!

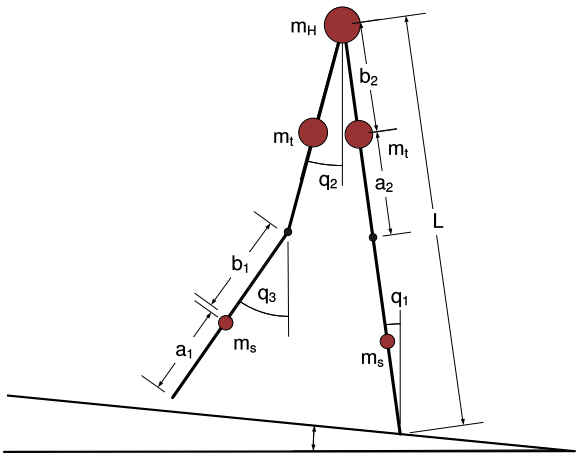



Figure 4.11 - The Kneed Walker

At the beginning of each step, the system is modeled as a three-link pendulum, like the ballistic walker[4, 11, 9]. The stance leg is the one in front, and it is the first link of a pendulum, with two point masses. The swing leg has two links, with the joint between them unconstrained until knee-strike. Given appropriate mass distributions and initial conditions, the swing leg bends the knee and swings forward. When the swing leg straightens out (the lower and upper length are aligned), knee-strike occurs. The knee-strike is modeled as a discrete inelastic collision, conserving angular momentum and changing velocities instantaneously.

After this collision, the knee is locked and we switch to the compass gait model with a different mass distribution. In other words, the system becomes a two-link pendulum. Again, the heel-strike is modeled as an inelastic collision. After the collision, the legs switch instantaneously. After heel-strike then, we switch back to the ballistic walker's three-link pendulum dynamics. This describes a full step cycle of the kneed walker, which is shown above.

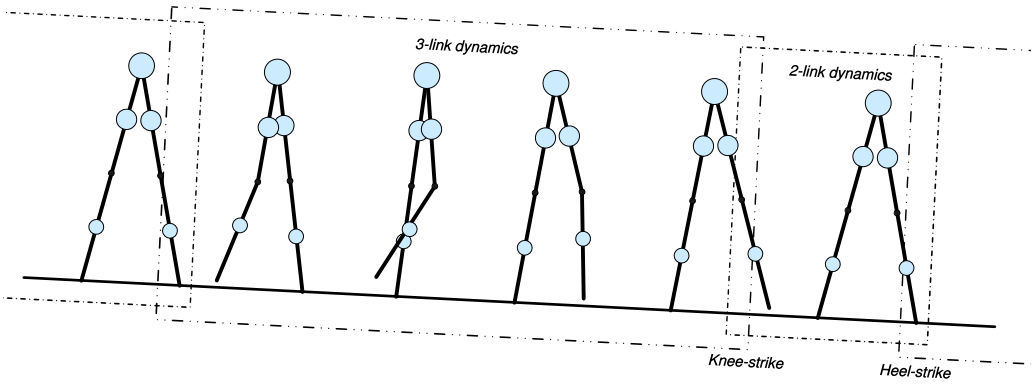



Figure 4.12 - Limit cycle trajectory for kneed walker

By switching between the dynamics of the continuous three-link and two-link pendulums with the two discrete collision events, we characterize a full point-feed kneed walker walking cycle. After finding appropriate parameters for masses and link lengths, a stable gait is found. As with the compass gait's limit cycle, there is a swing phase (top) and a stance phase (bottom). In addition to the two heel-strikes, there are two more instantaneous velocity changes from the knee-strikes as marked in the figure. This limit cycle is traversed clockwise.

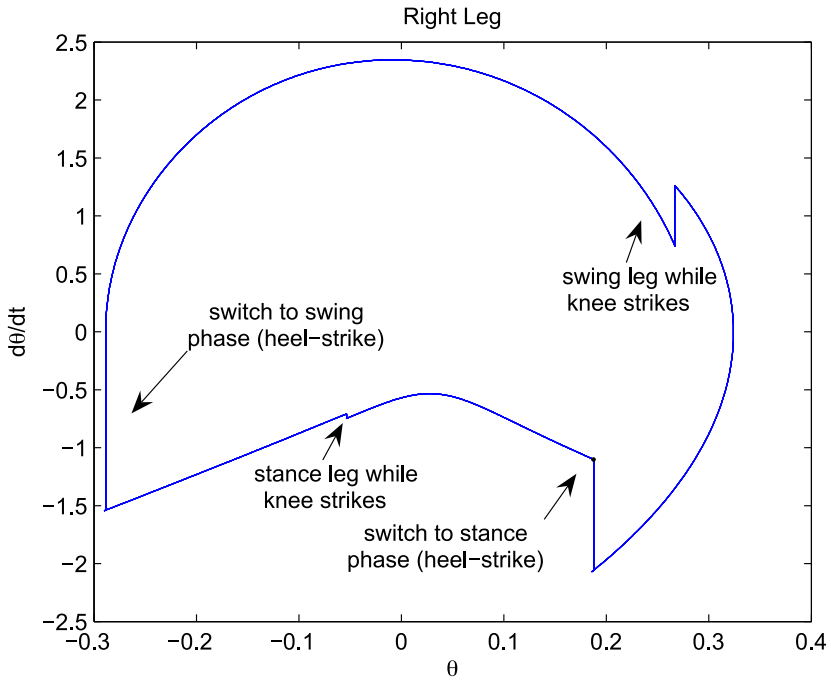



Figure 4.13 - Limit cycle (phase portrait) of the kneed walker

### 4.2.4 Curved feet

The region of attraction of the kneed-walking limit cycle can be improved considerably by the addition of curved feet...

Figure 4.14 - Tad McGeer's kneed walker. Here is Matt Haberland's guide to launching it successfully. It's nontrivial!

### 4.2.5 And beyond...

The world has seen all sorts of great variations on the passive-dynamic walker

theme. Almost certainly the most impressive are the creations of Dutch artist Theo Jansen -- he likes to say that he is creating "new forms of life" which he calls the Strandbeest. There are many variations of these amazing machines -- including his beach walkers which are powered only by the wind (I've actually been able to visit Theo's workshop once; it is *very* windy there).

These results are very real. Robin Deits (an exceptionally talented student who felt particularly inspired once on a long weekend) once reproduced one of Theo's earliest creations in DRAKE.

Figure 4.15 - Robin Deits' simulation of the Strandbeest.

## 4.3 SIMPLE MODELS OF RUNNING

Just like walking, our understanding of the dynamics and control of running has evolved by a nice interplay between the field of biomechanics and the field of robotics. But in running, I think it's fair to say that the roboticists got off the starting blocks first. And I think it's fair to say that it started with a series of hopping machines built by Marc Raibert and the Leg Laboratory (which was first at CMU, but moved to MIT) in the early 1980's. At a time where many roboticsts were building relatively clumsy walking robots that moved very slowly (when they managed to stay up!), the Leg Laboratory produced a series of hopping machines that threw themselves through the air with considerable kinetic energy and considerable flair.

To this day, I'm still a bit surprised that impressive running robots (assuming you accept hopping as the first form of running) appeared before the impressive walking robots. I would have thought that running would be a more difficult control and engineering task than walking. But these hopping machines turned out to be an incredibly clever way to build something simple and very dynamic.

Shortly after Raibert built his machines, Dan Koditschek and Martin Buehler started analyzing them with simpler models [2]. Years later, in collaboration with biologist Bob Full, they started to realize that the simple models used to describe the dynamics of Raibert's hoppers could also be used to describe experimental data obtained from running animals. In fact, they described an incredible range of experimental results, for animals ranging in size from a cockroach up to a horse[12] (I think the best overall summary of this line of work is [13]). The most successful of the simple models was the so-called "spring-loaded inverted pendulum" (or "SLIP", for short).

### 4.3.1 The Spring-Loaded Inverted Pendulum (SLIP)

The model is a point mass, $m$, on top of a massless, springy leg with rest length of $l_0$, and spring constant $k$. The configuration of the system is given by the $x, z$ position of the center of mass, and the length, $l$, and angle $\theta$ of the leg. The dynamics are modeled piecewise - with one dynamics governing the flight phase, and another governing the stance phase.

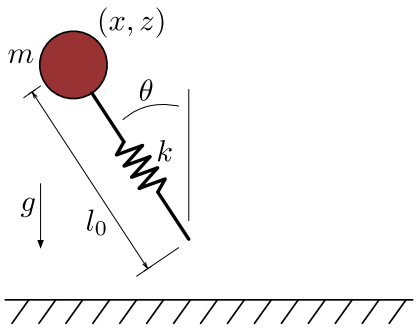



Figure 4.16 - The Spring-Loaded Inverted Pendulum (SLIP) model

In SLIP, we actually use different state variables for each phase of the dynamics. During the flight phase, we use the state variables: $\mathbf{x} = [x, z, \dot{x}, \dot{z}]^T$. The flight dynamics are simply

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ 0 \\ -g \end{bmatrix}.$$

Since the leg is massless, we assume that the leg can be controlled to any position instantaneously, so it's reasonable to take the leg angle as the control input $\theta = u$.

During the "stance phase", we write the dynamics in polar coordinates, with the foot anchored at the origin. Using the state variables: $\mathbf{x} = [r, \theta, \dot{r}, \dot{\theta}]^T$, the location of the mass is given by

$$\mathbf{x}_m = \begin{bmatrix} -r\sin\theta \\ r\cos\theta \end{bmatrix}.$$

The total energy is given by the kinetic energy (of the mass) and the potential energy from gravity and from the leg spring:

$$T = \frac{m}{2}(\dot{r}^2 + r^2\dot{\theta}^2), \quad U = mgr\cos\theta + \frac{k}{2}(r_0 - r)^2.$$

Plugging these into Lagrange yields the stance dynamics:

$$\begin{aligned} m\ddot{r} - mr\dot{\theta}^2 + mg\cos\theta - k(r_0 - r) = 0, \\ mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta} - mgr\sin\theta = 0. \end{aligned}$$

We assume that the stance phase is entirely ballistic; there are no control inputs during this phase.

Unlike the rimless wheel model, the idealization of a spring leg with a massless toe means that no energy is lost during the collision with the ground. The collision event causes a transition to a different state space, and to a different dynamics model, but no instantaneous change in velocity. The transition from flight to stance happens when $z - l_0\cos\theta \leq 0$. The transition from stance to flight happens when the stance leg reaches its rest length: $r \geq l_0$.

### Example 4.5 (Simulation of the SLIP model)

You can simulate the spring-loaded inverted pendulum using:


Take a moment to read the code that defines the `SpringLoadedInvertedPendulum` system. You can see that we carefully register the hybrid guards (witness functions) and reset maps, which tells the numerical integrator to use zero-crossing event detection to isolate precisely the time of the event and to apply the reset update at this event. It is quite elegant, and we've gone to some length to make sure that the numerical integration routines in DRAKE can support it.

We used the same witness functions and resets to simulate the walking models, but those were implemented in C++. This is a purely Python implementation, making it a bit easier to see those details.

## Analysis on the apex-to-apex map

The rimless wheel had only two state variables ($\theta$, and $\dot{\theta}$), and since the surface of section drops one variable, we ended up with a one-dimensional Poincare map. We have a lot more state variables here. Is there any hope for graphical analysis?

For this system, one can still make progress if we define the Poincare section to be at the apex of the hop (when $\dot{z} = 0$), and analyze the dynamics from apex to apex. We will ignore the absolute horizontal position, $x$, and the leg angle, $\theta$, as they are irrelevant to the flight dynamics. This still leaves us with two state variables at the apex: the horizontal velocity, $\dot{x}$, and the hopping height, $z$.

Importantly, at the apex, these two variables are linked -- when you know one, you know the other. This is because no energy is ever added or removed from the system. If you know the (constant) total energy in the system, and you know the height at the apex, then you know the horizontal velocity (up to a sign):

$$E_{apex}(z, \dot{x}) = \frac{1}{2}m\dot{x}^2 + mgz = E_0.$$

To make our analysis less dependent on our specific choice of parameters, we typically report any analysis using a dimensionless total energy, $\tilde{E} = \frac{E}{mgl_0}$, and a dimensionless spring constant, $\tilde{k} = \frac{kl_0}{mg}$ [14].

Putting all of these together, we can numerically evaluate the apex-to-apex return map, given a known total energy of the system.

### Example 4.6 (Numerical SLIP apex-to-apex map)

Numerically integrating the equations of motion from apex-to-apex, using zero-crossing event detection, can give us the apex-to-apex map.

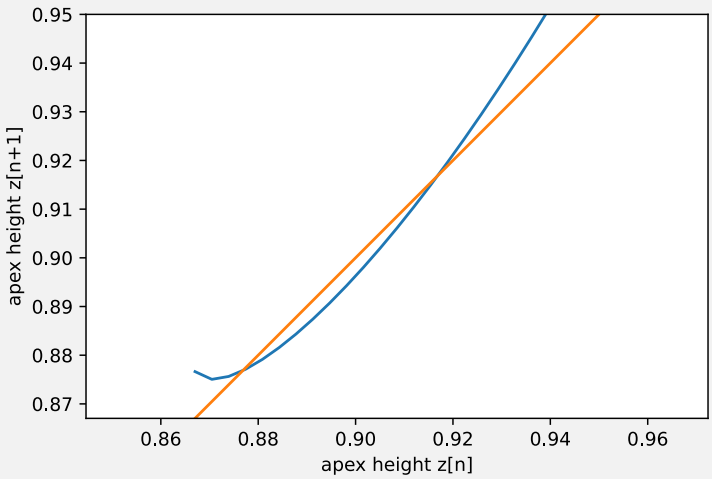



Figure 4.17 - Apex-to-apex return map of the SLIP model given $\tilde{k} = 10.7$, $\tilde{E} = 1.61$, and a fixed touchdown angle of $\theta = 30$ deg (the parameters used in [14]).

Using our graphical staircase analysis (for discrete-time systems), we can see that there are two fixed points of this map: one "stable" fixed point around 0.87 meters, and one unstable fixed point around 0.92 meters.

As always, you should run this code yourself to make sure you understand:

Open in Colab

I'd also recommend running the simulation using initial conditions at various points on this plot. You should ask: why does the plot cut off so abruptly just below the stable fixed point? Well the apex-to-apex map becomes undefined if the apex is smaller that $l_0 \cos\theta$ (the height at touchdown). Furthemore, when $z$ is too large (relative to $\tilde{E}$), then the system will not have enough horizontal energy to transition through the stance phase and continue forward motion: the center of mass will compress the spring and then fall backwards in the direction from which it came.

The system's conservation of total energy is quite convenient for our return map analysis, but it should give you pause. The rimless wheel actually *relied on the dissipation* due to collisions with the ground to acquire its stability. Is there any chance for a system that is energy conserving to be stable? The apex-to-apex return maps visualized above reveal "stable" fixed-points, but you must understand that the projection from $(z, \dot{x}])$ to just $z$ is using the assumption that the energy is conserved. As a result, our analysis on the one-dimensional Poincare map can only make a statement about *partial stability* (i.s.L., asymptotic, or exponential); our analysis does not say anything about stability against disturbances that change the total energy of the system.

Despite this limitation, it is quite interesting that such a simple system converges to particular hopping heights from various initial conditions. The observation that these "piecewise-Hamiltonian" systems (Hamiltonian because they conserve total energy) can exhibit this partial stability generated quite a bit of interest from the theoretical physics community. You can sense that enthusiasm when reading [13].

[14] also gives a more thorough analysis of this apex-to-apex map, including a closed-form approximation to the return map dynamics based on a linear (small angle) approximation of the stance dynamics.

## SLIP extensions

The original SLIP model, perhaps inspired by the hopping robots turned out to be quite effective in describing the vertical center of mass dynamics of a wide range of animals [13]. Remarkably, experiments showed that the dimensionless individual leg stiffness was very close to 10 for a huge range of animals, ranging from a $0.001\ kg$ cockroach to a $135\ kg$ horse [15]. But the SLIP model, and modest extensions to it, has also been used to understand lateral stability in cockroaches, resulting in some of my favorite experimental videos of all time [16].

## SLIP Control

## 4.3.2 Hopping robots from the MIT Leg Laboratory

## The Planar Monopod Hopper

A three-part control strategy: decoupling the control of hopping height, body attitude, and forward velocity [17].

## Running on four legs as though they were one

bipeds, quadrupeds, and backflips...[18]

## 4.3.3 Towards human-like running

# 4.4 A SIMPLE MODEL THAT CAN WALK AND RUN

# 4.5 EXERCISES

## Exercise 4.1 (One-Dimensional Hopper)

In this exercise we implement a one-dimensional version of the controller proposed in [19]. The goal is to control the vertical motion of a hopper by generating a stable resonant oscillation that causes the system to hop off the ground at a given height. We accomplish this via a careful energy analysis of the hopping cycle: all the details are in this notebook. Besides understanding in

detail the dynamics of the hopping system, in the notebook, you are asked to write two pieces of code:

a. A function that, given the state of the hopper, returns its total mechanical energy.
b. The hopping controller class. This is a DRAKE VectorSystem that, at each sampling time, reads the state of the hopper and returns the preload of the hopper spring necessary for the system to hop at the desired height. All the necessary information for the synthesis of this controller are given in the notebook.

## REFERENCES


1. Steven H. Strogatz, "Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering",Perseus Books , 1994.

2. Daniel E. Koditschek and Martin Buehler, "Analysis of a Simplified Hopping Robot", *International Journal of Robotics Research*, vol. 10, no. 6, pp. 587-605, Dec, 1991.

3. Steven H. Collins and Martijn Wisse and Andy Ruina, "A Three-Dimensional Passive-Dynamic Walking Robot with Two Legs and Knees", *International Journal of Robotics Research*, vol. 20, no. 7, pp. 607-615, July, 2001.

4. Simon Mochon and Thomas A. McMahon, "Ballistic walking: An improved model", *Mathematical Biosciences*, vol. 52, no. 3-4, pp. 241-260, December, 1980.

5. Tad McGeer, "Passive Dynamic Walking", *International Journal of Robotics Research*, vol. 9, no. 2, pp. 62-82, April, 1990.

6. Michael J. Coleman, "A Stability Study of a Three-Dimensional Passive-Dynamic Model of Human Gait", PhD thesis, Cornell University, 1998.

7. Ambarish Goswami and Benoit Thuilot and Bernard Espiau, "Compass-Like Biped Robot Part {I} : Stability and Bifurcation of Passive Gaits", Tech. Report, RR-2996, October, 1996.

8. A. Goswami, "Postural stability of biped robots and the foot rotation indicator ({FRI}) point", *International Journal of Robotics Research*, vol. 18, no. 6, 1999.

9. Mark W. Spong and Gagandeep Bhatia, "Further Results on Control of the Compass Gait Biped", *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 1933-1938, 2003.

10. Vanessa Hsu, "Passive dynamic walking with knees: A Point-foot model", , February, 2007. [ link ]

11. Simon Mochon and Thomas A. McMahon, "Ballistic Walking", *Journal of Biomechanics*, vol. 13, pp. 49-57, 1980.

12. R.J. Full and D.E. Koditschek, "Templates and anchors: neuromechanical hypotheses of legged locomotion on land", *Journal of Experimental Biology*, vol. 202, no. 23, pp. 3325-3332, 1999.

13. Philip Holmes and Robert J. Full and Dan Koditschek and John Guckenheimer, "The Dynamics of Legged Locomotion: Models, Analyses, and Challenges", *Society for Industrial and Applied Mathematics (SIAM)*

*Review*, vol. 48, no. 2, pp. 207--304, 2006.

14. Hartmut Geyer, "Simple models of legged locomotion based on compliant limb behavior", PhD thesis, University of Jena, 2005.

15. Daniel E Koditschek and Robert J Full and Martin Buehler, "Mechanical aspects of legged locomotion control", *Arthropod structure \& development*, vol. 33, no. 3, pp. 251--272, 2004.

16. DL Jindrich and RJ Full, "Dynamic stabilization of rapid hexapedal locomotion", *J Exp Biol*, vol. 205, no. Pt 18, pp. 2803-23, Sep, 2002.

17. Marc H. Raibert, "Legged Robots That Balance",The MIT Press , 1986.

18. Raibert and M. H. and Chepponis and M. and Brown and H. B., "Running on four legs as though they were one", *IEEE Journal of Robotics and Automation*, vol. 2, no. 2, pp. 70--82, 1986.

19. Marc H. Raibert, "Hopping in legged systems: Modeling and simulation for the 2D one-legged case", *IEEE Trans. Systems, Man, and Cybernetics*, vol. 14, pp. 451-463, 1984.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

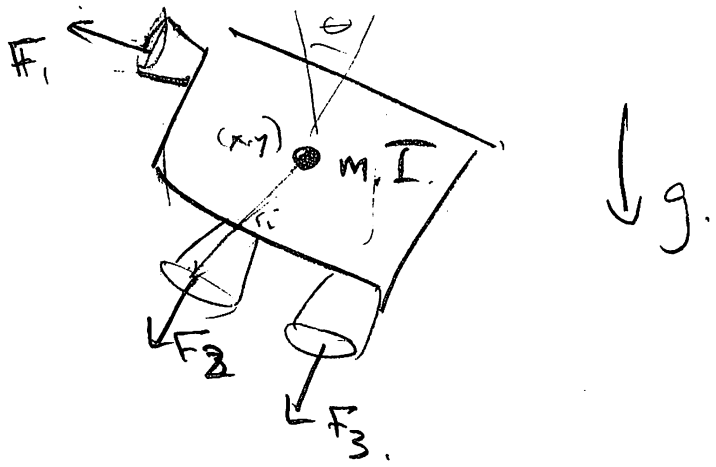
# Chapter 5
# Highly-articulated Legged Robots

The passive dynamic walkers and hopping robots described in the last chapter capture the fundamental dynamics of legged locomotion -- dynamics which are fundamentally nonlinear and punctuated by impulsive events due to making and breaking contact with the environment. But if you start reading the literature on humanoid robots, or many-legged robots like quadrupeds, then you will find a quite different set of ideas taking center stage: ideas like the "zero-moment point" and footstep planning. The goal of this chapter is to penetrate that world.

I'd like to start the discussion with a model that might seem quite far from the world of legged robots, but I think it's a very useful way to think about the problem.

# 5.1 Center of Mass Dynamics

## 5.1.1 A hovercraft model




Imagine you have a flying vehicle modeled as a single rigid body in a gravity field with some number of force "thrusters" attached. We'll describe the configuration of the vehicle by its orientation, $\theta$ and the location of its center of mass $(x, z)$. The vector from the center of mass to thruster $i$ is given by $r_i$, yielding the equations of motion:

$$\begin{aligned} m\ddot{x} &= \sum_i F_{i,x}, \\ m\ddot{z} &= \sum_i F_{i,z} - mg, \\ I\ddot{\theta} &= \sum_i [r_i \times F_i]_y, \end{aligned}$$

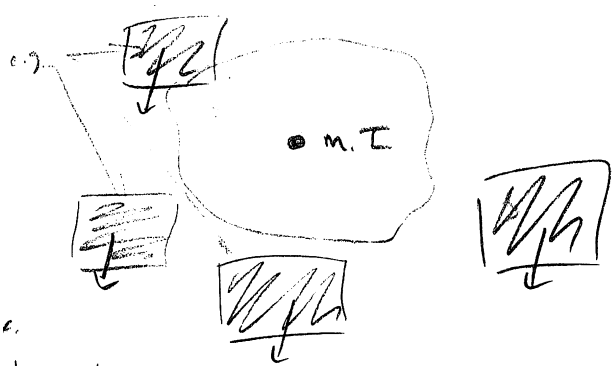
where I've used $F_{i,x}$ to represent the component of the force in the $x$ direction and have taken just the $y$-axis component of the cross product on the last line.

Our goal is to move the hovercraft to a desired position/orientation and to keep it there. If we take the inputs to be $F_i$, then the dynamics are affine (linear plus a constant term). As such, we can stabilize a stabilizable fixed point using a change of coordinates plus LQR or even plan/track a desired trajectory using time-varying LQR. If I were to add additional linear constraints, for instance constraining $F_{min} \leq F_i \leq F_{max}$, then I can still use linear model-predictive control (MPC) to plan and stabilize a desired motion. By most accounts, this is a relatively easy control problem. (Note that it would be considerably harder if my control input or input constraints depend on the orientation, $\theta$, but we don't need that here yet).

Now imagine that you have just a small number of thrusters (let's say two) each with severe input constraints. To make things more interesting, let us say that you're allowed to move the thrusters, so $\dot{r}_i$ becomes an additional control input, but with some important limitations: you have to turn the thrusters off when they are moving (e.g. $|F_i||\dot{r}_i| = 0$) and there is a limit to how quickly you can move the thrusters $|\dot{r}|_i \leq v_{max}$. This problem is now a lot more difficult, due especially to the fact that constraints of the form $u_1 u_2 = 0$ are non-convex.

I find this a very useful thought exercise; somehow our controller needs to make an effectively discrete decision to turn off a thruster and move it. The framework of optimal control should support this - you are sacrificing a short-term loss of control authority for the long-term benefit of having the thruster positioned where you would like, but we haven't developed tools yet that deal well with this discrete plus continuous decision making. We'll need to address that here.

Unfortunately, although the problem is already difficult, we need to continue to add constraints. Now let's say additionally, that the thrusters can only be turned on in certain locations, as cartooned here:



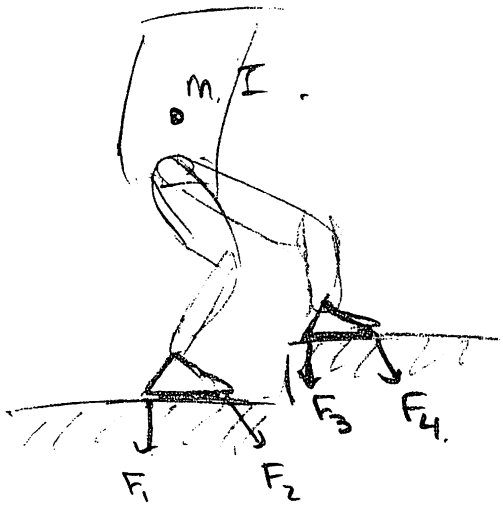

The union of these regions need not form a convex set. Furthermore, these locations could be specified in the robot frame, but may also be specified in the world frame, which I'll call $\mathbf{p}_i$:

$$\mathbf{p}_i = \mathbf{r}_i - \begin{bmatrix} x \\ 0 \\ z \end{bmatrix}.$$

This problem still feels like it should be tractable, but it's definitely getting hard.

## 5.1.2 Robots with (massless) legs

In my view, the hovercraft problem above is a central component of the walking problem. If we consider a walking robot with massless legs, then the feet are exactly like movable thrusters. As above, they are highly constrained - they can only produce force when they are touching the ground, and (typically) they can only produce forces in certain directions, for instance as described by a "friction cone" (you can push on the ground but not pull on the ground, and with Coulomb friction the forces tangential to the ground must be smaller than the forces normal to the ground, as described by a coefficient of friction, e.g. $|F_\parallel| < \mu |F_\perp|$).




The constraints on where you can put your feet / thrusters will depend on the kinematics of your leg, and the speed at which you can move them will depend on the full dynamics of the leg -- these are difficult constraints to deal with. But the actual dynamics of the rigid body are actually still affine, and still simple!

### 5.1.3 Capturing the full robot dynamics

We don't actually need to have massless legs for this discussion to make sense. If we use the coordinates $x, z$ to describe the location of the center of mass (CoM) of the entire robot, and $m$ to represent the entire mass of the robot, then the first two equations remain unchanged. The center of mass is a configuration dependent point, and does not have an orientation, but one important generalization of the orientation dynamics is given by the centroidal momentum matrix, $A(\mathbf{q})$, where $A(\mathbf{q})\dot{\mathbf{q}}$ captures the linear and angular momentum of the robot around the center of mass [1]. Note that the center of mass dynamics are still affine -- even for a big complicated humanoid -- but the centroidal momentum dynamics are nonlinear.

### 5.1.4 Impact dynamics

In the previous chapter we devoted relatively a lot of attention to dynamics of impact, characterized for instance by a guard that resets dynamics in a hybrid dynamical system. In those models we used impulsive ground-reaction forces (these forces are instantaneously infinite, but doing finite work) in order to explain the discontinuous change in velocity required to avoid penetration with the ground. This story can be extended naturally to the dynamics of the center of mass.

For an articulated robot, though, there are a number of possible strategies for the legs which can affect the dynamics of the center of mass. For example, if the robot hits the ground with a stiff leg like the rimless wheel, then the angular momentum about the point of collision will be conserved, but any momentum going into the ground will be lost. However, if the robot has a spring leg and a massless toe like the SLIP model, then no energy need be lost.

One strategy that many highly-articulated legged robots use is to keep their center of mass at a constant height,

$$z = c \quad \Rightarrow \quad \dot{z} = \ddot{z} = 0,$$

and minimize their angular momentum about the center of mass (here $\ddot{\theta} = 0$). Using this strategy, if the swinging foot lands roughly below the center of mass, then even with a stiff leg there is no energy dissipated in the collision - all of the momentum is

conserved. This often (but does not always) justify ignoring the impacts in the center of mass dynamics of the system.

## 5.2 THE SPECIAL CASE OF FLAT TERRAIN

While not the only important case, it is extremely common for our robots to be walking over flat, or nearly flat terrain. In this situation, even if the robot is touching the ground in a number of places (e.g., two heels and two toes), the constraints on the center of mass dynamics can be summarized very efficiently.

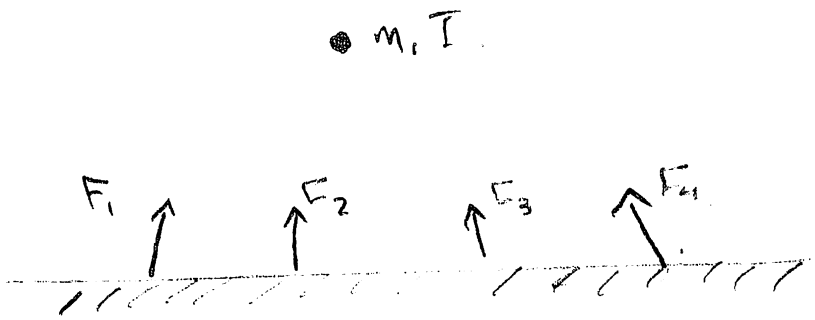



Figure 5.3 - External forces acting on a robot pushing on a flat ground

First, on flat terrain $F_{i,z}$ represents the force that is normal to the surface at contact point $i$. If we assume that the robot can only push on the ground (and not pull), then this implies

$$\forall i, F_{i,z} \geq 0 \Rightarrow \sum_i F_{i,z} \geq 0 \Rightarrow \ddot{z} \geq -g.$$

In other words, if I cannot pull on the ground, then my center of mass cannot accelerate towards the ground faster than gravity.

Furthermore, if we use a Coulomb model of friction on the ground, with friction coefficient $\mu$, then

$$\forall i, |F_{i,x}| \leq \mu F_{i,z} \Rightarrow \sum_i |F_{i,x}| \leq \mu \sum_i F_z \Rightarrow |\ddot{x}| \leq \mu(\ddot{z} + g).$$

For instance, if I keep my center of mass at a constant height, then $\ddot{z} = 0$ and $|\ddot{x}| \leq \mu g$ ; this is a nice reminder of just how important friction is if you want to be able to move around in the world.

Even better, let us define the "center of pressure" (CoP) as the point on the ground where

$$x_{cop} = \frac{\sum_i p_{i,x} F_{i,z}}{\sum_i F_{i,z}},$$

and since all $p_{i,z}$ are equal on flat terrain, $z_{cop}$ is just the height of the terrain. It turns out that the center of pressure is a "zero-moment point" (ZMP) -- a property that we will demonstrate below -- and moment-balance equation gives us a very important relationship between the location of the CoP and the dynamics of the CoM:

$$(m\ddot{z} + mg)(x_{cop} - x) = (z_{cop} - z)m\ddot{x} - I\ddot{\theta}.$$

If we use the strategy proposed above for ignoring collision dynamics, $\ddot{z} = \ddot{\theta} = 0$, then we have $z - z_{cop}$ is a constant height $h$, and the result is the famous "ZMP equations":

$$\ddot{x} = -\frac{g}{h}(x_{cop} - x).$$

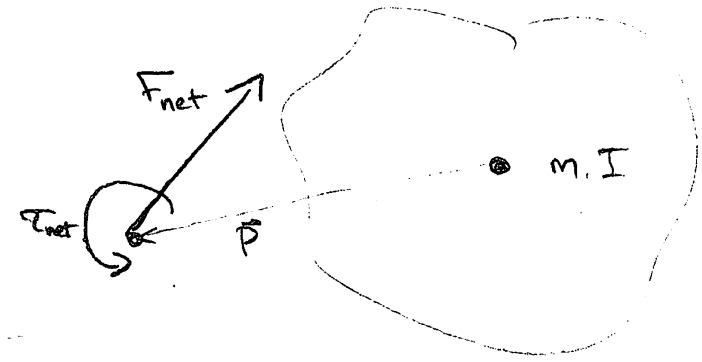
So the location of the center of pressure completely determines the acceleration of the center of mass, and vice versa! What's more, this relationship is affine -- a property that we can exploit in a number of ways.

As an example, we can easily relate constraints on the CoP to constraints on $\ddot{x}$. It is easy to verify from the definition that the CoP must live inside the convex hull of the ground contact points. Therefore, if we use the $\ddot{z} = \ddot{\theta} = 0$ strategy, then this directly implies strict bounds on the possible accelerations of the center of mass given the locations of the ground contacts.

### 5.2.1 An aside: the zero-moment point derivation

The zero-moment point (ZMP) is discussed very frequently in the current literature on legged robots. It also has an unfortunate tendency to be surrounded by some confusion; a number of people have defined the ZMP is slightly different ways (see e.g. [2] for a summary). Therefore, it makes sense to provide a simple derivation here.

First let us recall that for rigid body systems I can always summarize the contributions from many external forces as a single *wrench* (force and torque) on the object -- this is simply because the $F_i$ terms enter our equations of motion linearly. Specifically, given any point in space, $r$, in coordinates relative to $(x, z)$:




I can re-write the equations of motion as

$$
\begin{aligned}
m\ddot{x} &= \sum_i F_{i,x} = F_{net,x}, \\
m\ddot{z} &= \sum_i F_{i,z} - mg = F_{net,z} - mg, \\
I\ddot{\theta} &= \sum_i [r_i \times F_i]_y = (\mathbf{r} \times \mathbf{F}_{net})_y + \tau_{net},
\end{aligned}
$$

where $\mathbf{F}_{net} = \sum_i \mathbf{F}_i$ and the value of $\tau_{net}$ depends on the location $\mathbf{r}$. For some choices of $\mathbf{r}$, we can make $\tau_{net} = 0$. Examining

$$
(\mathbf{r} \times \mathbf{F}_{net})_y = r_z F_{net,x} - r_x F_{net,z} = [r_i \times F_i]_y,
$$

we can see that when $F_{net,z} > 0$ there is an entire line of solutions, $r_x = a r_z + b$, including one that will intercept the terrain. For walking robots, it is this point on the ground from which the external wrench can be described by a single force vector (and no moment) that is the famous "zero-moment point" (ZMP). Substituting the back in to replace $F_{net}$, we can see that

$$
r_x = \frac{r_z m \ddot{x} - I\ddot{\theta}}{m\ddot{z} + mg}.
$$

If we assume that $\ddot{z} = \ddot{\theta} = 0$ and replace the relative coordinates with the global coordinates $\mathbf{r} = \mathbf{p} - [x, 0, z]^T$, then we arrive at precisely the equation presented

above.

Furthermore, since

$$[r_i \times F_i]_y = \sum_i (r_{i,z} F_{i,x} - r_{i,x} F_{i,z}),$$

and for *flat terrain* we have

$$r_z F_{net,x} = \sum_i r_{i,z} F_{i,x},$$

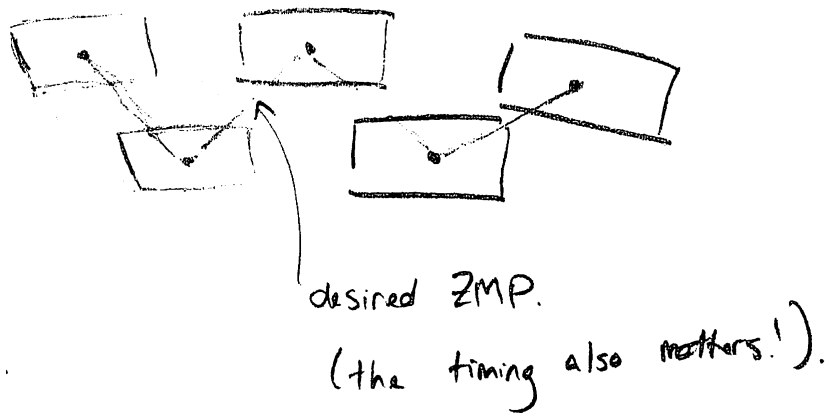
then we can see that this ZMP is exactly the CoP:

$$r_x = \frac{\sum_i r_{i,x} F_{i,z}}{F_{net,z}}.$$

In three dimensions, we solve for the point on the ground where $\tau_{net,y} = \tau_{net,x} = 0$, but allow $\tau_{net,z} \neq 0$ to extract the analogous equations in the $y$-axis:

$$r_y = \frac{r_z m\ddot{y} - I\ddot{\theta}}{m\ddot{z} + mg}.$$

## 5.3 ZMP PLANNING




### 5.3.1 From a CoM plan to a whole-body plan

## 5.4 WHOLE-BODY CONTROL

Coming soon. For a description of our approach with Atlas, see [3, 4].

## 5.5 FOOTSTEP PLANNING AND PUSH RECOVERY

Coming soon. For a description of our approach with Atlas, see [5, 4]. For nice geometric insights on push recovery, see [6].

## 5.6 BEYOND ZMP PLANNING

Coming soon. For a description of our approach with Atlas, see [7, 4].

### Example 5.1 (LittleDog gait optimization)

Open in Colab

## 5.7 Exercises

### Exercise 5.1 (Footstep Planning via Mixed-Integer Optimization)

In this exercise we implement a simplified version of the footstep planning method proposed in [5]. You will find all the details in this notebook. Your goal is to code most of the components of the mixed-integer program at the core of the footstep planner:

a. The constraint that limits the maximum step length.
b. The constraint for which a foot cannot be in two stepping stones at the same time.
c. The constraint that assigns each foot to a stepping stone, for each step of the robot.
d. The objective function that minimizes the sum of the squares of the step lengths.

## 5.8 References


1. David E. Orin and Ambarish Goswami and Sung-Hee Lee, "Centroidal dynamics of a humanoid robot", *Autonomous Robots*, no. September 2012, pp. 1--16, jun, 2013.

2. A. Goswami, "Postural stability of biped robots and the foot rotation indicator ({FRI}) point", *International Journal of Robotics Research*, vol. 18, no. 6, 1999.

3. Scott Kuindersma and Frank Permenter and Russ Tedrake, "An Efficiently Solvable Quadratic Program for Stabilizing Dynamic Locomotion", *Proceedings of the International Conference on Robotics and Automation*, May, 2014. [ link ]

4. Scott Kuindersma and Robin Deits and Maurice Fallon and Andr\'{e}s Valenzuela and Hongkai Dai and Frank Permenter and Twan Koolen and Pat Marion and Russ Tedrake, "Optimization-based Locomotion Planning, Estimation, and Control Design for the {A}tlas Humanoid Robot", *Autonomous Robots*, vol. 40, no. 3, pp. 429--455, 2016. [ link ]

5. Robin Deits and Russ Tedrake, "Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization", *Proceedings of the 2014 IEEE/RAS International Conference on Humanoid Robots (Humanoids 2014)*, 2014. [ link ]

6. Twan Koolen and Tomas de Boer and John Rebula and Ambarish Goswami and Jerry Pratt, "Capturability-based analysis and control of legged locomotion, Part 1: Theory and application to three simple gait models", *The International Journal of Robotics Research*, vol. 31, no. 9, pp. 1094-1113, 2012.

7. Hongkai Dai and Andr\'es Valenzuela and Russ Tedrake, "Whole-body Motion Planning with Centroidal Dynamics and Full Kinematics", *IEEE-RAS*

*International Conference on Humanoid Robots*, 2014. [ link ]

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 6
# Model Systems with Stochasticity

Open in Colab

My goals for this chapter are to build intuition for the beautiful and rich behavior of nonlinear dynamical system that are subjected to random (noise/disturbance) inputs. So far we have focused primarily on systems described by

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \quad \text{or} \quad \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]).$$

In this chapter, I would like to broaden the scope to think about

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)) \quad \text{or} \quad \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n], \mathbf{w}[n]),$$

where this additional input $\mathbf{w}$ is the (vector) output of some random process. In other words, we can begin thinking about stochastic systems by simply understanding the dynamics of our existing ODEs subjected to an additional random input.

This form is extremely general as written. $\mathbf{w}(t)$ can represent time-varying random disturbances (e.g. gusts of wind), or even constant model errors/uncertainty. One thing that we are not adding, yet, is measurement uncertainty. There is a great deal of work on observability and state estimation that study the question of how you can infer the true state of the system given noise sensor readings. For this chapter we are assuming perfect measurements of the full state, and are focused instead on the way that "process noise" shapes the long-term dynamics of the system.

I will also stick primarily to discrete time dynamics for this chapter, simply because it is easier to think about the output of a discrete-time random process, $\mathbf{w}[n]$, than a $\mathbf{w}(t)$. But you should know that all of the ideas work in continuous time, too. Also, most of our examples will take the form of *additive noise*:

$$\mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]) + \mathbf{w}[n],$$

which is a particular useful and common specialization of our general form. And this form doesn't give up much -- the disturbance on step $k$ can pass through the nonlinear function $f$ on step $k+1$ giving rich results -- but is often much easier to work with.

## 6.1 The Master Equation

Let's start by looking at some simple examples.

### Example 6.1 (A Bistable System + Noise)

Let's consider (a time-reversed version of) one of my favorite one-dimensional systems:

$$\dot{x} = x - x^3.$$

This deterministic system has stable fixed points at $x^* = \{-1, 1\}$ and an unstable fixed point at $x^* = 0$.

A reasonable discrete-time approximation of these dynamics, now with additive noise, is

$$x[n+1] = x[n] + h(x[n] - x[n]^3 + w[n]).$$

When $w[n] = 0$, this system has the same fixed points and stability properties as the continuous time system. But let's examine the system when $w[n]$ is instead the result of a *zero-mean Gaussian white noise process*, defined by:

$$\forall n, E\left[w[n]\right] = 0,$$
$$E\left[w[i]w[j]\right] = \begin{cases} \sigma^2, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Here $\sigma$ is the standard deviation of the Gaussian.

When you simulate this system for small values of $\sigma$, you will see trajectories move roughly towards one of the two fixed points (for the deterministic system), but every step is modified by the noise. In fact, even if the trajectory were to arrive exactly at what was once a fixed point, it is almost surely going to move again on the very next step. In fact, if we plot many runs of the simulation from different initial conditions all on the same plot, we will see something like the figure below.

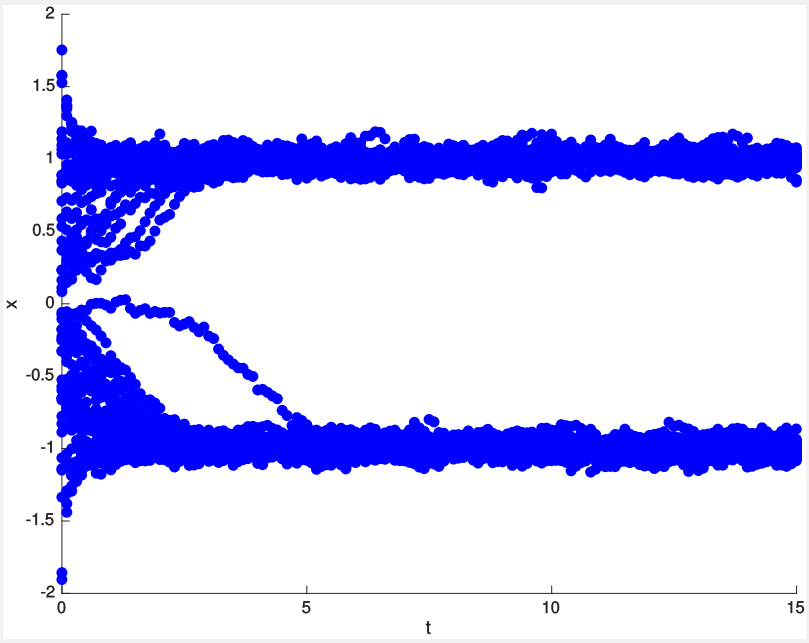



Figure 6.1 - Simulation of the bistable system with noise ($\sigma = 0.01$) from many initial conditions.

During any individual simulation, the state jumps around randomly for all time, and could even transition from the vicinity of one fixed point to the other fixed point. Another way to visualize this output is to animate a histogram of the particles over time.

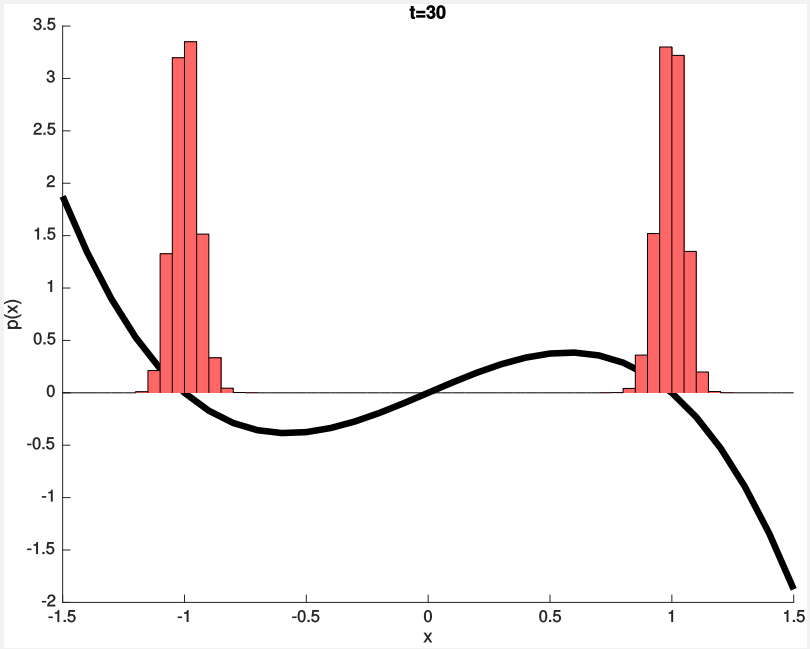



Figure 6.2 - Histogram of the states of the bistable system with noise ($\sigma = 0.01$) after simulating from random initial conditions until $t = 30$.
Click here to see the animation

You can run this demo for yourself:

Open in Colab

Let's take a moment to appreciate the implications of what we've just observed. Every time that we've analyzed a system to date, we've asked questions like "given x[0], what is the long-term behavior of the system, $\lim_{n\to\infty} x[n]$?", but now $x[n]$ is a *random variable*. The trajectories of this system do not converge, and the system does not exhibit any form of stability that we've introduced so far.

All is not lost. If you watch the animation closely, you might notice the *distribution* of this random variable is actually very well behaved. This is the key idea for this chapter.

Let us use $p_n(x)$ to denote the probability density function over the random variable $x$ at time $n$. Note that this density function must satisfy

$$\int_{-\infty}^{\infty} p_n(x)dx = 1.$$

It is actually possible to write the *dynamics of the probability density* with the simple relation

$$p_{n+1}(x) = \int_{-\infty}^{\infty} p(x|x')p_n(x')dx',$$

where $p(x|x')$ encodes the stochastic dynamics as a conditional distribution of the next state (here $x$) as a function of the current state (here $x'$). Dynamical systems that can be encoded in this way are known as *continuous-state Markov Processes*,

and the governing equation above is often referred to as the "master equation" for the stochastic process. In fact this update is even linear(!) ; a fact that can enable closed-form solutions to some impressive long-term statistics, like mean time to failure or first passage times[1]. Unfortunately, it is often difficult to perform the integral in closed form, so we must often resort to discretization or numerical analysis.

## Example 6.2 (The Logistic Map)

In fact, one does not actually need stochastic dynamics in order for the dynamics of the distribution to be the meaningful object of study; random initial conditions can be enough. One of the best examples comes from perhaps the simplest and most famous example of a chaotic system: the logistic map. This example is described beautifully in [2].

Consider the following difference equation:

$$x[n+1] = 4x[n](1 - x[n]),$$

which we will study over the (invariant) interval $x \in [0, 1]$.

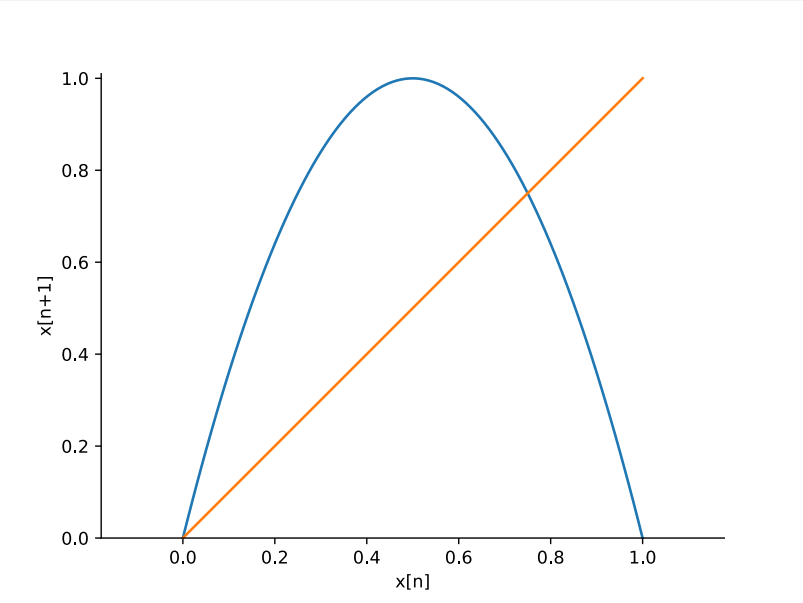



Figure 6.3 - The discrete-time dynamics of the "logistic map" (plotted alongside the line $x[n+1] = x[n]$).

It takes only a moment of tracing your finger along the plot using the "staircase method" to see what makes this system so interesting -- rollouts from a single initial condition end up bouncing all over the interval $(0, 1)$, and neighboring initial conditions will end up taking arbitrarily different trajectories (this is the hallmark of a "chaotic" system).

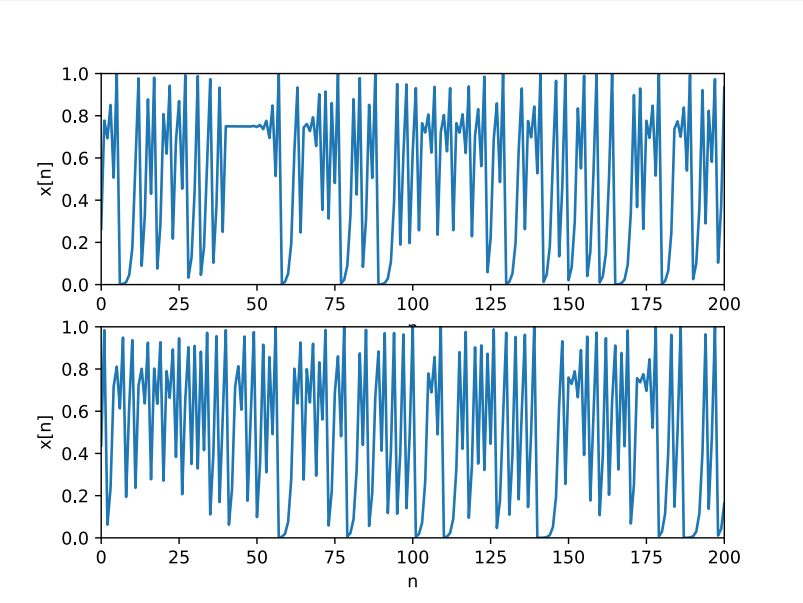



Figure 6.4 - Two simulations of the logistic map from different initial conditions. Remember, there is no stochasticity here -- the dynamics are entirely deterministic!

Here's what's completely fascinating -- even though the dynamics of any one initial condition for this system are extremely complex, if we study the dynamics of a distribution of states through the system, they are surprisingly simple and well-behaved. This system is one of the rare cases when we can write the master equation in closed form[2]:

$$p_{n+1}(x) = \frac{1}{4\sqrt{1-x}}\left[p_n\left(\frac{1}{2} - \frac{1}{2}\sqrt{1-x}\right) + p_n\left(\frac{1}{2} + \frac{1}{2}\sqrt{1-x}\right)\right].$$

Moreover, this master equation has a steady-state solution:

$$p_*(x) = \frac{1}{\pi\sqrt{x(1-x)}}.$$

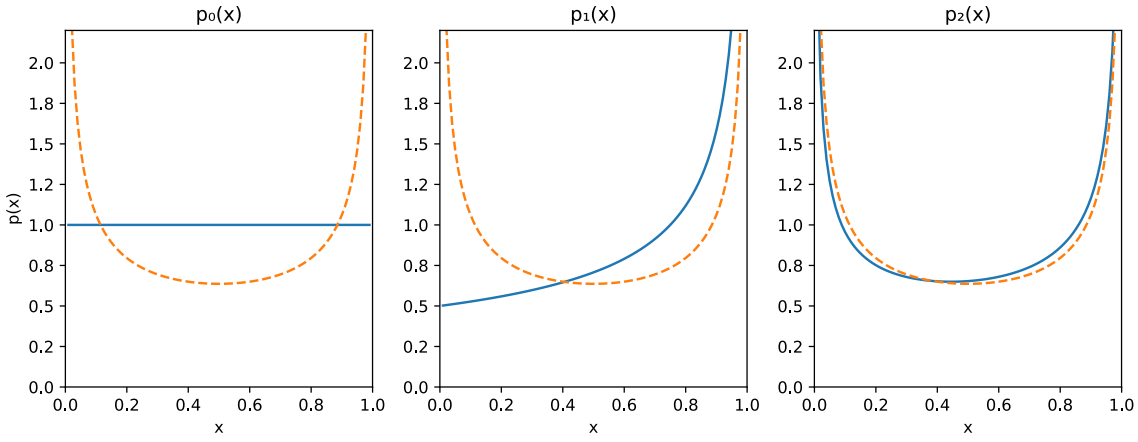



Figure 6.5 - Plotting the (closed-form) evolution of the master equation for the logistic map, initialized with $p_0(x) = 1$, reveals surprising simplicity, and rapid convergence to a stationary distribution (dashed orange line).

For this system (and many chaotic systems), the dynamics of a single initial condition are complicated, but the dynamics a *distribution* of initial conditions are

beautiful and simple!

Note: when the dynamical system under study has deterministic dynamics (but a distribution of initial conditions), the linear map given by the master equation is known as the *Perron-Frobenius operator*, and it gives rise to the Liouville equation that we will study later in the chapter.

The slightly more general form of the master equation, which works for multivariate distributions with state-domain $\mathcal{X}$, and systems with control inputs $\mathbf{u}$, is

$$p_{n+1}(\mathbf{x}) = \int_{\mathcal{X}} p(\mathbf{x}|\mathbf{x}', \mathbf{u}) p_n(\mathbf{x}') d\mathbf{x}'.$$

This is a (continuous-state) Markov Decision Process.

Continuous-time formulations are also possible -- these lead to the so-called Fokker-Planck equation.

## 6.2 Stationary Distributions

In the example above, the histogram is our numerical approximation of the probability density. If you simulate it a few times, you will probably believe that, although the individual trajectories of the system *do not* converge, the probability distribution actually *does* converge to what's known as a *stationary distribution* -- a fixed point of the master equation. Instead of thinking about the dynamics of the trajectories, we need to start thinking about the dynamics of the distribution.

### Example 6.3 (The Linear Gaussian System)

Let's consider the one-dimensional linear system with additive noise:

$$x[n+1] = ax[n] + w[n].$$

When $w[n] = 0$, the system is stable (to the origin) for $-1 < a < 1$. Let's make sure that we can understand the dynamics of the master equation for the case when $w[n]$ is Gaussian white noise with standard deviation $\sigma$.

First, recall that the probability density function of a Gaussian with mean $\mu$ is given by

$$p(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w-\mu)^2}{2\sigma^2}}.$$

When conditioned on $x[n]$, the distribution given by the dynamics subjected to mean-zero Gaussian white noise is simply another Gaussian, with the mean given by $ax[n]$:

$$p(x[n+1]|x[n]) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x[n+1]-ax[n])^2}{2\sigma^2}},$$

yielding the master equation

$$p_{n+1}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{(x-ax')^2}{2\sigma^2}} p_n(x') dx'.$$

Now here's the magic. Let's push a distribution, $p_n(x)$, which is zero-mean, with standard deviation $\sigma_n$, through the master equation:

$$p_{n+1}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{1}{\sqrt{2\pi\sigma_n^2}} \int_{-\infty}^{\infty} e^{-\frac{(x-ax')^2}{2\sigma^2}} e^{-\frac{(x')^2}{2\sigma_n^2}} dx',$$
$$= \frac{1}{\sqrt{2\pi(\sigma^2 + a^2\sigma_n^2)}} e^{-\frac{x^2}{2(\sigma^2+a^2\sigma_n^2)}}.$$

The result is another mean-zero Gaussian with $\sigma_{n+1}^2 = \sigma^2 + a^2\sigma_n^2$. This result generalizes to the multi-variate case, too, and might be familiar to you e.g. from the process update of the Kalman filter.

Taking it a step further, we can see that a stationary distribution for this system is given by a mean-zero Gaussian with

$$\sigma_*^2 = \frac{\sigma^2}{1 - a^2}.$$

Note that this distribution is well defined when $-1 < a < 1$ (only when the system is stable).

The stationary distribution of the linear Gaussian system reveals the fundamental and general balance between two terms in the governing equations of any stochastic dynamical system: the stability of the deterministic system is bringing trajectories together (smaller $a$ means faster convergence of the deterministic system and results in a more narrow distribution), but the noise in the system is forcing trajectories apart (larger $\sigma$ means larger noise and results in a wider distribution).

Given how rich the dynamics can be for deterministic nonlinear systems, you can probably imagine that the possible long-term dynamics of the probability are also extremely rich. If we simply flip the signs in the dynamics we examined above, we'll get our next example:

## Example 6.4 (The Cubic Example + Noise)

Now let's consider the discrete-time approximation of

$$\dot{x} = -x + x^3,$$

again with additive noise:

$$x[n+1] = x[n] + h(-x[n] + x[n]^3 + w[n]).$$

With $w[n] = 0$, the system has only a single stable fixed point (at the origin), whose deterministic region of attraction is given by $x \in (-1, 1)$. If we again simulate the system from a set of random initial conditions and plot the histogram, we will see something like the figure below.

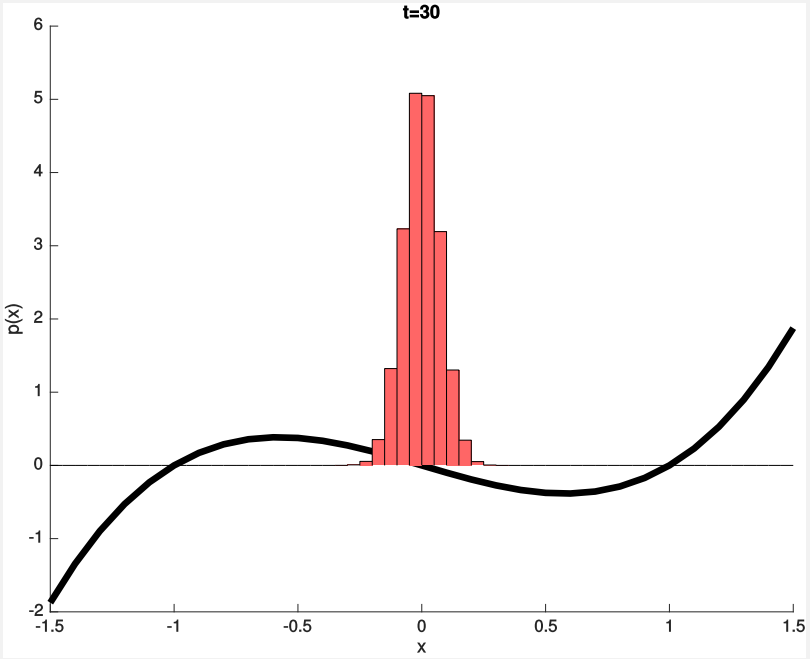



Figure 6.6 - Histogram of the states of the bistable system with noise ($\sigma = 0.01$) after simulating from random initial conditions until $t = 30$.
Click here to see the animation

Be sure to watch the animation. Or better yet, run the simulation for yourself by changing the sign of the derivative in the bistability example and re-running:

You can run this demo for yourself:

Open in Colab

Now try increasing the noise (e.g. pre-multiply the noise input, $w$, in the dynamics by a scalar like 2).

Click here to see the animation

What is the stationary distribution for this system? In fact, there isn't one. Although we initially collect probability density around the stable fixed point, you should notice a slow leak -- on every step there is some probability of transitioning past unstable fixed points and getting driven by the unstable dynamics off towards infinity. If we run the simulation long enough, there won't be any probability density left at $x = 0$.

## Example 6.5 (The Stochastic Van der Pol Oscillator)

One more example; this is a fun one. Let's think about one of the simplest examples that we had for a system that demonstrates limit cycle stability -- the Van der Pol oscillator -- but now we'll add Gaussian white noise,

$$\ddot{q} + (q^2 - 1)\dot{q} + q = w(t),$$

Here's the question: if we start with a small set of initial conditions centered around one point on the limit cycle, then what is the long-term behavior of this distribution?

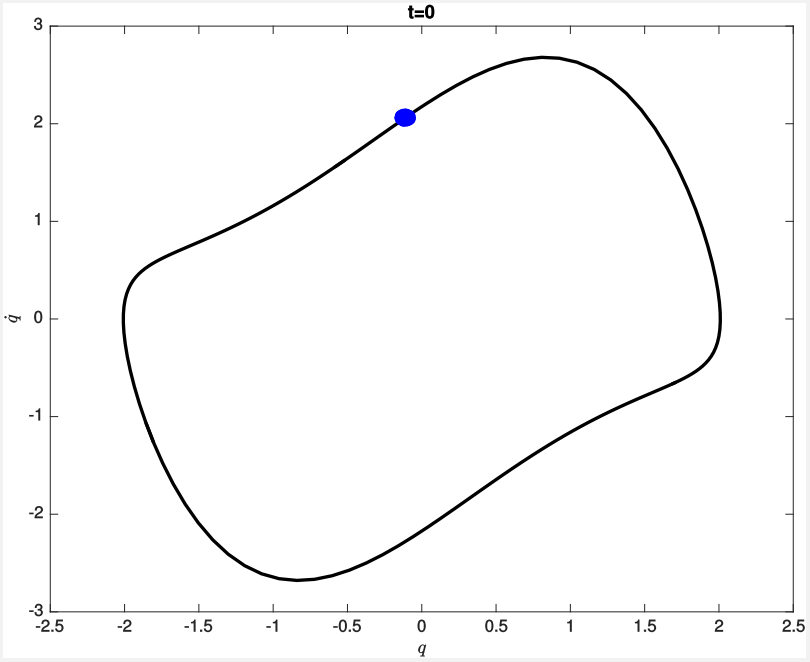



Figure 6.7 - Randomly sampled initial conditions pictured with the stable limit cycle of the Van der Pol oscillator.

Since the long-term behavior of the deterministic system is periodic, it would be very logical to think that the state distribution for this stochastic system would fall into a stable periodic solution, too. But think about it a little more, and then watch the animation (or run the simulation yourself).

Open in Colab

Click here to see the animation (first 20 seconds)

Click here to see the particles after 2000 seconds of simulation

The explanation is simple: the periodic solution of the system is only *orbitally stable*; there is no stability along the limit cycle. So any disturbances that push the particles along the limit cycle will go unchecked. Eventually, the distribution will "mix" along the entire cycle. Perhaps surprisingly, this system that has a limit cycle stability when $w = 0$ eventually reaches a stationary distribution (fixed point) in the master equation.

## 6.3 Extended Example: The Rimless Wheel on Rough Terrain

My favorite example of a meaningful source of randomness on a model underactuated system is the rimless wheel rolling down stochastically "rough" terrain[3]. Generating interesting/relevant probabilistic models of terrain in general can be quite complex, but the rimless wheel makes it easy -- since the robot only contacts that ground at the point foot, we can model almost arbitrary rough terrain by simply taking the ramp angle, $\gamma$, to be a random variable. If we restrict our analysis to rolling only in one direction (e.g. only downhill), then we can even consider this ramp angle to be i.i.d.; after each footstep we will independently draw a new ramp angle $\gamma[n]$ for the next step.

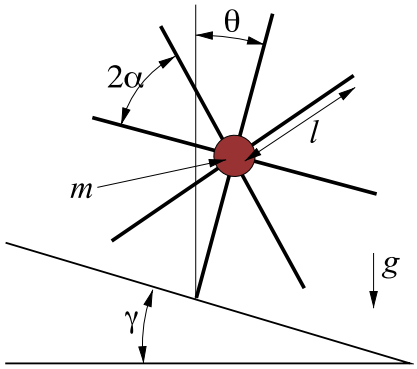



Figure 6.8 - The rimless wheel. The orientation of the stance leg, $\theta$, is measured clockwise from the vertical axis.

In our original analysis of the rimless wheel, we derived the "post-collision" return map -- mapping the angular velocity from the beginning of the stance phase to the (post-collision) angular velocity at the next stance phase. But now that the relative location of the ground is changing on every step, we instead write the "apex-to-apex" return map, which maps the angular velocity from one apex (the moment that the leg is vertical) to the next, which is given by:

$$\dot{\theta}[n+1] = \sqrt{\cos^2\alpha\left(\dot{\theta}^2[n] + \frac{2g}{l}(1-\cos(\alpha+\gamma[n]))\right) - \frac{2g}{l}(1-\cos(\alpha-\gamma[n]))}.$$

More coming soon. Read the paper [3] and/or watch the video.

## 6.4 Noise models for real robots/systems.

Sensor models. Beam model from probabilistic robotics. RGB-D dropouts.

Perception subsystem. Output of a perception system is not Gaussian noise, it's missed detections/drop-outs...

Distributions over tasks/environments.

## References

1. Katie Byl and Russ Tedrake, "Metastable Walking on Stochastically Rough Terrain", *Proceedings of Robotics: Science and Systems IV*, 2008. [ link ]

2. Andrzej Lasota and Michael C Mackey, "Chaos, fractals, and noise: stochastic aspects of dynamics",Springer Science \& Business Media , vol. 97, 2013.

3. Katie Byl and Russ Tedrake, "Metastable Walking Machines", *International Journal of Robotics Research*, vol. 28, no. 8, pp. 1040-1064, August 1, 2009. [ link ]

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 7
# Dynamic Programming

Open in Colab

In chapter 2, we spent some time thinking about the phase portrait of the simple pendulum, and concluded with a challenge: can we design a nonlinear controller to *reshape* the phase portrait, with a very modest amount of actuation, so that the upright fixed point becomes globally stable? With unbounded torque, feedback-cancellation solutions (e.g., invert gravity) can work well, but can also require an unnecessarily large amount of control effort. The energy-based swing-up control solutions presented for the acrobot and cart-pole systems are considerably more appealing, but required some cleverness and might not scale to more complicated systems. Here we investigate another approach to the problem, using computational optimal control to synthesize a feedback controller directly.

## 7.1 Formulating control design as an optimization

In this chapter, we will introduce optimal control - a control design process using optimization. This approach is powerful for a number of reasons. First and foremost, it is very general - allowing us to specify the goal of control equally well for fully- or under-actuated, linear or nonlinear, deterministic or stochastic, and continuous or discrete systems. Second, it permits concise descriptions of potentially very complex desired behaviours, specifying the goal of control as a scalar objective (plus a list of constraints). Finally, and most importantly, optimal control is very amenable to numerical solutions. [1] is a fantastic reference on this material for those who want a somewhat rigorous treatment; [2] is an excellent (free) reference for those who want something more approachable.

The fundamental idea in optimal control is to formulate the goal of control as the *long-term* optimization of a scalar cost function. Let's introduce the basic concepts by considering a system that is even simpler than the simple pendulum.

**Example 7.1 (Optimal Control Formulations for the Double Integrator)**

Consider the double integrator system

$$\ddot{q} = u, \quad |u| \leq 1.$$

If you would like a mechanical analog of the system (I always do), then you can think about this as a unit mass brick moving along the x-axis on a frictionless surface, with a control input which provides a horizontal force, $u$. The task is to design a control system, $u = \pi(\mathbf{x}, t)$, $\mathbf{x} = [q, \dot{q}]^T$ to regulate this brick to $\mathbf{x} = [0, 0]^T$.


Figure 7.1 - The double integrator as a unit-mass brick on a frictionless surface

In order to formulate this control design problem using optimal control, we must define a scalar objective which scores the long-term performance of running each candidate control policy, $\pi(\mathbf{x}, t)$, from each initial condition, $(\mathbf{x}_0, t_0)$, and a list of constraints that must be satisfied. For the task of driving the double integrator to the origin, one could imagine a number of optimal control formulations which would accomplish the task, e.g.:

- Minimum time: $\min_\pi t_f$, subject to $\mathbf{x}(t_0) = \mathbf{x}_0$, $\mathbf{x}(t_f) = \mathbf{0}$.
- Quadratic cost: $\min_\pi \int_{t_0}^{\infty} \left[\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t)\right] dt$, $\mathbf{Q} \succ 0$.

where the first is a constrained optimization formulation which optimizes time, and the second accrues a penalty at every instance according to the distance that the state is away from the origin (in a metric space defined by the matrix $\mathbf{Q}$), and therefore encourages trajectories that go more directly towards the goal, possibly at the expense of requiring a longer time to reach the goal (in fact it will result in an exponential approach to the goal, where as the minimum-time formulation will arrive at the goal in finite time). Note that both optimization problems only have well-defined solutions if it is possible for the system to actually reach the origin, otherwise the minimum-time problem cannot satisfy the terminal constraint, and the integral in the quadratic cost would not converge to a finite value as time approaches infinity (fortunately the double integrator system is controllable, and therefore can be driven to the goal in finite time).

Note that the input limits, $|u| \leq 1$ are also required to make this problem well-posed; otherwise both optimizations would result in the optimal policy using infinite control input to approach the goal infinitely fast. Besides input limits, another common approach to limiting the control effort is to add an additional quadratic cost on the input (or "effort"), e.g. $\int \left[\mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t)\right] dt$, $\mathbf{R} \succ 0$. This could be added to either formulation above. We will examine many of these formulations in some details in the examples worked out at the end of this chapter.

Optimal control has a long history in robotics. For instance, there has been a great deal of work on the minimum-time problem for pick-and-place robotic manipulators, and the linear quadratic regulator (LQR) and linear quadratic regulator with Gaussian noise (LQG) have become essential tools for any practicing controls engineer. With increasingly powerful computers and algorithms, the popularity of numerical optimal control has grown at an incredible pace over the last few years.

## Example 7.2 (The minimum time problem for the double integrator)

For more intuition, let's do an informal derivation of the solution to the minimum time problem for the double integrator with input constraints:

$$\begin{aligned} \text{minimize}_{\pi} \quad & t_f \\ \text{subject to} \quad & \mathbf{x}(t_0) = \mathbf{x}_0, \\ & \mathbf{x}(t_f) = \mathbf{0}, \\ & \ddot{q}(t) = u(t), \\ & |u| \leq 1. \end{aligned}$$

What behavior would you expect an optimal controller exhibit?

Your intuition might tell you that the best thing that the brick can do, to reach the goal in minimum time with limited control input, is to accelerate maximally towards the goal until reaching a critical point, then hitting the brakes in order to come to a stop exactly at the goal. This would be called a *bang-bang* control policy; these are often optimal for systems with bounded input, and it is in fact optimal for the double integrator, although we will not prove it until we have developed more tools.

Let's work out the details of this bang-bang policy. First, we can figure out the states from which, when the brakes are fully applied, the system comes to rest precisely at the origin. Let's start with the case where $q(0) < 0$, and $\dot{q}(0) > 0$, and "hitting the brakes" implies that $u = -1$ . Integrating the equations, we have

$$\begin{aligned} \ddot{q}(t) &= u = -1 \\ \dot{q}(t) &= \dot{q}(0) - t \\ q(t) &= q(0) + \dot{q}(0)t - \frac{1}{2}t^2. \end{aligned}$$

Substituting $t = \dot{q}(0) - \dot{q}$ into the solution reveals that the system orbits are parabolic arcs:

$$q = -\frac{1}{2}\dot{q}^2 + c_-,$$

with $c_- = q(0) + \frac{1}{2}\dot{q}^2(0)$.

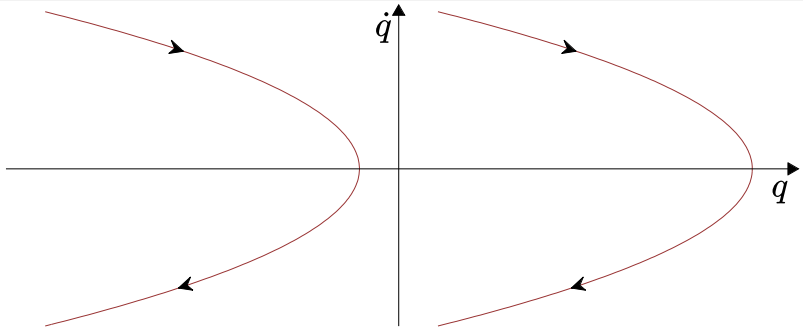



Figure 7.2 - Two solutions for the system with $u = -1$

Similarly, the solutions for $u = 1$ are $q = \frac{1}{2}\dot{q}^2 + c_+$, with $c_+ = q(0) - \frac{1}{2}\dot{q}^2(0)$.

Perhaps the most important of these orbits are the ones that pass directly through the origin (e.g., $c_- = 0$). Following our initial logic, if the system is going slower than this $\dot{q}$ for any $q$, then the optimal thing to do is to slam on the accelerator ( $u = -\text{sgn}(q)$). If it's going faster than the $\dot{q}$ that we've solved for, then still the best thing to do is to brake; but inevitably the system will overshoot the origin and have to come back. We can summarize this policy with:

$$u = \begin{cases} +1 & \text{if } (\dot{q} < 0 \text{ and } q \leq \frac{1}{2}\dot{q}^2) \text{ or } (\dot{q} \geq 0 \text{ and } q < -\frac{1}{2}\dot{q}^2) \\ 0 & \text{if } q = 0 \text{ and } \dot{q} = 0 \\ -1 & \text{otherwise} \end{cases}$$

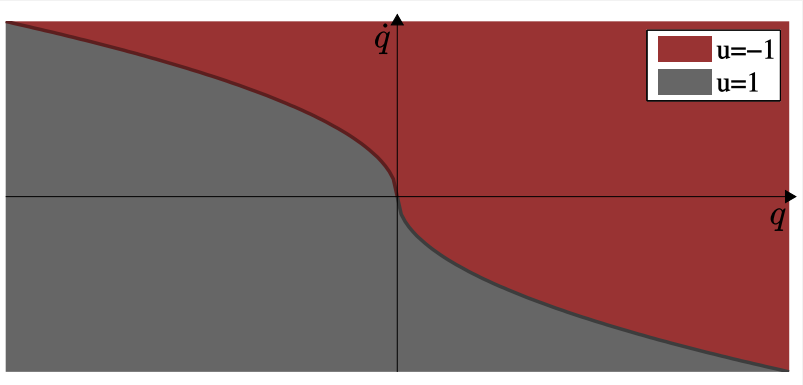



Figure 7.3 - Candidate optimal (bang-bang) policy for the minimum-time double integrator problem.

and illustrate some of the optimal solution trajectories:

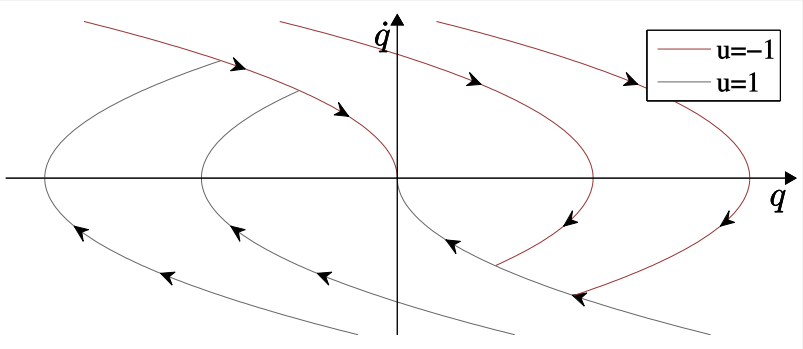



Figure 7.4 - Solution trajectories of system using the optimal policy

And for completeness, we can compute the optimal time to the goal by solving for the amount of time required to reach the switching surface plus the amount of time spent moving along the switching surface to the goal. With a little algebra, you will find that the time to goal, $T(\mathbf{x})$, is given by

$$T(\mathbf{x}) = \begin{cases} 2\sqrt{\frac{1}{2}\dot{q}^2 - q} - \dot{q} & \text{for } u = +1 \text{ regime,} \\ 0 & \text{for } u = 0, \\ \dot{q} + 2\sqrt{\frac{1}{2}\dot{q}^2 + q} & \text{for } u = -1, \end{cases}$$

plotted here:

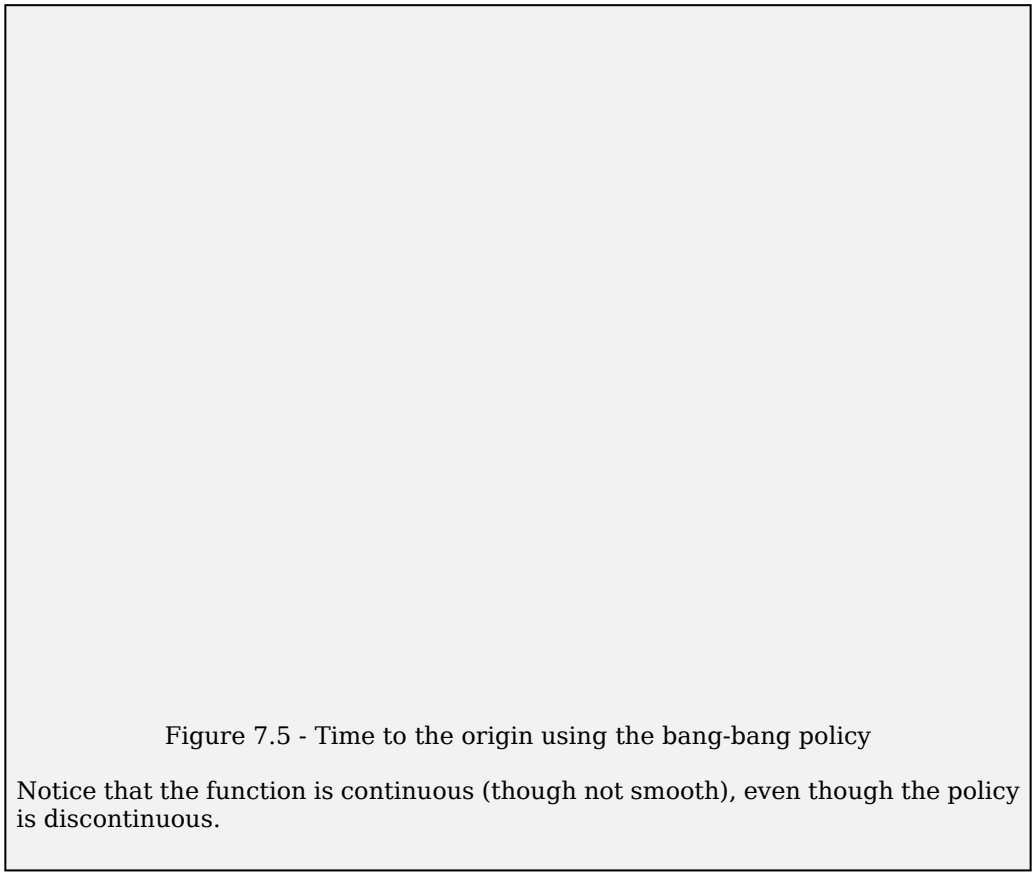


Figure 7.5 - Time to the origin using the bang-bang policy

Notice that the function is continuous (though not smooth), even though the policy is discontinuous.

### 7.1.1 Additive cost

As we begin to develop theoretical and algorithmic tools for optimal control, we will see that some formulations are much easier to deal with than others. One important example is the dramatic simplification that can come from formulating objective functions using *additive cost*, because they often yield recursive solutions. In the additive cost formulation, the long-term "score" for a trajectory can be written as

$$\int_0^T \ell(x(t), u(t))dt,$$

where $\ell()$ is the instantaneous cost (also referred to as the "running cost"), and $T$ can be either a finite real number or $\infty$. We will call a problem specification with a finite $T$ a "finite-horizon" problem, and $T = \infty$ an "infinite-horizon" problem. Problems and solutions for infinite-horizon problems tend to be more elegant, but care is required to make sure that the integral converges for the optimal controller (typically by having an achievable goal that allows the robot to accrue zero-cost).

The quadratic cost function suggested in the double integrator example above is clearly written as an additive cost. At first glance, our minimum-time problem formulation doesn't appear to be of this form, but we actually can write it as an additive cost problem using an infinite horizon and the instantaneous cost

$$\ell(x, u) = \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{otherwise.} \end{cases}$$

We will examine a number of approaches to solving optimal control problems throughout the next few chapters. For the remainder of this chapter, we will focus on additive-cost problems and their solution via *dynamic programming*.

## 7.2 OPTIMAL CONTROL AS GRAPH SEARCH

For systems with continuous states and continuous actions, dynamic programming is a set of theoretical ideas surrounding additive cost optimal control problems. For systems with a finite, discrete set of states and a finite, discrete set of actions, dynamic programming also represents a set of very efficient numerical *algorithms* which can compute optimal feedback controllers. Many of you will have learned it before as a tool for graph search.

Imagine you have a directed graph with states (or nodes) $\{s_1, s_2, \dots\} \in S$ and "actions" associated with edges labeled as $\{a_1, a_2, \dots\} \in A$, as in the following trivial example:

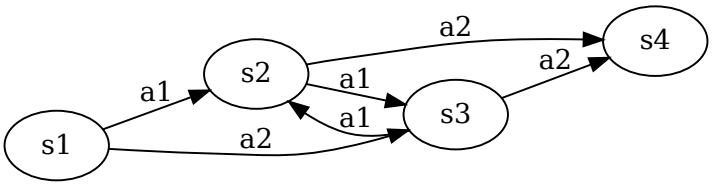



Figure 7.6 - A simple directed graph.

Let us also assume that each edge has an associate weight or cost, using $\ell(s, a)$ to denote the cost of being in state $s$ and taking action $a$. Furthermore we will denote the transition "dynamics" using

$$s[n+1] = f(s[n], a[n]).$$

For instance, in the graph above, $f(s_1, a_1) = s_2$.

There are many algorithms for finding (or approximating) the optimal path from a start to a goal on directed graphs. In dynamic programming, the key insight is that we can find the shortest path from every node by solving recursively for the optimal *cost-to-go* (the cost that will be accumulated when running the optimal controller) from every node to the goal. One such algorithm starts by initializing an estimate $\hat{J}^* = 0$ for all $s_i$, then proceeds with an iterative algorithm which sets

$$\hat{J}^*(s_i) \Leftarrow \min_{a \in A} \left[ \ell(s_i, a) + \hat{J}^*\left(f(s_i, a)\right) \right]. \tag{1}$$

In software, $\hat{J}^*$ can be represented as a vector with dimension equal to the number of discrete states. This algorithm, appropriately known as *value iteration*, is guaranteed to converge to the optimal cost-to-go up to a constant factor, $\hat{J}^* \to J^* + c$ [3], and in practice does so rapidly. Typically the update is done in *batch* -- e.g. the estimate is updated for all $i$ at once -- but the *asynchronous* version where states are updated one at a time is also known to converge, so long as every state is eventually updated infinitely often. Assuming the graph has a goal state with a zero-cost self-transition, then this cost-to-go function represents the weighted shortest distance to the goal.

Value iteration is an amazingly simple algorithm, but it accomplishes something quite amazing: it efficiently computes the long-term cost of an optimal policy from *every* state by iteratively evaluating the one-step cost. If we know the optimal cost-to-go, then it's easy to extract the optimal policy, $a = \pi^*(s)$:

$$\pi^*(s_i) = \text{argmin}_a \left[ \ell(s_i, a) + J^*\left(f(s_i, a)\right) \right]. \tag{2}$$

It's a simple algorithm, but playing with an example can help our intuition.

### Example 7.3 (Grid World)

Imagine a robot living in a grid (finite state) world. Wants to get to the goal location. Possibly has to negotiate cells with obstacles. Actions are to move up, down, left, right, or do nothing. [4]

Click here for the animation.

Figure 7.7 - The one-step cost for the grid-world minimum-time problem. The goal state has a cost of zero, the obstacles have a cost of 10, and every other state has a cost of 1.

Open in Colab

### Example 7.4 (Dynamic Programming for the Double Integrator)

You can run value iteration for the double integrator (using barycentric interpolation to interpolate between nodes) in DRAKE using:

Open in Colab

Please do take some time to try different cost functions by editing the code yourself.

Let's take a minute to appreciate how amazing this is. Our solution to finding the optimal controller for the double integrator wasn't all that hard, but it required some mechanical intuition and solutions to differential equations. The resulting policy was non-trivial -- bang-bang control with a parabolic switching surface. The value iteration algorithm doesn't use any of this directly -- it's a simple algorithm for graph search. But remarkably, it can generate effectively the same policy with just a few moments of computation.

It's important to note that there *are* some differences between the computed policy and the optimal policy that we derived, due to discretization errors. We will ask you to explore these in the problems.

The real value of this numerical solution, however, is unlike our analytical solution for the double integrator, we can apply this same algorithm to any number of dynamical systems virtually without modification. Let's apply it now to the simple pendulum, which was intractable analytically.

### Example 7.5 (Dynamic Programming for the Simple Pendulum)

You can run value iteration for the simple pendulum (using barycentric interpolation to interpolate between nodes) in DRAKE using:

Open in Colab

Again, you can easily try different cost functions by editing the code yourself.

## 7.3 CONTINUOUS DYNAMIC PROGRAMMING

I find the graph search algorithm extremely satisfying as a first step, but also become quickly frustrated by the limitations of the discretization required to use it. In many cases, we can do better; coming up with algorithms which work more natively on continuous dynamical systems. We'll explore those extensions in this section.

### 7.3.1 The Hamilton-Jacobi-Bellman Equation

It's important to understand that the value iteration equations, equations (1) and (2), are more than just an algorithm. They are also sufficient conditions for optimality: if we can produce a $J^*$ and $\pi^*$ which satisfy these equations, then $\pi^*$ must be an optimal controller. There are an analogous set of conditions for the continuous systems. For a system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ and an infinite-horizon additive cost $\int_0^\infty \ell(\mathbf{x}, \mathbf{u}) dt$, we have:

$$0 = \min_{\mathbf{u}} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \right], \tag{3}$$

$$\pi^*(\mathbf{x}) = \text{argmin}_{\mathbf{u}} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \right]. \tag{4}$$

Equation 3 is known as the *Hamilton-Jacobi-Bellman* (HJB) equation. [5] gives an informal derivation of these equations as the limit of a discrete-time approximation of the dynamics, and also gives the famous "sufficiency theorem". The treatment in [5] is for the finite-horizon case; I have modified it to one particular form of an infinite-horizon case here.

Technically, a Hamilto Jacobi equation is a P whose *time derivative* depends on the first- order partial derivati over state, which we in the finite-time deriviation; Eq 3 is th steady-state solution the Hamilton-Jacobi equation.

**Theorem 7.1 - HJB Sufficiency Theorem *(stated informally)***

Consider a system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ and an infinite-horizon additive cost $\int_0^\infty \ell(\mathbf{x}, \mathbf{u}) dt$, with $f$ and $\ell$ continuous functions, and $\ell$ a strictly-positive-definite function that obtains zero only when $x = \mathbf{x}^*$. Suppose $J(\mathbf{x})$ is a positive-definite solution to the HJB equation; that is $J$ is continuously differentiable in $\mathbf{x}$ and is such that

$$0 = \min_{\mathbf{u} \in U} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \right], \quad \text{for all } \mathbf{x},$$

and that $\pi(\mathbf{x})$ is the minimizer for all $\mathbf{x}$. Then, under some technical conditions on the existence and boundedness of solutions, we have that $J(\mathbf{x}) - J(\mathbf{x}^*)$ is the optimal cost-to-go and $\pi$ is an optimal policy.

Here is a more formal version from a personal communication with Sasha Megretski.

Given an open subset $\Omega \subset \mathbb{R}^n$, with a selected element $x^*$, a subset $U \subset \mathbb{R}^m$, continuous functions $f : \Omega \times U \to \mathbb{R}^n$, $g : \Omega \times U \to [0, \infty)$, continuously differentiable function $V : \Omega \to [0, \infty)$, and a function $\mu : \Omega \to U$, such that

a. $g(x, u)$ is strictly positive definite, in the sense that $\lim_{k\to\infty} x_k = x^*$ for every sequence of vectors $x_k \in \Omega$, $u_k \in U$ $(k = 1, 2, \ldots)$ such that $\lim_{k\to\infty} g(x_k, u_k) = 0$;
b. the function $g_V : \Omega \times U \to \mathbb{R}$, defined by

$$g_V(x, u) = g(x, u) + \frac{dV(x)}{dx} f(x, u), \tag{5}$$

is non-negative, and achieves zero value whenever $u = \mu(x)$;

c. for every $x_0 \in \Omega$, the system of equations

$$\dot{x}(t) = f(x(t), u(t)),\ x(0) = x_0 \quad \left(\text{i.e., } x(t) = x_0 + \int_0^t f(x(\tau), u(\tau) d\tau\right) \tag{6}$$

$$u(t) = \mu(x(t)), \tag{7}$$

has at least one piecewise continuous solution $x : [0, \infty) \to \Omega$, $u : [0, \infty) \to U$.

Then, for every $x_0 \in \Omega$, the functional

$$J(x, u) = \int_0^\infty g(x(t), u(t)) dt,$$

defined on the set of all piecewise continuous pairs $(x, u)$ satisfying (6), achieves its minimal value of $V(x_0) - V(x^*)$ at every piecewise continuous solution of (6), (7).

**Proof.** First, observe that, for all piecewise continuous $(x, u)$ satisfying (6),

I. due to (5),

$$\int_0^T g(x(t), u(t)) dt = V(x_0) - V(x(T)) + \int_0^T g_V(x(t), u(t)) dt; \tag{8}$$

II. whenever $J(x, u) < \infty$, there exists a sequece of positive numbers $T_k$, $k = 1, 2, \ldots$, $T_k \to +\infty$, such that $g(x(T_k), u(T_k)) \to 0$, which, according to assumption (a), implies $x(T_k) \to x^*$ as $k \to \infty$.

In particular, for a piecewise continuous solution $(x, u)$ of (6), satisfying, in addition, equation (7), condition (I) means

$$\int_0^T g(x(t), u(t)) dt = V(x_0) - V(x(T)) \le V(x_0).$$

Since the upper bound $V(x_0)$ does not depend on $T$, and $g(x(t), u(t)) \ge 0$ is non-negative, we conclude that

$$\int_0^\infty g(x(t), u(t)) dt < \infty.$$

Therefore observation (**) applies, and, for the corresponding $T_k$,

$$\begin{aligned} J(x, u) = \int_0^\infty g(x(t), u(t)) dt &= \lim_{k \to \infty} \int_0^{T_k} g(x(t), u(t)) dt \\ &= \lim_{k \to \infty} V(x_0) - V(x(T_k)) = V(x_0) - V(x^*). \end{aligned}$$

Moreover, since $g$ is non-negative, the inequality $V(x_0) \ge V(x^*)$ must hold for every $x_0 \in \Omega$.

To finish the proof, for an arbitrary piecewise continuous solution of (6), equation (8), together with non-negativity of $g_V$, implies

$$J(x, u) = \int_0^T g(x(t), u(t)) dt \ge V(x_0) - V(x(T)).$$

When $J(x,u) < \infty$, applying this to $T = T_k$, where $T_k$ are described in observation (II), and taking the limit as $k \to \infty$ (and $x(T_k) \to x^*$) yields $J(x,u) \geq V(x_0) - V(x^*)$. $\square$

It is possible to prove sufficiency under different assumptions, too. The particular assumptions here were chosen to ensure that $J(x(0)) < \infty$ implies that $x(t) \to x^*$. As Sasha says, "without something like this, all sorts of counter-examples emerge."

As a tool for verifying optimality, the HJB equations are actually surprisingly easy to work with: we can verify optimality for an infinite-horizon objective without doing any integration; we simply have to check a derivative condition on the optimal cost-to-go function $J^*$. Let's see this play out on the double integrator example.

## Example 7.6 (HJB for the Double Integrator)

Consider the problem of regulating the double integrator (this time without input limits) to the origin using a quadratic cost:

$$\ell(\mathbf{x}, \mathbf{u}) = q^2 + \dot{q}^2 + u^2.$$

I claim (without derivation) that the optimal controller for this objective is

$$\pi(\mathbf{x}) = -q - \sqrt{3}\dot{q}.$$

To convince you that this is indeed optimal, I have produced the following cost-to-go function:

$$J(\mathbf{x}) = \sqrt{3}q^2 + 2q\dot{q} + \sqrt{3}\dot{q}^2.$$

Taking

$$\frac{\partial J}{\partial q} = 2\sqrt{3}q + 2\dot{q}, \qquad \frac{\partial J}{\partial \dot{q}} = 2q + 2\sqrt{3}\dot{q},$$

we can write

$$\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) = q^2 + \dot{q}^2 + u^2 + (2\sqrt{3}q + 2\dot{q})\dot{q} + (2q + 2\sqrt{3}\dot{q})u$$

This is a convex quadratic function in $u$, so we can find the minimum with respect to $u$ by finding where the gradient with respect to $u$ evaluates to zero.

$$\frac{\partial}{\partial u}\left[\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u})\right] = 2u + 2q + 2\sqrt{3}\dot{q}.$$

Setting this equal to $0$ and solving for $u$ yields:

$$u^* = -q - \sqrt{3}\dot{q},$$

thereby confirming that our policy $\pi$ is in fact the minimizer. Substituting $u^*$ back into the HJB reveals that the right side does in fact simplify to zero. I hope you are convinced!

Note that evaluating the HJB for the time-to-go of the minimum-time problem for the double integrator will also reveal that the HJB is satisfied wherever that gradient is well-defined. This is certainly mounting evidence in support of our bang-bang policy being optimal, but since $\frac{\partial J}{\partial \mathbf{x}}$ is not defined everywhere, it does not actually satisfy the requirements of the sufficiency theorem as stated above. In some sense, the assumption in the sufficiency theorem that $\frac{\partial J}{\partial \mathbf{x}}$ is defined everywhere makes it very

weak.

### 7.3.2 Solving for the minimizing control

We still face a few barriers to actually using the HJB in an algorithm. The first barrier is the minimization over $u$. When the action set was discrete, as in the graph search version, we could evaluate the one-step cost plus cost-to-go for every possible action, and then simply take the best. For continuous action spaces, in general we cannot rely on the strategy of evaluating a finite number of possible $\mathbf{u}$'s to find the minimizer.

All is not lost. In the quadratic cost double integrator example above, we were able to solve explicitly for the minimizing $\mathbf{u}$ in terms of the cost-to-go. It turns out that this strategy will actually work for a number of the problems we're interested in, even when the system (which we are given) or cost function (which we are free to pick, but which should be expressive) gets more complicated.

Recall that I've already tried to convince you that a majority of the systems of interest are *control affine*, e.g. I can write

$$f(\mathbf{x}, \mathbf{u}) = f_1(\mathbf{x}) + f_2(\mathbf{x})\mathbf{u}.$$

We can make another dramatic simplification by restricting ourselves to instantaneous cost functions of the form

$$\ell(\mathbf{x}, \mathbf{u}) = \ell_1(\mathbf{x}) + \mathbf{u}^T\mathbf{R}\mathbf{u}, \qquad \mathbf{R} = \mathbf{R}^T \succ 0.$$

In my view, this is not very restrictive - many of the cost functions that I find myself choosing to write down can be expressed in this form. Given these assumptions, we can write the HJB as

$$0 = \min_{\mathbf{u}} \left[ \ell_1(\mathbf{x}) + \mathbf{u}^T\mathbf{R}\mathbf{u} + \frac{\partial J}{\partial \mathbf{x}}[f_1(\mathbf{x}) + f_2(\mathbf{x})\mathbf{u}] \right].$$

Since this is a positive quadratic function in $\mathbf{u}$, if the system does not have any constraints on $\mathbf{u}$, then we can solve in closed-form for the minimizing $\mathbf{u}$ by taking the gradient of the right-hand side:

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R} + \frac{\partial J}{\partial \mathbf{x}} f_2(\mathbf{x}) = 0,$$

and setting it equal to zero to obtain

$$\mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1} f_2^T(\mathbf{x}) \frac{\partial J}{\partial \mathbf{x}}^T.$$

If there are linear constraints on the input, such as torque limits, then more generally this could be solved (at any particular $\mathbf{x}$) as a quadratic program.

What happens in the case where our system is not control affine or if we really do need to specify an instantaneous cost function on $\mathbf{u}$ that is not simply quadratic? If the goal is to produce an iterative algorithm, like value iteration, then one common approach is to make a (positive-definite) quadratic approximation in $\mathbf{u}$ of the HJB, and updating that approximation on every iteration of the algorithm. This broad approach is often referred to as *differential dynamic programming* (c.f. [6]).

### 7.3.3 Numerical solutions for $J^*$

The other major barrier to using the HJB in a value iteration algorithm is that the estimated optimal cost-to-go function, $\hat{J}^*$, must somehow be represented with a finite set of numbers, but we don't yet know anything about the potential form it must take. In fact, knowing the time-to-goal solution for minimum-time problem with

the double integrator, we see that this function might need to be non-smooth for even very simple dynamics and objectives.

One natural way to parameterize $\hat{J}^*$ -- a scalar valued-function defined over the state space -- is to define the values on a mesh. This approach then admits algorithms with close ties to the relatively very advanced numerical methods used to solve other partial differential equations (PDEs), such as the ones that appear in finite element modeling or fluid dynamics. One important difference, however, is that our PDE lives in the dimension of the state space, while many of the mesh representations from the other disciplines are optimized for two or three dimensional space. Also, our PDE may have discontinuities (or at least discontinuous gradients) at locations in the state space which are not known apriori.

A slightly more general view of the problem would describe the mesh (and the associated interpolation functions) as just one form of representations for function approximation. Using a neural network to represent the cost-to-go also falls under the domain of function approximation, perhaps representing the other extreme in terms of complexity; using deep networks in approximate dynamic programming is common in deep reinforcement learning, which we will discuss more later in the book.

## Value iteration with function approximation

If we approximate $J^*$ with a finitely-parameterized function $\hat{J}^*_\alpha$, with parameter vector $\alpha$, then this immediately raises many important questions:

- What if the true cost-to-go function does not live in the prescribed function class; e.g., there does not exist an $\alpha$ which satisfies the sufficiency conditions for all $\mathbf{x}$? (This seems very likely to be the case.)
- What update should we apply in the iterative algorithm?
- What can we say about its convergence?

Let us start by considering updates given by a least-squares approximation of the value iteration update.

Using the least squares solution in a value iteration update is sometimes referred to as *fitted value iteration*, where $\mathbf{x}_k$ are some number of samples taken from the continuous space and for discrete-time systems the iterative approximate solution to

$$J^*(\mathbf{x}_0) = \min_{u[\cdot]} \sum_{n=0}^{\infty} \ell(\mathbf{x}[n], \mathbf{u}[n]),$$
$$\text{s.t. } \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]), \mathbf{x}[0] = \mathbf{x}_0$$

becomes

$$J^d_k = \min_{\mathbf{u}} \left[ \ell(\mathbf{x}_k, \mathbf{u}) + \hat{J}^*_\alpha \left( f(\mathbf{x}_k, \mathbf{u}) \right) \right], \tag{9}$$
$$\alpha \Leftarrow \text{argmin}_\alpha \sum_k \left( \hat{J}^*_\alpha(\mathbf{x}_k) - J^d_k \right)^2. \tag{10}$$

Since the desired values $J^d_k$ are only an initial guess of the cost-to-go, we will apply this algorithm iteratively until (hopefully) we achieve some numerical convergence.

Note that the update in (10) is not *quite* the same as doing least-squares optimization of

$$\sum_k \left( \hat{J}^*_\alpha(\mathbf{x}_k) - \min_{\mathbf{u}} \left[ \ell(\mathbf{x}_k, \mathbf{u}) + \hat{J}^*_\alpha \left( f(\mathbf{x}_k, \mathbf{u}) \right) \right] \right)^2,$$

because in this equation $\alpha$ has an effect on both occurences of $\hat{J}^*$. In (10), we cut that dependence by taking $J^d_k$ as fixed desired values; this version performs better in practice. Like many things, this is an old idea that has been given a new name

in the deep reinforcement learning literature -- people think of the $\hat{J}^*_\alpha$ on the right hand side as being the output from a fixed "target network". For nonlinear function approximators, the update to $\alpha$ in (10) is often replaced with just one step of gradient descent.

### Example 7.7 (Neural Value Iteration)

Let us try reproducing our double-integrator value iteration examples using neural networks in PyTorch:


In general, the convergence and accuracy guarantees for value iteration with generic function approximators are quite weak. But we do have some results for the special case of *linear function approximators*. A linear function approximator takes the form:

$$\hat{J}^*_\alpha(\mathbf{x}) = \sum_i \alpha_i \psi_i(\mathbf{x}) = \psi^T(\mathbf{x})\alpha,$$

where $\psi(\mathbf{x})$ is a vector of potentially nonlinear features. Common examples of features include polynomials, radial basis functions, or most interpolation schemes used on a mesh. The distinguishing feature of a linear function approximator is the ability to exactly solve for $\alpha$ in order to represent a desired function optimally, in a least-squares sense. For linear function approximators, this is simply:

$$\alpha \Leftarrow \begin{bmatrix} \psi^T(\mathbf{x}_1) \\ \vdots \\ \psi^T(\mathbf{x}_K) \end{bmatrix}^+ \begin{bmatrix} J^d_1 \\ \vdots \\ J^d_K \end{bmatrix},$$

where the $^+$ notation refers to a Moore-Penrose pseudoinverse. Remarkably, for linear function approximators, this update is still known to converge to the globally optimal $\alpha^*$.

## Value iteration on a mesh

Imagine that we use a mesh to approximate the cost-to-go function over that state space with $K$ mesh points $\mathbf{x}_k$. We would like to perform the value iteration update:

$$\forall k, \hat{J}^*(\mathbf{x}_k) \Leftarrow \min_{\mathbf{u}} \left[ \ell(\mathbf{x}_k, \mathbf{u}) + \hat{J}^*\left(f(\mathbf{x}_k, \mathbf{u})\right) \right], \tag{11}$$

but must deal with the fact that $f(\mathbf{x}_k, \mathbf{u})$ might not result in a next state that is directly at a mesh point. Most interpolation schemes for a mesh can be written as some weighted combination of the values at nearby mesh points, e.g.

$$\hat{J}^*(\mathbf{x}) = \sum_i \beta_i(\mathbf{x}) \hat{J}^*(\mathbf{x}_i), \quad \sum_i \beta_i = 1$$

with $\beta_i$ the relative weight of the $i$th mesh point. In DRAKE we have implemented barycentric interpolation[7]. Taking $\alpha_i = \hat{J}^*(\mathbf{x}_i)$, the cost-to-go estimate at mesh point $i$, we can see that this is precisely an important special case of fitted value iteration with linear function approximation. Furthermore, assuming $\beta_i(\mathbf{x}_i) = 1$, (e.g., the only point contributing to the interpolation *at a mesh point* is the value at the mesh point itself), the update in Eq. (11) is precisely the least-squares update (and it achieves zero error). This is the representation used in the value iteration examples that you've already experimented with above.

## Continuous-time systems

For solutions to systems with continuous-time dynamics, I have to uncover one of the details that I've so far been hiding to keep the notation simpler. Let us consider a problem with a finite-horizon:

$$\min_{\mathbf{u}[\cdot]} \sum_{n=0}^{N} \ell(\mathbf{x}[n], \mathbf{u}[n]),$$
$$\text{s.t. } \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]), \mathbf{x}[0] = \mathbf{x}_0$$

In fact, the way that we compute this is by solving the *time-varying cost-to-go function* backwards in time:

$$J^*(\mathbf{x}, N) = \min_{\mathbf{u}} \ell(\mathbf{x}, \mathbf{u})$$
$$J^*(\mathbf{x}, n-1) = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + J^*(f(\mathbf{x}, \mathbf{u}), n)\right].$$

The convergence of the value iteration update is equivalent to solving this time-varying cost-to-go backwards in time until it reaches a steady-state solution (the infinite-horizon solution). Which explains why value iteration only converges if the optimal cost-to-go is bounded.

Now let's consider the continuous-time version. Again, we have a time-varying cost-to-go, $J^*(\mathbf{x}, t)$. Now

$$\frac{dJ^*}{dt} = \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t},$$

and our sufficiency condition is

$$0 = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t}\right].$$

But since $\frac{\partial J^*}{\partial t}$ doesn't depend on $\mathbf{u}$, we can pull it out of the min and write the (true) HJB:

$$-\frac{\partial J^*}{\partial t} = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u})\right].$$

The standard numerical recipe [8] for solving this is to approximate $\hat{J}^*(\mathbf{x}, t)$ on a mesh and then integrate the equations backwards in time (until convergence, if the goal is to find the infinite-horizon solution). If, for mesh point $\mathbf{x}_i$ we have $\alpha_i(t) = \hat{J}^*(\mathbf{x}_i, t)$, then:

$$-\dot{\alpha}_i(t) = \min_{\mathbf{u}} \left[\ell(\mathbf{x}_i, \mathbf{u}) + \frac{\partial J^*(\mathbf{x}_i, t)}{\partial \mathbf{x}} f(\mathbf{x}_i, \mathbf{u})\right],$$

where the partial derivative is estimated with a suitable finite-difference approximation on the mesh and often some "viscosity" terms are added to the right-hand side to provide additional numerical robustness; see the Lax-Friedrichs scheme [8] (section 5.3.1) for an example.

Probably most visible and consistent campaign using numerical HJB solutions in applied control (at least in robotics) has come from Claire Tomlin's group at Berkeley. Their work leverages Ian Mitchell's Level Set Toolbox, which solves the Hamilton-Jacobi PDEs on a Cartesian mesh using the technique cartooned above, and even includes the minimum-time problem for the double integrator as a tutorial example[9].

## 7.4 Extensions

There are many many nice extensions to the basic formulations that we've presented so far. I'll try to list a few of the most important ones here. I've also had a number of students in this course explore very interesting extensions; for example [10] looked a

imposing a low-rank structure on the (discrete-state) value function using ideas from matrix completion to obtain good estimates despite updating only a small fraction of the states.

### 7.4.1 Stochastic control for finite MDPs

One of the most amazing features of the dynamic programming, additive cost approach to optimal control is the relative ease with which it extends to optimizing stochastic systems.

For discrete systems, we can generalize our dynamics on a graph by adding action-dependent transition probabilities to the edges. This new dynamical system is known as a Markov Decision Process (MDP), and we write the dynamics completely in terms of the transition probabilities

$$\Pr(s[n+1] = s' | s[n] = s, a[n] = a).$$

For discrete systems, this is simply a big lookup table. The cost that we incur for any execution of system is now a random variable, and so we formulate the goal of control as optimizing the expected cost, e.g.

$$J^*(s[0]) = \min_{a[\cdot]} E\left[\sum_{n=0}^{\infty} \ell(s[n], a[n])\right]. \tag{12}$$

Note that there are many other potential objectives, such as minimizing the worst-case error, but the expected cost is special because it preserves the dynamic programming recursion:

$$J^*(s) = \min_a E\left[\ell(s,a) + J^*(s')\right] = \min_a \left[\ell(s,a) + \sum_{s'} \Pr(s'|s,a) J^*(s')\right].$$

Remarkably, if we use these optimality conditions to construct our value iteration algorithm

$$\hat{J}(s) \Leftarrow \min_a \left[\ell(s,a) + \sum_{s'} \Pr(s'|s,a) \hat{J}(s')\right],$$

then this algorithm has the same strong convergence guarantees of its counterpart for deterministic systems. And it is essentially no more expensive to compute!

#### Stochastic interpretation of deterministic, continuous-state value iteration

There is a particularly nice observation to be made here. Let's assume that we have discrete control inputs and discrete-time dynamics, but a continuous state space. Recall the fitted value iteration on a mesh algorithm described above. In fact, the resulting update is exactly the same as if we treated the system as a discrete state MDP with $\beta_i$ representing the probability of transitioning to state $x_i$! This sheds some light on the impact of discretization on the solutions -- discretization error here will cause a sort of diffusion corresponding to the probability of spreading across neighboring nodes.

This is a preview of a much more general toolkit that we develop later for stochastic and robust control.

### 7.4.2 Linear Programming Approach

For discrete MDPs, value iteration is a magical algorithm because it is simple, but known to converge to the global optimal, $J^*$. However, other important algorithms are known; one of the most important is a solution approach using linear

programming. This formulation provides an alternative view, but may also be more generalizable and even more efficient for some instances of the problem.

Recall the optimality conditions from Eq. (1). If we describe the cost-to-go function as a vector, $J_i = J(s_i)$, then these optimality conditions can be rewritten in vector form as

$$\mathbf{J} = \min_a [\ell(a) + \mathbf{T}(a)\mathbf{J}], \tag{13}$$

where $\ell_i(a) = \ell(s_i, a)$ is the cost vector, and $T_{i,j}(a) = \Pr(s_j | s_i, a)$ is the transition probability matrix. Let us take $\mathbf{J}$ as the vector of decision variables, and replace Eq. (13) with the constraints:

$$\forall a, \mathbf{J} \leq \ell(a) + \mathbf{T}(a)\mathbf{J}. \tag{14}$$

Clearly, for finite $a$, this is finite list of linear constraints, and for any vector $\mathbf{J}$ satisfying these constraints we have $\mathbf{J} \leq \mathbf{J}^*$ (elementwise). Now write the linear program:

$$\begin{aligned} &\text{maximize}_{\mathbf{J}} \quad \mathbf{c}^T\mathbf{J}, \\ &\text{subject to} \quad \forall a, \mathbf{J} \leq \ell(a) + \mathbf{T}(a)\mathbf{J}, \end{aligned}$$

where $c$ is any positive vector. The solution to this problem is $\mathbf{J} = \mathbf{J}^*$.

Perhaps even more interesting is that this approach can be generalized to linear function approximators. Taking a vector form of my notation above, now we have a matrix of features with $\mathbf{\Psi}_{i,j} = \psi_j^T(\mathbf{x}_i)$ and we can write the LP

$$\text{maximize}_{\alpha} \quad \mathbf{c}^T\mathbf{\Psi}\alpha, \tag{15}$$

$$\text{subject to} \quad \forall a, \mathbf{\Psi}\alpha \leq \ell(a) + \mathbf{T}(a)\mathbf{\Psi}\alpha. \tag{16}$$

This time the solution is not necessarily optimal, because $\mathbf{\Psi}\alpha^*$ only approximates $\mathbf{J}^*$, and the relative values of the elements of $\mathbf{c}$ (called the "state-relevance weights") can determine the relative tightness of the approximation for different features [11].

## 7.5 Exercises

### Exercise 7.1 (Discounting and the Convergence of Value Iteration)

Let's consider the discrete time, state, and action version of value iteration:

$$\hat{J}^*(s_i) \Leftarrow \min_{a \in A} \left[\ell(s_i, a) + \hat{J}^*\left(f(s_i, a)\right)\right], \tag{17}$$

which finds the optimal policy for the infinite horizon cost function $\sum_{n=0}^{\infty} \ell(s[n], a[n])$. If $J^*$ is the true optimal cost-to-go, show that any solution $\hat{J}^* = J^* + c$, where $c$ is any constant scalar, is a fixed point of this value iteration update. Is the controller still optimal, even if the estimated cost-to-go function is off by a constant factor?

Surprisingly, adding a discount factor can help with this. Consider the infinite-horizon discounted cost: $\sum_{n=0}^{\infty} \gamma^n \ell(s[n], a[n])$, where $0 < \gamma \leq 1$ is the discount factor. The corresponding value iteration update is

$$\hat{J}^*(s_i) \Leftarrow \min_{a \in A} \left[\ell(s_i, a) + \gamma\hat{J}^*\left(f(s_i, a)\right)\right]. \tag{18}$$

For simplicity, assume that there exists a state $s_i$ that is a zero-cost fixed point under the optimal policy; e.g. $\ell(s_i, \pi^*(s_i)) = 0$ and $f(s_i, \pi^*(s_i)) = s_i$. Is $\hat{J}^* = J^* + c$

still a fixed point of the value iteration update when $\gamma < 1$? Show your work.

## Exercise 7.2 (Choosing a Cost Function)

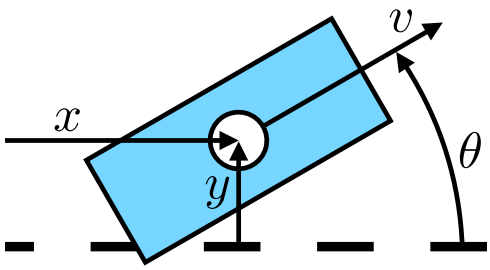



Figure 7.8 - Autonomous car moving at velocity $v$ on a straight road.

The figure above shows an autonomous car moving at constant velocity $v > 0$ on a straight road. Let $x$ be the (longitudinal) position of the car along the road, $y$ its (transversal) distance from the centerline, and $\theta$ the angle between the centerline and the direction of motion. The only control action is the steering velocity $u$, which is constrained in the interval $[u_{\min}, u_{\max}]$ (where $u_{\min} < 0$ and $u_{\max} > 0$). We describe the car dynamics with the simple kinematic model

$$\begin{aligned} \dot{x} &= v \cos\theta, \\ \dot{y} &= v \sin\theta, \\ \dot{\theta} &= u. \end{aligned}$$

Let $\mathbf{x} = [x, y, \theta]^T$ be the state vector. To optimize the car trajectory we consider a quadratic objective function

$$J = \int_0^\infty [\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + Ru^2(t)]dt,$$

where $\mathbf{Q}$ is a constant positive-semidefinite (hence symmetric) matrix and $R$ is a constant nonnegative scalar (note that $R = 0$ is allowed here).

a. Suppose our only goal is to keep the distance $y$ between the car and the centerline as small as possible, as fast as possible, without worrying about anything else. What would be your choice for $\mathbf{Q}$ and $R$?
b. How would the behavior of the car change if you were to multiply the weights $\mathbf{Q}$ and $R$ from the previous point by an arbitrary positive coefficient $\alpha$?
c. The cost function from point (a) might easily lead to excessively sharp turns. Which entry of $\mathbf{Q}$ or $R$ would you increase to mitigate this issue?
d. Country roads are more slippery on the sides than in the center. Is this class of objective functions (quadratic objectives) rich enough to include a penalty on sharp turns that increases with the distance of the car from the centerline?
e. With this dynamics model and this objective function, would you ever choose a weight matrix $\mathbf{Q}$ which is strictly positive definite (independent of the task you want the car to accomplish)? Why?

## Exercise 7.3 (Ill-Posed Optimal Control Problem)

In this exercise we will see how seemingly simple cost functions can give surprising results. Consider the single-integrator system $\dot{x} = u$ with initial state $x(0) = 0$. We would like to find the control signal $u(t)$ that minimizes the seemingly

innocuous cost function

$$J = \int_0^T x^2(t) + (u^2(t) - 1)^2 dt,$$

with $T$ finite. To this end, we consider a square-wave control parameterized by $\tau > 0$:

$$u_\tau(t) = \begin{cases} 1 & \text{if} \quad t \in [0, \tau) \cup [3\tau, 5\tau) \cup [7\tau, 9\tau) \cup \cdots \\ -1 & \text{if} \quad t \in [\tau, 3\tau) \cup [5\tau, 7\tau) \cup [9\tau, 11\tau) \cup \cdots \end{cases}.$$

a. What are the states $x$ and the inputs $u$ for which the running cost

$$\ell(x, u) = x^2 + (u^2 - 1)^2$$

is equal to zero?

b. Consider now two control signals $u_{\tau_1}(t)$ and $u_{\tau_2}(t)$, with $\tau_1 = \tau_2/2$. Which one of the two incurs the lower cost $J$? (Hint: start by sketching how the system state $x(t)$ evolves under these two control signals.)

c. What happens to the cost $J$ when $\tau$ gets smaller and smaller? What does the optimal control signal look like? Could you implement it with a finite-bandwidth controller?

## Exercise 7.4 (A Linear System with Quadratic Cost)

Consider the scalar control differential equation

$$\dot{x} = x + u,$$

and the infinite horizon cost function

$$J = \int_0^\infty [3x^2(t) + u^2(t)] dt.$$

As we will see in the chapter on linear-quadratic regulation, the optimal cost-to-go for a problem of this kind assumes the form $J^* = Sx^2$. It is not hard to see that this, in turn, implies that the optimal controller has the form $u^* = -Kx$.

a. Imagine that you plugged the expression $J^* = Sx^2$ in the HJB equations for this problem, you solved them for $S$, and you got $S \leq 0$. Would this result ring any alarm bells? Explain your answer.

b. Use the HJB sufficiency theorem to derive the optimal values of $S$ and $K$.

c. Working with digital controllers, we typically have to sample the dynamics of our systems, and approximate them with discrete-time models. Let us introduce a time step $h > 0$ and discretize the dynamics as

$$\frac{x[n+1] - x[n]}{h} = x[n] + u[n],$$

and the objective as

$$h \sum_{n=0}^{\infty} (3x^2[n] + u^2[n]).$$

One of the following expressions is the correct cost-to-go $J_h^*(x)$ for this discretized problem. Can you identify it without solving the discrete-time HJB equation? Explain your answer.

i. $J_h^*(x) = S_h x^4$ with $S_h = 3 + h + \sqrt{6}h^2$.
ii. $J_h^*(x) = S_h x^2$ with $S_h = 1 + 2h + 2\sqrt{h^2 + h + 1}$.
iii. $J_h^*(x) = S_h x^2$ with $S_h = 3 + 2h^2 + \sqrt{h^2 + h + 2}$.

## Exercise 7.5 (Value Iteration for Minimum-Time Problems)

In this exercise we analyze the performance of the value-iteration algorithm, considering its application to the minimum time problem for the double integrator. In this python notebook, you will find everything you need for this analysis. Take the time to go through the notebook and understand the code in it, then answer the following questions.

a. At the end of the notebook section "Performance of the Value-Iteration Policy", we plot the state trajectory of the double integrator in closed loop with the value-iteration policy, as well as the resulting control signal.
    i. Does the state-space trajectory follow the theoretically-optimal quadratic arcs we have seen in the example?
    ii. Is the control policy we get from value iteration a bang-bang policy? In other words, does the control signal take values in the set $\{-1, 0, 1\}$ exclusively?
    iii. Explain in a couple of sentences, what is the reason for this behavior.
b. In the "Value Iteration Algorithm" section of the notebook, increase the number of knot points on the $q$ and the $\dot{q}$ axes to refine the state-space mesh used in the value iteration. Does this fix the issues you have seen in point (a)?
c. In the final section of the notebook, implement the theoretically-optimal control policy from the example, and use the plots to verify that the closed-loop system behaves as expected.

## Exercise 7.6 (Fitted Value Iteration for the Double Integrator)

In this exercise we will implement fitted value iteration with a neural network, and use it to generate a controler for the double integrator. In this python notebook, you will need to implement a few things. Take the time to go through the notebook and understand the code in it, and then answer the following questions. The written questions will also be listed in the notebook for your convenience.

a. Work through the coding sections in the notebook.
b. Although it gets close, in the notebook why might our implementation not succeed in stabilizing the system exactly at the origin?
c. In the notebook we implemented "fitted value iteration". In the questions below, you'll be asked to think about the differences between the "graph search" version of value iteration we saw at the start of chapter 7, and fitted value iteration. Answer the questions below with a brief explanation:
    i. Practically, can graph search value iteration perform directly in a continuous state space? Can fitted value iteration?
    ii. Practically, can graph search value iteration perform directly with a continuous action space? Can fitted value iteration?
d. For our implementation, we assume deterministic transitions between states. This means that given a particular state $x_t$ and action $u_t$, we will

always end up in the same unique $x_{t+1}$. Mathematically:

$$x_{t+1} = f(x_t, u_t)$$

In real world scenarios we often have to deal with disturbances or dynamics not captured by our model, which can cause our state transitions to be stochastic. Mathematically:

$$P(x_{t+1}|x_t, u_t) = f(x_t, u_t)$$

If we had stochastic transitions, which steps of the notebook implementation would need to change, and how would they change? The possible steps to choose from are listed below - you can select multiple. When describing the change, a verbal description is fine you do not need to include any code:

i. Sampling the set of points `X` before performing the fit. Note that this does not include `Xnext`.
ii. Generating the costs `G` before performing the fit. Assume we are using the quadratic cost.
iii. Updating the cost function target in the training loop: `Jd, ind = torch.min(G + discount*Jnext, dim=1)`.
iv. Choosing the best action in the policy when simulating the system.

## References


1. Dimitri P. Bertsekas, "Dynamic Programming and Optimal Control",Athena Scientific , 2000.

2. Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction",MIT Press , 2018.

3. Dimitri P. Bertsekas and John N. Tsitsiklis, "Neuro-Dynamic Programming",Athena Scientific , October, 1996.

4. Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction",MIT Press , 1998.

5. Dimitri P. Bertsekas, "Dynamic Programming \& Optimal Control",Athena Scientific , vol. I and II, May 1, 2005.

6. David H. Jacobson and David Q. Mayne, "Differential Dynamic Programming",American Elsevier Publishing Company, Inc. , 1970.

7. Remi Munos and Andrew Moore, "Barycentric Interpolators for Continuous Space and Time Reinforcement Learning", *Advances in Neural Information Processing Systems*, vol. 11, pp. 1024--1030, 1998.

8. Stanley Osher and Ronald Fedkiw, "Level Set Methods and Dynamic Implicit Surfaces",Springer , 2003.

9. Ian M. Mitchell and Jeremy A. Templeton, "A Toolbox of {H}amilton-{J}acobi Solvers for Analysis of Nondeterministic Continuous and Hybrid Systems", *Proceedings of Hybrid Systems Computation and Control*, March, 2005.

10. Yuzhe Yang and Guo Zhang and Zhi Xu and Dina Katabi, "Harnessing Structures for Value-Based Planning and Reinforcement Learning", *International Conference on Learning Representations*, 2020.

11. Daniela Pucci de Farias, "The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application", PhD thesis, Stanford University, June, 2002.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 8
# Linear Quadratic Regulators

Open in Colab

While solving the dynamic programming problem for continuous systems is very hard in general, there are a few very important special cases where the solutions are very accessible. Most of these involve variants on the case of linear dynamics and quadratic cost. The simplest case, called the linear quadratic regulator (LQR), is formulated as stabilizing a time-invariant linear system to the origin.

The linear quadratic regulator is likely the most important and influential result in optimal control theory to date. In this chapter we will derive the basic algorithm and a variety of useful extensions.

## 8.1 Basic Derivation

Consider a linear time-invariant system in state-space form,

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

with the infinite-horizon cost function given by

$$J = \int_0^\infty \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right] dt, \quad \mathbf{Q} = \mathbf{Q}^T \succeq \mathbf{0}, \mathbf{R} = \mathbf{R}^T \succ 0.$$

Our goal is to find the optimal cost-to-go function $J^*(\mathbf{x})$ which satisfies the HJB:

$$\forall \mathbf{x}, \quad 0 = \min_{\mathbf{u}} \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \frac{\partial J^*}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})\right].$$

There is one important step here -- it is well known that for this problem the optimal cost-to-go function is quadratic. This is easy to verify. Let us choose the form:

$$J^*(\mathbf{x}) = \mathbf{x}^T\mathbf{S}\mathbf{x}, \quad \mathbf{S} = \mathbf{S}^T \succeq 0.$$

The gradient of this function is

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}.$$

Since we have guaranteed, by construction, that the terms inside the $\min$ are quadratic and convex (because $\mathbf{R} \succ 0$), we can take the minimum explicitly by finding the solution where the gradient of those terms vanishes:

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R} + 2\mathbf{x}^T\mathbf{S}\mathbf{B} = 0.$$

This yields the optimal policy

$$\mathbf{u}^* = \pi^*(\mathbf{x}) = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}\mathbf{x} = -\mathbf{K}\mathbf{x}.$$

Inserting this back into the HJB and simplifying yields

$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + 2\mathbf{S}\mathbf{A}\right]\mathbf{x}.$$

All of the terms here are symmetric except for the $2\mathbf{S}\mathbf{A}$, but since $\mathbf{x}^T\mathbf{S}\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{A}^T\mathbf{S}\mathbf{x}$, we can write

$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S}\right]\mathbf{x}.$$

and since this condition must hold for all $\mathbf{x}$, it is sufficient to consider the matrix equation

$$0 = \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{Q}.$$

This extremely important equation is a version of the *algebraic Riccati equation*. Note that it is quadratic in $\mathbf{S}$, making its solution non-trivial, but it is well known that the equation has a single positive-definite solution if and only if the system is controllable and there are good numerical methods for finding that solution, even in high-dimensional problems. Both the optimal policy and optimal cost-to-go function are available from DRAKE by calling `(K,S) = LinearQuadraticRegulator(A,B,Q,R)`.

If the appearance of the quadratic form of the cost-to-go seemed mysterious, consider that the solution to the linear system $\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}$ takes the form $\mathbf{x}(t) = e^{(\mathbf{A}-\mathbf{B}\mathbf{K})t}\mathbf{x}(0)$, and try inserting this back into the integral cost function. You'll see that the cost takes the form $J = \mathbf{x}^T(0)\mathbf{S}\mathbf{x}(0)$.

It is worth examining the form of the optimal policy more closely. Since the value function represents cost-to-go, it would be sensible to move down this landscape as quickly as possible. Indeed, $-\mathbf{S}\mathbf{x}$ is in the direction of steepest descent of the value function. However, not all directions are possible to achieve in state-space. $-\mathbf{B}^T\mathbf{S}\mathbf{x}$ represents precisely the projection of the steepest descent onto the control space, and is the steepest descent achievable with the control inputs $\mathbf{u}$. Finally, the pre-scaling by the matrix $\mathbf{R}^{-1}$ biases the direction of descent to account for relative weightings that we have placed on the different control inputs. Note that although this interpretation is straight-forward, the slope that we are descending (in the value function, $\mathbf{S}$) is a complicated function of the dynamics and cost.

## Example 8.1 (LQR for the Double Integrator)

Now can use LQR to reproduce our HJB example from the previous chapter:

examples/double_integrator/lqr.py

As in the hand-derived example, our numerical solution returns

$$\mathbf{K} = [1, \sqrt{3}], \qquad \mathbf{S} = \begin{bmatrix} \sqrt{3} & 1 \\ 1 & \sqrt{3} \end{bmatrix}.$$

### 8.1.1 Local stabilization of nonlinear systems

LQR is extremely relevant to us even though our primary interest is in nonlinear dynamics, because it can provide a local approximation of the optimal control solution for the nonlinear system. Given the nonlinear system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, and a stabilizable operating point, $(\mathbf{x}_0, \mathbf{u}_0)$, with $f(\mathbf{x}_0, \mathbf{u}_0) = 0$. We can define a relative coordinate system

$$\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}_0, \quad \bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}_0,$$

and observe that

$$\dot{\bar{\mathbf{x}}} = \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),$$

which we can approximate with a first-order Taylor expansion to

$$\dot{\bar{\mathbf{x}}} \approx f(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) = \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}}.$$

Similarly, we can define a quadratic cost function in the error coordinates, or take a (positive-definite) second-order approximation of a nonlinear cost function about the operating point (linear and constant terms in the cost function can be easily incorporated into the derivation by parameterizing a full quadratic form for $J^*$, as seen in the Linear Quadratic Tracking derivation below).

The resulting controller takes the form $\bar{\mathbf{u}}^* = -\mathbf{K}\bar{\mathbf{x}}$ or

$$\mathbf{u}^* = \mathbf{u}_0 - \mathbf{K}(\mathbf{x} - \mathbf{x}_0).$$

For convenience, DRAKE allows you to call `controller = LinearQuadraticRegulator(system, context, Q, R)` on most dynamical systems (including block diagrams built up of many subsystems); it will perform the linearization for you.

**Example 8.2 (LQR for Acrobots, Cart-Poles, and Quadrotors)**

LQR provides a very satisfying solution to the canonical "balancing" problem that we've already described for a number of model systems. Here is the notebook with those examples, again:


I find it very compelling that the same derivation (and effectively identical code) can stabilize such a diversity of systems!

## 8.2 FINITE-HORIZON FORMULATIONS

Recall that the cost-to-go for finite-horizon problems is time-dependent, and therefore the HJB sufficiency condition requires an additional term for $\frac{\partial J^*}{\partial t}$.

$$\forall \mathbf{x}, \forall t \in [t_0, t_f], \quad 0 = \min_{\mathbf{u}} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t} \right].$$

### 8.2.1 Finite-horizon LQR

Consider systems governed by an LTI state-space equation of the form

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

and a finite-horizon cost function, $J = h(\mathbf{x}(t_f)) + \int_0^{t_f} \ell(\mathbf{x}(t), \mathbf{u}(t))dt$, with

$$h(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}_f\mathbf{x}, \quad \mathbf{Q}_f = \mathbf{Q}_f^T \succeq \mathbf{0}$$
$$\ell(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}, \quad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0$$

Writing the HJB, we have

$$0 = \min_{\mathbf{u}} \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \frac{\partial J^*}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \frac{\partial J^*}{\partial t}\right].$$

Due to the positive definite quadratic form on $\mathbf{u}$, we can find the minimum by setting the gradient to zero:

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R} + \frac{\partial J^*}{\partial \mathbf{x}}\mathbf{B} = 0$$
$$\mathbf{u}^* = \pi^*(\mathbf{x}, t) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T\frac{\partial J^*}{\partial \mathbf{x}}^T$$

In order to proceed, we need to investigate a particular form for the cost-to-go function, $J^*(\mathbf{x}, t)$. Let's try a solution of the form:

$$J^*(\mathbf{x}, t) = \mathbf{x}^T\mathbf{S}(t)\mathbf{x}, \quad \mathbf{S}(t) = \mathbf{S}^T(t) \succ \mathbf{0}.$$

In this case we have

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}(t), \quad \frac{\partial J*}{\partial t} = \mathbf{x}^T\dot{\mathbf{S}}(t)\mathbf{x},$$

and therefore

$$\mathbf{u}^* = \pi^*(\mathbf{x}, t) = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t)\mathbf{x}$$
$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}(t) + \dot{\mathbf{S}}(t)\right]\mathbf{x}.$$

Therefore, $\mathbf{S}(t)$ must satisfy the condition (known as the continuous-time *differential Riccati equation*):

$$-\dot{\mathbf{S}}(t) = \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}(t) - \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \mathbf{Q},$$

and the terminal condition

$$\mathbf{S}(t_f) = \mathbf{Q}_f.$$

Since we were able to satisfy the HJB with the minimizing policy, we have met the sufficiency condition, and have found the optimal policy and optimal cost-to-go function.

Note that the infinite-horizon LQR solution described in the prequel is exactly the steady-state solution of this equation, defined by $\dot{\mathbf{S}}(t) = 0$. Indeed, for controllable systems this equation is stable (backwards in time), and as expected the finite-horizon solution converges on the infinite-horizon solution as the horizon time limits to infinity.

## 8.2.2 Time-varying LQR

The derivation above holds even if the dynamics are given by

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(\mathbf{t})\mathbf{u}.$$

Similarly, the cost functions $\mathbf{Q}$ and $\mathbf{R}$ can also be time-varying. This is quite surprising, as the class of time-varying linear systems is a quite general class of systems. It requires essentially no assumptions on how the time-dependence enters,

except perhaps that if $\mathbf{A}$ or $\mathbf{B}$ is discontinuous in time then one would have to use the proper techniques to accurately integrate the differential equation.

### 8.2.3 Local trajectory stabilization for nonlinear systems

One of the most powerful applications of time-varying LQR involves linearizing around a nominal trajectory of a nonlinear system and using LQR to provide a trajectory controller. This will tie in very nicely with the algorithms we develop in the chapter on trajectory optimization.

Let us assume that we have a nominal trajectory, $\mathbf{x}_0(t), \mathbf{u}_0(t)$ defined for $t \in [t_1, t_2]$. Similar to the time-invariant analysis, we begin by defining a local coordinate system relative to the trajectory:

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_0(t), \quad \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_0(t).$$

Now we have

$$\dot{\bar{\mathbf{x}}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}_0 = f(\mathbf{x}, \mathbf{u}) - f(\mathbf{x}_0, \mathbf{u}_0),$$

which we can again approximate with a first-order Taylor expansion to

$$\dot{\bar{\mathbf{x}}} \approx f(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) - f(\mathbf{x}_0, \mathbf{u}_0) = \mathbf{A}(t)\bar{\mathbf{x}} + \mathbf{B}(t)\bar{\mathbf{u}}.$$

This very similar to using LQR to stablize a fixed-point, but with some important differences. First, the linearization is time-varying. Second, our linearization is valid for any state along a feasible trajectory (not just fixed-points), because the coordinate system is moving along with the trajectory.

Similarly, we can define a quadratic cost function in the error coordinates, or take a (positive-definite) second-order approximation of a nonlinear cost function along the trajectory (linear and constant terms in the cost function can be easily incorporated into the derivation by parameterizing a full quadratic form for $J^*$, as seen in the Linear Quadratic Tracking derivation below).

The resulting controller takes the form $\bar{\mathbf{u}}^* = -\mathbf{K}(t)\bar{\mathbf{x}}$ or

$$\mathbf{u}^* = \mathbf{u}_0(t) - \mathbf{K}(t)(\mathbf{x} - \mathbf{x}_0(t)).$$

DRAKE provides a `FiniteHorizonLinearQuadraticRegulator` method; if you pass it a nonlinear systems it will perform the linearization in the proper coordinates for you using automatic differentiation.

Remember that stability is a statement about what happens as time goes to infinity. In order to talk about stabilizing a trajectory, the trajectory must be defined for all $t \in [t_1, \infty)$. This can be accomplished by considering a finite-time trajectory which terminates at a stabilizable fixed-point at a time $t_2 \geq t_1$, and remains at the fixed point for $t \geq t_2$. In this case, the finite-horizon Riccati equation is initialized with the infinite-horizon LQR solution: $S(t_2) = S_\infty$, and solved backwards in time from $t_2$ to $t_1$ for the remainder of the trajectory. And *now* we can say that we have *stabilized* the trajectory!

### 8.2.4 Linear Quadratic Optimal Tracking

For completeness, we consider a slightly more general form of the linear quadratic regulator. The standard LQR derivation attempts to drive the system to zero. Consider now the problem:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$h(\mathbf{x}) = (\mathbf{x} - \mathbf{x}_d(t_f))^T \mathbf{Q}_f (\mathbf{x} - \mathbf{x}_d(t_f)), \quad \mathbf{Q}_f = \mathbf{Q}_f^T \succeq 0$$
$$\ell(\mathbf{x}, \mathbf{u}, t) = (\mathbf{x} - \mathbf{x}_d(t))^T \mathbf{Q} (\mathbf{x} - \mathbf{x}_d(t)) + (\mathbf{u} - \mathbf{u}_d(t))^T \mathbf{R} (\mathbf{u} - \mathbf{u}_d(t)),$$
$$\mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0$$

Now, guess a solution

$$J^*(\mathbf{x}, t) = \mathbf{x}^T \mathbf{S}_{xx}(t) \mathbf{x} + 2\mathbf{x}^T \mathbf{s}_x(t) + s_0(t), \quad \mathbf{S}_{xx}(t) = \mathbf{S}_{xx}^T(t) \succ \mathbf{0}.$$

In this case, we have

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T \mathbf{S}_{xx}(t) + 2\mathbf{s}_x^T(t), \quad \frac{\partial J^*}{\partial t} = \mathbf{x}^T \dot{\mathbf{S}}_{xx}(t) \mathbf{x} + 2\mathbf{x}^T \dot{\mathbf{s}}_x(t) + \dot{s}_0(t).$$

Using the HJB,

$$0 = \min_{\mathbf{u}} \left[ (\mathbf{x} - \mathbf{x}_d(t))^T \mathbf{Q} (\mathbf{x} - \mathbf{x}_d(t)) + (\mathbf{u} - \mathbf{u}_d(t))^T \mathbf{R} (\mathbf{u} - \mathbf{u}_d(t)) + \frac{\partial J^*}{\partial \mathbf{x}} (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \frac{\partial J^*}{\partial t} \right],$$

we have

$$\frac{\partial}{\partial \mathbf{u}} = 2(\mathbf{u} - \mathbf{u}_d(t))^T \mathbf{R} + (2\mathbf{x}^T \mathbf{S}_{xx}(t) + 2\mathbf{s}_x^T(t)) \mathbf{B} = 0,$$
$$\mathbf{u}^*(t) = \mathbf{u}_d(t) - \mathbf{R}^{-1} \mathbf{B}^T \left[ \mathbf{S}_{xx}(t) \mathbf{x} + \mathbf{s}_x(t) \right]$$

The HJB can be satisfied by integrating backwards

$$\begin{aligned}
-\dot{\mathbf{S}}_{xx}(t) =& \mathbf{Q} - \mathbf{S}_{xx}(t) \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{S}_{xx}(t) + \mathbf{S}_{xx}(t) \mathbf{A} + \mathbf{A}^T \mathbf{S}_{xx}(t) \\
-\dot{\mathbf{s}}_x(t) =& -\mathbf{Q}\mathbf{x}_d(t) + \left[ \mathbf{A}^T - \mathbf{S}_{xx} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \right] \mathbf{s}_x(t) + \mathbf{S}_{xx}(t) \mathbf{B} \mathbf{u}_d(t) \\
-\dot{s}_0(t) =& \mathbf{x}_d(t)^T \mathbf{Q} \mathbf{x}_d(t) - \mathbf{s}_x^T(t) \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{s}_x(t) + 2\mathbf{s}_x(t)^T \mathbf{B} \mathbf{u}_d(t),
\end{aligned}$$

from the final conditions

$$\begin{aligned}
\mathbf{S}_{xx}(t_f) =& \mathbf{Q}_f \\
\mathbf{s}_x(t_f) =& -\mathbf{Q}_f \mathbf{x}_d(t_f) \\
s_0(t_f) =& \mathbf{x}_d^T(t_f) \mathbf{Q}_f \mathbf{x}_d(t_f).
\end{aligned}$$

Notice that the solution for $\mathbf{S}_{xx}$ is the same as the simpler LQR derivation, and is symmetric (as we assumed). Note also that $s_0(t)$ has no effect on control (even indirectly), and so can often be ignored.

A quick observation about the quadratic form, which might be helpful in debugging. We know that $J(\mathbf{x}, t)$ must be uniformly positive. This is true iff $\mathbf{S}_{xx} \succ 0$ and $s_0 > \mathbf{s}_x^T \mathbf{S}_{xx}^{-1} \mathbf{s}_x$, which comes from evaluating the function at $\mathbf{x}_{min}(t)$ defined by $\left[ \frac{\partial J^*(\mathbf{x}, t)}{\partial \mathbf{x}} \right]_{\mathbf{x} = \mathbf{x}_{min}(t)} = 0$.

## 8.2.5 Linear Final Boundary Value Problems

The finite-horizon LQR formulation can be used to impose a strict final boundary value condition by setting an infinite $\mathbf{Q}_f$. However, integrating the Riccati equation backwards from an infinite initial condition isn't very practical. To get around this, let us consider solving for $\mathbf{P}(t) = \mathbf{S}(t)^{-1}$. Using the matrix relation $\frac{d\mathbf{S}^{-1}}{dt} = -\mathbf{S}^{-1} \frac{d\mathbf{S}}{dt} \mathbf{S}^{-1}$, we have:

$$-\dot{\mathbf{P}}(t) = -\mathbf{P}(t) \mathbf{Q} \mathbf{P}(t) + \mathbf{B} \mathbf{R}^{-1} \mathbf{B} - \mathbf{A}\mathbf{P}(t) - \mathbf{P}(t) \mathbf{A}^T,$$

with the final conditions

$$\mathbf{P}(t_f) = 0.$$

This Riccati equation can be integrated backwards in time for a solution.

It is very interesting, and powerful, to note that, if one chooses $\mathbf{Q} = 0$, therefore imposing no position cost on the trajectory before time $T$, then this inverse Riccati equation becomes a linear ODE which can be solved explicitly. These relationships are used in the derivation of the controllability Grammian, but here we use them to design a feedback controller.

## 8.3 Variations and Extensions

### 8.3.1 Discrete-time Riccati Equations

Essentially all of the results above have a natural correlate for discrete-time systems. What's more, the discrete time versions tend to be simpler to think about in the model-predictive control (MPC) setting that we'll be discussing below and in the next chapters.

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If we once again take

$$J(\mathbf{x}, n) = \mathbf{x}^T\mathbf{S}[\mathbf{n}]\mathbf{x}, \quad \mathbf{S}[n] = \mathbf{S}^T[n] \succ 0,$$

then we have

$$\mathbf{u}^*[n] = -\mathbf{K}[\mathbf{n}]\mathbf{x}[n] = -(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}\mathbf{B}^T\mathbf{S}[n]\mathbf{A}\mathbf{x}[n],$$

yielding

$$\mathbf{S}[n-1] = \mathbf{Q} + \mathbf{A}^T\mathbf{S}[n]\mathbf{A} - (\mathbf{A}^T\mathbf{S}[n]\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}[n]\mathbf{A}), \quad \mathbf{S}[N] = 0,$$

which is the famous *Riccati difference equation*. The infinite-horizon LQR solution is given by the (positive-definite) fixed-point of this equation:

$$\mathbf{S} = \mathbf{Q} + \mathbf{A}^T\mathbf{S}\mathbf{A} - (\mathbf{A}^T\mathbf{S}\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{S}\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}\mathbf{A}).$$

Like in the continuous time case, this equation is so important that we have special numerical recipes for solving this discrete-time algebraic Riccati equation (DARE). Drake delegates to these numerical methods automatically when you evaluate the `LinearQuadraticRegulator` method on a system that has only discrete state and a single periodic timestep.

**Example 8.3 (Discrete-time vs Continuous-time LQR)**

You can explore the relationship between the discrete-time and continuous-time formulations in this notebook:

### 8.3.2 LQR with input and state constraints

A natural extension for linear optimal control is the consideration of strict constraints on the inputs or state trajectory. Most common are linear inequality constraints, such as $\forall n, |\mathbf{u}[n]| \leq 1$ or $\forall n, \mathbf{x}[n] \geq -2$ (any linear constraints of the form $\mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \leq \mathbf{e}$ can be solved with the same tools). Much is known about the solution to this problem in the discrete-time case, but it's computation is signficantly harder than the unconstrained case. Almost always, we give up on solving for the best control policy in closed form, and instead solve for the optimal control trajectory $\mathbf{u}[\cdot]$ from a particular initial condition $\mathbf{x}[0]$ over some finite horizon. Fortunately, this problem is a convex optimization and we can often solve it quickly and reliably enough to solve it at every timestep, effectively turning a motion planning algorithm into a feedback controller; this idea is famously known as model-predictive control (MPC). We will provide the details in the trajectory optimization chapter.

We do actually understand what the optimal policy of the inequality-constrained LQR problem looks like, thanks to work on "explicit MPC" [1] -- the optimal policy is now piecewise-linear (though still continuous), with each piece describe by a polytope, and the optimal cost-to-go is piecewise-quadratic on the same polytopes. Unfortunately, the number of pieces grows exponentially with the number of constraints and the horizon of the problem, making it impractical to compute for all but very small problems. There are, howeer, a number of promising approaches to approximate explicit MPC (c.f. [2]).

One important case that does have closed-form solutions is LQR with linear *equality* constraints (c.f. [3], section IIIb). This is particularly relevant in the case of stabilizing robots with kinematic constraints such as a closed kinematic chain, which appears in four-bar linkages or even for the linearization of a biped robot with two feet on the ground.

### 8.3.3 LQR as a convex optimization

One can also design the LQR gains using linear matrix inequalities (LMIs). I will defer the derivation til we cover the policy gradient view of LQR, because the LMI formulation is based on a change of variables from the basic policy evaluation criterion. If you want to look ahead, you can find that formulation here.

Solving the algebraic Riccati equation is still the preferred way of computing the LQR solution. But it is helpful to know that one could also compute it with convex optimization. In addition to deepening our understanding, this can be useful for generalizing the basic LQR solution (e.g. for robust stabilization) or to solve for the LQR gains jointly as part of a bigger optimization.

### 8.3.4 Finite-horizon LQR via least squares

We can also obtain the solution to the discrete-time finite-horizon (including the time-varying or tracking variants) LQR problem using optimization -- in this case it actually reduces to a simple least-squares problem. The presentation in this section can be viewed as a simple implementation of the Youla parameterization (sometimes called "Q-parameterization") from controls. Small variations on the formulation here play an important role in the minimax variants of LQR (which optimize a worst-case performance), which we will discuss in the robust control chapter (e.g. [4, 5]).

First, let us appreciate that the default parameterization is not convex. Given

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0$$
$$\text{subject to} \quad \mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$
$$\mathbf{x}[0] = \mathbf{x}_0$$

if we wish to search over controllers of the form

$$\mathbf{u}[n] = \mathbf{K}_n \mathbf{x}[n],$$

then we have

$$\begin{aligned}
\mathbf{x}[1] &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{K}_0\mathbf{x}_0, \\
\mathbf{x}[2] &= \mathbf{A}(\mathbf{A} + \mathbf{B}\mathbf{K}_0)\mathbf{x}_0 + \mathbf{B}\mathbf{K}_1(\mathbf{A} + \mathbf{B}\mathbf{K}_0)\mathbf{x}_0 \\
\mathbf{x}[n] &= \left(\prod_{i=0}^{n-1}(\mathbf{A} + \mathbf{B}\mathbf{K}_i)\right)\mathbf{x}_0
\end{aligned}$$

As you can see, the $\mathbf{x}[n]$ terms in the cost function include our decision variables multiplied together -- resulting in a non-convex objective. The trick is to re-parameterize the decision variables, and write the feedback in the form:

$$\mathbf{u}[n] = \tilde{\mathbf{K}}_n \mathbf{x}_0,$$

leading to

$$\begin{aligned}
\mathbf{x}[1] &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\tilde{\mathbf{K}}_0\mathbf{x}_0, \\
\mathbf{x}[2] &= \mathbf{A}(\mathbf{A} + \mathbf{B}\tilde{\mathbf{K}}_0)\mathbf{x}_0 + \mathbf{B}\tilde{\mathbf{K}}_1\mathbf{x}_0 \\
\mathbf{x}[n] &= \left(\mathbf{A}^n + \sum_{i=0}^{n-1}\mathbf{A}^{n-i-1}\mathbf{B}\tilde{\mathbf{K}}_i\right)\mathbf{x}_0
\end{aligned}$$

Now all of the decision variables, $\tilde{\mathbf{K}}_i$, appear linearly in the solution to $\mathbf{x}[n]$ and therefore (convex) quadratically in the objective.

We still have an objective function that depends on $\mathbf{x}_0$, but we would like to find the optimal $\tilde{\mathbf{K}}_i$ *for all* $\mathbf{x}_0$. To achieve this let us evaluate the optimality conditions of this least squares problem, starting by taking the gradient of the objective with respect to $\tilde{\mathbf{K}}_i$, which is:

$$\mathbf{x}_0\mathbf{x}_0^T\left(\tilde{\mathbf{K}}_i^T\left(\mathbf{R} + \sum_{m=i+1}^{N-1}\mathbf{B}^T(\mathbf{A}^{m-i-1})^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right) + \sum_{m=i+1}^{N-1}(\mathbf{A}^m)^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right).$$

We can satisfy this optimality condition for all $\mathbf{x}_0$ by solving the *linear* matrix equation:

$$\tilde{\mathbf{K}}_i^T\left(\mathbf{R} + \sum_{m=i+1}^{N-1}\mathbf{B}^T(\mathbf{A}^{m-i-1})^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right) + \sum_{m=i+1}^{N-1}(\mathbf{A}^m)^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B} = 0.$$

We can always solve for $\tilde{\mathbf{K}}_i$ since it's multiplied by a (symmetric) positive definite matrix (it is the sum of a positive definite matrix and many positive semi-definite matrices), which is always invertible.

If you need to recover the original $\mathbf{K}_i$ parameters, you can extract them recursively with

$$\begin{aligned}
\tilde{\mathbf{K}}_0 &= \mathbf{K}_0, \\
\tilde{\mathbf{K}}_n &= \mathbf{K}_n\prod_{i=0}^{n-1}(\mathbf{A} + \mathbf{B}\mathbf{K}_i), \qquad 0 < n \leq N-1.
\end{aligned}$$

But often this is not actually necessary. In some applications it's enough to know the performance cost under LQR control, or to handle the response to disturbances explicitly with the disturbance-based feedback (which I've already promised for the robust control chapter). Afterall, the problem formulation that we've written here, which makes no mention of disturbances, assumes the model is perfect and the controls $\tilde{\mathbf{K}}_n\mathbf{x}_0$ are just as suitable for deployment as $\mathbf{K}_n\mathbf{x}[n]$.

"System-Level Synthesis" (SLS) is the name for an important and slightly different

approach, where one optimizes the *closed-loop response* directly[6]. Although SLS is a very general tool, for the particular formulation we are considering here it reduces to creating additional decision variables $\mathbf{\Phi}_i$, such at that

$$\mathbf{x}[n] = \mathbf{\Phi}_n \mathbf{x}[0],$$

and writing the optimization above as

$$\begin{aligned} \min_{\tilde{\mathbf{K}}_\star, \mathbf{\Phi}_\star} & \sum_{n=0}^{N-1} \mathbf{x}^T[0] \left( \mathbf{\Phi}_n^T \mathbf{Q} \mathbf{\Phi}_n + \tilde{\mathbf{K}}_n^T \mathbf{R} \tilde{\mathbf{K}}_n \right) \mathbf{x}[0], \\ \text{subject to} & \quad \forall n, \quad \mathbf{\Phi}_{n+1} = \mathbf{A}\mathbf{\Phi}_n + \mathbf{B}\tilde{\mathbf{K}}_n. \end{aligned}$$

Once again, the algorithms presented here are not as efficient as solving the Riccati equation if we only want the solution to the simple case, but they become very powerful if we want to combine the LQR synthesis with other objectives/constraints. For instance, if we want to add some sparsity constraints (e.g. enforcing that some elements of $\tilde{\mathbf{K}}_i$ are zero), then we could solve the quadratic optimization subject to linear equality constraints [7].

## 8.4 Exercises

## 8.5 Notes

### 8.5.1 Finite-horizon LQR derivation (general form)

For completeness, I've included here the derivation for continuous-time finite-horizon LQR with all of the bells and whistles.

Consider an time-varying affine (approximation of a) continuous-time dynamical system in state-space form:

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u} + \mathbf{c}(t),$$

and a running cost function in the general quadratic form:

$$\ell(t, \mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \mathbf{Q}(t) \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix}^T \mathbf{R}(t) \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} + 2\mathbf{x}^T \mathbf{N}(t) \mathbf{u},$$

$$\forall t \in [t_0, t_f], \quad \mathbf{Q}(t) = \begin{bmatrix} \mathbf{Q}_{xx}(t) & \mathbf{q}_x(t) \\ \mathbf{q}_x^T(t) & q_0(t) \end{bmatrix}, \mathbf{Q}_{xx}(t) \succeq 0, \quad \mathbf{R}(t) = \begin{bmatrix} \mathbf{R}_{uu}(t) & \mathbf{r}_u(t) \\ \mathbf{r}_u^T(t) & r_0(t) \end{bmatrix}, \mathbf{R}_{uu}(t) \succ 0.$$

Observe that our LQR "optimal tracking" derivation fits in this form, as we can always write

$$(\mathbf{x} - \mathbf{x}_d(t))^T \mathbf{Q}_t (\mathbf{x} - \mathbf{x}_d(t)) + (\mathbf{u} - \mathbf{u}_d(t))^T \mathbf{R}_t (\mathbf{u} - \mathbf{u}_d(t)) + 2(\mathbf{x} - \mathbf{x}_d(t))^T \mathbf{N}_t (\mathbf{u} - \mathbf{u}_d(t)),$$

by taking

$$\begin{aligned} \mathbf{Q}_{xx} &= \mathbf{Q}_t, \quad \mathbf{q}_x = -\mathbf{Q}_t \mathbf{x}_d - \mathbf{N}_t \mathbf{u}_d, \quad q_0 = \mathbf{x}_d^T \mathbf{Q}_t \mathbf{x}_d + 2\mathbf{x}_d^T \mathbf{N}_t \mathbf{u}_d, \\ \mathbf{R}_{uu} &= \mathbf{R}_t, \quad \mathbf{r}_u = -\mathbf{R}_t \mathbf{u}_d - \mathbf{N}_t^T \mathbf{x}_d, \quad r_0 = \mathbf{u}_d^T \mathbf{R}_t \mathbf{u}_d, \quad \mathbf{N} = \mathbf{N}_t. \end{aligned}$$

Of course, we can also add a quadratic final cost with $\mathbf{Q}_f$. Let's search for a positive quadratic, time-varying cost-to-go function of the form:

$$J(t, \mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \mathbf{S}(t) \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \quad \mathbf{S}(t) = \begin{bmatrix} \mathbf{S}_{xx}(t) & \mathbf{s}_x(t) \\ \mathbf{s}_x^T(t) & s_0(t) \end{bmatrix}, \mathbf{S}_{xx}(t) \succ 0,$$

$$\frac{\partial J}{\partial \mathbf{x}} = 2\mathbf{x}^T \mathbf{S}_{xx} + 2\mathbf{s}_x^T, \quad \frac{\partial J}{\partial t} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \dot{\mathbf{S}} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}.$$

Writing out the HJB:

$$\min_{\mathbf{u}} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}} [\mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u} + \mathbf{c}(t)] + \frac{\partial J}{\partial t} \right] = 0,$$

we can find the minimizing $\mathbf{u}$ with

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}} &= 2\mathbf{u}^T \mathbf{R}_{uu} + 2\mathbf{r}_u^T + 2\mathbf{x}^T \mathbf{N} + (2\mathbf{x}^T \mathbf{S}_{xx} + 2\mathbf{s}_x^T)\mathbf{B} = 0 \\ \mathbf{u}^* &= -\mathbf{R}_{uu}^{-1} \begin{bmatrix} \mathbf{N} + \mathbf{S}_{xx}\mathbf{B} \\ \mathbf{r}_u^T + \mathbf{s}_x^T \mathbf{B} \end{bmatrix}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = -\mathbf{K}(t) \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = -\mathbf{K}_x(t)\mathbf{x} - \mathbf{k}_0(t). \end{aligned}$$

Inserting this back into the HJB gives us the updated Riccati differential equation. Since this must hold for all $\mathbf{x}$, we can collect the quadratic, linear, and offset terms and set them each individually equal to zero, yielding:

$$\begin{aligned} -\dot{\mathbf{S}}_{xx} =& \mathbf{Q}_{xx} - (\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})\mathbf{R}_{uu}^{-1}(\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})^T + \mathbf{S}_{xx}\mathbf{A} + \mathbf{A}^T\mathbf{S}_{xx}, \\ -\dot{\mathbf{s}}_x =& \mathbf{q}_x - (\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})\mathbf{R}_{uu}^{-1}(\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x) + \mathbf{A}^T\mathbf{s}_x + \mathbf{S}_{xx}\mathbf{c}, \\ -\dot{s}_0 =& q_0 + r_0 - (\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x)^T\mathbf{R}_{uu}^{-1}(\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x) + 2\mathbf{s}_x^T\mathbf{c}, \end{aligned}$$

with the final conditions $\mathbf{S}(t_f) = \mathbf{Q}_f$.

In the discrete-time version, we have...

Phew! Numerical solutions to these equations can be obtained by calling the `FiniteHorizonLinearQuadraticRegulator` methods in DRAKE. May you never have to type them in and unit test them yourself.

## REFERENCES


1. A. Alessio and A. Bemporad, "A survey on explicit model predictive control", *Int. Workshop on Assessment and Future Directions of Nonlinear Model Predictive Control*, 2009.

2. Tobia Marcucci and Robin Deits and Marco Gabiccini and Antonio Bicchi and Russ Tedrake, "Approximate Hybrid Model Predictive Control for Multi-Contact Push Recovery in Complex Environments", *Humanoid Robots (Humanoids), 2017 IEEE-RAS 17th International Conference on*, 2017. [ link ]

3. Michael Posa and Scott Kuindersma and Russ Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems", *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 1366-1373, May, 2016. [ link ]

4. J. Lofberg, "Approximations of closed-loop minimax {MPC}", *42nd {IEEE} {International} {Conference} on {Decision} and {Control} ({IEEE} {Cat}. {No}.03CH37475)*, vol. 2, pp. 1438--1442 Vol.2, Dec, 2003.

5. Sadra Sadraddini and Russ Tedrake, "Robust Output Feedback Control with Guaranteed Constraint Satisfaction", *In the Proceedings of 23rd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 12, April, 2020. [ link ]

6. James Anderson and John C. Doyle and Steven Low and Nikolai Matni, "System {Level} {Synthesis}", *arXiv:1904.01634 [cs, math]*, apr, 2019.

7. Yuh-Shyang Wang and Nikolai Matni and John C. Doyle, "Localized {LQR} {Optimal} {Control}", *arXiv:1409.6404 [cs, math]*, September, 2014.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 9
# Lyapunov Analysis

Open in Colab

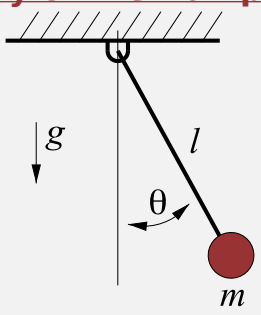
Optimal control provides a powerful framework for formulating control problems using the language of optimization. But solving optimal control problems for nonlinear systems is hard! In many cases, we don't really care about finding the *optimal* controller, but would be satisfied with any controller that is guaranteed to accomplish the specified task. In many cases, we still formulate these problems using computational tools from optimization, and in this chapter we'll learn about tools that can provide guaranteed control solutions for systems that are beyond the complexity for which we can find the optimal feedback.

There are many excellent books on Lyapunov analysis; for instance [1] is an excellent and very readable reference and [2] can provide a rigorous treatment. In this chapter I will summarize (without proof) some of the key theorems from Lyapunov analysis, but then will also introduce a number of numerical algorithms... many of which are new enough that they have not yet appeared in any mainstream textbooks.

## 9.1 Lyapunov Functions

Let's start with our favorite simple example.

**Example 9.1 (Stability of the Damped Pendulum)**



Recall that the equations of motion of the damped simple pendulum are given by

$$ml^2\ddot{\theta} + mgl\sin\theta = -b\dot{\theta},$$

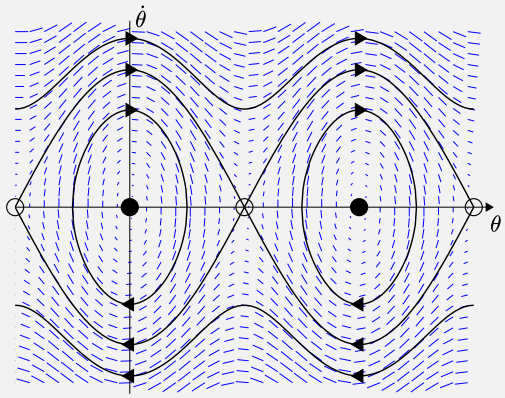
which I've written with the damping on the right-hand side to remind us that it is an external torque that we've modeled.

These equations represent a simple second-order differential equation; in chapter 2 we discussed at some length what was known about the solutions to this differential equation--in practice we do not have a closed-form solution for $\theta(t)$ as a function of the initial conditions. Since we couldn't provide a solution analytically, in chapter 2 we resorted to a graphical analysis, and confirmed the intuition that there are fixed points in the system (at $\theta = k\pi$ for every integer $k$) and that the fixed points at $\theta = 2\pi k$ are asymptotically stable with a large basin of attraction. The graphical analysis gave us this intuition, but can we actually prove this stability property? In a way that might also work for much more complicated systems?

One route forward was from looking at the total system energy (kinetic + potential), which we can write down:

$$E(\theta, \dot{\theta}) = \frac{1}{2}ml^2\dot{\theta}^2 - mgl\cos\theta.$$

Recall that the contours of this energy function are the orbits of the undamped pendulum.




A natural route to proving the stability of the downward fixed points is by arguing that energy decreases for the damped pendulum (with $b > 0$) and so the system will eventually come to rest at the minimum energy, $E = -mgl$, which happens at $\theta = 2\pi k$. Let's make that argument slightly more precise.

Evaluating the time derivative of the energy reveals

$$\frac{d}{dt}E = -b\dot{\theta}^2 \leq 0.$$

This is sufficient to demonstrate that the energy will never increase, but it doesn't actually prove that the energy will converge to the minimum when $b > 0$ because there are multiple states(not only the minimum) for which $\dot{E} = 0$. To take the last step, we must observe that set of states with $\dot{\theta} = 0$ is not an invariant set; that if the system is in, for instance $\theta = \frac{\pi}{4}, \dot{\theta} = 0$ that it will not stay there, because $\ddot{\theta} \neq 0$. And once it leaves that state, energy will decrease once again. In fact, the fixed points are the only subset the set of states where $\dot{E} = 0$ which do form an invariant set. Therefore we can conclude that as $t \to \infty$, the system will indeed come to rest at a fixed point (though it could be any fixed point with an energy less than or equal to the initial energy in the system, $E(0)$).

This is an important example. It demonstrated that we could use a relatively simple function -- the total system energy -- to describe something about the long-term dynamics of the pendulum even though the actual trajectories of the system are (analytically) very complex. It also demonstrated one of the subtleties of using an energy-like function that is non-increasing (instead of strictly decreasing) to prove asymptotic stability.

Lyapunov functions generalize this notion of an energy function to more general systems, which might not be stable in the sense of some mechanical energy. If I can find any positive function, call it $V(\mathbf{x})$, that gets smaller over time as the system evolves, then I can potentially use $V$ to make a statement about the long-term behavior of the system. $V$ is called a *Lyapunov function*.

Recall that we defined three separate notions for stability of a fixed-point of a nonlinear system: stability i.s.L., asymptotic stability, and exponential stability. We can use Lyapunov functions to demonstrate each of these, in turn.

## Theorem 9.1 - Lyapunov's Direct Method

Given a system $\dot{\mathbf{x}} = f(\mathbf{x})$, with $f$ continuous, and for some region $\mathcal{B}$ around the origin (specifically an open subset of $\mathbf{R}^n$ containing the origin), if I can produce a scalar, continuously-differentiable function $V(\mathbf{x})$, such that

$$V(\mathbf{x}) > 0, \forall \mathbf{x} \in \mathcal{B} \setminus \{0\} \quad V(0) = 0, \text{ and}$$
$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) \leq 0, \forall \mathbf{x} \in \mathcal{B} \setminus \{0\} \quad \dot{V}(0) = 0,$$

then the origin ($\mathbf{x} = 0$) is stable in the sense of Lyapunov (i.s.L.). [Note: the notation $A \setminus B$ represents the set $A$ with the elements of $B$ removed.]

If, additionally, we have

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) < 0, \forall \mathbf{x} \in \mathcal{B} \setminus \{0\},$$

then the origin is (locally) asymptotically stable. And if we have

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) \leq -\alpha V(x), \forall \mathbf{x} \in \mathcal{B} \setminus \{0\},$$

for some $\alpha > 0$, then the origin is (locally) exponentially stable.

Note that for the sequel we will use the notation $V \succ 0$ to denote a *positive-definite function*, meaning that $V(0) = 0$ and $V(\mathbf{x}) > 0$ for all $\mathbf{x} \neq 0$ (and also $V \succeq 0$ for positive semi-definite, $V \prec 0$ for negative-definite functions).

The intuition here is exactly the same as for the energy argument we made in the pendulum example: since $\dot{V}(x)$ is always zero or negative, the value of $V(x)$ will only get smaller (or stay the same) as time progresses. Inside the subset $\mathcal{B}$, for every $\epsilon$ -ball, I can choose a $\delta$ such that $|x(0)|^2 < \delta \Rightarrow |x(t)|^2 < \epsilon, \forall t$ by choosing $\delta$ sufficiently small so that the sublevel set of $V(x)$ for the largest value that $V(x)$ takes in the $\delta$ ball is completely contained in the $\epsilon$ ball. Since the value of $V$ can only get smaller (or stay constant) in time, this gives stability i.s.L.. If $\dot{V}$ is strictly negative away from the origin, then it must eventually get to the origin (asymptotic stability). The exponential condition is implied by the fact that $\forall t > 0, V(\mathbf{x}(t)) \leq V(\mathbf{x}(0))e^{-\alpha t}$.

Notice that the system analyzed above, $\dot{\mathbf{x}} = f(\mathbf{x})$, did not have any control inputs. Therefore, Lyapunov analysis is used to study either the passive dynamics of a system or the dynamics of a closed-loop system (system + control in feedback). We will see generalizations of the Lyapunov functions to input-output systems later in the text.

### 9.1.1 Global Stability

The notion of a fixed point being stable i.s.L. is inherently a local notion of stability (defined with $\epsilon$- and $\delta$- balls around the origin), but the notions of asymptotic and exponential stability can be applied globally. The Lyapunov theorems work for this case, too, with only minor modification.

**Theorem 9.2 - Lyapunov analysis for global stability**

Given a system $\dot{\mathbf{x}} = f(\mathbf{x})$, with $f$ continuous, if I can produce a scalar, continuously-differentiable function $V(\mathbf{x})$, such that

$$\begin{aligned} V(\mathbf{x}) &\succ 0, \\ \dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) &\prec 0, \text{ and} \\ V(\mathbf{x}) \to \infty &\text{ whenever } ||x|| \to \infty, \end{aligned}$$

then the origin ($\mathbf{x} = 0$) is globally asymptotically stable (G.A.S.).

If additionally we have that

$$\dot{V}(\mathbf{x}) \preceq -\alpha V(\mathbf{x}),$$

for some $\alpha > 0$, then the origin is globally exponentially stable.

The new condition, on the behavior as $||\mathbf{x}|| \to \infty$ is known as "radially unbounded", and is required to make sure that trajectories cannot diverge to infinity even as $V$ decreases; it is only required for global stability analysis.

### 9.1.2 LaSalle's Invariance Principle

Perhaps you noticed the disconnect between the statement above and the argument that we made for the stability of the pendulum. In the pendulum example, using the mechanical energy resulted in a Lyapunov function time derivative that was only negative semi-definite, but we eventually argued that the fixed points were asymptotically stable. That took a little extra work, involving an argument about the fact that the fixed points were the only place that the system could stay with $\dot{E} = 0$ ; every other state with $\dot{E} = 0$ was only transient. We can formalize this idea for the more general Lyapunov function statements--it is known as LaSalle's Theorem.

**Theorem 9.3 - LaSalle's Theorem**

Given a system $\dot{\mathbf{x}} = f(\mathbf{x})$ with $f$ continuous. If we can produce a scalar function $V(\mathbf{x})$ with continuous derivatives for which we have

$$V(\mathbf{x}) \succ 0, \quad \dot{V}(\mathbf{x}) \preceq 0,$$

and $V(\mathbf{x}) \to \infty$ as $||\mathbf{x}|| \to \infty$, then $\mathbf{x}$ will converge to the largest *invariant set* where $\dot{V}(\mathbf{x}) = 0$.

To be clear, an *invariant set*, $\mathcal{G}$, of the dynamical system is a set for which $\mathbf{x}(0) \in \mathcal{G} \Rightarrow \forall t > 0, \mathbf{x}(t) \in \mathcal{G}$. In other words, once you enter the set you never leave. The "largest invariant set" need not be connected; in fact for the pendulum example each fixed point is an invariant set, so the largest invariant set is the *union* of all the fixed points of the system. There are also variants of LaSalle's Theorem which work over a region.

Finding a Lyapunov function which $\dot{V} \prec 0$ is more difficult than finding one that has $\dot{V} \preceq 0$. LaSalle's theorem gives us the ability to make a statement about *asymptotic* stability even in this case. In the pendulum example, every state with $\dot{\theta} = 0$ had $\dot{E} = 0$ , but only the fixed points are in the largest invariant set.

## Example 9.2 (Swing-up for the Cart-Pole System)

Recall the example of using partial-feedback linearization to generate a swing-up controller for the cart-pole system. We first examined the dynamics of the pole (pendulum) only, by writing it's energy:

$$E(\mathbf{x}) = \frac{1}{2}\dot{\theta}^2 - \cos\theta,$$

desired energy, $E^d = 1$, and the difference $\tilde{E}(\mathbf{x}) = E(\mathbf{x}) - E^d$. We were able to show that our proposed controller produced

$$\dot{\tilde{E}} = -k\dot{\theta}^2 \cos^2\theta \tilde{E},$$

where $k$ is a positive gain that we get to choose. And we said that was good!

Now we have the tools to understand that really we have a Lyapunov function

$$V(\mathbf{x}) = \frac{1}{2}\tilde{E}^2(\mathbf{x}),$$

, and what we have shown is that $\dot{V} \leq 0$. By LaSalle, we can only argue that the closed-loop system will converge to the largest invariant set, which here is the entire homoclinic orbit: $\tilde{E}(\mathbf{x}) = 0$. We have to switch to the LQR controller in order to stabilize the upright.

Figure 9.1 - Lyapunov function: $V(\mathbf{x}) = \frac{1}{2}\tilde{E}^2(\mathbf{x})$.

Figure 9.2 - Time-derivative of the Lyapunov function: $\dot{V}(\mathbf{x})$.

As you can see from the plots, $\dot{V}(\mathbf{x})$ ends up being a quite non-trivial function! We'll develop the computational tools for verifying the Lyapunov/LaSalle conditions for systems of this complexity in the upcoming sections.

### 9.1.3 Relationship to the Hamilton-Jacobi-Bellman equations

At this point, you might be wondering if there is any relationship between Lyapunov functions and the cost-to-go functions that we discussed in the context of dynamic programming. After all, the cost-to-go functions also captured a great deal about the long-term dynamics of the system in a scalar function. We can see the connection if we re-examine the HJB equation

$$0 = \min_{\mathbf{u}} \left[ \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}). \right]$$

Let's imagine that we can solve for the optimizing $\mathbf{u}^*(\mathbf{x})$, then we are left with $0 = \ell(\mathbf{x}, \mathbf{u}^*) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}^*)$ or simply

$$\dot{J}^*(\mathbf{x}) = -\ell(\mathbf{x}, \mathbf{u}^*) \qquad \text{vs} \qquad \dot{V}(\mathbf{x}) \preceq 0.$$

In other words, in optimal control we must find a cost-to-go function which matches this gradient for every $\mathbf{x}$; that's very difficult and involves solving a potentially high-dimensional partial differential equation. By contrast, Lyapunov analysis is asking for much less - any function which is going downhill (at any rate) for all states. This can be much easier, for theoretical work, but also for our numerical algorithms. Also note that if we do manage to find the optimal cost-to-go, $J^*(\mathbf{x})$, then it can also serve as a Lyapunov function so long as $\ell(\mathbf{x}, \mathbf{u}^*(\mathbf{x})) \succeq 0$.

## 9.2 LYAPUNOV ANALYSIS WITH CONVEX OPTIMIZATION

One of the primary limitations in Lyapunov analysis as I have presented it so far is that it is potentially very difficult to come up with suitable Lyapunov function candidates for interesting systems, especially for underactuated systems. ("Underactuated" is almost synonymous with "interesting" in my vocabulary.) Even if somebody were to give me a Lyapunov candidate for a general nonlinear system, the Lyapunov conditions can be difficult to check -- for instance, how would I check that $\dot{V}$ is strictly negative for all $\mathbf{x}$ except the origin if $\dot{V}$ is some arbitrarily complicated nonlinear function over a vector $\mathbf{x}$?

In this section, we'll look at some computational approaches to verifying the Lyapunov conditions, and even to searching for (the coefficients of) the Lyapunov functions themselves.

If you're imagining numerical algorithms to check the Lyapunov conditions for complicated Lyapunov functions and complicated dynamics, the first thought is probably that we can evaluate $V$ and $\dot{V}$ at a large number of sample points and check whether $V$ is positive and $\dot{V}$ is negative. This does work, and could potentially be combined with some smoothness or regularity assumptions to generalize beyond the sample points. But in many cases we will be able to do better -- providing optimization algorithms that will rigorously check these conditions *for all* $\mathbf{x}$ without dense sampling; these will also give us additional leverage in formulating the search for Lyapunov functions.

### 9.2.1 Lyapunov analysis for linear systems

Let's take a moment to see how things play out for linear systems.

**Theorem 9.4 - Lyapunov analysis for stable linear systems**

Imagine you have a linear system, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, and can find a Lyapunov function

$$V(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{x}, \quad \mathbf{P} = \mathbf{P}^\mathbf{T} \succ 0,$$

which also satisfies

$$\dot{V}(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}^T\mathbf{P}\mathbf{x} \prec 0.$$

Then the origin is globally asymptotically stable.

Note that the radially-unbounded condition is satisfied by $\mathbf{P} \succ 0$, and that the derivative condition is equivalent to the matrix condition

$$\mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P} \prec 0.$$

For stable linear systems the existence of a quadratic Lyapunov function is actually a necessary (as well as sufficient) condition. Furthermore, a Lyapunov function can always be found by finding the positive-definite solution to the matrix Lyapunov equation

$$\mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P} = -\mathbf{Q}, \tag{1}$$

for any $\mathbf{Q} = \mathbf{Q}^T \succ 0$.

This is a very powerful result - for nonlinear systems it will be potentially difficult to find a Lyapunov function, but for linear systems it is straight-forward. In fact, this result is often used to propose candidates for non-linear systems, e.g., by linearizing the equations and solving a local linear Lyapunov function which should be valid in

the vicinity of a fixed point.

### 9.2.2 Lyapunov analysis as a semi-definite program (SDP)

Lyapunov analysis for linear systems has an extremely important connection to convex optimization. In particular, we could have also formulated the Lyapunov conditions for linear systems above using *semi-definite programming* (SDP). Semidefinite programming is a subset of convex optimization -- an extremely important class of problems for which we can produce efficient algorithms that are guaranteed find the global optima solution (up to a numerical tolerance and barring any numerical difficulties).

If you don't know much about convex optimization or want a quick refresher, please take a few minutes to read the optimization preliminaries in the appendix. The main requirement for this section is to appreciate that it is possible to formulate efficient optimization problems where the constraints include specifying that one or more matrices are positive semi-definite (PSD). These matrices must be formed from a linear combination of the decision variables. For a trivial example, the optimization

$$\min_a a, \quad \text{subject to} \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix} \succeq 0,$$

returns $a = 0$ (up to numerical tolerances).

The value in this is immediate for linear systems. For example, we can formulate the search for a Lyapunov function for the linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ by using the parameters $\mathbf{p}$ to populate a symmetric matrix $\mathbf{P}$ and then write the SDP:

$$\underset{\mathbf{P}}{\text{find}} \quad \text{subject to} \quad \mathbf{P} \succeq 0, \quad \mathbf{P}\mathbf{A} + \mathbf{A}^T\mathbf{P} \preceq 0. \tag{2}$$

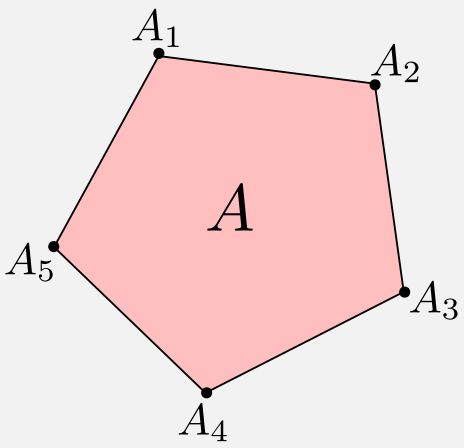
Note that you would probably never use that particular formulation, since there specialized algorithms for solving the simple Lyapunov equation which are more efficient and more numerically stable. But the SDP formulation does provide something new -- we can now easily formulate the search for a "*common Lyapunov function*" for uncertain linear systems.

#### Example 9.3 (Common Lyapunov analysis for linear systems)

Suppose you have a system governed by the equations $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, where the matrix $\mathbf{A}$ is unknown, but its uncertain elements can be bounded. There are a number of ways to write down this uncertainty set; let us choose to write this by describing $\mathbf{A}$ as the convex combination of a number of known matrices,

$$\mathbf{A} = \sum_i \beta_i \mathbf{A}_i, \quad \sum_i \beta_i = 1, \quad \forall i, \beta_i > 0.$$

This is just one way to specify the uncertainty; geometrically it is describing a polygon of uncertain parameters (in the space of elements of $\mathbf{A}$ with each $\mathbf{A}_i$ as one of the vertices in the polygon.




Now we can formulate the search for a common Lyapunov function using

$$\underset{\mathbf{P}}{\text{find}} \quad \text{subject to} \quad \mathbf{P} \succeq 0, \quad \forall_i, \mathbf{P}\mathbf{A}_i + \mathbf{A}_i^T\mathbf{P} \preceq 0.$$

The solver will then return a matrix $\mathbf{P}$ which satisfies all of the constraints, or return saying "problem is infeasible". It can easily be verified that if $\mathbf{P}$ satisfies the Lyapunov condition at all of the vertices, then it satisfies the condition for every $\mathbf{A}$ in the set:

$$\mathbf{P}(\sum_i \beta_i \mathbf{A}_i) + (\sum_i \beta_i \mathbf{A}_i)^T\mathbf{P} = \sum_i \beta_i(\mathbf{P}\mathbf{A}_i + \mathbf{A}_i^T\mathbf{P}) \preceq 0,$$

since $\forall i$, $\beta_i > 0$. Note that, unlike the simple Lyapunov equation for a known linear system, this condition being satisfied is a sufficient but not a necessary condition -- it is possible that the set of uncertain matrices $\mathbf{A}$ is robustly stable, but that this stability cannot be demonstrated with a common quadratic Lyapunov function.

You can try this example for yourself in DRAKE.

Open in Colab

As always, make sure that you open up the code and take a look.

There are many small variants of this result that are potentially of interest. For instance, a very similar set of conditions can certify "mean-square stability" for linear systems with multiplicative noise (see e.g. [3], § 9.1.1).

This example is very important because it establishes a connection between Lyapunov functions and (convex) optimization. But so far we've only demonstrated this connection for linear systems where the PSD matrices provide a magical recipe for establishing the positivity of the (quadratic) functions for all $\mathbf{x}$. Is there any hope of extending this type of analysis to more general nonlinear systems? Surprisingly, it turns out that there is.

## 9.2.3 Lyapunov analysis for polynomial systems

Sums of squares optimization provides a natural generalization of SDP to optimizing over positive polynomials (if you are not familiar, take a moment to read the appendix). This suggests that it may be possible to generalize the optimization approach using SDP to search for Lyapunov functions for linear systems to searching for Lyapunov functions for at least the polynomial systems: $\dot{\mathbf{x}} = f(\mathbf{x})$, where $f$ is a vector-valued polynomial function. If we parameterize a *fixed-degree* Lyapunov

candidate as a polynomial with unknown coefficients, e.g.,

$$V_\alpha(\mathbf{x}) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_1 x_2 + \alpha_4 x_1^2 + \ldots,$$

then the time-derivative of $V$ is also a polynomial, and I can formulate the optimization:

$$\begin{aligned} \underset{\alpha}{\text{find}}, \quad \text{subject to} \quad & V_\alpha(\mathbf{x}) \text{ is SOS} \\ & -\dot{V}_\alpha(\mathbf{x}) = -\frac{\partial V_\alpha}{\partial \mathbf{x}} f(\mathbf{x}) \text{ is SOS}. \end{aligned}$$

Because this is a convex optimization, the solver will return a solution if one exists.

## Example 9.4 (Verifying a Lyapunov candidate via SOS)

This example is example 7.2 from [4]. Consider the nonlinear system:

$$\begin{aligned} \dot{x}_0 &= -x_0 - 2x_1^2 \\ \dot{x}_1 &= -x_1 - x_0 x_1 - 2x_1^3, \end{aligned}$$

and the *fixed* Lyapunov function $V(x) = x_0^2 + 2x_1^2$, test if $\dot{V}(x)$ is negative definite.

The numerical solution can be written in a few lines of code, and is a convex optimization.


## Example 9.5 (Cart-pole swingup (again))

In the cart-pole swingup example, we took

$$V(\mathbf{x}) = \frac{1}{2}\tilde{E}^2(\mathbf{x}) = \frac{1}{2}(\frac{1}{2}\dot{\theta}^2 - \cos\theta - 1)^2.$$

This is clearly a sum of squares. Furthermore, we showed that

$$\dot{V}(\mathbf{x}) = -\dot{\theta}^2 \cos^2\theta \tilde{E}^2(\mathbf{x}),$$

which is also a sum of squares. So the proposal of using sums-of-squares optimization is not so different, actually, than the recipes that nonlinear control theorists have been using (on pen and paper) for years. In this case, I would need a basis vector that includes many more monomials: $[1, \dot{\theta}, \cos\theta, \ldots, \dot{\theta}^3, \ldots]^T$, and setting up the equality constraints in an optimization requires more complicated term matching when trigonometric functions are involved, but the recipe still works.

## Example 9.6 (Searching for a Lyapunov function via SOS)

Verifying a candidate Lyapunov function is all well and good, but the real excitement starts when we use optimization to *find* the Lyapunov function. In the following code, we parameterize $V(x)$ as the polynomial containing all monomials up to degree 2, with the coefficients as decision variables:

$$V(x) = c_0 + c_1 x_0 + c_2 x_1 + c_3 x_0^2 + c_4 x_0 x_1 + c_5 x_1^2.$$

We will set the scaling (arbitrarily) to avoid numerical issues by setting $V(0) = 0$, $V([1, 0]) = 1$. Then we write:

$$\underset{\mathbf{c}}{\text{find}} \quad \text{subject to } V \text{ is sos},$$
$$-\dot{V} \text{ is sos}.$$


Up to numerical convergence tolerance, it discovers the same coefficients that we chose above (zeroing out the unnecessary terms).

It is important to remember that there are a handful of gaps which make the existence of this solution a sufficient condition (for proving that every sublevel set of $V$ is an invariant set of $f$) instead of a necessary one. First, there is no guarantee that a stable polynomial system can be verified using a polynomial Lyapunov function (of any degree, and in fact there are known counter-examples [5]) and here we are only searching over a fixed-degree polynomial. Second, even if a polynomial Lyapunov function does exist, there is a gap between the SOS polynomials and the positive polynomials.

Despite these caveats, I have found this formulation to be surprisingly effective in practice. Intuitively, I think that this is because there is relatively a lot of flexibility in the Lyapunov conditions -- if you can find one function which is a Lyapunov function for the system, then there are also many "nearby" functions which will satisfy the same constraints.

## 9.3 Lyapunov functions for estimating regions of attraction

There is another very important connection between Lyapunov functions and the concept of an invariant set: *any sublevel set of a Lyapunov function is also an invariant set*. This gives us the ability to use sublevel sets of a Lyapunov function as approximations of the region of attraction for nonlinear systems.

### Theorem 9.5 - Lyapunov invariant set and region of attraction theorem

Given a system $\dot{\mathbf{x}} = f(\mathbf{x})$ with $f$ continuous, if we can find a scalar function $V(\mathbf{x}) \succ 0$ and a sublevel set

$$\mathcal{G} : \{\mathbf{x} | V(\mathbf{x}) \leq \rho\}$$

on which

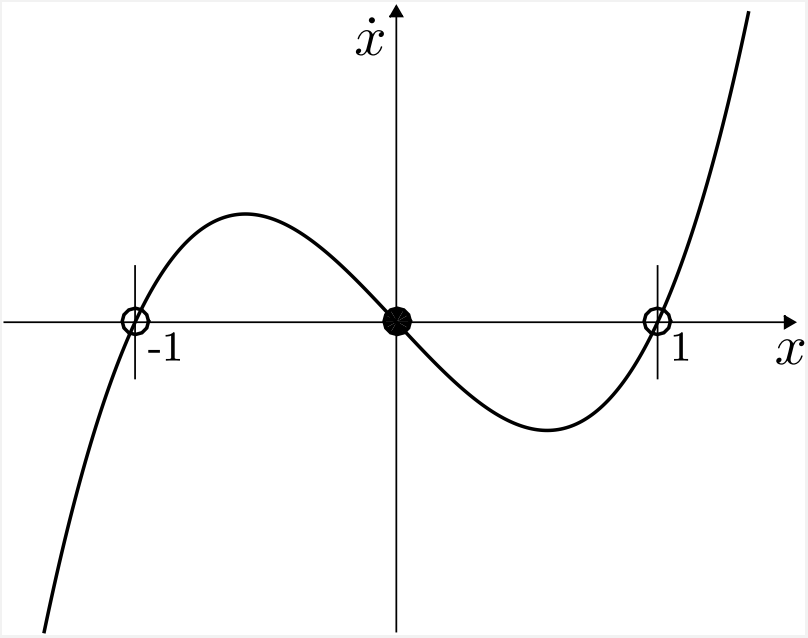
$$\forall \mathbf{x} \in \mathcal{G}, \dot{V}(\mathbf{x}) \preceq 0,$$

then $\mathcal{G}$ is an invariant set. By LaSalle, $\mathbf{x}$ will converge to the largest invariant subset of $\mathcal{G}$ on which $\dot{V} = 0$.

Furthermore, if $\dot{V}(\mathbf{x}) \prec 0$ in $\mathcal{G}$, then the origin is locally asymptotically stable and the set $\mathcal{G}$ is inside the region of attraction of this fixed point. Alternatively, if $\dot{V}(\mathbf{x}) \preceq 0$ in $\mathcal{G}$ and $\mathbf{x} = 0$ is the only invariant subset of $\mathcal{G}$ where $\dot{V} = 0$, then the origin is asymptotically stable and the set $\mathcal{G}$ is inside the region of attraction of this fixed point.

### Example 9.7 (Region of attraction for a one-dimensional system)

Consider the first-order, one-dimensional system $\dot{x} = -x + x^3$. We can quickly understand this system using our tools for graphical analysis.




In the vicinity of the origin, $\dot{x}$ looks like $-x$, and as we move away it looks increasingly like $x^3$. There is a stable fixed point at the origin and unstable fixed points at $\pm 1$. In fact, we can deduce visually that the region of attraction to the stable fixed point at the origin is $\mathbf{x} \in (-1, 1)$. Let's see if we can demonstrate this with a Lyapunov argument.

First, let us linearize the dynamics about the origin and use the Lyapunov equation for linear systems to find a candidate $V(\mathbf{x})$. Since the linearization is $\dot{x} = -x$, if we take $\mathbf{Q} = 1$, then we find $\mathbf{P} = \frac{1}{2}$ is the positive definite solution to the algebraic Lyapunov equation (1). Proceeding with

$$V(\mathbf{x}) = \frac{1}{2}x^2,$$

we have

$$\dot{V} = x(-x + x^3) = -x^2 + x^4.$$

This function is zero at the origin, negative for $|x| < 1$, and positive for $|x| > 1$. Therefore we can conclude that the sublevel set $V < \frac{1}{2}$ is invariant and the set $x \in (-1, 1)$ is inside the region of attraction of the nonlinear system. In fact, this estimate is tight.

## 9.3.1 Robustness analysis using "common Lyapunov functions"

While we will defer most discussions on robustness analysis until later in the notes, the idea of a *common Lyapunov function*, which we introduced briefly for linear systems in the example above, can be readily extended to nonlinear systems and region of attraction analysis. Imagine that you have a model of a dynamical system

but that you are uncertain about some of the parameters. For example, you have a model of a quadrotor, and are fairly confident about the mass and lengths (both of which are easy to measure), but are not confident about the moment of inertia. One approach to robustness analysis is to define a bounded uncertainty, which could take the form of

$$\dot{\mathbf{x}} = f_\alpha(\mathbf{x}), \quad \alpha_{min} \leq \alpha \leq \alpha_{max},$$

with $\alpha$ a vector of uncertain parameters in your model. Richer specifications of the uncertainty bounds are also possible, but this will serve for our examples.

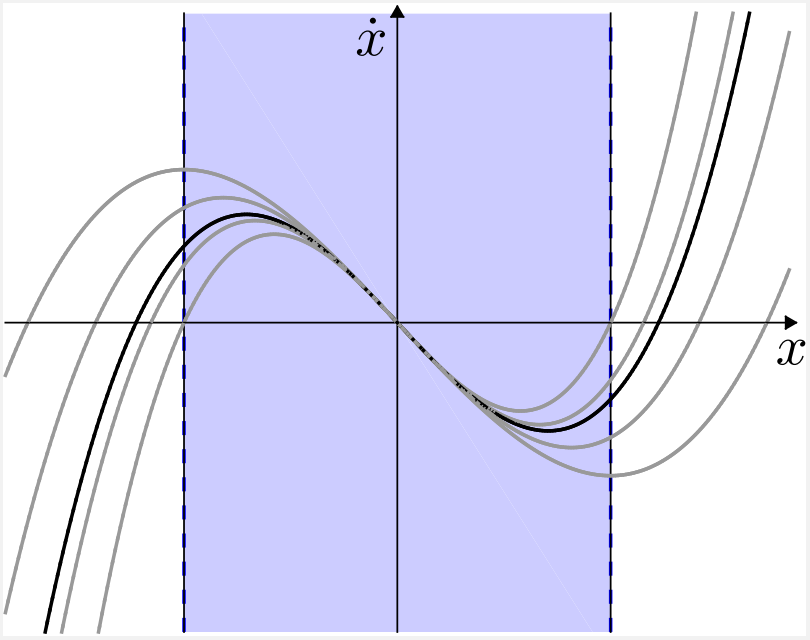
In standard Lyapunov analysis, we are searching for a function that goes downhill for all $\mathbf{x}$ to make statements about the long-term dynamics of the system. In common Lyapunov analysis, we can make many similar statements about the long-term dynamics of an uncertain system if we can find a single Lyapunov function that goes downhill *for all possible values of* $\alpha$. If we can find such a function, then we can use it to make statements with all of the variations we've discussed (local, regional, or global; in the sense of Lyapunov, asymptotic, or exponential).

## Example 9.8 (A one-dimensional system with gain uncertainty)

Let's consider the same one-dimensional example used above, but add an uncertain parameter into the dynamics. In particular, consider the system:

$$\dot{x} = -x + \alpha x^3, \quad \frac{3}{4} < \alpha < \frac{3}{2}.$$

Plotting the graph of the one-dimensional dynamics for a few values of $\alpha$, we can see that the fixed point at the origin is still stable, but the *robust region of attraction* to this fixed point (shaded in blue below) is smaller than the region of attraction for the system with $\alpha = 1$.




Taking the same Lyapunov candidate as above, $V = \frac{1}{2}x^2$, we have

$$\dot{V} = -x^2 + \alpha x^4.$$

This function is zero at the origin, and negative for all $\alpha$ whenever $x^2 > \alpha x^4$, or

$$|x| < \frac{1}{\sqrt{\alpha_{max}}} = \sqrt{\frac{2}{3}}.$$

Therefore, we can conclude that $|x| < \sqrt{\frac{2}{3}}$ is inside the robust region of attraction of the uncertain system.

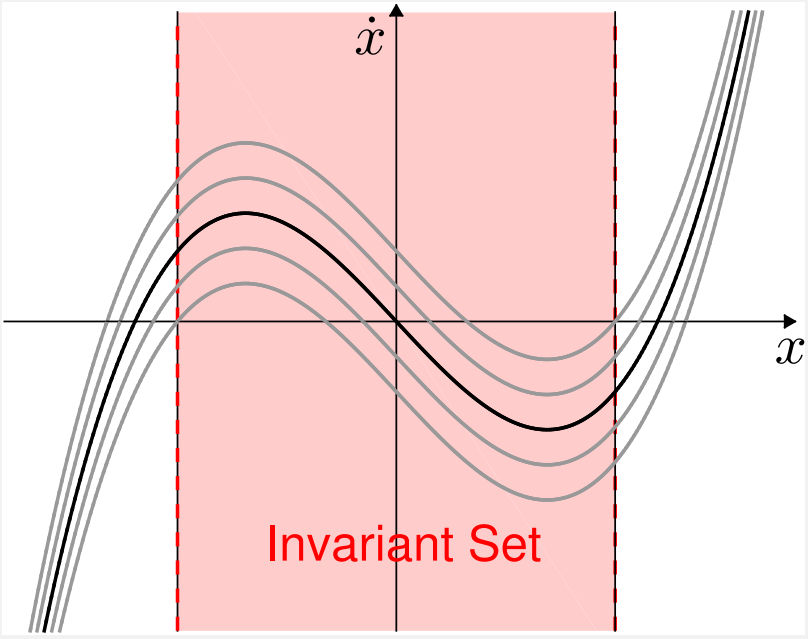
Not all forms of uncertainty are as simple to deal with as the gain uncertainty in that example. For many forms of uncertainty, we might not even know the location of the fixed points of the uncertain system. In this case, we can often still use common Lyapunov functions to give some guarantees about the system, such as guarantees of *robust set invariance*. For instance, if you have uncertain parameters on a quadrotor model, you might be ok with the quadrotor stabilizing to a pitch of 0.01 radians, but you would like to guarantee that it definitely does not flip over and crash into the ground.

## Example 9.9 (A one-dimensional system with additive uncertainty)

Now consider the system:

$$\dot{x} = -x + x^3 + \alpha, \quad -\frac{1}{4} < \alpha < \frac{1}{4}.$$

Plotting the graph of the one-dimensional dynamics for a few values of $\alpha$, this time we can see that the fixed point is not necessarily at the origin; the location of the fixed point moves depending on the value of $\alpha$. But we should be able to guarantee that the uncertain system will stay near the origin if it starts near the origin, using an invariant set argument.




Taking the same Lyapunov candidate as above, $V = \frac{1}{2}x^2$, we have

$$\dot{V} = -x^2 + x^4 + \alpha x.$$

This Lyapunov function allows us to easily verify, for instance, that $V \leq \frac{1}{3}$ is a

*robust invariant set*, because whenever $V = \frac{1}{3}$, we have

$$\forall \alpha \in [\alpha_{min}, \alpha_{max}], \quad \dot{V}(x, \alpha) < 0.$$

Therefore $V$ can never start at less than one-third and cross over to greater than one-third. To see this, see that

$$V = \frac{1}{3} \quad \Rightarrow \quad x = \pm\sqrt{\frac{2}{3}} \quad \Rightarrow \quad \dot{V} = -\frac{2}{9} \pm \alpha\sqrt{\frac{2}{3}} < 0, \forall \alpha \in \left[-\frac{1}{4}, \frac{1}{4}\right].$$

Note that not all sublevel sets of this invariant set are invariant. For instance $V < \frac{1}{32}$ does not satisfy this condition, and by visual inspection we can see that it is in fact not robustly invariant.

### 9.3.2 Region of attraction estimation for polynomial systems

Now we have arrived at the tool that I believe can be a work-horse for many serious robotics applications. Most of our robots are not actually globally stable (that's not because they are robots -- if you push me hard enough, I will fall down, too), which means that understanding the regions where a particular controller can be guaranteed to work can be of critical importance.

Sums-of-squares optimization effectively gives us an oracle which we can ask: is this polynomial positive for all $\mathbf{x}$? To use this for regional analysis, we have to figure out how to modify our questions to the oracle so that the oracle will say "yes" or "no" when we ask if a function is positive over a certain region which is a subset of $\mathbb{R}^n$. That trick is called the S-procedure. It is closely related to the Lagrange multipliers from constrained optimization, and has deep connections to "Positivstellensatz" from algebraic geometry.

## The S-procedure

Consider a scalar polynomial, $p(\mathbf{x})$, and a semi-algebraic set $g(\mathbf{x}) \preceq 0$, where $g$ is a vector of polynomials. If I can find a polynomial "multiplier", $\lambda(\mathbf{x})$, such that

$$p(\mathbf{x}) + \lambda^T(\mathbf{x}) g(\mathbf{x}) \text{ is SOS}, \quad \text{and} \quad \lambda(\mathbf{x}) \text{ is SOS},$$

then this is sufficient to demonstrate that

$$p(\mathbf{x}) \geq 0 \quad \forall \mathbf{x} \in \{\mathbf{x} | g(\mathbf{x}) \leq 0\}.$$

To convince yourself, observe that when $g(\mathbf{x}) \leq 0$, it is only harder to be positive, but when $g(\mathbf{x}) > 0$, it is possible for the combined function to be SOS even if $p(\mathbf{x})$ is negative. We will sometimes find it convenient to use the short-hand:

$$g(\mathbf{x}) \leq 0 \Rightarrow p(\mathbf{x}) \geq 0$$

to denote the implication certified by the S-procedure (e.g. "whenever $g(\mathbf{x}) \leq 0$, we have $p(\mathbf{x}) \geq 0$").

We can also handle equality constraints with only a minor modification -- we no longer require the multiplier to be positive. If I can find a polynomial "multiplier", $\lambda(\mathbf{x})$, such that

$$p(\mathbf{x}) + \lambda^T(\mathbf{x}) g(\mathbf{x}) \text{ is SOS}$$

then this is sufficient to demonstrate that

$$p(\mathbf{x}) \geq 0 \quad \forall \mathbf{x} \in \{\mathbf{x} | g(\mathbf{x}) = 0\}.$$

Here the intuition is that $\lambda(x)$ can add arbitrary positive terms to help me be SOS, but those terms contribute nothing precisely when $g(x) = 0$.

## Basic region of attraction formulation

The S-procedure gives us the tool we need to evaluate positivity only over a region of state space, which is precisely what we need to certify the Lyapunov conditions for a region-of-attraction analysis. Let us start with a positive-definite polynomial Lyapunov candidate, $V(\mathbf{x}) \succ 0$, then we can write the Lyapunov conditions:

$$\dot{V}(\mathbf{x}) \prec 0 \quad \forall \mathbf{x} \in \{\mathbf{x} | V(\mathbf{x}) \leq \rho\},$$

using sums-of-squares and the S-procedure:

$$-\dot{V}(\mathbf{x}) + \lambda(\mathbf{x})(V(\mathbf{x}) - \rho) \text{ is SOS}, \quad \text{and} \quad \lambda(\mathbf{x}) \text{ is SOS},$$

where $\lambda(\mathbf{x})$ is a multiplier polynomial with free coefficients that are to be solved for in the optimization.

I think it's easiest to see the details in an example.

### Example 9.10 (Region of attraction for the one-dimensional cubic system)

Let's return to our example from above:

$$\dot{x} = -x + x^3$$

and try to use SOS optimization to demonstrate that the region of attraction of the fixed point at the origin is $x \in (-1, 1)$, using the Lyapunov candidate $V = x^2$.

First, define the multiplier polynomial,

$$\lambda(x) = c_0 + c_1 x + c_2 x^2.$$

Then define the optimization

$$\begin{aligned} \underset{\mathbf{c}}{\text{find}} & \\ \text{subject to} \quad & -\dot{V}(x) + \lambda(x)(V(x) - 1) \text{ is SOS} \\ & \lambda(x) \text{ is SOS} \end{aligned}$$

You can try this example for yourself in DRAKE.


In this example, we only verified that the one-sublevel set of the pre-specified Lyapunov candidate is negative (certifying the ROA that we already understood). Even more useful is if you are able to search for the largest $\rho$ that can satisfy these conditions. Unfortunately, in this first formulation, optimizing $\rho$ directly would make the optimization problem non-convex because we would have terms like $\rho c_0$, $\rho c_1 x$, ... which are bilinear in the decision variables; we need the sums-of-squares constraints to be only linear in the decision variables.

Fortunately, because the problem is convex with $\rho$ fixed (and therefore can be solved reliably), and $\rho$ is just a scalar, we can perform a simple line search on $\rho$ to find the largest value for which the convex optimization returns a feasible solution. This will be our estimate for the region of attraction.

There are a number of variations to this basic formulation; I will describe a few of them below. There are also important ideas like rescaling and degree-matching that

can have a dramatic effect on the numerics of the problem, and potentially make them much better for the solvers. But you do not need to master them all in order to use the tools effectively.

### Example 9.11 (Region of Attraction codes in Drake)

In DRAKE, we have packaged most of the work in setting up and solving the sums-of-squares optimization for regions of attraction into a single method `RegionOfAttraction(system, context, options)`. This makes it as simple as, for instance:

```
x = Variable("x")
sys = SymbolicVectorSystem(state=[x], dynamics=[-x+x**3])
context = sys.CreateDefaultContext()
V = RegionOfAttraction(sys, context)
```


Remember that although we have tried to make it convenient to call these functions, they are not a black box. I highly recommend opening up the `RegionOfAttraction` method and understanding how it works. There are lots of different options / formulations, and numerous numerical recipes to improve the numerics of the optimization problem.

## The equality-constrained formulation

Here is one important variation for finding the level set of a candidate region of attraction, which turns an inequality in the S-procedure into an equality. This formulation *is* jointly convex in $\lambda(\mathbf{x})$ and $\rho$, so one can optimize them in a single convex optimization. It appeared informally in an example in [4], and was discussed a bit more in [6].

Under the assumption that the Hessian of $\dot{V}(\mathbf{x})$ is negative-definite at the origin (which is easily checked), we can write

$$\begin{aligned} \max_{\rho, \lambda(\mathbf{x})} \quad & \rho \\ \text{subject to} \quad & (\mathbf{x}^T\mathbf{x})^d(V(\mathbf{x}) - \rho) + \lambda(\mathbf{x})\dot{V}(\mathbf{x}) \text{ is SOS}, \end{aligned}$$

with $d$ a fixed positive integer. As you can see, $\rho$ no longer multiplies the coefficient of $\lambda(\mathbf{x})$. But why does this certify a region of attraction?

You can read the sums-of-squares constraint as certifying the implication that whenever $\dot{V}(x) = 0$, we have that either $V(\mathbf{x}) \geq \rho$ OR $\mathbf{x} = 0$. Multiplying by some multiple of $\mathbf{x}^T\mathbf{x}$ is just a clever way to handle the "or $\mathbf{x} = 0$" case, which is necessary since we expect $\dot{V}(0) = 0$. This implication is sufficient to prove that $\dot{V}(\mathbf{x}) \leq 0$ whenever $V(\mathbf{x}) \leq \rho$, since $V$ and $\dot{V}$ are smooth polynomials; we examine this claim in full detail in one of the exercises.

Using the S-procedure with equalities instead of inequalities also has the potential advantage of removing the SOS constraint on $\lambda(\mathbf{x})$. But perhaps the biggest advantage of this formulation is the possibility of dramatically simplifying the problem using the quotient ring of this algebraic variety, and in particular some recent results for exact certification using sampling varieties[6].

## Searching for $V(\mathbf{x})$

The machinery so far has used optimization to find the largest region of attraction that can be certified *given a candidate Lyapunov function*. This is not necessarily a

bad assumption. For most stable fixed points, we can certify the local stability with a linear analysis, and this linear analysis gives us a candidate quadratic Lyapunov function that can be used for nonlinear analysis over a region. Practically speaking, when we solve an LQR problem, the cost-to-go from LQR is a good candidate Lyapunov function for the nonlinear system. If we are simply analyzing an existing system, then we can obtain this candidate by solving a Lyapunov equation (Eq 1).

But what if we believe the system is regionally stable, despite having an indefinite linearization? Or perhaps we can certify a larger volume of state space by using a Lyapunov candidate with degree greater than two. Can we use sums-of-squares optimization to find that in the region of attraction case, too?

To accomplish this, we will now make $V(x)$ a polynomial (of some fixed degree) with the coefficients as decision variables. First we will need to add constraints to ensure that

$$V(0) = 0 \quad and \quad V(\mathbf{x}) - \epsilon \mathbf{x}^T \mathbf{x} \text{ is SOS},$$

where $\epsilon$ is some small positive constant. This $\epsilon$ term simply ensures that $V$ is strictly positive definite. Now let's consider our basic formulation:

$$-\dot{V}(\mathbf{x}) + \lambda(\mathbf{x})(V(\mathbf{x}) - 1) \text{ is } \quad \text{SOS}, \quad \text{and} \quad \lambda(\mathbf{x}) \text{ is SOS}.$$

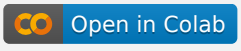
Notice that I've replaced $\rho = 1$; now that we are searching for $V$ the scale of the level-set can be set aribtrarily to 1. In fact, it's better to do so -- if we did not set $V(0) = 0$ and $V(\mathbf{x}) \leq 1$ as the sublevel set, then the optimization problem would be underconstrained, and might cause problems for the solver.

Unfortunately, the derivative constraint is now nonconvex (bilinear) in the decision variables, since we are searching for both the coefficients of $\lambda$ and $V$, and they are multiplied together. Our equality-constrained formulation doesn't get us around this one, either (since the coefficients of $V$ also appear in $\dot{V}$). Here we have to resort to some weaker form of optimization. In practice, we have had good practical success using bilinear alternations: start with an initial $V$ (e.g. from LQR), and search for $\lambda$; then fix $\lambda$ and search for $V$ and repeat until convergence (see, for instance [7, 8]). A typical choice for the objective is to maximize some convex surrogate for the volume of the certified region of attraction in state space; typically we use the determinant of the quadratic form describing a contained ellipse.

Barring numerical issues, this algorithm is guaranteed to have recursive feasibility and tends to converge in just a few alternations, but there is no guarantee that it finds the optimal solution.

### Example 9.12 (Searching for Lyapunov functions)

(Details coming soon...)




## Convex outer approximations of the ROA

All of our region of attraction approximations so far have been "inner approximations" -- the certified region is guaranteed to be contained within the true region of attraction for the system. This is what we want if we are to give some guarantee of stability. As we increase the degree of our polynomial and multipliers, we expect that the estimated regions of attraction will converge on the true regions.

It turns out that if we instead consider outer approximations, which converge on the true ROA from the other side, we can write formulations that enable searching for the Lyapunov function directly as a convex optimization. As we will see below, this approach also allows one to write convex formulations for controller design. These methods have been explored in a very nice line of work by Henrion and Korda

(e.g. [9]), using the method of moments from [10] also called "occupation measures". Their treatment emphasizes infinite-dimensional linear programs and heirarchies of LMI relaxations; these are just dual formulations to the SOS optimizations that we've been writing here. I find the SOS form more clear, so will stick to it here. Some of the occupation measure papers include the dual formulation of the infinite-dimension linear program which will look quite similar; my coauthors and I typically call out the sums-of-squares version in our papers on the topic (e.g. [11]).

To find an outer approximation, instead of solving for a Lyapunov function that certifies convergence to the origin, we use a very related set of conditions to search for a Lyapunov-like "barrier certificate", $\mathcal{B}(\mathbf{x})$. Like a Lyapunov function, we'd like $\dot{\mathcal{B}}(\mathbf{x}) \leq 0$; this time we'll ask for this to be true everywhere (or at least in some set that is sufficiently larger than the ROA of interest). Then we will set $\mathcal{B}(0) > 0$. If we can find such a function, then certainly any state for which $\mathcal{B}(\mathbf{x}) < 0$ is outside of the region of attraction of the fixed point -- since $\mathcal{B}$ cannot increase it can never reach the origin which has $\mathcal{B} > 0$. The *super*level set $\{\mathbf{x} | \mathcal{B}(\mathbf{x}) \geq 0\}$ is an outer approximation of the true region of attraction:

$$-\dot{\mathcal{B}}(\mathbf{x}) \text{ is SOS} \quad \text{and} \quad \mathcal{B}(0) > 0.$$

In order to find the smallest such outer approximation (given a fixed degree polynomial), we choose an objective that tries to "push-down" on the value of $\mathcal{B}(\mathbf{x})$. We typically accomplish this by introducing another polynomial function $W(\mathbf{x})$ with the requirements that

$$W(\mathbf{x}) \geq 0 \quad \text{and} \quad W(\mathbf{x}) \geq \mathcal{B}(\mathbf{x}) + 1,$$

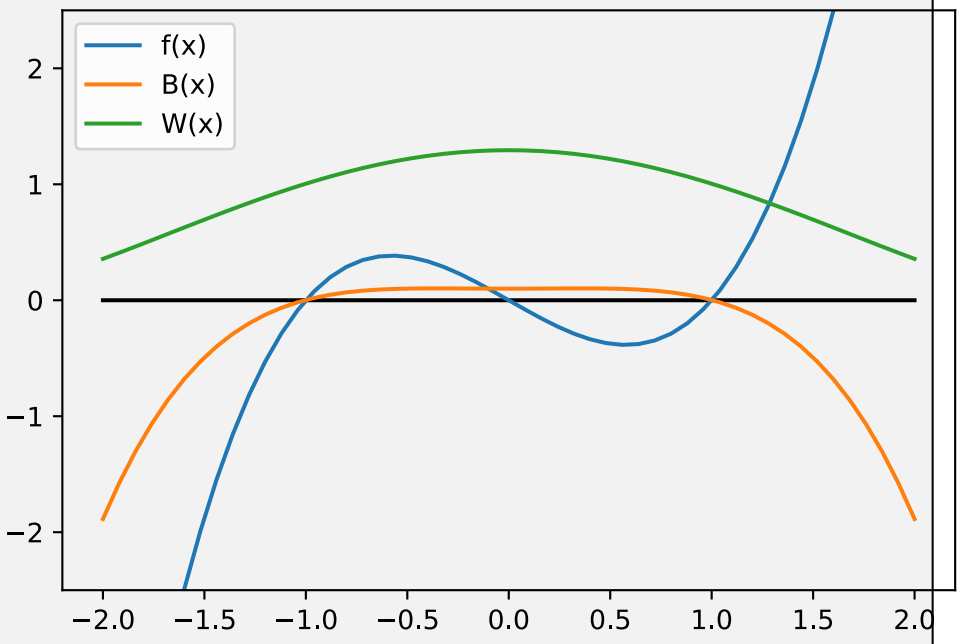
implemented as SOS constraints. Then we minimize the integral $\int_{\mathbf{x}} W(\mathbf{x}) d\mathbf{x}$, which can be readily computed over a compact set like a ball in $\mathbb{R}^n$ [12] and is linear in the coefficients. More sophisticated alternatives also exist[13].

## Example 9.13 (Outer approximation for the cubic polynomial)

Let's revisit my favorite simple dynamical system, $\dot{x} = -x + x^3$, which has a region of attraction for the fixed point at zero of $\{x | |x| < 1\}$. This time, we'll estimate the ROA using the outer approximation. We can accomplish this with the following program:

$$\begin{aligned} \min_{\mathcal{B}(x), W(x)} \quad & \int_{-2}^{2} W(x) dx, & \\ \text{subject to} \quad & -\dot{\mathcal{B}}(x) & \text{is SOS}, \\ & W(x) & \text{is SOS}, \\ & W(x) - \mathcal{B}(x) - 1.0 & \text{is SOS}, \\ & \mathcal{B}(0) \geq 0. & \end{aligned}$$

To make the problem a little numerically better, you'll see in the code that I've asked for $\dot{\mathcal{B}}(x)$ to be strictly negative definite, for $\mathcal{B}(0) \geq 0.1$, and I've chosen to only include even-degree monomials in $\mathcal{B}(x)$ and $W(x)$. Plotting the solution reveals:




As you can see, the superlevel set, $\mathcal{B}(x) \geq 0$ is a tight outer-approximation of the true region of attraction. In the limit of increasing the degrees of the polynomials to infinity, we would expect that $W(\mathbf{x})$ would converge to the indicator function that is one inside the region of attraction, and zero outside (we are quite far from that here, but nevertheless have a tight approximation from $\mathcal{B}(\mathbf{x}) \geq 0$).

Open in Colab

## 9.4 Finite-time Reachability

So far we have used Lyapunov analysis to analyze *stability*, which is fundamentally a description of the system's behavior as time goes to infinity. But Lyapunov analysis can also be used to analyze the finite-time behavior of nonlinear systems. We will see a number of applications of this finite-time analysis over the next few chapters. It can be applied even to unstable systems, or systems that are stable to a limit cycle instead of a fixed-point. It can also be applied to systems that are only defined over a finite interval of time, as might be the case if we are executing a planned trajectory with a bounded duration.

### 9.4.1 Time-varying dynamics and Lyapunov functions

Before focusing on the finite time, let us first realize that the basic (infinite-time) Lyapunov analysis can also be applied to time-varying systems: $\dot{\mathbf{x}} = f(t, \mathbf{x})$. We can analyze the stability (local, regional, or global) of this system with very little change. If we find

$$V(\mathbf{x}) \succ 0,$$
$$\forall t, \forall \mathbf{x} \neq 0, \quad \dot{V}(t, \mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(t, \mathbf{x}) < 0, \quad \dot{V}(t, 0) = 0,$$

then all of our previous statement still hold. In our SOS formulations, $t$ is simply one more indeterminate.

Similarly, even for a time-invariant system, it is also possible to define a time-varying Lyapunov function, $V(t, \mathbf{x})$ and establish local, regional, or global stability using the almost the same conditions:

$$\forall t, \forall \mathbf{x} \neq 0, \quad V(t, \mathbf{x}) > 0, \quad V(t, 0) = 0,$$
$$\forall t, \forall \mathbf{x} \neq 0, \quad \dot{V}(t, \mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}) + \frac{\partial V}{\partial t} < 0, \quad \dot{V}(t, 0) = 0.$$

These two ideas each stand on their own, but they very often go together, as time-varying dynamics are perhaps the best motivator for studying a time-varying Lyapunov function. Afterall, we do know that a stable time-invariant system must have a time-invariant Lyapunov function that demonstrates this stability (from the "converse Lyapunov function" theorems). But we do not know apriori how to represent this function; as an example, remember we know that there are stable polynomial systems that cannot be verified with a polynomial Lyapunov function. For global stability analysis, the time-varying Lyapunov analysis does not add modeling power: since the conditions must be satisfied for all $t$, we could have just set $t$ to a constant and used the time-invariant function. But there may be cases where a time-varying analysis could afford a different analysis for the regional or local stability cases. We will see a good example of this when we study limit cycles.

## 9.4.2 Finite-time reachability

Finite-time reachability analysis is an important concept for control design and verification, where we seek to understand the behavior of a system over only a finite time interval, $[t_1, t_2]$. It is almost always a region-based analysis, and attempts to make a statement of the form:

$$\mathbf{x}(t_1) \in \mathcal{X}_1 \Rightarrow \mathbf{x}(t_2) \in \mathcal{X}_2,$$

where $\mathcal{X}_1, \mathcal{X}_2 \subset \mathbb{R}^n$ are regions of state space. More generally, we might like to understand the time-varying reachable set $\forall t \in [t_1, t_2], \mathcal{X}(t)$.

*Reachability analysis* can be done forward in time: we choose $\mathcal{X}_1$ and try to find the *smallest* region $\mathcal{X}_2$ for which we can make the statement hold. $\mathcal{X}(t)$ would be called the *forward-reachable set* (FRS), and can be very useful for certifying e.g. a motion plan. For instance, you might like to prove that your UAV does not crash into a tree in the next 5 seconds. In this case $\mathcal{X}_1$ might be take to be a point representing the current state of the vehicle, or a region representing an outer-approximation of the current state if the state in uncertain. In this context, we would call $\mathcal{X}_2$ a forward-reachable set. In this use case, we would typically choose any approximations in our Lyapunov analysis to certify that an estimate of the reachable region is also an outer-approximation: $\mathcal{X}_2 \subseteq \hat{\mathcal{X}}_2$.

Reachability analysis can also be done backward in time: we choose $\mathcal{X}_2$ and try to *maximize* the region $\mathcal{X}_1$ for which the statement can be shown to hold. Now $\mathcal{X}(t)$ is called the *backward-reachable set* (BRS), and for robustness we typically try to certify that our estimates are an inner-approximation, $\hat{\mathcal{X}}_1 \subseteq \mathcal{X}_1$. The region-of-attraction analysis we studied above can be viewed as a special case of this, with $\mathcal{X}_2$ taken to be the fixed-point, $t_2 = 0$ and $t_1 = -\infty$. But finite-time BRS also have an important role to play, for instance when we are composing multiple controllers in order to achieve a more complicated task, which we will study soon.

## 9.4.3 Reachability via Lyapunov functions

Lyapunov functions can be used to certify finite-time reachability, even for continuous-time systems. The basic recipe is to certify the Lyapunov conditions over a (potentially time-varying) invariant set. Once again, we typically represent this as

a (time-varying) level set of the Lyapunov function containing the origin, $V(\mathbf{x}) \leq \rho(t)$, where $\rho(t)$ is now a positive scalar function of time. Since we already have time as a decision variable, we can easily accommodate time-varying dynamics and Lyapunov functions, as well, so:

$$\begin{aligned} \forall t \in [t_1, t_2], \quad \forall \mathbf{x}, \qquad & V(t, \mathbf{x}) > 0, \qquad & V(t, 0) = 0 \\ \forall t \in [t_1, t_2], \quad \forall \mathbf{x} \in \{\mathbf{x} | V(t, \mathbf{x}) = \rho(t)\}, \qquad & \dot{V}(t, x) < \dot{\rho}(t). \end{aligned}$$

Note that finite-time reachability is about proving invariance of the set, not stability, and the $\dot{V}$ condition need only be certified at the boundary of the level set. If $V$ is decreasing at the boundary, then trajectories can never leave. One can certainly ask for more -- we may want to show that the system is converging towards $V(t, \mathbf{x}) = 0$, perhaps even at some rate -- but only invariance is required to certify reachability.

Again, for polynomial systems and dynamics, sums-of-squares optimization is a powerful tool for certifying these properties numerically, and optimizing the volume of the estimated regions. Having taken $t$ as an indeterminate, we can use the S-procedure again to certify the conditions $\forall t \in [t_1, t_2]$.

Like in the case for region of attraction, we have many formulations. We can certify an existing Lyapunov candidate, $V(t, \mathbf{x})$, and just try to maximize/minimize $\rho(t)$. Or we can search for the parameters of $V(t, \mathbf{x})$, too. Again, we can initialize that search using the time-varying version of the Lyapunov equation, or the solutions to a time-varying LQR Riccati equation.

In practice, we often certify the Lyapunov conditions over $\mathbf{x}$ at only a finite set of samples $t_i \in [t_1, t_2]$. I don't actually have anything against sampling in one dimension; there are no issues with scaling to higher dimensions, and one can make practical rigorous statement about bounding the sampling error. And in these systems, adding $t$ into all of the equation otherwise can dramatically increase the degree of the polynomials required for the SOS certificates. All of this was written up nicely in [14], and robust variants of it were developed in [15, 8].

## 9.5 Rigid-body dynamics are (rational) polynomial

We've been talking a lot in this chapter about numerical methods for polynomial systems. But even our simple pendulum has a $\sin\theta$ in the dynamics. Have I been wasting your time? Must we just resort to polynomial approximations of the non-polynomial equations? It turns out that our polynomial tools can perform exact analysis of the manipulation equation for almost all†of our robots. We just have to do a little more work to reveal that structure.

†the most notable exception to this is if your robot has helical screw joints (see [16]

Let us first observe that rigid-body kinematics are polynomial (except the helical joint). This is fundamental -- the very nature of a "rigid body" assumption is that Euclidean distance is preserved between points on the body; if $\mathbf{p}_1$ and $\mathbf{p}_2$ are two points on a body, then the kinematics enforce that $|\mathbf{p}_1 - \mathbf{p}_2|_2^2$ is constant -- these are polynomial constraints. Of course, we commonly write the kinematics in a minimal coordinates using $\sin\theta$ and $\cos\theta$. But because of rigid body assumption, these terms only appear in the simplest forms, and we can simply make new variables $s_i = \sin\theta_i, c_i = \cos\theta_i$, and add the constraint that $s_i^2 + c_i^2 = 1$. For a more thorough discussion see, for instance, [16] and [17]. Since the potential energy of a multi-body system is simply an accumulation of weight times the vertical position for all of the points on the body, the potential energy is polynomial.

If configurations (positions) of our robots can be described by polynomials, then velocities can as well: forward kinematics $\mathbf{p}_i = f(\mathbf{q})$ implies that $\dot{\mathbf{p}}_i = \frac{\partial f}{\partial \mathbf{q}} \dot{\mathbf{q}}$, which is polynomial in $s, c, \dot{\theta}$. Since the kinetic energy of our robot is given by the accumulation of the kinetic energy of all the mass, $T = \sum_i \frac{1}{2} m_i v_i^T v_i$, the kinetic energy is polynomial, too (even when we write it with inertial matrices and angular velocities).

Finally, the equations of motion can be obtained by taking derivatives of the Lagrangian (kinetic minus potential). These derivatives are still polynomial!

## Example 9.14 (Global stability of the simple pendulum via SOS)

We opened this chapter using our intuition about energy to discuss stability on the simple pendulum. Now we'll replace that intuition with convex optimization (because it will also work for more difficult systems where our intuition fails).

Let's change coordinates from $[\theta, \dot{\theta}]^T$ to $\mathbf{x} = [s, c, \dot{\theta}]^t$, where $s \equiv \sin\theta$ and $c \equiv \cos\theta$. Then we can write the pendulum dynamics as

$$\dot{\mathbf{x}} = \begin{bmatrix} c\dot{\theta} \\ -s\dot{\theta} \\ -\frac{1}{ml^2}\left(b\dot{\theta} + mgls\right) \end{bmatrix}.$$

Now let's parameterize a Lyapunov candidate $V(s, c, \dot{\theta})$ as the polynomial with unknown coefficients which contains all monomials up to degree 2:

$$V = \alpha_0 + \alpha_1 s + \alpha_2 c + \ldots \alpha_9 s^2 + \alpha_{10} sc + \alpha_{11} s\dot{\theta}.$$

Now we'll formulate the feasibility problem:

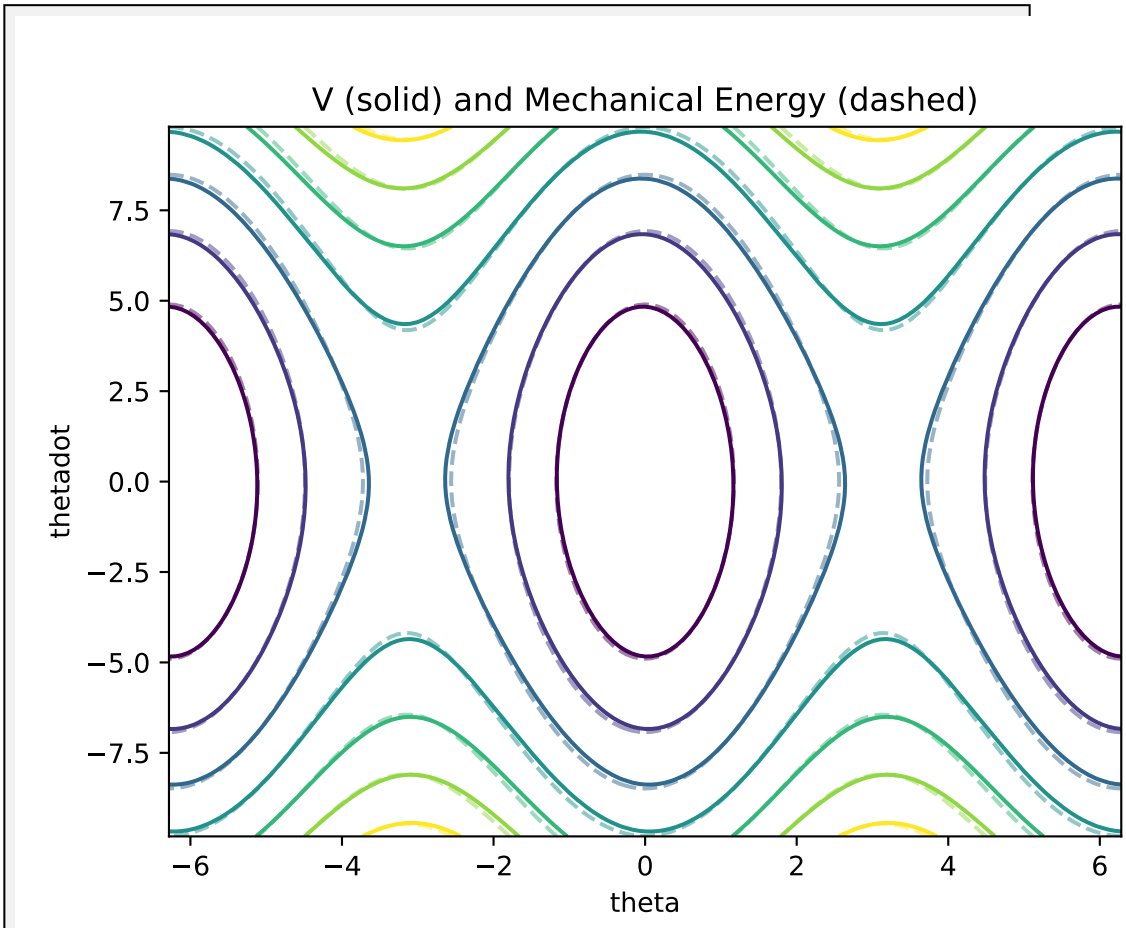
$$\underset{\alpha}{\text{find}} \quad \text{subject to} \quad V \text{ is SOS}, \quad -\dot{V} \text{ is SOS}.$$

In fact, this is asking too much -- really $\dot{V}$ only needs to be negative when $s^2 + c^2 = 1$. We can accomplish this with the S-procedure, and instead write

$$\underset{\alpha,\lambda}{\text{find}} \quad \text{subject to} \quad V \text{ is SOS}, \quad -\dot{V} - \lambda(\mathbf{x})(s^2 + c^2 - 1) \text{ is SOS}.$$

(Recall that $\lambda(\mathbf{x})$ is another polynomial with free coefficients which the optimization can use to make terms arbitrarily more positive when $s^2 + c^2 \neq 1$.) Finally, for style points, in the code example in DRAKE we ask for exponential stability:


As always, make sure that you open up the code and take a look. The result is a Lyapunov function that looks familiar (visualized as a contour plot here):




Aha! Not only does the optimization finds us coefficients for the Lyapunov function which satisfy our Lyapunov conditions, but the result looks a lot like mechanical energy. In fact, the result is a little better than energy... there are some small extra terms added which prove exponential stability without having to invoke LaSalle's Theorem.

The one-degree-of-freedom pendulum did allow us to gloss over one important detail: while the manipulator equations $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \ldots$ are polynomial, in order to solve for $\ddot{\mathbf{q}}$ we actually have to multiply both sides by $\mathbf{M}^{-1}$. This, unfortunately, is *not* a polynomial operation, so in fact the final dynamics of the multibody systems are *rational* polynomial. Not only that, but evaluating $\mathbf{M}^{-1}$ symbolically is not advised -- the equations get very complicated very fast. But we can actually write the Lyapunov conditions using the dynamics in implicit form, e.g. by writing $V(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and asking it to satisfy the Lyapunov conditions everywhere that $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \ldots = \ldots + \mathbf{B}\mathbf{u}$ is satisfied, using the S-procedure.

## Example 9.15 (Verifying dynamics in implicit form)

Typically we write our differential equations in the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. But let us consider for a moment the case where the dynamics are given in the form

$$\mathbf{g}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{x}}) = 0.$$

This form is strictly more general because I can always write $\mathbf{g}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{x}}) = f(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}}$, but importantly here I can also write the bottom rows of $\mathbf{g}$ as $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \tau_g - \mathbf{B}\mathbf{u}$. This form can also represent differential algebraic

equations (DAEs) which are more general than ODEs; $\mathbf{g}$ could even include algebraic constraints like $s_i^2 + c^2 - 1$. Most importantly, for manipulators, $\mathbf{g}$ can be polyonimal, even if $\mathbf{f}$ would have been rational. DRAKE provides access to continuous-time dynamics in implicit form via the `CalcImplicitTimeDerivativesResidual` method.

Interestingly, we can check the Lyapunov conditions, $\dot{V}(\mathbf{x}) \leq 0$, directly on a system (with no inputs) in its implicit form, $\mathbf{g}(\mathbf{x}, \dot{\mathbf{x}}) = 0$. Simply define a new function $Q(\mathbf{x}, \mathbf{z}) = \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\mathbf{z}$. If we can show $Q(\mathbf{x}, \mathbf{z}) \leq 0, \forall \mathbf{x}, \mathbf{z} \in \{\mathbf{x}, \mathbf{z} | \mathbf{g}(\mathbf{x}, \mathbf{z}) = 0\}$ using SOS, then we have verified that $\dot{V}(\mathbf{x}) \leq 0$, albeit at the non-trivial cost of adding indeterminates $\mathbf{z}$ and an additional S-procedure.

There are a few things that *do* break this clean polynomial view of the world. Rotary springs, for instance, if modeled as $\tau = k(\theta_0 - \theta)$ will mean that $\theta$ appears alongside $\sin\theta$ and $\cos\theta$, and the relationship between $\theta$ and $\sin\theta$ is sadly *not polynomial*. Linear feedback from LQR actually looks like the linear spring, although writing the feedback as $u = -\mathbf{K}\sin\theta$ is a viable alternative.

In practice, you can also Taylor approximate any smooth nonlinear function using polynomials. This can be an effective strategy in practice, because you can limit the degrees of the polynomial, and because in many cases it is possible to provide conservative bounds on the errors due to the approximation.

One final technique that is worth knowing about is a change of coordinates, often referred to as the stereographic projection, that provides a coordinate system in which we can replace $\sin$ and $\cos$ with polynomials:

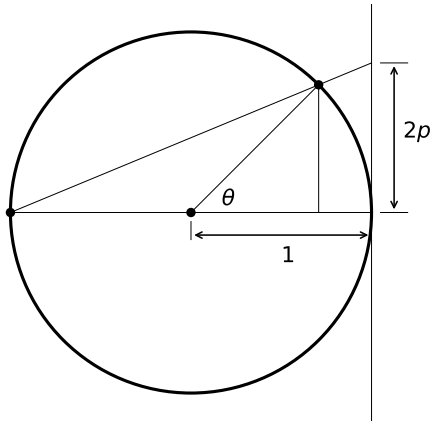



Figure 9.7 - The stereographic projection.

By projecting onto the line, and using similar triangles, we find that $p = \frac{\sin\theta}{1+\cos\theta}$. Solving for $\sin\theta$ and $\cos\theta$ reveals

$$\cos\theta = \frac{1-p^2}{1+p^2}, \quad \sin\theta = \frac{2p}{1+p^2}, \quad \text{and} \quad \frac{\partial p}{\partial \theta} = \frac{1+p^2}{2},$$

where $\frac{\partial p}{\partial \theta}$ can be used in the chain rule to derive the dynamics $\dot{p}$. Although the equations are rational, they share the denominator $1+p^2$ and can be treated efficiently in mass-matrix form. Compared to the simple substitution of $s = \sin\theta$ and $c = \cos\theta$, this is a minimal representation (scalar to scalar, no $s^2 + c^2 = 1$ required); unfortunately it does have a singularity at $\theta = \pi$, so likely cannot be used for global analysis.

## 9.6 CONTROL DESIGN

Throughout this chapter, we've developed some very powerful tools for reasoning about stability of a closed-loop system (with the controller already specified). I hope you've been asking yourself -- can I use these tools to design better controllers? Of course the answer is "yes!". In this section, I'll discuss the control approaches that are the most direct consequences of the convex optimization approaches to Lyapunov functions. Another very natural idea is to use these tools in the content of a feedback motion planning algorithm, which is the subject of an upcoming chapter.

### 9.6.1 State feedback for linear systems

Let's re-examine the Lyapunov conditions for linear systems from Eq. 2, but now add in a linear state feedback $\mathbf{u} = \mathbf{K}\mathbf{x}$, resulting in the closed-loop dynamics $\dot{\mathbf{x}} = (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x}$. One can show that the set of all stabilizing $\mathbf{K}$ can be characterized by the following two matrix inequalities [3]:

$$\mathbf{P} = \mathbf{P}^T \succ 0, \qquad \mathbf{P}(\mathbf{A} + \mathbf{B}\mathbf{K}) + (\mathbf{A} + \mathbf{B}\mathbf{K})^T\mathbf{P} \prec 0.$$

Unfortunately, these inequalities are *bilinear* in the decision variables, $\mathbf{P}$ and $\mathbf{K}$, and therefore non-convex. It turns out that we can turn these into *linear* matrix inequalities (LMIs) through a simple change of coordinates, $\mathbf{Q} = \mathbf{P}^{-1}, \mathbf{Y} = \mathbf{K}\mathbf{P}^{-1}$, resulting in

$$\mathbf{Q} = \mathbf{Q}^T \succ 0, \qquad \mathbf{A}\mathbf{Q} + \mathbf{Q}\mathbf{A}^T + \mathbf{B}\mathbf{Y} + \mathbf{Y}^T\mathbf{B}^T \prec 0.$$

Furthermore, given matrices $\mathbf{A}$ and $\mathbf{B}$, there exists a matrix $\mathbf{K}$ such that $(\mathbf{A} + \mathbf{B}\mathbf{K})$ is stable if and only if there exist matrices $\mathbf{Q}$ and $\mathbf{Y}$ which satisfy this (strict) linear matrix inequality.

### 9.6.2 Control design via alternations

For control design using convex optimization, we will lean heavily on the assumption of the dynamics being control-affine. Let me write it this time as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{i=0}^{m-1} u_i \mathbf{f}_i(x).$$

As we have discussed, for mechanical systems this assumption is perfectly reasonable.

For linear optimal control, we found controllers of the form, $\mathbf{u} = -\mathbf{K}\mathbf{x}$. The natural generalization of this to polynomial analysis will be to look for controllers of the form $\mathbf{u} = \pi(\mathbf{x})$, where $\pi(\mathbf{x})$ is a polynomial. (For mechanical systems like the pendulum above, we might make $\pi$ a polynomial in $s$ and $c$.)

If we apply this control to the Lyapunov conditions (for global analysis), we quickly see the problem. We have

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} \left[ \mathbf{f}(x) + \sum_{i=0}^{m-1} \mathbf{f}_i(\mathbf{x})\pi_i(\mathbf{x}) \right],$$

and as a result if we are trying to search for the parameters of $V$ and $\pi$ simultaneously, the decision variables multiply and the problem will be non-convex.

One very natural strategy is to use alternations. The idea is simple, we will fix $\pi$ and optimize $V$, then fix $V$ and optimize $\pi$ and repeat until convergence. This approach has roots in the famous "DK iterations" for robust control (e.g. [18]). It takes advantage of the structured convex optimization at each step.

For this approach, it is important to start with an initial feasible $V$ or $\pi$. For example, one can think of locally stabilizing a nonlinear system with LQR, and then searching for a linear control law (or even a higher-order polynomial control law) that achieves a larger basin of attraction. But note that once we move from global stability to region of attraction optimization, we now need to alternate between three sets of variables: $V(\mathbf{x}), \pi(\mathbf{x}), \lambda(\mathbf{x})$, where $\lambda(]bx)$ was the multiplier polynomial for the S-procedure. We took exactly this approach in [19]. In that paper, we showed that alternations could increase the certified region of attraction for the Acrobot.

The alternation approach takes advantage of convex optimization in the inner loop, but it is still only a local approach to solving the nonconvex joint optimization. It is subject to local minima. The primary advantage is that, barring numerical issues, we expect recursive feasibility (once we have a controller and Lyapunov function that achieves stability, even with a small region of attraction, we will not lose it) and monotonic improvement on the objective. It is also possible to attempt to optimize these objectives more directly with other non-convex optimization procedures (like stochastic gradient descent, or sequential quadratic programming) (e.g. [6]), but strict feasibility is harder to guarantee. Often times the Lyapunov conditions are just evaluated at samples, or along sample trajectories; we can still certify the performance using just a single SOS verification step the with controller fixed before deplying.

## 9.6.3 Control-Lyapunov Functions

Another core idea connecting Lyapunov analysis with control design is the "control-Lyapunov function". Given a system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, a control-Lyapunov function $V$ is a positive-definite function for which

$$\forall \mathbf{x} \neq 0, \exists \mathbf{u} \quad \dot{V}(\mathbf{x}, \mathbf{u}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) < 0 \quad \text{and} \quad \exists \mathbf{u} \quad \dot{V}(0, \mathbf{u}) = 0.$$

In words, for all $\mathbf{x}$, there exists a control that would allow the $V$ to decrease. Once again, we adorn this basic property with extra conditions (e.g. radially unbounded, or satisfied over a control- invariant set) in order to construct global or regional stability statements. What is important to understand is that we can design control-Lyapunov functions without explicitly parameterizing the controller; often the control action is not even determined until execution time by finding a particular $\mathbf{u}$ that goes downhill.

Our sums-of-squares toolkit is well-suited for addressing questions with the $\forall$ quantifier over indeterminates. Working with the $\exists$ quantifier is much more difficult; we don't have an S-procedure-like solution for it. Interestingly, there is one approach that we've discussed above that effectively turns the $\exists$ into a $\forall$ -- that is the outer approximation approach to region of attraction analysis.

In the outer-approximation, we produce a barrier certificate to find the set of states where the controller cannot go (for any $\mathbf{u}$). Our barrier certificate now has the form

$$\forall x, \forall u, \quad \dot{\mathcal{B}}(\mathbf{x}, \mathbf{u}) \leq 0.$$

Certainly, $\mathcal{B}(\mathbf{x}) \geq 0$ is now an outer-approximation of the true "backward-reachable set" (BRS) of the fixed point [9, 20]. Again [11] has some nice examples of this written directly in sums-of-squares form.

Now here is where it gets a little frustrating. Certainly, sublevel sets of $\mathcal{B}(\mathbf{x})$ are control-invariant (via the proper choice of $\mathbf{u}$). But we do not (cannot) expect that the entire estimated region (the 0-superlevel set) can be rendered invariant. The estimated region is an *outer* approximation of the backward reachable set. [21] gave a recipe for extracting a polynomial control law from the BRS; an inner approximation of this control law can be obtained via the original SOS region of attraction tools. This is unfortunatley suboptimal/conservative. Although we would like to certify the control-Lyapunov conditions directly in an inner approximation, the $\exists$ quantifier remains as an obstacle.

## 9.6.4 Approximate dynamic programming with SOS

We have already established the most important connection between the HJB conditions and the Lyapunov conditions:

$$\dot{J}^*(\mathbf{x}) = -\ell(\mathbf{x}, \mathbf{u}^*) \qquad \text{vs} \qquad \dot{V}(\mathbf{x}) \preceq 0.$$

The HJB involves solving a complex PDE; by changing this to an inequality, we relax the problem and make it amenable to convex optimization.

## Upper and lower bounds on cost-to-go

Asking for $\dot{V}(\mathbf{x}) \preceq 0$ is sufficient for proving stability. But we can also use this idea to provide rigorous certificates as upper or lower bounds of the cost-to-go. Given a control-dynamical system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, and a fixed controller $\pi(\mathbf{x})$ we can find a function $V(\mathbf{x})$:

- $\forall \mathbf{x}, \dot{V}^\pi(\mathbf{x}) \leq -\ell(\mathbf{x}, \pi(\mathbf{x}))$ to provide an **upper bound**, or
- $\forall \mathbf{x}, \dot{V}^\pi(\mathbf{x}) \geq -\ell(\mathbf{x}, \pi(\mathbf{x}))$ to provide an **lower bound**.

To see this, take the integral of both sides along any *solution* trajectory, $\mathbf{x}(t), \mathbf{u}(t)$. For the upper-bound we get

$$\int_0^\infty \dot{V}^\pi(\mathbf{x}) dt = V^\pi(\mathbf{x}(\infty)) - V^\pi(\mathbf{x}(0)) \leq \int_0^\infty -\ell(\mathbf{x}(t), \pi(\mathbf{x}(t))) dt.$$

Assuming $V^\pi(\mathbf{x}(\infty)) = 0$, we have

$$V^\pi(\mathbf{x}(0)) \geq \int_0^\infty \ell(\mathbf{x}(t), \pi(\mathbf{x}(t))) dt.$$

The upper bound is the one that we would want to use in a certification procedure -- it provides a *guarantee* that the total cost achieved by the system started in $\mathbf{x}$ is less than $V(\mathbf{x})$. But it turns out that the lower bound is much better for control design. This is because we can write

$$\forall \mathbf{x}, \min_{\mathbf{u}} [\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u})] \geq 0 \quad \equiv \quad \forall \mathbf{x}, \forall \mathbf{u}, \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \geq 0.$$

Therefore, without having to specify apriori a controller, if we can find a function $V(\mathbf{x})$ such that $\forall \mathbf{x}, \forall \mathbf{u}, \dot{V}(\mathbf{x}, \mathbf{u}) \geq -\ell(\mathbf{x}, \mathbf{u})$, then we have a lower-bound on the *optimal* cost-to-go.

You should take a minute to convince yourself that, unfortunately, the same trick does not work for the upper-bound. Again, we would need $\exists$ as the quantifier on $\mathbf{u}$ instead of $\forall$.

## Sums-of-squares formulation

Combining a number of ideas we've seen already, this leads to a natural sums-of-squares formulation for optimal control:

$$\begin{aligned} \max_{V(\mathbf{x})} \quad & \int_{\mathbf{X}} V(\mathbf{x}) d\mathbf{x}, \\ \text{subject to} \quad & \ell(\mathbf{x}, \mathbf{u}) + \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \qquad \text{is SOS}, \\ & V(0) = 0. \end{aligned}$$

The SOS constraint enforces the lower bound, and the objective "pushes up" on the lower-bound by maximizing the integral over some compact region. Once again, we can then try to extract a control law by either using this lower bound as a control-Lyapunov function, or by extracting (and certifying) a polynomial controller.

Perhaps you noticed that this is a natural extension of the linear programming approach to dynamic programming. For systems with continuous state, the LP approach verified the inequality conditions only at sample points; here we verify them for all $\mathbf{x}, \mathbf{u}$. This is an important generalization: not so much because it can certify better (the lower bound is not a particularly valuable thing to certify), but because it can scale to dimensions were dense sampling is not viable. This provides something of a spectrum between the mesh-based value iteration and the very scalable LQR.

The biggest challenge to this approach, however, is not the number of dimensions, but the degree of the polynomial required to achieve a meaningful approximation. Remember that the optimal cost-to-go for even the min-time double integrator problem is non-smooth. Like in the pseudo-spectral methods that we will see in the context of trajectory optimization; choosing the right polynomial basis can make a huge difference for the numerical robustness of this approach.

**Example 9.16 (SOS ADP for the cubic polynomial)**

**Example 9.17 (SOS ADP for the pendulum swing-up problem)**

## 9.7 Alternative computational approaches

### 9.7.1 "Satisfiability modulo theories" (SMT)

Satisiability modulo theories (SMT). dReal is available in Drake.

### 9.7.2 Mixed-integer programming (MIP) formulations

### 9.7.3 Continuation methods

## 9.8 Neural Lyapunov functions

## 9.9 Contraction metrics

## 9.10 Exercises

### Exercise 9.1 (Valid Lyapunov Function for Global Stability)

For the system

$$\dot{x}_1 = -\frac{6x_1}{(1+x_1^2)^2} + 2x_2,$$
$$\dot{x}_2 = -\frac{2(x_1+x_2)}{(1+x_1^2)^2},$$

you are given the positive definite function $V(\mathbf{x}) = \frac{x_1^2}{1+x_1^2} + x_2^2$ (plotted here) and told that, for this system, $\dot{V}(\mathbf{x})$ is negative definite over the entire space. Is $V(\mathbf{x})$ a valid Lyapunov function to prove global asymptotic stability of the origin for the

system described by the equations above? Motivate your answer.

## Exercise 9.2 (Invariant Sets and Regions of Attraction)

You are given a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$, with $f$ continuous, which has a fixed point at the origin. Let $B_r$ be a ball of (finite) radius $r > 0$ centered at the origin: $B_r = \{\mathbf{x} : \|\mathbf{x}\| \leq r\}$. Assume you found a continuously-differentiable scalar function $V(\mathbf{x})$ such that: $V(0) = 0$, $V(\mathbf{x}) > 0$ for all $\mathbf{x} \neq 0$ in $B_r$, and $\dot{V}(\mathbf{x}) < 0$ for all $\mathbf{x} \neq 0$ in $B_r$. Determine whether the following statements are true or false. Briefly justify your answer.

a. $B_r$ is an invariant set for the given system, i.e.: if the initial state $\mathbf{x}(0)$ lies in $B_r$, then $\mathbf{x}(t)$ will belong to $B_r$ for all $t \geq 0$.
b. $B_r$ is a subset of the ROA of the fixed point $\mathbf{x} = 0$, i.e.: if $\mathbf{x}(0)$ lies in $B_r$, then $\lim_{t\to\infty} \mathbf{x}(t) = 0$.

## Exercise 9.3 (Are Lyapunov Functions Unique?)

If $V_1(\mathbf{x})$ and $V_2(\mathbf{x})$ are valid Lyapunov functions that prove global asymptotic stability of the origin, does $V_1(\mathbf{x})$ necessarily equal $V_2(\mathbf{x})$?

## Exercise 9.4 (Proving Global Asymptotic Stability)

Consider the system given by

$$\begin{aligned}\dot{x}_1 &= x_2 - x_1^3,\\ \dot{x}_2 &= -x_1 - x_2^3.\end{aligned}$$

Show that the Lyapunov function $V(\mathbf{x}) = x_1^2 + x_2^2$ proves global asymptotic stability of the origin for this system.

## Exercise 9.5 (Gradient Flow in Euclidean Space)

We can use Lyapunov analysis to analyze the behavior of optimization algorithms like gradient descent. Consider an objective function $\ell : \mathbf{R}^n \to \mathbf{R}$, and we want to minimize $\ell(x)$. The gradient flow is a continuous-time analog of gradient descent and is defined as $\dot{x} = -\frac{\partial \ell}{\partial x}$.

a. Show that the objective function $\ell(x) - \ell(x^*)$ is a Lyapunov function of the gradient flow where $x^*$ is a unique minimizer.
b. We can use Lyapunov to argue that an optimization problem will converge to a global optimum, even if it is non-convex. Suppose that the Lyapunov function $\ell$, has negative definite $\dot{\ell}$. Show that the objective function $\ell$ has a unique minimizer at the origin.
c. Consider the objective function in the figure below. Could this be a valid Lyapunov function for showing global asymptotic stability?

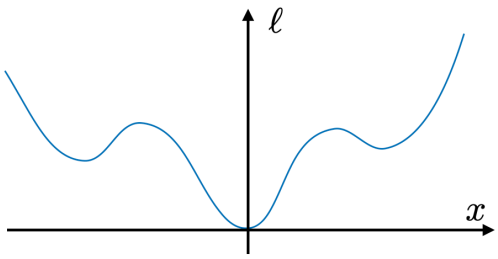



Figure 9.8 - Lyapunov function candidate for asymptotic stability.

d. Consider the objective function $\ell(x) = x_1^4 - 8x_1^3 + 18x_1^2 + x_2^2$ with $x \in \mathbf{R}^2$. Find the largest $\rho$ such that $\{x \mid \ell(x) < \rho\}$ is a valid region of attraction for the origin.

## Exercise 9.6 (Control-Lyapunov Function for a Wheeled Robot)

In this exercise, we examine the idea of a control-Lyapunov function to drive a wheeled robot, implementing the controller proposed in [22].

Similar to this previous example, we use a kinematic model of the robot. We represent with $z_1$ and $z_2$ its Cartesian position and with $z_3$ its orientation. The controls are the linear $u_1$ and angular $u_2$ velocities. The equations of motion read

$$\begin{aligned} \dot{z}_1 &= u_1 \cos z_3, \\ \dot{z}_2 &= u_1 \sin z_3, \\ \dot{z}_3 &= u_2. \end{aligned}$$

The goal is to design a feedback law $\pi(\mathbf{z})$ that drives the robot to the origin $\mathbf{z} = 0$ from any initial condition. As pointed out in [22], this problem becomes dramatically easier if we analyze it in polar coordinates. As depicted below, we let $x_1$ be the radial and $x_2$ the angular coordinate of the robot, and we define $x_3 = x_2 - z_3$. Analyzing the figure, basic kinematic considerations lead to

$$\begin{aligned} \dot{x}_1 &= u_1 \cos x_3, \\ \dot{x}_2 &= -\frac{u_1 \sin x_3}{x_1}, \\ \dot{x}_3 &= -\frac{u_1 \sin x_3}{x_1} - u_2. \end{aligned}$$

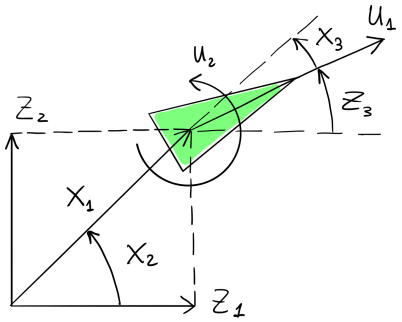



Figure 9.9 - Wheeled robot with Cartesian $\mathbf{z}$ and polar $\mathbf{x}$ coordinate system.

a. For the candidate Lyapunov function $V(\mathbf{x}) = V_1(x_1) + V_2(x_2, x_3)$, with $V_1(x_1) = \frac{1}{2}x_1^2$ and $V_2(x_2, x_3) = \frac{1}{2}(x_2^2 + x_3^2)$, compute the time derivatives $\dot{V}_1(\mathbf{x}, u_1)$ and $\dot{V}_2(\mathbf{x}, \mathbf{u})$.

b. Show that the choice

$$u_1 = \pi_1(\mathbf{x}) = -x_1 \cos x_3,$$
$$u_2 = \pi_2(\mathbf{x}) = x_3 + \frac{(x_2 + x_3)\cos x_3 \sin x_3}{x_3},$$

makes $\dot{V}_1(\mathbf{x}, \pi_1(\mathbf{x})) \leq 0$ and $\dot{V}_2(\mathbf{x}, \pi(\mathbf{x})) \leq 0$ for all $\mathbf{x}$. (Technically speaking, $\pi_2(\mathbf{x})$ is not defined for $x_3 = 0$. In this case, we let $\pi_2(\mathbf{x})$ assume its limiting value $x_2 + 2x_3$, ensuring continuity of the feedback law.)

c. Explain why Lyapunov's direct method does not allow us to establish asymptotic stability of the closed-loop system.

d. Substitute the control law $\mathbf{u} = \pi(\mathbf{x})$ in the equations of motion, and derive the closed-loop dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \pi(\mathbf{x}))$. Use LaSalle's theorem to show (global) asymptotic stability of the closed-loop system.

e. In this python notebook we set up a simulation environment for you to try the controller we just derived. Type the control law from point (b) in the dedicated cell, and use the notebook plot to check your work.

## Exercise 9.7 (Limitations of SOS Polynomials in Lyapunov Analysis)

a. Are there positive definite functions that are not representable as sums of squares?

b. If a fixed point of our dynamical system does not admit a SOS Lyapunov function, what can we conclude about its stability?

## Exercise 9.8 (ROA Estimation for the Time-Reversed Van der Pol Oscillator)

In this exercise you will use SOS optimization to approximate the ROA of the time-reversed Van der Pol oscillator (a variation of the classical Van der Pol oscillator which evolves backwards in time). In this python notebook, you are asked to test the following SOS formulations.

a. The one from the example above, augmented with a line search that maximizes the area of the ROA.

b. A single-shot SOS program that can directly maximize the area of the ROA, without any line search.

c. An improved version of the previous, where less SOS constraints are imposed in the optimization problem.

## References


1. Jean-Jacques E. Slotine and Weiping Li, "Applied Nonlinear Control",Prentice Hall , October, 1990.

2. Hassan K. Khalil, "Nonlinear Systems",Prentice Hall , December, 2001.

3. S. Boyd and L. El Ghaoui and E. Feron and V. Balakrishnan, "Linear Matrix Inequalities in System and Control Theory",SIAM , 1994.

4. Pablo A. Parrilo, "Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization", PhD thesis, California Institute of Technology, May 18, 2000.

5. Amir Ali Ahmadi and Miroslav Krstic and Pablo A. Parrilo, "A Globally Asymptotically Stable Polynomial Vector Field with no Polynomial Lyapunov Function", *Proceedings of the Conference on Decision and Control*, 2011.

6. Shen Shen and Russ Tedrake, "Sampling Quotient-Ring Sum-of-Squares Programs for Scalable Verification of Nonlinear Systems", *Proceedings of the 2020 59th IEEE Conference on Decision and Control (CDC)*, 2020. [ link ]

7. Russ Tedrake and Ian R. Manchester and Mark M. Tobenkin and John W. Roberts, "{LQR-Trees}: Feedback Motion Planning via Sums of Squares Verification", *International Journal of Robotics Research*, vol. 29, pp. 1038--1052, July, 2010. [ link ]

8. Anirudha Majumdar, "Funnel Libraries for Real-Time Robust Feedback Motion Planning", PhD thesis, Massachusetts Institute of Technology, Jun, 2016. [ link ]

9. Didier Henrion and Milan Korda, "Convex computation of the region of attraction of polynomial control systems", *IEEE Transactions on Automatic Control*, vol. 59, no. 2, pp. 297-312, 2014.

10. Jean Bernard Lasserre, "Moments, Positive Polynomials and Their Applications",World Scientific , vol. 1, 2010.

11. Michael Posa and Twan Koolen and Russ Tedrake, "Balancing and Step Recovery Capturability via Sums-of-Squares Optimization", *Robotics: Science and Systems*, 2017. [ link ]

12. Gerald B Folland, "How to integrate a polynomial over a sphere", *The American Mathematical Monthly*, vol. 108, no. 5, pp. 446--448, 2001.

13. D. Henrion and J.B. Lasserre and C. Savorgnan, "Approximate volume and integration for basic semialgebraic sets", *SIAM Review*, vol. 51, no. 4, pp. 722--743, 2009.

14. Mark M. Tobenkin and Ian R. Manchester and Russ Tedrake, "Invariant Funnels around Trajectories using Sum-of-Squares Programming", *Proceedings of the 18th IFAC World Congress, extended version available online: arXiv:1010.3013 [math.DS]*, 2011. [ link ]

15. Anirudha Majumdar, "Robust Online Motion Planning with Reachable Sets", , May, 2013. [ link ]

16. Charles W. Wampler and Andrew J. Sommese, "Numerical algebraic geometry and algebraic kinematics", *Acta Numerica*, vol. 20, pp. 469-567, 2011.

17. A.J. Sommese and C.W. Wampler, "The Numerical solution of systems of polynomials arising in engineering and science",World Scientific Pub Co Inc , 2005.

18. R. Lind and G.J. Balas and A. Packard, "Evaluating {D-K} iteration for control design", *American Control Conference, 1994*, vol. 3, pp. 2792 - 2797 vol.3, 29 June-1 July, 1994.

19. Anirudha Majumdar and Amir Ali Ahmadi and Russ Tedrake, "Control Design along Trajectories with Sums of Squares Programming", *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4054-4061, 2013. [ link ]

20. Milan Korda and Didier Henrion and Colin N Jones, "Controller design and region of attraction estimation for nonlinear dynamical systems", *The 19th World Congress of the International Federation of Automatic Control*

*(IFAC)*, 2014.

21. Anirudha Majumdar and Ram Vasudevan and Mark M. Tobenkin and Russ Tedrake, "Convex Optimization of Nonlinear Feedback Controllers via Occupation Measures", *Proceedings of Robotics: Science and Systems (RSS)*, 2013. [ link ]

22. Michele Aicardi and Giuseppe Casalino and Antonio Bicchi and Aldo Balestrino, "Closed loop steering of unicycle like vehicles via Lyapunov techniques", *IEEE Robotics \& Automation Magazine*, vol. 2, no. 1, pp. 27--35, 1995.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 10
# Trajectory Optimization


I've argued that optimal control is a powerful framework for specifying complex behaviors with simple objective functions, letting the dynamics and constraints on the system shape the resulting feedback controller (and vice versa!). But the computational tools that we've provided so far have been limited in some important ways. The numerical approaches to dynamic programming which involve putting a mesh over the state space do not scale well to systems with state dimension more than four or five. Linearization around a nominal operating point (or trajectory) allowed us to solve for locally optimal control policies (e.g. using LQR) for even very high-dimensional systems, but the effectiveness of the resulting controllers is limited to the region of state space where the linearization is a good approximation of the nonlinear dynamics. The computational tools for Lyapunov analysis from the last chapter can provide, among other things, an effective way to compute estimates of those regions. But we have not yet provided any real computational tools for approximate optimal control that work for high-dimensional systems beyond the linearization around a goal. That is precisely the goal for this chapter.

In order to scale to high-dimensional systems, we are going to formulate a simpler version of the optimization problem. Rather than trying to solve for the optimal feedback controller for the entire state space, in this chapter we will instead attempt to find an optimal control solution that is valid from only a single initial condition. Instead of representing this as a feedback control function, we can represent this solution as a *trajectory*, $\mathbf{x}(t), \mathbf{u}(t)$, typically defined over a finite interval.

## 10.1 PROBLEM FORMULATION

Given an initial condition, $\mathbf{x}_0$, and an input trajectory $\mathbf{u}(t)$ defined over a finite interval, $t \in [t_0, t_f]$, we can compute the long-term (finite-horizon) cost of executing that trajectory using the standard additive-cost optimal control objective,

$$J_{\mathbf{u}(\cdot)}(\mathbf{x}_0) = \ell_f(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \ell(\mathbf{x}(t), \mathbf{u}(t)) dt.$$

We will write the trajectory optimization problem as

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \ell_f(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \ell(\mathbf{x}(t), \mathbf{u}(t)) dt \\ \text{subject to} \quad & \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [t_0, t_f] \\ & \mathbf{x}(t_0) = \mathbf{x}_0. \end{aligned}$$

Some trajectory optimization problems may also include additional constraints, such as collision avoidance (e.g., where the constraint is that the signed distance between the robot's geometry and the obstacles stays positive) or input limits (e.g. $\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}$ ), which can be defined for all time or some subset of the trajectory.

As written, the optimization above is an optimization over continuous trajectories. In order to formulate this as a numerical optimization, we must parameterize it with a finite set of numbers. Perhaps not surprisingly, there are many different ways to write down this parameterization, with a variety of different properties in terms of speed, robustness, and accuracy of the results. We will outline just a few of the most popular below; I would recommend [1, 2] for additional details.

It is worth contrasting this parameterization problem with the one that we faced in our continuous-dynamic programming algorithms. For trajectory optimization, we need a finite-dimensional parameterization in only one dimension (time), whereas in the mesh-based value iteration algorithms we had to work in the dimension of the state space. Our mesh-based discretizations scaled badly with this state dimension, and led to numerical errors that were difficult to deal with. There is relatively much more known about discretizing solutions to differential equations over time, including work on error-controlled integration. And the number of parameters required for trajectory parameterizations scales linearly with the state dimension, instead of exponentially in mesh-based value iteration.

## 10.2 Convex Formulations for Linear Systems

Let us first consider the case of linear systems. In fact, if we start in discrete time, we can even defer the question of how to best discretize the continuous-time problem. There are a few different ways that we might "transcribe" this optimization problem into a concrete mathematical program.

### 10.2.1 Direct Transcription

For instance, let us start by writing both $\mathbf{u}[\cdot]$ and $\mathbf{x}[\cdot]$ as decision variables. Then we can write:

$$\begin{aligned} \min_{\mathbf{x}[\cdot], \mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n], \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints}. \end{aligned}$$

We call this modeling choice -- adding $\mathbf{x}[\cdot]$ as decision variables and modeling the discrete dynamics as explicit constraints -- the "*direct transcription*". Importantly, for linear systems, the dynamics constraints are linear constraints in these decision variables. As a result, if we can restrict our additional constraints to linear inequality constraints and our objective function to being linear/quadratic in $\mathbf{x}$ and $\mathbf{u}$, then the resulting trajectory optimization is a convex optimization (specifically a linear program or quadratic program depending on the objective). As a result, we can reliably solve these problems to global optimality at quite large scale; these days it is common to solve these optimization online inside a high-rate feedback controller.

### Example 10.1 (Trajectory optimization for the Double Integrator)

We've looked at a few optimal control problems for the double integrator using value iteration. For one of them -- the quadratic objective with no constraints on $\mathbf{u}$ -- we know now that we could have solved the problem "exactly" using LQR. But we have not yet given satisfying numerical solutions for the minimum-time problem, nor for the constrained LQR problem.

In the trajectory formulation, we can solve these problems exactly for the discrete-time double integrator, and with better accuracy for the continuous-time double integrator. Take a moment to appreciate that! The bang-bang policy and cost-to-go functions are fairly nontrivial functions of state; it's quite satisfying that we can evaluate them using convex optimization! The limitation, of course, is that we are only solving them for one initial condition at a time.


If you have not studied convex optimization before, you might be surprised by the modeling power of even of this framework. Consider, for instance, an objective of the form

$$\ell(\mathbf{x}, \mathbf{u}) = |\mathbf{x}| + |\mathbf{u}|.$$

This can be formulated as a linear program. To do it, add additional decision variables $\mathbf{s}_x[\cdot]$ and $\mathbf{s}_u[\cdot]$ -- these are commonly referred to as *slack variables* -- and write

$$\min_{\mathbf{x},\mathbf{u},\mathbf{s}_x,\mathbf{s}_u} \sum_{n}^{N-1} \mathbf{s}_x[n] + \mathbf{s}_u[n], \quad \text{s.t.} \quad \mathbf{s}_x[n] \geq x[n], \quad \mathbf{s}_x[n] \geq -x[n], \quad \ldots$$

The field of convex optimization is replete with tricks like this. Knowing and recognizing them are skills of the (optimization) trade. But there are also many relevant constraints which cannot be recast into convex constraints (in the original coordinates) with any amount of skill. An important example is obstacle avoidance. Imagine a vehicle that must decide if it should go left or right around an obstacle. This represents a fundamentally non-convex constraint in $\mathbf{x}$; we'll discuss the implications of using non-convex optimization for trajectory optimization below.

## 10.2.2 Direct Shooting

The savvy reader might have noticed that adding $\mathbf{x}[\cdot]$ as decision variables was not strictly necessary. If we know $\mathbf{x}[0]$ and we know $\mathbf{u}[\cdot]$, then we should be able to solve for $\mathbf{x}[n]$ using forward simulation. For our discrete-time linear systems, this is particularly nice:

$$\begin{aligned}\mathbf{x}[1] =& \mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0]\\ \mathbf{x}[2] =& \mathbf{A}(\mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0]) + \mathbf{B}\mathbf{u}[1]\\ \mathbf{x}[n] =& \mathbf{A}^n\mathbf{x}[0] + \sum_{k=0}^{n-1} \mathbf{A}^{n-1-k}\mathbf{B}u[k].\end{aligned}$$

What's more, the solution is still linear in $\mathbf{u}[\cdot]$. This is amazing... we can get rid of a bunch of decision variables, and turn a constrained optimization problem into an unconstrained optimization problem (assuming we don't have any other constraints). This approach -- using $\mathbf{u}[\cdot]$ but *not* $\mathbf{x}[\cdot]$ as decision variables and using forward simulation to obtain $\mathbf{x}[n]$ -- is called the *direct shooting* transcription. For linear systems with linear/quadratic objectives in $\mathbf{x}$, and $\mathbf{u}$, it is still a convex optimization, and has less decision variables and constraints than the direct transcription.

### 10.2.3 Computational Considerations

So is direct shooting uniformly better than the direct transcription approach? I think it is not. There are a few potential reason that one might prefer the direct transcription:

- Numerical conditioning. Shooting involves calculating $\mathbf{A}^n$ for potentially large $n$, which can lead to a large range of coefficient values in the constraints. This problem (sometimes referred to as the "tail wagging the dog") is somewhat fundamental in trajectory optimization: the control input $\mathbf{u}[0]$ really does have more opportunity to have a large impact on the total cost than control input $\mathbf{u}[N-1]$. But the direct transcription approach combats the numerical issue by spreading this effect out over a large number of well-balanced constraints.
- Adding state constraints. Having $\mathbf{x}[n]$ as explicit decision variables makes it very easy/natural to add additional state constraints; and the solver effectively reuses the computation of $\mathbf{A}^n$ for each constraint. In shooting, one has to unroll those terms for each new constraint.
- Parallelization. For larger problems, evaluating the constraints can be a substantial cost. In direct transcription, one can evaluate the dynamics/ constraints in parallel (because each iteration begins with $\mathbf{x}[n]$ already given), whereas shooting is more fundamentally a serial operation.

For linear convex problems, the solvers are mature enough that these differences often don't amount to much. For nonlinear optimization problems, the differences can be substantial. If you look at trajectory optimization papers in mainstream robotics, you will see that both direct transcription and direct shooting approaches are used. (It's possible you could guess which research lab wrote the paper simply by the transcription they use!)

It is also worth noting that the problems generated by the direct transcription have an important and exploitable "banded" sparsity pattern -- most of the constraints touch only a small number of variables. This is actually the same pattern that we exploit in the Riccati equations. Thanks to the importance of these methods in real applications, numerous specialized solvers have been written to explicitly exploit this sparsity (e.g. [3]).

### 10.2.4 Continuous Time

If we wish to solve the continuous-time version of the problem, then we can discretize time and use the formulations above. The most important decision is the discretization / numerical integration scheme. For linear systems, if we assume that the control inputs are held constant for each time step (aka zero-order hold), then we can integrate the dynamics perfectly:

$$\mathbf{x}[n+1] = \mathbf{x}[n] + \int_{t_n}^{t_n+h} [\mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}]dt = e^{\mathbf{A}h}\mathbf{x}[n] + \mathbf{A}^{-1}(e^{\mathbf{A}h} - \mathbf{I})\mathbf{B}\mathbf{u}[n],$$

is the simple case (when $\mathbf{A}$ is invertible). But in general, we can use any finitely-parameterized representation of $\mathbf{u}(t)$ and any numerical integration scheme to obtain $\mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n])$.

## 10.3 Nonconvex Trajectory Optimization

I strongly recommend that you study the convex trajectory optimization case; it can lead you to mental clarity and sense of purpose. But in practice trajectory optimization is often used to solve nonconvex problems. Our formulation can become nonconvex for a number of reasons. For example, if the dynamics are nonlinear, then the dynamic constraints become nonconvex. You may also wish to have a nonconvex objective or nonconvex additional constraint (e.g. collision avoidance). Typically we

formulate these problems using tools from nonlinear programming.

### 10.3.1 Direct Transcription and Direct Shooting

The formulations that we wrote for direct transcription and direct shooting above are still valid when the dynamics are nonlinear, it's just that the resulting problem is nonconvex. For instance, the direct transcription for discrete-time systems becomes the more general:

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n]), \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints.} \end{aligned}$$

Direct shooting still works, too, since on each iteration of the algorithm we can compute $\mathbf{x}[n]$ given $\mathbf{x}[0]$ and $\mathbf{u}[\cdot]$ by forward simulation. But things get a bit more interesting when we consider continuous-time systems.

For nonlinear dynamics, we have many choices for how to approximate the discrete dynamics

$$\mathbf{x}[n+1] = \mathbf{x}[n] + \int_{t[n]}^{t[n+1]} f(\mathbf{x}(t), \mathbf{u}(t)) dt, \quad \mathbf{x}(t[n]) = \mathbf{x}[n].$$

For instance, in DRAKE we have an entire suite of numerical integrators that achieve different levels of simulation speed and/or accuracy, both of which can be highly dependent on the details of $\mathbf{f}(\mathbf{x}, \mathbf{u})$.

One very important idea in numerical integration of differential equations is the use of variable-step integration as a means for controlling integration error. Runge-Kutta-Fehlberg, also known as "RK45", is one of the most famous variable-step integrators. We typically avoid using variable steps inside a constraint (it can lead to discontinuous gradients), but it is possible to accomplish something similar in trajectory optimization by allowing the sample times, $t[\cdot]$, themselves to be decision variables. This allows the optimizer to stretch or shrink the time intervals in order to solve the problem, and is particularly useful if you do not know apriori what the total duration of the trajectory should be. Adding some constraints to these time variables is essential in order to avoid trivial solutions (like collapsing to a trajectory of zero duration). One could potentially even add constraints to bound the integration error.

### 10.3.2 Direct Collocation

It is very satisfying to have a suite of numerical integration routines available for our direct transcription. But numerical integrators are designed to solve forward in time, and this represents a design constraint that we don't actually have in our direct transcription formulation. If our goal is to obtain an accurate solution to the differential equation with a small number of function evaluations / decision variables / constraints, then some new formulations are possible that take advantage of the constrained optimization formulation. These include the so-called *collocation methods*.

In direct collocation (c.f., [4]), both the input trajectory and the state trajectory are represented explicitly as piecewise polynomial functions. In particular, the sweet spot for this algorithm is taking $\mathbf{u}(t)$ to be a first-order polynomial and $\mathbf{x}(t)$ to be a cubic polynomial.

It turns out that in this sweet spot, the only decision variables we need in our optimization are the sample values $\mathbf{u}(t)$ and $\mathbf{x}(t)$ at the so called "break" points of the spline. You might think that you would need the coefficients of the cubic spline

parameters, but you do not. For the first-order interpolation on $\mathbf{u}(t)$, given $\mathbf{u}(t_k)$ and $\mathbf{u}(t_{k+1})$, we can solve for every value $\mathbf{u}(t)$ over the interval $t \in [k, k+1]$. But we also have everything that we need for the cubic spline: given $\mathbf{x}(t_k)$ and $\mathbf{u}(t_k)$, we can compute $\dot{\mathbf{x}}(t_k) = f(\mathbf{x}(t_k), \mathbf{u}(t_k))$; and the four values $\mathbf{x}(t_k), \mathbf{x}(t_{k+1}), \dot{\mathbf{x}}(t_k), \dot{\mathbf{x}}(t_{k+1})$ completely define all of the parameters of the cubic spline over the interval $t \in [t_k, t_{k+1}]$. This is very convenient, because it is easy for us to add additional constraints to $\mathbf{u}$ and $\mathbf{x}$ at the sample points (and would have been relatively harder to have to convert every constraint into constraints on the spline coefficients).

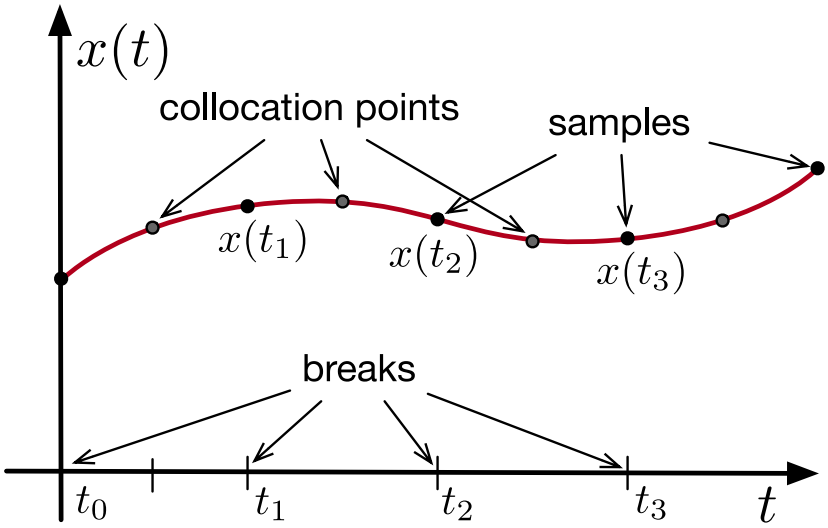


Figure 10.1 - Cubic spline parameters used in the direct collocation method.

It turns out that we need one more constraint per time segment to enforce the dynamics and to fully specify the trajectory. In direct collocation, we add a derivative constraint at the so-called *collocation points*. In particular, if we choose the collocation points to be the midpoints of the spline, then we have that

$$
\begin{aligned}
t_{c,k} = \frac{1}{2}(t_k + t_{k+1}), \qquad h_k = t_{k+1} - t_k,\\
\mathbf{u}(t_{c,k}) = \frac{1}{2}(\mathbf{u}(t_k) + \mathbf{u}(t_{k+1})),\\
\mathbf{x}(t_{c,k}) = \frac{1}{2}(\mathbf{x}(t_k) + \mathbf{x}(t_{k+1})) + \frac{h}{8}(\dot{\mathbf{x}}(t_k) - \dot{\mathbf{x}}(t_{k+1})),\\
\dot{\mathbf{x}}(t_{c,k}) = -\frac{3}{2h}(\mathbf{x}(t_k) - \mathbf{x}(t_{k+1})) - \frac{1}{4}(\dot{\mathbf{x}}(t_k) + \dot{\mathbf{x}}(t_{k+1})).
\end{aligned}
$$

These equations come directly from the equations that fit the cubic spline to the end points/derivatives then interpolate them at the midpoint. They give us precisely what we need to add the dynamics constraint to our optimization at the collocation points:

$$
\begin{aligned}
\min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} h_n \ell(\mathbf{x}[n], \mathbf{u}[n]) \\
\text{subject to} \quad & \dot{\mathbf{x}}(t_{c,n}) = f(\mathbf{x}(t_{c,n}), \mathbf{u}(t_{c,n})), \qquad \forall n \in [0, N-1] \\
& \mathbf{x}[0] = \mathbf{x}_0 \\
& + \text{additional constraints.}
\end{aligned}
$$

I hope this notation is clear -- I'm using $\mathbf{x}[k] = \mathbf{x}(t_k)$ as the decision variables, and the collocation constraint at $t_{c,k}$ depends on the decision variables: $\mathbf{x}[k], \mathbf{x}[k+1], \mathbf{u}[k], \mathbf{u}[k+1]$. The actual equations of motion get evaluated at both the break points, $t_k$, and the collocation points, $t_{c,k}$.

Once again, direct collocation effectively integrates the equations of motion by satisfying the constraints of the optimization -- this time producing an integration of the dynamics that is accurate to third-order with effectively two evaluations of the

plant dynamics per segment (since we use $\dot{\mathbf{x}}(t_k)$ for two intervals). [4] claims, without proof, that as the break points are brought closer together, the trajectory will converge to a true solution of the differential equation. Once again it is very natural to add additional terms to the cost function or additional input/state constraints, and very easy to calculate the gradients of the objective and constraints. I personally find it very nice to explicitly account for the parametric encoding of the trajectory in the solution technique.

**Example 10.2 (Direct Collocation for the Pendulum, Acrobot, and Cart-Pole)**

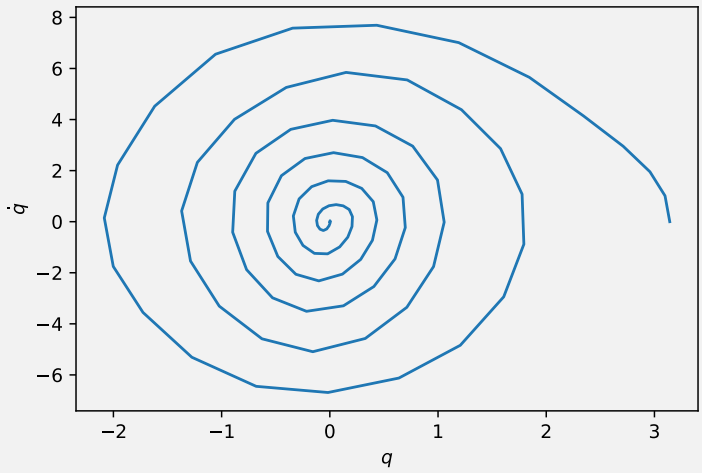



Figure 10.2 - A swing-up trajectory for the simple pendulum (with severe torque limits) optimized using direct collocation.

Direct collocation also easily solves the swing-up problem for the pendulum, Acrobot, and cart-pole system. Try it for yourself:

Open in Colab

As always, make sure you take a look at the code!

## 10.3.3 Pseudo-spectral Methods

The direct collocation method of [4] was our first example of explicitly representing the solution of the optimal control problem as a parameterized trajectory, and adding constraints to the derivatives at a series of collocation points. In the algorithm above, the representation of choice was *piecewise-polynomials*, e.g. cubic spines, and the spline coefficients were the decision variables. A closely related approach, often called "pseudo-spectral" optimal control, uses the same collocation idea, but represents the trajectories instead using a linear combination of *global, polynomial* basis functions. These methods use typically much higher-degree polynomials, but can leverage clever parameterizations to write sparse collocation objectives and to select the collocation points [5, 6]. Interestingly, The continuously-differentiable nature of the representation of these methods has led to comparatively more theorems and analysis than we have seen for other direct trajectory optimization methods [6] -- but despite some of the language used in these articles please remember they are still local optimization methods trying to solve a nonconvex optimization problem. While the direct collocation method above might be expected to converge to the true optimal solution by adding more segments to the piecewise polynomial (and having each segment represent a smaller interval of time), here we expect convergence to happen as we increase the degree of the polynomials.

The pseudo-spectral methods are also sometimes knowns as "orthogonal collocation" because the $N$ basis polynomials, $\phi_j(t)$, are chosen so that at the $N$th collocation point $t_j$, we have

$$\phi_i(t_j) = \begin{cases} 1 & i = j, \\ 0 & \text{otherwise.} \end{cases}$$

This can be accomplished by choosing

$$\phi_j(t) = \prod_{i=0, i\neq j}^{N} \frac{t - t_i}{t_j - t_i}.$$

Note that for both numerical reasons and for analysis, time is traditionally rescaled from the interval $[t_0, t_f]$ to $[-1, 1]$. Collocation points are chosen based on small variations of Gaussian quadrature, known as the "Gauss-Lobatto" which includes collocation points at $t = -1$ and $t = 1$.

Interestingly, a number of papers have also infinite-horizon pseudo-spectral optimization by the nonlinear rescaling of the time interval $t \in [0, \infty)$ to the half-open interval $\tau \in [-1, 1)$ via $\tau = \frac{t-1}{t+1}$ [5, 6]. In this case, one chooses the collocation times so that they include $\tau = -1$ but do not include $\tau = 1$, using the so-called "Gauss-Radau" points [5].

## Dynamic constraints in implicit form

There is another, seemingly subtle but potentially important, opportunity that can be exploited in a few of these transcriptions, if our main interest is in optimizing systems with significant multibody dynamics. In some cases, we can actually write the dynamics constraints directly in their implicit form. We've introduced this idea already in the context of Lyapunov analysis. In many cases, it is nicer or more efficient to obtain the equations of motion in an implicit form, $\mathbf{g}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{x}}) = 0$, and to avoid ever having to solve for the explicit form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. This can become even more important when we consider systems for which the explicit form doesn't have a unique solution -- we will see examples of this when we study trajectory optimization through contact because the Coulomb model for friction actually results in a differential inclusion instead of a differential equation.

The collocation methods, which operate on the dynamic constraints at collocation points directly in their continuous form, can use the implicit form directly. It is possible to write a time-stepping (discrete-time approximation) for direct transcription using implicit integrators -- again providing constraints in implicit form. The implicit form is harder to exploit in the shooting methods.

## 10.3.4 Solution techniques

The different transcriptions presented above represent different ways to map the (potentially continuous-time) optimal control problem into a finite set of decision variables, objectives, and constraints. But even once that choice is made, there are numerous approaches to solving this optimization problem. Any general approach to nonlinear programming can be applied here; in the python examples we've included so far, the problems are handed directly to the sequential-quadratic programming (SQP) solver SNOPT, or to the interior-point solver IPOPT.

There is also quite a bit of exploitable problem-specific structure in these trajectory optimization problems due to the sequential nature of the problem. As a result, there are some ideas that are fairly specific to the trajectory optimization formulation of optimal control, and customized solvers can often (and sometimes dramatically) outperform general purpose solvers.

This trajectory-optimization structure is easiest to discuss, and implement, in unconstrained formulations, so we will start there. In fact, in recent years we

have seen a surge in popularity in robotics for doing trajectory optimization using (often special-purpose) solvers for unconstrained trajectory optimization, where the constrained problems are transformed into unconstrained problem via penalty methods. I would say penalty methods based on the augmented Lagrangian are particularly popular for trajectory optimization these days [7, 8].

## Efficiently computing gradients

Providing gradients of the objectives and constraints to the solver is not strictly required -- most solvers will obtain them from finite differences if they are not provided -- but I feel strongly that the solvers are faster and more robust when exact gradients are provided. Providing the gradients for the direct transcription methods is very straight-forward -- we simply provide the gradients for each constraint individually. But in the direct shooting approach, where we have removed the $\mathbf{x}$ decision variables from the program but still write objectives and constraints in terms of $\mathbf{x}$, it would become very inefficient to compute the gradients of each objective/constraint independently. We need to leverage the chain rule.

To be concise (and slightly more general), let us define $\mathbf{x}[n+1] = f_d(\mathbf{x}[n], \mathbf{u}[n])$ as the discrete-time approximation of the continuous dynamics; for example, the forward Euler integration scheme used above would give $f_d(\mathbf{x}[n], \mathbf{u}[n]) = \mathbf{x}[n] + f(\mathbf{x}[n], \mathbf{u}[n])dt$. Then we have

$$\frac{\partial J}{\partial \mathbf{u}_k} = \frac{\partial \ell_f(\mathbf{x}[N])}{\partial \mathbf{u}_k} + \sum_{n=0}^{N-1} \left( \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}[n]} \frac{\partial \mathbf{x}[n]}{\partial \mathbf{u}_k} + \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}_k} \right),$$

where the gradient of the state with respect to the inputs can be computed during the "forward simulation",

$$\frac{\partial \mathbf{x}[n+1]}{\partial \mathbf{u}_k} = \frac{\partial f_d(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}[n]} \frac{\partial \mathbf{x}[n]}{\partial \mathbf{u}_k} + \frac{\partial f_d(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}_k}.$$

These simulation gradients can also be used in the chain rule to provide the gradients of any constraints. Note that there are a lot of terms to keep around here, on the order of (state dim) $\times$ (control dim) $\times$ (number of timesteps). Ouch. Note also that many of these terms are zero; for instance with the Euler integration scheme above $\frac{\partial \mathbf{u}[n]}{\partial \mathbf{u}_k} = 0$ if $k \neq n$. (If this looks like I'm mixing two notations here, recall that I'm using $\mathbf{u}_k$ to represent the decision variable and $\mathbf{u}[n]$ to represent the input used in the $n$th step of the simulation.)

## The special case of direct shooting without state constraints

By solving for $\mathbf{x}(\cdot)$ ourselves, we've removed a large number of constraints from the optimization. If no additional state constraints are present, and the only gradients we need to compute are the gradients of the objective, then a surprisingly efficient algorithm emerges. I'll give the steps here without derivation, but will derive it in the Pontryagin section below:

1. Simulate forward:

   $$\mathbf{x}[n+1] = f_d(\mathbf{x}[n], \mathbf{u}_n),$$

   from $\mathbf{x}[0] = \mathbf{x}_0$.
2. Calculate backwards:

   $$\lambda[n-1] = \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}[n]}^T + \frac{\partial f(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}[n]}^T \lambda[n],$$

   from $\lambda[N-1] = \frac{\partial \ell_f(\mathbf{x}[N])}{\partial \mathbf{x}[N]}$.
3. Extract the gradients:

$$\frac{\partial J}{\partial \mathbf{u}[n]} = \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}[n]} + \lambda[n]^T \frac{\partial f(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}[n]},$$

with $\frac{\partial J}{\partial \mathbf{u}_k} = \sum_n \frac{\partial J}{\partial \mathbf{u}[n]} \frac{\partial \mathbf{u}[n]}{\partial \mathbf{u}_k}$.

Here $\lambda[n]$ is a vector the same size as $\mathbf{x}[n]$ which has an interpretation as $\lambda[n] = \frac{\partial J}{\partial \mathbf{x}[n+1]}^T$. The equation governing $\lambda$ is known as the *adjoint equation*, and it represents a dramatic efficiency improvement over calculating the huge number of simulation gradients described above. In case you are interested, yes the adjoint equation is exactly the *backpropagation algorithm* that is famous in the neural networks literature, or more generally a bespoke version of reverse-mode automatic differentiation.

### Getting good solutions... in practice.

As you begin to play with these algorithms on your own problems, you might feel like you're on an emotional roller-coaster. You will have moments of incredible happiness -- the solver may find very impressive solutions to highly nontrivial problems. But you will also have moments of frustration, where the solver returns an awful solution, or simply refuses to return a solution (saying "infeasible"). The frustrating thing is, you cannot distinguish between a problem that is actually infeasible, vs. the case where the solver was simply stuck in a local minima.

So the next phase of your journey is to start trying to "help" the solver along. There are two common approaches.

The first is tuning your cost function -- some people spend a lot of time adding new elements to the objective or adjusting the relative weight of the different components of the objective. This is a slippery slope, and I tend to try to avoid it (possibly to a fault; other groups tend to put out more compelling videos!).

The second approach is to give a better initial guess to your solver to put it in the vicinity of the "right" local minimal. I find this approach more satisfying, because for most problems I think there really is a "correct" formulation for the objective and constraints, and we should just aim to find the optimal solution. Once again, we do see a difference here between the direct shooting algorithms and the direct transcription / collocation algorithms. For shooting, we can only provide the solver with an initial guess for $\mathbf{u}(\cdot)$, whereas the other methods allow us to also specify an initial guess for $\mathbf{x}(\cdot)$ directly. I find that this can help substantially, even with very simple initializations. In the direct collocation examples for the swing-up problem of the Acrobot and cart-pole, simply providing the initial guess for $\mathbf{x}(\cdot)$ as a straight line trajectory between the start and the goal was enough to help the solver find a good solution; in fact it was necessary.

## 10.4 Local Trajectory Feedback Design

Once we have obtained a locally optimal trajectory from trajectory optimization, we have found an open-loop trajectory that (at least locally) minimizes our optimal control cost. Up to numerical tolerances, this pair $\mathbf{u}_0(t), \mathbf{x}_0(t)$ represents a feasible solution trajectory of the system. But we haven't done anything, yet, to ensure that this trajectory is locally stable.

In fact, there are a few notable approximations that we've already made to get to this point: the integration accuracy of our trajectory optimization tends to be much less than the accuracy used during forward simulation (we tend to take bigger time steps during optimization to avoid adding too many decision variables), and the default convergence tolerance from the optimization toolboxes tend to satisfy the dynamic constraints only to around $10^{-6}$. As a result, if you were to simulate the optimized control trajectory directly *even from the exact initial conditions* used in / obtained from trajectory optimization, you might find that the state trajectory diverges from your planned trajectory.

There are a number of things we can do about this. It is possible to evaluate the local stability of the trajectory during the trajectory optimization, and add a cost or constraint that rewards open-loop stability (e.g. [9, 10]). This can be very effective (though it does tend to be expensive). But open-loop stability is a quite restrictive notion. A potentially more generally useful approach is to design a feedback controller to regulate the system back to the planned trajectory.

### 10.4.1 Finite-horizon LQR

We have already developed on approach for trajectory stabilization in the LQR chapter. This is one of my favorite approaches to trajectory feedback, because it provides a (numerically) closed-form solution for the controller, $\mathbf{K}(t)$, and even comes with a time-varying quadratic cost-to-go function, $S(t)$, that can be used for Lyapunov analysis.

The basic procedure is to create a time-varying linearization along the trajectory in the error coordinates: $\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_0(t)$, $\bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_0(t)$, and $\dot{\bar{\mathbf{x}}}(t) = \mathbf{A}(t)\bar{\mathbf{x}}(t) + \mathbf{B}(t)\bar{\mathbf{u}}(t)$. This linearization uses all of the same gradients of the dynamics that we have been using in our trajectory optimization algorithms. Once we have the time-varying linearization, then we can apply finite-horizon LQR (see the LQR chapter for the details).

A major virtue of this approach is that we can proceed immediately to verifying the performance of the closed-loop system under the LQR policy. Specifically, we can apply the finite-time reachability analysis to obtain "funnels" that certify a desired notion of invariance -- guaranteeing that trajectories which start near the planned trajectory will stay near the planned trajectory. Please see the finite-time reachability analysis section for those details. We will put all of these ideas together in the perching case-study below.

### 10.4.2 Model-Predictive Control

The maturity, robustness, and speed of solving trajectory optimization using convex optimization leads to a beautiful idea: if we can optimize trajectories quickly enough, then we can use our trajectory optimization as a feedback policy. The recipe is simple: (1) measure the current state, (2) optimize a trajectory from the current state, (3) execute the first action from the optimized trajectory, (4) let the dynamics evolve for one step and repeat. This recipe is known as *model-predictive control* (MPC).

Despite the very computational nature of this controller (there is no closed-form representation of this policy; it is represented only implicitly as the solution of the optimization), there is a bounty of theoretical and algorithmic results on MPC [11, 12]. And there are a few core ideas that practitioners should really understand.

One core idea is the concept of *receding-horizon* MPC. Since our trajectory optimization problems are formulated over a finite-horizon, we can think each optimization as reasoning about the next $N$ timesteps. If our true objective is to optimize the performance over a horizon longer than $N$ (e.g. over the infinite horizon), then it is standard to continue solving for an $N$ step horizon on each evaluation of the controller. In this sense, the total horizon under consideration continues to move forward in time (e.g. to recede).

Some care must be taken in receding-horizon formulations because on each new step we are introducing new costs and constraints into the problem (the ones that would have been associated with time $N+1$ on the previous solve) -- it would be very bad to march forward in time solving convex optimization problems only to suddenly encounter a situation where the solver returns "infeasible!". One can design MPC formulations that guarantee *recursive feasibility* -- e.g. guarantee that if a feasible solution is found at time $n$, then the solver will also find a feasible solution at time $n+1$.

Perhaps the simplest mechanism for guaranteeing recursive feasibility in an optimization for stabilizing a fixed point, $(\mathbf{x}^*, \mathbf{u}^*)$, is to add a final-value constraint to the receding horizon, $\mathbf{x}[N] = \mathbf{x}^*$. This idea is simple but important. Considering the trajectories/constraints in absolute time, then on step $k$ of the algorithm, we are optimizing for $\mathbf{x}[k], \ldots, \mathbf{x}[k+N]$, and $\mathbf{u}[k], \ldots, \mathbf{u}[k+N-1]$; let us say that we have found a feasible solution for this problem. The danger in receding-horizon control is that when we shift to the next step $(k+1)$ we introduce constraints on the system at $\mathbf{x}[k+N+1]$ for the first time. But if our feasible solution in step $k$ had $\mathbf{x}[k+N] = \mathbf{x}^*$, then we know that setting $\mathbf{x}[k+N+1] = \mathbf{x}^*, \mathbf{u}[k+N] = \mathbf{u}^*$ is guaranteed to provide a feasible solution to the new optimization problem in step $k+1$. With feasibility guaranteed, the solver is free to search for a lower-cost solution (which may be available now because we've shifted the final-value constraint further into the future). It is also possible to formulate MPC problems that guarantee recursive feasibility even in the presence of modeling errors and disturbances (c.f. [13]).

The theoretical and practical aspects of Linear MPC are so well understood today that it is considered the de-facto generalization of LQR for controlling a linear system subject to (linear) constraints.

## 10.5 Case Study: A glider that can land on a perch like a bird

From 2008 til 2014, my group conducted a series of increasingly sophisticated investigations [14, 15, 16, 17, 18, 19] which asked the question: can a fixed-wing UAV land on a perch like a bird?

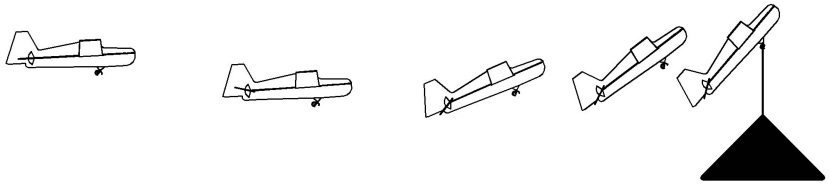


Figure 10.3 - Basic setup for the glider landing on a perch.

At the outset, this was a daunting task. When birds land on a perch, they pitch up and expose their wings to an "angle-of-attack" that far exceeds the typical flight envelope. Airplanes traditionally work hard to avoid this regime because it leads to aerodynamic "stall" -- a sudden loss of lift after the airflow separates from the wing. But this loss of lift is also accompanied by a significant increase in drag, and birds exploit this when they rapidly decelerate to land on a perch. Post-stall aerodynamics are a challenge for control because (1) the aerodynamics are time-varying (characterized by periodic vortex shedding) and nonlinear, (2) it is much harder to build accurate models of this flight regime, at least in a wind tunnel, and (3) stall implies a loss of attached flow on the wing and therefore on the control surfaces, so a potentially large reduction in control authority.

We picked the project initially thinking that it would be a nice example for model-free control (like reinforcement learning -- since the models were unknown). In the end, however, it turned out to be the project that really taught me about the power of model-based trajectory optimization and linear optimal control. By conducting dynamic system identification experiments in a motion capture environment, we were able to fit both surprisingly simple models (based on flat-plate theory) to the dynamics[14], and also more accurate models using "neural-network-like" terms to capture the residuals between the model and the data [19]. This made model-based control viable, but the dynamics were still complex -- while trajectory optimization should work, I was quite dubious about the potential for regulating to those trajectories with only linear feedback.

I was wrong. Over and over again, I watched time-varying linear quadratic regulators take highly nontrivial corrective actions -- for instance, dipping down early in the trajectory to gain kinetic energy or tipping up to dump energy out of the

system -- in order to land on the perch at the final time. Although the quality of the linear approximation of the dynamics did degrade the farther that we got from the nominal trajectory, the validity of the controller dropped off much less quickly (even as the vector field changed, the direction that the controller needed to push did not). This was also the thinking that got me initially so interested in understanding the regions of attraction of linear control on nonlinear systems.

In the end, the experiments were very successful. We started searching for the "simplest" aircraft that we could build that would capture the essential control dynamics, reduce complexity, and still accomplish the task. We ended up building a series of flat-plate foam gliders (no propellor) with only a single actuator to control the elevator. We added dihedral to the wings to help the aircraft stay in the longitudinal plane. The simplicity of these aircraft, plus the fact that they could be understood through the lens of quite simple models makes them one of my favorite canonical underactuated systems.

Figure 10.4 - The original perching experiments from [14] in a motion capture arena with a simple rope acting as the perch. The main videos were shot with high-speed cameras; an entire perching trajectory takes about .8 seconds.

### 10.5.1 The Flat-Plate Glider Model

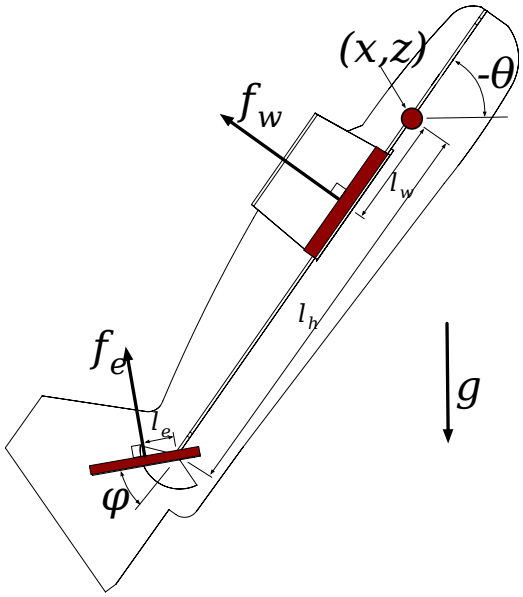



Figure 10.5 - The flat-plate glider model. Note that traditionally aircraft coordinates are chosen so that positive pitch brings the nose up; but this means that positive z is down (to have a right-handed system). For consistency, I've chosen to stick with the vehicle coordinate system that we use throughout the text -- positive z is up, but positive pitch is down.

In our experiments, we found the dynamics of our aircraft were captured very well by the so-called "flat plate model" [14]. In flat plate theory lift and drag forces of a wing are summarized by a single force at the center-of-pressure of the wing acting normal to the wing, with magnitude:

$$f_n(S, \mathbf{n}, \mathbf{v}) = \rho S \sin\alpha |\mathbf{v}|^2 = -\rho S (\mathbf{n} \cdot \mathbf{v}) |\mathbf{v}|,$$

where $\rho$ is the density of the air, $S$ is the area of the wing, $\alpha$ is the angle of attack of the surface, $\mathbf{n}$ is the normal vector of the lifting surface, and $\mathbf{v}$ is the velocity of the center of pressure relative to the air. This corresponds to having lift and drag coefficients

$$c_{\text{lift}} = 2 \sin\alpha \cos\alpha, \quad c_{\text{drag}} = 2 \sin^2 \alpha.$$

In our glider model, we summarize all of the aerodynamic forces by contributions of two lifting surfaces, the wing (including some contribution from the horizontal fuselage) denoted by subscript $w$ and the elevator denoted by subscript $e$, with centers at $\mathbf{p}_w = [x_w, z_w]^T$ and $\mathbf{p}_e = [x_e, z_e]^T$ given by the kinematics:

$$\mathbf{p}_w = \mathbf{p} - l_w \begin{bmatrix} c_\theta \\ -s_\theta \end{bmatrix}, \quad \mathbf{p}_e = \mathbf{p} - l_h \begin{bmatrix} c_\theta \\ -s_\theta \end{bmatrix} - l_e \begin{bmatrix} c_{\theta+\phi} \\ -s_{\theta+\phi} \end{bmatrix},$$

where the origin of our vehicle coordinate system, $\mathbf{p} = [x, z]^T$, is chosen to be the center of mass. We assume still air, so $\mathbf{v} = \dot{\mathbf{p}}$ for both the wing and the elevator. We assume that the elevator is massless, and the actuator controls velocity directly (note that this breaks our "control affine" structure, but is more realistic for the tiny hobby servos we were dealing with). This gives us a total of 7 state variables $\mathbf{x} = [x, z, \theta, \phi, \dot{x}, \dot{z}, \dot{\theta}]^T$ and one control input $\mathbf{u} = \dot{\phi}$. The resulting equations of motion are:

$$\mathbf{n}_w = \begin{bmatrix} s_\theta \\ c_\theta \end{bmatrix}, \quad \mathbf{n}_e = \begin{bmatrix} s_{\theta+\phi} \\ c_{\theta+\phi} \end{bmatrix},$$
$$f_w = f_n(S_w, \mathbf{n}_w, \dot{\mathbf{p}}_w), \quad f_e = f_n(S_e, \mathbf{n}_e, \dot{\mathbf{p}}_e),$$
$$\ddot{x} = \frac{1}{m}(f_w s_\theta + f_e s_{\theta+\phi}),$$
$$\ddot{z} = \frac{1}{m}(f_w c_\theta + f_e c_{\theta+\phi}) - g,$$
$$\ddot{\theta} = \frac{1}{I}(l_w f_w + (l_h c_\phi + l_e) f_e).$$

### 10.5.2 Trajectory optimization


### 10.5.3 Trajectory stabilization

### 10.5.4 Trajectory funnels

### 10.5.5 Beyond a single trajectory

The linear controller around a nominal trajectory was surprisingly effective, but it's not enough. We'll return to this example again when we talk about "feedback motion planning", in order to discuss how to find a controller that can work for many more initial conditions -- ideally all of the initial conditions of interest for which the aircraft is capable of getting to the goal.

## 10.6 Pontryagin's Minimum Principle

The tools that we've been developing for numerical trajectory optimization are closely tied to theorems from (analytical) optimal control. Let us take one section to appreciate those connections.

What precisely does it mean for a trajectory, $\mathbf{x}(\cdot), \mathbf{u}(\cdot)$, to be locally optimal? It means that if I were to perturb that trajectory in any way (e.g. change $\mathbf{u}_3$ by $\epsilon$), then I would either incur higher cost in my objective function or violate a constraint. For an unconstrained optimization, a *necessary condition* for local optimality is that the gradient of the objective at the solution be exactly zero. Of course the gradient can also vanish at local maxima or saddle points, but it certainly must vanish at local minima. We can generalize this argument to constrained optimization using *Lagrange multipliers*.

### 10.6.1 Lagrange multiplier derivation of the adjoint equations

Let us use Lagrange multipliers to derive the necessary conditions for our constrained trajectory optimization problem in discrete time

$$\min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \ell_f(\mathbf{x}[N]) + \sum_{n=0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]),$$
$$\text{subject to} \quad \mathbf{x}[n+1] = f_d(\mathbf{x}[n], \mathbf{u}[n]).$$

Formulate the Lagrangian,

$$L(\mathbf{x}[\cdot], \mathbf{u}[\cdot], \lambda[\cdot]) = \ell_f(\mathbf{x}[N]) + \sum_{n=0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) + \sum_{n=0}^{N-1} \lambda^T[n] \left(f_d(\mathbf{x}[n], \mathbf{u}[n]) - \mathbf{x}[n+1]\right),$$

and set the derivatives to zero to obtain the adjoint equation method described for the shooting algorithm above:

$$\begin{aligned}
\forall n \in [0, N-1], \frac{\partial L}{\partial \lambda[n]} &= f_d(\mathbf{x}[n], \mathbf{u}[n]) - \mathbf{x}[n+1] = 0 \Rightarrow \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]) \\
\forall n \in [0, N-1], \frac{\partial L}{\partial \mathbf{x}[n]} &= \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}} + \lambda^T[n] \frac{\partial f_d(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}} - \lambda^T[n-1] = 0 \\
\Rightarrow \lambda[n-1] &= \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}}^T + \frac{\partial f_d(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}}^T \lambda[n]. \\
\frac{\partial L}{\partial \mathbf{x}[N]} &= \frac{\partial \ell_f}{\partial \mathbf{x}}^T - \lambda^T[N-1] = 0 \Rightarrow \lambda[N-1] = \frac{\partial \ell_f}{\partial \mathbf{x}} \\
\forall n \in [0, N-1], \frac{\partial L}{\partial \mathbf{u}[n]} &= \frac{\partial \ell(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}} + \lambda^T[n] \frac{\partial f_d(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}} = 0.
\end{aligned}$$

Therefore, if we are given an initial condition $\mathbf{x}_0$ and an input trajectory $\mathbf{u}[\cdot]$, we can verify that it satisfies the necessary conditions for optimality by simulating the system forward in time to solve for $\mathbf{x}[\cdot]$, solving the adjoint equation backwards in time to solve for $\lambda[\cdot]$, and verifying that $\frac{\partial L}{\partial \mathbf{u}[n]} = 0$ for all $n$. The fact that $\frac{\partial J}{\partial \mathbf{u}} = \frac{\partial L}{\partial \mathbf{u}}$ when $\frac{\partial L}{\partial \mathbf{x}} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$ follows from some basic results in the calculus of variations.

## 10.6.2 Necessary conditions for optimality in continuous time

You won't be surprised to hear that these necessary conditions have an analogue in continuous time. I'll state it here without derivation. Given the initial conditions, $\mathbf{x}_0$, a continuous dynamics, $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, and the instantaneous cost $\ell(\mathbf{x}, \mathbf{u})$, for a trajectory $\mathbf{x}(\cdot), \mathbf{u}(\cdot)$ defined over $t \in [t_0, t_f]$ to be optimal it must satisfy the conditions that

$$\begin{aligned}
&\forall t \in [t_0, t_f], \quad \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), && \mathbf{x}(0) = \mathbf{x}_0 \\
&\forall t \in [t_0, t_f], \quad -\dot{\lambda} = \frac{\partial \ell}{\partial \mathbf{x}}^T + \frac{\partial f}{\partial \mathbf{x}}^T \lambda, && \lambda(T) = \frac{\partial \ell_f}{\partial \mathbf{x}}^T \\
&\forall t \in [t_0, t_f], \quad \frac{\partial \ell}{\partial \mathbf{u}} + \lambda^T \frac{\partial f}{\partial \mathbf{u}} = 0.
\end{aligned}$$

In fact the statement can be generalized even beyond this to the case where $\mathbf{u}$ has constraints. The result is known as Pontryagin's minimum principle -- giving *necessary conditions* for a trajectory to be optimal.

### Theorem 10.1 - Pontryagin's Minimum Principle

Adapted from [20]. Given the initial conditions, $\mathbf{x}_0$, a continuous dynamics, $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, and the instantaneous cost $\ell(\mathbf{x}, \mathbf{u})$, for a trajectory $\mathbf{x}^*(\cdot), \mathbf{u}^*(\cdot)$ defined over $t \in [t_0, t_f]$ to be optimal it must satisfy the conditions that

$$\begin{aligned}
&\forall t \in [t_0, t_f], \quad \dot{\mathbf{x}}^* = f(\mathbf{x}^*, \mathbf{u}^*), && \mathbf{x}^*(0) = \mathbf{x}_0 \\
&\forall t \in [t_0, t_f], \quad -\dot{\lambda}^* = \frac{\partial \ell}{\partial \mathbf{x}}^T + \frac{\partial f}{\partial \mathbf{x}}^T \lambda^*, && \lambda^*(T) = \frac{\partial \ell_f}{\partial \mathbf{x}}^T \\
&\forall t \in [t_0, t_f], \quad u^* = \text{argmin}_{\mathbf{u} \in \mathcal{U}} \left[\ell(\mathbf{x}^*, \mathbf{u}) + (\lambda^*)^T f(\mathbf{x}^*, \mathbf{u})\right].
\end{aligned}$$

Note that the terms which are minimized in the final line of the theorem are commonly referred to as the Hamiltonian of the optimal control problem,

$$H(\mathbf{x}, \mathbf{u}, \lambda, t) = \ell(\mathbf{x}, \mathbf{u}) + \lambda^T f(\mathbf{x}, \mathbf{u}).$$

It is distinct from, but inspired by, the Hamiltonian of classical mechanics. Remembering that $\lambda$ has an interpretation as $\frac{\partial J}{\partial \mathbf{x}}^T$, you should also recognize it from the HJB.

## 10.7 Variations and Extensions

### 10.7.1 Differential Flatness

There are some very important cases where nonconvex trajectory optimization can be turned back into convex trajectory optimization based on a clever change of variables. One of the key ideas in this space is the notion of "differential flatness", which is philosophically quite close to the idea of partial feedback linearization that we discussed for acrobots and cart-pole systems. But perhaps the most famous applciation of differential flatness, which we will explore here, is actually for quadrotors.

One of the most important lessons from partial feedback linearization, is the idea that if you have $m$ actuators, then you basically get to control exactly $m$ quantities of your system. Differential flatness exploits this same idea (choosing $m$ outputs), but in the opposite direction. The idea is that, for some systems, if you give me a trajectory in those $m$ coordinates, it may in fact dictate what all of the states/actuators must have been doing. The fact that you can only execute a subset of possible trajectories can, in this case, make trajectory planning much easier!

Let's start with an example...

**Example 10.3 (Differential flatness for the Planar Quadrotor)**

The planar quadrotor model described earlier has 3 degrees of freedom $(x, y, \theta)$ but only two actuators (one for each propellor). My claim is that, if you give me a trajectory for just the location of the center of mass: $x(t), y(t), \forall t \in [t_0, t_f]$, then I will be able to infer what $\theta(t)$ must be over that same interval in order to be feasible. Furthermore, I can even infer $\mathbf{u}(t)$. There is one technical condition required: the trajectory you give me for $x(t)$ and $y(t)$ needs to be continuously differentiable (four times).

To see this, recall the equations of motion for this system were given by:

$$m\ddot{x} = -(u_1 + u_2)\sin\theta, \tag{1}$$
$$m\ddot{y} = (u_1 + u_2)\cos\theta - mg, \tag{2}$$
$$I\ddot{\theta} = r(u_1 - u_2) \tag{3}$$

Observe that from (1) and (2) we have

$$\frac{-m\ddot{x}}{m\ddot{y} + mg} = \frac{(u_1 + u_2)\sin\theta}{(u_1 + u_2)\cos\theta} = \tan\theta.$$

In words, given $\ddot{x}(t)$ and $\ddot{y}(t)$, I can solve for $\theta$(t). I can differentiate this relationship (in time) twice more to obtain $\ddot{\theta}$. Using (3) with (1) or (2) give us two linear equations for $u_1$ and $u_2$ which are easily solved.

Now you can see why we need the original trajectories to be smooth -- the solution to $\mathbf{u}(t)$ depends on $\ddot{\theta}(t)$ which depends on $\frac{d^4x(t)}{dt^4}$ and $\frac{d^4y(t)}{dt^4}$; we need those derivatives to exist along the entire trajectory.

What's more -- if we ignore input limits for a moment -- *any* sufficiently smooth

trajectory of $x(t), y(t)$ is feasible, so if I can simply find one that avoids the obstacles, then I have designed my state trajectory. As we will see, optimizing even high-degree piecewise-polynomials is actually an easy problem (it works out to be a quadratic program), assuming the constraints are convex. In practice, this means that once you have determined whether to go left or right around the obstacles, trajectory design is easy and fast.

I've coded up a simple example of that for you here:

Open in Colab

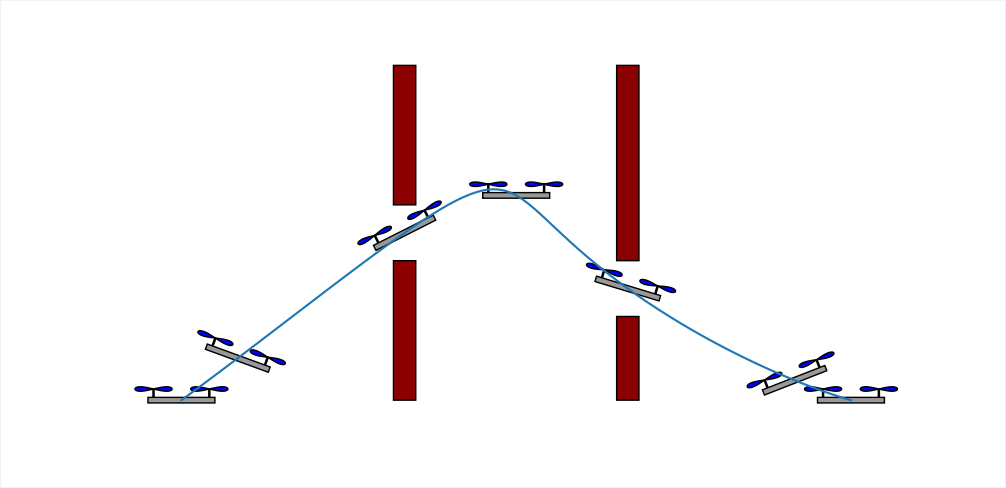


Figure 10.6 - Differential flatness for the planar quadrotor -- by solving a simple optimization to find a smooth trajectory for $x(t)$ and $y(t)$, I can back out $\theta(t)$ and even $\mathbf{u}(t)$.

The example above demonstrates that the planar quadrotor system is differentially flat in the outputs $x(t)$, $y(t)$. More generally, if we have a dynamical system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),$$

and we design some output coordinates (essentially "task-space"):

$$\mathbf{z}(t) = h\left(\mathbf{x}, \mathbf{u}, \frac{d\mathbf{u}}{dt}, \ldots, \frac{d^k\mathbf{u}}{dt^k}\right),$$

such that we can write the $\mathbf{x}$ and $\mathbf{u}$ purely as a function of the output and it's time derivatives,

$$\mathbf{x}(t) = \mathbf{x}\left(\mathbf{z}, \frac{d\mathbf{z}}{dt}, \ldots, \frac{d^k\mathbf{z}}{dt^k}\right),$$
$$\mathbf{u}(t) = \mathbf{u}\left(\mathbf{z}, \frac{d\mathbf{z}}{dt}, \ldots, \frac{d^k\mathbf{z}}{dt^k}\right),$$

then we say that the system $f$ is differentially flat in the outputs $\mathbf{z}$ [21]. And the requirement for showing that a system is differentially flat in those outputs is to write the function which solves for $\mathbf{x}(t)$ and $\mathbf{u}(t)$ as a function of only $\mathbf{z}(t)$ and its time derivatives.

I'm not aware of any numerical recipes for showing that a system is differentially flat nor for finding potential flat outputs, but I admit I haven't worked on it nor looked for those recipes. That would be interesting! I think of differential flatness as a property that one must find for your system -- typically via a little mechanical intuition and a lot of algebra. Once found, it can be very useful.

### Example 10.4 (Differential flatness for the 3D Quadrotor)

Probably the most famous example of differential flatness is on the full 3D quadrotors. [22] showed that the 3D quadrotor is differentially flat in the outputs $\{x, y, z, \theta_{yaw}\}$. They used this, to dramatic effect, to perform all sorts of acrobatics. The resulting videos are awesome (and probably deserve a lot of credit for the widespread popularity of quadrotors in academia and industry over the next few years).

Figure 10.7 - Aggressive quadrotor trajectories using differential flatness from [22].

A few things to note about these examples, just so we also understand the limitations. First, the technique above is great for designing trajectories, but additional work is required to stabilizing those trajectories (we'll cover that topic

in more detail later in the notes). Trajectory stabilization benefits greatly from good state estimation, and the examples above were all performed in a motion capture arena. Also, the "simple" version of the trajectory design that is fast enough to be used for flying through moving hoops is restricted to convex optimization formulations -- which means one has to hard-code apriori the decisions about whether to go left or right / up or down around each obstacle.

### 10.7.2 Iterative LQR and Differential Dynamic Programming

There is another approach to trajectory optimization (at least for initial-value problems) that has gotten quite popular lately. Iterative LQR (iLQR)[23] also known as Sequential Linear Quadratic optimal control) [24]. The idea is simple enough: given an initial guess at the input and state trajectory, make a linear approximation of the dynamics and a quadratic approximation of the cost function. Then compute and simulate the time-varying LQR controller to find a new input and state trajectory. Repeat until convergence.

The motivation behind iterative LQR is quite appealing -- it leverages the surprising structure of the Riccati equations to come up with a second-order update to the trajectory after a single backward pass of the Riccati equation followed by a forward simulation. Anecdotally, the convergence is fast and robust. Numerous applications...[25, 26].

One key limitation of iLQR is that it only supports unconstrained trajectory optimization -- at least via the Riccati solution. The natural extension for constrained optimization would be to replace the Riccati solution with an iterative linear model-predictive control (MPC) optimization; this would result in a quadratic program and is very close to what is happening in SQP. But a more common approach in the current robotics literature is to add any constraints into the objective function using penalty methods, especially using Augmented Lagrangian.

Differential Dynamic Programming (DDP) [27] is inspired by the idea of solving locally *quadratic* approximations of the HJB equations. The resulting algorithm is almost identical to iLQR; the only difference is that DDP uses a second-order approximation of the plant dynamics instead of a first-order (linear) one. The conventional wisdom is that taking these extra plant derivatives is probably not worth the extra cost of computing them; in most cases iLQR converges just as quickly (though the details will be problem dependent). Unfortunately, despite the distinction between these algorithms being unambiguous and uncontroversial, you will find many papers that incorrectly write DDP when they are actually using iLQR.

### 10.7.3 Mixed-integer convex optimization for non-convex constraints

**Example 10.5 (Mixed-integer planning around obstacles)**


[28], [29], [30, 31].

Recent work makes a much stronger connection between the optimization-approaches to graph search with our continuous optimization [32].

### 10.7.4 Explicit model-predictive control

## 10.8 Exercises

### Exercise 10.1 (Direct Shooting vs Direct Transcription)

In this coding exercise we explore in detail the computational advantages of direct transcription over direct shooting methods. The exercise can be completed entirely in this python notebook. To simplify the analysis, we will apply these two methods to a finite-horizon LQR problem. You are asked to complete four pieces of code:

a. Use the given implementation of direct shooting to analyze the numerical conditioning of this approach.
b. Complete the given implementation of the direct transcription method.
c. Verify that the cost to go from direct trascription approaches the LQR cost to go as the time horizon grows.
d. Analyze the numerical conditioning of direct transcription.
e. Implement the dynamic programming recursion (a.k.a. "Riccati recursion") to efficiently solve the linear algebra underlying the direct transcription method.

### Exercise 10.2 (Orbital Transfer via Trajectory Optimization)

For this exercise you will work exclusively in this notebook. You are asked to find, via nonlinear trajectory optimization, a path that efficiently transfers a rocket from the Earth to Mars, while avoiding a cloud of asteroids. The skeleton of the optimization problem is already there, but several important pieces are missing. More details are in the notebook, but you will need to:

a. Enforce the maximum thrust constraints.
b. Enforce the maximum velocity constraints.
c. Ensure that the rocket does not collide with any of the asteroids.
d. Add an objective function to minimize fuel consumption.

### Exercise 10.3 (Iterative Linear Quadratic Regulator)

The exercise is self-contained in this notebook. In this exercise you will derive and implement the iterative Linear Quadratic Regulator (iLQR). You will evaluate it's functionality by planning trajectories for an autonomous vehicle. You will need to:

a. Define an integrator for continuous dynamics.
b. Compute a trajectory given an initial state and a control trajectory.
c. Sum up the total cost of that trajectory.
d. Derive the coefficients of the quadratic Q-function.
e. Optimize for the optimal control law.
f. Derive the update of the value function backwards in time.
g. Implement the forward pass of iLQR.
h. Implement the backward pass of iLQR.

## References

1. John T. Betts, "Survey of numerical methods for trajectory optimization",

*Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193-207, 1998.

2. John T. Betts, "Practical Methods for Optimal Control Using Nonlinear Programming",Society for Industrial and Applied Mathematics , 2001.

3. Yang Wang and Stephen Boyd, "Fast model predictive control using online optimization", *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267--278, 2009.

4. C. R. Hargraves and S. W. Paris, "Direct Trajectory Optimization using Nonlinear Programming and Collocation", *J Guidance*, vol. 10, no. 4, pp. 338-342, July-August, 1987.

5. Divya Garg and Michael Patterson and Camila Francolin and Christopher Darby and Geoffrey Huntington and William Hager and Anil Rao, "Direct trajectory optimization and costate estimation ofÂ finite-horizon and infinite-horizon optimal control problems using a Radau pseudospectral method", *Computational Optimization and Applications*, vol. 49, pp. 335-358, 2011.

6. I. Michael Ross and Mark Karpenko, "A review of pseudospectral optimal control: {From} theory to flight", *Annual Reviews in Control*, vol. 36, no. 2, pp. 182--197, dec, 2012.

7. TC Lin and JS Arora, "Differential dynamic programming technique for constrained optimal control", *Computational Mechanics*, vol. 9, no. 1, pp. 27--40, 1991.

8. Marc Toussaint, "A Novel Augmented Lagrangian Approach for Inequalities and Convergent Any-Time Non-Central Updates", , 2014.

9. Katja D. Mombaur and Hans Georg Bock and Johannes P. Schloder and Richard W. Longman, "Open-loop stable solutions of periodic optimal control problems in robotics", *Z. Angew. Math. Mech. (ZAMM)*, vol. 85, no. 7, pp. 499 Ã¢â‚¬â€œ 515, 2005.

10. Aaron M Johnson and Jennifer E King and Siddhartha Srinivasa, "Convergent planning", *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1044--1051, 2016.

11. Carlos E Garcia and David M Prett and Manfred Morari, "Model predictive control: theory and practiceâ€”a survey", *Automatica*, vol. 25, no. 3, pp. 335--348, 1989.

12. Eduardo F Camacho and Carlos Bordons Alba, "Model predictive control",Springer Science \& Business Media , 2013.

13. Alberto Bemporad and Manfred Morari, "Robust model predictive control: A survey", *Robustness in identification and control*, vol. 245, pp. 207-226, 1999.

14. Rick Cory and Russ Tedrake, "Experiments in Fixed-Wing {UAV} Perching", *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pp. 1-12, 2008. [ link ]

15. John W. Roberts and Rick Cory and Russ Tedrake, "On the Controllability of Fixed-Wing Perching", *Proceedings of the American Control Conference (ACC)*, 2009. [ link ]

16. Rick Cory, "Supermaneuverable Perching", PhD thesis, Massachusetts Institute of Technology, June, 2010. [ link ]

17. Joseph Moore, "Powerline Perching with a Fixed-wing {UAV}", , May, 2011.

[ link ]

18. Joseph Moore and Russ Tedrake, "Control Synthesis and Verification for a Perching {UAV} using {LQR}-Trees", *In Proceedings of the IEEE Conference on Decision and Control*, pp. 8, December, 2012. [ link ]

19. Joseph Moore, "Robust Post-Stall Perching with a Fixed-Wing UAV", PhD thesis, Massachusetts Institute of Technology, September, 2014. [ link ]

20. Dimitri P. Bertsekas, "Dynamic Programming and Optimal Control",Athena Scientific , 2000.

21. Richard M. Murray and Muruhan Rathinam and Willem Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems", *ASME international mechanical engineering congress and exposition*, 1995.

22. D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors", *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520--2525, 2011.

23. Weiwei Li and Emanuel Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems.", *International Conference on Informatics in Control, Automation and Robotics*, pp. 222--229, 2004.

24. Athanasios Sideris and James E. Bobrow, "A Fast Sequential Linear Quadratic Algorithm for Solving Unconstrained Nonlinear Optimal Control Problems", , February, 2005.

25. Farbod Farshidian and Edo Jelavic and Asutosh Satapathy and Markus Giftthaler and Jonas Buchli, "Real-time motion planning of legged robots: A model predictive control approach", *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 577--584, 2017.

26. Y. Tassa and T. Erez and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization", *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4906--4913, 2012.

27. David H. Jacobson and David Q. Mayne, "Differential Dynamic Programming",American Elsevier Publishing Company, Inc. , 1970.

28. A.~Richards and J.P.~How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming", *Proceedings of the 2002 American Control Conference*, vol. 3, pp. 1936--1941, 2002.

29. Daniel Mellinger and Alex Kushleyev and Vijay Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams", *2012 IEEE international conference on robotics and automation*, pp. 477--483, 2012.

30. Robin Deits and Russ Tedrake, "Efficient Mixed-Integer Planning for {UAVs} in Cluttered Environments", *Proceedings of the {IEEE} International Conference on Robotics and Automation ({ICRA})*, 2015. [ link ]

31. Benoit Landry and Robin Deits and Peter R. Florence and Russ Tedrake, "Aggressive Quadrotor Flight through Cluttered Environments Using Mixed Integer Programming", *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May, 2016. [ link ]

32. Tobia Marcucci and Jack Umenberger and Pablo A. Parrilo and Russ Tedrake, "Shortest Paths in Graphs of Convex Sets", *arXiv preprint*, 2021. [ link ]

33. O. Junge and J. E. Marsden and S. Ober-Bloebaum, "Discrete mechanics and optimal control", *Proceedings of the 16th IFAC World Congress*, 2005.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 11
# Policy Search

Open in Colab

In our case study on perching aircraft, we solved a challenging control problem, but our approach to control design was based on only linear optimal control (around an optimized trajectory). We've also discussed some approaches to nonlinear optimal control that could scale beyond small, discretized state spaces. These were based on estimating the cost-to-go function, including value iteration using function approximation and approximate dynamic programming as a linear program or as a sums-of-squares program.

There are a lot of things to like about methods that estimate the cost-to-go function (aka value function). The cost-to-go function reduces the long-term planning problem into a one-step planning problem; it encodes all relevant information about the future (and nothing more). The HJB gives us optimality conditions for the cost-to-go that give us a strong algorithmic handle to work with.

In this chapter, we will explore another very natural idea: let us parameterize a controller with some decision variables, and then search over those decision variables directly in order to achieve a task and/or optimize a performance objective. One specific motivation for this approach is the (admittedly somewhat anecdotal) observation that often times simple policies can perform very well on complicated robots and with potentially complicated cost-to-go functions.

We'll refer to this broad class of methods as "policy search" or, when optimization methods are used, we'll sometimes use "policy optimization". This idea has not received quite as much attention from the controls community, probably because we know many relatively simple cases where it does not work well. But it has become very popular again lately due to the empirical success of "policy gradient" algorithms in reinforcement learning (RL). This chapter includes a discussion of the "model-based" version of these RL policy-gradient algorithms; we'll describe their "model-free" versions in a future chapter.

## 11.1 Problem formulation

Consider a static full-state feedback policy,

$$\mathbf{u} = \pi_\alpha(\mathbf{x}),$$

where $\pi$ is potentially a nonlinear function, and $\alpha$ is the vector of parameters that describe the controller. The control might take time as an input, or might even have it's own internal state, but let's start with this simple form.

Using our prescription for optimal control using additive costs, we can evaluate the performance of this controller from any initial condition using, e.g.:

$$\begin{aligned} J_\alpha(\mathbf{x}) &= \int_0^\infty \ell(\mathbf{x}(t), \mathbf{u}(t)) dt, \\ \text{subject to} \quad \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}), \quad \mathbf{u} = \pi_\alpha(\mathbf{x}), \quad \mathbf{x}(0) = \mathbf{x}. \end{aligned}$$

In order to provide a scalar cost for each set of policy parameters, $\alpha$, we need one more piece: a relative importance for the different initial conditions.

As we will further elaborate when we discuss stochastic optimal control, a very natural choice -- one that preserves the recursive structure of the HJB -- is to optimize the expected value of the cost, given some distribution over initial conditions:

$$\min_\alpha E_{\mathbf{x} \sim \mathcal{X}_0} \left[ J_\alpha(\mathbf{x}) \right],$$

where $\mathcal{X}_0$ is a probability distribution over initial conditions, $\mathbf{x}(0)$.

## 11.2 Linear Quadratic Regulator

To start thinking about the problem of searching directly in the policy parameters, it's very helpful to start with a problem we know and can understand well. In LQR problems for linear, time-invariant systems, we know that the optimal policy is a linear function: $\mathbf{u} = -\mathbf{K}\mathbf{x}$. So far, we have always obtained $\mathbf{K}$ indirectly -- by solving a Riccati equation to find the cost-to-go and then backing out the optimizing policy. Here, let us study the case where we parameterize the elements of $\mathbf{K}$ as decision variables, and attempt to optimize the expected cost-to-go directly.

### 11.2.1 Policy Evaluation

First, let's evaluate our objective for a given $\mathbf{K}$. This step is known as "*policy evaluation*". If we use a Gaussian with mean zero and covariance $\mathbf{\Omega}$ as our distribution over initial conditions, then for LQR we have

$$\begin{aligned} &E\left[\int_0^\infty [\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}] dt\right], \\ \text{subject to} \quad &\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad \mathbf{u} = -\mathbf{K}\mathbf{x}, \quad \mathbf{x}(0) \sim \mathcal{N}(0, \mathbf{\Omega}). \end{aligned}$$

This problem is also known as the $\mathcal{H}_2$ optimal control problem[1], since the expected-cost-to-go here is the $\mathcal{H}_2$-norm of the linear system from disturbance input (here only an impulse that generates our initial conditions) to performance output (which here would be e.g. $\mathbf{z} = \begin{bmatrix} \mathbf{Q}^{\frac{1}{2}}\mathbf{x} \\ \mathbf{R}^{\frac{1}{2}}\mathbf{u} \end{bmatrix}$).

To evaluate a policy $\mathbf{K}$, let us first re-arrange the cost function slightly, using the properties of the matrix trace:

$$\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{x}^T\mathbf{K}^T\mathbf{R}\mathbf{K}\mathbf{x} = \mathbf{x}^T(\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K})\mathbf{x}^T = \text{tr}\left((\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K})\mathbf{x}\mathbf{x}^T\right),$$

and the linearity of the integral and expected value:

$$E\left[\int_0^\infty \text{tr}((\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K})\mathbf{x}\mathbf{x}^T) dt\right] = \text{tr}\left((\mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K}) E\left[\int_0^\infty \mathbf{x}\mathbf{x}^T dt\right]\right),$$

For any given initial condition, the solution of the closed-loop dynamics is given by the matrix exponential:

$$\mathbf{x}(t) = e^{(\mathbf{A}-\mathbf{BK})t}\mathbf{x}(0).$$

For the distribution of initial conditions, we have

$$E\left[\mathbf{x}(t)\mathbf{x}(t)^T\right] = e^{(\mathbf{A}-\mathbf{BK})t}E\left[\mathbf{x}(0)\mathbf{x}(0)^T\right]e^{(\mathbf{A}-\mathbf{BK})^Tt} = e^{(\mathbf{A}-\mathbf{BK})t}\mathbf{\Omega}e^{(\mathbf{A}-\mathbf{BK})^Tt},$$

which is just a (symmetric) matrix function of $t$. The integral of this function, call it $\mathbf{X}$, represents the expected 'energy' of the closed-loop response:

$$\mathbf{X} = E\left[\int_0^\infty \mathbf{x}\mathbf{x}^T dt\right].$$

Assuming $\mathbf{K}$ is stabilizing, $\mathbf{X}$ can be computed as the (unique) solution to the Lyapunov equation:

$$(\mathbf{A}-\mathbf{BK})\mathbf{X} + \mathbf{X}(\mathbf{A}-\mathbf{BK})^T + \mathbf{\Omega} = 0.$$

(You can see a closely related derivation here). Finally, the total policy evaluation is given by

$$E_{\mathbf{x}\sim\mathcal{N}(0,\mathbf{\Omega})}[J_{\mathbf{K}}(\mathbf{x})] = \operatorname{tr}\left((\mathbf{Q}+\mathbf{K}^T\mathbf{R}\mathbf{K})\mathbf{X}\right). \tag{1}$$

## 11.2.2 A nonconvex objective in $\mathbb{K}$

Unfortunately, the Lyapunov equation represents a nonlinear constraint (on the pair $\mathbf{K}$, $\mathbf{X}$). Indeed, it is well known that even the set of controllers that stabilizing a linear systems can be nonconvex in the parameters $\mathbf{K}$ when there are 3 or more state variables[2].

### Example 11.1 (The set of stabilizing $\mathbb{K}$ can be non-convex)

The following example was given in [2]. Consider a discrete-time linear system with $\mathbf{A} = \mathbf{I}_{3\times 3}, \mathbf{B} = \mathbf{I}_{3\times 3}$. The controllers given by

$$\mathbf{K}_1 = \begin{bmatrix} 1 & 0 & -10 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{K}_2 = \begin{bmatrix} 1 & -10 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix},$$

are both stabilizing controllers for this system (all eigenvalues of $\mathbf{A}-\mathbf{BK}$ are inside the unit circle of the complex plane). However the controller $\hat{\mathbf{K}} = (\mathbf{K}_1 + \mathbf{K}_2)/2$ has two eigenvalues outside the unit circle.

Since the set of controllers that achieve finite total cost is non-convex, clearly the cost function we consider here is also non-convex.

As an aside, for this problem we do actually know a change of variables that make the problem convex. Let's introduce a new variable $\mathbf{Y} = \mathbf{KX}$. Since $\mathbf{X}$ is PSD, we can back out $\mathbf{K} = \mathbf{YX}^{-1}$. Now we can rewrite the optimization:

$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y}} \quad & \operatorname{tr}\mathbf{Q}^{\frac{1}{2}}\mathbf{X}\mathbf{Q}^{\frac{1}{2}} + \operatorname{tr}\mathbf{R}^{\frac{1}{2}}\mathbf{Y}\mathbf{X}^{-1}\mathbf{Y}^T\mathbf{R}^{\frac{1}{2}} \\ \text{subject to} \quad & \mathbf{A}\mathbf{X} - \mathbf{B}\mathbf{Y} + \mathbf{X}\mathbf{A}^T - \mathbf{Y}^T\mathbf{B}^T + \mathbf{\Omega} = 0, \\ & \mathbf{X} \succ 0. \end{aligned}$$

The second term in the objective appears to be nonconvex, but is actually convex. In

order to write it as a SDP, we can replace it exactly with one more slack variable, $\mathbf{Z}$, and a Schur complement[3]:

$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y},\mathbf{Z}} \quad & \operatorname{tr} \mathbf{Q}^{\frac{1}{2}} \mathbf{X} \mathbf{Q}^{\frac{1}{2}} + \operatorname{tr} \mathbf{Z} \\ \text{subject to} \quad & \mathbf{A}\mathbf{X} - \mathbf{B}\mathbf{Y} + \mathbf{X}\mathbf{A}^T - \mathbf{Y}^T\mathbf{B}^T + \mathbf{\Omega} = 0, \\ & \begin{bmatrix} \mathbf{Z} & \mathbf{R}^{\frac{1}{2}}\mathbf{Y} \\ \mathbf{Y}^T\mathbf{R}^{\frac{1}{2}} & \mathbf{X} \end{bmatrix} \succeq 0. \end{aligned}$$

Nevertheless, our original question is asking about searching directly in the original parameterization, $\mathbf{K}$. If the objective in nonconvex in those parameters, then how should we perform the search?

### 11.2.3 No local minima

Although convexity is sufficient to guarantee that an optimization landscape does not have any local minima, it is not actually necessary. [2] showed that for this LQR objective, all local optima are in fact global optima. This analysis was extended in [4] to give a simpler analysis and include convergence rates.

How does one show that an optimization landscape has no local minima (even though it may be non-convex)? One of the most popular tools is to demonstrate *gradient dominance* with the famous *Polyak-Łojasiewicz (PL) inequality* [5]. For an optimization problem

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x})$$

we first assume the function $f$ is $L$-smooth (Lipschitz gradients):

$$\forall \mathbf{x}, \mathbf{x}', \quad \|\nabla f(\mathbf{x}') - \nabla f(\mathbf{x})\|_2 \leq L \|\mathbf{x}' - \mathbf{x}\|_2.$$

Then we say that the function satisfies the PL-inequality if the following holds for some $\mu > 0$:

$$\forall \mathbf{x}, \quad \frac{1}{2} \|\nabla f(\mathbf{x})\|_F^2 \geq \mu (f(\mathbf{x}) - f^*),$$

where $f^*$ is a value obtained at the optima. In words, the gradient of the objective must grow faster than the gradient of a quadratic function. Note, however, that the distance here is measured in $f(\mathbf{x})$, not $\mathbf{x}$; we do not require (nor imply) that the optimal solution is unique. It clearly implies that for any minima, $\mathbf{x}'$ with $\nabla f(x') = 0$, since the left-hand side is zero, we must have the right-hand side also be zero, so $x'$ is also a global optima: $f(x') = f^*$.

**Example 11.2 (A nonconvex function with no local minima)**

Consider the function

$$f(x) = x^2 + 3 \sin^2(x).$$

We can establish that this function is not convex by observing that for $a = \frac{\pi}{4}$, $b = \frac{3\pi}{4}$, we have

$$f\left(\frac{a+b}{2}\right) = \frac{\pi^2}{4} + 3 \approx 5.47 > \frac{f(a)+f(b)}{2} = \frac{5\pi^2}{16} + \frac{3}{2} \approx 4.58.$$

We can establish the PL conditions using the gradient

$$\nabla f(x) = 2x + 6\sin(x)\cos(x).$$

We can establish that this function is $L$-smooth with $L = 8$ by

$$\|\nabla f(b) - \nabla f(a)\|_2 = |2b - 2a + 6\sin(b-a)| \leq 8|b-a|,$$

because $\sin(x) \leq x$. Finally, we have gradient-dominance from the PL-inequality:

$$\frac{1}{2}(2x + 6\sin(x)\cos(x))^2 \geq \mu(x^2 + 3\sin^2(x)),$$

with $\mu = 0.175$. (I confirmed this with a small dReal program).

[5] gives a convergence rate for convergence to an optima for gradient descent given the PL conditions. [4] showed that the gradients of the LQR cost we examine here with respect to $\mathbf{K}$ satisfy the PL conditions on any sublevel set of the cost-to-go function.

### 11.2.4 True gradient descent

The results described above suggest that one can use gradient descent to obtain the optimal controller, $\mathbf{K}^*$ for LQR. For the variations we've seen so far (where we know the model), I would absolutely recommend that solving the Riccati equations is a much better algorithm; it is faster and more robust, with no parameters like step-size to tune. But gradient descent becomes more interesting / viable when we think of it as a model for a less perfect algorithm, e.g. where the plant model is not given and the gradients are estimated from noisy samples.

It is a rare luxury, due here to our ability to integrate the linear plants/controllers, quadratic costs, and Gaussian initial conditions, that we could compute the value function exactly in (1). We can also compute the true gradient -- this is a pinnacle of exactness we should strive for in our methods but will rarely achieve again. The gradient is given by

$$\frac{\partial E[J_{\mathbf{K}}(\mathbf{x})]}{\partial \mathbf{K}} = 2(\mathbf{R}\mathbf{K} - \mathbf{B}^T\mathbf{P})\mathbf{X},$$

where $\mathbf{P}$ satisfies another Lyapunov equation:

$$(\mathbf{A} - \mathbf{B}\mathbf{K})^T\mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{B}\mathbf{K}) + \mathbf{Q} + \mathbf{K}^T\mathbf{R}\mathbf{K} = 0.$$

Note that the term *policy gradient* used in reinforcement learning typically refers to the slightly different class of algorithms I hinted at above. In those algorithms, we use the true gradients of the policy (only), but estimate the remainder of the terms in the gradient through sampling. These methods typically require many samples to estimate the gradients we compute here, and should only be weaker (less efficient) than the algorithms in this chapter. The papers investigating the convergence of gradient descent for LQR have also started exploring these cases. We will study these so-called model-free" policy search algorithms soon.

## 11.3 More convergence results and counter-examples

LQR / $\mathcal{H}_2$ control is one of the good cases, where we know that for the objective parameterized directly in $\mathbf{K}$, all local optima are global optima. [6] extended this result for mixed $\mathcal{H}_2/\mathcal{H}_\infty$ control. [7] gives a recent treatment of the tabular (finite) MDP case.

For LQR, we also know alternative parameterizations of the controller which make the objective actually convex, including the LMI formulation and the Youla parameterization. Their utility in a policy search setting was studied initially in [8].

Unfortunately, we do not expect these nice properties to hold in general. There are a number of nearby problems which are known to be nonconvex in the original parameters. The case of *static output feedback* is an important one. If we extend our plant model to include (potentially limited) observations: $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \mathbf{y} = \mathbf{C}\mathbf{x}$, then searching directly over controllers, $\mathbf{u} = -\mathbf{K}\mathbf{y}$, is known to be NP-hard[9]. This time, the set of stabilizing $\mathbf{K}$ matrices may be not only nonconvex, but actually disconnected. We can see that with a simple example (given to me once during a conversation with Alex Megretski).

**Example 11.3 (Parameterizations of Static Output Feedback)**

Consider the single-input, single-output LTI system

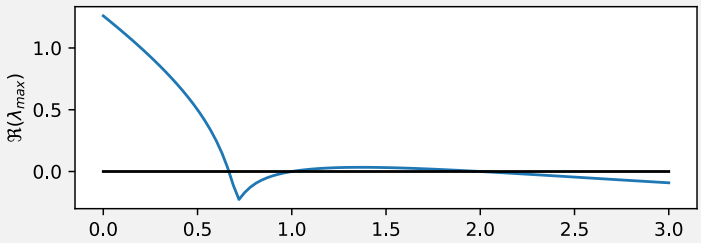
$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u, \quad y = \mathbf{C}\mathbf{x},$$

with

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 1 & 3 \end{bmatrix}.$$

Here the linear static-output-feedback policy can be written as $u = -ky$, with a single scalar parameter $k$.

Here is a plot of the maximum eigenvalue (real-part) of the closed-loop system, as a function of $k$. The system is only stable when this maximum value is less than zero. You'll find the set of stabilizing $k$'s is a disconnected set.




There are many other known counter-examples. [10] gives a particularly simple one with a two state MDP. One of the big open questions is whether deep network parameterizations are an example of a good case.

## 11.4 TRAJECTORY-BASED POLICY SEARCH

Although we cannot expect gradient descent to converge to a global minima in general, it is still very reasonable to try using gradient descent to find policies for more complicated nonlinear control problems. In the general form, this means that the first step of optimizing

$$\min_{\alpha} E_{\mathbf{x} \sim \mathcal{X}_0} [J_\alpha(\mathbf{x})],$$

is estimating

$$\frac{\partial}{\partial \alpha} E_{\mathbf{x} \sim \mathcal{X}_0} [J_\alpha(\mathbf{x})].$$

In the LQR problem, we were able to compute these terms exactly; with the biggest simplification coming from the fact that the response of a linear system to Gaussian initial conditions stays Gaussian. This is not true for more general nonlinear systems. So what are we to do?

The most common/general technique (despite it not being very efficient), is to approximate the expected cost-to-go using a sampling (Monte-Carlo) approximation using a large number, $N$, of samples:

$$E_{\mathbf{x} \sim \mathcal{X}_0}[J_\alpha(\mathbf{x})] \approx \frac{1}{N} \sum_{i=0}^{N-1} J_\alpha(\mathbf{x}_i), \quad \mathbf{x}_i \sim \mathcal{X}_0.$$

The gradients follow easily:

$$\frac{\partial}{\partial \alpha} E_{\mathbf{x} \sim \mathcal{X}_0}[J_\alpha(\mathbf{x})] \approx \frac{1}{N} \sum_{i=0}^{N-1} \frac{\partial J_\alpha(\mathbf{x}_i)}{\partial \alpha}, \quad \mathbf{x}_i \sim \mathcal{X}_0.$$

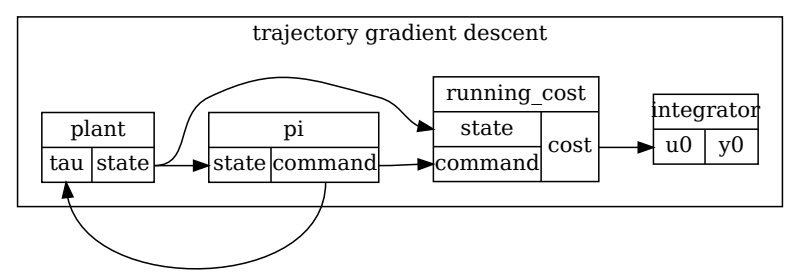
Our confidence in the accuracy of this estimate will improve as we increase $N$; see

e.g. Section 4.2 of [11] for details on the confidence intervals. The most optimistic among us will say that it's quite fine to have only a noisy estimate of the gradient -- this leads to stochastic gradient descent which can have some very desirable properties. But it does make the algorithm harder to analyze and debug.

Using the Monte-Carlo estimator, the total gradient update is just a sum over gradients with respect to particular initial conditions, $\frac{\partial J_\alpha(\mathbf{x}_i)}{\partial \alpha}$. But for finite-horizon objectives, these are precisely the gradients that we have already studied in the context of trajectory optimization. They can be computed efficiently using an adjoint method. The difference is that here we think of $\alpha$ as the parameters of a feedback controller, whereas before we thought of them as the parameters of a trajectory, but this makes no difference to the chain rule.

**Example 11.4 (LQR with true gradients vs approximate gradients)**

**Example 11.5 (Pendulum Swing-Up And Balance)**




Open in Colab

### 11.4.1 Infinite-horizon objectives

### 11.4.2 Search strategies for global optimization

To combat local minima... Evolutionary strategies, ...

## 11.5 Policy Iteration

In the chapter on Lyapunov analysis we also explored a handful techniques for control design. One of these even directly parameterized the controller (as a polynomial feedback), and used alternations to switch between policy evaluation and policy optimization. Another generated lower-bounds to the optimal cost-to-go.

Coming soon...

## References


1. Ben M. Chen, "H2 {Optimal} {Control}", *Encyclopedia of {Systems} and {Control}*, pp. 515--520, 2015.

2. Maryam Fazel and Rong Ge and Sham M. Kakade and Mehran Mesbahi, "Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator", *International Conference on Machine Learning*, 2018.

3. Erik A Johnson and Baris Erkus, "Dissipativity and performance analysis of smart dampers via LMI synthesis", *Structural Control and Health Monitoring: The Official Journal of the International Association for Structural Control and Monitoring and of the European Association for the Control of Structures*, vol. 14, no. 3, pp. 471--496, 2007.

4. Hesameddin Mohammadi and Armin Zare and Mahdi Soltanolkotabi and Mihailo R Jovanovi{\'c}, "Global exponential convergence of gradient methods over the nonconvex landscape of the linear quadratic regulator", *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 7474--7479, 2019.

5. Hamed Karimi and Julie Nutini and Mark Schmidt, "Linear convergence of gradient and proximal-gradient methods under the polyak-{\l}ojasiewicz condition", *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795--811, 2016.

6. Kaiqing Zhang and Bin Hu and Tamer Basar, "Policy Optimization for $\mathcal{H}_2$ Linear Control with $\mathcal{H}_\infty$ Robustness Guarantee: Implicit Regularization and Global Convergence", *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, vol. 120, pp. 179--190, 10--11 Jun, 2020.

7. Alekh Agarwal and Sham M. Kakade and Jason D. Lee and Gaurav Mahajan, "On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift", , 2020.

8. John Roberts and Ian Manchester and Russ Tedrake, "Feedback Controller Parameterizations for Reinforcement Learning", *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2011. [ link ]

9. Vincent Blondel and John N Tsitsiklis, "NP-hardness of some linear control design problems", *SIAM journal on control and optimization*, vol. 35, no. 6, pp. 2118--2127, 1997.

10. Jalaj Bhandari and Daniel Russo, "Global {Optimality} {Guarantees} {For} {Policy} {Gradient} {Methods}", *arXiv:1906.01786 [cs, stat]*, oct, 2020.

11. Reuven Y Rubinstein and Dirk P Kroese, "Simulation and the Monte Carlo method",John Wiley \& Sons , vol. 10, 2016.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 12
# Motion Planning as Search

The term "motion planning" is a hopelessly general term which almost certainly encompasses the dynamic programming, feedback design, and trajectory optimization algorithms that we have already discussed. However, there are a number of algorithms and ideas that we have not yet discussed which have grown from the idea of formulating motion planning as a search problem -- for instance searching for a path from a start to a goal in a graph which is too large solve completely with dynamic programming. Some, but certainly not all, of these algorithms sacrifice optimality in order to find any path if it exists, and the notion of a planner being "complete" -- guaranteed to find a path if one exists -- is highly valued. This is precisely our goal for this chapter, to add some additional tools that will be able to provide some form of solutions for our most geometrically complex, highly non-convex, robot control problems.

[1] is a very nice book on planning algorithms in general and on motion planning algorithms in particular. Compared to other planning problems, *motion planning* typically refers to problems where the planning domain is continuous (e.g. continuous state space, continuous action space), but many motion planning algorithms trace their origins back to ideas in discrete domains (e.g., graph search).

For this chapter, we will consider the following problem formulation: given a system defined by the nonlinear dynamics (in continuous- or discrete-time)

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \quad \text{or} \quad \mathbf{x}[n+1] = f(\mathbf{x}[n], \mathbf{u}[n]),$$

and given a start state $\mathbf{x}(0) = \mathbf{x}_0$ and a goal region $\mathcal{G}$, find any finite-time trajectory from $\mathbf{x}_0$ to to $\mathbf{x} \in \mathcal{G}$ if such a trajectory exists.

## 12.1 Artificial Intelligence as Search

A long history... some people feel that the route to creating intelligent machines is to collect large ontologies of knowledge, and then perform very efficient search. (The more modern view of AI is focused instead on machine learning, the right answer probably involves pieces of both.) Samuels checker players, Deep Blue playing chess, theorem proving, Cyc, IBM Watson,...

One of the key ideas is the use of "heuristics" to guide the search. "Is it possible to find and optimal path from the start to a goal without visiting every node?". $A^*$. Admissible heuristics. Example: google maps.

Online planning. $D^*$, $D^*$-Lite.

## 12.2 Randomized motion planning

If you remember how we introduced dynamic programming initially as a graph search, you'll remember that there were some challenges in discretizing the state space. Let's assume that we have discretized the continuous space into some finite set of discrete nodes in our graph. Even if we are willing to discretize the action space for the robot (this might be even be acceptable in practice), we had a problem where discrete actions from one node in the graph, integrated over some finite interval $h$, are extremely unlikely to land exactly on top of another node in the graph. To combat this, we had to start working on methods for interpolating the value function estimate between nodes.

Interpolation can work well if you are trying to solve for the cost-to-go function over the entire state space, but it's less compatible with search methods which are trying to find just a single path through the space. If I start in node 1, and land between node 2 and node 3, then which node to I continue to expand from?

One approach to avoiding this problem is to build a *search tree* as the search executes, instead of relying on a predefined mesh discretization. This tree will contains nodes rooted in the continuous space at exactly the points where system can reach.

Another other problem with any fixed mesh discretization of a continuous space, or even a fixed discretization of the action space, is that unless we have specific geometric / dynamic insights into our continuous system, it very difficult to provide a *complete* planning algorithm. Even if we can show that no path to the goal exists on the tree/graph, how can we be certain that there is no path for the continuous system? Perhaps a solution would have emerged if we had discretized the system differently, or more finely?

One approach to addressing this second challenge is to toss out the notion of fixed discretizations, and replace them with random sampling (another approach would be to adaptively add resolution to the discretization as the algorithm runs). Random sampling, e.g. of the action space, can yield algorithms that are *probabilistically complete* for the continuous space -- if a solution to the problem exists, then a probabilistically complete algorithm will find that solution with probability 1 as the number of samples goes to infinity.

With these motivations in mind, we can build what is perhaps the simplest probabilistically complete algorithm for finding a path from the a starting state to some goal region with in a continuous state and action space:

### Example 12.1 (Planning with a Random Tree)

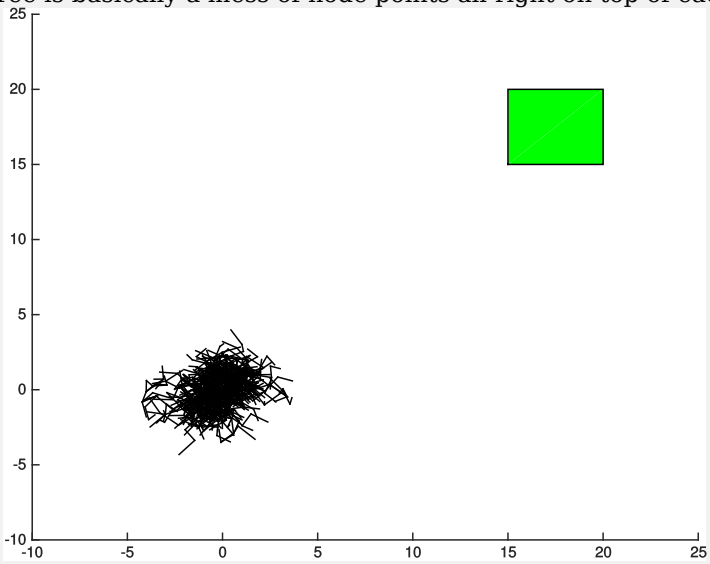
Let us denote the data structure which contains the tree as $\mathcal{T}$. The algorithm is very simple:

- Initialize the tree with the start state: $\mathcal{T} \leftarrow \mathbf{x}_0$.
- On each iteration:
    - Select a random node, $\mathbf{x}_{rand}$, from the tree, $\mathcal{T}$
    - Select a random action, $\mathbf{u}_{rand}$, from a distribution over feasible actions.
    - Compute the dynamics: $\mathbf{x}_{new} = f(\mathbf{x}_{rand}, \mathbf{u}_{rand})$
    - If $\mathbf{x}_{new} \in \mathcal{G}$, then terminate. Solution found!
    - Otherwise add the new node to the tree, $\mathcal{T} \leftarrow \mathbf{x}_{new}$.

It can be shown that this algorithm is, in fact, probabilistically complete. However, without strong heuristics to guide the selection of the nodes scheduled for expansion, it can be extremely inefficient. For a simple example, consider the system $\mathbf{x}[n] = \mathbf{u}[n]$ with $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{u}_i \in [-1, 1]$. We'll start at the origin and put the goal region as $\forall i, 15 \leq x_i \leq 20$. Try it yourself:

```
T = struct('parent',zeros(1,1000),'node',zeros(2,1000));  % pre-allocate memory for the "tr
for i=2:size(T.parent,2)
T.parent(i) = randi(i-1);
x_rand = T.node(:,T.parent(i));
u_rand = 2*rand(2,1)-1;
x_new = x_rand+u_rand;
if (15<=x_new(1) && x_new(1)<=20 && 15<=x_new(2) && x_new(2)<=20)
  disp('Success!'); break;
end
T.node(:,i) = x_new;
end
clf;
line([T.node(1,T.parent(2:end));T.node(1,2:end)],[T.node(2,T.parent(2:end));T.node(2,2:end)
patch([15,15,20,20],[15,20,20,15],'g')
axis([-10,25,-10,25]);
```

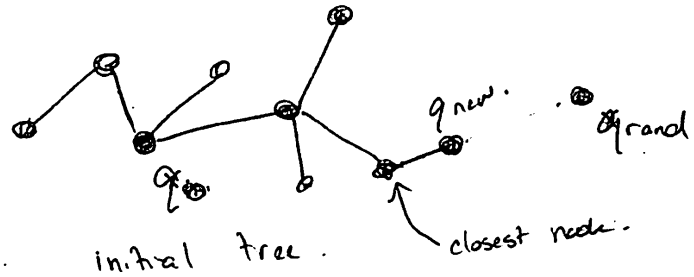
Again, this algorithm *is* probabilistically complete. But after expanding 1000 nodes, the tree is basically a mess of node points all right on top of each other:




We're nowhere close to the goal yet, and it's not exactly a hard problem.

While the idea of generating a tree of feasible points has clear advantages, we have lost the ability to cross off a node (and therefore a region of space) once it has been explored. It seems that, to make randomized algorithms effective, we are going to at the very least need some form of heuristic for encouraging the nodes to spread out and explore the space.

### 12.2.1 Rapidly-Exploring Random Trees (RRTs)



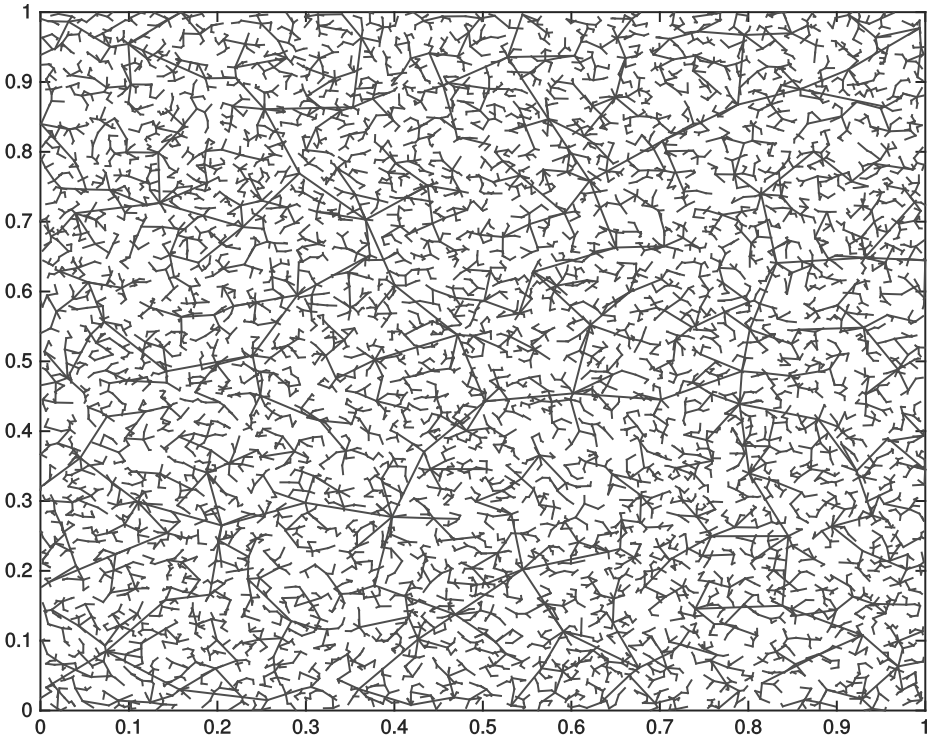




Figure 12.3 - (Click here to watch the animation)

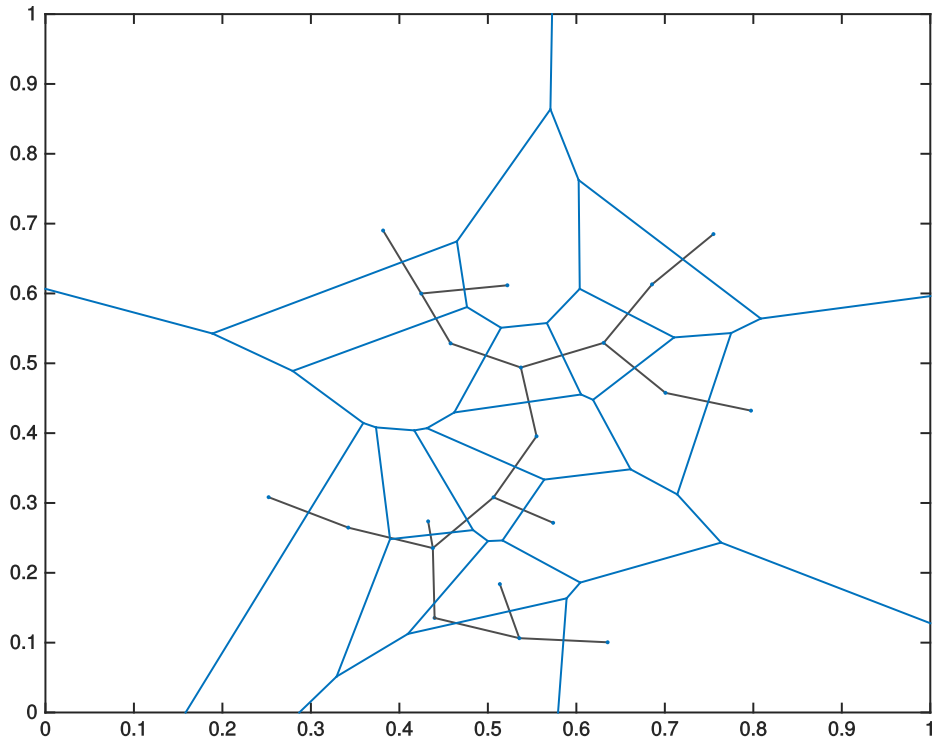


Figure 12.4 - (Click here to watch the animation)

### 12.2.2 RRTs for robots with dynamics

### 12.2.3 Variations and extensions

Multi-query planning with PRMs, ...

RRT*, RRT-sharp, RRTx, ...

Kinodynamic-RRT*, LQR-RRT(*)

Complexity bounds and dispersion limits

### 12.2.4 Discussion

Not sure yet whether randomness is fundamental here, or whether is a temporary "crutch" until we understand geometric and dynamic planning better.

## 12.3 Decomposition methods

Cell decomposition...

Mixed-integer planning.

Approximate decompositions for complex environments (e.g. IRIS)

## 12.4 Exercises

### Exercise 12.1 (RRT Planning)

In this notebook we will write code for the Rapidly-Exploring Random Tree (RRT). Building on this implementation we will also implement RRT*, a variant of RRT that converges towards an optimal solution.

a. Implement RRT
b. Implement RRT*

## References


1. Steven M. LaValle, "Planning Algorithms",Cambridge University Press , 2006.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.



# Chapter 13
# Feedback Motion Planning

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 14
# Robust and Stochastic Control

So far in the notes, we have concerned ourselves with only known, deterministic systems. In this chapter we will begin to consider uncertainty. This uncertainy can come in many forms... we may not know the governing equations (e.g. the coefficient of friction in the joints), our robot may be walking on unknown terrain, subject to unknown disturbances, or even be picking up unknown objects. There are a number of mathematical frameworks for considering this uncertainty; for our purposes this chapter will generalizing our thinking to equations of the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t) \qquad \text{or} \qquad \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n], \mathbf{w}[n], n),$$

where $\mathbf{w}$ is a new *random* input signal to the equations capturing all of this potential variability. Although it is certainly possible to work in continuous time, and treat $\mathbf{w}(t)$ as a continuous-time random signal (c.f. Wiener process), it is notationally simpler to work with $\mathbf{w}[n]$ as a discrete-time random signal. For this reason, we'll devote our attention in this chapter to the discrete-time systems.

In order to simulate equations of this form, or to design controllers against them, we need to define the random process that generates $\mathbf{w}[n]$. It is typical to assume the values $\mathbf{w}[n]$ are independent and identically distributed (i.i.d.), meaning that $\mathbf{w}[i]$ and $\mathbf{w}[j]$ are uncorrelated when $i \neq j$. As a result, we typically define our distribution via a probability density $p_{\mathbf{w}}(\mathbf{w}[n])$. This is not as limiting as it may sound... if we wish to treat temporally-correlated noise (e.g. "colored noise") the format of our equations is rich enough to support this by adding additional state variables; we'll give an example below of a "whitening filter" for modeling wind gusts. The other source of randomness that we will now consider in the equations is randomness in the initial conditions; we will similarly define a probabilty density $p_{\mathbf{x}}(\mathbf{x}[0])$.

This modeling framework is rich enough for us to convey the key ideas; but it is not quite sufficient for all of the systems I am interested in. In DRAKE we go to additional lengths to support more general cases of stochastic systems. This includes modeling system parameters that are drawn from random each time the model is initialized, but are fixed over the duration of a simulation; it is possible but inefficient to model these as additional state variables that have no dynamics. In other problems, even the *dimension of the state vector* may change in different realizations of the

problem! Consider, for instance, the case of a robot manipulating random numbers of dishes in a sink. I do not know many control formulations that handle this type of randomness well, and I consider this a top priority to think more about! (We'll begin to address it in the output feedback chapter.)

Roughly speaking, I will refer to "stochastic control" as the discipline of synthesizing controllers that govern the probabilistic evolution of the equations. "Stochastic optimal control" defines a cost function (now a random variable), and tries to find controllers that optimize some metric such as the expected cost. When we use the terms "robust control", we are typically referring to a class of techniques that try to guarantee a worst-case performance or a worst-case bound on the effect of randomness on the input on the randomness on the output. Interestingly, for many robust control formulations we do not attempt to know the precise probability distribution of $\mathbf{x}[0]$ and $\mathbf{w}[n]$, but instead only define the *sets* of possible values that they can take. This modeling is powerful, but can lead to conservative controllers and pessimistic estimates of performance.

My goal of presenting a relatively consumable survey of a few of the main ideas is perhaps more important in this chapter than any other. It's been said that "robust control is encrypted" (as in you need to know the secret code to get in). The culture in the robust control community has been to leverage high-powered mathematics, sometimes at the cost of offering more simple explanations. This is unfortunate, I think, because robotics and machine learning would benefit from a richer connection to these tools, and are perhaps destined to reinvent many of them.

The classic reference for robust control is [1]. [2] has a treatment that does more of it's development in the time-domain and via Riccati equations.

## 14.1 Finite Markov Decision Processes

We already had quick preview into stochastic optimal control in one of the cases where it is particularly easy: finite Markov Decision Processes (MDPs).

## 14.2 Linear optimal control

### 14.2.1 Analysis

### Common Lyapunov functions

We've already seen one nice example of robustness analysis for linear systems when we wrote a small optimization to find a common Lyapunov function for uncertain linear systems. That example studied the dynamics $\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n]$ where the coefficients of $\mathbf{A}$ were unknown but bounded.

We also saw that essentially the same technique can be used to certify stability against disturbances, e.g.:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{w}[n], \qquad \mathbf{w}[n] \in \mathcal{W},$$

where $\mathcal{W}$ describes some bounded set of possible uncertainties. In order to be compatible with convex optimization, we often choose to describe $\mathcal{W}$ as either an ellipsoid or as a convex polytope[3]. But let's remember a very important point: in this situation with additive disturbances, we can no-longer expect the system to converge stably to the origin. Our example before used the common Lyapunov function only to certify the *invariance* of a region (if I start inside some region, then I'll never leave).

### $L_2$ gain

In some sense, the common-Lyapunov analysis above is probably the wrong analysis

for linear systems (perhaps other systems as well). It might be unreasonable to assume that disturbances are bounded. Moreover, we know that the response to an input (including the disturbance input) is linear, so we expect the magnitude of the deviation in $\mathbf{x}$ compared to the undisturbed case to be proportional to the magnitude of the disturbance, $\mathbf{w}$. A more natural bound for a linear system's response is to bound the magnitude of the response (from zero initial conditions) relative to the magnitude of the disturbance.

Typically, this is done with the a scalar "$L_2$ gain", $\gamma$, defined as:

$$\text{argmin}_\gamma \quad \text{subject to} \quad \sup_{\mathbf{w}(\cdot) \in \int \|\mathbf{w}(t)\|^2 dt \leq \infty} \frac{\int_0^T \|\mathbf{x}(t)\|^2 dt}{\int_0^T \|\mathbf{w}(t)\|^2 dt} \leq \gamma^2, \qquad \text{or}$$

$$\text{argmin}_\gamma \quad \text{subject to} \quad \sup_{\mathbf{w}[\cdot] \in \sum_n \|\mathbf{w}[n]\|^2 \leq \infty} \frac{\sum_0^N \|\mathbf{x}[n]\|^2}{\sum_0^N \|\mathbf{w}[n]\|^2} \leq \gamma^2.$$

The name "$L_2$ gain" comes from the use of the $\ell_2$ norm on the signals $\mathbf{w}(t)$ and $\mathbf{x}(t)$, which is assumed only to be finite.

More often, these gains are written not in terms of $\mathbf{x}[n]$ directly, but in terms of some "performance output", $\mathbf{z}[n]$. For instance, if would would like to bound the cost of a quadratic regulator objective as a function of the magnitude of the disturbance, we can minimize

$$\min_\gamma \quad \text{subject to} \quad \sup_{\mathbf{w}[n]} \frac{\sum_0^N \|\mathbf{z}[n]\|^2}{\sum_0^N \|\mathbf{w}[n]\|^2} \leq \gamma^2, \qquad \mathbf{z}[n] = \begin{bmatrix} \sqrt{\mathbf{Q}}\mathbf{x}[n] \\ \sqrt{\mathbf{R}}\mathbf{u}[n] \end{bmatrix}.$$

This is a simple but important idea, and understanding it is the key to understanding the language around robust control. In particular the $H_2$ norm of a system (from input $\mathbf{w}$ to output $\mathbf{z}$) is the energy of the impulse response; when $\mathbf{z}$ is chosen to represent the quadratic regulator cost as above, it corresponds to the expected LQR cost. The $H_\infty$ norm of a system (from $\mathbf{w}$ to $\mathbf{z}$) is the largest singular value of the transfer function; it corresponds to the $L_2$ gain.

## Small-gain theorem

Coming soon...

## Dissipation inequalities

Coming soon... See, for instance, [4] or [5, Ch. 2].

## 14.2.2 $H_2$ design

## 14.2.3 $H_\infty$ design

## 14.2.4 Linear Exponential-Quadratic Gaussian (LEQG)

[6] observed that it is also straight-forward to minimize the objective:

$$J = E\left[\prod_{n=0}^{\infty} e^{\mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n]} e^{\mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n]}\right] = E\left[e^{\sum_{n=0}^{\infty} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n]}\right],$$

with $\mathbf{Q} = \mathbf{Q}^T \geq \mathbf{0}, \mathbf{R} = \mathbf{R}^T > 0$, by observing that the cost is monotonically related to $\log J$, and therefore has the same minima (this same trick forms the basis for "geometric programming" [7]). This is known as the "Linear Exponential-Quadratic Gaussian" (LEG or LEQG), and for the deterministic version of the problem (no process nor measurement noise) the solution is identical to the LQR problem; it adds no new modeling power. But with noise, the LEQG optimal controllers are different

from the LQG controllers; they depend explicitly on the covariance of the noise.

[8] provides an extensive treatment of this framework; nearly all of the analysis from LQR/LQG (including Riccati equations, Hamiltonian formulations, etc) have analogs for their versions with exponential cost, and he argues that LQG and H-infinity control can (should?) be understood as special cases of this approach.

### 14.2.5 Adaptive control

The standard criticism of $H_2$ optimal control is that minimizing the expected value does not allow any guarantees on performance. The standard criticism of $H_\infty$ optimal control is that it concerns itself with the worst case, and may therefore be conservative, especially because distributions over possible disturbances chosen a priori may be unnecessarily conservative. One might hope that we could get some of this performance back if we are able to update our models of uncertainty online, adapting to the statistics of the disturbances we actually receive. This is one of the goals of adaptive control.

One of the fundamental problems in online adaptive control is the trade-off between exploration and exploitation. Some inputs might drive the system to build more accurate models of the dynamics / uncertainty quickly, which could lead to better performance. But how can we formalize this trade-off?

There has been some nice progress on this challenge in machine learning in the setting of (contextual) multi-armed bandit problems. For our purposes, you can think of bandits as a limiting case of an optimal control problem where there are no dynamics (the effects of one control action do not effect the results of the next control action). In this simpler setting, the online optimization community has developed exploration-exploitation strategies based on the notion of minimizing *regret* -- typically the accumulated difference in the performance achieved by my online algorithm vs the performance that would have been achieved if I had been privy to the data from my experiments before I started. This has led to methods that make use of concepts like upper-confidence bound (UCB) and more recently bounds using a least-squares squares confidence bound [9] to provide bounds on the regret.

In the last few years, we've see these results translated into the setting of linear optimal control...

### 14.2.6 Structured uncertainty

### 14.2.7 Linear parameter-varying (LPV) control

## 14.3 Trajectory optimization

### 14.3.1 Monte-carlo trajectory optimization

### 14.3.2 Iterative $H_2$

### 14.3.3 Finite-time (reachability) analysis

Coming soon...

### 14.3.4 Robust MPC

## 14.4 Nonlinear analysis and control

## 14.5 Domain randomization

## 14.6 Extensions

### 14.6.1 Alternative risk/robustness metrics

## References


1. Kemin Zhou and John C. Doyle, "Essentials of Robust Control",Prentice Hall , 1997.

2. I. R. Petersen and V. A. Ugrinovskii and A. V. Savkin, "Robust Control Design using H-infinity Methods",Springer-Verlag , 2000.

3. Stephen Boyd and Lieven Vandenberghe, "Convex Optimization",Cambridge University Press , 2004.

4. Christian Ebenbauer and Tobias Raff and Frank Allgower, "Dissipation inequalities in systems theory: An introduction and recent results", *Invited Lectures of the International Congress on Industrial and Applied Mathematics 2007*, pp. 23-42, 2009.

5. Carsten Scherer and Siep Weiland, "Linear {Matrix} {Inequalities} in {Control}",Online Draft , pp. 293, 2015.

6. D. Jacobson, "Optimal stochastic linear systems with exponential performance criteria and their relation to deterministic differential games", *IEEE Transactions on Automatic Control*, vol. 18, no. 2, pp. 124--131, apr, 1973.

7. S. Boyd and S.-J. Kim and L. Vandenberghe and A. Hassibi, "A Tutorial on Geometric Programming", *Optimization and Engineering*, vol. 8, no. 1, pp. 67-127, 2007.

8. Peter Whittle, "Risk-sensitive optimal control",Wiley New York , vol. 20, 1990.

9. Dylan Foster and Alexander Rakhlin, "Beyond {UCB}: Optimal and Efficient Contextual Bandits with Regression Oracles", *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, pp. 3199--3210, 13--18 Jul, 2020.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 15
# Output Feedback (aka Pixels-to-Torques)

In this chapter we will start considering systems of the form:

$$\mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n], \mathbf{w}[n], n)$$
$$\mathbf{y}[n] = \mathbf{g}(\mathbf{x}[n], \mathbf{u}[n], \mathbf{v}[n], n).$$

In other words, we'll finally start addressing the fact that we have to make decisions based on sensor measurements -- most of our discussions until now have tacitly assumed that we have access to the true state of the system for use in our feedback controllers (and that's already been a hard problem).

In some cases, we will see that the assumption of "full-state feedback" is not so bad -- we do have good tools for state estimation from raw sensor data. But even our best state estimation algorithms do add some dynamics to the system in order to filter out noisy measurements; if the time constants of these filters is near the time constant of our dynamics, then it becomes important that we include the dynamics of the estimator in our analysis of the closed-loop system.

In other cases, it's entirely too optimistic to design a controller assuming that we will have an estimate of the full state of the system. Some state variables might be completely unobservable, others might require specific "information-gathering" actions on the part of the controller.

For me, the problem of robot manipulation is the application domain where more direct approaches to output feedback become critically important. Imagine you are trying to design a controller for a robot that needs to button the buttons on your dress shirt. If step one is to estimate the state of the shirt (how many degrees of freedom does my shirt have?), then it feels like we're not going to be successful. Or if you want to program a robot to make a salad -- what's the state of the salad? Do I really need to know the positions and velocities of every piece of lettuce in order to be successful?

## 15.1 The Classical Perspective

To some extent, this idea of calling out "output feedback" as a special, advanced topic is a new problem. Before state space and optimization-based approaches to control ushered in "modern control", we had "classical control". Classical control focused predominantly (though not exclusively) on linear time-invariant (LTI) systems, and made very heavy use of frequency-domain analysis (e.g. via the Fourier Transform/Laplace Transform). There are many excellent books on the subject; [1, 2] are nice examples of modern treatments that start with state-space representations but also treat the frequency-domain perspective.

What's important for us to acknowledge here is that in classical control, basically everything was built around the idea of output feedback. The fundamental concept is the transfer function of a system, which is a input-to-output map (in frequency domain) that can completely characterize an LTI system. Core concepts like pole placement and loop shaping were fundamentally addressing the challenge of output feedback that we are discussing here. Sometimes I feel that, despite all of the things we've gain with modern, optimization-based control, I worry that we've lost something in terms of considering rich characterizations of closed-loop performance (rise time, dwell time, overshoot, ...) and perhaps even in practical robustness of our systems to unmodeled errors.

## 15.2 Observer-based Feedback

### 15.2.1 Luenberger Observer

### 15.2.2 Linear Quadratic Regulator w/ Gaussian Noise (LQG)

### 15.2.3 Partially-observable Markov Decision Processes

## 15.3 Static Output Feedback

### 15.3.1 For Linear Systems

## 15.4 Disturbance-based feedback

### 15.4.1 System-Level Synthesis

## References


1. Joao P. Hespanha, "Linear Systems Theory",Princeton Press , 2009.

2. Karl Johan {\AA}str{\"o}m and Richard M Murray, "Feedback systems: an introduction for scientists and engineers",Princeton university press , 2010.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 16
# Algorithms for Limit Cycles


The discussion of walking and running robots in Chapter 4 motivated the notion of limit cycle stability. Linear systems are not capable of producing stable limit cycle behavior, so this rich topic is unique to nonlinear systems design and analysis. Furthermore, the tools that are required to design, stabilize, and verify limit cycles will have applicability beyond simple periodic motions.

The first natural question we must ask is, given a system $\dot{\mathbf{x}} = f(\mathbf{x})$, or a control system $\dot{x} = f(\mathbf{x}, \mathbf{u})$, how do we go about finding periodic solutions which may be passively stable, open-loop stable, or stabilizable via closed-loop feedback? It turns out that the trajectory optimization tools that we developed already are very well suited to this task.

## 16.1 TRAJECTORY OPTIMIZATION

I introduced the trajectory optimization tools as means for optimizing a control trajectory starting from a particular known initial condition. But the fundamental idea of optimizing over individual trajectories of the system is useful more broadly. Even for a passive system, we can formulate the search for a periodic solution as an optimization over trajectories that satisfy the dynamic constraints and *periodicity constraints*, $\mathbf{x}[0] = \mathbf{x}[N]$:

$$\begin{aligned} \underset{\mathbf{x}[\cdot]}{\text{find}} \quad \text{subject to} \quad & \mathbf{x}[n+1] = f(\mathbf{x}[n]), \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}[N]. \end{aligned}$$

Certainly we can add control inputs back into the formulation, too, but let's start with this simple case. Take a moment to think about the feasible solutions to this problem formulation. Certainly a fixed point $\mathbf{x}[n] = \mathbf{x}^*$ will satisfy the constraints; if we don't want these solutions to come out of the solver we might need to exclude them with constraints or add an objective that guides the solver towards the desired solutions. The other possible solutions are trajectories that are periodic in exactly $N$ steps. That's pretty restrictive.

We can do better if we use the continuous-time formulations. For instance, in our

introduction of direct collocation, we wrote

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} h_n \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \dot{\mathbf{x}}(t_{c,n}) = f(\mathbf{x}(t_{c,n}), \mathbf{u}(t_{c,n})), \qquad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints.} \end{aligned}$$

But we can also add $h_n$ as decision variables in the optimization (reminder: I recommend setting a lower-bound $h_n \geq h_{min} > 0$). This allows our $N$-step trajectory optimization to scale and shrink time in order to satisfy the periodicity constraint. The result is simple and powerful.

### Example 16.1 (Finding the limit cycle of the Van der Pol oscillator)

Recall the dynamics of the Van der Pol oscillator given by

$$\ddot{q} + \mu(q^2 - 1)\dot{q} + q = 0, \quad \mu > 0,$$

which exhibited a stable limit cycle.

Formulate the direct collocation optimization:

$$\begin{aligned} \underset{\mathbf{x}[\cdot],h}{\text{find}} \quad \text{subject to} \quad & q[0] = 0, \quad \dot{q}[0] > 0, \\ & \mathbf{x}[N] = \mathbf{x}[0], \text{(periodicity constraint)} \\ & \text{collocation dynamic constraints} \\ & 0.01 \leq h \leq 0.5 \end{aligned}$$

Try it yourself:


As always, make sure that you take a look at the code. Poke around. Try changing some things.

One of the things that you should notice in the code is that I provide an initial guess for the solver. In most of the examples so far I've been able to avoid doing that--the solver takes small random numbers as a default initial guess and solves from there. But for this problem, I found that it was getting stuck in a local minima. Adding the initial guess that the solution moves around a circle in state space was enough.

## 16.2 Lyapunov Analysis

Recall the important distinction between stability of a trajectory in time and stability of a limit cycle was that the limit cycle does not converge in phase -- trajectories near the cycle converge to the cycle, but trajectories on the cycle may not converge with each other. This is type of stability, also known as *orbital stability* can be written as stability to the manifold described by a trajectory $\mathbf{x}^*(t)$,

$$\min_{\tau} ||x(t) - x^*(\tau)|| \to 0.$$

In the case of limit cycles, this manifold is a periodic solution with $\mathbf{x}^*(t + t_{period}) = \mathbf{x}^*(t)$. Depending on exactly how that convergence happens, we can define orbital stability in the sense of Lyapunov, asymptotic orbital stability, exponential orbital stability, or even finite-time orbital stability.

In order to prove that a system is orbitally stable (locally, over a region, or globally), or to analyze the region of attraction of a limit cycle, we can use a Lyapunov function. When we analyzed our simple models of walking and running, we carried out the analysis on the Poincare map. Indeed, if we can find a Lyapunov function that proves (i.s.L., asympototic, or exponential) stability of the discrete Poincare map, then this implies (i.s.L, asymptotic, or exponential) *orbital* stability of the cycle. But Lyapunov functions of this form are difficult to verify for a pretty fundamental reason: we rarely have an analytical expression for the Poincare map, since it is the result of integrating a nonlinear dynamics over the cycle.

Instead, we will focus our attention on constructing Lyapunov functions in the full state space, which use the continuous dynamics. In particular, we would like to consider Lyapunov functions which have the form cartooned below; they vanish to zero everywhere along the cycle, and are strictly positive everywhere away from the cycle.

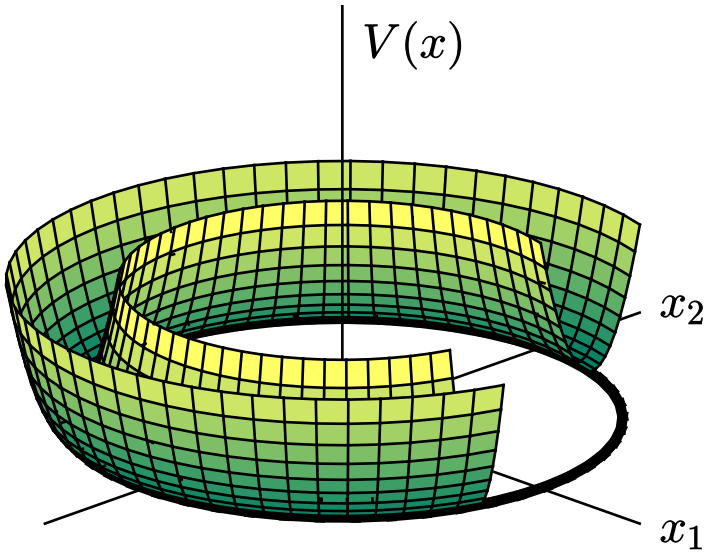



Figure 16.1 - Cartoon of a Lyapunov function which vanishes on a limit cycle, and is strictly positive everywhere else. (a small segment has been removed solely for the purposes of visualization).

### 16.2.1 Transverse coordinates

How can we parameterize this class of functions? For arbitrary cycles this could be very difficult in the original coordinates. For simple cycles like in the cartoon, one could imagine using polar coordinates. More generally, we will define a new coordinate system relative to the orbit, with coordinates

- $\tau$ - the phase along the orbit
- $\mathbf{x}_\perp(\tau)$ - the remaining coordinates, linearly independent from $\tau$.

Given a state $\mathbf{x}$ in the original coordinates, we must define a smooth mapping $\mathbf{x} \rightarrow (\tau, \mathbf{x}_\perp)$ to this new coordinate system. For example, for a simple ring oscillator we might have:

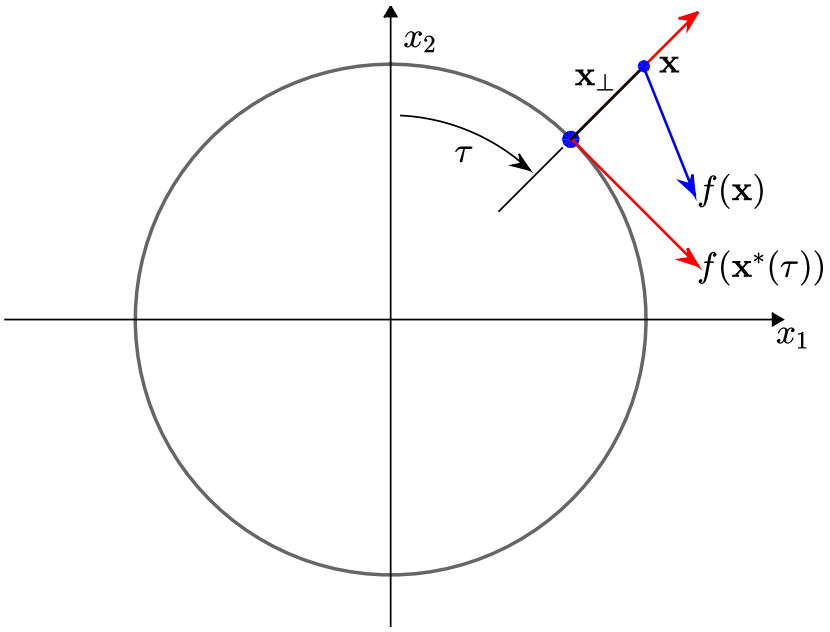



Figure 16.2 - A moving coordinate system along the limit cycle.

In general, for an $n$-dimensional state space, $\tau$ will always be a scalar, and $\mathbf{x}_\perp$ will be an $(n-1)$-dimensional vector defining the remaining coordinates relative to $\mathbf{x}^*(\tau)$ . In fact, although we use the notation $\mathbf{x}_\perp$ the coordinate system need not be strictly orthogonal to the orbit, but must simply be *transversal* (not parallel). Having defined the smooth mapping $\mathbf{x} \to (\tau, \mathbf{x}_\perp)$, we can always rewrite the dynamics in this new coordinate system:

$$\dot{\tau} = f_\tau(\mathbf{x}_\perp, \tau)$$
$$\dot{\mathbf{x}}_\perp = f_\perp(\mathbf{x}_\perp, \tau).$$

To keep our notation simpler for the remainder of these notes, we will assume that the origin of this new coordinate system is on the nominal trajectory ( $\min_\tau |\mathbf{x} - \mathbf{x}^*(\tau)| = 0 \Rightarrow \mathbf{x}_\perp = 0$). Similarly, by taking $\tau$ to be the phase variable, we will leverage the fact that on the nominal trajectory, we have $\dot{\tau} = f_\tau(0, \tau) = 1$.

The value of this construction for Lyapunov analysis was proposed in [1] and has been extended nicely to control design in [2] and for region of attraction estimation in [3]. A quite general numerical strategy for defining the transversal coordinates is given in [4].

## Theorem 16.1 - A Lyapunov theorem for orbital stability

For a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^n$, $f$ continuous, a continuous periodic solution $\mathbf{x}^*(\tau)$, and a smooth mapping $\mathbf{x} \to (\tau, \mathbf{x}_\perp)$ where $\mathbf{x}_\perp$ vanishes on $\mathbf{x}^*$, then for some $n-1$ dimensional ball $\mathcal{B}$ around the origin, if I can produce a $V(\mathbf{x}_\perp, \tau)$ such that

$$\forall \tau, V(0, \tau) = 0,$$
$$\forall \tau, \forall \mathbf{x}_\perp \in \mathcal{B}, \mathbf{x}_\perp \neq 0, V(\mathbf{x}_\perp, \tau) > 0,$$

with

$$\forall \tau, \dot{V}(0, \tau) = 0,$$
$$\forall \tau, \forall \mathbf{x}_\perp \in \mathcal{B}, \mathbf{x}_\perp \neq 0, \dot{V}(\mathbf{x}_\perp, \tau) < 0,$$

then the solution $\mathbf{x}^*(t)$ is *locally orbitally asymptotically stable*.

Orbital stability in the sense of Lyapunov and exponential orbital stability can also be verified (with $\dot{V}_\perp \leq 0$ and $\dot{V}_\perp \leq \alpha V_\perp$, respectively).

## Example 16.2 (Simple ring oscillator)

Perhaps the simplest oscillator is the first-order system which converges to the unit circle. In cartesian coordinates, the dynamics are

$$\begin{aligned}\dot{x}_1 =& x_2 - \alpha x_1 \left(1 - \frac{1}{\sqrt{x_1^2 + x_2^2}}\right)\\ \dot{x}_2 =& -x_1 - \alpha x_2 \left(1 - \frac{1}{\sqrt{x_1^2 + x_2^2}}\right),\end{aligned}$$

where $\alpha$ is a positive scalar gain.

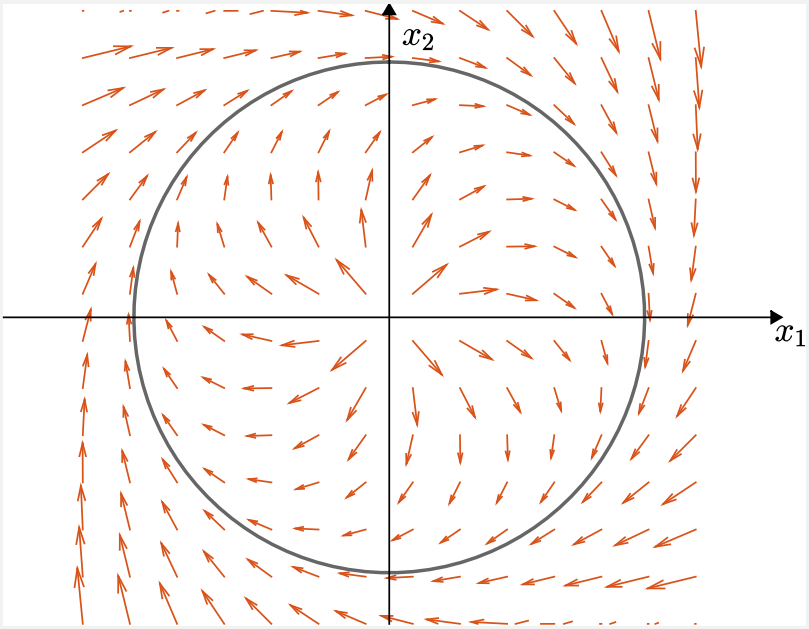



Figure 16.3 - Vector field of the ring oscillator

If we take the transverse coordinates to be the polar coordinates, shifted so that $x_\perp$ is zero on the unit circle,

$$\begin{aligned}\tau =& \text{atan2}(-x_2, x_1)\\ x_\perp =& \sqrt{x_1^2 + x_2^2} - 1,\end{aligned}$$

which is valid when $x_\perp > -1$, then the simple transverse dynamics are revealed:

$$\begin{aligned}\dot{\tau} =& 1\\ \dot{x}_\perp =& -\alpha x_\perp.\end{aligned}$$

Taking a Lyapunov candidate $V(x_\perp, \tau) = x_\perp^2$, we can verify that

$$\dot{V} = -2\alpha x_\perp^2 < 0, \quad \forall x_\perp > -1.$$

This demonstrates that the limit cycle is locally asymptotically stable, and furthermore that the invariant open-set $V < 1$ is inside the region of attraction of that limit cycle. In fact, we know that all $x_\perp > -1$ are in the region of attraction that limit cycle, but this is not proven by the Lyapunov argument above.

Let's compare this approach again with the approach that we used in the chapter on walking robots, where we used a Poincaré map analysis to investigate limit cycle stability. In transverse coordinates approach, there is an additional burden to construct the coordinate system along the entire trajectory, instead of only at a single surface of section. In fact, the transverse coordinates approach is sometimes referred to as a "moving Poincaré section". Again, the reward for this extra work is that we can check a condition that only involves the instantaneous dynamics, $f(\mathbf{x})$, as opposed to having to integrate the dynamics over an entire cycle to generate the discrete Poincaré map, $\mathbf{x}_p[n+1] = P(\mathbf{x}_p[n])$. As we will see below, this approach will also be more compatible with designing continuous feedback controller that stabilize the limit cycle.

### 16.2.2 Transverse linearization

In the case of Lyapunov analysis around a fixed point, there was an important special case: for stable linear systems, we actually have a recipe for constructing a Lyapunov function. As a result, for nonlinear systems, we often found it convenient to begin our search by linearizing around the fixed point and using the Lyapunov candidate for the linear system as an initial guess. That same approach can be extended to limit cycle analysis.

In particular, let us make a linear approximation of the transverse dynamics:

$$\dot{\mathbf{x}}_\perp = f_\perp(\mathbf{x}_\perp, \tau) \approx \frac{\partial f_\perp}{\partial \mathbf{x}_\perp}\mathbf{x}_\perp = \mathbf{A}_\perp(\tau)\mathbf{x}_\perp.$$

Note that I've assumed that $\mathbf{x}_\perp$ is zero on the nominal trajectory, so don't need the additional notation of working in the error coordinates here. Remember that $\tau$ is the phase along the trajectory, but [1] showed that (exponential) stability of the *time-varying* linear system $\dot{\mathbf{x}}_\perp = \mathbf{A}_\perp(t)\mathbf{x}_\perp$ implies local exponential orbital stability of the original system. In particular, if this transverse linear system is periodic and orbitally stable, then for each $\mathbf{Q} = \mathbf{Q}^T \succ 0$, there exists a unique positive definite solution to the periodic Riccati equation:

$$V(\mathbf{x}_\perp, \tau) = \mathbf{x}_\perp^T \mathbf{P}(\tau)\mathbf{x}_\perp, \qquad \mathbf{P}^T(\tau) = \mathbf{P}(\tau), \quad \mathbf{P}(\tau + t_{period}) = \mathbf{P}(\tau),$$
$$-\dot{\mathbf{P}}(\tau) = \mathbf{P}(\tau)\mathbf{A}(\tau) + \mathbf{A}(\tau)\mathbf{P}^T(\tau) + \mathbf{Q}, \qquad \mathbf{Q} = \mathbf{Q}^T \succ 0.$$

There is a surprisingly rich literature on the numerical methods for finding the periodic solutions to these Lyapunov (and corresponding Riccati) equations. In practice, for every problem I've ever tried, I've simply integrated the equations backwards in time until the integration converges to a periodic solution (this is not guaranteed to work, but almost always does).

This is very powerful. It gives us a general way to construct a Lyapunov candidate that certifies local stability of the cycle, and which can be used as a candidate for nonlinear Lyapunov analysis.

### 16.2.3 Region of attraction estimation using sums-of-squares

Coming soon. If you are interested, see [4, 3].

## 16.3 Feedback Design

Many of the tools we've developed for stabilization of fixed points or stabilization of trajectories can be adapted to this new setting. When we discussed control for the spring-loaded inverted pendulum model, we considered discrete control decisions, made once per cycle, which could stabilize the discrete Poincare map, $\mathbf{x}_p[n+1] = f_p(\mathbf{x}_p[n], \mathbf{u}[n])$, with $\mathbf{u}[n]$ the once-per-cycle decisions. While it's possible to

get the local linear approximations of this map using our adjoint methods, remember that we rarely have an analytical expression for the nonlinear $f_p$.

How limiting is it to restrict ourselves to once-per-cycle decisions? Of course we can improve performance in general with continuous feedback. But there are also some cases where once-per-cycle decisions feel quite reasonable. Foot placement for legged robots is a good example -- once we've chosen a foot placement we might make minor corrections just before we place the foot, but rarely change that decision once the foot is on the ground; once per footstep seems like a reasonable approximation. Another good example is flapping flight: here the wing beat cycle times can be much faster than the dynamic timescales of the center of mass, and changing the parameters of the wing stroke one per flap seems pretty reasonable.

But the tools we've developed above also give us the machinery we need to consider continuous feedback throughout the trajectory. Let's look at a few important formulations.

### 16.3.1 For underactuation degree one.

It turns out many of our simple walking models -- particularly ones with a point foot that are derived in the minimal coordinates -- are only short one actuator (between the foot and the ground). One can represent even fairly complex robots this way; much of the theory that I'll elude to here was originally developed by Jessy Grizzle and his group in the context of the bipedal robot RABBIT. Jessy's key observation was that limit cycle stability is effectively stability in $n-1$ degrees of freedom, and you can often achieve it easily with $n-1$ actuators -- he called this line of work "Hybrid Zero Dynamics" (HZD). We'll deal with the "hybrid" part of that in the next chapter, but here is a simple example to illustrate the "zero dynamics" concept. [Coming soon...]

Stabilizing the zero dynamics is synonymous with achieving orbital stability. Interestingly, it doesn't actually guarantee that you will go around the limit cycle. One "feature" of the HZD walkers is that you can actually stand in front of them and stop their forward progress with your hand -- the control system will continue to act, maintaining itself on the cycle, but progress along the cycle has stopped. In some of the robots you could even push them backwards and they would move through the same walking cycle in reverse. If one wants to certify that forward progress will be achieved, then the tasks is reduced to inspecting the one-dimensional dynamics of the phase variable along the cycle.

The notion of "zero dynamics" is certainly not restricted to systems with underactuation of degree one. In general, we can easily stabilize a manifold of dimension $m$ with $m$ actuators (we saw this in the section on task-space partial feedback linearization), and if being on that manifold is sufficient to achieve our task, or if we can certify that the resulting dynamics on the manifold are sufficient for our task, then life is good. But in the "underactuation degree one" case, the manifold under study is a trajectory/orbit, and the tools from this chapter are immediately applicable.

### 16.3.2 Transverse LQR

Another natural approach to stabilizing a cycle uses the transverse linearization. In particular, for a nominal (control) trajectory, $[\mathbf{x}_0(t), \mathbf{u}_0(t)]$, with $\dot{\mathbf{x}}_0 = f(\mathbf{x}_0, \mathbf{u}_0)$, we can approximate the transverse dynamics:

$$\dot{\mathbf{x}}_\perp = f_\perp(\mathbf{x}_\perp, \tau, \mathbf{u}) \approx \frac{\partial f_\perp}{\partial \mathbf{x}_\perp}\mathbf{x}_\perp + \frac{\partial f_\perp}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0(\tau)) = \mathbf{A}_\perp(\tau)\mathbf{x}_\perp + \mathbf{B}_\perp(\tau)\bar{\mathbf{u}}.$$

[2] showed that the (periodic) time-varying LQR controller that stabilized this system achieves *orbital* stabilization of the original system.

Let's take a minute to appreciate the difference between this approach and time-

varying LQR in the full coordinates. Of course, the behavior is quite different: time-varying LQR in the full coordinates will try to slow down or speed up to keep time with the nominal trajectory, where this controller makes no attempt to stabilize the phase variable. You might think you could design the original time-varying controller, $\bar{\mathbf{u}} = \mathbf{K}(t)\bar{\mathbf{x}}$, and just "project to the closest time" during execution (e.g. find the time $t = \text{argmin}_\tau |\mathbf{x} - \mathbf{x}_0(\tau)|$) and use the feedback corresponding to that time. But this projection is not guaranteed to be safe; you can find systems that are stabilizable by the transverse linearization are unstable in closed-loop using a feedback based on this projection.

**Example 16.3 (Time-varying LQR simply projected in time can lead to instability.)**

Coming soon...

But there is an even more important reason to like the transverse LQR approach. If orbital stability is all you need, then this is a better formulation of the problem because you are asking for less. As we've just discussed with hybrid zero-dynamics, stabilizing $m$ degrees of freedom with $m$ actuators is potentially very different than stabilizing $m+1$ degrees of freedom. In the transverse formulation, we are asking LQR to minimize the cost only in the transverse coordinates (it is mathematically equivalent to designing a cost function in the full coordinates that is zero in the direction of the nominal trajectory). In practice, this can result is much smaller cost-to-go matrices, $\mathbf{S}(t)$, and smaller feedback gains, $\mathbf{K}(t)$. For underactuated systems, this difference can be dramatic.

### 16.3.3 Orbital stabilization for non-periodic trajectories

The observation that orbital stabilization of a trajectory can ask much less of your underactuated control system (and lead to smaller LQR feedback gains), is quite powerful. It is not limited to stabilizing periodic motions. If your aim is to stabilize a non-periodic path through state-space, but do not actually care about the timing, then formulating your stabilization can be a very good choice. You can find examples where a system is stabilizable in the transverse coordinates, but not in the full coordinates.

**Example 16.4 (Stabilizable in the transverse coordinates, but not the full coordinates.)**

Coming soon...

Take care, though, as the converse can also be true: it's possible that a system could be stabilizable in the full coordinates but *not* in some transverse coordinates; if you choose those coordinates badly.

**Example 16.5 (Stabilizable in the full coordinates, but not the transverse coordinates.)**

Consider a trajectory with $u(t) = 1$ (from any initial condition) for the double integrator. We can certainly stablize this trajectory in the full coordinates using LQR. Technically, one can choose transverse coordinates $\tau = \dot{q}$, and $x_\perp = q$. Clearly $x_\perp$ is transverse to the nominal trajectory everywhere. But this transverse linearization is also clearly not stabilizable. It's a bad choice!

The general approach to designing the transverse coordinates [4] could address this

concern by including a criteria for controllability when optimizing the transverse coordinates.

## References


1. John Hauser and Chung Choo Chung, "Converse {L}yapunov functions for exponentially stable periodic orbits", *Systems & Control Letters*, vol. 23, no. 1, pp. 27 -- 34, 1994.

2. Anton S. Shiriaev and Leonid B. Freidovich and Ian R. Manchester, "Can We Make a Robot Ballerina Perform a Pirouette? Orbital Stabilization of Periodic Motions of Underactuated Mechanical Systems", *Annual Reviews in Control*, vol. 32, no. 2, pp. 200--211, Dec, 2008.

3. Ian R. Manchester and Mark M. Tobenkin and Michael Levashov and Russ Tedrake, "Regions of Attraction for Hybrid Limit Cycles of Walking Robots", *Proceedings of the 18th IFAC World Congress, extended version available online: arXiv:1010.2247 [math.OC]*, 2011.

4. Ian R. Manchester, "Transverse Dynamics and Regions of Stability for Nonlinear Hybrid Limit Cycles", *Proceedings of the 18th IFAC World Congress, extended version available online: arXiv:1010.2241 [math.OC]*, Aug-Sep, 2011.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 17
# Planning and Control through Contact

Open in Colab

So far we have developed a fairly strong toolbox for planning and control with "smooth" systems -- systems where the equations of motion are described by a function $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ which is smooth everywhere. But our discussion of the simple models of legged robots illustrated that the dynamics of making and breaking contact with the world are more complex -- these are often modeled as hybrid dynamics with impact discontinuities at the collision event and constrained dynamics during contact (with either soft or hard constraints).

My goal for this chapter is to extend our computational tools into this richer class of models. Many of our core tools still work: trajectory optimization, Lyapunov analysis (e.g. with sums-of-squares), and LQR all have natural equivalents.

Let's start with a warm-up exercise: trajectory optimization for the rimless wheel. We already have basically everything that we need for this, and it will form a nice basis for generalizing our approach throughout the chapter.

**Example 17.1 (Trajectory optmization for the rimless wheel)**

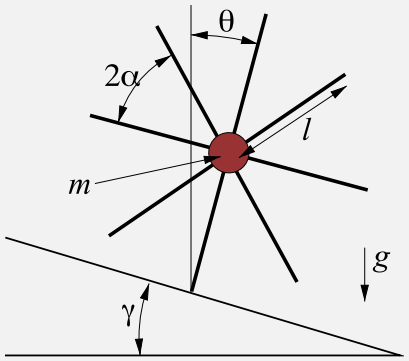


Figure 17.1 - The rimless wheel. The orientation of the stance leg, $\theta$, is measured clockwise from the vertical axis.

The rimless wheel was our simplest example of a passive-dynamic walker; it has no control inputs but exhibits a passively stable rolling fixed point. We've also already seen that trajectory optimization can be used as a tool for finding limit cycles of a smooth passive system, e.g. by formulating a direct collocation problem:

$$\begin{aligned} \underset{\mathbf{x}[\cdot],h}{\text{find}} \quad \text{subject to} \quad & \text{collocation} \quad \text{constraints}(\mathbf{x}[n], \mathbf{x}[n+1], h), \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}[N], \\ & h_{min} \leq h \leq h_{max}, \end{aligned}$$

where $h$ was the timestep between the trajectory break points.

It turns out that applying this to the rimless wheel is quite straight forward. We still want to find a periodic trajectory, but now have to take into account the collision event. We can do this by modifying the periodicity condition. Let's force the initial state to be just after the collision, and the final state to be just before the collision, and make sure they are related to each other via the collision equation:

$$\begin{aligned} \underset{\mathbf{x}[\cdot],h}{\text{find}} \quad \text{subject to} \quad & \text{collocation} \quad \text{constraints}(\mathbf{x}[n], \mathbf{x}[n+1], h), \quad \forall n \in [0, N-1] \\ & \theta[0] = \gamma - \alpha, \\ & \theta[N] = \gamma + \alpha, \\ & \dot{\theta}[0] = \dot{\theta}[N] \cos(2\alpha) \\ & h_{min} \leq h \leq h_{max}. \end{aligned}$$

Although it is likely not needed for this simple example (since the dynamics are sufficiently limited), for completeness one should also add constraints to ensure that none of the intermediate points are in contact,

$$\gamma - \alpha \leq \theta[n] < \gamma + \alpha, \quad \forall n \in [1, N-1].$$

The result is a simple and clean numerical algorithm for finding the rolling limit cycle solution of the rimless wheel. Please take it for a spin:


The specific case of the rimless wheel is quite clean. But before we apply it to the compass gait, the kneed compass gait, the spring-loaded inverted pendulum, etc, then we should stop and figure out a more general form.

## 17.1 (Autonomous) Hybrid Systems

Recall how we modeled the dynamics of the simple legged robots. First, we derived the equations of motion (independently) for each possible contact configuration -- for example, in the spring-loaded inverted pendulum (SLIP) model we had one set of equations governing the $(x, y)$ position of the mass during the flight phase, and a completely separate set of equations written in polar coordinates, $(r, \theta)$, describing the stance phase. Then we did a little additional work to describe the transitions between these models -- e.g., in SLIP we transitioned from flight to stance when the foot first touches the ground. When simulating this model, it means that we have a discrete "event" which occurs at the moment of foot collision, and an immediate discontinuous change to the state of the robot (in this case we even change out the state variables).

The language of *hybrid systems* gives us a rich language for describing systems of this form, and a suite of tools for analyzing and controlling them. The term "hybrid systems" is a bit overloaded, here we use "hybrid" to mean both discrete- and

continuous-time, and the particular systems we consider here are sometimes called *autonomous* hybrid systems because the internal dynamics can cause the discrete changes without any exogeneous input†. In the hybrid systems formulation, we describe a system by a set of *modes* each described by (ideally smooth) continuous dynamics, a set of *guards* which here are continuous functions whose zero-level set describes the conditions which trigger an event, and a set of *resets* which describe the discrete update to the state that is triggered by the guard. Each guard is associated with a particular mode, and we can have multiple guards per mode. Every guard has at most one reset. You will occasionally here guards referred to as "witness functions", since they play that role in simulation, and resets are sometimes referred to as "transition functions".

†This is in contrast to for instance, the mode of a power-train when a change in gears comes as an external input.

The imagery that I like to keep in my head for hybrid systems is illustrated below for a simple example of a robot's heel striking the ground. A solution trajectory of the hybrid system has a continuous trajectory inside each mode, punctuated by discrete updates when the trajectory hits the zero-level set of the guard (here the distance between the heel and the ground becomes zero), with the reset describing the discrete change in the state variables.

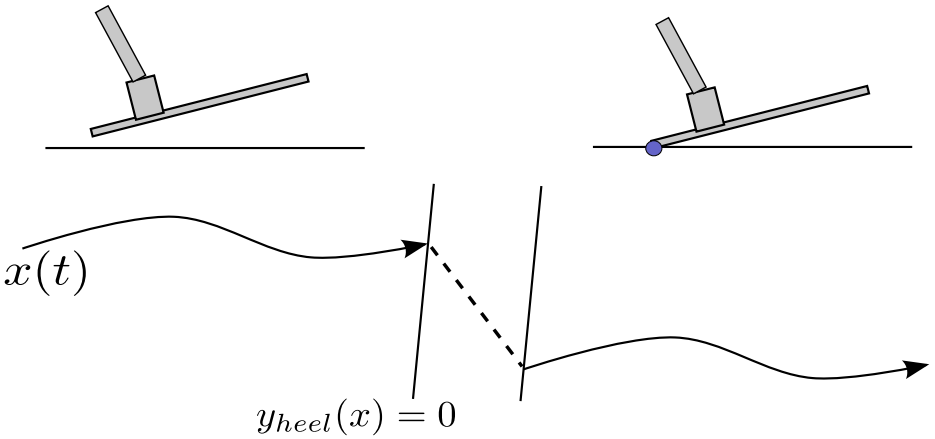



Figure 17.2 - Modeling contact as a hybrid system.

For this robot foot, we can decompose the dynamics into distinct modes: (1) foot in the air, (2) only heel on the ground, (3) heel and toe on the ground, (4) only toe on the ground (push-off). More generally, we will write the dynamics of mode $i$ as $\mathbf{f}_i$, the guard which signals the transition mode $i$ to mode $j$ as $\phi_{i,j}$ (where $\phi_{i,j}(\mathbf{x}_i) > 0$ inside mode $i$), and the reset map from $i$ to $j$ as $\mathbf{\Delta}_{i,j}$, as illustrated in the following figure:

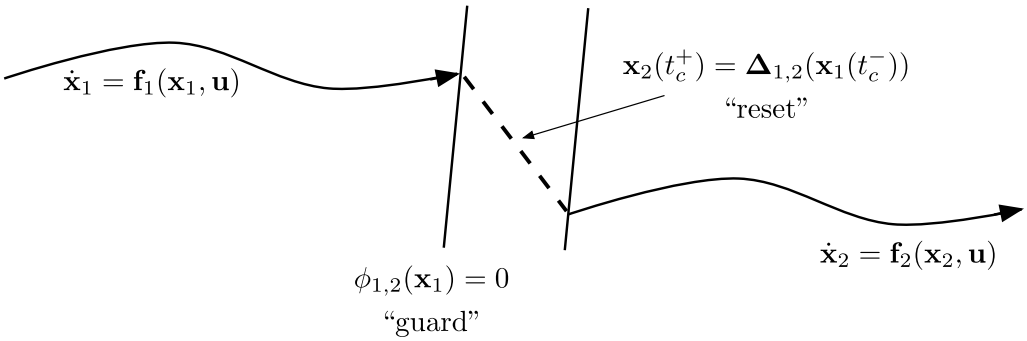



Figure 17.3 - The language of hybrid systems: modes, guards, and reset maps.

### 17.1.1 Hybrid trajectory optimization

Using the more general language of modes, guards, and resets, we can begin to formulate the "hybrid trajectory optimization" problem. In hybrid trajectory optimization, there is a major distinction between trajectory optimization where *the mode sequence is known apriori* and the optimization is just attempting to solve for

the continuous-time trajectories, vs one in which we must also discover the mode sequence.

For the case when the mode sequence is fixed, then hybrid trajectory optimization is as simple as stitching together multiple individual mathematical programs into a single mathematical program, with the boundary conditions constrained to enforce the guard/reset constraints. For a simple hybrid system with a given a mode sequence and using the shorthand $\mathbf{x}_k$, etc, for the state in mode in the $k$th segment of the sequence, we can write:

$$\begin{aligned}
\underset{\mathbf{x}_k[\cdot], h_k}{\text{find}} \quad \text{subject to} \quad & \mathbf{x}_0[0] = \mathbf{x}_0, \\
\forall k \quad & \phi_{k,k+1}(\mathbf{x}_k[N_k]) = 0, \\
& \mathbf{x}_{k+1}[0] = \boldsymbol{\Delta}_{i,j}(\mathbf{x}_k[N_k]), \\
& \phi_{k,k'}(\mathbf{x}_k[n_k]) > 0, \quad \forall n_k \in [0, N_k], \forall k' \\
& h_{min} \le h_k \le h_{max}, \\
& \text{collocation constraints}_k(\mathbf{x}_k[n_k], \mathbf{x}_k[n_k+1], h_k), \quad \forall n_k \in [0, N_k - 1].
\end{aligned}$$

It is then natural to add control inputs (as additional decision variables), and to add an objective and any more constraints.

## Example 17.2 (A basketball trick shot.)

As a simple example of this hybrid trajectory optimization, I thought it would be fun to see if we can formulate the search for initial conditions that optimizes a basketball "trick shot". A quick search turned up this video for inspiration.

Let's start simpler -- with just a "bounce pass". We can capture the dynamics of a bouncing ball (in the plane, ignoring spin) with some very simple dynamics:

$$\mathbf{q} = \begin{bmatrix} x \\ z \end{bmatrix}, \qquad \ddot{\mathbf{q}} = \begin{bmatrix} 0 \\ -g \end{bmatrix}.$$

During any time interval without contact of duration $h$, we can actually integrate these dynamics perfectly:

$$\mathbf{x}(t+h) = \begin{bmatrix} x(t) + h\dot{x}(t) \\ z(t) + h\dot{z}(t) - \frac{1}{2}gh^2 \\ \dot{x}(t) \\ \dot{z}(t) - hg \end{bmatrix}.$$

With the bounce pass, we just consider collisions with the ground, so we have a guard, $\phi(\mathbf{x}) = z$, which triggers when $z = 0$, and a reset map which assumes an elastic collision with coefficient of restitution $e$:

$$\mathbf{x}^+ = \boldsymbol{\Delta}(\mathbf{x}^-) = \begin{bmatrix} x^- & z^- & \dot{x}^- & -e\dot{z}^- \end{bmatrix}^T.$$

We'll formulate the problem as this: given an initial ball position $(x = 0, z = 1)$, a final ball position 4m away $(x = 4, z = 1)$, find the initial velocity to achieve that goal in 5 seconds. Clearly, this implies that $\dot{x}(0) = 4/5$. The interesting question is -- what should we do with $\dot{z}(0)$? There are multiple solutions -- which involve bouncing a different number of times. We can find them all with a simple hybrid trajectory optimization, using the observation that there are two possible solutions for each number of bounces -- one that starts with a positive $\dot{z}(0)$ and one with a negative $\dot{z}(0)$.

Open in Colab

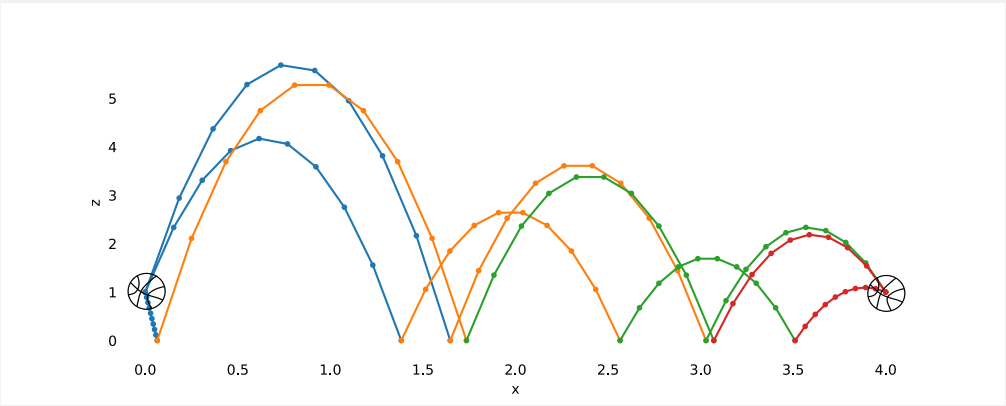



Figure 17.4 - Trajectory optimization to find solutions for a "bounce pass". I've plotted all solutions that were found for 2, 3, or 4 bounces... but I think it's best to stick to a single bounce if you're using this on the court.

Now let's try our trick shot. I'll move our goal to $x_f = -1m, z_f = 3m$, and introduce a vertical wall at $x = 0$, and move our initial conditions back to $x_0 = -.25m$. The collision dynamics, which now must take into account the spin of the ball, are in the appendix. The first bounce is against the wall, the second is against the floor. I'll also constrain the final velocity to be down (have to approach the hoop from above). Try it out.

Open in Colab

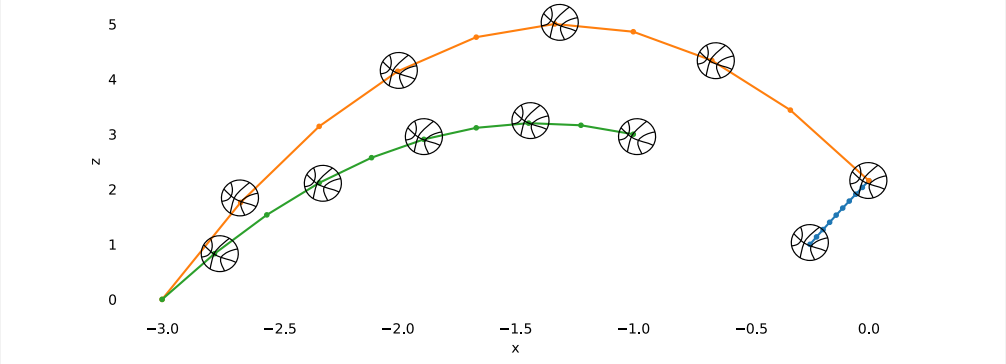



Figure 17.5 - Trajectory optimization for the "trick shot". Nothing but the bottom of the net! The crowd is going wild!

In this example, we could integrate the dynamics in each segment analytically. That is the exception, not the rule. But you can take the same steps with a little more code to use, e.g. direct transcription or collocation with multiple break points in each segment.

## 17.1.2 Stabilizing hybrid models.

## 17.1.3 Deriving hybrid models: minimal vs floating-base coordinates

There is some work to do in order to derive the equations of motion in this form. Do you remember how we did it for the rimless wheel and compass gait examples? In both cases we assumed that exactly one foot was attached to the ground and that it would not slip, this allowed us to write the Lagrangian as if there was a pin

joint attaching the foot to the ground to obtain the equations of motion. For the SLIP model, we derived the flight phase and stance phase using separate Lagrangian equations each with different state representations. I would describe this as the *minimal coordinates* modeling approach -- it is elegant and has some important computational advantages that we will come to appreciate in the algorithms below. But it's a lot of work! For instance, if we also wanted to consider friction in the foot contact of the rimless wheel, we would have to derive yet another set of equations to describe the sliding mode (adding, for instance, a prismatic joint that moved the foot along the ramp), plus the guards which compute the contact force for a given state and the distance from the boundary of the friction cone, and on and on.

Fortunately, there is an alternative modeling approach for deriving the modes, guards, and resets for contact that is more general (though admittedly also more complex). We can instead model the robot in the *floating-base coordinates* -- we add a fictitious six degree-of-freedom "floating-base" joint connecting some part of the robot to the world (in planar models, we use just three degrees-of-freedom, e.g. $(x, z, \theta)$). We can derive the equations of motion for the floating-base robot once, without considering contact, then add the additional constraints that come from being in contact as contact forces which get applied to the bodies. The resulting manipulator equations take the form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u} + \sum_i \mathbf{J}_i^T(\mathbf{q})\lambda_i, \tag{1}$$

where $\lambda_i$ are the constraint forces and $\mathbf{J}_i$ are the constraint Jacobians. Conveniently, if the guard function in our contact equations is the signed distance from contact, $\phi_i(\mathbf{q})$, then this Jacobian is simply $\mathbf{J}_i(\mathbf{q}) = \frac{\partial \phi_i}{\partial \mathbf{q}}$. I've written the basic derivations for the common cases (position constraints, velocity constraints, impact equations, etc) in the appendix. What is important to understand here is that this is an alternative formulation for the equations governing the modes, guards, and resets, but that is it no longer a minimal coordinate system -- the equations of motion are written in $2N$ state variables but the system might actually be constrained to evolve only along a lower dimensional manifold (if we write the rimless wheel equations with three configuration variables for the floating base, it still only rotates around the toe when it is in stance and is inside the friction cone). This will have implications for our algorithms.

## 17.2 EXERCISES

### Exercise 17.1 (Finding the compass gait limit cycle)

In this exercise we use trajectory optimization to identify a limit cycle for the compass gait. We use a rather general approach: the robot dynamics is described in floating-base coordinates and frictional contacts are accurately modeled. In this notebook, you are asked to code many of the constraints this optimization problem requires:

a. Enforce the contact between the stance foot and the ground at all the break points.
b. Enforce the contact between the swing foot and the ground at the initial time.
c. Prevent the penetration of the swing foot in the ground at all the break points. (In this analysis, we will neglect the scuffing between the swing foot and the ground which arises when the swing leg passes the stance leg.)
d. Ensure that the contact force at the stance foot lies in the friction cone at all the break points.
e. Ensure that the impulse generated by the collision of the swing foot with the ground lies in the friction cone.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Chapter 18
# System Identification

Open in Colab

My primary focus in these notes has been to build algorithms that design of analyze a control system *given a model* of the plant. In fact, we have in some places gone to great lengths to understand the structure in our models (the structure of the manipulator equations in particular) and tried to write algorithms which exploit that structure.

Our ambitions for our robots have grown over recent years to where it makes sense to question this assumption. If we want to program a robot to fold laundry, spread peanut butter on toast, or make a salad, then we should absolutely not assume that we are simply given a model (and the ability to estimate the state of that model). This has led many researchers to focus on the "model-free" approaches to optimal control that are popular in reinforcement learning. But I worry that the purely model-free approach is "throwing the baby out with the bathwater". We have fantastic tools for making long-term decisions given a model; the model-free approaches are correspondingly much much weaker.

So in this chapter I would like to cover the problem of learning a model. This is far from a new problem. The field of "system identification" is as old as controls itself, but new results from machine learning have added significantly to our algorithms and our analyses, especially in the high-dimensional and finite-sample regimes. But well before the recent machine learning boom, system identification as a field had a very strong foundation with thorough statistical understanding of the basic algorithms, at least in the asymptotic regime (the limit where the amount of data goes to infinity). My goal for this chapter is to establish this foundation, and to provide some pointers, but I will stop short of providing the full statistical viewpoint here.

## 18.1 Problem Formulation: Equation Error vs Simulation Error


Our problem formulation inevitably begins with the data. In practice, if we have access to a physical system, instrumented using digital electronics, then we have a system in which we can apply input commands, $\mathbf{u}_n$, at some discrete rate, and measure the outputs, $\mathbf{y}_n$ of the system at some discrete rate. We normally assume these rates our fixed, and often attempt to fit a state-space model of the form

$$\mathbf{x}[n+1] = f_\alpha(\mathbf{x}[n], \mathbf{u}[n]), \qquad \mathbf{y}[n] = g_\alpha(\mathbf{x}[n], \mathbf{u}[n]), \tag{1}$$

where I have used $\alpha$ again here to indicate a vector of parameters. In this setting, a natural formulation is to minimize a least-squares estimation objective:

$$\min_{\alpha, \mathbf{x}[0]} \sum_{n=0}^{N-1} \|\mathbf{y}[n] - \mathbf{y}_n\|_2^2, \qquad \text{subject to } (1).$$

I have written purely deterministic models to start, but in general we expect both the state evolution and the measurements to have randomness. Sometimes, as I have written, we fit a deterministic model to the data and rely on our least-squares objective to capture the errors; more generally we will look at fitting stochastic models to the data.

We often separate the identification procedure into two parts, were we first estimate the state $\hat{\mathbf{x}}_n$ given the input-output data $\mathbf{u}_n, \mathbf{y}_n$, and then focus on estimating the state-evolution dynamics in a second step. The dynamics estimation algorithms fall into two main categories:

- *Equation error* minimizes only the one-step prediction error:

$$\min_{\alpha} \sum_{n=0}^{N-2} \|f_\alpha(\hat{\mathbf{x}}_n, \mathbf{u}_n) - \hat{\mathbf{x}}_{n+1}\|_2^2.$$

- *Simulation error* captures the long-term prediction error:

$$\min_{\alpha} \sum_{n=1}^{N-1} \|\mathbf{x}[n] - \hat{\mathbf{x}}_n\|_2^2, \qquad \text{subject to} \quad \mathbf{x}[n+1] = f_\alpha(\mathbf{x}[n], \mathbf{u}_n), \mathbf{x}[0] = \hat{\mathbf{x}}_0,$$

The equation-error formulations often result in much more tractable optimization problems, but unfortunately we will see that optimizing the one-step error can still result in arbitrarily large simulation errors. Therefore, we generally consider the simulation error to be the true objective we hope to optimize, and the equation error only as a potentially useful surrogate.

## 18.2 Parameter Identification for Mechanical Systems

My primary focus throughout these notes is on (underactuated) mechanical systems, but when it comes to identification there is an important distinction to make. For some mechanical systems we know the structure of the model, including number of state variables and the topology of the kinematic tree. Legged robots like Spot or Atlas are good examples here -- the dynamics are certainly nontrivial, but the general form of the equations are known. In this case, the task of identification is really the task of estimating the parameters in a structured model. That is the

subject of this section.

The examples of folding laundry or making a salad fall into a different category. In those examples, I might not even know a priori the number of state variables needed to provide a reasonable description of the behavior. That will force a more general examination of the the identification problem, which we will explore in the remaining sections.

Let's start with the problem of identifying a canonical underactuated mechanical system, like an Acrobot, Cart-Pole or Quadrotor, where we know the structure of the equations, but just need to fit the parameters. We will further assume that we have the ability to directly observe all of the state variables, albeit with noisy measurements (e.g. from joint sensors and/or inertial measurement units). The stronger state estimation algorithms that we will discuss soon assume a model, so we typically do not use them directly here.

Consider taking a minute to review the example of deriving the manipulator equations for the double pendulum before we continue.

### 18.2.1 Kinematic parameters and calibration

We can separate the parameters in the multibody equations again into kinematic parameters and dynamic parameters. The kinematic parameters, like link lengths, describe the coordinate transformation from one joint to another joint in the kinematic tree. It is certainly possible to write an optimization procedure to calibrate these parameters; you can find a fairly thorough discussion in e.g. Chapter 11 of [1]. But I guess I'm generally of the opinion that if you don't have accurate estimates of your link lengths, then you should probably invest in a tape measure before you invest in nonlinear optimization.

One notable exception to this is calibration with respect to joint offsets. This one can be a real nuisance in practice. Joint sensors can slip, and some robots even use relative rotary encoders, and rely on driving the joint to some known hard joint limit each time the robot is powered on in order to obtain the offset. I've worked on one humanoid robot that had a quite elaborate and painful kinematic calibration procedure which involve fitting additional hardware over the joints to ensure they were in a known location and then running a script. Having a an expensive and/or unreliable calibration procedure can put a damper on any robotics project. For underactuated systems, in particular, it can have a dramatic effect on performance.

**Example 18.1 (Acrobot balancing with calibration error)**

Small kinematic calibration errors can lead to large steady-state errors when attempting to stabilize a system like the Acrobot. I've put together a simple notebook to show the effect here:


Our tools from robust / stochastic control are well-suited to identifying (and bounding / minimizing) these sensitivities, at least for the linearized model we use in LQR.

The general approach to estimating joint offsets from data is to write the equations with the joint offset as a parameter, e.g. for the double pendulum we would write the forward kinematics as:

$$\mathbf{p}_1 = l_1 \begin{bmatrix} \sin(\theta_1 + \bar{\theta}_1) \\ -\cos(\theta_1 + \bar{\theta}_1) \end{bmatrix}, \quad \mathbf{p}_2 = \mathbf{p}_1 + l_2 \begin{bmatrix} \sin(\theta_1 + \bar{\theta}_1 + \theta_2 + \bar{\theta}_2) \\ -\cos(\theta_1 + \bar{\theta}_1 + \theta_2 + \bar{\theta}_2) \end{bmatrix}.$$

We try to obtain independent measurements of the end-effector position (e.g. from motion capture, from perhaps some robot-mounted cameras, or from some mechanical calibration rig) with their corresponding joint measurements, to obtain data points of the form $\langle \mathbf{p}_2, \theta_1, \theta_2 \rangle$. Then we can solve a small nonlinear optimization problem to estimate the joint offsets to minimize a least-squares residual.

If independent measurements of the kinematics are not available, it it possible to estimate the offsets along with the dynamic parameters, using the trigonometric identities, e.g. $s_{\theta+\bar{\theta}} = s_\theta c_{\bar{\theta}} + c_\theta s_{\bar{\theta}}$, and then including the $s_{\bar{\theta}}, c_{\bar{\theta}}$ terms (separately) in the "lumped parameters" we discuss below.

### 18.2.2 Least-squares formulation (of the inverse dynamics).

Now let's thinking about estimating the dynamic parameters of multibody system. We've been writing the manipulation equations in the form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u} + \text{friction, etc.} \tag{2}$$

Each of the terms in this equation can depend on the parameters $\alpha$ that we're trying to estimate. But the parameters enter the multibody equations in a particular structured way: the equations are *affine in the* ***lumped parameters***. More precisely, the manipulator equations above can be factored into the form

$$\mathbf{W}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{u})\alpha_l(\alpha) + \mathbf{w}_0(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{u}) = 0,$$

where $\alpha_l$ are the "lumped parameters". We sometimes refer to $\mathbf{W}$ as the "data matrix".

**Example 18.2 (Lumped parameters for the simple pendulum)**

The now familiar equations of motion for the simple pendulum are

$$ml^2\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = \tau.$$

For parameter estimation, we will factor this into

$$\begin{bmatrix} \ddot{\theta} & \dot{\theta} & \sin\theta \end{bmatrix} \begin{bmatrix} ml^2 \\ b \\ mgl \end{bmatrix} - \tau = 0.$$

The terms $ml^2$, $b$, and $mgl$ together form the "lumped parameters".

It is worth taking a moment to reflect on this factorization. First of all, it does represent a somewhat special statement about the multibody equations: the nonlinearities enter only in a particular way. For instance, if I had terms in the equations of the form, $\sin(m\theta)$, then I would *not* be able to produce an affine decomposition separating $m$ from $\theta$. Fortunately, that doesn't happen in our mechanical systems [1]. Furthermore, this structure is particular to the *inverse dynamics*, as we have written here. If you were to write the forward dynamics, multiplying by $\mathbf{M}^{-1}$ in order to solve for $\ddot{\mathbf{q}}$, then once again you would destroy this affine structure.

This is super interesting! It is tempting to thing about parameter estimation for general dynamical systems in our standard state-space form: $\mathbf{x}[n+1] = f_\alpha(\mathbf{x}[n], \mathbf{u}[n])$. But for multibody systems, it seems that this would be the wrong thing to do, as it destroys this beautiful affine structure.

## Example 18.3 (Multibody parameters in DRAKE)

Very few robotics simulators have any way for you to access the parameters of the dynamics. In Drake, we explicitly declare all of the parameters of a multibody system in a separate data structure to make them available, and we can leverage Drake's symbolic engine to extract and manipulate the equations with respect to those variables.

As a simple example, I've loaded the cart-pole system model from URDF, created a symbolic version of the `MultibodyPlant`, and populated the `Context` with symbolic variables for the quantities of interest. Then I can evaluate the (inverse) dynamics in order to obtain my equations.

Open in Colab

The output looks like:

```
Symbolic dynamics:
(0.10000000000000001 * v(0) - u(0) + (pow(v(1), 2) * mp * l * sin(q(1))) + (vd(0) * mc) + 
(0.10000000000000001 * v(1) - (vd(0) * mp * l * cos(q(1))) + (vd(1) * mp * pow(l, 2)) + 9.8
```

Go ahead and compare these with the cart-pole equations that we derived by hand.

Drake offers a method `DecomposeLumpedParameters` that will take this expression and factor it into the affine expression above. For this cart-pole example, it extracts the lumped parameters $[m_c + m_p, m_p l, m_p l^2]$.

The existence of the lumped-parameter decomposition reveals that the *equation error* for lumped-parameter estimation, with the error taken in the torque coordinates, can be solved using least squares. As such, we can leverage all of the strong results and variants from linear estimation. For instance, we can add terms to regularize the estimate (e.g. to stay close to an initial guess), and we can write efficient recursive estimators for optimal online estimation of the parameters using recursive least-squares. My favorite recursive least-squares algorithm uses incremental QR factorization[2].

Importantly, because we have reduced this to a least-squares problem, we can also understand when it will *not* work. In particular, it is quite possible that some parameters cannot be estimated from any amount of joint data taken on the robot. As a simple example, consider a robotic arm bolted to a table; the inertial parameters of the first link of the robot will not be identifiable from any amount of joint data. Even on the second link, only the inertia relative to the first joint axis will be identifiable; the inertial parameters corresponding to the other dimensions will not. In our least-squares formulation, this is quite easy to understand: we simply check the (column) rank of the data matrix, $\mathbf{W}$. In particular, we can extract the **identifiable lumped parameters** by using, e.g., $\mathbf{R}_1\alpha_l$ from the QR factorization:

$$\mathbf{W} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}\begin{bmatrix}\mathbf{R}_1 \\ 0\end{bmatrix}, \quad \Rightarrow \quad \mathbf{W}\alpha_l = \mathbf{Q}_1(\mathbf{R}_1\alpha_l).$$

## Example 18.4 (Parameter estimation for the Cart-Pole)

Having extracted the lumped parameters from the URDF file above, we can now take this to fruition. I've kept the example simple: I've simulated the cart-pole for a few seconds running just simple sine wave trajectories at the input, then constructed the data matrix and performed the least squares fit.

Open in Colab

The output looks like this:

```
Estimated Lumped Parameters:
(mc + mp).  URDF: 11.0,  Estimated: 10.905425349337081
(mp * l).  URDF: 0.5,  Estimated: 0.5945797067753813
(mp * pow(l, 2)).  URDF: 0.25,  Estimated: 0.302915745122919
```

Note that we could have easily made the fit more accurate with more data (or more carefully selected data).

Should we be happy with only estimating the (identifiable) lumped parameters? Isn't it the true original parameters that we are after? The linear algebra decomposition of the data matrix (assuming we apply it to a sufficiently rich set of data), is actually revealing something fundamental for us about our system dynamics. Rather than feel disappointed that we cannot accurately estimate some of the parameters, we should embrace that *we don't need to estimate those parameters* for any of the dynamic reasoning we will do about our equations (simulation, verification, control design, etc). The identifiable lumped parameters are precisely the subset of the lumped parameters that we need.

For practical reasons, it might be convenient to take your estimates of the lumped parameters, and try to back out the original parameters (for instance, if you want to write them back out into a URDF file). For this, I would recommend a final post-processing step that e.g. finds the parameters $\hat{\alpha}$ that are as close as possible (e.g. in the least-squares sense) to your original guess for the parameters, subject to the nonlinear constraint that $\mathbf{R}_1 \alpha_l(\hat{\alpha})$ matches the estimated identifiable lumped parameters.

There are still a few subtleties worth considering, such as how we parameterize the inertial matrices. Direct estimation of the naive parameterization, the six entries of a symmetric 3x3 matrix, can lead to non-physical inertial matrices. [3] describes a parameter estimation formulation that includes a convex formulation of the physicality constraints between these parameters.

### 18.2.3 Identification using energy instead of inverse dynamics.

In addition to leveraging tools from linear algebra, there are a number of other refinements to the basic recipe that leverage our understanding of mechanics. One important example is the "energy formulations" of parameter estimation [4].

We have already observed that evaluating the equation error in torque space (inverse dynamics) is likely better than evaluating it in state space space (forward dynamics), because we can factorized the inverse dynamics. But this property is not exclusive to inverse dynamics. The total energy of the system (kinetic + potential) is also affine in the lumped parameters. We can use the relation

$$\dot{E}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \dot{\mathbf{q}}^T(\mathbf{B}\mathbf{u} + \text{ friction}, \quad ...).$$

Why might we prefer to work in energy coordinates rather than torque? The differences are apparent in the detailed numerics. In the torque formulation, we find ourselves using $\ddot{\mathbf{q}}$ directly. Conventional wisdom holds that joint sensors can reliably report joint positions and velocities, but that joint accelerations, often obtained by numerically differentiating *twice*, tend to be much more noisy. In some cases, it might be numerically better to apply finite differences to the total energy of the system rather than to the individual joints †. [5] provides a study of various filtering formulations which leads to a recommendation in [4] that the energy formulation

† Sometimes these methods are written a

tends to be numerically superior.

### 18.2.4 Residual physics models with linear function approximators

The term "residual physics" has become quite popular recently (e.g. [6]) as people are looking for ways to combine the modeling power of deep neural networks with our best tools from mechanics. But actually this idea is quite old, and there is a natural class of residual models that fit very nicely into our least-squares lumped-parameter estimation. Specifically, we can consider models of the form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \alpha_r \mathbf{\Phi}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{B}\mathbf{u} + \text{friction}, \quad \text{etc.}, \tag{3}$$

with $\alpha_r$ the additional parameters of the residual and $\mathbf{\Phi}$ a set of (fixed) nonlinear basis functions. The hope is that these residual models can capture any "slop terms" in the dynamics that are predictable, but which we did not include in our original parametric model. Nonlinear friction and aerodynamic drag are commonly cited examples.

Common choices for these basis functions, $\mathbf{\Phi}(\mathbf{q},\dot{\mathbf{q}})$, for use with the manipulator equations include radial basis functions[7] or wavelets [8]. Although allowing the basis functions to depend on $\ddot{\mathbf{q}}$ or $\mathbf{u}$ would be fine for the parameter estimation, we tend to restrict them to $\mathbf{q}$ and $\dot{\mathbf{q}}$ to maintain some of the other nice properties of the manipulator equations (e.g. control-affine).

Due to the maturity of least-squares estimation, it is also possible to use least-squares to effectively determine a subset of basis functions that effectively describe the dynamics of your system. For example, in [9], we applied least-squares to a wide range of physically-inspired basis functions in order to make a better ODE approximation of the post-stall aerodynamics during perching, and ultimately discarded all but the small number of basis functions that best described the data. Nowadays, we could apply algorithms like LASSO for least-squares regression with an $\ell_1$-regularization, or [10] uses an alternative based on sequential thresholded least-squares.

### 18.2.5 Experiment design as a trajectory optimization

One key assumption for any claims about our parameter estimation algorithms recovering the true identifiable lumped parameters is that the data set was sufficiently rich; that the trajectories were "parametrically exciting". Basically we need to assume that the trajectories produced motion so that the data matrix, $\mathbf{W}$ contains information about all of the identifiable lumped parameters. Thanks to our linear parameterization, we can evaluate this via numerical linear algebra on $\mathbf{W}$.

Moreover, if we have an opportunity to change the trajectories that the robot executes when collecting the data for parameter estimation, then we can design trajectories which try to maximally excite the parameters, and produce a numerically-well conditioned least squares problem. One natural choice is to minimize the condition number of the data matrix, $\mathbf{W}$. The condition number of a matrix is the ratio of the largest to smallest singular values $\frac{\sigma_{max}(\mathbf{W})}{\sigma_{min}(\mathbf{W})}$. The condition number is always greater than one, by definition, and the lower the value the better the condition. The condition number of the data matrix is a nonlinear function of the data taken over the entire trajectory, but it can still be optimized in a nonlinear trajectory optimization ([1], § 12.3.4).

### 18.2.6 Online estimation and adaptive control

The field of adaptive control is a huge and rich literature; many books have been written on the topic (e.g [11]). Allow me to make a few quick references to that

literature here.

I have mostly discussed parameter estimation so far as an offline procedure, but one important approach to adaptive control is to perform online estimation of the parameters as the controller executes. Since our estimation objective can be linear in the (lumped) parameters, this often amounts to the recursive least-squares estimation. To properly analyze a system like this, we can think of the system as having an augmented state space, $\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \alpha \end{bmatrix}$, and study the closed-loop dynamics of the state and parameter evolution jointly. Some of the strongest results in adaptive control for robotic manipulators are confined to fully-actuated manipulators, but for instance [12] gives a nice example of analyzing an adaptive controller for underactuated systems using many of the tools that we have been developing in these notes.

As I said, adaptive control is a rich subject. One of the biggest lessons from that field, however, is that one may not need to achieve convergence to the true (lumped) parameters in order to achieve a task. Many of the classic results in adaptive control make guarantees about task execution but explicitly do *not* require/guarantee convergence in the parameters.

### 18.2.7 Identification with contact

Can we apply these same techniques to e.g. walking robots that are making and breaking contact with the environment?

There is certainly a version of this problem that works immediately: if we know the contact Jacobians and have measurements of the contact forces, then we can add these terms directly into the manipulator equations and continued with the least-squares estimation of the lumped parameters, even including frictional parameters.

One can also study cases where the contact forces are not measured directly. For instance, [13] studies the extreme case of identifiability of the inertial parameters of a passive object with and without explicit contact force measurements.

## 18.3 Identifying (time-domain) linear dynamical systems

If multibody parameter estimation forms the first relevant pillar of established work in system identification, then identification of linear systems forms the second. Linear models have been a primary focus of system identification research since the field was established, and have witnessed a resurgence in popularity during just the last few years as new results from machine learning have contributed new bounds and convergence results, especially in the finite-sample regime (e.g. [14, 15, 16, 17]).

A significant portion of the linear system identification literature (e.g. [18]) is focused on identifying linear models in the frequency domain. Indeed, transfer-function realizations provide important insights and avoid some of the foibles that we'll need to address with state-space realizations. However, I will focus my attention in this chapter on time-domain descriptions of linear dynamical systems; some of the lessons here are easier to generalize to nonlinear dynamics (plus we unfortunately haven't built the foundations for the frequency domain techniques yet in these notes).

### 18.3.1 From state observations

Let's start our treatment with the easy case: fitting a linear model from direct (potentially noisy) measurements of the state. Fitting a discrete-time model,

$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n] + \mathbf{w}[n]$, to sampled data $(\mathbf{u}_n, \mathbf{x}_n = \mathbf{x}[n] + \mathbf{v}[n])$ using the *equation-error* objective is just another *linear* least-squares problem. Typically, we form some data matrices and write our least-squares problem as:

$$\mathbf{X}' \approx \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix} \begin{bmatrix}\mathbf{X} \\ \mathbf{U}\end{bmatrix}, \qquad \text{where}$$

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{N-2} \\ | & | & & | \end{bmatrix}, \quad \mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{N-1} \\ | & | & & | \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{N-2} \\ | & | & & | \end{bmatrix}.$$

By the virtues of linear least squares, this estimator is unbiased with respect to the uncorrelated process and/or measurement noise, $\mathbf{w}[n]$ and $\mathbf{v}[n]$.

### Example 18.5 (Cart-pole balancing with an identified model)

I've provide a notebook demonstrating what a practical application of linear identification might look like for the cart-pole system, in a series of steps. First, I've designed an LQR balancing controller, but using the *wrong* parameters for the cart-pole (I changed the masses, etc). This LQR controller is enough to keep the cart-pole from falling down, but it doesn't converge nicely to the upright. I wanted to ask the question, can I use data generated from this experiment to identify a better linear model around the fixed-point, and improve my LQR controller?

Interestingly, the simple answer is "no". If you only collect the input and state data from running this controller, you will see that the least-squares problem that we formulate above is rank-deficient. The estimated $\mathbf{A}$ and $\mathbf{B}$ matrices, denoted $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ describe the data, but do not reveal the true linearization. And if you design a controller based on these estimates, you will be disappointed!

Fortunately, we could have seen this problem by checking the rank of the least-squares solution. Generating more examples won't fix this problem. Instead, to generate a richer dataset, I've added a small additional signal to the input: $u(t) = \pi_{lqr}(\mathbf{x}(t)) + 0.1\sin(t)$. That makes all of the difference.


I hope you try the code. The basic algorithm for estimation is disarmingly simple, but there are a lot of details to get right to actually make it work.

## Model-based Iterative Learning Control (ILC)

### Example 18.6 (The Hydrodynamic Cart-Pole)

One of my favorite examples of model-based ILC was a series of experiments in which we explored the dynamics of a "hydrodynamic cart-pole" system. Think of it as a cross between the classic cart-pole system and a fighter jet (perhaps a little closer to the cartpole)!

Here we've replaced the pole with an airfoil (hydrofoil), turned the entire system on its side, and dropped it into a water tunnel. Rather than swing-up and balance the pole against gravity, the task is to balance the foil in its unstable configuration against the hydrodynamic forces. These forces are the result of unsteady fluid-body interactions; unlike the classic cart-pole system, this time we do not have an tractable parameterized ODE model for the system. It's a perfect problem for system identification and ILC.

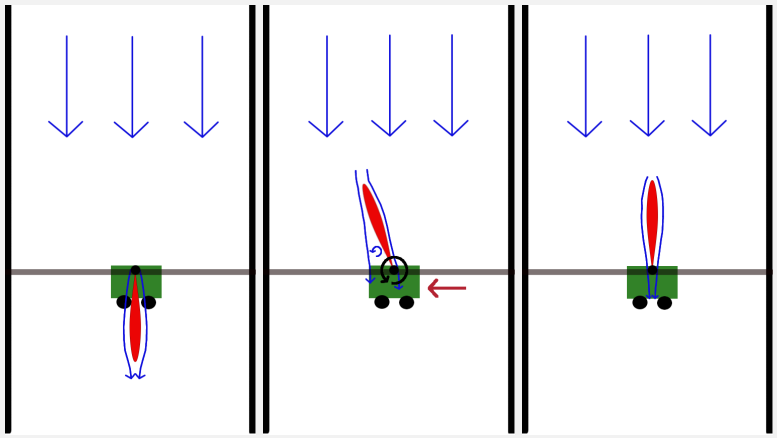


Figure 18.2 - A cartoon of the hydronamic cart-pole system. The cart is actuated horizontally, the foil pivots around a passive joint, and the fluid is flowing in the direction of the arrows. (The entire system is horizontal, so there is no effect from gravity.) The aerodynamic center of the airfoil is somewhere in the middle of the foil; because the pin joint is at the tail, the passive system will "weather vane" to the stable "downward" equilibrium (left). Balancing corresponds to stabilizing the unstable "upward" equilibrium (right). The fluid-body dynamics during the transition (center) are unsteady and very nonlinear.

In a series of experiments, first we attempted to stabilize the system using an LQR controller derived with an approximate model (using flat-plate theory). This controller didn't perform well, but was just good enough to collect relevant data in the vicinity of the unstable fixed point. Then we fit a linear model, recomputed the LQR controller using the model, and got notably better performance.

To investigate more aggressive maneuvers we considered making a rapid step change in the desired position of the cart (like a fighter jet rapidly changing altitude). Using only the time-invariant LQR balancing controller with a shifted set point, we naturally observed a very slow step-response. Using trajectory optimization on the time-invariant linear model used in balancing, we could do much better. But we achieved considerably better performance by iteratively fitting a time-varying linear model in the neighborhood of this trajectory and performing model-based ILC.

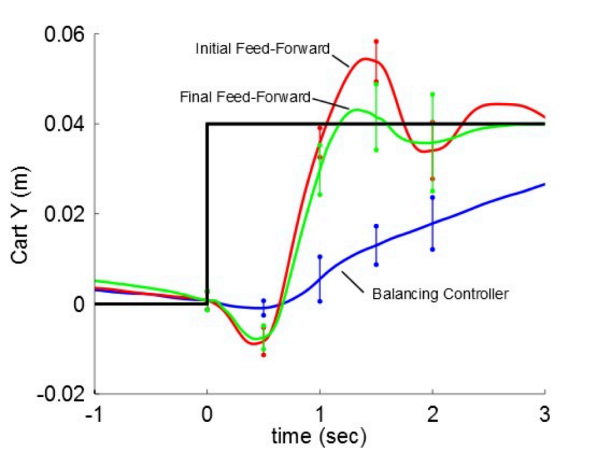



Figure 18.3 - Comparison of the step response using three different controllers: the balancing LQR controller (blue), LQR with a feed-forward term obtained from trajectory optimization with the LTI model (red), and a controller obtained via ILC with a time-varying linear model and iLQR (green).

These experiments were quite beautiful and very visual. They went on from here to consider the effect of stabilizing against incoming vortex disturbances using

real-time perception of the oncoming fluid. If you're at all interested, I would encourage you to check out John Robert's thesis[19] and/or even the video of his thesis defense.

## Compression using the dominant eigenmodes

For high-dimensional state or input vectors, one can use singular value decomposition (SVD) to solve this least-squares problem using only the dominant eigenvalues (and corresponding eigenvectors) of $\mathbf{A}$ [20, Section 7.2] using the so-called "Dynamic Mode Decomposition" (DMD). There have been many empirical success stories of using a small number of dominant modes to produce an excellent fit to the data (this is especially relevant in problems like fluid dynamics where the state vector $\mathbf{x}$ might be an entire image corresponding to a fluid flow). In DMD, we would write the linear dynamics in the coordinates of these eigenmodes (which can always be projected back to the full coordinates).

## Linear dynamics in a nonlinear basis

A potentially useful generalization that we can consider here is identifying dynamics that evolve linearly in a coordinate system defined by some (potentially high-dimensional) basis vectors $\phi(\mathbf{x})$. In particular, we might consider dynamics of the form $\dot{\phi} = \frac{\partial \phi}{\partial \mathbf{x}}\dot{\mathbf{x}} = \mathbf{A}\phi(x)$. Much ado has been made of this particular form, due to the connection to Koopman operator theory that we will discuss briefly below. For our purposes, we also need a control input, so might consider a form like $\dot{\phi} = \mathbf{A}\phi(x) + \mathbf{B}\mathbf{u}$.

Once again, thanks to the maturity of least-squares, with this approach it is possible to include list many possible basis functions, then use sparse least-squares and/or the modal decomposition to effectively find the important subset.

Note that multibody parameter estimation described above is not this, although it is closely related. The least-squares lumped-parameter estimation for the manipulator equations uncovered dynamics that were still *nonlinear* in the state variables.

## 18.3.2 From input-output data (the state-realization problem)

In the more general form, we would like to estimate a model of the form

$$\begin{aligned}\mathbf{x}[n+1] &= \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n] + \mathbf{w}[n]\\ \mathbf{y}[n] &= \mathbf{C}\mathbf{x}[n] + \mathbf{D}\mathbf{u}[n] + \mathbf{v}[n].\end{aligned}$$

Once again, we will apply least-squares estimation, but combine this with the famous "Ho-Kalman" algorithm (also known as the "Eigen System Realization" (ERA) algorithm) [21]. My favorite presentation of this algorithm is [16].

First, observe that

$$\begin{aligned}\mathbf{y}[0] =& \mathbf{C}\mathbf{x}[0] + \mathbf{D}\mathbf{u}[0] + \mathbf{v}[0],\\ \mathbf{y}[1] =& \mathbf{C}(\mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0] + \mathbf{w}[0]) + \mathbf{D}\mathbf{u}[1] + \mathbf{v}[1],\\ \mathbf{y}[n] =& \mathbf{C}\mathbf{A}^n\mathbf{x}[0] + \mathbf{D}\mathbf{u}[n] + \mathbf{v}[n] + \sum_{k=0}^{n-1}\mathbf{C}\mathbf{A}^{n-k-1}(\mathbf{B}\mathbf{u}[k] + \mathbf{w}[k]).\end{aligned}$$

For the purposes of identification, let's write $y[n]$ as a function of the most recent $N+1$ inputs (for $k \geq N$):

$$\begin{aligned}\mathbf{y}[n] =& \begin{bmatrix}\mathbf{C}\mathbf{A}^{N-1}\mathbf{B} & \mathbf{C}\mathbf{A}^{N-2}\mathbf{B} & \cdots & \mathbf{C}\mathbf{B} & \mathbf{D}\end{bmatrix} \begin{bmatrix}\mathbf{u}[n-N] \\ \mathbf{u}[n-N+1] \\ \vdots \\ \mathbf{u}[n]\end{bmatrix} + \delta[n] \\ =& \mathbf{G}[n]\bar{\mathbf{u}}[n] + \delta[n]\end{aligned}$$

where $\delta[n]$ captures the remaining terms from initial conditions, noise and control inputs before the (sliding) window. $\mathbf{G}[n] = \begin{bmatrix}\mathbf{C}\mathbf{A}^{N-1}\mathbf{B} & \mathbf{C}\mathbf{A}^{N-2}\mathbf{B} & \cdots & \mathbf{C}\mathbf{B} & \mathbf{D}\end{bmatrix}$, and $\bar{\mathbf{u}}[n]$ represents the concatenated $\mathbf{u}[n]$'s from time $n-N$ up to $n$. Importantly we have that $\delta[n]$ is uncorrelated with $\bar{\mathbf{u}}[n]$: $\forall k > n$ we have $E[\mathbf{u}_k \delta_n] = 0$. This is sometimes known as Neyman orthogonality, and it implies that we can estimate $\mathbf{G}$ using simple least-squares $\hat{\mathbf{G}} = \text{argmin}_{\mathbf{G}} \sum_{n \geq N} \|\mathbf{y}_n - \mathbf{G}\bar{\mathbf{u}}_n\|^2$. [16] gives bounds on the norm of the estimation error as a function of the number of samples and the variance of the noise.

How should we pick the window size $N$? By Neyman orthogonality, we know that our estimates will be unbiased for any choice of $N \geq 0$. But if we choose $N$ too small, then the term $\delta[n]$ will be large, leading to a potentially large variance in our estimate. For stable systems, the $\delta$ terms will get smaller as we increase $N$. In practice, we choose $N$ based on the characteristic settling time in the data (roughly until the impulse response becomes sufficiently small).

If you've studied linear systems, $\mathbf{G}$ will look familiar; it is precisely this (multi-input, multi-output) matrix impulse response, also known as the "Markov parameters". In fact, estimating $\hat{\mathbf{G}}$ may even be sufficient for control design, as it is closely-related to the parameterization used in disturbance-based feedback for partially-observable systems [22, 23]. But the Ho-Kalman algorithm can be used to extract good estimates $\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}, \hat{\mathbf{D}}$ with state-dimension $\dim(\mathbf{x}) = n_x$ from $\hat{\mathbf{G}}$, assuming that the true system is observable and controllable with order at least $n_x$ and the data matrices we form below are sufficiently rich[16].

It is important to realize that many system matrices, $\mathbf{A}, \mathbf{B}, \mathbf{C}$, can describe the same input-output data. In particular, for any invertible matrix (aka similarity transform), $\mathbf{T}$, the system matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ with,

$$\mathbf{A}' = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}, \quad \mathbf{B}' = \mathbf{T}^{-1}\mathbf{B}, \quad \mathbf{C}' = \mathbf{C}\mathbf{T},$$

describe the same input-output behavior. The Ho-Kalman algorithm returns a *balanced realization*. A balanced realization is one in which we have effectively applied a similarity transform, $\mathbf{T}$, which makes the controllability and observability Grammians equal and diagonal[20, Ch. 9] and which orders the states in terms of diminishing effect on the input/output behavior. This ordering is relevant for determining the system order and for model reduction.

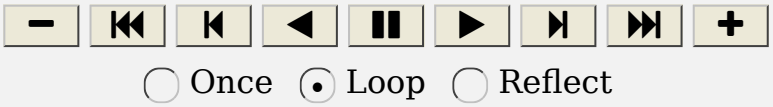
Note that $\hat{\mathbf{D}}$ is the last block in $\hat{\mathbf{G}}$ so is extracted trivially. The Ho-Kalman algorithm tells us how to extract $\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}$, with another application of the SVD on suitably constructed data matrices (see e.g. [16, §5.1], [24, §10.5], or [20, §9.3]).

### Example 18.7 (Ho-Kalman identification of the cart-pole from keypoints)

Let's repeat the cart-pole example. But this time, instead of getting observations of the joint position and velocities directly, we will consider the problem of identifying the dynamics from a camera rendering the scene, but let us proceed thoughtfully.

The output of a standard camera is an RGB image, consisting of 3 x width x height real values. We could certainly columnate these into a vector and use this as the outputs/observations, $y(t)$. But I don't recommend it. This "pixel-space" is not a

nice space. For example, you could easily find a pixel that has the color of the cart in one frame, but after an incremental change in the position of the cart, now (discontinuously) takes on the color of the background. Deep learning has given us fantastic new tools for transforming raw RGB images into better "feature spaces", that will be robust enough to deploy on real systems but can make our modeling efforts much more tractable. My group has made heavy use of "keypoint networks" [25] and self-supervised "dense correspondences" [26, 27, 28] to convert RGB outputs into a more consumable form.



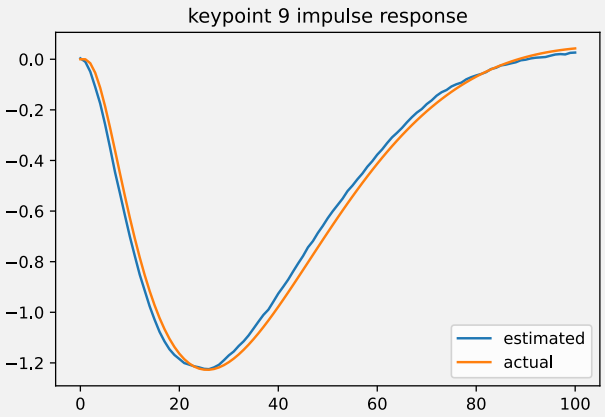
◯ Once ◉ Loop ◯ Reflect

The existence of these tools makes it reasonable to assume that we have observations representing the 2D positions of some number of "key points" that are rigidly fixed to the cart and to the pole. The location of any keypoint $K$ on a pole, for instance, is given by the cart-pole kinematics:

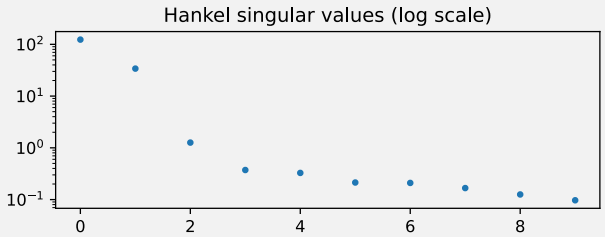
$$ {}^{W}\mathbf{p}^{K} = \begin{bmatrix} x \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} {}^{P}\mathbf{p}^{K}, $$

where $x$ is the position of the cart, and $\theta$ is the angle of the pole. I've used Monogram notation to denote that ${}^{P}\mathbf{p}^{K}$ is the Cartesian position of a point $A$ in the pole's frame, $P$, and ${}^{W}\mathbf{p}^{K}$ is that same position in the camera/world frame, $W$. While it is probably unreasonable to linearize the RGB outputs as a function of the cart-pole positions, $x$ and $\theta$, it *is* reasonable to approximate the keypoint positions by linearizing this equation for small angles.

You'll see in the example that we recover the impulse response quite nicely.




The big question is: does Ho-Kalman discover the 4-dimensional state vector that we know and like for the cart-pole? Does it find something different? We would typically choose the order by looking at the magnitude of the singular values of the Hankel data matrix. In this example, in the case of no noise, you see clearly that 2 states describe most of the behavior, and we have diminishing returns after 4 or 5:




As an exercise, try adding process and/or measurement noise to the system that generated the data, and see how they effect this result.

Open in Colab

## 18.3.3 Adding stability constraints

Details coming soon. See, for instance [29].

## 18.3.4 Autoregressive models

Another important class of linear models predict the output directly from a history of recent inputs and outputs:

$$\begin{aligned}\mathbf{y}[n+1] =& \mathbf{A}_0\mathbf{y}[n] + \mathbf{A}_1\mathbf{y}[n-1] + \ldots + \mathbf{A}_k\mathbf{y}[n-k] \\ &+ \mathbf{B}_0\mathbf{u}[n] + \mathbf{B}_1\mathbf{u}[n-1] + \ldots \mathbf{B}_k\mathbf{u}[n-k]\end{aligned}$$

These are the so-called "AutoRegressive models with eXogenous input (ARX)" models. The coefficients of ARX models can be fit directly from input-output data using linear least squares.

While it is certainly possible to think of these models as state-space models, where we collect the finite history $\mathbf{y}$ and $\mathbf{u}$ (except not $\mathbf{u}[n]$) into our state vector. But this is not necessarily a very efficient representation; the Ho-Kalman algorithm might be able to find a much smaller state representation. This is particularly important for partially-observable systems that might require a very long history, $k$.

**Example 18.8 (An ARX model of cart-pole keypoint dynamics)**

## 18.4 Identification of finite (PO)MDPs

There is one more case that we can understand well: when states, actions, and observations (and time) are all discrete. Recall that in the very beginnings of our discussion about optimal control and dynamic programming, we used graph search and discretization of the state space to give a particularly nice version of the value iteration algorithm. In general, we can write the *stochastic* dynamics using conditional probabilities:

$$\Pr(s[n+1]|s[n], a[n]), \qquad \Pr(o[n]|s[n], a[n]),$$

where I've (again) used $s$ for discrete states, $a$ for discrete actions, and $o$ for discrete observations. With everything discrete, these conditional probabilities are represented naturally with matrices/tensors. We'll use the tensors $T_{ijk} \equiv \Pr(s[n+1] = s_i | s[n] = s_j, a[n] = a_k)$ to denote the transition matrix, and $O_{ijk} \equiv \Pr(o[n] = o_i | s[n] = s_j, a[n] = a_k)$ for the observation matrix.

### 18.4.1 From state observations

Following analogously to our discussion on linear dynamical systems, the first case to understand is when we have direct access to state and action trajectories: $s[\cdot], a[\cdot]$. This corresponds to identifying a Markov chain or Markov decision process (MDP). The transition matrices can be extracted directly from the statistics of the transitions:

$$T_{ijk} = \frac{\text{number of transitions to } s_i \text{ from } s_j, a_k}{\text{total number of transitions from } s_j, a_k}.$$

Not surprisingly, for this estimate to converge to the true transition probabilities asymptotically, we need the identification (exploration) dynamics to be ergodic (for every state/action pair to be visited infinitely often).

For large MDPs, analogous to the modal decomposition that we described for linear dynamical systems, we can also consider factored representations of the transition matrix. [Coming soon...]

### 18.4.2 Identifying Hidden Markov Models (HMMs)

## 18.5 Neural network models

If multibody parameter estimation, linear system identification, and finite state systems are the first pillars of system identification, then (at least these days) deep learning is perhaps the last major pillar. It's no coincidence that I've put this section just after the section on linear dynamical systems. Neural networks are powerful tools, but they can be a bit harder to think about carefully. Thankfully, we'll see that many of the basic lessons and insights from linear dynamical systems still apply here. In particular, we can follow the same progression: learning the dynamics function directly from state observations, learning state-space models from input-output data (with recurrent networks), and learning input-output (autoregressive) models with feedforward networks using a history of outputs.

Deep learning tools allow us to quite reliably tune the neural networks to fit our training data. The major challenges are with respect to data efficiency/ generalization, and with respect to actually designing planners / controllers based

on these models that have such complex descriptions.

### 18.5.1 Generating training data

One extremely interesting question that arises when fitting rich nonlinear models like neural networks is the question of how to generate the training data. You might have the general sense that we would like data that provides some amount of "coverage" of the state space; this is particularly challenging for underactuated systems since we have dynamic constraints restricting our trajectories in state space.

For multibody parameter estimation we discussed using the condition number of the data matrix as an explicit objective for experiment design. This was a luxury of using *linear* least-squares optimization for the regression and it takes advantage of the specialized knowledge we have from the manipulator equations about the model structure. Unfortunately, we don't have an analogous concept in the more general case of nonlinear least squares with general function approximators. This approach also works for identifying linear dynamical systems. The challenge is here perhaps considered less severe since for linear models in the noise-free case even data generated from the impulse response is sufficient for identification.

This topic has received considerable attention lately in the context of *model-based reinforcement learning* (e.g. [30]). Broadly speaking, in the ideal case we would like the training data we use for system identification to match the distribution of data that we will encounter when executing the optimal policy. But one cannot know this distribution until we've designed our controller, which (in our current discussion) requires us to have a model. It is a classic "chicken and the egg" problem. In most cases, it speaks to the importance to interleaving system identification and control design instead of the simpler notion of performing identification once and then using the model forevermore.

### 18.5.2 From state observations

### 18.5.3 State-space models from input-output data (recurrent networks)

### 18.5.4 Input-output (autoregressive) models

## 18.6 Alternatives for nonlinear system identification

## 18.7 Identification of hybrid systems

## 18.8 Task-relevant models

## 18.9 Exercises

### Exercise 18.1 (Linear System Identification with different objective functions)

Consider a discrete-time linear system of the form

$$x[n+1] = Ax[n] + Bu[n]$$

where $x[n] \in \mathbf{R}^p$ and $u[n] \in \mathbf{R}^q$ are state and control at step $n$. The system matrix $A \in \mathbf{R}^{p \times p}$ and $B \in \mathbf{R}^{p \times q}$ are unknown parameters of the model, and your task is to identify the parameters given a simulated trajectory. Using the trajectory simulation provided in this python notebook, implement the solution for the following problems.

a. Identify the model parameters by solving

$$\min_{A,B} \sum_{n=0}^{N-1} \|x[n+1] - Ax[n] - Bu[n]\|_2^2.$$

b. Identify the model parameters by solving

$$\min_{A,B} \sum_{n=0}^{N-1} \|x[n+1] - Ax[n] - Bu[n]\|_\infty.$$

c. Identify the model parameters by solving

$$\min_{A,B} \sum_{n=0}^{N-1} \|x[n+1] - Ax[n] - Bu[n]\|_1.$$

## Exercise 18.2 (System Identification for the Perching Glider)

In this exercise we will use physically-inspired basis functions to fit the nonlinear dynamics of a perching glider. In this python notebook, you will need to implement least-squares fitting and find the best set of basis functions that describe the dynamics of the glider. Take the time to go through the notebook and understand the code in it, and then answer the following questions. The written question will also be listed in the notebook for your convenience.

a. Work through the coding sections in the notebook.
b. All of the basis configurations we tested used at most 3 basis functions to compute a single acceleration. If we increase the number of basis functions used to compute a single acceleration to 4, the least-squares residual goes down. Why would we limit ourselves to 3 basis functions if by using more we can generate a better fit?

## References


1. W Khalil and E Dombre, "Modeling, Identification and Control of Robots",Elsevier , 2004.

2. M. Kaess and A. Ranganathan and F. Dellaert, "i{SAM}: Incremental Smoothing and Mapping", *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365-1378, 2008.

3. Patrick M Wensing and Sangbae Kim and Jean-Jacques E Slotine, "Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution", *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 60--67, 2017.

4. Maxime Gautier, "Dynamic identification of robots with power model", *Proceedings of the {IEEE} International Conference on Robotics and Automation*, vol. 3, pp. 1922--1927 vol.3, 1997.

5. Maxime Gautier, "A comparison of filtered models for dynamic identification of robots", *Proceedings of the 35th {IEEE} Conference on Decision and Control*, vol. 1, pp. 875â€“880, 1996.

6. Andy Zeng and Shuran Song and Johnny Lee and Alberto Rodriguez and Thomas Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics", *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307--1319, 2020.

7. R. M. Sanner and J. E. Slotine, "Gaussian Networks for Direct Adaptive Control", *1991 American Control Conference*, pp. 2153-2159, 1991.

8. Robert M Sanner and Jean-Jacques E Slotine, "Structurally dynamic wavelet networks for adaptive control of robotic systems", *International Journal of Control*, vol. 70, no. 3, pp. 405--421, 1998.

9. Warren Hoburg and Russ Tedrake, "System Identification of Post Stall Aerodynamics for {UAV} Perching", *Proceedings of the AIAA Infotech@Aerospace Conference*, April, 2009. [ link ]

10. Steven L Brunton and Joshua L Proctor and J Nathan Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems", *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932--3937, 2016.

11. Karl J. Åström and Björn Wittenmark, "Adaptive {Control}: {Second} {Edition}",Courier Corporation , apr, 2013.

12. Joseph Moore and Russ Tedrake, "Adaptive Control Design for Underactuated Systems Using Sums-of-Squares Optimization", *Proceedings of the 2014 American Control Conference (ACC)*, June, 2014. [ link ]

13. Nima Fazeli and Roman Kolbert and Russ Tedrake and Alberto Rodriguez, "Parameter and contact force estimation of planar rigid-bodies undergoing frictional contact", *International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1437-1454, 2017. [ link ]

14. Moritz Hardt and Tengyu Ma and Benjamin Recht, "Gradient descent learns linear dynamical systems", *arXiv preprint arXiv:1609.05191*, 2016.

15. Elad Hazan and Holden Lee and Karan Singh and Cyril Zhang and Yi Zhang, "Spectral filtering for general linear dynamical systems", *arXiv preprint arXiv:1802.03981*, 2018.

16. Samet Oymak and Necmiye Ozay, "Non-asymptotic identification of lti systems from a single trajectory", *2019 American control conference (ACC)*, pp. 5655--5661, 2019.

17. Max Simchowitz and Ross Boczar and Benjamin Recht, "Learning linear dynamical systems with semi-parametric least squares", *Conference on Learning Theory*, pp. 2714--2802, 2019.

18. L. Ljung, "System Identification: Theory for the User",Prentice Hall , 1999.

19. John W. Roberts, "Control of Underactuated Fluid-Body Systems with Real-Time Particle Image Velocimetry", PhD thesis, Massachusetts Institute of Technology, June, 2012. [ link ]

20. Steven L Brunton and J Nathan Kutz, "Data-driven science and engineering: Machine learning, dynamical systems, and control",Cambridge University Press , 2019.

21. BL Ho and Rudolf E K{\'a}lm{\'a}n, "Effective construction of linear state-variable models from input/output functions", *at-Automatisierungstechnik*,

vol. 14, no. 1-12, pp. 545--548, 1966.

22. Sadra Sadraddini and Russ Tedrake, "Robust Output Feedback Control with Guaranteed Constraint Satisfaction", *In the Proceedings of 23rd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 12, April, 2020. [ link ]

23. Max Simchowitz and Karan Singh and Elad Hazan, "Improper learning for non-stochastic control", *Conference on Learning Theory*, pp. 3320--3436, 2020.

24. Jer-Nan Juang and Minh Q. Phan, "Identification and {Control} of {Mechanical} {Systems}",Cambridge University Press , aug, 2001.

25. Lucas Manuelli* and Wei Gao* and Peter Florence and Russ Tedrake, "{kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation}", *arXiv e-prints*, pp. arXiv:1903.06684, Mar, 2019. [ link ]

26. Peter R. Florence* and Lucas Manuelli* and Russ Tedrake, "Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation", *Conference on Robot Learning (CoRL)*, October, 2018. [ link ]

27. Peter Florence and Lucas Manuelli and Russ Tedrake, "Self-Supervised Correspondence in Visuomotor Policy Learning", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 492-499, April, 2020. [ link ]

28. Lucas Manuelli and Yunzhu Li and Pete Florence and Russ Tedrake, "Keypoints into the Future: Self-Supervised Correspondence in Model-Based Reinforcement Learning", *Conference on Robot Learning (CoRL)*, 2020. [ link ]

29. Jack Umenberger and Johan W{\aa}gberg and Ian R Manchester and Thomas B Sch{\"o}n, "Maximum likelihood identification of stable linear dynamical systems", *Automatica*, vol. 96, pp. 280--292, 2018.

30. Alekh Agarwal and Nan Jiang and Sham M. Kakade and Wen Sun, "Reinforcement Learning: Theory and Algorithms",Online Draft , 2020.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.



# Chapter 19
# State Estimation

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# CHAPTER 20
# Model-Free Policy Search

Reinforcement learning (RL) is a collection of algorithms for solving the same optimal control problem that we've focused on through the text, but the real gems from the RL literature are the algorithms for (almost) *black-box optimization* of stochastic optimal control problems. The idea of an algorithm that only has a "black-box" interface to the optimization problem means that it can obtain (potentially noisy) samples of the optimal cost via trial and error, but does not have access to the underlying model, and does not have direct access to complete gradient information.

This is a hard problem! In general we cannot expect RL algorithms to optimize as quickly as more structured optimization, and we can typically only guarantee convergence to a local optima at best. But the framework is extremely general, so it can be applied to problems that are inaccessible to any of the other algorithms we've examined so far. My favorite examples for reinforcement learning are control in complex fluid dynamics (e.g. [1]). These systems are often very difficult to model, or the model is so high-dimensional and complicated so that it's prohibitive for control design. In these problems, it might actually be faster to optimize via trial-and-error in physical experiments.

In this chapter we will examine one particular style of reinforcement learning that attempts to find good controllers by explicitly parameterizing a family of policies (e.g. via a parameter vector $\alpha$), then searching directly for the parameters that optimize the long-term cost. For a stochastic optimal control problem with our favorite additive form of the objective, this might look like:

$$\min_{\alpha} E\left[\sum_{n=0}^{N} \ell(\mathbf{x}[n], \mathbf{u}[n])\right] \tag{1}$$

where the random variables are drawn from probability densities

$$\begin{aligned} \mathbf{x}[0] &\sim p_0(\mathbf{x}), \\ \mathbf{x}[n] &\sim p(\mathbf{x}[n]|\mathbf{x}[n-1], \mathbf{u}[n-1]), \\ \mathbf{u}[n] &\sim p_\alpha(\mathbf{u}[n]|\mathbf{x}[n]). \end{aligned}$$

The last equation is a probabilistic representation of the control policy -- on each time-step actions $\mathbf{u}$ are drawn from a distribution that is conditioned on the current state, $\mathbf{x}$.

Of course, the controls community has investigated ideas like this, too, for instance under the umbrellas of *extremum-seeking control* and *iterative learning control*. I'll try to make connections whenever possible.

## 20.1 Policy Gradient Methods

One of the standard approaches to policy search in RL is to estimate the gradient of the expected long-term cost with respect to the policy parameters by evaluating some number of sample trajectories, then performing (stochastic) gradient descent. Many of these so-called "policy gradient" algorithms leverage a derivation called the *likelihood ratio method* that was perhaps first described in [2] then popularized in the the REINFORCE algorithm [3]. It is based on what looks like a trick with logarithms to estimate the gradient; I feel like this trick is often presented with an air of mystery about it. Let's try to make sure we understand it.

### 20.1.1 The Likelihood Ratio Method (aka REINFORCE)

Let's start with a simpler optimization over a stochastic function:

$$\min_\alpha E\left[\ell(\mathbf{x})\right] \text{ with } \mathbf{x} \sim p_\alpha(\mathbf{x})$$

I hope the notation is clear? $\mathbf{x}$ is a random vector, drawn from the distribution $p_\alpha(\mathbf{x})$, with the subscript indicating that the distribution depends on the parameter vector $\alpha$. What is the gradient of this function? The REINFORCE derivation is

$$\begin{aligned}\frac{\partial}{\partial\alpha}E\left[g(\mathbf{x})\right] =& \frac{\partial}{\partial\alpha}\int d\mathbf{x}\; g(\mathbf{x})p_\alpha(\mathbf{x}) \\ =& \int d\mathbf{x}\; g(\mathbf{x})\frac{\partial}{\partial\alpha}p_\alpha(\mathbf{x}) \\ =& \int d\mathbf{x}\; g(\mathbf{x})p_\alpha(\mathbf{x})\frac{\partial}{\partial\alpha}\log p_\alpha(\mathbf{x}) \\ =& E\left[g(\mathbf{x})\frac{\partial}{\partial\alpha}\log p_\alpha(\mathbf{x})\right].\end{aligned}$$

To achieve this, we used the derivative of the logarithm:

$$\begin{aligned}y &= \log u \\ \frac{\partial y}{\partial u} &= \frac{1}{u}\frac{\partial u}{\partial x}\end{aligned}$$

to write

$$\frac{\partial}{\partial\alpha}p_\alpha(\mathbf{x}) = p_\alpha(\mathbf{x})\frac{\partial}{\partial\alpha}\log p_\alpha(\mathbf{x}).$$

This suggests a simple Monte Carlo algorithm for estimating the policy gradient: draw $N$ random samples $\mathbf{x}_i$ then estimate the gradient as

$$\frac{\partial}{\partial\alpha}E\left[g(\mathbf{x})\right] \approx \frac{1}{N}\sum_i g(\mathbf{x}_i)\frac{\partial}{\partial\alpha}\log p_\alpha(\mathbf{x}_i).$$

This trick is even more potent in the optimal control case. For a finite-horizon problem we have

$$\begin{aligned}\frac{\partial}{\partial\alpha}E\left[\sum_{n=0}^N \ell(\mathbf{x}[n],\mathbf{u}[n])\right] =& \int d\mathbf{x}[\cdot]d\mathbf{u}[\cdot]\left[\sum_{n=0}^N \ell(\mathbf{x}[n],\mathbf{u}[n])\right]\frac{\partial}{\partial\alpha}p_\alpha(\mathbf{x}[\cdot],\mathbf{u}[\cdot]) \\ =& E\left[\left(\sum_{n=0}^N \ell(\mathbf{x}[n],\mathbf{u}[n])\right)\frac{\partial}{\partial\alpha}\log p_\alpha(\mathbf{x}[\cdot],\mathbf{u}[\cdot])\right],\end{aligned}$$

where $\mathbf{x}[\cdot]$ is shorthand for the entire trajectory $\mathbf{x}[0], \ldots, \mathbf{x}[n]$, and

$$p_\alpha(\mathbf{x}[\cdot], \mathbf{u}[\cdot]) = p_0(\mathbf{x}[0]) \left( \prod_{n=1}^{N} p(\mathbf{x}[n] | \mathbf{x}[n-1], \mathbf{u}[n-1]) \right) \left( \prod_{n=0}^{N} p_\alpha(\mathbf{u}[n] | \mathbf{x}[n]) \right).$$

Taking the log we have

$$\log p_\alpha(\mathbf{x}[\cdot], \mathbf{u}[\cdot]) = \log p_0(\mathbf{x}[0]) + \sum_{n=1}^{N} \log p(\mathbf{x}[n] | \mathbf{x}[n-1], \mathbf{u}[n-1]) + \sum_{n=0}^{n} \log p_\alpha(\mathbf{u}[n] | \mathbf{x}[n]).$$

Only the last terms depend on $\alpha$, which yields

$$\begin{aligned} \frac{\partial}{\partial \alpha} E\left[ \sum_{n=0}^{N} \ell(\mathbf{x}[n], \mathbf{u}[n]) \right] &= E\left[ \left( \sum_{n=0}^{N} \ell(\mathbf{x}[n], \mathbf{u}[n]) \right) \left( \sum_{n=0}^{N} \frac{\partial}{\partial \alpha} \log p_\alpha(\mathbf{u}[n] | \mathbf{x}[n]) \right) \right] \\ &= E\left[ \sum_{n=0}^{N} \left( \ell(\mathbf{x}[n], \mathbf{u}[n]) \sum_{k=0}^{n} \frac{\partial}{\partial \alpha} \log p_\alpha(\mathbf{u}[n] | \mathbf{x}[n]) \right) \right], \end{aligned}$$

where the last step uses the fact that $E\left[ \ell(\mathbf{x}[n], \mathbf{u}[n]) \frac{\partial}{\partial \alpha} \log p_\alpha(\mathbf{u}[k] | \mathbf{x}[k]) \right] = 0$ for all $k > n$.

This update should surprise you. It says that I can find the gradient of the long-term cost by taking only the gradient of the policy... but not the gradient of the plant, nor the cost! The intuition is that one can obtain the gradient by evaluating the policy along a number of (random) trajectory roll-outs of the closed-loop system, evaluating the cost on each, then increasing the probability in the policy of taking the actions correlated with lower long-term costs.

This derivation is often presented as **the** policy gradient derivation. The identity is certainly correct, but I'd prefer that you think of this as just one way to obtain the policy gradient. It's a particularly clever one in that it uses exactly the information that we have available in reinforcement learning -- we have access to the instantaneous costs, $\ell(\mathbf{x}[n], \mathbf{u}[n])$, and to the policy -- so are allowed to take gradients of the policy -- but does not require any understanding whatsoever of the plant model. Although it's clever, it is not particularly efficient -- the Monte Carlo approximations of the expected value have a high variance, so many samples are required for an accurate estimate. Other derivations are possible, some even simpler, and others that **do** make use of the plant gradients if you have access to them, and these will perform differently in the finite-sample approximations.

For the remainder of this section, I'd like to dig in and try to understand the nature of the stochastic update a little better, to help you understand what I mean.

## 20.1.2 Sample efficiency

Let's take a step back and think more generally about how one can use gradient descent in black-box (unconstrained) optimization. Imagine you have a simple optimization problem:

$$\min_\alpha g(\alpha),$$

and you can directly evaluate $g()$, but not $\frac{\partial g}{\partial \alpha}$. How can you perform gradient descent?

A standard technique for estimating the gradients, in lieu of analytical gradient information, is the method of finite differences[4]. The finite differences approach to estimating the gradient involves making the same small perturbation, $\epsilon$ to the input parameters in every dimension independently, and using:

$$\frac{\partial g}{\partial \alpha_i} \approx \frac{g(\alpha + \epsilon_i) - g(\alpha)}{\epsilon},$$

where $\epsilon_i$ is the column vector with $\epsilon$ in the $i$th row, and zeros everywhere else. Finite difference methods can be computationally very expensive, requiring $n+1$ evaluations of the function for every gradient step, where $n$ is the length of the input vector.

What if each function evaluation is expensive? Perhaps it means picking up a physical robot and running it for 10 seconds for each evaluation of $g()$. Suddenly there is a premium on trying to optimize the cost function with the fewest number of evaluations on $g()$. This is the name of the game in reinforcement learning -- it's often called RL's *sample complexity*. Can we perform gradient descent using less evaluations?

## 20.1.3 Stochastic Gradient Descent

This leads us to the question of doing approximate gradient descent, or "stochastic" gradient descent. Thinking of the cost landscape as a Lyapunov function†, then any update that moves downhill on each step will eventually reach the optimal. More generally, any update that moves downhill on average will eventually reach a minimum... and sometimes "stochastic" gradient descent updates that occassionally go uphill but go downhill on average can even have desirable properties like bouncing out of small local minima. The figure below gives some graphical intuition; for a formal treatment of stochastic gradient methods, see e.g. [5].

† the Lyapunov functi $V(\alpha) = g(\alpha) - g(\alpha^*)$ is commonly used in convergence/stability analysis of optimizati algorithms.

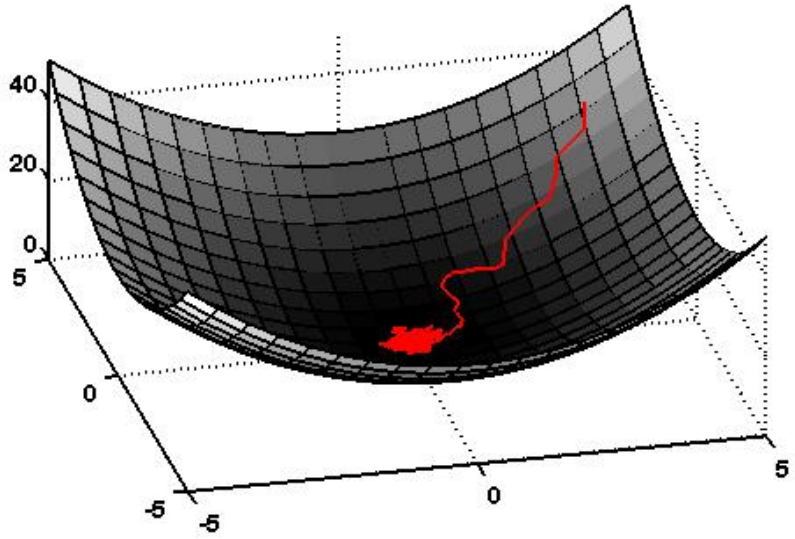



Figure 20.1 - Stochastic gradient descent of a quadratic cost function .

## 20.1.4 The Weight Pertubation Algorithm

Instead of sampling each dimension independently, consider making a single small random change, $\beta$, to the parameter vector, $\alpha$. Now consider the update of the form:

$$\Delta\alpha = -\eta[g(\alpha+\beta) - g(\alpha)]\beta.$$

The intuition here is that if $g(\alpha+\beta) < g(\alpha)$, then $\beta$ was a good change and we'll move in the direction of $\beta$. If the cost increased, then we'll move in the opposite direction. Assuming the function is smooth and $\beta$ is small, then we will always move downhill on the Lyapunov function.

Even more interesting, on average, the update is actually in the direction of the true gradient. To see this, again we can assume the function is locally smooth and $\beta$ is small to write the Taylor expansion:

$$g(\alpha+\beta) \approx g(\alpha) + \frac{\partial g}{\partial \alpha}\beta.$$

Now we have

$$\Delta\alpha \approx -\eta\left[\frac{\partial g}{\partial \alpha}\beta\right]\beta = -\eta\beta\beta^T\frac{\partial g}{\partial \alpha}^T$$
$$E\left[\Delta\alpha\right] \approx -\eta E\left[\beta\beta^T\right]\frac{\partial g}{\partial \alpha}^T$$

If we select each element of $\beta$ independently from a distribution with zero mean and variance $\sigma_\beta^2$, or $E\left[\beta_i\right] = 0, E\left[\beta_i\beta_j\right] = \sigma_\beta^2\delta_{ij}$, then we have

$$E\left[\Delta\alpha\right] \approx -\eta\sigma_\beta^2\frac{\partial g}{\partial \alpha}^T.$$

Note that the distribution $p_\alpha(\mathbf{x})$ need not be Gaussian, but it is the variance of the distribution which determines the scaling on the gradient.

### 20.1.5 Weight Perturbation with an Estimated Baseline

The weight perturbation update above requires us to evaluate the function $g$ twice for every update of the parameters. This is low compared to the method of finite differences, but let us see if we can do better. What if, instead of evaluating the function twice per update [$g(\alpha+\beta)$ and $g(\alpha)$], we replace the evaluation of $g(\alpha)$ with an estimator, $b = \hat{g}(\alpha)$, obtained from previous trials? In other words, consider an update of the form:

$$\Delta\alpha = -\frac{\eta}{\sigma_\beta^2}[g(\alpha+\beta) - b]\beta \tag{2}$$

The estimator can take any of a number of forms, but perhaps the simplest is based on the update (after the $n$th trial):

$$b[n+1] = \gamma g[n] + (1-\gamma)b[n], \quad b[0] = 0, 0 \leq \gamma \leq 1,$$

where $\gamma$ parameterizes the moving average. Let's compute the expected value of the update, using the same Taylor expansion that we used above:

$$\begin{aligned}
E[\Delta\alpha] &= -\frac{\eta}{\sigma_\beta^2}E\left[[g(\alpha) + \frac{\partial g}{\partial \alpha}\beta - b]\beta\right] \\
&= -\frac{\eta}{\sigma_\beta^2}E\left[[g(\alpha) - b]\beta\right] - \frac{\eta}{\sigma_\beta^2}E\left[\beta\beta^T\right]\frac{\partial g}{\partial \alpha}^T \\
&= -\eta\frac{\partial g}{\partial \alpha}^T
\end{aligned}$$

In other words, the baseline does not effect our basic result - that the expected update is in the direction of the gradient. Note that this computation works for any baseline estimator which is uncorrelated with $\beta$, as it should be if the estimator is a function of the performance in previous trials.

Although using an estimated baseline doesn't effect the average update, it can have a dramatic effect on the performance of the algorithm. As we will see in a later section, if the evaluation of $g$ is stochastic, then the update with a baseline estimator can actually outperform an update using straight function evaluations.

Let's take the extreme case of $b = 0$. This seems like a very bad idea... on each step we will move in the direction of every random perturbation, but we will move more or less in that direction based on the evaluated cost. In average, we will still move in the direction of the true gradient, but only because we will eventually move more downhill than we move uphill. It feels very naive.

### 20.1.6 REINFORCE w/ additive Gaussian noise

Now let's consider the simple form of the REINFORCE update:

$$\frac{\partial}{\partial \alpha} E\left[g(\mathbf{x})\right] = E\left[g(\mathbf{x}) \frac{\partial}{\partial \alpha} \log p_\alpha(\mathbf{x})\right].$$

It turns out that weight perturbation is a REINFORCE algorithm. To see this, take $\mathbf{x} = \alpha + \beta$, $\beta \in N(0, \sigma^2)$, to have

$$p_\alpha(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^N} e^{\frac{-(\mathbf{x}-\alpha)^T(\mathbf{x}-\alpha)}{2\sigma^2}}$$

$$\log p_\alpha(\mathbf{x}) = \frac{-(\mathbf{x}-\alpha)^T(\mathbf{x}-\alpha)}{2\sigma^2} + \text{terms that don't} \quad \text{depend on } \alpha$$

$$\frac{\partial}{\partial \alpha} \log p_\alpha(\mathbf{x}) = \frac{1}{\sigma^2}(\alpha - \mathbf{x})^T = \frac{1}{\sigma^2}\beta^T.$$

If we use only a single trial for each Monte-Carlo evaluation, then the REINFORCE update is

$$\Delta\alpha = -\frac{\eta}{\sigma^2} g(\alpha + \beta)\beta,$$

which is precisely the weight perturbation update (the crazy version with $b = 0$ discussed above). Although it will move in the direction of the gradient on average, it can be highly inefficient. In practice, people use many more than one sample to estimate the poilicy gradient.

### 20.1.7 Summary

The policy gradient "trick" from REINFORCE using log probabilities provides one means of estimating the true policy gradient. It is not the only way to obtain the policy gradient... in fact the trivial weight-perturbation update obtains the same gradient for the case of a policy with a mean linear in $\alpha$ and a fixed diagonal covariance. It's cleverness lies in the fact that it makes use of the information we do have (instantaneous cost values and the gradients of the policy), and that it provides an unbiased estimate of the gradient (note that taking the gradients of a model that is only slightly wrong might not have this virtue). But its inefficiency comes from having a potentially very high variance. Variance reduction for policy gradient, often through baseline estimation, continues to be an active area of research.

## 20.2 SAMPLE PERFORMANCE VIA THE SIGNAL-TO-NOISE RATIO.

### 20.2.1 Performance of Weight Perturbation

The simplicity of the REINFORCE / weight perturbation updates makes it tempting to apply them to problems of arbitrary complexity. But a major concern for the algorithm is its performance - although we have shown that the update is in the direction of the true gradient *on average*, it may still require a prohibitive number of computations to obtain a local minima.

In this section, we will investigate the performance of the weight perturbation algorithm by investigating its *signal-to-noise* ratio (SNR). This idea was explored in [1] -- my goal is just to give you a taste here. The SNR is the ratio of the power in the signal (here desired update in the direction of the true gradient) and the power in the noise (the remaining component of the update, so that $\Delta\alpha = -\eta \frac{\partial g}{\partial \alpha}^T + \text{noise}$)†

† SNR could alternatively be defin as the ratio of the

$$\text{SNR} = \frac{-\eta \frac{\partial g}{\partial \alpha}^T{}^2}{E\left[\Delta\alpha + \eta \frac{\partial g}{\partial \alpha}^T{}^2\right]}.$$

In the special case of the unbiased update, the equation reduces to:

$$\text{SNR} = \frac{E[\Delta\alpha]^T E[\Delta\alpha]}{E\left[(\Delta\alpha)^T(\Delta\alpha)\right] - E[\Delta\alpha]^T E[\Delta\alpha]}.$$

For the weight perturbation update we have:

$$E[\Delta\alpha]^T E[\Delta\alpha] = \eta^2 \frac{\partial g}{\partial \alpha}\frac{\partial g}{\partial \alpha}^T = \eta^2 \sum_{i=1}^{N}\left(\frac{\partial g}{\partial \alpha_i}\right)^2,$$

$$\begin{aligned}
E\left[(\Delta\alpha)^T(\Delta\alpha)\right] =& \frac{\eta^2}{\sigma_\beta^4} E\left[[g(\alpha+\beta) - g(\alpha)]^2 \beta^T\beta\right] \\
\approx& \frac{\eta^2}{\sigma_\beta^4} E\left[\left[\frac{\partial g}{\partial \alpha}\beta\right]^2 \beta^T\beta\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} E\left[\left(\sum_i \frac{\partial g}{\partial \alpha_i}\beta_i\right)\left(\sum_j \frac{\partial g}{\partial \alpha_j}\beta_j\right)\left(\sum_k \beta_k^2\right)\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} E\left[\sum_i \frac{\partial g}{\partial \alpha_i}\beta_i \sum_j \frac{\partial g}{\partial \alpha_j}\beta_j \sum_k \beta_k^2\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} \sum_{i,j,k} \frac{\partial g}{\partial \alpha_i}\frac{\partial g}{\partial \alpha_j} E\left[\beta_i\beta_j\beta_k^2\right]
\end{aligned}$$

$$E\left[\beta_i\beta_j\beta_k^2\right] = \begin{cases} 0 & i \neq j \\ \sigma_\beta^4 & i = j \neq k \\ \mu_4(\beta) & i = j = k \end{cases}$$

$$= \eta^2(N-1)\sum_i \left(\frac{\partial g}{\partial \alpha_i}\right)^2 + \frac{\eta^2 \mu_4(z)}{\sigma_\beta^4}\sum_i \left(\frac{\partial g}{\partial \alpha_i}\right)^2$$

where $\mu_n(z)$ is the $n$th central moment of $z$:

$$\mu_n(z) = E\left[(z - E[z])^n\right].$$

Putting it all together, we have:

$$\text{SNR} = \frac{1}{N - 2 + \frac{\mu_4(\beta_i)}{\sigma_\beta^4}}.$$

**Example 20.1 (Signal-to-noise ratio for additive Gaussian noise)**

For $\beta_i$ drawn from a Gaussian distribution, we have $\mu_1 = 0, \mu_2 = \sigma_\beta^2, \mu_3 = 0, \mu_4 = 3\sigma_\beta^4$, simplifying the above expression to:

$$\text{SNR} = \frac{1}{N+1}.$$

**Example 20.2 (Signal-to-noise ratio for additive uniform noise)**

For $\beta_i$ drawn from a uniform distribution over [-a,a], we have $\mu_1 = 0, \mu_2 = \frac{a^2}{3} = \sigma_\beta^2, \mu_3 = 0, \mu_4 = \frac{a^4}{5} = \frac{9}{5}\sigma_\beta^4$, simplifying the above to:

$$\text{SNR} = \frac{1}{N - \frac{1}{5}}.$$

Performance calculations using the SNR can be used to design the parameters of the algorithm in practice. For example, based on these results it is apparent that noise added through a uniform distribution yields better gradient estimates than the Gaussian noise case for very small $N$, but that these differences are neglibile for large $N$.

Similar calculations can yield insight into picking the size of the additive noise (scaled by $\sigma_\beta$). The calculations in this section seem to imply that larger $\sigma_\beta$ can only reduce the variance, overcoming errors or noise in the baseline estimator, $\tilde{b}$; this is a short-coming of our first-order Taylor expansion. If the cost function is not linear in the parameters, then an examination of higher-order terms reveals that large $\sigma_\beta$ can increase the SNR. The derivation with a second-order Taylor expansion is left for an exercise.

## REFERENCES


1. John W. Roberts, "Motor Learning on a Heaving Plate via Improved-{SNR} Algorithms", , February, 2009. [ link ]

2. Peter W. Glynn, "Likelihood ratio gradient estimation for stochastic systems", *Communications of the ACM*, vol. 33, no. 10, pp. 75--84, oct, 1990.

3. R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine Learning*, vol. 8, pp. 229-256, 1992.

4. William H. Press and Saul A. Teukolsky and William T. Vetterling and Brian P. Flannery, "Numerical Recipes in C: The Art of Scientific Computing",Cambridge University Press , 1992.

5. Dimitri P Bertsekas, "Nonlinear programming",Athena scientific Belmont , 1999.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Appendix A
# Drake

Drake is the main software toolbox that we use for this text, which in fact originated in a large part due to the MIT underactuated course. The Drake website is the main source for information and documentation. The goal of this chapter is to provide any additional information that can help you get up and running with the examples and exercises provided in these notes.

## A.1 Pydrake

Drake is primarily a C++ library, with rigorous coding standards and a maturity level intended to support even professional applications in industry. In order to provide a gentler introduction, and to facilitate rapid prototyping, I've written these notes exclusively in python, using Drake's python bindings (pydrake). These bindings are less mature than the C++ backend; your feedback (and even contributions) are very welcome. It is still improving rapidly.

In particular, while the C++ API documentation is excellent, the autogenerated python docs are still a work in progress. I currently recommend using the C++ documentation to find what you need, then checking the Python documentation only if you need to understand how the class or method is spelled in pydrake.

There are also a number of tutorials in Drake that can help you get started.

## A.2 Online Jupyter Notebooks

I will provide nearly all examples and exercises in the form of a Jupyter Notebook so that we can leverage the fantastic and relatively recent availability of (free) cloud resources. In particular I have chosen to focus on using Google's Colaboratory (Colab) as our primary platform. Most of the chapters have a corresponding notebook that you can run on the cloud without installing anything on your machine!

Colab is a fantastic resource, with generous computational resources and a very convenient interface that allows you to save/load/share your work on Google Drive. Although we cannot provision the machines ahead of time, we can provision them

quickly with a slightly ugly install script at the top of every notebook. The first time you open each notebook it takes about two minutes to provision the machine. At the time of this writing, it seems that the machine will stay provisioned for you for 90 minutes if you close the browser, or 12 hours if you keep it open. Provisioning is per user and appears relatively unlimited.

## A.3 Running on your own machine

As you get more advanced, you will likely want to run (and extend) these examples on your own machine. The Drake website has a number of installation options, including precompiled binaries and Docker instances. Here I provide an example workflow of setting up drake from the precompiled binaries and running the textbook examples.

First, pick your platform (click on your OS):

**Ubuntu Linux (Bionic)** | Mac Homebrew

### A.3.1 Install Drake

The links below indicate the specific distributions that the examples in this text have been tested against.

#### Download the binaries

```
curl -o drake.tar.gz https://drake-packages.csail.mit.edu/drake/nightly/drake-latest-bionic
```

#### Unpack and set your PYTHONPATH and Test

```
sudo tar -xvzf drake.tar.gz -C /opt
sudo /opt/drake/share/drake/setup/install_prereqs
export PYTHONPATH=/opt/drake/lib/python3.6/site-packages:${PYTHONPATH}

python3 -c 'import pydrake; print(pydrake.__file__)'
```

### A.3.2 Download the textbook supplement

```
git clone https://github.com/RussTedrake/underactuated.git
```

and install the prerequisites using the platform-specific installation script provided:

```
cd underactuated
sudo scripts/setup/ubuntu/18.04/install_prereqs.sh
pip3 install --requirement requirements.txt
export PYTHONPATH=`pwd`:${PYTHONPATH}
```

Note that, as always, I would strongly recommend running `pip` commands in a virtualenv.

### A.3.3 Run Jupyter Notebook

You'll likely want to start from the underactuated root directory. Then launch your notebook with:

```
jupyter notebook
```

The examples for each chapter that has them will be in a .ipynb file right alongside the chapter's html file, and the notebook exercises are all located in the `exercises` subdirectory.

## A.4 Getting help

If you have trouble with Drake, please follow the advice here. If you have trouble with the underactuated repo, you can check for known issues (and potentially file a new one) here.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# Appendix B
# Multi-Body Dynamics

## B.1 Deriving the equations of motion

The equations of motion for a standard robot can be derived using the method of Lagrange. Using $T$ as the total kinetic energy of the system, and $U$ as the total potential energy of the system, $L = T - U$, and $Q_i$ as the generalized force corresponding to $q_i$, the Lagrangian dynamic equations are:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i. \tag{1}$$

You can think of them as a generalization of Newton's equations. For a particle, we have $T = \frac{1}{2}m\dot{x}^2$, so $\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} = m\ddot{x}$, and $\frac{\partial L}{\partial x} = -\frac{\partial U}{\partial x} = f$ amounting to $f = ma$. But the Lagrangian derivation works in generalized coordinate systems and for constrained motion.

Without going into the full details, the key idea for handling constraints is the "principle of virtual work" (and the associated D'Almbert's principle). To describe even a pendulum at equilibrium in the Newtonian approach, we would have to compute both external forces (like gravity) and internal forces (like the forces keeping the pendulum arm attached to the table), and have their sum equal zero. Computing internal forces is slightly annoying for a pendulum; it becomes untenable for more complex mechanisms. If the sum of the forces equals zero, then the work done by those forces in some (infinitesmal) virtual displacement is certainly also equal to zero. Now here is the trick: if we consider only virtual displacements that are consistent with the kinematic constraints (e.g. rotations around the pin joint of the pendulum), then we can compute that virtual work and set it equal to zero without ever computing the internal forces! Extending this idea to the dynamics case (via D'Almbert's and Hamilton's principles) results eventually in the Lagrangian equations above.

If you are not comfortable with these equations, then any good book chapter on rigid body mechanics can bring you up to speed. [1] is an excellent practical guide to robot kinematics/dynamics. But [2] is my favorite mechanics book, by far; I highly recommend it if you want something more. It's also ok to continue on for now thinking of Lagrangian mechanics simply as a recipe than you can apply in a great

many situations to generate the equations of motion.

For completeness, I've included a derivation of the Lagrangian from the principle of stationary action below.

### Example B.1 (Simple Double Pendulum)

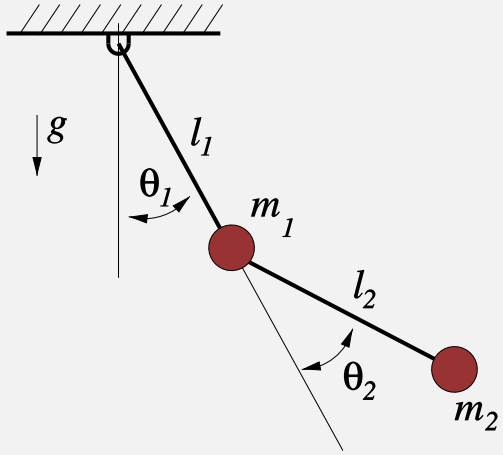



Figure B.1 - Simple double pendulum

Consider the simple double pendulum with torque actuation at both joints and all of the mass concentrated in two points (for simplicity). Using $\mathbf{q} = [\theta_1, \theta_2]^T$, and $\mathbf{p}_1, \mathbf{p}_2$ to denote the locations of $m_1, m_2$, respectively, the kinematics of this system are:

$$\begin{aligned}\mathbf{p}_1 =& l_1 \begin{bmatrix} s_1 \\ -c_1 \end{bmatrix}, \mathbf{p}_2 = \mathbf{p}_1 + l_2 \begin{bmatrix} s_{1+2} \\ -c_{1+2} \end{bmatrix} \\ \dot{\mathbf{p}}_1 =& l_1 \dot{q}_1 \begin{bmatrix} c_1 \\ s_1 \end{bmatrix}, \dot{\mathbf{p}}_2 = \dot{\mathbf{p}}_1 + l_2 (\dot{q}_1 + \dot{q}_2) \begin{bmatrix} c_{1+2} \\ s_{1+2} \end{bmatrix}\end{aligned}$$

Note that $s_1$ is shorthand for $\sin(q_1)$, $c_{1+2}$ is shorthand for $\cos(q_1 + q_2)$, etc. From this we can write the kinetic and potential energy:

$$\begin{aligned}T =& \frac{1}{2} m_1 \dot{\mathbf{p}}_1^T \dot{\mathbf{p}}_1 + \frac{1}{2} m_2 \dot{\mathbf{p}}_2^T \dot{\mathbf{p}}_2 \\ =& \frac{1}{2}(m_1 + m_2) l_1^2 \dot{q}_1^2 + \frac{1}{2} m_2 l_2^2 (\dot{q}_1 + \dot{q}_2)^2 + m_2 l_1 l_2 \dot{q}_1 (\dot{q}_1 + \dot{q}_2) c_2 \\ U =& m_1 g y_1 + m_2 g y_2 = -(m_1 + m_2) g l_1 c_1 - m_2 g l_2 c_{1+2}\end{aligned}$$

Taking the partial derivatives $\frac{\partial T}{\partial q_i}$, $\frac{\partial T}{\partial \dot{q}_i}$, and $\frac{\partial U}{\partial q_i}$ ($\frac{\partial U}{\partial \dot{q}_i}$ terms are always zero), then $\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i}$, and plugging them into the Lagrangian, reveals the equations of motion:

$$\begin{aligned}(m_1 + m_2) l_1^2 \ddot{q}_1 + m_2 l_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 l_1 l_2 (2\ddot{q}_1 + \ddot{q}_2) c_2 \\ - m_2 l_1 l_2 (2\dot{q}_1 + \dot{q}_2) \dot{q}_2 s_2 + (m_1 + m_2) l_1 g s_1 + m_2 g l_2 s_{1+2} = \tau_1 \\ m_2 l_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 l_1 l_2 \ddot{q}_1 c_2 + m_2 l_1 l_2 \dot{q}_1^2 s_2 + m_2 g l_2 s_{1+2} = \tau_2\end{aligned}$$

As we saw in chapter 1, numerically integrating (and animating) these equations in DRAKE produces the expected result.

## B.2 The Manipulator Equations

If you crank through the Lagrangian dynamics for a few simple robotic manipulators, you will begin to see a pattern emerge - the resulting equations of motion all have a characteristic form. For example, the kinetic energy of your robot can always be

written in the form:

$$T = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}, \tag{2}$$

where $\mathbf{M}$ is the state-dependent inertia matrix (aka mass matrix). This observation affords some insight into general manipulator dynamics - for example we know that $\mathbf{M}$ is always positive definite, and symmetric[3, p.107] and has a beautiful sparsity pattern[4] that we should take advantage of in our algorithms.

Continuing our abstractions, we find that the equations of motion of a general robotic manipulator (without kinematic loops) take the form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}, \tag{3}$$

where $\mathbf{q}$ is the joint position vector, $\mathbf{M}$ is the inertia matrix, $\mathbf{C}$ captures Coriolis forces, and $\tau_g$ is the gravity vector. The matrix $\mathbf{B}$ maps control inputs $\mathbf{u}$ into generalized forces. Note that we pair $\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}}$ on the left side because "... the equations of motion depend on the choice of coordinates $\mathbf{q}$. For this reason neither $\mathbf{M}\ddot{\mathbf{q}}$ nor $\mathbf{C}\dot{\mathbf{q}}$ individually should be thought of as a generalized force; only their sum is a force"[5](s.10.2). These terms represent the contribution from kinetic energy to the Lagrangian:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\mathbf{q}}}^T - \frac{\partial T}{\partial \mathbf{q}}^T = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}.$$

Whenever I write Eq. (3), I see $ma = f$.

### Example B.2 (Manipulator Equation form of the Simple Double Pendulum)

The equations of motion from Example 1 can be written compactly as:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} (m_1+m_2)l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 c_2 & m_2 l_2^2 + m_2 l_1 l_2 c_2 \\ m_2 l_2^2 + m_2 l_1 l_2 c_2 & m_2 l_2^2 \end{bmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} 0 & -m_2 l_1 l_2 (2\dot{q}_1 + \dot{q}_2) s_2 \\ m_2 l_1 l_2 \dot{q}_1 s_2 & 0 \end{bmatrix}$$

$$\tau_g(\mathbf{q}) = -g \begin{bmatrix} (m_1+m_2) l_1 s_1 + m_2 l_2 s_{1+2} \\ m_2 l_2 s_{1+2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Note that this choice of the $\mathbf{C}$ matrix was not unique.

The manipulator equations are very general, but they do define some important characteristics. For example, $\ddot{\mathbf{q}}$ is (state-dependent) linearly related to the control input, $\mathbf{u}$. This observation justifies the control-affine form of the dynamics assumed throughout the notes. There are many other important properties that might prove useful. For instance, we know that the $i$ element of $\mathbf{C}\dot{q}$ is given by [3, §5.2.3]:

$$[\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}]_i = \sum_{j,k} c_{ijk}\dot{q}_j \dot{q}_k, \qquad c_{ijk}(\mathbf{q}) = \frac{\partial M_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial M_{jk}}{\partial q_k},$$

(e.g. it also quadratic in $\dot{\mathbf{q}}$) and, for the appropriate choice of $\mathbf{C}$, we have that $\mathbf{M}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is skew-symmetric[6].

Note that we have chosen to use the notation of second-order systems (with $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ appearing in the equations) throughout this book. Although I believe it provides more clarity, there is an important limitation to this notation: it is impossible to describe 3D rotations in "minimal coordinates" using this notation without introducing kinematic singularities (like the famous "gimbal lock"). For instance, a common singularity-free choice for representing a 3D rotation is the unit quaternion, described by 4 real values (plus a norm constraint). However we can still represent the rotational velocity without singularities using just 3 real values. This means that

the length of our velocity vector is no longer the same as the length of our position vector. For this reason, you will see that most of the software in DRAKE uses the more general notation with $\mathbf{v}$ to represent velocity, $\mathbf{q}$ to represent positions, and the manipulator equations are written as

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}, \tag{4}$$

which is not necessarily a second-order system. See [7] for a nice discussion of this topic.

## B.2.1 Recursive Dynamics Algorithms

The equations of motions for our machines get complicated quickly. Fortunately, for robots with a tree-link kinematic structure, there are very efficient and natural recursive algorithms for generating these equations of motion. For a detailed reference on these methods, see [8]; some people prefer reading about the Articulated Body Method in [9]. The implementation in DRAKE uses a related formulation from [10].

## B.2.2 Hamiltonian Mechanics

In some cases, it might be preferable to use the Hamiltonian formulation of the dynamics, with state variables $\mathbf{q}$ and $\mathbf{p}$ (instead of $\dot{\mathbf{q}}$), where $\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}^T = \mathbf{M}\dot{\mathbf{q}}$ results in the dynamics:

$$\begin{aligned}\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} =& \mathbf{p} \\ \dot{\mathbf{p}} =& \mathbf{c}_H(\mathbf{q}, \mathbf{p}) + \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}, \qquad \text{where} \quad \mathbf{c}_H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\left(\frac{\partial \left[\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}\right]}{\partial \mathbf{q}}\right)^T.\end{aligned}$$

Recall that the Hamiltonian is $H = \mathbf{p}^T\dot{\mathbf{q}} - L$; these equations of motion are the familiar $\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}}^T, \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}}^T$.

## B.2.3 Bilateral Position Constraints

If our robot has closed-kinematic chains, for instance those that arise from a four-bar linkage, then we need a little more. The Lagrangian machinery above assumes "minimal coordinates"; if our state vector $\mathbf{q}$ contains all of the links in the kinematic chain, then we do not have a minimal parameterization -- each kinematic loop adds (at least) one constraint so should remove (at least) one degree of freedom. Although some constraints can be solved away, the more general solution is to use the Lagrangian to derive the dynamics of the unconstrained system (a kinematic tree without the closed-loop constraints), then add additional generalized forces that ensure that the constraint is always satisfied.

Consider the constraint equation

$$\mathbf{h}(\mathbf{q}) = 0. \tag{5}$$

For the case of the kinematic closed-chain, this can be the kinematic constraint that the location of one end of the chain is equal to the location of the other end of the chain. The equations of motion can be written

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u} + \mathbf{H}^T(\mathbf{q})\lambda, \tag{6}$$

where $\mathbf{H}(\mathbf{q}) = \frac{\partial \mathbf{h}}{\partial \mathbf{q}}$ and $\lambda$ is the constraint force. Let's use the shorthand

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \tau(q, \dot{q}, u) + \mathbf{H}^T(\mathbf{q})\lambda. \tag{7}$$

Using

$$\dot{\mathbf{h}} = \mathbf{H}\dot{\mathbf{q}}, \tag{8}$$

$$\ddot{\mathbf{h}} = \mathbf{H}\ddot{\mathbf{q}} + \dot{\mathbf{H}}\dot{\mathbf{q}}, \tag{9}$$

we can solve for $\lambda$, by observing that when the constraint is imposed, $\mathbf{h} = 0$ and therefore $\dot{\mathbf{h}} = 0$ and $\ddot{\mathbf{h}} = 0$. Inserting the dynamics (7) into (9) yields

$$\lambda = -(\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T)^+(\mathbf{H}\mathbf{M}^{-1}\tau + \dot{\mathbf{H}}\dot{\mathbf{q}}). \tag{10}$$

The $^+$ notation refers to a Moore-Penrose pseudo-inverse. In many cases, this matrix will be full rank (for instance, in the case of multiple independent four-bar linkages) and the traditional inverse could have been used. When the matrix drops rank (multiple solutions), then the pseudo-inverse will select the solution with the smallest constraint forces in the least-squares sense.

To combat numerical "constraint drift", one might like to add a restoring force in the event that the constraint is not satisfied to numerical precision. To accomplish this, rather than solving for $\ddot{\mathbf{h}} = 0$ as above, we can instead solve for

$$\ddot{\mathbf{h}} = \mathbf{H}\ddot{\mathbf{q}} + \dot{\mathbf{H}}\dot{\mathbf{q}} = -2\alpha\dot{\mathbf{h}} - \alpha^2\mathbf{h}, \tag{11}$$

where $\alpha > 0$ is a stiffness parameter. This is known as Baumgarte's stabilization technique, implemented here with a single parameter to provide a critically-damped response. Carrying this through yields

$$\lambda = -(\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T)^+(\mathbf{H}\mathbf{M}^{-1}\tau + (\dot{\mathbf{H}} + 2\alpha\mathbf{H})\dot{\mathbf{q}} + \alpha^2\mathbf{h}). \tag{12}$$

There is an important optimization-based derivation/interpretation of these equations, which we will build on below, using Gauss's Principle of Least Constraint. This principle says that we can derive the constrained dynamics as :

$$\begin{aligned} \min_{\ddot{\mathbf{q}}} \quad & \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}_{uc})^T\mathbf{M}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}_{uc}), \\ \text{subject to} \quad & \ddot{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = 0 = \mathbf{H}\ddot{\mathbf{q}} + \dot{\mathbf{H}}\dot{\mathbf{q}}, \end{aligned} \tag{13}$$

where $\ddot{\mathbf{q}}_{uc} = \mathbf{M}^{-1}\tau$ is the "unconstrained acceleration" [11]. Equation (6) comes right out of the optimality conditions with $\lambda$ acting precisely as the Lagrange multiplier. This is a convex quadratic program with equality constraints, which has a closed-form solution that is precisely Equation (10) above. It is also illustrative to look at the dual formulation of this optimization, which can be written as

$$\min_{\lambda} \frac{1}{2}\lambda^T\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T\lambda - \lambda^T(\mathbf{H}\mathbf{M}^{-1}\tau + \dot{\mathbf{H}}\dot{\mathbf{q}}).$$

Observe that the linear term is contains the acceleration of $\mathbf{h}$ if the dynamics evolved without the constraint forces applied; let's call that $\ddot{\mathbf{h}}_{uc}$:

$$\min_{\lambda} \frac{1}{2}\lambda^T\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T\lambda - \lambda^T\ddot{\mathbf{h}}_{uc}.$$

The primal formulation has accelerations as the decision variables, and the dual formulation has constraint forces as the decision variables. For now this is merely a cute observation; we'll build on it more when we get to discussing contact forces.

## B.2.4 Bilateral Velocity Constraints

Consider the constraint equation

$$\mathbf{h}_v(\mathbf{q}, \dot{\mathbf{q}}) = 0, \tag{14}$$

where $\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}} \neq 0$. These are less common, but arise when, for instance, a joint is driven through a prescribed motion. Here, the manipulator equations are given by

$$\mathbf{M}\ddot{\mathbf{q}} = \tau + \frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}^T \lambda. \tag{15}$$

Using

$$\dot{\mathbf{h}}_v = \frac{\partial \mathbf{h}_v}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\ddot{\mathbf{q}}, \tag{16}$$

setting $\dot{\mathbf{h}}_v = 0$ yields

$$\lambda = -\left(\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\mathbf{M}^{-1}\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\right)^+ \left[\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\mathbf{M}^{-1}\tau + \frac{\partial \mathbf{h}_v}{\partial \mathbf{q}}\dot{\mathbf{q}}\right]. \tag{17}$$

Again, to combat constraint drift, we can ask instead for $\dot{\mathbf{h}}_v = -\alpha \mathbf{h}_v$, which yields

$$\lambda = -\left(\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\mathbf{M}^{-1}\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\right)^+ \left[\frac{\partial \mathbf{h}_v}{\partial \dot{\mathbf{q}}}\mathbf{M}^{-1}\tau + \frac{\partial \mathbf{h}_v}{\partial \mathbf{q}}\dot{\mathbf{q}} + \alpha \mathbf{h}_v\right]. \tag{18}$$

## B.3 The Dynamics of Contact

The dynamics of multibody systems that make and break contact are closely related to the dynamics of constrained systems, but tend to be much more complex. In the simplest form, you can think of non-penetration as an *inequality* constraint: the signed distance between collision bodies must be non-negative. But, as we have seen in the chapters on walking, the transitions when these constraints become active correspond to collisions, and for systems with momentum they require some care. We'll also see that frictional contact adds its own challenges.

There are three main approaches used to modeling contact with "rigid" body systems: 1) rigid contact approximated with stiff compliant contact, 2) hybrid models with collision event detection, impulsive reset maps, and continuous (constrained) dynamics between collision events, and 3) rigid contact approximated with time-averaged forces (impulses) in a time-stepping scheme. Each modeling approach has advantages/disadvantages for different applications.

Before we begin, there is a bit of notation that we will use throughout. Let $\phi(\mathbf{q})$ indicate the relative (signed) distance between two rigid bodies. For rigid contact, we would like to enforce the unilateral constraint:

$$\phi(\mathbf{q}) \geq 0. \tag{19}$$

We'll use $\mathbf{n} = \frac{\partial \phi}{\partial \mathbf{q}}$ to denote the "contact normal", as well as $\mathbf{t}_1$ and $\mathbf{t}_2$ as a basis for the tangents at the contact (orthonormal vectors in the Cartesian space, projected into joint space), all represented as row vectors in joint coordinates.

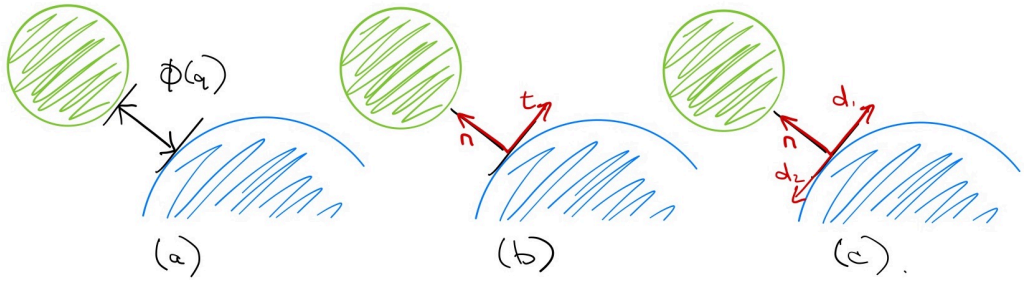



Figure B.2 - Contact coordinates in 2D. (a) The signed distance between contact bodies, $\phi(\mathbf{q})$. (b) The normal ($\mathbf{n}$) and tangent ($\mathbf{t}$) contact vectors -- note that these can be drawn in 2D when the $\mathbf{q}$ is the $x, y$ positions of one of the bodies, but more generally the vectors live in the configuration space. (c) Sometimes it will be helpful for us to express the tangential coordinates using $d_1$ and $d_2$; this will make more sense in the 3D case.

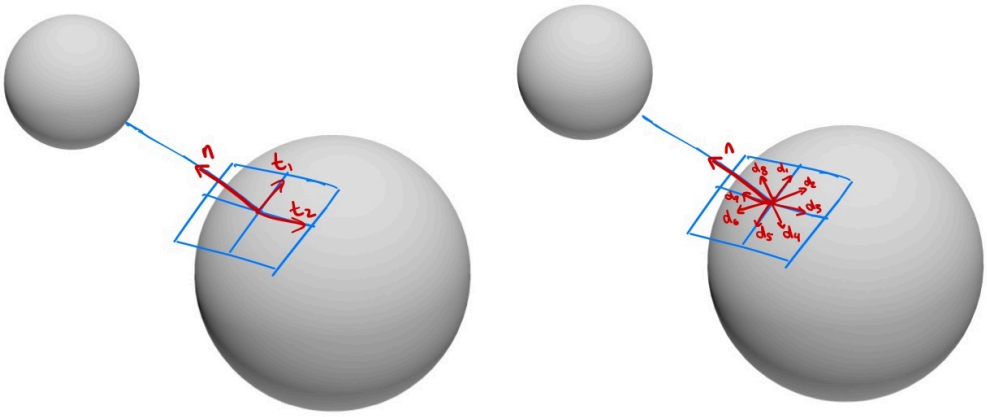



Figure B.3 - Contact coordinates in 3D.

We will also find it useful to assemble the contact normal and tangent vectors into a single matrix, $\mathbf{J}$, that we'll call the *contact Jacobian*:

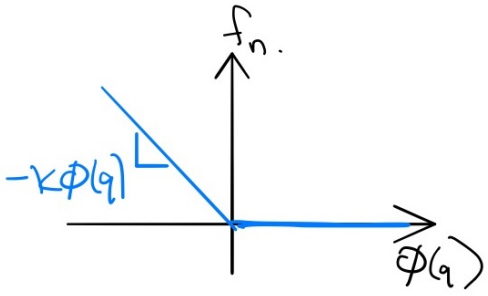
$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{n} \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix}.$$

As written, $\mathbf{Jv}$ gives the relative velocity of the closest points in the contact coordinates; it can be also be extended with three more rows to output a full spatial velocity (e.g. when modeling torsional friction). The generalized force produced by the contact is given by $\mathbf{J}^T\lambda$, where $\lambda = [f_n, f_{t1}, f_{t2}]^T$ :

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u} + \mathbf{J}^T(\mathbf{q})\lambda. \tag{20}$$

## B.3.1 Compliant Contact Models

Most compliant contact models are conceptually straight-forward: we will implement contact forces using a stiff spring (and damper[12]) that produces forces to resist penetration (and grossly model the dissipation of collision and/or friction). For instance, for the normal force, $f_n$, we can use a simple (piecewise) linear spring law:




Coulomb friction is described by two parameters -- $\mu_{static}$ and $\mu_{dynamic}$ -- which are the coefficients of static and dynamic friction. When the contact tangential velocity (given by $\mathbf{tv}$) is zero, then friction will produce whatever force is necessary to resist motion, up to the threshold $\mu_{static}f_n$. But once this threshold is exceeding, we have slip (non-zero contact tangential velocity); during slip friction will produce a constant drag force $|f_t| = \mu_{dynamic}f_n$ in the direction opposing the motion. This behaviour is not a simple function of the current state, but we can approximate it with a continuous function as pictured below.

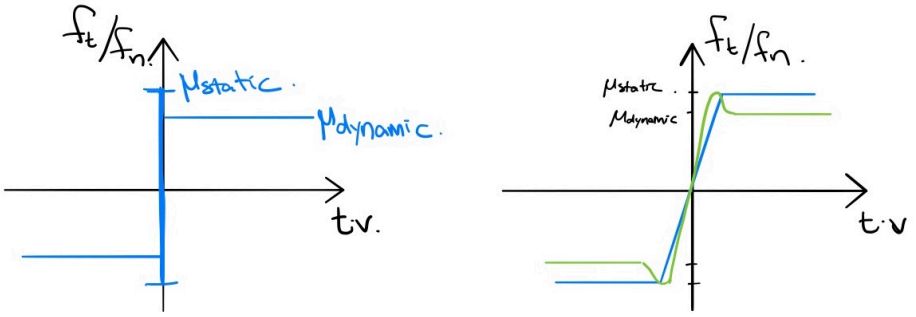



Figure B.5 - (left) The Coloumb friction model. (right) A continuous piecewise-linear approximation of friction (green) and the Stribeck approximation of Coloumb friction (blue); the $x$-axis is the contact tangential velocity, and the $y$-axis is the friction coefficient.

With these two laws, we can recover the contact forces as relatively simple functions of the current state. However, the devil is in the details. There are a few features of this approach that can make it numerically unstable. If you've ever been working in a robotics simulator and watched your robot take a step only to explode out of the ground and go flying through the air, you know what I mean.

In order to tightly approximate the (nearly) rigid contact that most robots make with the world, the stiffness of the contact "springs" must be quite high. For instance, I might want my 180kg humanoid robot model to penetrate into the ground no more than 1mm during steady-state standing. The challenge with adding stiff springs to our model is that this results in stiff differential equations (it is not a coincidence that the word *stiff* is the common term for this in both springs and ODEs). As a result, the best implementations of compliant contact for simulation make use of special-purpose numerical integration recipes (e.g. [13]) and compliant contact models are often difficult to use in e.g. trajectory/feedback optimization.

But there is another serious numerical challenge with the basic implementation of this model. Computing the signed-distance function, $\phi(\mathbf{q})$, when it is non-negative is straightforward, but robustly computing the "penetration depth" once two bodies are in collision is not. Consider the case of use $\phi(\mathbf{q})$ to represent the maximum penetration depth of, say, two boxes that are contacting each other. With compliant contact, we must have some penetration to produce any contact force. But the direction of the normal vector, $\mathbf{n} = \frac{\partial \phi}{\partial \mathbf{q}}$, can easily change discontinuously as the point of maximal penetration moves, as I've tried to illustrate here:

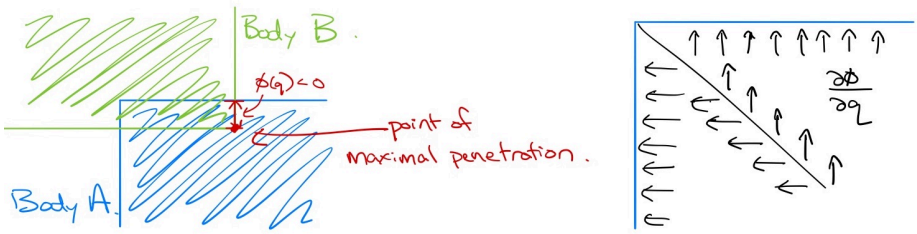



Figure B.6 - (Left) Two boxes in penetration, with the signed distance defined by the maximum penetration depth. (Right) The contact normal for various points of maximal penetration.

If you really think about this example, it actually even more of the foibles of trying to even define the contact points and normals consistently. It seems reasonable to define the point on body B as the point of maximal penetration into body A, but notice that as I've drawn it, that isn't actually unique! The corresponding point on body A should probably be the point on the surface with the minimal distance from the max penetration point (other tempting choices would likely cause the distance to be discontinuous at the onset of penetration). But this whole strategy is asymmetric; why shouldn't I have picked the vector going with maximal penetration into body B? My point is simply that these cases exist, and that even our best

software implementations get pretty unsatisfying when you look into the details. And in practice, it's a lot to expect the collision engine to give consistent and smooth contact points out in every situation.

Some of the advantages of this approach include (1) the fact that it is easy to implement, at least for simple geometries, (2) by virtue of being a continuous-time model it can be simulated with error-controlled integrators, and (3) the tightness of the approximation of "rigid" contact can be controlled through relatively intuitive parameters. However, the numerical challenges of dealing with penetration are very real, and they motivate our other two approaches that attempt to more strictly enforce the non-penetration constraints.

## B.3.2 Rigid Contact with Event Detection

## Impulsive Collisions

The collision event is described by the zero-crossings (from positive to negative) of $\phi$. Let us start by assuming frictionless collisions, allowing us to write

$$\mathbf{M}\ddot{\mathbf{q}} = \tau + \lambda \mathbf{n}^T, \tag{21}$$

where $\mathbf{n}$ is the "contact normal" we defined above and $\lambda \geq 0$ is now a (scalar) impulsive force that is well-defined when integrated over the time of the collision (denoted $t_c^-$ to $t_c^+$). Integrate both sides of the equation over that (instantaneous) interval:

$$\int_{t_c^-}^{t_c^+} dt \left[\mathbf{M}\ddot{\mathbf{q}}\right] = \int_{t_c^-}^{t_c^+} dt \left[\tau + \lambda \mathbf{n}^T\right]$$

Since $\mathbf{M}$, $\tau$, and $\mathbf{n}$ are constants over this interval, we are left with

$$\mathbf{M}\dot{\mathbf{q}}^+ - \mathbf{M}\dot{\mathbf{q}}^- = \mathbf{n}^T \int_{t_c^-}^{t_c^+} \lambda dt,$$

where $\dot{\mathbf{q}}^+$ is short-hand for $\dot{\mathbf{q}}(t_c^+)$. Multiplying both sides by $\mathbf{n}\mathbf{M}^{-1}$, we have

$$\mathbf{n}\dot{\mathbf{q}}^+ - \mathbf{n}\dot{\mathbf{q}}^- = \mathbf{n}\mathbf{M}^{-1}\mathbf{n}^T \int_{t_c^-}^{t_c^+} \lambda dt.$$

After the collision, we have $\dot{\phi}^+ = -e\dot{\phi}^-$, with $0 \leq e \leq 1$ denoting the coefficient of restitution, yielding:

$$\mathbf{n}\dot{\mathbf{q}}^+ - \mathbf{n}\dot{\mathbf{q}}^- = -(1+e)\mathbf{n}\dot{\mathbf{q}}^-,$$

and therefore

$$\int_{t_c^-}^{t_c^+} \lambda dt = -(1+e)\left[\mathbf{n}\mathbf{M}^{-1}\mathbf{n}^T\right]^{\#}\mathbf{n}\dot{\mathbf{q}}^-.$$

I've used the notation $A^{\#}$ here to denote the pseudoinverse of $A$ (I would normally write $A^+$, but have changed it for this section to avoid confusion). Substituting this back in above results in

$$\dot{\mathbf{q}}^+ = \left[\mathbf{I} - (1+e)\mathbf{M}^{-1}\mathbf{n}^T\left[\mathbf{n}\mathbf{M}^{-1}\mathbf{n}^T\right]^{\#}\mathbf{n}\right]\dot{\mathbf{q}}^-. \tag{22}$$

We can add friction at the contact. Rigid bodies will almost always experience contact at only a point, so we typically ignore torsional friction [8], and model only tangential friction by extending $\mathbf{n}$ to a matrix

$$\mathbf{J} = \begin{bmatrix} \mathbf{n} \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix},$$

to capture the gradient of the contact location tangent to the contact surface. Then $\lambda$ becomes a Cartesian vector representing the contact impulse, and (for infinite friction) the post-impact velocity condition becomes $\mathbf{J}\dot{\mathbf{q}}^+ = \text{diag}(-e, 0, 0)\mathbf{J}\dot{\mathbf{q}}^-$, resulting in the equations:

$$\dot{\mathbf{q}}^+ = \left[\mathbf{I} - \mathbf{M}^{-1}\mathbf{J}^T \text{diag}(1+e, 1, 1)\left[\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right]^{\#}\mathbf{J}\right]\dot{\mathbf{q}}^-. \tag{23}$$

If $\lambda$ is restricted to a friction cone, as in Coulomb friction, in general we have to solve an optimization to solve for $\dot{\mathbf{q}}^+$ subject to the inequality constraints.

**Example B.3 (A spinning ball bouncing on the ground.)**

Imagine a ball (a hollow-sphere) in the plane with mass $m$, radius $r$. The configuration is given by $q = [x, z, \theta]^T$. The equations of motion are

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & \frac{2}{3}mr^2 \end{bmatrix}\ddot{\mathbf{q}} = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & r \end{bmatrix}\begin{bmatrix} \lambda_z \\ \lambda_x \end{bmatrix} = \tau_g + \mathbf{J}^T\lambda,$$

where $\lambda$ are the contact forces (which are zero except during the impulsive collision). Given a coefficient of restitution $e$ and a collision with a horizontal ground, the post-impact velocities are:

$$\dot{\mathbf{q}}^+ = \begin{bmatrix} \frac{3}{5} & 0 & -\frac{2}{5}r \\ 0 & -e & 0 \\ -\frac{3}{5r} & 0 & \frac{2}{5} \end{bmatrix}\dot{\mathbf{q}}^-.$$

## Putting it all together

We can put the bilateral constraint equations and the impulsive collision equations together to implement a hybrid model for unilateral constraints of the form. Let us generalize

$$\phi(\mathbf{q}) \geq 0, \tag{24}$$

to be the vector of all pairwise (signed) distance functions between rigid bodies; this vector describes all possible contacts. At every moment, some subset of the contacts are active, and can be treated as a bilateral constraint ($\phi_i = 0$). The guards of the hybrid model are when an inactive constraint becomes active ($\phi_i > 0 \rightarrow \phi_i = 0$), or when an active constraint becomes inactive ($\lambda_i > 0 \rightarrow \lambda_i = 0$ and $\dot{\phi}_i > 0$). Note that collision events will (almost always) only result in new active constraints when $e = 0$ , e.g. we have pure inelastic collisions, because elastic collisions will rarely permit sustained contact.

If the contact involves Coulomb friction, then the transitions between sticking and sliding can be modeled as additional hybrid guards.

## B.3.3 Time-stepping Approximations for Rigid Contact

So far we have seen two different approaches to enforcing the inequality constraints of contact, $\phi(\mathbf{q}) \geq 0$ and the friction cone. First we introduced compliant contact which effectively enforces non-penetration with a stiff spring. Then we discussed event detection as a means to switch between different models which treat the active constraints as equality constraints. But, perhaps surprisingly, one of the most

popular and scalable approaches is something different: it involves formulating a mathematical program that can solve the inequality constraints directly on each time step of the simulation. These algorithms may be more expensive to compute on each time step, but they allow for stable simulations with potentially much larger and more consistent time steps.

## Complementarity formulations

What class of mathematical program due we need to simulate contact? The discrete nature of contact suggests that we might need some form of combinatorial optimization. Indeed, the most common transcription is to a Linear Complementarity Problem (LCP) [14], as introduced by [15] and [16]. An LCP is typically written as

$$\begin{aligned} \underset{\mathbf{w},\mathbf{z}}{\text{find}} \quad \text{subject to} \quad & \mathbf{w} = \mathbf{q} + \mathbf{M}\mathbf{z}, \\ & \mathbf{w} \geq 0, \mathbf{z} \geq 0, \mathbf{w}^T\mathbf{z} = 0. \end{aligned}$$

They are directly related to the optimality conditions of a (potentially non-convex) quadratic program and [16]> showed that the LCP's generated by our rigid-body contact problems can be solved by Lemke's algorithm. As short-hand we will write these complementarity constraints as

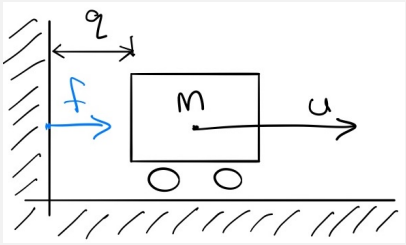
$$\underset{\mathbf{z}}{\text{find}} \quad \text{subject to} \quad 0 \leq (\mathbf{q} + \mathbf{M}\mathbf{z}) \perp \mathbf{z} \geq 0.$$

Rather than dive into the full transcription, which has many terms and can be relatively difficult to parse, let me start with two very simple examples.

### Example B.4 (Time-stepping LCP: Normal force)

Consider our favorite simple mass being actuated by a horizontal force (with the double integrator dynamics), but this time we will add a wall that will prevent the cart position from taking negative values: our non-penetration constraint is $q \geq 0$. Physically, this constraint is implemented by a normal (horizontal) force, $f$, yielding the equations of motion:

$$m\ddot{q} = u + f.$$




$f$ is defined as the force required to enforce the non-penetration constraint; certainly the following are true: $f \geq 0$ and $q \cdot f = 0$. $q \cdot f = 0$ is the "complementarity constraint", and you can read it here as "either q is zero or force is zero" (or both); it is our "no force at a distance" constraint, and it is clearly non-convex. It turns out that satisfying these constraints, plus $q \geq 0$, is sufficient to fully define $f$.

To define the LCP, we first discretize time, by approximating

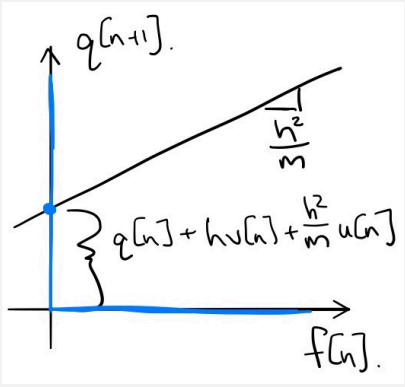
$$\begin{aligned} q[n+1] &= q[n] + hv[n+1], \\ v[n+1] &= v[n] + \frac{h}{m}(u[n] + f[n]). \end{aligned}$$

This is almost the standard Euler approximation, but note the use of $v[n+1]$ in the right-hand side of the first equation -- this is actually a semi-implicit Euler approximation, and this choice is essential in the derivation.

Given $h, q[n], v[n]$, and $u[n]$, we can solve for $f[n]$ and $q[n+1]$ simultaneously, by solving the following LCP:

$$q[n+1] = \left[q[n] + hv[n] + \frac{h^2}{m}u[n]\right] + \frac{h^2}{m}f[n]$$
$$q[n+1] \geq 0, \quad f[n] \geq 0, \quad q[n+1] \cdot f[n] = 0.$$

Take a moment and convince yourself that this fits into the LCP prescription given above.




Note: Please don't confuse this visualization with the more common visualization of the solution space for an LCP (in two or more dimensions) in terms of "complementary cones"[14].

Perhaps it's also helpful to plot the solution, $q[n+1], f[n]$ as a function of $q[n], v[n]$. I've done it here for $m = 1, h = 0.1, u[n] = 0$:

In the (time-stepping, "impulse-velocity") LCP formulation, we write the contact problem in it's combinatorial form. In the simple example above, the complementarity constraints force any solution to lie on *either* the positive x-axis ( $f[n] \geq 0$) *or* the positive y-axis ($q[n+1] \geq 0$). The equality constraint is simply a line that will intersect with this constraint manifold at the solution. But in this frictionless case, it is important to realize that these are simply the optimality conditions of a convex optimization problem: the discrete-time version of Gauss's principle that we used above. Using $\mathbf{v}'$ as shorthand for $\mathbf{v}[n+1]$, and replacing $\ddot{\mathbf{q}} = \frac{\mathbf{v}'-\mathbf{v}}{h}$ in Eq (13) and scaling the objective by $h^2$ we have:

$$\begin{aligned} \min_{\mathbf{v}'} \quad & \frac{1}{2}\left(\mathbf{v}' - \mathbf{v} - h\mathbf{M}^{-1}\tau\right)^T \mathbf{M}\left(\mathbf{v}' - \mathbf{v} - h\mathbf{M}^{-1}\tau\right) \\ \text{subject to} \quad & \frac{1}{h}\phi(\mathbf{q}') = \frac{1}{h}\phi(\mathbf{q} + h\mathbf{v}') \approx \frac{1}{h}\phi(\mathbf{q}) + \mathbf{n}\mathbf{v}' \geq 0. \end{aligned}$$

The objective is even cleaner/more intuitive if we denote the next step velocity that would have occurred before the contact impulse is applied as $\mathbf{v}^- = \mathbf{v} + h\mathbf{M}^{-1}\tau$:

$$\begin{aligned} \min_{\mathbf{v}'} \quad & \frac{1}{2}\left(\mathbf{v}' - \mathbf{v}^-\right)^T \mathbf{M}\left(\mathbf{v}' - \mathbf{v}^-\right) \\ \text{subject to} \quad & \frac{1}{h}\phi(\mathbf{q}') \geq 0. \end{aligned}$$

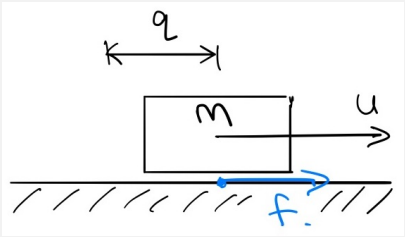
The LCP for the frictionless contact dynamics is precisely the optimality conditions of this convex (because $\mathbf{M} \succeq 0$) quadratic program, and once again the contact impulse, $\mathbf{f} \geq 0$, plays the role of the Lagrange multiplier (with units $N \cdot s$).

So why do we talk so much about LCPs instead of QPs? Well LCPs can also represent a wider class of problems, which is what we arrive at with the standard transcription

of Coulomb friction. In the limit of infinite friction, then we could add an additional constraint that the tangential velocity at each contact point was equal to zero (but these equation may not always have a feasible solution). Once we admit limits on the magnitude of the friction forces, the non-convexity of the disjunctive form rear's it's ugly head.

### Example B.5 (Time-stepping LCP: Coulomb Friction)

We can use LCP to find a feasible solution with Coulomb friction, but it requires some gymnastics with slack variables to make it work. For this case in particular, I believe a very simple example is best. Let's take our brick and remove the wall and the wheels (so we now have friction with the ground).




The dynamics are the same as our previous example,

$$m\ddot{q} = u + f,$$

but this time I'm using $f$ for the friction force which is inside the friction cone if $\dot{q} = 0$ and on the boundary of the friction cone if $\dot{q} \neq 0$; this is known as the principle of maximum dissipation[15]. Here the magnitude of the normal force is always $mg$, so we have $|f| \leq \mu mg$, where $\mu$ is the coefficient of friction. And we will use the same semi-implicit Euler approximation to cast this into discrete time.

Now, to write the friction constraints as an LCP, we must introduce some slack variables. First, we'll break the force into a positive component and a negative component: $f[n] = f^+ - f^-$. And we will introduce one more variable $v_{abs}$ which will be non-zero if the velocity is non-zero (in either direction). Now we can write the LCP:

$$\begin{aligned} &\underset{v_{abs}, f^+, f^-}{\text{find}} \quad \text{subject to} \\ 0 \leq v_{abs} + v[n+1] \quad &\perp \quad f^+ \geq 0, \\ 0 \leq v_{abs} - v[n+1] \quad &\perp \quad f^- \geq 0, \\ 0 \leq \mu mg - f^+ - f^- \quad &\perp \quad v_{abs} \geq 0, \end{aligned}$$

where each instance of $v[n+1]$ we write out with

$$v[n+1] = v[n] + \frac{h}{m}(u + f^+ - f^-).$$

It's enough to make your head spin! But if you work it out, all of the constraints we want are there. For example, for $v[n+1] > 0$, then we must have $f^+ = 0$, $v_{abs} = v[n+1]$, and $f^- = \mu mg$. When $v[n+1] = 0$, we can have $v_{abs} = 0$, $f^+ - f^- \leq \mu mg$, and those forces must add up to make $v[n+1] = 0$.

We can put it all together and write an LCP with both normal forces and friction forces, related by Coulomb friction (using a polyhedral approximation of the friction cone in 3d)[15]. Although the solution to any LCP can also be represented as the solution to a QP, the QP for this problem is non-convex.

## Anitescu's convex formulation

In [17], Anitescu described the natural convex formulation of this problem by dropping the maximum dissipation inequalities and allowing any force inside the friction cone. Consider the following optimization:

$$\begin{aligned} \min_{\mathbf{v}'} \quad & \frac{1}{2}\left(\mathbf{v}' - \mathbf{v}^-\right)^T \mathbf{M} \left(\mathbf{v}' - \mathbf{v}^-\right) \\ \text{subject to} \quad & \frac{1}{h}\phi(\mathbf{q}') + \mu \mathbf{d}_i \mathbf{v}' \geq 0, \qquad \forall i \in 1, \dots, m \end{aligned}$$

where $\mathbf{d}_i, \forall i \in 1, \dots, m$ are a set of tangent row vectors in joint coordinates parameterizing the polyhedral approximation of the friction cone (as in [15]). Importantly, for each $\mathbf{d}_i$ we must also have $-\mathbf{d}_i$ in the set. By writing the Lagrangian,

$$L(\mathbf{v}', \beta) = \frac{1}{2}(\mathbf{v}' - \mathbf{v}^-)^T \mathbf{M} (\mathbf{v}' - \mathbf{v}^-) - \sum_i \beta_i \left( \frac{1}{h}\phi(\mathbf{q}) + (\mathbf{n} + \mu \mathbf{d}) \mathbf{v}' \right),$$

and checking the stationarity condition:

$$\frac{\partial L}{\partial \mathbf{v}'}^T = \mathbf{M}(\mathbf{v}' - \mathbf{v}) - h\tau - \sum_i \beta_i (\mathbf{n} + \mu \mathbf{d}_i)^T = 0,$$

we can see that the Lagrange multiplier, $\beta_i \geq 0$, associated with the $i$th constraint is the magnitude of the impulse (with units $N \cdot s$) in the direction $\mathbf{n} - \mu \mathbf{d}_i$, where $\mathbf{n} = \frac{\partial \phi}{\partial \mathbf{q}}$ is the contact normal; the sum of the forces is an affine combination of these extreme rays of the polyhedral friction cone. As written, the optimization is a QP.

But be careful! Although the primal solution was convex, and the dual problems are always convex, the objective here can be positive semi-definite. This isn't a mistake -- [17] describes a few simple examples where the solution to $\mathbf{v}'$ is unique, but the impulses that produce it are not (think of a table with four legs).

When the tangential velocity is zero, this constraint is tight; for sliding contact the relaxation effectively causes the contact to "hydroplane" up out of contact, because $\phi(\mathbf{q}') \geq h \mu \mathbf{d}_i \mathbf{v}'$. It seems like a quite reasonable approximation to me, especially for small $h$!

Let's continue on and write the dual program. To make the notation a little better, the us stack the Jacobian vectors into $\mathbf{J}_\beta$, such that the $i$th row is $\mathbf{n} + \mu \mathbf{d_i}$, so that we have

$$\frac{\partial L}{\partial \mathbf{v}'}^T = \mathbf{M}(\mathbf{v}' - \mathbf{v}) - h\tau - \mathbf{J}_\beta^T \beta = 0.$$

Substituting this back into the Lagrangian give us the dual program:

$$\min_{\beta \geq 0} \frac{1}{2} \beta^T \mathbf{J}_\beta \mathbf{M}^{-1} \mathbf{J}_\beta^T \beta + \frac{1}{h} \phi(\mathbf{q}) \sum_i \beta_i.$$

One final note: I've written just one contact point here to simplify the notation, but of course we repeat the constraint(s) for each contact point; [17] studies the typical case of only including potential contact pairs with $\phi(\mathbf{q}) \leq \epsilon$, for small $\epsilon \geq 0$.

## Todorov's convex formulation

According to [18], the popular MuJoCo simulator uses a slight variation of this convex relaxation, with the optimization:

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2} \lambda^T \mathbf{J} \mathbf{M}^{-1} \mathbf{J}^T \lambda + \frac{1}{h} \lambda^T \left( \mathbf{J} \mathbf{v}^- - \dot{\mathbf{x}}^d \right), \\ \text{subject to} \quad & \lambda \in \mathcal{FC}(\mathbf{q}), \end{aligned}$$

where $\dot{\mathbf{x}}^d$ is a "desired contact velocity", and $\mathcal{FC}(\mathbf{q})$ describes the friction cone. The friction cone constraint looks new but is not; observe that $\mathbf{J}^T \lambda = \mathbf{J}_\beta^T \beta$, where $\beta \geq 0$

is a clever parameterization of the friction cone in the polyhedral case. The bigger difference is in the linear term: [18] proposed $\dot{\mathbf{x}}^d = \mathbf{J}\mathbf{v} - h\mathcal{B}\mathbf{J}\mathbf{v} - h\mathcal{K}[\phi(\mathbf{q}), 0, 0]^T$, with $\mathcal{B}$ and $\mathcal{K}$ stabilization gains that are chosen to yield a critically damped response.

How should we interpret this? If you expand the linear terms, you will see that this is almost exactly the dual form we get from the position equality constraints formulation, including the Baumgarte stabilization. It's interesting to think about the implications of this formulation -- like the Anitescu relaxation, it is possible to get some "force at a distance" because we have not in any way expressed that $\lambda = 0$ when $\phi(\mathbf{q}) > 0$. In Anitescu, it happened in a velocity-dependent boundary layer; here it could happen if you are "coming in hot" -- moving towards contact with enough relative velocity that the stabilization terms want to slow you down.

In dual form, it is natural to consider the full conic description of the friction cone:

$$\mathcal{FC}(\mathbf{q}) = \left\{ \lambda = [f_n, f_{t1}, f_{t2}]^T \; f_n \geq 0, \sqrt{f_{t1}^2 + f_{t2}^2} \leq \mu f_n \right\}.$$

The resulting dual optimization is has a quadratic objective and second-order cone constraints (SOCP).

There are a number of other important relaxations that happen in MuJoCo. To address the positive indefiniteness in the objective, [18] relaxes the dual objective by adding a small term to the diagonal. This guarantees convexity, but the convex optimization can still be poorly conditioned. The stabilization terms in the objective are another form of relaxation, and [18] also implements adding additional stabilization to the inertial matrix, called the "implicit damping inertia". These relaxations can dramatically improve simulation speed, both by making each convex optimization faster to solve, and by allowing the simulator to take larger integration time steps. MuJoCo boasts a special-purpose solver that can simulate complex scenes at seriously impressive speeds -- often orders of magnitude faster than real time. But these relaxations can also be abused -- it's unfortunately quite common to see physically absurd MuJoCo simulations, especially when researchers who don't care much about the physics start changing simulation parameters to optimize their learning curves!

### Beyond Point Contact

Coming soon... Box-on-plane example. Multi-contact.

Also a single point force cannot capture effects like torsional friction, and performs badly in some very simple cases (imaging a box sitting on a table). As a result, many of the most effective algorithms for contact restrict themselves to very limited/simplified geometry. For instance, one can place "point contacts" (zero-radius spheres) in the four corners of a robot's foot; in this case adding forces at exactly those four points as they penetrate some other body/mesh can give more consistent contact force locations/directions. A surprising number of simulators, especially for legged robots, do this.

In practice, most collision detection algorithms will return a list of "collision point pairs" given the list of bodies (with one point pair per pair of bodies in collision, or more if they are using the aforementioned "multi-contact" heuristics), and our contact force model simply attaches springs between these pairs. Given a point-pair, $p_A$ and $p_B$, both in world coordinates, ...

Hydroelastic model in drake[19].

## B.4 Principle of Stationary Action

[20] says

> The principle of least action -- really the principle of stationary action -- is

> the most compact form of the classical laws of physics. This simple rule (it can be written in a single line) summarizes everything! Not only the principles of classical mechanics, but electromagnetism, general relativity, quantum mechanics, everything known about chemistry -- right down to the ultimate known constituents of matter, elementary particles.

Sounds important!

I find a lot of presentations of the principle of stationary action derivation of the Lagrangian unnecessarily confusing. Here's my version, in case it helps.

Consider a trajectory $\mathbf{q}(t)$ defined over $t \in [t_0, t_1]$. The principle of stationary action states that this trajectory is a valid solution to the differential equation governing our system if it represents a stationary point of the "action", defined as

$$A = \int_{t_0}^{t_1} L(\mathbf{q}, \dot{\mathbf{q}}) dt.$$

What does it mean for a trajectory to be a stationary point? Using the calculus of variations, we think of an infinitesimal variation of this trajectory: $\mathbf{q}(t) + \epsilon(t)$, where $\epsilon(t)$ is an arbitrary differentiable function that is zero at $t_0$ and $t_1$. That this variation should not change the action means $\delta A = 0$ for any small $\epsilon(t)$:

$$\begin{aligned}\delta A =& A(\mathbf{q}(t) + \epsilon(t)) - A(\mathbf{q}(t)) \\ =& \int_{t_0}^{t_1} L\left(\mathbf{q}(t) + \epsilon(t),\, \dot{\mathbf{q}}(t) + \dot{\epsilon}(t)\right) dt - \int_{t_0}^{t_1} L\left(\mathbf{q}(t),\, \dot{\mathbf{q}}(t)\right) dt.\end{aligned}$$

We can expand the term in the first integral:

$$L\left(\mathbf{q}(t) + \epsilon(t),\, \dot{\mathbf{q}}(t) + \dot{\epsilon}(t)\right) = L\left(\mathbf{q}(t),\, \dot{\mathbf{q}}(t)\right) + \frac{\partial L}{\partial \mathbf{q}(t)}\epsilon(t) + \frac{\partial L}{\partial \dot{\mathbf{q}}(t)}\dot{\epsilon}(t).$$

Substituting this back in and simplifying leaves:

$$\delta A = \int_{t_0}^{t_1} \left[\frac{\partial L}{\partial \mathbf{q}(t)}\epsilon(t) + \frac{\partial L}{\partial \dot{\mathbf{q}}(t)}\dot{\epsilon}(t)\right] dt.$$

Now use integration by parts on the second term:

$$\int_{t_0}^{t_1} \frac{\partial L}{\partial \dot{\mathbf{q}}(t)}\dot{\epsilon}(t) dt = \left[\frac{\partial L}{\partial \dot{\mathbf{q}}(t)}\epsilon(t)\right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}(t)}\epsilon(t) dt,$$

and observe that the first term in the right-hand side is zero since $\epsilon(t_0) = \epsilon(t_1) = 0$. This leaves

$$\delta A = \int_{t_0}^{t_1} \left[\frac{\partial L}{\partial \mathbf{q}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}}\right]\epsilon(t) dt = 0.$$

Since this must integrate to zero for any $\epsilon(t)$, we must have

$$\frac{\partial L}{\partial \mathbf{q}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}} = 0,$$

which can be multiplied by negative one to obtain the familiar form of the (unforced) Lagrangian equations of motion. The forced version follows from the variation

$$\delta A = \int_{t_0}^{t_1} L(\mathbf{q}(t), \dot{\mathbf{q}}(t)) dt + \int_{t_0}^{t_1} \mathbf{Q}^T(t)\epsilon(t) dt = 0.$$

For a much richer treatment, see [2].

# References


1. John Craig, "Introduction to Robotics: Mechanics and Control",Addison Wesley , January, 1989.

2. Cornelius Lanczos, "The variational principles of mechanics",University of Toronto Press , no. no. 4, 1970.

3. H. Asada and J.E. Slotine, "Robot Analysis and Control", , pp. 93-131, 1986.

4. Roy Featherstone, "Efficient Factorization of the Joint Space Inertia Matrix for Branched Kinematic Trees", *International Journal of Robotics Research*, vol. 24, no. 6, pp. 487–500, 2005.

5. H. Choset and K. M. Lynch and S. Hutchinson and G. Kantor and W. Burgard and L. E. Kavraki and S. Thrun, "Principles of Robot Motion-Theory, Algorithms, and Implementations",The MIT Press , 2005.

6. Jean-Jacques E. Slotine and Weiping Li, "Applied Nonlinear Control",Prentice Hall , October, 1990.

7. Vincent Duindam, "Port-Based Modeling and Control for Efficient Bipedal Walking Robots", PhD thesis, University of Twente, March, 2006.

8. Roy Featherstone, "Rigid Body Dynamics Algorithms",Springer , 2007.

9. Brian Mirtich, "Impulse-based Dynamic Simulation of Rigid Body Systems", PhD thesis, University of California at Berkeley, 1996.

10. Abhinandan Jain, "Robot and Multibody Dynamics: Analysis and Algorithms",Springer US , 2011.

11. Firdaus E Udwadia and Robert E Kalaba, "A new perspective on constrained motion", *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1906, pp. 407--410, 1992.

12. K. H. Hunt and F. R. E. Crossley, "Coefficient of restitution interpreted as damping in vibroimpact", *Journal of Applied Mechanics*, vol. 42 Series E, no. 2, pp. 440-445, 1975.

13. Alejandro M Castro and Ante Qu and Naveen Kuppuswamy and Alex Alspach and Michael Sherman, "A Transition-Aware Method for the Simulation of Compliant Contact with Regularized Friction", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1859--1866, 2020.

14. Katta G Murty and Feng-Tien Yu, "Linear complementarity, linear and nonlinear programming",Citeseer , vol. 3, 1988.

15. D.E. Stewart and J.C. Trinkle, "AN IMPLICIT TIME-STEPPING SCHEME FOR RIGID BODY DYNAMICS WITH INELASTIC COLLISIONS AND COULOMB FRICTION", *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673--2691, 1996.

16. M. Anitescu and F.A. Potra, "Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems", *Nonlinear Dynamics*, vol. 14, no. 3, pp. 231--247, 1997.

17. Mihai Anitescu, "Optimization-based simulation of nonsmooth rigid multibody dynamics", *Mathematical Programming*, vol. 105, no. 1, pp. 113--143, jan, 2006.

18. Emanuel Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in {MuJoCo}", *2014*

*IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6054--6061, 2014.

19. Ryan Elandt and Evan Drumwright and Michael Sherman and Andy Ruina, "A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects", , pp. 8238-8245, 11, 2019.

20. Leonard Susskind and George Hrabovsky, "The theoretical minimum: what you need to know to start doing physics",Basic Books , 2014.

# UNDERACTUATED ROBOTICS

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

# APPENDIX C Optimization and Mathematical Programming


The view of dynamics and controls taken in these notes builds heavily on tools from optimization -- and our success in practice depends heavily on the effective application of numerical optimization. There are many excellent books on optimization, for example [1] is an excellent reference on smooth optimization and [2] is an excellent reference on convex optimization (which we use extensively). I will provide more references for the specific optimization formulations below.

The intention of this chapter is, therefore, mainly to provide a launchpad and to address some topics that are particularly relevant in robotics but might be more hidden in the existing general treatments. You can get far very quickly as a consumer of these tools with just a high-level understanding. But, as with many things, sometimes the details matter and it becomes important to understand what is going on under the hood. We can often formulate an optimization problem in multiple ways that might be mathematically equivalent, but perform very differently in practice.

## C.1 OPTIMIZATION SOFTWARE

Some of the algorithms from optimization are quite simple to implement yourself; stochastic gradient descent is perhaps the classic example. Even more of them are conceptually fairly simple, but for some of the algorithms, the implementation details matter a great deal -- the difference between an expert implementation and a novice implementation of the numerical recipe, in terms of speed and/or robustness, can be dramatic. These packages often use a wealth of techniques for numerically conditioning the problems, for discarding trivially valid constraints, and for warm-starting optimization between solves. Not all solvers support all types of objectives and constraints, and even if we have two commercial solvers that both claim to solve, e.g. quadratic programs, then they might perform differently on the particular structure/conditioning of your problem. In some cases, we also have nicely designed open-source alternatives to these commercial solvers (often written by academics who are experts in optimization) -- sometimes they compete well with the commercial solvers or even outperform them in particular problem types.

As a result, there are also a handful of software packages that attempt to provide a layer of abstraction between the formulation of a mathematical program and the instantiation of that problem in each of the particular solvers. Famous examples of this include CVX and YALMIP for MATLAB, and JuMP for Julia. DRAKE's MathematicalProgram classes provide a middle layer like this for C++ and Python; its creation was motivated initially and specifically by the need to support the optimization formulations we use in this text.

We have a number of tutorials on mathematical programming in DRAKE, starting with a general introduction here. Drake supports a number of custom, open-source, and commercial solvers (and even some of the commercial solvers are free to academic users).

## C.2 GENERAL CONCEPTS

The general formulation is ...

It's important to realize that even though this formulation is incredibly general, it does have it's limits. As just one example, when write optimizations to plan trajectories of a robot, in this formulation we typically have to choose a-priori as particular number of decision variables that will encode the solution. Although we can of course write algorithms that change the number of variables and call the optimizer again, somehow I feel that this "general" formulation fails to capture, for instance, the type of mathematical programming that occurs in sample-based optimization planning -- where the resulting solutions can be described by any finite number of parameters, and the computation flexibly transitions amongst them.

### C.2.1 Convex vs nonconvex optimization

Local optima. Convex functions, convex constraints.

If an optimization problem is nonconvex, it does not necessarily mean that the optimization is hard. There are many cases in deep learning where we can reliably solve seemingly very high-dimensional and nonconvex optimization problems. Our understanding of these phenomenon is evolving rapidly, and I suspect anything I write here will shortly become outdated. But the current thinking in supervised learning mostly revolves around the idea that over-parameterization is key -- that many of these success stories are happening in a regime where we actually have more decision variables than we have data, and that the search space is actually dense with solutions that can fit the data perfectly (the so-called "interpolating solutions"), that not all global minima are equally robust, and that the optimization algorithms are performing some form of implicit regularization in order to pick a "good" interpolating solution.

### C.2.2 Constrained optimization with Lagrange multipliers

Given the equality-constrained optimization problem

$$\text{minimize}_{\mathbf{z}} \ell(\mathbf{z}) \quad \text{subject to} \quad \phi(\mathbf{z}) = 0,$$

where $\phi$ is a vector. Define a vector $\lambda$ of Lagrange multipliers, the same size as $\phi$, and the scalar Lagrangian function,

$$L(\mathbf{z}, \lambda) = \ell(\mathbf{z}) + \lambda^T \phi(\mathbf{z}).$$

A necessary condition for $\mathbf{z}^*$ to be an optimal value of the constrained optimization is that the gradients of $L$ vanish with respect to both $\mathbf{z}$ and $\lambda$:

$$\frac{\partial L}{\partial \mathbf{z}} = 0, \quad \frac{\partial L}{\partial \lambda} = 0.$$

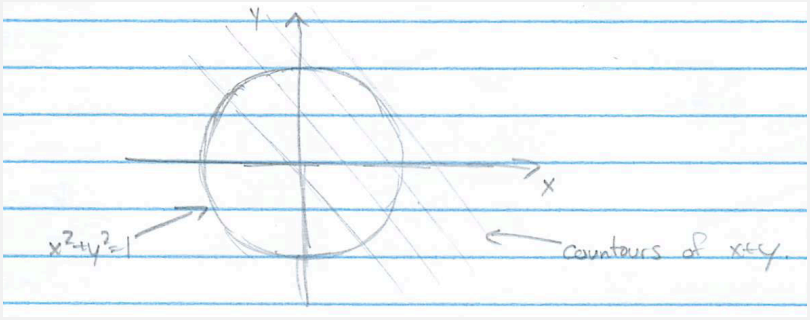
Note that $\frac{\partial L}{\partial \lambda} = \phi(\mathbf{z})$, so requiring this to be zero is equivalent to requiring the constraints to be satisfied.

### Example C.1 (Optimization on the unit circle)

Consider the following optimization:

$$\min_{x,y} x + y, \quad \text{subject to} \quad x^2 + y^2 = 1.$$

The level sets of $x + y$ are straight lines with slope $-1$, and the constraint requires that the solution lives on the unit circle.




Simply by inspection, we can determine that the optimal solution should be $x = y = -\frac{\sqrt{2}}{2}$. Let's make sure we can obtain the same result using Lagrange multipliers.

Formulating

$$L = x + y + \lambda(x^2 + y^2 - 1),$$

we can take the derivatives and solve

$$\begin{aligned}
\frac{\partial L}{\partial x} = 1 + 2\lambda x = 0 \quad &\Rightarrow \quad \lambda = -\frac{1}{2x}, \\
\frac{\partial L}{\partial y} = 1 + 2\lambda y = 1 - \frac{y}{x} = 0 \quad &\Rightarrow \quad y = x, \\
\frac{\partial L}{\partial \lambda} = x^2 + y^2 - 1 = 2x^2 - 1 = 0 \quad &\Rightarrow \quad x = \pm\frac{1}{\sqrt{2}}.
\end{aligned}$$

Given the two solutions which satisfy the necessary conditions, the negative solution is clearly the minimizer of the objective.

## C.3 Convex optimization

### C.3.1 Linear Programs/Quadratic Programs/Second-Order Cones

Example: Balancing force control on Atlas

## C.3.2 Semidefinite Programming and Linear Matrix Inequalities

### Example C.2 (Semidefinite programming relaxation of non-convex quadratic constraints)

Consider the problem:

$$\begin{aligned} \min_{x} \quad & \|x-a\|^2 \\ \text{subject to} \quad & \|x-b\| \geq 1 \end{aligned}$$

We can write this as

$$\begin{aligned} \min_{x,y} \quad & y - 2ax + a^2 \\ \text{subject to} \quad & y - 2bx + b^2 \geq 1 \\ & y \geq x^2 \end{aligned}$$

where we write $y \geq x^2$ as the semidefinite constraint $\begin{bmatrix} y & x \\ x & 1 \end{bmatrix}$.

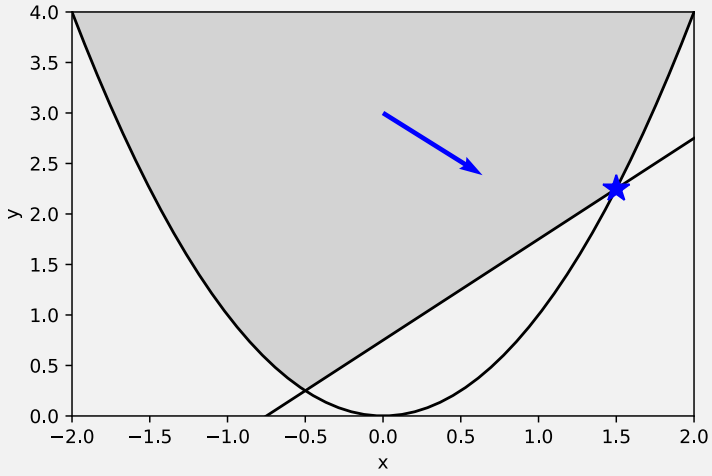



Figure C.2 - Optimization landscape for $a = .8, b = .5$.

I've plotted the feasible region and the objective with an arrow. As you can see, the feasible region is the epigraph of $y = x^2$ intersected with the linear constraint. Now here is the key point: for linear objectives, the optimal solution will lie on the boundary only if the cost is directly orthogonal to the objective; otherwise it will lie at a vertex. So in this case, the solution will only lie on the interior if $a = b$; for every other value, this relaxation will give the optimal solution. Note that we could have equally well written a quadratic equality constraint.

I find this incredibly clever, and only frustrating that it is somewhat particular to quadratics. (The fact that the entire exterior of the quadratic region could be an optimal solution does not hold if we try to use the same approach for e.g. being outside of a polytope).

Open in Colab

## C.3.3 Sums-of-squares optimization

It turns out that in the same way that we can use SDP to search over the positive quadratic equations, we can generalize this to search over the positive polynomial equations. To be clear, for quadratic equations we have

$$\mathbf{P} \succeq 0 \quad \Rightarrow \quad \mathbf{x}^T\mathbf{P}\mathbf{x} \geq 0, \quad \forall \mathbf{x}.$$

It turns out that we can generalize this to

$$\mathbf{P} \succeq 0 \quad \Rightarrow \quad \mathbf{m}^T(\mathbf{x})\mathbf{P}\mathbf{m}(\mathbf{x}) \geq 0, \quad \forall \mathbf{x},$$

where $\mathbf{m}(\mathbf{x})$ is a vector of polynomial equations, typically chosen as a vector of *monomials* (polynomials with only one term). The set of positive polynomials parameterized in this way is exactly the set of polynomials that can be written as a *sum of squares*[3]. While it is known that not all positive polynomials can be written in this form, much is known about the gap. For our purposes this gap is very small (papers have been written about trying to find polynomials which are uniformly positive but not sums of squares); we should remember that it exists but not worry about it too much for now.

Even better, there is quite a bit that is known about how to choose the terms in $\mathbf{m}(\mathbf{x})$. For example, if you give me the polynomial

$$p(x) = 2 - 4x + 5x^2$$

and ask if it is positive for all real $x$, I can convince you that it is by producing the sums-of-squares factorization

$$p(x) = 1 + (1 - 2x)^2 + x^2,$$

and I know that I can formulate the problem without needing any monomials of degree greater than 1 (the square-root of the degree of $p$) in my monomial vector. In practice, what this means for us is that people have authored optimization front-ends which take a high-level specification with constraints on the positivity of polynomials and they automatically generate the SDP problem for you, without having to think about what terms should appear in $\mathbf{m}(\mathbf{x})$ (c.f. [4]). This class of optimization problems is called Sums-of-Squares (SOS) optimization.

As it happens, the particular choice of $\mathbf{m}(\mathbf{x})$ can have a huge impact on the numerics of the resulting semidefinite program (and on your ability to solve it with commercial solvers). DRAKE implements some particularly novel/advanced algorithms to make this work well [5].

We will write optimizations using sums-of-squares constraints as

$$p(\mathbf{x}) \text{ is SOS}$$

as shorthand for the constraint that $p(\mathbf{x}) \geq 0$ for all $\mathbf{x}$, as demonstrated by finding a sums-of-squares decomposition.

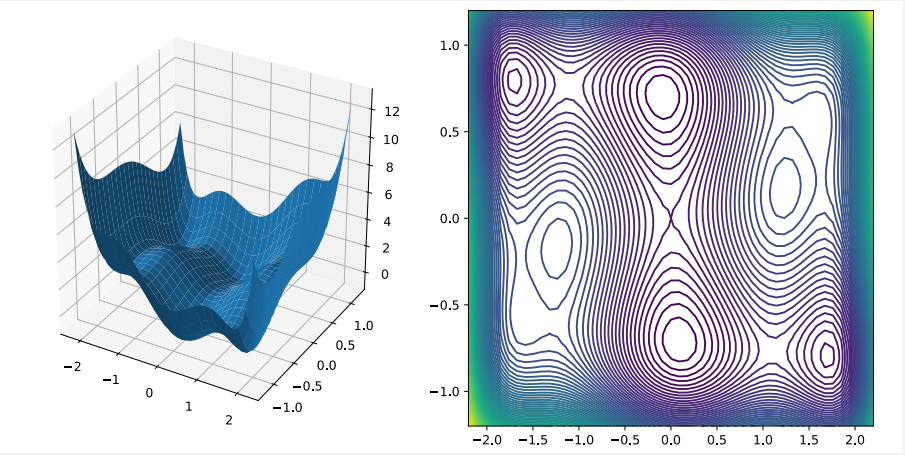
This is surprisingly powerful. It allows us to use convex optimization to solve what appear to be very non-convex optimization problems.

### Example C.3 (Global minimization via SOS)

Consider the following famous non-linear function of two variables (called the "six-hump-camel":

$$p(x) = 4x^2 + xy - 4y^2 - 2.1x^4 + 4y^4 + \frac{1}{3}x^6.$$

This function has six local minima, two of them being global minima [6].




By formulating a simple sums-of-squares optimization, we can actually find the minimum value of this function (technically, it is only a lower bound, but in this case and many cases, it is surprisingly tight) by writing:

$$\max_{\lambda} \quad \lambda$$
$$\text{s.t. } p(x) - \lambda \text{ is sos.}$$

Go ahead and play with the code (most of the lines are only for plotting; the actual optimization problem is nice and simple to formulate).

Open in Colab

Note that this finds the minimum value, but does not actually produce the $\mathbf{x}$ value which mimimizes it. This is possible [6], but it requires examining the dual of the sums-of-squares solutions (which we don't need for the goals of this chapter).

## Sums of squares on a Semi-Algebraic Set

The S-procedure.

## Sums of squares optimization on an Algebraic Variety

The S-procedure

Using the quotient ring

Quotient rings via sampling

## DSOS and SDSOS

## C.3.4 Solution techniques

Interior point (Gurobi, Mosek, Sedumi, ...), First order methods

# C.4 Nonlinear programming

The generic formulation of a nonlinear optimization problem is

$$\min_z c(z) \quad \text{subject to} \quad \phi(z) \leq 0,$$

where $z$ is a vector of *decision variables*, $c$ is a scalar *objective function* and $\phi$ is a vector of *constraints*. Note that, although we write $\phi \leq 0$, this formulation captures positivity constraints on the decision variables (simply multiply the constraint by $-1$ ) and equality constraints (simply list both $\phi \leq 0$ and $-\phi \leq 0$) as well.

The picture that you should have in your head is a nonlinear, potentially non-convex objective function defined over (multi-dimensional) $z$, with a subset of possible $z$ values satisfying the constraints.

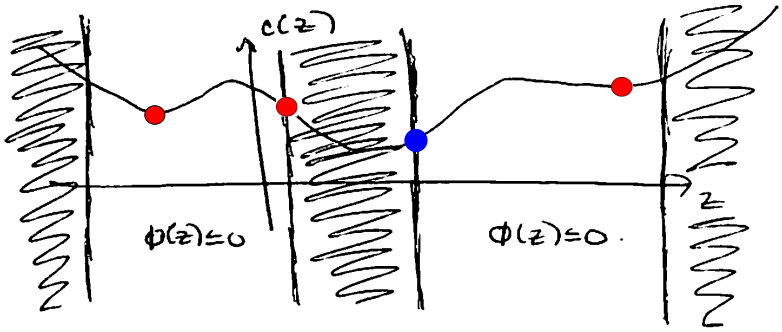



Figure C.3 - One-dimensional cartoon of a nonlinear optimization problem. The red dots represent local minima. The blue dot represents the optimal solution.

Note that minima can be the result of the objective function having zero derivative *or* due to a sloped objective up against a constraint.

Numerical methods for solving these optimization problems require an initial guess, $\hat{z}$, and proceed by trying to move down the objective function to a minima. Common approaches include *gradient descent*, in which the gradient of the objective function is computed or estimated, or second-order methods such as *sequential quadratic programming (SQP)* which attempts to make a local quadratic approximation of the objective function and local linear approximations of the constraints and solves a quadratic program on each iteration to jump directly to the minimum of the local approximation.

While not strictly required, these algorithms can often benefit a great deal from having the gradients of the objective and constraints computed explicitly; the alternative is to obtain them from numerical differentiation. Beyond pure speed considerations, I strongly prefer to compute the gradients explicitly because it can help avoid numerical accuracy issues that can creep in with finite difference methods. The desire to calculate these gradients will be a major theme in the discussion below, and we have gone to great lengths to provide explicit gradients of our provided functions and automatic differentiation of user-provided functions in DRAKE.

When I started out, I was of the opinion that there is nothing difficult about implementing gradient descent or even a second-order method, and I wrote all of the solvers myself. I now realize that I was wrong. The commercial solvers available for nonlinear programming are substantially higher performance than anything I wrote myself, with a number of tricks, subtleties, and parameter choices that can make a huge difference in practice. Some of these solvers can exploit sparsity in the problem (e.g., if the constraints operate in a sparse way on the decision variables). Nowadays, we make heaviest use of SNOPT [7], which now comes bundled with the binary distributions of DRAKE, but also support a large suite of numerical solvers. Note that while I do advocate using these tools, you do not need to use them as a black box. In many cases you can improve the optimization performance by understanding and selecting non-default configuration parameters.

### C.4.1 Second-order methods (SQP / Interior-Point)

### C.4.2 First-order methods (SGD / ADMM)

### Penalty methods

Augmented Lagrangian

### Projected Gradient Descent

### C.4.3 Zero-order methods (CMA)

### C.4.4 Example: Inverse Kinematics

## C.5 Combinatorial optimization

### C.5.1 Search, SAT, First order logic, SMT solvers, LP interpretation

### C.5.2 Mixed-integer convex optimization

An advanced, but very readable book on MIP [8]. Nice survey paper on MILP [9].

## C.6 "Black-box" optimization

Derivative-free methods. Some allow noisy evaluations.

## References


1. Jorge Nocedal and Stephen J. Wright, "Numerical optimization",Springer , 2006.

2. Stephen Boyd and Lieven Vandenberghe, "Convex Optimization",Cambridge University Press , 2004.

3. Pablo A. Parrilo, "Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization", PhD thesis, California Institute of Technology, May 18, 2000.

4. Stephen Prajna and Antonis Papachristodoulou and Peter Seiler and Pablo A. Parrilo, "SOSTOOLS: Sum of Squares Optimization Toolbox for MATLAB Userâ€™s guide", , June 1, 2004.

5. Frank Noble Permenter, "Reduction methods in semidefinite and conic optimization", PhD thesis, Massachusetts Institute of Technology, 2017. [ link ]

6. Didier Henrion and Jean-Bernard Lasserre and Johan Lofberg, "GloptiPoly 3: moments, optimization and semidefinite programming", *Optimization Methods \& Software*, vol. 24, no. 4-5, pp. 761--779, 2009.

7. Philip E. Gill and Walter Murray and Michael A. Saunders, "User's Guide for SNOPT Version 7: Software for Large -Scale Nonlinear Programming", ,

February 12, 2006.

8. Michele Conforti and G{\'e}rard Cornu{\'e}jols and Giacomo Zambelli and others, "Integer programming",Springer , vol. 271, 2014.

9. Juan Pablo Vielma, "Mixed integer linear programming formulation techniques", *Siam Review*, vol. 57, no. 1, pp. 3--57, 2015.

Russ Tedrake




Last modified 2021-6-13.

How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.


Previous Chapter | Table of contents | Next Chapter


# Appendix D
# An Optimization Playbook

Coming soon... a collection of tips and formulations that can make seemingly non-smooth constraints smooth, seemingly non-convex constraints convex, etc.

# Underactuated Robotics

***Algorithms for Walking, Running, Swimming, Flying, and Manipulation***

Russ Tedrake


Last modified 2021-6-13.
How to cite these notes, use annotations, and give feedback.

**Note:** These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2021 semester. Lecture videos are available on YouTube.

Previous Chapter Table of contents

# Appendix E
# Miscellaneous

## E.1 How to cite these notes

Thank you for citing these notes in your work. Please use the following citation:

Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832).* Downloaded on [date] from http://underactuated.mit.edu/

## E.2 Annotation tool etiquette

My primary goal for the annotation tool is to host a completely open dialogue on the intellectual content of the text. However, it has turned out to serve an additional purpose: it's a convenient way to point out my miscellaneous typos and grammatical blips. The only problem is that if you highlight a typo, and I fix it 10 minutes later, your highlight will persist forevermore. Ultimately this pollutes the annotation content.

There are two possible solutions:

- you can make a public editorial comment, but must promise to delete that comment once it has been addressed.
- You can join my "editorial" group and post your editorial comments using this group "scope".

Ideally, once I mark a comment as "done", I would appreciate it if you can delete that comment.

I highly value both the discussions and the corrections. Please keep them coming, and thank you!

## E.3 Some great final projects

Each semester students put together a final project using the tools from this class. Many of them are fantastic! Here is a small sample.

**Spring 2020:**

- Collision Free Mixed Integer Planning for Quadrotors Using Convex Safe Regions by Bernhard Paus Græsdal
- Pancake flipping via trajectory optimization by Charles Dawson
- Dynamic Soaring by Lukas Lao Beyer
- Acrobatic Humanoid by Matt Chignoli and AJ Miller
- Trajectory Optimization for the Furuta Pendulum by Samuel Cherna and Philip Murzynowski

Even before we started posting project presentations on YouTube, some great projects from class have turned into full publications. Here are a few examples:

- Feedback Control of the Pusher-Slider System: A Story of Hybrid and Underactuated Contact Dynamics
- Harnessing Structures for Value-Based Planning and Reinforcement Learning
- the famous Mini Cheetah backflip in Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control

## E.4 Please give me feedback!

I'm very interested in your feedback. The annotation tool is one mechanism, but you can also comment directly on the YouTube lectures, or even add issues to the github repo that hosts these course notes.

I've also created this simple survey to collect your general comments/feedback.