Underactuated Robotics: Learning, Planning, and Control for Efficient and Agile Machines

Course Notes for MIT 6.832

Russ Tedrake

Massachusetts Institute of Technology

Contents

CHAPTER 1

Fully Actuated vs. Underactuated Systems

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 over the last 12 years, resulting in a humanoid robot they call ASIMO (Advanced Step in Innovative MObility). Honda's ASIMO is widely considered to be 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 in chapter 5... for now I just want you to become familiar with the look and feel of ASIMO's movements [watch asimo video now[1]].

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 like somebody that is unfamiliar with his/her dynamics. It's 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)[25]. 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.

For contrast, let's now consider a very different type of walking robot, called a passive dynamic walker. This "robot" has no motors, no controllers, no computer, but is still capable of walking stably down a small ramp, powered only by gravity. Most people will agree that the passive gait of this machine is more natural than ASIMO's; it is certainly

[1] http://world.honda.com/ASIMO/

more efficient. [watch PDW videos now[2]]. 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.

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.

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[6], 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[73]; a barn swallow has a roll rate in excess of 5000 deg/sec[73]. 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[86]. 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[87]. 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[89], 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 un-

[2] `http://www-personal.engin.umich.edu/~shc/robots.html`

derstanding is partially limited by experimental accessibility - it's 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[87]. 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[90] - 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[10].

1.1.3 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 system which take advantage of the dynamics, not cancel them out. That is the topic of this course.

Surprisingly, there are relatively few formal control ideas that consider "exploiting" the dynamics. In order to convince a control theorist to consider the dynamics (efficiency arguments are not enough), you have to do something drastic, like taking away his 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, $\mathbf{q}$, and a vector of velocities, $\dot{\mathbf{q}}$, and (possibly) time. The general form for a second-order controllable dynamical system is:

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

where $\mathbf{u}$ is the control vector. As we will see, the forward 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:

$$\ddot{\mathbf{q}} = \mathbf{f}_1(\mathbf{q}, \dot{\mathbf{q}}, t) + \mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)\mathbf{u}; . \tag{1.1}$$

DEFINITION 1 (Fully-Actuated)**.** A control system described by equation 1.1 is fully-actuated in configuration $(\mathbf{q}, \dot{\mathbf{q}}, t)$ if it is able to command an instantaneous acceleration in an arbitrary direction in $\mathbf{q}$:

$$\text{rank}\left[\mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)\right] = \dim\left[\mathbf{q}\right]. \tag{1.2}$$

DEFINITION 2 (Underactuated)**.** A control system described by equation 1.1 is underactuated in configuration $(\mathbf{q}, \dot{\mathbf{q}}, t)$ if it is not able to command an instantaneous

acceleration in an arbitrary direction in $\mathbf{q}$:

$$\text{rank}\left[\mathbf{f}_2(\mathbf{q}, \dot{\mathbf{q}}, t)\right] < \dim\left[\mathbf{q}\right]. \tag{1.3}$$

Notice that whether or not a control system is underactuated may depend on the state of the system, although for most systems (including all of the systems in this book) underactuation is a global property of the system.

In words, underactuated control systems are those in which the control input cannot accelerate the state of the robot in arbitrary directions. As a consequence, unlike fully-actuated systems, underactuated system cannot be commanded to follow arbitrary trajectories.

EXAMPLE 1.1 Robot Manipulators

Consider the simple robot manipulator illustrated in Figure 1.1. As described in Appendix A, the equations of motion for this system are quite simple to derive, and take the form of the standard "manipulator equations":

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

It is well known that the inertial matrix, $\mathbf{H}(\mathbf{q})$ is (always) uniformly symmetric and positive definite, and is therefore invertible. Putting the system into the form of equation 1.1 yields:

$$\begin{aligned}\ddot{\mathbf{q}} =& \mathbf{H}^{-1}(\mathbf{q})\left[\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\right] \\ &+ \mathbf{H}^{-1}(\mathbf{q})\mathbf{B}(\mathbf{q})\mathbf{u}.\end{aligned}$$

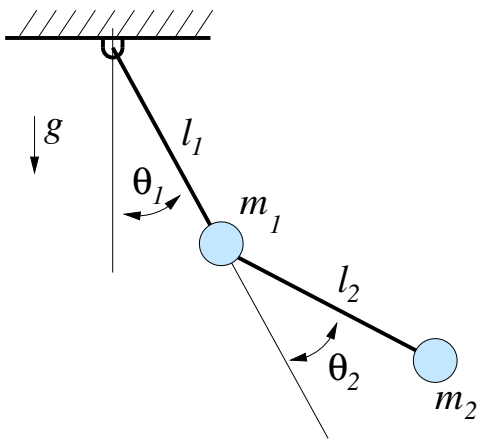


FIGURE 1.1 Simple double pendulum

Because $\mathbf{H}^{-1}(\mathbf{q})$ is always full rank, we find that a system described by the manipulator equations is fully-actuated if and only if $\mathbf{B}(\mathbf{q})$ is full row rank.

For this particular example, $\mathbf{q} = [\theta_1, \theta_2]^T$ and $\mathbf{u} = [\tau_1, \tau_2]^T$, and $\mathbf{B}(\mathbf{q}) = \mathbf{I}_{2\times 2}$. The system is fully actuated. Now 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 exactly the dynamics we will use to study the compass gait model of walking in chapter 5.

The matrix $\mathbf{f}_2$ is equation 1.1 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 $\dim[\mathbf{u}] < \dim[\mathbf{q}]$. 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 that 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). That control system is underactuated.

A quick note about notation. Throughout this class I will try to be consistent in using $\mathbf{q}$ and $\dot{\mathbf{q}}$ for positions and velocities, respectively, and reserve $\mathbf{x}$ for the full state ($\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$). Unless otherwise noted, vectors are always treated as column vectors. Vectors and matrices are bold (scalars are not).

1.3 FEEDBACK LINEARIZATION

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), it is possible to use feedback to effectively change a nonlinear control problem into a linear control problem. The field of linear control is incredibly advanced, and there are many well-known solutions for controlling linear systems.

The trick is called feedback linearization. When $\mathbf{f}_2$ is full row rank, it is invertible. Consider the nonlinear feedback law:

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

where $\mathbf{u}'$ is some additional control input. Applying this feedback controller to equation 1.1 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, [75]).

EXAMPLE 1.2 Feedback-Linearized 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:

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

This is easily achieved[3] using

$$\mathbf{u} = \mathbf{B}^{-1}\left[\mathbf{C}\dot{\mathbf{q}} + \mathbf{G} + \mathbf{H}\begin{bmatrix} -\frac{g}{l}c_1 - b\dot{q}_1 \\ 0 \end{bmatrix}\right].$$

This idea can, and does, make control look 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.

---

[3]Note that our chosen dynamics do not actually stabilize θ_2 - this detail was left out for clarity, but would be necessary for any real implementation.

The underactuated systems are not feedback linearizable. Therefore, unlike fully-actuated systems, the control designer has no choice but to reason about the nonlinear dynamics of the plant in the control design. This dramatically complicates feedback controller design.

1.4 INPUT AND STATE CONSTRAINTS

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

DEFINITION 3 (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 $\phi(\mathbf{x}, \mathbf{u}, t) \leq 0$.

In practice it can useful to separate out constraints which depend only on the input, e.g. $\phi(\mathbf{u}) \leq 0$, 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. $\phi(\mathbf{q}) \leq 0$.

Although distinct from underactuation constraints, these constraints can complicate control design in similar ways (i.e., the robot cannot follow arbitrary trajectories), and often require similar tools to find a control solution.

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 $\mathbf{u} \in \Re^N$ and $\mathbf{q} \in \Re^{N+6}$, equation 1.3 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 w/ 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. Note that the force-closure analogy has an interesting parallel in legged robots.

Even fully-actuated control 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 tools from computer science which be used to design feedback control for underactuated systems. This includes tools from numerical optimal control, motion planning, machine learning. 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 MATLAB, the course notes will provide most of the material and references required for the course.

PART ONE

NONLINEAR DYNAMICS AND CONTROL

CHAPTER 2

The Simple Pendulum

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.

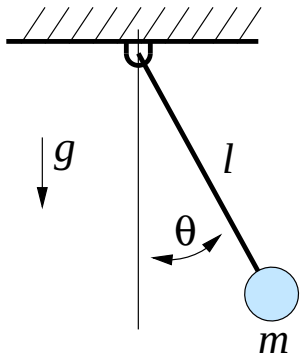


FIGURE 2.1 The Simple Pendulum

The Lagrangian derivation (e.g, [35]) of the equations of motion of the simple pendulum yields:

$$I\ddot{\theta}(t) + mgl\sin\theta(t) = Q,$$

where I is the moment of inertia, and $I = ml^2$ for the simple pendulum. 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 W/ 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:

$$I\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = u_0. \tag{2.1}$$

These are relatively simple equations, so we should be able to integrate them to obtain $\theta(t)$ given $\theta(0), \dot{\theta}(0)$... 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. If what we care about is the long-term behavior of the system, then we can investigate the system using a graphical solution method. These methods are described beautifully in a book by Steve Strogatz[83].

2.2.1 The Overdamped Pendulum

Let's start by studying a special case, when $\frac{b}{I} \gg 1$. This is the case of heavy damping - for instance if the pendulum was moving in molasses. In this case, the b term dominates the acceleration term, and we have:

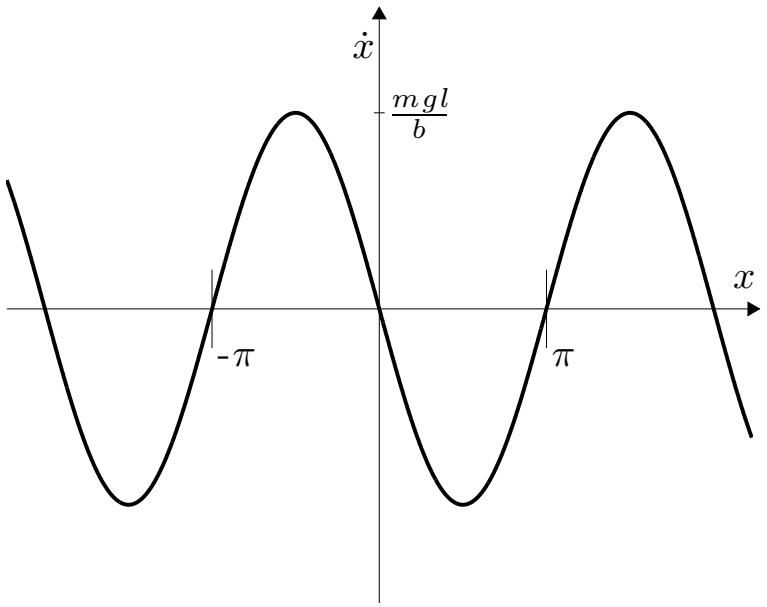
$$u_0 - mgl\sin\theta = I\ddot{\theta} + b\dot{\theta} \approx b\dot{\theta}.$$

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

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

$$b\dot{x} = u_0 - mgl\sin x. \tag{2.2}$$

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 $x^* = \{..., -\pi, 0, \pi, 2\pi, ...\}$. 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. Examing 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 three types of local stability:

- Locally stable in the sense of Lyapunov (i.s.L.). A fixed point, x^* is locally stable i.s.L. if for every small ϵ, I can produce a δ such that if $\|x(0) - x^*\| < \delta$ then $\forall t$ $\|x(t) - x^*\| < \epsilon$. In words, this means that for any ball of size ϵ around the fixed point, I can create a ball of size δ which guarantees that if the system is started inside the δ ball then it will remain inside the ϵ ball for all of time.

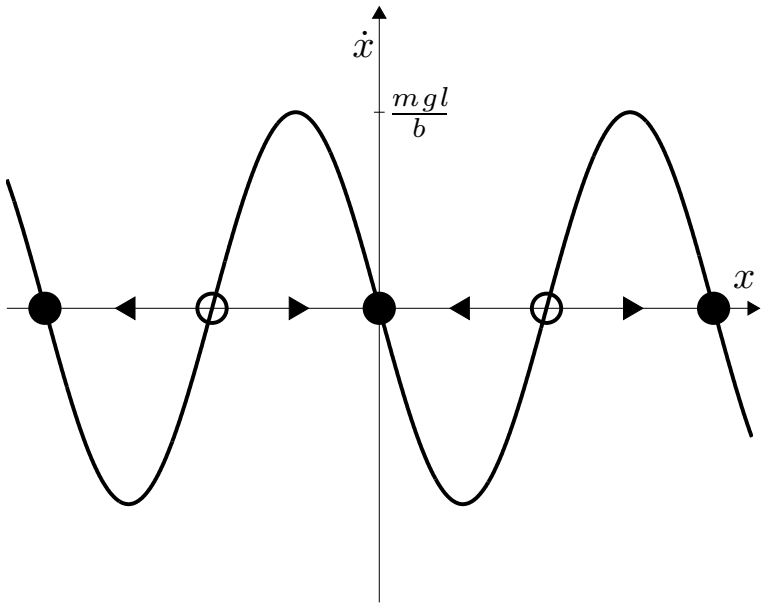
- Locally asymptotically stable. A fixed point is locally asymptotically stable if $x(0) = x^* + \epsilon$ implies that $x(\infty) = x^*$.

- 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 α.

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 $t \rightarrow \infty$, 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, and an asymptotically stable fixed point is also stable i.s.L., but the converse of these is not necessarily true.

Our graph of $\dot{x}$ vs. x can be used to convince ourselves of i.s.L. and asymptotic stability. Exponential stability could potentially be inferred if the function could be bounded by a negatively-sloped line through the fixed point, but this requires some care. 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 $x(\infty) = 0$. 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 $x^* = \{.., -\pi, \pi, 3\pi, ...\}$ 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 2.1 Nonlinear autapse

Consider the following system:

$$\dot{x} + x = \tanh(wx) \tag{2.3}$$

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[71]. 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.

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 figure 2.2.

Our pendulum equations also have a (saddle-node) bifurcation when we change the constant torque input, u_0. This is the subject of exercise 1. Finally, let's return to the

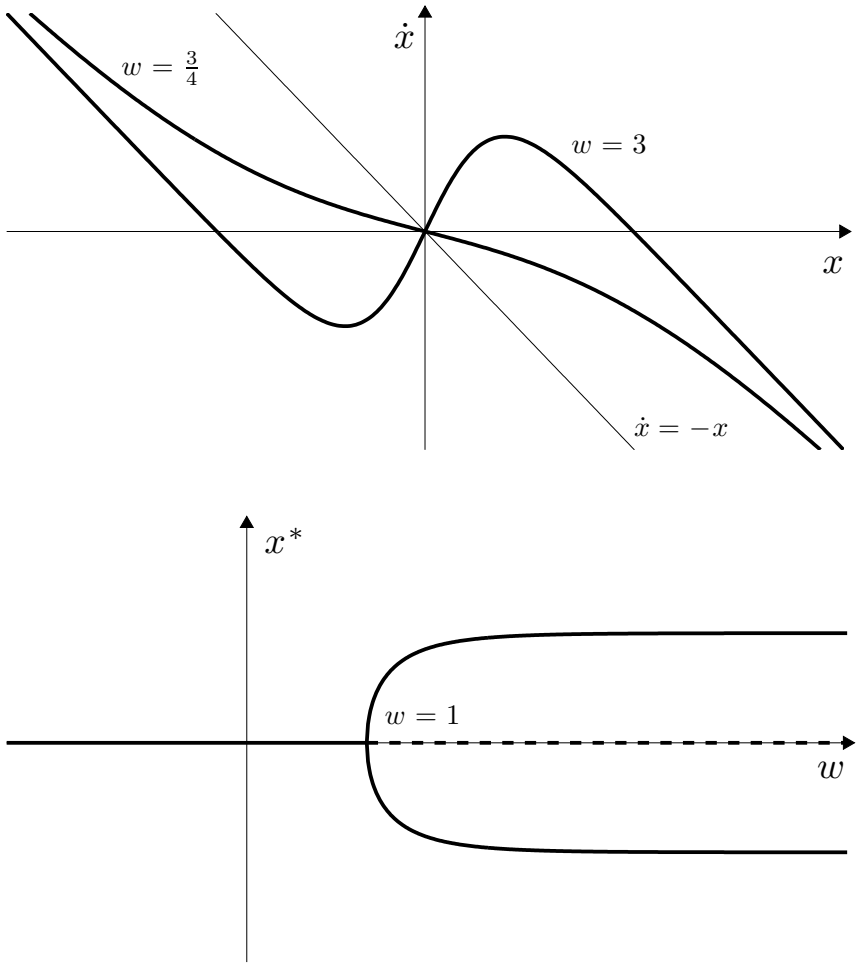


FIGURE 2.2 Bifurcation diagram of the nonlinear autapse.

original equations in θ, instead of in x. Only one point to make: because of the wrap-around, this system will *appear* have oscillations. In fact, the graphical analysis reveals that the pendulum will turn forever whenever $|u_0| > mgl$.

2.2.2 The Undamped Pendulum w/ Zero Torque

Consider again the system

$$I\ddot{\theta} = u_0 - mgl\sin\theta - b\dot{\theta},$$

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,

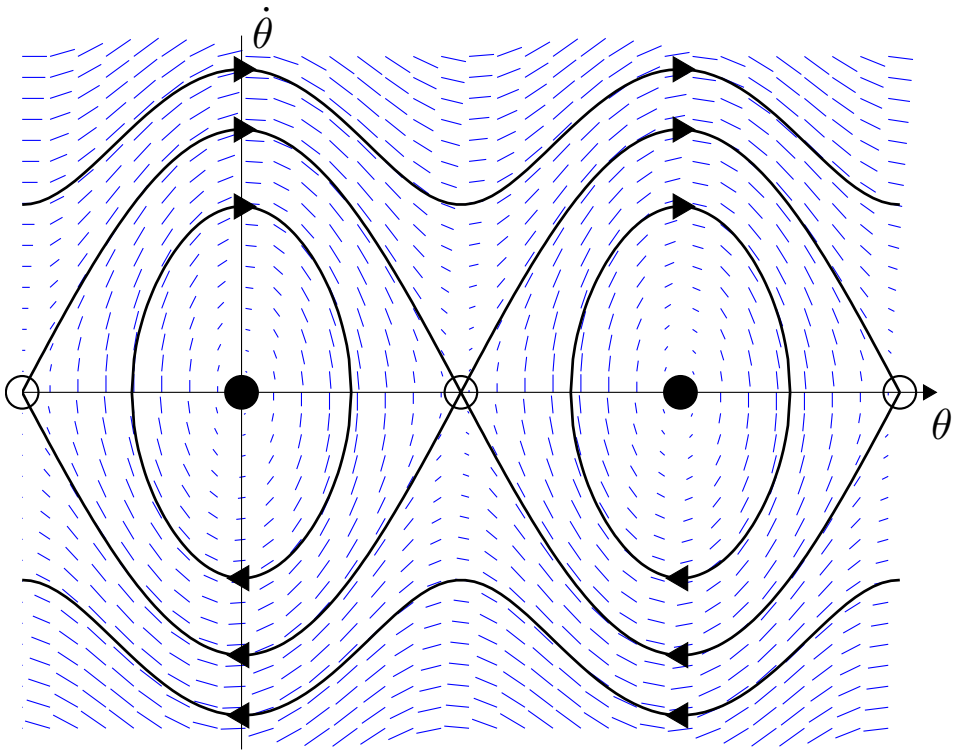
$$\ddot{q} = f(q, \dot{q}, u).$$

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 θ-axis. The x-component of the vector field here is zero, the y-component is $-mgl\sin\theta$. As expected, we have fixed points at $\pm\pi, ...$ 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.

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

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

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:

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

This equation is valid (the squareroot evaluates to a real number) when $\cos(\theta) > \cos(\theta_{max})$, where

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

Furthermore, differentiating this equation with respect to time indeed results in the equations of motion.

Trajectory Calculations.

Solving for $\theta(t)$ is a bit harder, because it cannot be accomplished using elementary functions. We begin the integration with

$$\frac{d\theta}{dt} = \sqrt{\frac{2}{I}\left[E + mgl\cos\left(\theta(t)\right)\right]}$$
$$\int_{\theta(0)}^{\theta(t)} \frac{d\theta}{\sqrt{\frac{2}{I}\left[E + mgl\cos\left(\theta(t)\right)\right]}} = \int_0^t dt' = t$$

The integral on the left side of this equation is an (incomplete) elliptic integral of the first kind. Using the identity:

$$\cos(\theta) = 1 - 2\sin^2(\frac{1}{2}\theta),$$

and manipulating, we have

$$t = \sqrt{\frac{I}{2(E+mgl)}}\int_{\theta(0)}^{\theta(t)} \frac{d\theta}{\sqrt{1 - k_1^2\sin^2(\frac{\theta}{2})}}, \quad \text{with } k_1 = \sqrt{\frac{2mgl}{E+mgl}}.$$

In terms of the incomplete elliptic integral function,

$$F(\phi,k) = \int_0^{\phi} \frac{d\theta}{\sqrt{1 - k^2\sin^2\theta}},$$

accomplished by a change of variables. If $E <= mgl$, which is the case of closed-orbits, we use the following change of variables to ensure $0 < k < 1$:

$$\phi = \sin^{-1}\left[k_1\sin\left(\frac{\theta}{2}\right)\right]$$
$$\cos(\phi)d\phi = \frac{1}{2}k_1\cos\left(\frac{\theta}{2}\right)d\theta = \frac{1}{2}k_1\sqrt{1 - \frac{\sin^2(\phi)}{k_1^2}}d\theta$$

we have

$$t = \frac{1}{k_1}\sqrt{\frac{2I}{(E+mgl)}}\int_{\phi(0)}^{\phi(t)} \frac{d\phi}{\sqrt{1-\sin^2(\phi)}}\frac{\cos(\phi)}{\sqrt{1-\frac{\sin^2\phi}{k_1^2}}}$$
$$= \sqrt{\frac{I}{mgl}}\left[F\left(\phi(t),k_2\right) - F\left(\phi(0),k_2\right)\right], \quad k_2 = \frac{1}{k_1}.$$

The inverse of F is given by the Jacobi elliptic functions (sn,cn,...), yielding:

$$\sin(\phi(t)) = \text{sn}\left(t\sqrt{\frac{mgl}{I}} + F\left(\phi(0),k_2\right),k_2\right)$$
$$\theta(t) = 2\sin^{-1}\left[k_2\text{sn}\left(t\sqrt{\frac{mgl}{I}} + F\left(\phi(0),k_2\right),k_2\right)\right]$$

The function sn used here can be evaluated in MATLAB by calling

$$\text{sn}(u,k) = \texttt{ellipj}(u,k^2).$$

The function F is not implemented in MATLAB, but implementations can be downloaded.. (note that $F(0,k) = 0$).

For the open-orbit case, $E > mgl$, we use

$$\phi = \frac{\theta}{2}, \quad \frac{d\phi}{d\theta} = \frac{1}{2},$$

yielding

$$t = \frac{2I}{E+mgl}\int_{\phi(0)}^{\phi(t)} \frac{d\phi}{\sqrt{1-k_1^2\sin^2(\phi)}}$$
$$\theta(t) = 2\tan^{-1}\left[\frac{\text{sn}\left(t\sqrt{\frac{E+mgl}{2I}} + F\left(\frac{\theta(0)}{2},k_1\right)\right)}{\text{cn}\left(t\sqrt{\frac{E+mgl}{2I}} + F\left(\frac{\theta(0)}{2},k_1\right)\right)}\right]$$

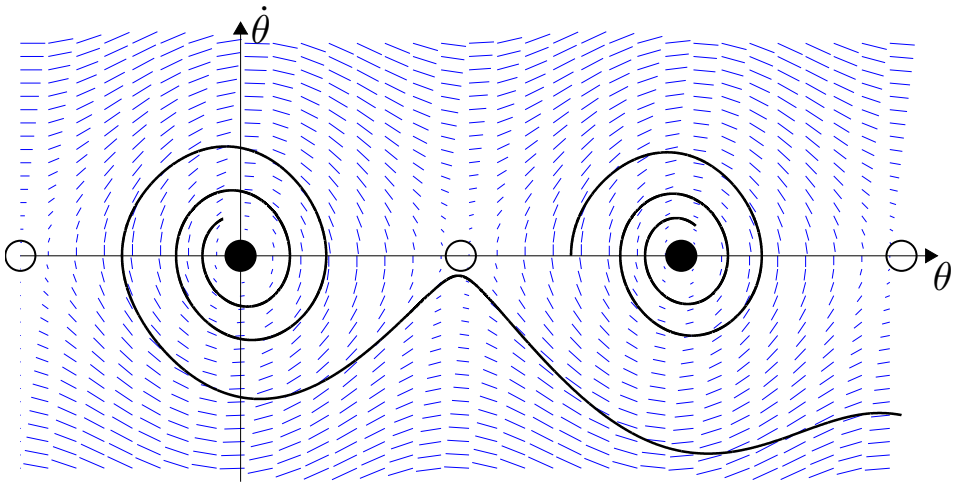
Notes: Use MATLAB's `atan2` and `unwrap` to recover the complete trajectory.

2.2.3 The Undamped Pendulum w/ a Constant Torque

Now what happens if we add a constant torque? Fixed points come together, towards $q = \frac{\pi}{2}, \frac{5\pi}{2}, ...$, until they disappear. Right fixed-point is unstable, left is stable.

2.2.4 The Dampled Pendulum

Add damping back. You can still add torque to move the fixed points (in the same way).



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

$$u = \pi(q, \dot{q}) = 2\frac{g}{l}\cos\theta.$$

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*? We'll learn that soon.

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 law to effectively invert gravity).

The problem begins to get interesting 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. Futhermore, 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 problem is still fully-actuated, its solution requires much of the same reasoning necessary for controller underactuated systems; this problem will be a workhorse for us as we introduce new algorithms throughout this book.

PROBLEMS

2.1. *Bifurcation diagram of the simple pendulum.*

(a) Sketch the bifurcation diagram by varying the continuous torque, u_0, in the *over-damped* simple pendulum described in Equation (2.2) over the range $[-\frac{\pi}{2}, \frac{3\pi}{2}]$. Carefully label the domain of your plot.

(b) Sketch the bifurcation diagram of the underdamped pendulum over the same domain and range as in part (a).

2.2. (CHALLENGE) *The Simple Pendulum ODE.*

The chapter contained the closed-form solution for the undamped pendulum with zero torque.

(a) Find the closed-form solution for the pendulum equations with a constant torque.

(b) Find the closed-form solution for the pendulum equations with damping.

(c) Find the closed-form solution for the pendulum equations with both damping and a constant torque.

CHAPTER 3

The Acrobot and Cart-Pole

3.1 INTRODUCTION

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. These systems are trivially underactuated - both systems have two degrees of freedom, but only a single actuator.

3.2 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 (see Figure 3.1). It was first described in detail in [61]. The companion system, with an actuator at the shoulder but not at the elbow, is known as the Pendubot[76]. The Acrobot is so named because of its resemblence to a gymnist (or acrobat) on a parallel bar, who controls his 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.

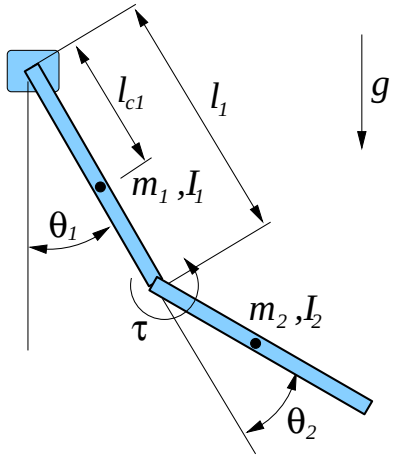


FIGURE 3.1 The Acrobot

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.

3.2.1 Equations of Motion

Figure 3.1 illustrates the model parameters used in our analysis. θ_1 is the shoulder joint angle, θ_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 state is the with both links pointed directly down. The moments of inertia, I_1, I_2 are taken about the pivots[1]. 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 kinematics are given by:

$$\mathbf{x}_1 = \begin{bmatrix} l_1 s_1 \\ -l_1 c_1 \end{bmatrix}, \quad \mathbf{x}_2 = \mathbf{x}_1 + \begin{bmatrix} l_2 s_{1+2} \\ -l_2 c_{1+2} \end{bmatrix}. \tag{3.1}$$

The energy[2] is given by:

$$T = T_1 + T_2, \quad T_1 = \frac{1}{2} I_1 \dot{q}_1^2 \tag{3.2}$$

$$T_2 = \frac{1}{2}(m_2 l_1^2 + I_2 + 2 m_2 l_1 l_{c2} c_2)\dot{q}_1^2 + \frac{1}{2} I_2 \dot{q}_2^2 + (I_2 + m_2 l_1 l_{c2} c_2)\dot{q}_1 \dot{q}_2 \tag{3.3}$$

$$U = -m_1 g l_{c1} c_1 - m_2 g (l_1 c_1 + l_2 c_{1+2}) \tag{3.4}$$

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

$$(I_1 + I_2 + m_2 l_1^2 + 2 m_2 l_1 l_{c2} c_2)\ddot{q}_1 + (I_2 + m_2 l_1 l_{c2} c_2)\ddot{q}_2 - 2 m_2 l_1 l_{c2} s_2 \dot{q}_1 \dot{q}_2 \tag{3.5}$$

$$-m_2 l_1 l_{c2} s_2 \dot{q}_2^2 + (m_1 l_{c1} + m_2 l_1) g s_1 + m_2 g l_2 s_{1+2} = 0 \tag{3.6}$$

$$(I_2 + m_2 l_1 l_{c2} c_2)\ddot{q}_1 + I_2 \ddot{q}_2 + m_2 l_1 l_{c2} s_2 \dot{q}_1^2 + m_2 g l_2 s_{1+2} = \tau \tag{3.7}$$

In standard, manipulator equation form, we have:

$$\mathbf{H}(\mathbf{q}) = \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2 m_2 l_1 l_{c2} c_2 & I_2 + m_2 l_1 l_{c2} c_2 \\ I_2 + m_2 l_1 l_{c2} c_2 & I_2 \end{bmatrix}, \tag{3.8}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -2 m_2 l_1 l_{c2} s_2 \dot{q}_2 & -m_2 l_1 l_{c2} s_2 \dot{q}_2 \\ m_2 l_1 l_{c2} s_2 \dot{q}_1 & 0 \end{bmatrix}, \tag{3.9}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} (m_1 l_{c1} + m_2 l_1) g s_1 + m_2 g l_2 s_{1+2} \\ m_2 g l_2 s_{1+2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3.10}$$

3.3 CART-POLE

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 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.

[1] [77] uses the center of mass, which differs only by an extra term in each inertia from the parallel axis theorem.

[2] The complicated expression for T_2 can be obtained by (temporarily) assuming the mass in link 2 comes from a discrete set of point masses, and using $T_2 = \sum_i m_i \dot{\mathbf{r}}_i^T \dot{\mathbf{r}}_i$, where l_i is the length along the second link of point $\mathbf{r}_i$. Then the expressions $I_2 = \sum_i m_i l_i^2$ and $l_{c2} = \frac{\sum_i m_i l_i}{\sum_i m_i}$, and $c_1 c_{1+2} + s_1 s_{1+2} = c_2$ can be used to simplify.

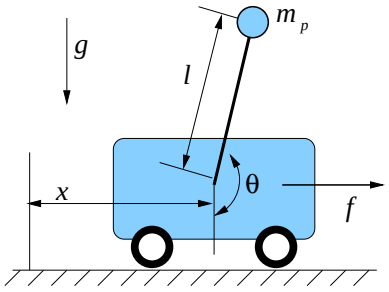


FIGURE 3.2 The Cart-Pole System

Figure 3.2 shows our parameterization of the system. x is the horizontal position of the cart, θ 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 $\mathbf{x} = [0, \pi, 0, 0]^T$.

3.3.1 Equations of Motion

The kinematics of the system are given by

$$\mathbf{x}_1 = \begin{bmatrix} x \\ 0 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} x + l\sin\theta \\ -l\cos\theta \end{bmatrix}. \tag{3.11}$$

The energy is given by

$$T = \frac{1}{2}(m_c + m_p)\dot{x}^2 + m_p\dot{x}\dot{\theta}l\cos\theta + \frac{1}{2}m_p l^2\dot{\theta}^2 \tag{3.12}$$

$$U = -m_p g l\cos\theta. \tag{3.13}$$

The Lagrangian yields the equations of motion:

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2\sin\theta = f \tag{3.14}$$

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p g l\sin\theta = 0 \tag{3.15}$$

In standard form, using $\mathbf{q} = [x, \theta]^T$, $\mathbf{u} = f$:

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

where

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

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

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

$$\ddot{x} = \frac{1}{m_c + m_p\sin^2\theta}\left[f + m_p\sin\theta(l\dot{\theta}^2 + g\cos\theta)\right] \tag{3.16}$$

$$\ddot{\theta} = \frac{1}{l(m_c + m_p\sin^2\theta)}\left[-f\cos\theta - m_p l\dot{\theta}^2\cos\theta\sin\theta - (m_c + m_p)g\sin\theta\right] \tag{3.17}$$

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

$$2\ddot{x} + \ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta = f \tag{3.18}$$

$$\ddot{x}\cos\theta + \ddot{\theta} + \sin\theta = 0. \tag{3.19}$$

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:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \approx \mathbf{f}(\mathbf{x}^*,\mathbf{u}^*) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^*,\mathbf{u}=\mathbf{u}^*}(\mathbf{x}-\mathbf{x}^*) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right]_{\mathbf{x}=\mathbf{x}^*,\mathbf{u}=\mathbf{u}^*}(\mathbf{u}-\mathbf{u}^*) \tag{3.20}$$

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:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{H}^{-1}(\mathbf{q})\left[\mathbf{B}\mathbf{u} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\right] \end{bmatrix}, \tag{3.21}$$

$$\approx \mathbf{A}(\mathbf{x}-\mathbf{x}^*) + \mathbf{B}(\mathbf{u}-\mathbf{u}^*), \tag{3.22}$$

where $\mathbf{A}$, and $\mathbf{B}$ are constant matrices. If you prefer, we can also define $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}^*$, $\bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}^*$, and write

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

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{H}^{-1}\frac{\partial \mathbf{G}}{\partial \mathbf{q}} & -\mathbf{H}^{-1}\mathbf{C} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^*,\mathbf{u}=\mathbf{u}^*} \tag{3.23}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{H}^{-1}\mathbf{B} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^*,\mathbf{u}=\mathbf{u}^*} \tag{3.24}$$

Note that the term involving $\frac{\partial \mathbf{H}^{-1}}{\partial q_i}$ disappears because $\mathbf{B}\mathbf{u} - \mathbf{C}\dot{\mathbf{q}} - \mathbf{G}$ must be zero at the fixed point. Many of the $\mathbf{C}\dot{\mathbf{q}}$ derivatives drop out, too, because $\dot{\mathbf{q}}^* = 0$.

Linearization of the Acrobot.
Linearizing around the (unstable) upright point, we have:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}, \tag{3.25}$$

$$\left[\frac{\partial \mathbf{G}}{\partial \mathbf{q}}\right]_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} -g(m_1 l_{c1} + m_2 l_1 + m_2 l_2) & -m_2 g l_2 \\ -m_2 g l_2 & -m_2 g l_2 \end{bmatrix} \tag{3.26}$$

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

Linearization of the Cart-Pole System.
Linearizing around the (unstable) fixed point in this system, we have:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}, \quad \left[\frac{\partial \mathbf{G}}{\partial \mathbf{q}}\right]_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} 0 & 0 \\ 0 & -m_p g l \end{bmatrix} \tag{3.27}$$

Again, the linear dynamics follow simply.

3.4.2 Controllability of Linear Systems

Consider the linear system

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

where $\mathbf{x}$ has dimension n. A system of this form is called *controllable* if it is possible to construct an unconstrained control signal which will transfer an initial state to any final state in a finite interval of time, $0 < t < t_f$ [65]. If every state is controllable, then the system is said to be completely state controllable. Because we can integrate this linear system in closed form, it is possible to derive the exact conditions of complete state controllability.

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 intution 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 λ_i is the ith 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 yeilding 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[65]. Therefore, we say that the system is controllable if and only if

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

Note 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 problem5, and (for $\mathbf{R} = \mathbf{I}$) is actually a simple function of the controllability Grammian[21].

A general solution.

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

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

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

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

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:

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

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

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

Not surprisingly, this has many special forms. For instance, $e^{\mathbf{A}t} = \mathbf{V}e^{\mathbf{\Lambda}t}\mathbf{V}^{-1}$, where $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ is the eigenvalue decomposition of $\mathbf{A}$ [82]. 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[65, 21]. Then we have,

$$\begin{aligned}\mathbf{x}(0) &= -\sum_{k=0}^{n-1} \mathbf{A}^k \mathbf{B} \int_0^{t_f} \alpha_k(\tau) u(\tau) d\tau \\ &= -\sum_{k=0}^{n-1} \mathbf{A}^k \mathbf{B} w_k, \text{ where } w_k = \int_0^{t_f} \alpha_k(\tau) u(\tau) d\tau \\ &= -\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A^2B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}_{n\times n} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n-1}\end{bmatrix}\end{aligned}$$

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 complete-state 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 columns of the controllability matrix are linearly independent. Therefore, the condition of controllability is that this controllability matrix is full rank.

Although we only treated the case of a scalar u, it is possible to extend the analysis to a vector $\mathbf{u}$ of size m, yielding the condition

$$\text{rank}\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A^2B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}_{n\times(nm)} = n.$$

In Matlab[3], you can obtain the controllability matrix using `Cm = ctrb(A,B)`, and evaluate its rank with `rank(Cm)`.

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 arbitrarly complex.

The controllability analysis presented here is for LTI systems. A comparable analysis exists for linear time-varying (LTV) systems. One would like to find a comparable analysis for controllability that would apply to nonlinear systems, but I do not know of any general tools for solving this problem.

[3] using the control systems toolbox

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 6, one can easily design an optimal feedback controller for a regulation task like balancing, so long as we are willing to define optimality in terms of a quadratic cost function:

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

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 practioners of LQR. Conveniently, Matlab has a function, `K = lqr(A,B,Q,R)`. Therefore, to use LQR, one simply needs to obtain the linearized system dynamics and 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.

Analysis of the close-loop response with LQR feedback shows that the task is indeed completed - and in an impressive manner. Often times 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 have right-half plane zeros - the system in non-minimum phase (acrobot had 3 right-half zeros, cart-pole had 1).

[*To do: Include trajectory example plots here*]

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., [96]), 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 linearizable. At least not completely. Although we cannot linearize 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 effected 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 law 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 *noncollocated* partial feedback linearization - a controller which linearizes the dynamics of the unactuated joints. The treatment presented here follows from [78].

3.5.1 PFL for the Cart-Pole System

Collocated.

Starting from equations 3.18 and 3.19, we have

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

Therefore, applying the feedback control law

$$f = (2 - c^2)\ddot{x}^d - sc - \dot{\theta}^2 s \tag{3.28}$$

results in

$$\ddot{x} = \ddot{x}^d$$
$$\ddot{\theta} = -\ddot{x}^d c - s,$$

which are valid globally.

Non-collocated.

Starting again from equations 3.18 and 3.19, we have

$$\ddot{x} = -\frac{\ddot{\theta} + s}{c}$$
$$\ddot{\theta}(c - \frac{2}{c}) - 2\tan\theta - \dot{\theta}^2 s = f$$

Applying the feedback control law

$$f = (c - \frac{2}{c})\ddot{\theta}^d - 2\tan\theta - \dot{\theta}^2 s \tag{3.29}$$

results in

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

Note that this expression is only valid when $\cos\theta \neq 0$. This is not surprising, as we know that the force cannot create a torque when the beam is perfectly horizontal.

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:

$$\mathbf{H}_{11}\ddot{\mathbf{q}}_1 + \mathbf{H}_{12}\ddot{\mathbf{q}}_2 + \phi_1 = 0, \tag{3.30}$$

$$\mathbf{H}_{21}\ddot{\mathbf{q}}_1 + \mathbf{H}_{22}\ddot{\mathbf{q}}_2 + \phi_2 = \tau, \tag{3.31}$$

with $\mathbf{q} \in \Re^n$, $\mathbf{q}_1 \in \Re^m$, $\mathbf{q}_2 \in \Re^l$, $l = n - m$. $\mathbf{q}_1$ represents all of the passive joints, and $\mathbf{q}_2$ represents all of the actuated joints, and the ϕ terms capture all of the Coriolis and gravitational terms, and

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

Fortunately, because $\mathbf{H}$ is uniformly positive definite, $\mathbf{H}_{11}$ and $\mathbf{H}_{22}$ are also positive definite.

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

$$\ddot{\mathbf{q}}_1 = -\mathbf{H}_{11}^{-1}\left[\mathbf{H}_{12}\ddot{\mathbf{q}}_2 + \phi_1\right] \tag{3.32}$$

$$(\mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12})\ddot{\mathbf{q}}_2 + \phi_{\mathbf{2}} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\phi_1 = \tau \tag{3.33}$$

It can be easily shown that the matrix $(\mathbf{H}_{22} - \mathbf{H}_{21}\mathbf{H}_{11}^{-1}\mathbf{H}_{12})$ is invertible[78]; we can see from inspection that it is symmetric. PFL follows naturally, and is valid globally.

Non-collocated linearization.

$$\ddot{\mathbf{q}}_2 = -\mathbf{H}_{12}^{+}\left[\mathbf{H}_{11}\ddot{\mathbf{q}}_1 + \phi_1\right] \tag{3.34}$$

$$(\mathbf{H}_{21} - \mathbf{H}_{22}\mathbf{H}_{12}^{+}\mathbf{H}_{11})\ddot{\mathbf{q}}_1 + \phi_{\mathbf{2}} - \mathbf{H}_{22}\mathbf{H}_{12}^{+}\phi_1 = \tau \tag{3.35}$$

Where $\mathbf{H}_{12}^{+}$ is a Moore-Penrose pseudo-inverse. This inverse provides a unique solution when the rank of $\mathbf{H}_{12}$ equals l, the number of passive degrees of freedom in the system (it cannot be more, since the matrix only has l rows). This rank condition is sometimes called the property of "Strong Inertial Coupling". It is state dependent. Global Strong Inertial Coupling if every state is coupled.

Task Space 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,

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

with $\mathbf{y} \in \Re^p$, which defines the task space. Define $\mathbf{J}_1 = \frac{\partial \mathbf{f}}{\partial \mathbf{q}_1}$, $\mathbf{J}_2 = \frac{\partial \mathbf{f}}{\partial \mathbf{q}_2}$, $\mathbf{J} = [\mathbf{J}_1, \mathbf{J}_2]$.

THEOREM 4 (Task Space PFL)**.** If the actuated joints are commanded so that

$$\ddot{\mathbf{q}}_2 = \bar{\mathbf{J}}^{+}\left[\mathbf{v} - \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}_1\mathbf{H}_{11}^{-1}\phi_1\right], \tag{3.36}$$

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

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

then we have

$$\ddot{\mathbf{y}} = \mathbf{v}. \tag{3.37}$$

subject to

$$\text{rank}\left(\bar{\mathbf{J}}\right) = p, \tag{3.38}$$

Proof. Differentiating the output function we have

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

Solving 3.30 for the dynamics of the unactuated joints we have:

$$\ddot{\mathbf{q}}_1 = -\mathbf{H}_{11}^{-1}(\mathbf{H}_{12}\ddot{\mathbf{q}}_2 + \phi_1) \tag{3.39}$$

Substituting, we have

$$\ddot{\mathbf{y}} = \dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{J}_1\mathbf{H}_{11}^{-1}(\mathbf{H}_{12}\ddot{\mathbf{q}}_2 + \phi_1) + \mathbf{J}_2\ddot{\mathbf{q}}_2 \tag{3.40}$$
$$= \dot{\mathbf{J}}\dot{\mathbf{q}} + \bar{\mathbf{J}}\ddot{\mathbf{q}}_2 - \mathbf{J}_1\mathbf{H}_{11}^{-1}\phi_1 \tag{3.41}$$
$$= \mathbf{v} \tag{3.42}$$

Note that the last line required the rank condition (3.38) on $\bar{\mathbf{J}}$ to ensure that the rows of $\bar{\mathbf{J}}$ are linearly independent, allowing $\bar{\mathbf{J}}\bar{\mathbf{J}}^+ = \mathbf{I}$. □

In order to execute a task space trajectory one could command

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

Assuming the internal dynamics are stable, this yields converging error dynamics, $(\mathbf{y}_d - \mathbf{y})$, when $\mathbf{K}_p, \mathbf{K}_d > 0$[75]. For a position control robot, the acceleration command of (3.36) suffices. Alternatively, a torque command follows by substituting (3.36) and (3.39) into (3.31).

EXAMPLE 3.1 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:

$$y = f(\mathbf{q}) = -l\cos\theta$$
$$\dot{y} = l\dot{\theta}\sin\theta$$

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.

Collocated and Non-Collocated PFL from Task Space derivation.

The task space derivation above provides a convenient generalization of the partial feedback linearization (PFL) [78], which emcompasses both the collocated and non-collocated results. If we choose $\mathbf{y} = \mathbf{q}_2$ (collocated), then we have

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

From this, the command in (3.36) reduces to $\ddot{\mathbf{q}}_2 = v$. The torque command is then

$$\tau = -\mathbf{H}_{21}\mathbf{H}_{11}^{-1}(\mathbf{H}_{12}v + \phi_1) + \mathbf{H}_{22}v + \phi_2,$$

and the rank condition (3.38) is always met.

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

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

The rank condition (3.38) requires that $\text{rank}(\mathbf{H}_{12}) = l$, in which case we can write $\bar{\mathbf{J}}^+ = -\mathbf{H}_{12}^+\mathbf{H}_{11}$, reducing the rank condition to precisely the "Strong Inertial Coupling" condition described in [78]. Now the command in (3.36) reduces to

$$\ddot{\mathbf{q}}_2 = -\mathbf{H}_{12}^+ \left[\mathbf{H}_{11}\mathbf{v} + \phi_1\right] \tag{3.43}$$

The torque command is found by substituting $\ddot{\mathbf{q}}_1 = v$ and (3.43) into (3.31), yielding the same results as in [78].

3.6 SWING-UP CONTROL

3.6.1 Energy Shaping

Recall the phase portraits that we used to understand the dynamics of the undamped, unactuated, simple pendulum ($u = b = 0$) in section 2.2.2. The orbits of this phase plot were defined by countours of constant energy. One very special orbit, known as a *homoclinic* orbit, is the orbit which passes through the unstable fixed point. In fact, visual inspection will reveal that any state that lies on this homoclinic orbit must pass into the unstable fixed point. Therefore, if we seek to design a nonlinear feedback control policy which drives the simple pendulum from any initial condition to the unstable fixed point, a very reasonable strategy would be to use actuation to regulate the energy of the pendulum to place it on this homoclinic orbit, then allow the system dynamics to carry us to the unstable fixed point.

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[30, 5, 94, 79, 54, 12, 61, 49], the energy shaping controllers tend to be the most natural to derive and perhaps the most well-known.

3.6.2 Simple Pendulum

Recall the equations of motion for the undamped simple pendulum were given by

$$ml^2\ddot{\theta} + mgl\sin\theta = u.$$

The total energy of the simple pendulum is given by

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

To understand how to control the energy, observe that

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

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

To drive the system to the homoclinic orbit, we must regulate the energy of the system to a particular desired energy,

$$E^d = mgl.$$

If we define $\tilde{E} = E - E^d$, then we have

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

If we apply a feedback controller of the form

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

then the resulting error dynamics are

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

These error dynamics imply an exponential convergence:

$$\tilde{E} \to 0,$$

except for states where $\dot{\theta} = 0$. 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? *No.* 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, one trajectories come into the vicinity of our swing-up controller, we prefer to switch to our LQR balancing controller to performance to complete the task.

3.6.3 Cart-Pole

Having thought about the swing-up problem for the simple pendulum, let's try to apply the same ideas to the cart-pole system. The basic idea, from [23], is to use collocated PFL to simplify the dynamics, use energy shaping to regulate the pendulum to it's homoclinic orbit, then to add a few terms to make sure that the cart stays near the origin. The collocated PFL (when all parameters are set to 1) left us with:

$$\ddot{x} = u \tag{3.44}$$

$$\ddot{\theta} = -uc - s \tag{3.45}$$

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 is satisfied for all but the most trivial trajectories.

Now we must return to the full system dynamics (which includes the cart). [23] and [80] use the simple pendulum energy controller with an addition PD controller designed to regulate the cart:

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

[23] provided a proof of convergence for this controller with some nominal parameters.

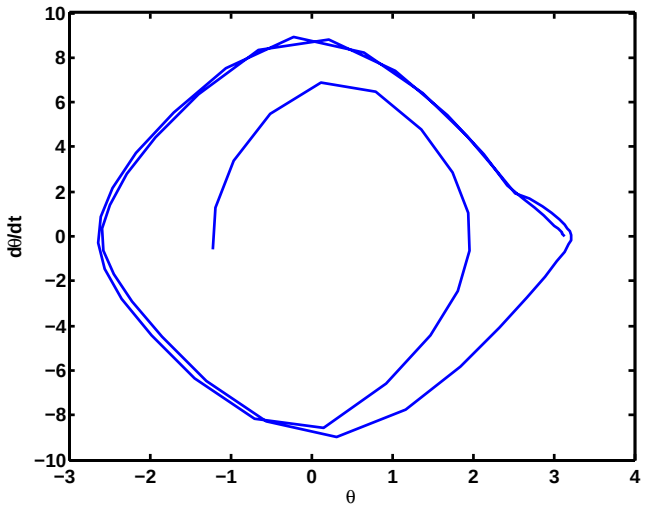


FIGURE 3.3 Cart-Pole Swingup: Example phase plot of the pendulum subsystem using energy shaping control. The controller drives the system to the homoclinic orbit, then switches to an LQR balancing controller near the top.

3.6.4 Acrobot

Swing-up control for the acrobot can be accomplished in much the same way. [79] - pump energy. Clean and simple. No proof. Slightly modified version (uses arctan instead of sat) in [77]. Clearest presentation in [80].

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

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

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

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

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

Extra PD terms prevent proof of asymptotic convergence to homoclinic orbit. Proof of another energy-based controller in [94].

3.6.5 Discussion

The energy shaping controller for swing-up presented here are pretty faithful representatives 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.

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
- Furata pendulum (horizontal rotation and vertical pend)
- Hovercraft
- Planar VTOL

CHAPTER 5

Walking

Practical legged locomotion is one of the fundamental unsolved problems in robotics. Many challenges are in mechanical design - a walking robot must carry all of it's actuators and power, making it difficult to carry ideal force/torque - controlled actuators. But many of the unsolved problems are because walking robots are underactuated control systems.

In this chapter we'll introduce some of the simple models of walking robots, the control problems that result, and a very brief summary of some of the control solutions described in the literature. Compared to the robots that we have studied so far, our investigations of legged locomotion will require additional tools for thinking about limit cycle dynamics and dealing with impacts.

5.1 LIMIT CYCLES

A limit cycle is an asymptotically stable or unstable periodic orbit[1]. One of the simplest models of limit cycle behavior is the Van der Pol oscillator. Let's examine that first...

EXAMPLE 5.1 Van der Pol Oscillator

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

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 seem nicely in the phase portrait in Figure 5.1(left).

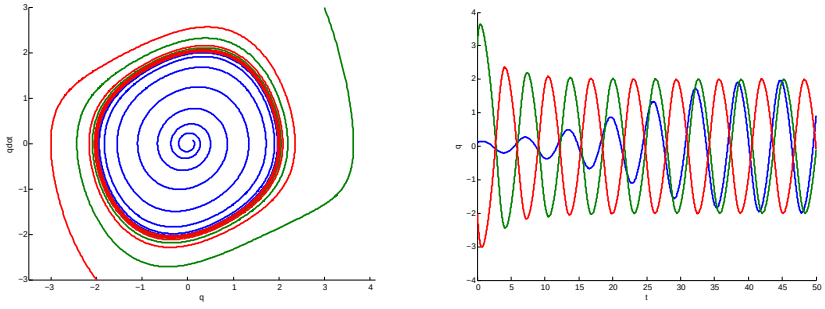


FIGURE 5.1 System trajectories of the Van der Pol oscillator with $\mu = .2$. (Left) phase portrait. (Right) time domain.

[1] marginally-stable orbits, such as the closed-orbits of the undamped simple pendulum, are typically not called limit cycles.

However, if we plot system trajectories in the time domain, then a slightly different picture emerges (see Figure 5.1(right)). 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.

The Van der Pol oscillator clearly demonstrates what we would think of as a stable limit cycle, but also exposes the subtlety in defining this limit cycle stability. Neighboring trajectories do not necessarily converge on a stable limit cycle. In contrast, defining the stability of a particular trajectory (parameterized by time) is relatively easy.

Let's make a few quick points about the existence of closed-orbits. If we can define a closed region of phase space which does not contain any fixed points, then it must contain a closed-orbit[83]. By closed, I mean that any trajectory which enters the region will stay in the region (this is the Poincare-Bendixson Theorem). It's also interesting to note that gradient potential fields (e.g. Lyapunov functions) cannot have a closed-orbit[83], and consquently Lyapunov analysis cannot be applied to limit cycle stability without some modification.

5.2 POINCARÉ MAPS

One definition for the stability of a limit cycle uses the method of Poincaré. Let's consider an n dimensional dynamical system, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. Define an $n-1$ dimensional *surface of section*, S. We will also require that S is tranverse 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 nth 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 nth crossing of S.

EXAMPLE 5.2 Return map for the Van der Pol Oscillator

Since the full system is two dimensional, the return map dynamics are one dimensional. One dimensional maps, like one dimensional flows, are amenable to graphical analysis. To define a Poincare section for the Van der Pol oscillator, let S be the line segment where $\dot{q} = 0, q > 0$.

If $\mathbf{P}(\mathbf{x}_p)$ exists for all $\mathbf{x}_p$, then this method turns the stability analysis for a limit cycle into the stability analysis of a fixed point on a discrete map. In practice it is often difficult or impossible to find $\mathbf{P}$ analytically, but it can be obtained quite reasonably numerically. Once $\mathbf{P}$ is obtained, we can infer local limit cycle stability with an eigenvalue analysis. There will always be a single eigenvalue of 1 - 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, λ_i, have magnitude less than one, $|\lambda_i| < 1$.

In fact, it is often possible to infer more global stability properties of the return map by examining, $\mathbf{P}$. [44] describes some of the stability properties known for *unimodal* maps.

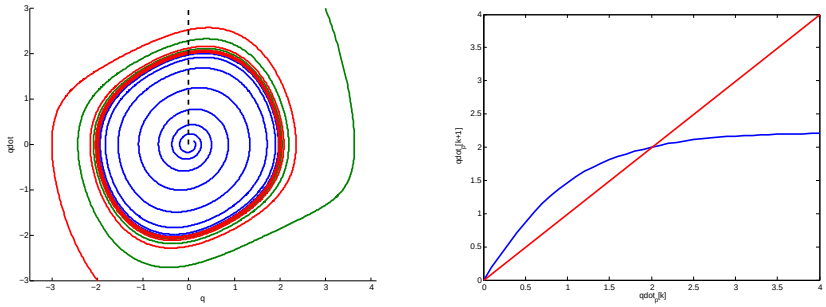


FIGURE 5.2 (Left) Phase plot with the surface of section, S drawn with a black dashed line. (Right) The resulting Poincare first-return map (blue), and the line of slope one (red).

A particularly graphical method for understanding the dynamics of a one-dimensional iterated map is with the staircase method. Sketch the Poincare map and also the line of slope one. Fixed points are the crossings with the unity line. Asymptotically stable if $|\lambda| < 1$. Unlike one dimensional flows, one dimensional maps can have oscillations (happens whenever $\lambda < 0$).

[insert staircase diagram of van der Pol oscillator return map here]

5.3 THE BALLISTIC WALKER

One of the earliest models of walking was proposed by McMahon[59], who argued that humans use a mostly ballistic (passive) gait. COM trajectory looks like a pendulum (roughly walking by vaulting). EMG activity in stance legs is high, but EMG in swing leg is very low, except for very beginning and end of swing. Proposed a three-link "ballistic walker" model, which models a single swing phase (but not transitions to the next swing nor stability). Interestingly, in recent years the field has developed a considerably deeper appreciation for the role of compliance during walking; simple walking-by-vaulting models are starting to fall out of favor.

McGeer[55] followed up with a series of walking machines, called "passive dynamic walkers". The walking machine by Collins and Ruina[26] is the most impressive passive walker to date.

5.4 THE RIMLESS WHEEL

The most elementary model of passive dynamic walking, first used in the context of walking by [55], is the rimless wheel. This simplified system has rigid legs and only a point mass at the hip as illustrated in Figure 5.3. 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).

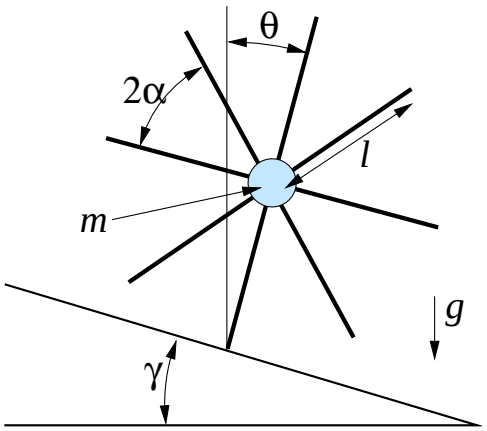


FIGURE 5.3 The rimless wheel. The orientation of the stance leg, θ, is measured clockwise from the vertical axis.

- $0 \leq \gamma < \frac{\pi}{2}, 0 < \alpha < \frac{\pi}{2}, l > 0.$

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 θ has changed).

The most comprehensive analysis of the rimless wheel was done by [24].

5.4.1 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

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

ω_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 $\gamma > \alpha$. 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.

5.4.2 Foot Collision

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:

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

The deterministic dynamics of the rimless wheel produce a stable limit cycle solution with a continuous phase punctuated by a discrete collision, as shown in Figure 5.4. The red dot on this graph represents the initial conditions, and this limit cycle actually moves counter-clockwise in phase space because for this trial the velocities were always negative. The collision represents as instantaneous change of velocity, and a transfer of the coordinate system to the new point of contact.

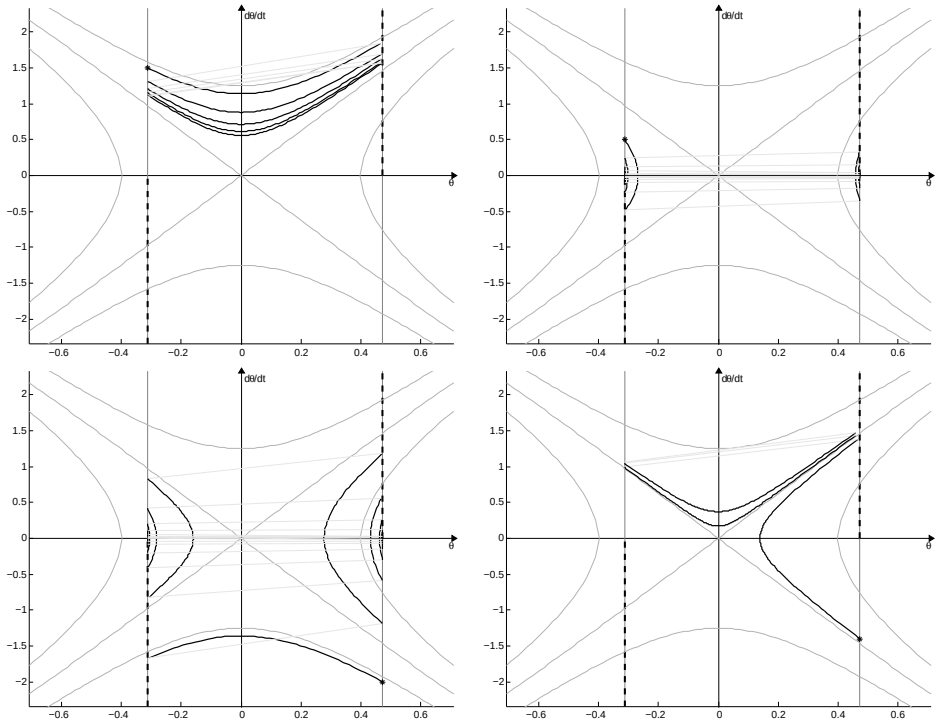


FIGURE 5.4 Phase portrait trajectories of the rimless wheel ($m = 1, l = 1, g = 9.8, \alpha = \pi/8, \gamma = 0.08$).

5.4.3 Return 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

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

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.

5.4.4 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 in Figure 5.4 illustrates a state-space trajectory in the vicinity of the rolling fixed point. ω^*_{stand} is a fixed point whenever $\gamma < \alpha$. ω^*_{roll} is a fixed point whenever $\omega^*_{roll} > \omega_1$. It is interesting to view these bifurcations in terms of γ. For small γ, ω_{stand} is the only fixed point, because energy lost from collisions with the ground is not compensated for by gravity. As we increase γ, 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 γ 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.

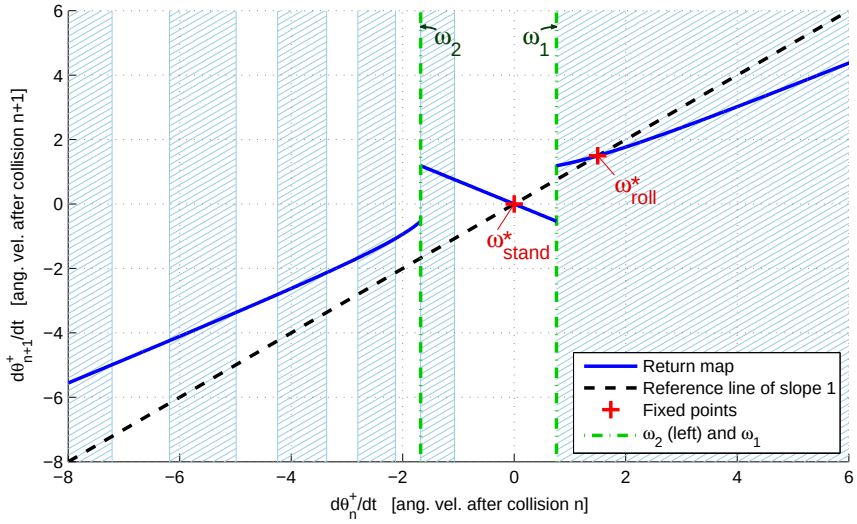


FIGURE 5.5 Limit cycle trajectory of the rimless wheel ($m = 1, l = 1, g = 9.8, \alpha = \pi/8, \gamma = 0.15$). All hatched regions converge to the rolling fixed point, ω^*_{roll}; the white regions converge to zero velocity (ω^*_{stand}).

5.5 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. For actuation, we first consider the case where there is a torque source at the hip - resulting in swing dynamics equivalent to an Acrobot (although in a different coordinate frame).

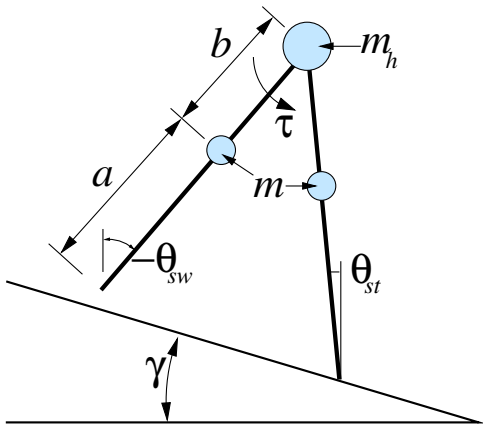


FIGURE 5.6 The compass gait

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 [36, 81], 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 $\mathbf{q} = [\theta_{sw}, \theta_{st}]^T$ and $\mathbf{u} = \tau$, we can write the dynamics as

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

with

$$\mathbf{H} = \begin{bmatrix} mb^2 & -mlb\cos(\theta_{st} - \theta_{sw}) \\ -mlb\cos(\theta_{st} - \theta_{sw}) & (m_h + m)l^2 + ma^2 \end{bmatrix}$$
$$\mathbf{C} = \begin{bmatrix} 0 & mlb\sin(\theta_{st} - \theta_{sw})\dot{\theta}_{st} \\ mlb\sin(\theta_{st} - \theta_{sw})\dot{\theta}_{sw} & 0 \end{bmatrix}$$
$$\mathbf{G} = \begin{bmatrix} mbg\sin(\theta_{sw}) \\ -(m_h l + ma + ml)g\sin(\theta_{st}) \end{bmatrix},$$
$$\mathbf{B} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

and $l = a + b$. These equations come straight out of [37].

The foot collision is an instantaneous change of velocity governed by the conservation of angular momentum around the point of impact:

$$\mathbf{Q}^+(\alpha)\dot{\mathbf{q}}^+ = \mathbf{Q}^-(\alpha)\dot{\mathbf{q}}^-,$$

where

$$\mathbf{Q}^-(\alpha) = \begin{bmatrix} -mab & -mab + (m_h l^2 + 2mal)\cos(2\alpha) \\ 0 & -mab \end{bmatrix}$$
$$\mathbf{Q}^+(\alpha) = \begin{bmatrix} mb(b - l\cos(2\alpha)) & ml(l - b\cos(2\alpha) + ma^2 + m_h l^2 \\ mb^2 & -mbl\cos(2\alpha) \end{bmatrix}$$

and $\alpha = \frac{\theta_{sw} - \theta_{st}}{2}$.

Numerical integration of these equations reveals a stable limit cycle, plotted in Figure 5.7. The cycle is composed of a swing phase (top) and a stance phase (bottom), punctuated by two instantaneous changes in velocity which correspond to the ground collisions. The dependence of this limit cycle on the system parameters has been studied extensively in [37].

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 analytically attractive, this lack of stability makes it difficult to implement on a physical device.

5.6 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 knee[41]. For this, we model each leg as two links with a point mass each.

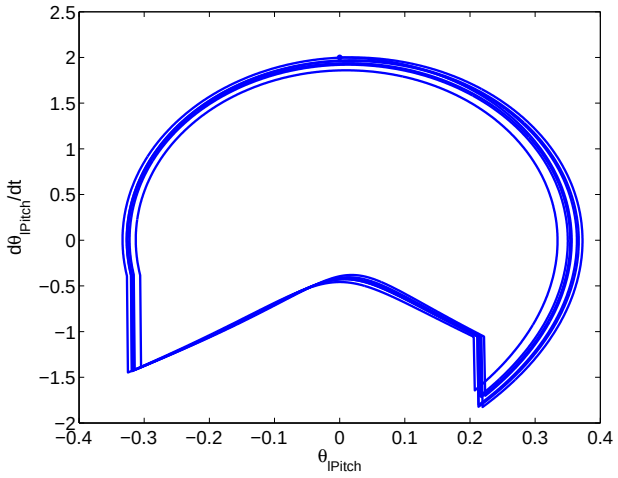


FIGURE 5.7 Limit cycle trajectory of the compass gait. ($m = 5$kg,$m_h = 10$kg,$a = b = 0.5$m,$\phi = 0.03$deg. $\mathbf{x}(0) = [0, 0, 2, -0.4]^T$). θ_{lPitch} is the pitch angle of the left leg, which is recovered from θ_{st} and θ_{sw} in the simulation with some simple book-keeping.

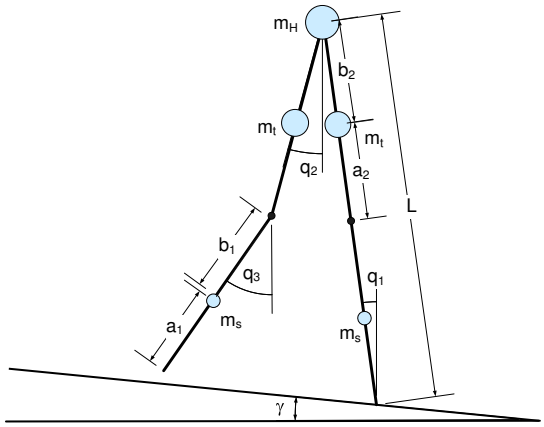


FIGURE 5.8 The Kneed Walker

At the beginning of each step, the system is modeled as a three-link pendulum, like the ballistic walker[59, 58, 81]. 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 in Figure 5.9.

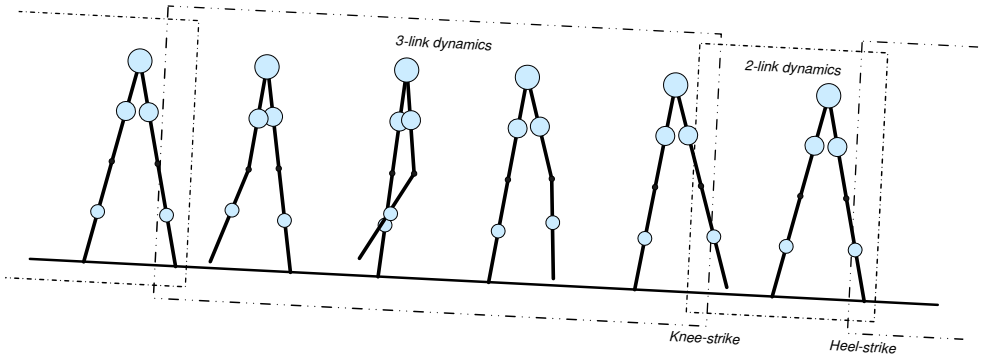


FIGURE 5.9 Limit cycle trajectory for kneed walker

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 Figure 5.10. This limit cycle is traversed clockwise.

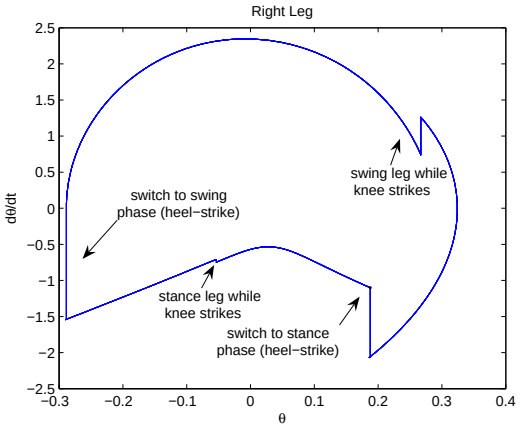


FIGURE 5.10 The Kneed Walker

5.7 NUMERICAL ANALYSIS

A note on integrating hybrid systems and/or evaluating return maps. The dynamics are often very sensitive to the switching plane. Often a good idea to back up integration to attempt to find collisions and transitions very accurately. MATLAB can handle this nicely with zero-crossings in the ODE solvers.

5.7.1 Finding Limit Cycles

You can find asymptotically stable limit cycles by integrating forward the ODE (until $t \rightarrow \infty$), as long as you can guess an initial condition inside the basin of attraction. This convergence rate will depend on the convergence rate around the fixed point and could be inefficient for complex systems, and guessing initial conditions is difficult for systems like the compass gait. This method won't work for finding unstable limit cycles.

Remember that a fixed point on the Poincare map:

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

is simply a zero-crossing of the (vector-valued) function

$$\mathbf{P}(\mathbf{x}_p[n]) - \mathbf{x}_p[n+1].$$

Therefore, an efficient way to obtain the fixed points numerically is to use an algorithm which finds the zero-crossings by a Newton method[16, 67][2]. These methods can be dramatically more efficient if one can efficiently estimate the gradient of the Poincare map.

Gradients of the Poincare map.
The Poincare map is defined by integrating the continuous dynamics,

$$\mathbf{x}(t_c^-[n+1]) = \mathbf{x}(t_c^+[n]) + \int_{t_c^+[n]}^{t_c^-[n+1]} \mathbf{f}(\mathbf{x}(t))dt, \quad \mathbf{x}(t_c^+[n]) = \mathbf{x}_p[n]$$

then applying the (discrete) impact dynamics

$$\mathbf{x}_p[n+1] = \mathbf{x}(t_c^+[n+1]) = \mathbf{F}(\mathbf{x}(t_c^-[n+1])),$$

where $t_c[k]$ is the time of the kth collision, and t_c^- indicates just prior to the collision and t_c^+ is just after the collision. In order to estimate the gradients of the Poincare map, $\frac{d\mathbf{x}_p[n+1]}{d\mathbf{x}_p[n]}$, we must take a little care in handling the effects of initial conditions on the time (and therefore the state) of collision (using the chain rule). Linearizing about a trajectory $\mathbf{x}_0(t), \mathbf{x}_{p_0}[n]$ with impacts at $t_{c_0}[n]$, we have:

$$\frac{d\mathbf{x}_p[n+1]}{d\mathbf{x}_p[n]} = \left[\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}}\left[\frac{\partial \mathbf{x}(t)}{\partial \mathbf{x}_p[n]} + \mathbf{f}(\mathbf{x})\frac{dt_c[n+1]}{d\mathbf{x}_p[n]}\right]\right]_{t=t_{c_0}^-[n+1], \mathbf{x}=\mathbf{x}_0(t_{c_0}^-[n+1])}$$

The switching dynamics are defined by the zeros of scalar collision function, $\phi(t, \mathbf{x})$. Although these dynamics are time-varying in general (e.g., moving obstacles), for the rimless

[2] I hand it to SNOPT as a constraint.

wheel the collision function can be as simple as $\phi(\mathbf{x}) = \text{sgn}(\dot{\theta})(\theta - \gamma) - \alpha$. This collision function allows us to compute $\frac{dt_c[n+1]}{d\mathbf{x}_p[n]}$:

$$\frac{d\phi(t_c[n+1], \mathbf{x}(t_c^-[n+1]))}{d\mathbf{x}_p[n]} = 0 = \left[\frac{\partial \phi(t,\mathbf{x})}{\partial \mathbf{x}}\left[\frac{d\mathbf{x}(t)}{d\mathbf{x}_p[n]} + \frac{\partial \mathbf{x}(t)}{\partial t}\frac{dt_c[n+1]}{d\mathbf{x}_p[n]}\right] + \frac{\partial \phi(t,\mathbf{x})}{\partial t}\frac{dt_c[n+1]}{d\mathbf{x}_p[n]}\right]_{t=t_{c_0}^-[n+1], \mathbf{x}=\mathbf{x}(t_{c_0}^-[n+1])}$$

$$\frac{dt_c[n+1]}{d\mathbf{x}_p[n]} = -\left[\frac{\frac{\partial \phi(t,\mathbf{x})}{\partial \mathbf{x}}\frac{\partial \mathbf{x}(t)}{\partial \mathbf{x}_p[n]}}{\frac{\partial \phi(t,\mathbf{x})}{\partial t} + \frac{\partial \phi(t,\mathbf{x})}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x})}\right]_{t=t_{c_0}^-[n+1], \mathbf{x}=\mathbf{x}(t_{c_0}^-[n+1])}$$

The final step, computing $\left[\frac{\partial \mathbf{x}(t)}{\partial \mathbf{x}_p[n]}\right]_{t=t_{c_0}^-[n+1]}$, can be done with a standard gradient calculation - see section 12.3.2 for the update.

EXAMPLE 5.3 Fixed points of the rimless wheel

Bifurcation diagram (as a function of slope) from computational search. Show that it agrees with analytical solution.

EXAMPLE 5.4 Fixed points of the compass gait

Bifurcation diagram (as a function of slope) - computational only.

5.7.2 Local Stability of Limit Cycle

In practice, the local stability analysis of a limit cycle is done by taking the derivatives around the fixed point of the return map. Again, this is often accomplished using numerical derivatives. Perturb the system in one direction at a time, evaluate the map and build the matrix ... From Goswami [37]. The eigenvalues of the derivative matrix of the Poincaré map, λ_i are called the *characteristic* or *Floquet* multipliers[83].

$$\mathbf{x}^* + \delta_1 = \mathbf{P}(\mathbf{x}^* + \delta_0) \approx \mathbf{P}(\mathbf{x}^*) + \left[\frac{\partial \mathbf{P}}{\partial \mathbf{x}}\right]_{\mathbf{x}^*} \delta_0.$$

$$\delta_1 \approx \left[\frac{\partial \mathbf{P}}{\partial \mathbf{x}}\right]_{\mathbf{x}^*} \delta_0.$$

A fixed point is stable if the $n-1$ non-trivial eigenvalues of this matrix are $|\lambda_i| < 1$.

Trivial multipliers vs. Non-trivial multipliers. Expect one *trivial* multiplier of 0, or 1 (which reveal the dynamics of a perturbation *along* the limit cycle orbit).

A standard numerical recipe for estimating $\frac{\partial \mathbf{P}}{\partial \mathbf{x}}$ is to perturb the system by a very small amount at least n times, once in each of the state variables, and watching the response. Be careful - your perturbation should be big enough to not get into integration errors, but small enough that it stays in the "linear regime". A good way to verify your results is to perturb the system in other directions, and other magnitudes, in an attempt to recover the same eigenvalues. In general, the matrix $\frac{\partial \mathbf{P}}{\partial \mathbf{x}}$ can be reconstructed from any

number of sampled trajectories by solving the equation

$$\begin{bmatrix} | & | & & | \\ \delta_1^1 & \delta_1^2 & \cdots & \delta_1^m \\ | & | & & | \end{bmatrix} = \left[\frac{\partial \mathbf{P}}{\partial \mathbf{x}}\right]_{\mathbf{x}^*} \begin{bmatrix} | & | & & | \\ \delta_0^1 & \delta_0^2 & \cdots & \delta_0^m \\ | & | & & | \end{bmatrix}$$

in a least-squares sense, where δ_0^i is the i-th perturbation (not a perturbation raised to a power!).

Lyapunov exponent. There is at least one quantifier of limit cycle (or trajectories, in general) stability that does not depend on the return map. Like a contraction mapping - perturb original trajectory in each direction, bound recovery by some exponential[83].

$$\|\delta(t)\| < \|\delta(0)e^{\mathbf{A}t}\|.$$

The eigenvalues of A are the Lyapunov exponents. Note that for a stable limit cycle, the largest Lyapunov exponent will be 1 (like the trivial floquet multiplier), and it is the remaining exponents that we will use to evaluate stability.

PROBLEMS

5.1. (CHALLENGE) *Closed-form solution for the rimless wheel.*

We have a closed-form expression for the return map, but can you find a solution for the entire return map dynamics? Given $\dot{\theta}[0]$, directly compute $\dot{\theta}[n]$ for any n. This would involve solving the quadratic difference equation.

5.2. (CHALLENGE) *Deadbeat control of the compass gait.*

Find a hip torque policy that produces a (locally) deadbeat controller.

PART TWO

OPTIMAL CONTROL AND MOTION PLANNING

CHAPTER 9

Dynamic Programming

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 linearization solutions (e.g., invert gravity) can work well, but can also require an unecessarily large amount of control effort. The energy-based swing-up control solutions presented in section 3.6.2 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.

9.1 INTRODUCTION TO OPTIMAL CONTROL

In this chapter, we will introduce optimal control - a control design process using optimization theory. 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 of all, this control formulation permits numerical solutions - we can harness the power of computational optimization theory to solve analytically intractable control problems. [13] is a fantastic reference on this material, and the source of a number of the examples worked out here.

The fundamental idea in optimal control is to formulate the goal of control as the *long-term* optimization of a scalar cost function. 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.

In this chapter, we will formulate the deterministic optimal feedback control problem, considering a system described by the equations:

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

This structure certainly emcompasses the robots that are the focus of this book (and many other dynamical systems), but also doesn't take advantage of the structure of the manipulator equations (that topic will be discussed in chapter ?). In the general formulation, our goal is to design a feedback policy, π, where

$$\mathbf{u}(t) = \pi(\mathbf{x}, t).$$

We will define π^*, the *optimal* feedback policy, as the policy which minimizes the scalar cost function.

9.2 FINITE HORIZON PROBLEMS

Some control problems have a natural duration. For example, perhaps it is 1pm and your robot needs to get to the bank before it closes at 5pm. Or imagine your robot is working on an assembly room floor, and is allocated a fixed amount of time to complete each assembly (or pick-and-place) task. As we will see later in the chapter, other problems are more naturally described independent of the time interval, or over an infinite horizon, and we will see that the infinite horizon derivations follow naturally from the finite horizon derivations in this section.

In finite horizon problems, we define a finite time interval, $t \in [0, T]$, where T is called the horizon time. In general, T can be given, or left as a free-variable to be optimized. If we run the system from an initial condition, $\mathbf{x}_0$, executing policy π then the system will move through a trajectory $[\mathbf{x}(t), \mathbf{u}(t)]$. Using $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$ to denote the entire continuous system trajectory over $[0, T]$, or

$$\bar{\mathbf{x}} = \{\mathbf{x}(0), ..., \mathbf{x}(T)\}, \quad \bar{\mathbf{u}} = \{\mathbf{u}(0), ..., \mathbf{u}(T)\},$$

then we can score the policy π by defining a cost function:

$$J^\pi(\mathbf{x}_0) = g(\bar{\mathbf{x}}, \bar{\mathbf{u}}), \quad \mathbf{x}(0) = \mathbf{x}_0, \mathbf{u}(t) = \pi(\mathbf{x}(t), t).$$

For general cost functions of this form, finding the optimal policy π for all $\mathbf{x}_0$ reduces to solving a (potentially very complex) nonlinear programming problem.

9.2.1 Additive Cost

It turns out that the optimization problem often becomes a lot more tractable (both analytically and computationally) if we are willing to design our cost functions with an additive form. In particular, let us restrict our costs to take the form:

$$J^\pi(\mathbf{x}_0) = h(\mathbf{x}(T)) + \int_0^T g(\mathbf{x}(t), \mathbf{u}(t), t)dt, \quad \mathbf{x}(0) = \mathbf{x}_0.$$

This form is actually quite general. Before we get into the mechanics of solving for the optimal policies, let's get a little intuition by considering some example problem formulations, and intuition about their solutions.

EXAMPLE 9.1 Minimum-time problems

Recall the simple pendulum...

EXAMPLE 9.2 Quadratic regulators

More examples here...

9.3 DYNAMIC PROGRAMMING IN DISCRETE TIME

Now we will begin to develop the machinery for finding and/or verifying an optimal policy. This machinery can be developed more easily in discrete-time, so let's temporarily consider

optimal control problems of the form:

$$\mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n], n) \tag{9.1}$$

$$J^\pi(\mathbf{x}_0) = h(\mathbf{x}[N]) + \sum_{n=0}^{N-1} g(\mathbf{x}[n], \mathbf{u}[n], n), \quad \mathbf{x}[0] = \mathbf{x}_0, \mathbf{u}[n] = \pi(\mathbf{x}[n], n). \tag{9.2}$$

Thanks to the additive form of this long-term cost, we can now start systematically solving for an optimal policy. Let's consider a subproblem, in which we start from $\mathbf{x}_m$ and time m and then continue on until N. Let us define the "cost-to-go" given that we are following policy π as $J^\pi(\mathbf{x}_m, m)$; it is given by

$$J^\pi(\mathbf{x}_m, m) = h(\mathbf{x}[N]) + \sum_{n=m}^{N-1} g(\mathbf{x}[n], \mathbf{u}[n]), \quad \mathbf{x}[m] = \mathbf{x}_m, \mathbf{u}[n] = \pi(\mathbf{x}, n).$$

Our original $J^\pi(\mathbf{x}_0)$ is simply the cost-to-go $J^\pi(\mathbf{x}_0, 0)$. For this reason, J^π is known as the *cost-to-go function* or the *value function*. The cost-to-go function can be written recursively:

$$\begin{aligned} J^\pi(\mathbf{x}, N) =& h(\mathbf{x}) \\ J^\pi(\mathbf{x}, N-1) =& g(\mathbf{x}[N-1], \mathbf{u}[N-1], N-1) + h(\mathbf{x}[N]) \\ &\vdots \\ J^\pi(\mathbf{x}, n) =& g(\mathbf{x}, \mathbf{u}, n) + J^\pi(\mathbf{x}[n+1], n+1) \end{aligned}$$

Even more important, the *optimal* cost-to-go function, J^* (simply defined as J^π for π^*), has almost the same recursive structure:

$$\begin{aligned} J^*(\mathbf{x}, N) =& h(\mathbf{x}) \\ J^*(\mathbf{x}, N-1) =& \min_{\mathbf{u}} \left[g(\mathbf{x}[N-1], \mathbf{u}, N-1) + h(\mathbf{x}[N]) \right] \\ &\vdots \\ J^*(\mathbf{x}, n) =& \min_{u} \left[g(\mathbf{x}, \mathbf{u}, n) + J^*(\mathbf{x}[n+1], n+1) \right]. \end{aligned}$$

Furthermore, the optimal policy is given by:

$$\pi^*(\mathbf{x}, n) = \operatorname*{argmin}_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}, n) + J^*(\mathbf{x}[n+1], n+1) \right].$$

This recursion, which starts at the terminal time and works backwards towards time 0, is known as *dynamic programming*. It is based on the key observation that for systems with a scalar additive cost, the cost-to-go function represents everything that the system needs to know about the future. The optimal action is simply the action which minimizes the sum of the one-step cost and the future optimal cost-to-go.

9.3.1 Discrete-State, Discrete-Action

For systems with continuous states and continuous actions, dynamic programming is a mathematical recipe for deriving the optimal policy and cost-to-go. For systems with a

Algorithm 1: The Dynamic Programming Algorithm.

Input: states S, actions A, horizon time N, dynamics $s' = f(s, a, n)$, instantaneous cost $g(s, a, n)$, and terminal cost $h(s)$

Output: optimal policy $\pi^*(s, n)$ and optimal cost-to-go $J^*(s, n)$

foreach *state* $s \in S$ **do**
  $J^*(s, N) \leftarrow h(s)$;
end
for $n \leftarrow N - 1$ **to** 0 **do**
  foreach *state* s **do**
    $J^*(s, n) \leftarrow \min_{a \in A} [g(s, a, n) + J^*(f(s, a, n), n + 1)]$;
    $\pi^*(s, n) \leftarrow \text{argmin}_{a \in A} [g(s, a, n) + J^*(f(s, a, n), n + 1)]$;
  end
end

finite, discrete set of states and a finite, discrete set of actions, dynamic programming also represents a very efficient *algorithm* which can compute optimal policies (see Algorithm 1). This algorithm uses the states $\{s_1, s_2, ...\} \in S$, where S is the finite state space, and actions as $\{a_1, a_2, ...\} \in A$, where A is the finite action space.

Let's experiment with dynamic programming in a discrete-state, discrete-action, discrete-time problem to further improve our intuition.

EXAMPLE 9.3 Grid World

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. [84].

STILL NEED TO FILL IN THIS EXAMPLE WITH PLOTS, ETC.

Dynamic programming and value iteration. Pseudo-code? (or matlab code). Convergence results. Scalability.

9.3.2 Continuous-State, Discrete-Action

Barycentric interpolation. Examples on pendulum, acrobot, cart-pole, ...

9.3.3 Continuous-State, Continous-Actions

For some cost functions, we can solve for $\min_u$ analytically, and build that into our algorithm.

9.4 INFINITE HORIZON PROBLEMS

9.5 VALUE ITERATION

9.6 VALUE ITERATION W/ FUNCTION APPROXIMATION

Point-based VI

Insert diagram of the brick with control force here.

FIGURE 9.1 The brick on ice - a mechanical double integrator.

9.6.1 Special case: Barycentric interpolation

see also stochastic optimal control.

9.7 DETAILED EXAMPLE: THE DOUBLE INTEGRATOR

Let's apply our dynamic progrmming algorithms to get some intuition about optimal control of the simplest possible second-order dynamical system:

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

This system is most commonly referred to as the double integrator. 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 ice (no friction), 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 $\mathbf{x} = [0, 0]^T$.

9.7.1 Pole placement

There are many ways to investigate and control this simple system. The state space representation of this system is

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

If we consider simple linear feedback policies of the form

$$u = -\mathbf{K}\mathbf{x} = -\begin{bmatrix} k_1 & k_2 \end{bmatrix} \mathbf{x} = -k_1 q - k_2 \dot{q},$$

then we have

$$\dot{\mathbf{x}} = (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x} = \begin{bmatrix} 0 & 1 \\ -k_1 & -k_2 \end{bmatrix} \mathbf{x}.$$

The closed-loop eigenvalues, λ_1 and λ_2, and eigenvectors, v_1 and v_2 of this system are

$$\lambda_{1,2} = \frac{-k_2 \pm \sqrt{k_2^2 - 4k_1}}{2}, \quad v_1 = \begin{bmatrix} 1 \\ \lambda_1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ \lambda_2 \end{bmatrix}.$$

The system is exponentially stable when both eigenvalues are strictly negative, e.g., $0 < \sqrt{k_2^2 - 4k_1} < k_2$. This implies that $k_1 > 0$. The eigenvalues are real when $k_2^2 \geq 4k_1$ (when $k_2^2 = 4k_1$, the system is critically damped, when $k_2^2 < 4k_1$ it is under-damped, and when $k_2^2 > 4k_1$ it is over-damped). As control designers, if we are to design k_1 and k_2, we could choose k_1 arbitrarily high, then choose $k_2 = 2\sqrt{k_1}$ to obtain an arbitrarily fast, critically damped, convergence.

An alternative way to say this same thing is to take the Laplace transform of the closed-loop system, $\ddot{q} = -k_1 q - k_2 \dot{q} + u'$, with the transfer function

$$H(s) = \frac{1}{s^2 + k_2 s + k_1} = \frac{1}{(s + \lambda_1)(s + \lambda_2)}.$$

By the same argument as above, we can easily select $k_1 = \frac{k_2^2}{4}$ to keep the poles on the real-axis, $H(s) = \frac{1}{(s+\frac{k_2}{2})^2}$, and drive those poles arbitrarily far into the left-half plane by choosing a large k_2. Or, with a more traditional root-locus analysis, we could amplify the gains k_1 and k_2 by the same linear multiplier k and see that for low k the system is underdamped (complex poles), and increasing k causes the poles to meet at the real axis and move off in opposite directions.

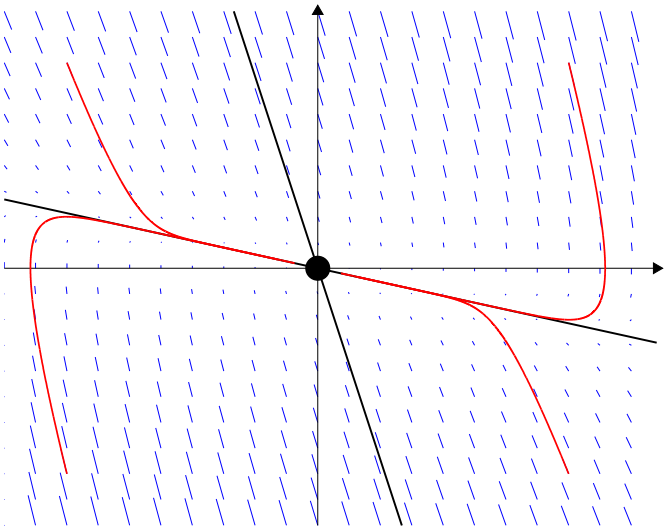


FIGURE 9.2 Phase plot of a double integrator with linear feedback, $k_1 = 1, k_2 = 4$.

9.7.2 The optimal control approach

Simple tuning of linear feedback gains can cause an arbitrarily fast convergence to the goal, but it may have many undesirable qualities. Perhaps the most obvious problem is that arbitrarily large feedback gains require arbitrarily large forces to implement. In some cases, we may have hard limits on the amount of force/torque that our actuators can produce. In other cases, since setting the gains to infinite is not a reasonable solution, we may like a more principled approach to balancing the cost of exerting large forces with the rate at which the system gets to the goal. In the remainder of this chapter we will consider to alternative formulations of optimal control problems for the double integrator to handle these two cases.

Thanks to the simplicity of the dynamics of this plant, each of the examples worked out here, will be computed analytically in the next chapters.

9.7.3 The minimum-time problem

Let's consider the problem of trying to get to the goal as fast as possible in the face of hard limits on the amount of force we can produce:

$$\ddot{q} = u, \quad u \in [-1, 1].$$

Informal derivation.

Before we do the real derivation, let's use our intuition to think through what the optimal control solution for this problem might look like.

We believe that the policy which optimizes this control problem is bang-bang; the control system should acclerate maximally towards the goal until a critical point at which it should hit the brakes in order to come perfectly to rest at the origin. Here we'll prove that this hypothesis is indeed correct.

First, we must define the bang-bang policy. We can do this by trying to figure out the manifold of states which, when the brakes are fully applied, bring the system to rest precisely at the origin. Integrating the equations, we have

$$\dot{q}(t) = ut + \dot{q}(0)$$
$$q(t) = \frac{1}{2}ut^2 + \dot{q}(0)t + q(0).$$

If we require that $q(t) = \dot{q}(t) = 0$, then these equations define a manifold of initial conditions $(q(0), \dot{q}(0))$ which come to rest at the origin:

$$\dot{q}(0) = -ut, \quad q(0) = \frac{1}{2}ut^2.$$

Cancelling t and using the "braking" policy on this manifold, $u = \text{sgn}(q)$, we have

$$\dot{q} = -\text{sgn}(q)\sqrt{2\,\text{sgn}(q)q}.$$

For all $\dot{q}$ greater than this switching surface, we apply $u = -1$, and less than, we apply $u = 1$. This policy is cartooned in Figure 9.3. Trajectories of the system executing this policy are also included - the fundamental characteristic is that the system is accelerated as quickly as possible toward the switching surface, then rides the switching surfance in to the origin.

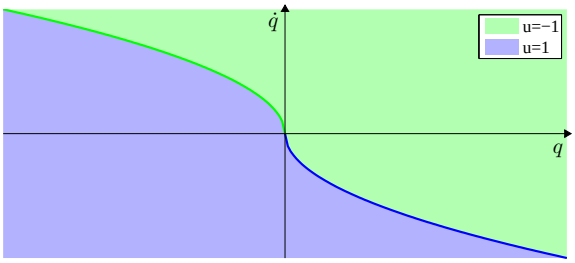


FIGURE 9.3 Candidate optimal (bang-bang) policy for the minimum-time double integrator problem.

Performance.

compare phase plots with linear control.

9.8 THE QUADRATIC REGULATOR

Performance.
compare phase plots with linear control. Moves the poles.

9.9 DETAILED EXAMPLE: THE SIMPLE PENDULUM

CHAPTER 10

Analytical Optimal Control with the Hamilton-Jacobi-Bellman Sufficiency Theorem

10.1 INTRODUCTION

10.1.1 Dynamic Programming in Continuous Time

Discrete time problems permit a simple derivation of dynamic programming. Indeed, for the numerical studies in the next chapter, and for digital (sampled-data) control systems in real robotics applications, the discrete-time treatment might be enough. However, for analytical studies it is often easier, and even more compact, to work directly in continuous time.

The Hamilton-Jacobi-Bellman Equation.

Let's develop the continuous time form of the cost-to-go function recursion by taking the limit as the time between control updates goes to zero.

$$\begin{aligned} J^*(\mathbf{x},T) =& h(\mathbf{x}) \\ J^*(\mathbf{x},t) =& \min_{[\mathbf{u}(t)\ldots\mathbf{u}(T)]} \left[h(\mathbf{x}(T)) + \int_t^T g(\mathbf{x}(t),\mathbf{u}(t))dt \right], \quad \mathbf{x}(t)=\mathbf{x}, \dot{\mathbf{x}}=\mathbf{f}(\mathbf{x},\mathbf{u}) \\ =& \lim_{dt\rightarrow 0} \min_{\mathbf{u}} \left[g(\mathbf{x},\mathbf{u})dt + J(\mathbf{x}(t+dt),t+dt) \right] \\ \approx& \lim_{dt\rightarrow 0} \min_{\mathbf{u}} \left[g(\mathbf{x},\mathbf{u})dt + J^*(\mathbf{x},t) + \frac{\partial J^*}{\partial \mathbf{x}}\dot{\mathbf{x}}dt + \frac{\partial J^*}{\partial t}dt \right] \end{aligned}$$

Simplifying, we are left with

$$\boxed{0 = \min_{\mathbf{u}} \left[g(\mathbf{x},\mathbf{u}) + \tfrac{\partial J^*}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x},\mathbf{u}) + \tfrac{\partial J^*}{\partial t} \right].} \tag{10.1}$$

This equation is well-known as the Hamilton-Jacobi-Bellman (HJB) equation.

Sufficiency theorem. The HJB equation assumes that the cost-to-go function is continuously differentiable in $\mathbf{x}$ and t, which is not necessarily the case. It therefore cannot be satisfied in all optimal control problems. It does, however, provide a sufficient condition for optimality.

Suppose we have found a policy, $\pi(\mathbf{x}, t)$, and a cost-to-go function, $J^\pi(\mathbf{x}, t)$. Suppose that π minimizes the right-hand-side of the HJB for all $\mathbf{x}$ and all $t \in [0, T]$, and that this minimum is zero for all $\mathbf{x}$ and all $t \in [0, T]$. Furthermore, suppose that $J^\pi(\mathbf{x}, T) = h(\mathbf{x})$. Then we have that

$$J^\pi(\mathbf{x}, t) = J^*(\mathbf{x}, t), \quad \pi(\mathbf{x}, t) = \pi^*(\mathbf{x}, t).$$

A more formal treatment of this theorem, including it's proof, can be found in [13].

The HJB provides limited utility for constructing optimal policies and value functions, but does provide a relatively inexpensive way to verify optimality if one is able to "guess" a solution.

Examples. The best way to get started using the HJB sufficiency theorem is to work through some examples.

EXAMPLE 10.1 First-order Regulator

Let's reconsider on of the first problems from this chapter:

$$\dot{x} = u, \quad |u| \leq 1,$$
$$h(x) = \frac{1}{2}x^2, \quad g(x, u, t) = 0.$$

Our candidate for an optimal policy was simply $\pi(x, t) = -\text{sgn}(x)$. To prove that this policy is optimal, we must first evaluate the policy to obtain J^π. The dynamics here are particularly simple (x moves one unit per second with maximal actuation), and we can quickly see that executing policy π from $x(0)$ will produce:

$$x(T) = \begin{cases} x(0) - T & x(0) > T \\ 0 & -T \leq x(0) \leq T \\ x(0) + T & x(0) < -T. \end{cases}$$

A similar expression can be written, involving $T-t$, for all starting times $t < T$. Therefore, we have

$$J^\pi(x, t) = \begin{cases} \frac{1}{2}\left[x - (T-t)\right]^2 & x > (T-t) \\ 0 & -(T-t) \leq x \leq (T-t) \\ \frac{1}{2}\left[x + (T-t)\right]^2 & x < -(T-t). \end{cases}.$$

Notice that, although the expression is written piecewise, this value function is continuously differentiable in both x and t. It can be summarized as

$$J^\pi(x, t) = \max\{0, \frac{1}{2}\left[|x| - (T-t)\right]^2\},$$

and it's derivatives are ...

$$\frac{\partial J^\pi}{\partial x} = \text{sgn}(x)\max\{0, |x| - (T-t)\}$$
$$\frac{\partial J^\pi}{\partial t} = \max\{0, |x| - (T-t)\}.$$

Substituting into the HJB, we have

$$\begin{aligned} 0 &= \min_u [0 + \text{sgn}(x) u \max\{0, |x| - (T-t)\} + \max\{0, |x| - (T-t)\}] \\ &= \min_u [(1 + \text{sgn}(x) u) \max\{0, |x| - (T-t)\}]. \end{aligned}$$

As we hoped, the policy $u = -\text{sgn}(x)$ does minimize this quantity, and the minimum is zero. By the sufficiency theorem, $\pi(x,t) = -\text{sgn}(x)$ is an optimal policy.

EXAMPLE 10.2 The Linear Quadratic Regulator (LQR)

The linear quadratic regulator is clearly the most important and influential result in optimal control theory to date. 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 with

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{x}^T \mathbf{Q}_f \mathbf{x}, \quad \mathbf{Q}_f = \mathbf{Q}_f^T \geq \mathbf{0} \\ g(\mathbf{x}, \mathbf{u}, t) &= \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}, \quad \mathbf{Q} = \mathbf{Q}^T \geq 0, \mathbf{R} = \mathbf{R}^T > 0 \end{aligned}$$

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:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}} &= 2\mathbf{u}^T \mathbf{R} + \frac{\partial J^*}{\partial \mathbf{x}} \mathbf{B} = 0 \\ \mathbf{u}^* = \pi^*(\mathbf{x}, t) &= -\frac{1}{2} \mathbf{R}^{-1} \mathbf{B}^T \frac{\partial J^*}{\partial \mathbf{x}}^T \end{aligned}$$

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:

$$J^*(\mathbf{x}, t) = \mathbf{x}^T \mathbf{S}(t) \mathbf{x}, \quad \mathbf{S}(t) = \mathbf{S}^T(t) > \mathbf{0}.$$

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

$$\begin{aligned} \mathbf{u}^* = \pi^*(\mathbf{x}, t) &= -\mathbf{R}^{-1} \mathbf{B}^T \mathbf{S}(t) \mathbf{x} \\ 0 &= \mathbf{x}^T \left[\mathbf{Q} - \mathbf{S}(t) \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{S}(t) + 2\mathbf{S}(t)\mathbf{A} + \dot{\mathbf{S}}(t) \right] \mathbf{x}. \end{aligned}$$

All of the matrices here are symmetric, except for $\mathbf{S}(t)\mathbf{A}$. But since $\mathbf{x}^T \mathbf{S}(t) \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{A}^T \mathbf{S}(t) \mathbf{x}$, we can equivalently write the symmetric form (which we assumed):

$$0 = \mathbf{x}^T \left[\mathbf{Q} - \mathbf{S}(t) \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{S}(t) + \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T \mathbf{S}(t) + \dot{\mathbf{S}}(t) \right] \mathbf{x}.$$

Therefore, $\mathbf{S}(t)$ must satisfy the condition (known as the continuous time 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

$$\mathbf{S}(T) = \mathbf{Q}_f.$$

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.

The dependence on time in both the value function and the feedback policy might surprise readers who have used LQR before. The more common usage is actually the infinite-horizon version of the problem, which we will develop in Example 6.

EXAMPLE 10.3 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:

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\
h(\mathbf{x}) &= (\mathbf{x} - \mathbf{x}^d(T))^T\mathbf{Q}_f(\mathbf{x} - \mathbf{x}^d(T)), \quad \mathbf{Q}_f = \mathbf{Q}_f^T \geq 0 \\
g(\mathbf{x}, \mathbf{u}, t) &= (\mathbf{x} - \mathbf{x}^d(t))^T\mathbf{Q}(\mathbf{x} - \mathbf{x}^d(t)) + (\mathbf{u} - \mathbf{u}^d(t))^T\mathbf{R}(\mathbf{u} - \mathbf{u}^d(t)), \\
\mathbf{Q} &= \mathbf{Q}^T \geq 0, \mathbf{R} = \mathbf{R}^T > 0
\end{aligned}$$

Now, guess a solution

$$J^*(\mathbf{x}, t) = \mathbf{x}^T\mathbf{S_2}(t)\mathbf{x} + \mathbf{x}^T\mathbf{s_1}(t) + s_0(t), \quad \mathbf{S_2}(t) = \mathbf{S_2}^T(t) > \mathbf{0}.$$

In this case, we have

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S_2}(t) + \mathbf{s_1}^T(t), \quad \frac{\partial J^*}{\partial t} = \mathbf{x}^T\dot{\mathbf{S_2}}(t)\mathbf{x} + \mathbf{x}^T\dot{\mathbf{s_1}}(t) + \dot{s_0}(t).$$

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

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{u}} &= 2(\mathbf{u} - \mathbf{u}^d(t))^T\mathbf{R} + (2\mathbf{x}^T\mathbf{S_2}(t) + \mathbf{s_1}^T(t))\mathbf{B} = 0, \\
\mathbf{u}^*(t) &= \mathbf{u}^d(t) - \mathbf{R}^{-1}\mathbf{B}^T\left[\mathbf{S_2}(t)\mathbf{x} + \frac{1}{2}\mathbf{s_1}(t)\right]
\end{aligned}$$

The HJB can be satisfied by integrating backwards

$$\begin{aligned}
-\dot{\mathbf{S_2}}(t) =& \mathbf{Q} - \mathbf{S_2}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S_2}(t) + \mathbf{S_2}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S_2}(t) \\
-\dot{\mathbf{s_1}}(t) =& -2\mathbf{Q}\mathbf{x}^d(t) + \left[\mathbf{A}^T - \mathbf{S}_2\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\right]\mathbf{s_1}(t) + 2\mathbf{S_2}(t)\mathbf{B}\mathbf{u}^d(t) \\
-\dot{s_0}(t) =& \mathbf{x}^d(t)^T\mathbf{Q}\mathbf{x}^d(t) - \frac{1}{4}\mathbf{s_1}^T(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{s_1}(t) + \mathbf{s_1}(t)^T\mathbf{B}\mathbf{u}^d(t),
\end{aligned}$$

from the final conditions

$$\begin{aligned}\mathbf{S_2}(T) =& \mathbf{Q}_f \\ \mathbf{s_1}(T) =& -2\mathbf{Q}_f\mathbf{x}^d(T) \\ s_0(T) =& \left[\mathbf{x}^d(T)\right]^T \mathbf{Q}_f \left[\mathbf{x}^d(T)\right].\end{aligned}$$

Notice that the solution for $\mathbf{S_2}$ 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 $J(x,t)$ must be uniformly positive. This is true iff $\mathbf{S}_2 > 0$ and $s_0 > \frac{1}{4}\mathbf{s}_1^T\mathbf{S}_2^{-1}\mathbf{s}_1$, which comes from evaluating the function at x_{min} defined by $\frac{\partial}{\partial x} = 0$.

EXAMPLE 10.4 Minimum-time Linear Quadratic Regulator

$$\begin{gathered}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ h(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}_f\mathbf{x}, \quad \mathbf{Q}_f = \mathbf{Q}_f^T \geq 0 \\ g(\mathbf{x},\mathbf{u},t) = 1 + \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}\mathbf{R}\mathbf{u}, \\ \mathbf{Q} = \mathbf{Q}^T \geq 0, \mathbf{R} = \mathbf{R}^T > 0\end{gathered}$$

EXAMPLE 10.5 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

$$\mathbf{P}(T) = 0.$$

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 law.

10.2 INFINITE-HORIZON PROBLEMS

Limit as $T \to \infty$. Value function converges (or blows up).

Examples and motivations.

10.2.1 The Hamilton-Jacobi-Bellman

For infinite-horizon, all dependence of J on t drops out, and the HBJ reduces to:

$$0 = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}) \right].$$

10.2.2 Examples

EXAMPLE 10.6 The Infinite-Horizon Linear Quadratic Regulator

Consider again a system in LTI state space form,

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

but now consider the infinite-horizon cost function given by

$$g(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}, \quad \mathbf{Q} = \mathbf{Q}^T \geq \mathbf{0}, \mathbf{R} = \mathbf{R}^T > 0.$$

Since we know that the optimal value function cannot depend on time in the infinite-horizon case, we will guess the form:

$$J^*(\mathbf{x}) = \mathbf{x}^T \mathbf{S} \mathbf{x}.$$

This yields the optimal policy

$$\mathbf{u}^* = \pi^*(\mathbf{x}) = -\mathbf{R}^{-1} \mathbf{B}^T \mathbf{S} \mathbf{x} = -\mathbf{K}\mathbf{x}.$$

In fact, this form can also be marched through the HJB and verified. The solution, not surprisingly, is the steady-state solution of the Riccati equation described in the finite-horizon problem. Setting $\dot{\mathbf{S}}(t) = 0$, we have

$$0 = \mathbf{S}\mathbf{A} + \mathbf{A}^T \mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T \mathbf{S} + \mathbf{Q}.$$

Note that this equation is not linear in $\mathbf{S}$, and therefore solving the equation is non-trivial (but well understood). Both the optimal policy and optimal value function are available from MATLAB by calling

```
[K,S]=lqr(A,B,Q,R).
```

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 10.7 Second-order Quadratic Regulator

Let's use the LQR result to solve for the optimal regulator of our trivial linear system:

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

EXAMPLE 10.8 The Overdamped Pendulum

PROBLEMS

10.1. (CHALLENGE) *Optimal control of the simple pendulum.*

Find the optimal policy for the minimum-time problem on the simple pendulum described by the dynamics:

$$\ddot{q} = u - \sin(q).$$

CHAPTER 12

Trajectory Optimization

So far, we have discussed a number of ways to solve optimal control problems via state space search (e.g., Dijkstra's and Dynamic Programming/Value Iteration). These methods have the drawback that they force you to discretize, typically both the state and action spaces. The resulting policy is optimal for the discretized system, but is only an approximation to the optimal solution for the continuous system. Today we're going to introduce a different type of algorithm which searches in the space of control policies, instead of in state space. These methods typically have the advantage that they scale (much) better to higher dimensional systems, and do not require any discretizations of state or action spaces. The disadvantage of this class of algorithms is that we must sacrifice guarantees of global optimality - they only guarantee local optimality, and can be subject to local minima.

12.1 THE POLICY SPACE

The fundamental idea is policy search methods is that, instead of discretizing an searching directly for the optimal policy, we define a class of policies by describing some parameterized control law. Then we search direcly for a setting of these parameters that (locally) optimize the cost function.

Notationally, we collect all of the parameters into a single vector, α, and write the controller (in general form) as $\pi_\alpha(\mathbf{x}, t)$. Here are some examples:

- linear feedback law:

$$\mathbf{u} = \mathbf{K}\mathbf{x} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \\ \alpha_5 & \alpha_6 & & \\ \vdots & & & \end{bmatrix} \mathbf{x}.$$

- open loop controller

$$\mathbf{u} = \alpha_n, \text{ where } n = \text{floor}(t/dt).$$

  Or α could be spline or fourier coefficients of the trajectory $\mathbf{u}(t)$.

- Neural networks and general function approximators.

12.2 NONLINEAR OPTIMIZATION

Having defined the policy space, our task is to search over α for the best policy. If the number of parameters is small, then brute force search may work. More generally, we must employ tools from nonlinear optimization.

12.2.1 Gradient Descent

First-order methods in general. Pros/cons of gradient descent. Suffer from:

- local minima
- have to chose a step-size (problems with anisotropy, or the golf-course problem)
- can require many iterations to converge

12.2.2 Sequential Quadratic Programming

Newton's method, etc. Intro chapter from Betts is a great guide. Powerful software packages (e.g., SNOPT).

12.3 SHOOTING METHODS

Perhaps the most obvious method for evaluating and optimizing the cost function from a single initial condition is by defining α as the decision variables and by evaluating $J^\alpha(\mathbf{x}_0)$ via forward simulation. Typically, the only way that these *shooting* methods take advantage of the additive cost structure of the optimal control problem is through efficient calculations of the policy gradient - $\frac{\partial J^\alpha(\mathbf{x}_0)}{\partial \alpha}$. Computing this gradient explicitly (rather than through numerical differentiation) greatly improves the performance of the optimization algorithms. We present two methods for computing this gradient here.

12.3.1 Computing the gradient with Backpropagation through time (BPTT)

Known for a long time in optimal control, but the name backprop-through-time came from the neural networks [66] community.

- Given the long-term cost function

$$J(\mathbf{x}_0) = \int_0^T g(\mathbf{x}(t), \mathbf{u}(t))dt, \quad \mathbf{x}(0) = \mathbf{x}_0.$$

- Starting from the initial condition $\mathbf{x}(0)$, integrate the equations of motion

$$\dot{\mathbf{x}} = f(\mathbf{x}, \pi_\alpha(\mathbf{x}, t)) \tag{12.1}$$

  forward in time to $t = T$. Keep track of $\mathbf{x}(t)$.

- Starting from the initial condition $\mathbf{y}(T) = \mathbf{0}$, integrate the *adjoint* equations

$$-\dot{\mathbf{y}} = \mathbf{F}_\mathbf{x}^T \mathbf{y} - \mathbf{G}_\mathbf{x}^T \tag{12.2}$$

  backward in time until $t = 0$, where

$$\mathbf{F}_\mathbf{x}(t) = \frac{\partial f}{\partial \mathbf{x}(t)} + \frac{\partial f}{\partial \mathbf{u}(t)} \frac{\partial \pi_\alpha}{\partial \mathbf{x}(t)}, \quad \mathbf{G}_\mathbf{x}(t) = \frac{\partial g}{\partial \mathbf{x}(t)} + \frac{\partial g}{\partial \mathbf{u}(t)} \frac{\partial \pi_\alpha}{\partial \mathbf{x}(t)},$$

  evaluated at $\mathbf{x}(t), \mathbf{u}(t)$.

- Finally, our gradients are given by

$$\frac{\partial J(\mathbf{x}_0)}{\partial \alpha} = \int_0^T dt \left[\mathbf{G}_\alpha^T - \mathbf{F}_\alpha^T \mathbf{y}\right], \tag{12.3}$$

where

$$\mathbf{F}_\alpha(t) = \frac{\partial f}{\partial \mathbf{u}(t)} \frac{\partial \pi_\alpha}{\partial \alpha}, \quad \mathbf{G}_\alpha(t) = \frac{\partial g}{\partial \mathbf{u}(t)} \frac{\partial \pi_\alpha}{\partial \alpha},$$

evaluated at $\mathbf{x}(t), \mathbf{u}(t)$.

Derivation w/ Lagrange Multipliers.
This algorithm minimizes the cost function

$$J^\alpha(\mathbf{x}_0) = \int_0^T g(\mathbf{x}, \pi_\alpha(\mathbf{x}, t))dt$$

subject to the constraint

$$\dot{\mathbf{x}} = f(\mathbf{x}, \pi_\alpha(\mathbf{x}, t)).$$

We will prove this using Lagrange multipliers and the calculus of variations, but it can also be shown using a finite-difference approximation of the unfolded network.

Consider the functional (a function of functions)

$$S[\mathbf{x}(t), \mathbf{y}(t)] = \int_0^T dt \left[g(\mathbf{x}, \pi_\alpha(\mathbf{x}, t)) + \mathbf{y}^T[\dot{\mathbf{x}} - f(\mathbf{x}, \pi_\alpha(\mathbf{x}, t))]\right].$$

If $\mathbf{x}(t)$ and $\mathbf{y}(t)$ are changed by small amounts $\delta\mathbf{x}(t)$ and $\delta\mathbf{y}(t)$, then the change in S is

$$\delta S = \int_0^T dt \left[\frac{\delta S}{\delta \mathbf{x}(t)}\delta\mathbf{x}(t) + \frac{\delta S}{\delta \mathbf{y}(t)}\delta\mathbf{y}(t)\right].$$

If $\frac{\delta S}{\delta \mathbf{x}(t)} = 0$ and $\frac{\delta S}{\delta \mathbf{y}(t)} = 0$ for all t, then $\delta S = 0$, and we say that S is at a stationary point with respect to variations in $\mathbf{x}$ and $\mathbf{y}$. To ensure that $\delta\mathbf{x}(0) = 0$, we will hold the initial conditions $\mathbf{x}(0)$ constant.

Now compute the functional derivatives:

$$\frac{\delta S}{\delta \mathbf{y}(t)} = [\dot{\mathbf{x}} - f(\mathbf{x}, \pi_\alpha(\mathbf{x}, t))]^T.$$

The forward dynamics of the algorithm gaurantee that this term is zero. To compute $\frac{\delta S}{\delta \mathbf{x}(t)}$, we first need to integrate by parts to handle the $\dot{\mathbf{x}}$ term:

$$\int_0^T \mathbf{y}^T \dot{\mathbf{x}} dt = \mathbf{y}^T \mathbf{x}|_0^T - \int_0^T \dot{\mathbf{y}}^T \mathbf{x} dt.$$

Rewrite S as

$$S = \int_0^T dt \left[g(\mathbf{x}, \pi_\alpha(\mathbf{x}, t)) - \mathbf{y}^T f(\mathbf{x}, \pi_\alpha(\mathbf{x}, t)) - \dot{\mathbf{y}}^T \mathbf{x}\right] + \mathbf{y}(T)^T \mathbf{x}(T) - \mathbf{y}(0)^T \mathbf{x}(0).$$

Therefore, the function derivative is

$$\frac{\delta S}{\delta \mathbf{x}(t)} = \mathbf{G}_\mathbf{x} - \mathbf{y}^T\mathbf{F}_\mathbf{x} - \dot{\mathbf{y}}^T + \mathbf{y}(T)^T\delta(t-T) - \mathbf{y}(0)^T\delta(t).$$

By choosing $\mathbf{y}(T) = \mathbf{0}$, the backward dynamics guarantee that this term is zero (except for at $t = 0$, but $\delta\mathbf{x}(0) = 0$).

The forward and backward passes put us at a stationary point of S with respect to variations in $\mathbf{x}(t)$ and $\mathbf{y}(t)$, therefore the only dependence of S on $\mathbf{w}$ is the explicit dependence:

$$\frac{\partial S}{\partial \alpha} = \int_0^T dt \left[\mathbf{G}_\alpha - \mathbf{y}^T\mathbf{F}_\alpha\right]$$

12.3.2 Computing the gradient w/ Real-Time Recurrent Learning (RTRL)

Backpropagating through time requires that the network maintains a trace of it's activity for the duration of the trajectory. This becomes very inefficient for long trajectories. Using Real-Time Recurrent Learning (RTRL), we solve the temporal credit assignment problem during the forward integration step by keeping a running estimate of the total effect that parameters α have on the state $\mathbf{x}$. The algorithm is

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \pi_\alpha(\mathbf{x}, t), t) \tag{12.4}$$

$$\dot{\mathbf{P}} = \mathbf{F}_\mathbf{x}\mathbf{P} + \mathbf{F}_\alpha \tag{12.5}$$

$$\frac{d}{dt}\frac{\partial J^\alpha(\mathbf{x}_0)}{\partial \alpha} = \mathbf{G}_\mathbf{x}\mathbf{P} + \mathbf{G}_\alpha \tag{12.6}$$

These equations are integrated forward in time with initial conditions $\mathbf{x}(0) = \mathbf{x}_0$, $\mathbf{P}(0) = \mathbf{0}$, $\frac{\partial J}{\partial \alpha} = \mathbf{0}$.

This algorithm can be computationally expensive, because we have to store $O(NM)$ variables in memory and process as many differential equations at every time step. On the other hand, we do not have to keep a trace of the trajectory, so the memory footprint does not depend on the duration of the trajectory. The major advantage, however, is that we do not have to execute the backward dynamics - the temporal credit assignment problem is solved during the forward pass.

Derivation.

Once again, we can show that this algorithm performs gradient descent on the cost function

$$J(\mathbf{x}_0) = \int_0^T g(\mathbf{x}, \mathbf{u}, t)dt.$$

Define

$$P_{ij} = \frac{\partial x_i}{\partial \alpha_j}.$$

The gradient calculations are straight-forward:

$$\frac{\partial J}{\partial \alpha} = \int_0^T dt \left[\frac{\partial g}{\partial \mathbf{x}(t)} \frac{\partial \mathbf{x}(t)}{\partial \alpha} + \frac{\partial g}{\partial \mathbf{u}(t)} \frac{\partial \pi_\alpha}{\partial \alpha} + \frac{\partial g}{\partial \mathbf{u}(t)} \frac{\partial \pi}{\partial \mathbf{x}(t)} \frac{\partial \mathbf{x}(t)}{\partial \alpha} \right]$$
$$= \int_0^T dt \left[\mathbf{G_x P} + \mathbf{G}_\alpha \right]$$

To calculate the dynamics of $\mathbf{p}$ and $\mathbf{q}$, differentiate the equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \pi_\alpha(\mathbf{x}, t), t)$$

to obtain

$$\frac{d}{dt}\frac{\partial \mathbf{x}}{\partial \alpha} = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \pi_\alpha}{\partial \mathbf{x}} \right] \frac{\partial \mathbf{x}}{\partial \alpha} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \pi_\alpha}{\partial \alpha}$$
$$\dot{\mathbf{P}} = \mathbf{F_x P} + \mathbf{F}_\alpha \quad \mathbf{P}(0) = \mathbf{0}$$

By initializing $\mathbf{P}(0) = \mathbf{0}$, we are simply assuming that the $\mathbf{x}(0)$ does not depend on α.

12.3.3 BPTT vs. RTRL

Both methods compute the gradient. So which should we prefer? The answer may be problem specific. The memory cost of the BPTT algorithm is that you must remember $\mathbf{x}(t)$ on the forward simulation, which is $dim(\mathbf{x}) \times length(T)$. The memory cost of RTRL is storing $\mathbf{P}$ which has size $dim(\mathbf{x}) \times dim(\alpha)$. Moreover, RTRL involves integrating $dim(\mathbf{x}) + dim(\mathbf{x}) \times dim(\alpha)$ ODEs, where as BPTT only integrates $2 \times dim(\mathbf{x})$ ODEs. For long trajectories and a small number of policy parameters, RTRL might more efficient that BPTT, and is probably easier to implement. For shorter trajectories with a lot of parameters, BPTT would probably be the choice.

Another factor which might impact the decision is the constraints. Gradients of constraint functions can be computed with either method, but if one seeks to constrain $\mathbf{x}(t)$ for all t, then RTRL might have an advantage since it explicitly stores $\frac{\partial \mathbf{x}(t)}{\partial \alpha}$.

12.4 DIRECT COLLOCATION

The shooting methods require forward simulation of the dynamics to compute the cost function (and its gradients), and therefore can be fairly expensive to evaluation. Additionally, they do not make very effective use of the capabilities of modern nonlinear optimization routines (like SNOPT) to enforce, and even take advantage of, constraints.

In the direct collocation approach, we formulate the decision parameters as *both* α, which in the open-loop case reduces to $\mathbf{u}[n]$, and additionally $\mathbf{x}[n]$. By over-parameterizing the system with both $\mathbf{x}$ and $\mathbf{u}$ are explicit decision variables, the cost function and its gradients can be evaluated without explicit simulation of the dynamics. Instead, the dynamics are imposed as constraints on the decision variables. The resulting formuation is:

$$\min_{\forall n, \mathbf{x}[n], \mathbf{u}[n]} J = \sum_{n=1}^{N} g(\mathbf{x}[n], \mathbf{u}[n]), \quad \text{s.t. } \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n]).$$

These methods are very popular today. The updates are fast, and it is very easy to implement constraints. The most commonly stated limitation of these methods is their

accuracy, since they use a fixed step-size integration (rather than a variable step solver). However, methods like BPTT or RTRL can also be used to implement the state-action constraints if accuracy is a premium.

A less obvious attribute of these methods is that they may be easier to initialize with trajectories (eg, specifying $\mathbf{x}_0(t)$ directly) that are in the vicinity of the desired minima.

12.5 LQR TRAJECTORY STABILIZATION

When the policy parameterization is explicitly the open-loop trajectory (u_{tape}), then trajectory optimization by shooting methods and/or direct collocation both have the property that the solution upon convergence satisfies the Pontryagin minimum principle. When the policy is parameterized with feedback, the story is a little more complicated (but similar). But there is no reason to believe that the system is stable along these trajectories - executing the locally optimal open-loop trajectories on the system with small changes in initial conditions, small disturbances or modeling errors, or even with a different integration step can cause the simulated trajectories to diverge from the planned trajectory.

In this section we will develop one set of tools for trajectory stabilization. There are many candidates for trajectory stabilization in fully-actuated sytems (many based on feedback linearization), but trajectory stabilization for underactuated systems can be easily implemented using a version of the Linear Quadratic Regulator (LQR) results from chapter 10.

12.5.1 Linearizing along trajectories

In order to apply the linear quadratic regulator, we must first linearize the dynamics. So far we have linearized around fixed points of the dynamics... linearizing around a non-fixed point is just slightly more subtle. Considering the system

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

if we perform a Taylor expansion around a random point $(\mathbf{x}_0, \mathbf{u}_0)$, the result is

$$\dot{\mathbf{x}} \approx \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) = \mathbf{c} + \mathbf{A}(\mathbf{x} - \mathbf{x}_0) + \mathbf{B}(\mathbf{u} - \mathbf{u}_0).$$

In other words, the resulting dynamics are not linear (but affine) in the coordinates of $\mathbf{x}$. The simple trick is to change coordinates to

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_0(t), \quad \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_0(t),$$

where $[\mathbf{x}_0(t), \mathbf{u}_0(t)]$ is a solution to the dynamics - a feasible trajectory. Then we have

$$\dot{\bar{\mathbf{x}}}(t) = \dot{\mathbf{x}}(t) - \dot{\mathbf{x}}_0(t) = \dot{\mathbf{x}}(t) - \mathbf{f}(\mathbf{x}_0(t), \mathbf{u}_0(t)),$$

and therefore

$$\begin{aligned}\dot{\bar{\mathbf{x}}}(t) =& \frac{\partial \mathbf{f}(\mathbf{x}_0(t), \mathbf{u}_0(t))}{\partial \mathbf{x}}(\mathbf{x}(t) - \mathbf{x}_0(t)) + \frac{\partial \mathbf{f}(\mathbf{x}_0(t), \mathbf{u}_0(t))}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0(t)) \\ =& \mathbf{A}(t)\bar{\mathbf{x}}(t) + \mathbf{B}(t)\bar{\mathbf{u}}(t).\end{aligned}$$

In other words, if we are willing to change to a coordinate system that moves along a feasible trajectory, than the Taylor expansion of the dynamics results in a time-varying

linear system. Linear time-varying (LTV) systems are a very rich class of systems which are still amenable to many of the linear systems analysis[21].

The big idea here is that we are using a particular solution trajectory to reparameterize the path through state-space as purely a function of time. [Add cartoon images here].

12.5.2 Linear Time-Varying (LTV) LQR

Given a linear time-varying approximation of the model dynamics along the trajectory,

$$\dot{\bar{\mathbf{x}}} = \mathbf{A}(t)\bar{\mathbf{x}} + \mathbf{B}(t)\bar{\mathbf{u}},$$

we can formulate a trajectory stabilization as minimizing the cost function

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

This cost function penalizes the system (quadratically) at time t for being away from $\mathbf{x}_0(t)$. Even with the time-varying components of the dynamics and the cost function, it is still quite simple to use the finite-horizon LQR solution from 2. If, as before, we guess

$$J(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T \mathbf{S}(t) \bar{\mathbf{x}},$$

we can satisfy the HJB with:

$$-\dot{\mathbf{S}}(t) = \mathbf{Q} - \mathbf{S}(t)\mathbf{B}(t)\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \mathbf{S}(t)\mathbf{A}(t) + \mathbf{A}^T(t)\mathbf{S}(t), \quad \mathbf{S}(T) = \mathbf{Q}_f.$$
$$\mathbf{u}^*(t) = \mathbf{u}_0(t) - \mathbf{R}^{-1}\mathbf{B}^T(t)\mathbf{S}(t)\bar{\mathbf{x}}(t)$$

In general, it is also trivial to make $\mathbf{Q}$ and $\mathbf{R}$ functions of time. It is also nice to observe that if one aims to stabilize an infinite-horizon trajectory, for which $\forall t \geq t_f, \mathbf{x}_0(t) = \mathbf{x}_0(t_f), \mathbf{u}_0(t) = \mathbf{u}_0(t_f)$, and $\mathbf{f}(\mathbf{x}_0(t_f), \mathbf{u}_0(t_f) = \mathbf{0}$, then we can use the boundary conditions $\mathbf{S}(T) = \mathbf{S}_\infty$, where $\mathbf{S}_\infty$ is the steady-state solution of the LTI LQR at the fixed point.

[Add simulation results from the pendulum, with and without feedback, and cart-pole, using both dircol and shooting?]

[Add image of value function estimates on top of pendulum trajectory]

Notice that, like the LTI LQR, this control derivation should scale quite nicely to high dimensional systems (simply involves integrating a $n \times n$ matrix backwards). Although dynamic programming, or some other nonlinear feedback design tool, could be used to design trajectory stabilizers for low-dimensional systems, for systems where open-loop trajectory optimization is the tool of choice, the LTV LQR stabilizers are a nice match.

12.6 ITERATIVE LQR

The LTV LQR solutions used to stabilize trajectories in the last section can also be modified to create an algorithm for unconstrained optimization of open-loop trajectories. Iterative LQR (iLQR), also known as Sequential LQR (SLQ)[74], and closely related to Differential Dynamic Programming (DDP)[42], can be thought of as almost a drop-in replacement for a shooting or direct collocation method.

The instantaneous cost function for trajectory stabilization took the form: $\bar{\mathbf{x}}^T\mathbf{Q}\bar{\mathbf{x}}$, with the result being that states off the desired trajectory are regulated back to the desired trajectory. But what happens if we use the LQR derivation to optimize a more arbitrary cost function? Given an instantaneous cost function, $g(\mathbf{x},\mathbf{u})$, we can form a quadratic approximation of this cost function with a Taylor expansion about $\mathbf{x}_0(t), \mathbf{u}_o(t)$:

$$g(\mathbf{x},\mathbf{u}) \approx g(\mathbf{x}_0,\mathbf{u}_0) + \frac{\partial g}{\partial \mathbf{x}}\bar{\mathbf{x}} + \frac{\partial g}{\partial \mathbf{u}}\bar{\mathbf{u}} + \frac{1}{2}\bar{\mathbf{x}}^T\frac{\partial^2 g}{\partial \mathbf{x}^2}\bar{\mathbf{x}} + \bar{\mathbf{x}}\frac{\partial^2 g}{\partial \mathbf{x}\partial \mathbf{u}}\bar{\mathbf{u}} + \frac{1}{2}\bar{\mathbf{u}}^T\frac{\partial^2 g}{\partial \mathbf{u}^2}\bar{\mathbf{u}}.$$

By inserting this (time-varying) cost function into the LQR solution, with the time-varying linearization around a nominal trajectory, we can once again derive an optimal control policy by integrating a Riccati equation backwards (almost identical to the derivation in example 3). [insert official derivation here]

Given an initial trajectory $\mathbf{x}_0(t), \mathbf{u}_0(t)$ (generated, for instance, by choosing $\mathbf{u}_0(t)$ to be some random initial trajectory, then simulating to get $\mathbf{x}_0(t)$), the cost function no longer rewards the system for stabilizing the desired trajectory, but rather for driving towards the minimum of the cost function (which has an interpretation as a different desired trajectory, $\mathbf{x}_d(t) \neq \mathbf{x}_0(t)$). The LQR solution will use the linearization along $\mathbf{x}_0(t)$ to try to stabilize the system on $\mathbf{x}_d(t)$. Because the LTV model is only valid near $\mathbf{x}_0(t)$, the resulting controller may do a poor job of getting to $\mathbf{x}_d(t)$, but will be better than the previous controller. The algorithm proceeds by replacing $\mathbf{u}_0(t)$ with the control actions executed by the LQR feedback policy from the initial conditions, computes the corresponding new $\mathbf{x}_0(t)$, and iterates.

[insert implementation snapshots here for intuition.]

Iterative LQR can be thought of as a second-order solution method (like SQP), which should converge on a trajectory which satisfies the Pontryagin minimum principle in a relatively small number of interations. It will likely require fewer trajectories to be considered that using BPTT, RTRL, or DIRCOL, because the LQR solution directly computes a second-order update, whereas the other methods as presented compute only $\frac{\partial J}{\partial \alpha}$, and rely on SNOPT to approximate the Hessian.

The very popular DDP method[42] is very similar to this solution, although it also uses a second-order expansion of the equations of motion. Many authors say that they are using DDP when they are actually using iLQR[1].

12.7 REAL-TIME PLANNING (AKA RECEDING HORIZON CONTROL)

LQR trajectory stabilization is one approach to making the optimized open-loop trajectories robust to disturbances. But if you can plan fast enough (say for steering a ship), then computing a short-term finite-horizon open-loop policy at every dt using the current state as initial condtions can be another reasonable approach to creating what is effectively a feedback policy. The technique is known as receding-horizon control.

CHAPTER 17

Model-free Policy Search

17.1 INTRODUCTION

In chapter 12, we talked about policy search as a nonlinear optimization problem, and discussed efficient ways to calculate the gradient of the long-term cost:

$$J^\alpha(\mathbf{x}_0) = \int_0^T g(\mathbf{x}, \pi_\alpha(\mathbf{x}, t))dt, \text{ s.t. } \mathbf{x}(0) = \mathbf{x}_0.$$

Assuming $J^\alpha(\mathbf{x}_0)$ is smooth over $\mathbf{w}$, we derived updates of the form:

$$\Delta\alpha = -\eta \left[\frac{\partial J^\alpha}{\partial \alpha}\right]^T_{\mathbf{x}_0,\alpha}.$$

But it may not always be possible, or practical, to compute the gradients exactly.

A standard technique for estimating the gradients, in lieu of analytical gradient information, is the method of finite differences[67]. The finite differences approach to estimating the gradient involves making the same small perturbation, ϵ to the input parameters in every dimension independently, and using:

$$\frac{\partial J^\alpha}{\partial \alpha_i} \approx \frac{J^{\alpha+\epsilon_i}(\mathbf{x}_0) - J^\alpha(\mathbf{x}_0)}{\epsilon},$$

where $\epsilon_\mathbf{i}$ is the column vector with ϵ in the ith 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.

In this chapter we will develop *stochastic* gradient descent algorithms which can, in expectation, descend a policy gradient with considerably less evaluations than those required for finite differences. In addition, we will develop methods that are robust to stochasticity in the function evaluation, which is inevitable if one is trying to descend a policy gradient on a real robot. This allows for the exciting possibility of optimizing a control policy for a system without requiring any model of the plant.

17.2 STOCHASTIC GRADIENT DESCENT

Requirements and guarantees. For a formal treatment, see Bertsekas.

17.3 THE WEIGHT PERTUBATION ALGORITHM

Instead of sampling each dimension independently, consider making a single small random change, β, to the parameter vector, α. Assuming that the function is locally smooth, we can approximate the result of this experiment using a Taylor expansion:

$$J^{\alpha+\beta}(\mathbf{x}_0) \approx J^\alpha(\mathbf{x}_0) + \frac{\partial J^\alpha(\mathbf{x}_0)}{\partial \alpha}\beta.$$

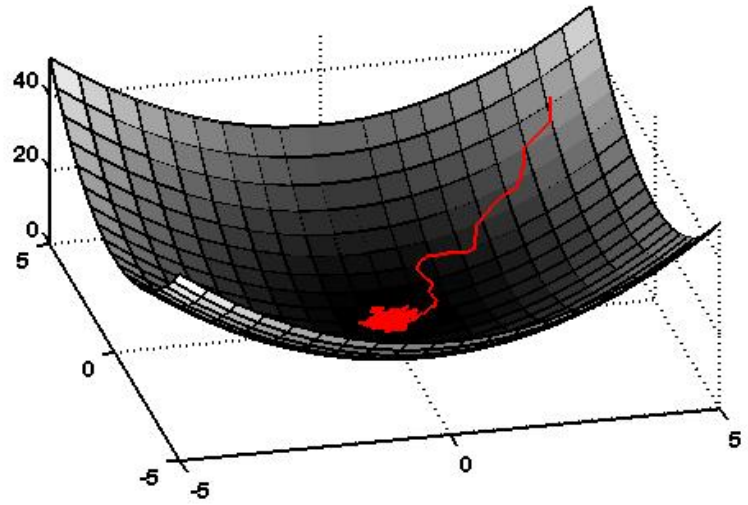


FIGURE 17.1 Stochastic gradient descent of a quadratic cost function. See problem 1 for details.

Now consider the update of the form:

$$\Delta\alpha = -\eta[J^{\alpha+\beta}(\mathbf{x}_0) - J^{\alpha}(\mathbf{x}_0)]\beta.$$

Since $J^{\alpha+\beta} - J^{\alpha}$ is (approximately) the dot product between $\frac{\partial J^{\alpha}}{\partial \alpha}$ and β, the sign of this term ensures that $\Delta\alpha$ is always within 90 degrees of true gradient descent.

Furthermore, on average, the update is in the direction of the true gradient:

$$\Delta\alpha \approx -\eta\beta\beta^T \frac{\partial J}{\partial \alpha}^T$$
$$E\left[\Delta\alpha\right] \approx -\eta E\left[\beta\beta^T\right] \frac{\partial J}{\partial \alpha}^T$$

If we select each element of β independently from a distribution with zero mean and variance σ_β^2, or $E\left[\beta_i\right] = 0, E\left[\beta_i\beta_j\right] = \sigma_\beta^2\delta_{ij}$, then we have

$$E\left[\Delta\alpha\right] \approx -\eta\sigma_\beta^2 \frac{\partial J}{\partial \alpha}^T.$$

Note that the distribution $f_{\alpha_i}(\alpha_i)$ need not be Gaussian, but it is the variance of the distribution which determines the scaling on the gradient. Bringing the σ_β^2 into the update, we have the weight perturbation algorithm:

$$\boxed{\Delta\alpha = -\tfrac{\eta}{\sigma_\beta^2}[J^{\alpha+\beta}(\mathbf{x}_0) - J^{\alpha}(\mathbf{x}_0)]\beta} \tag{17.1}$$

17.3.1 Performance of Weight Perturbation

The simplicity of the weight perturbation update makes it tempting to apply it 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). 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 $\Delta\alpha = -\eta\frac{\partial J}{\partial \alpha}^T + \text{noise}$)[1]

$$\text{SNR} = \frac{\left|-\eta\frac{\partial J}{\partial \alpha}^T\right|^2}{E\left[\left|\Delta\alpha + \eta\frac{\partial J}{\partial \alpha}^T\right|^2\right]}.$$

In the special case of the unbiased update, the equation reduces to:

$$\text{SNR} = \frac{E\left[\Delta\alpha\right]^T E\left[\Delta\alpha\right]}{E\left[(\Delta\alpha)^T(\Delta\alpha)\right] - E\left[\Delta\alpha\right]^T E\left[\Delta\alpha\right]}.$$

For the weight perturbation update we have:

$$E\left[\Delta\alpha\right]^T E\left[\Delta\alpha\right] = \eta^2 \frac{\partial J}{\partial \alpha}\frac{\partial J}{\partial \alpha}^T = \eta^2 \sum_{i=1}^{N}\left(\frac{\partial J}{\partial \alpha_i}\right)^2,$$

[1] SNR could alternatively be defined as the ratio of the power of the component of the update in the direction of the signal and the component orthogonal to the signal (does not penalize the magnitudes of the updates):

$$\frac{E[a^2]}{E\left[|\Delta\alpha - a\mathbf{v}|^2\right]}, \text{ where } \mathbf{v} = \frac{\frac{\partial J}{\partial \alpha}^T}{\left|\frac{\partial J}{\partial \alpha}^T\right|}, a = \Delta\alpha \cdot \mathbf{v}.$$

This form is probably a better definition, but is more difficult to work with.

$$\begin{aligned}
E\left[(\Delta\alpha)^T(\Delta\alpha)\right] =& \frac{\eta^2}{\sigma_\beta^4} E\left[\left[J^{\alpha+\beta} - J^\alpha\right]^2 \beta^T\beta\right] \\
\approx& \frac{\eta^2}{\sigma_\beta^4} E\left[\left[\frac{\partial J}{\partial \alpha}\beta\right]^2 \beta^T\beta\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} E\left[\left(\sum_i \frac{\partial J}{\partial \alpha_i}\beta_i\right)\left(\sum_j \frac{\partial J}{\partial \alpha_j}\beta_j\right)\left(\sum_k \beta_k^2\right)\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} E\left[\sum_i \frac{\partial J}{\partial \alpha_i}\beta_i \sum_j \frac{\partial J}{\partial \alpha_j}\beta_j \sum_k \beta_k^2\right] \\
=& \frac{\eta^2}{\sigma_\beta^4} \sum_{i,j,k} \frac{\partial J}{\partial \alpha_i}\frac{\partial J}{\partial \alpha_j} E\left[\beta_i\beta_j\beta_k^2\right]
\end{aligned}$$

$$E\left[\beta_i\beta_j\beta_k^2\right] = \begin{cases} 0 & i \neq j \\ \sigma_\beta^4 & i = j \neq k \\ \mu_4(\beta) & i = j = k \end{cases}$$

$$= \eta^2(N-1)\sum_i \left(\frac{\partial J}{\partial \alpha_i}\right)^2 + \frac{\eta^2\mu_4(z)}{\sigma_\beta^4}\sum_i \left(\frac{\partial J}{\partial \alpha_i}\right)^2$$

where $\mu_n(z)$ is the nth 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 17.1 Signal-to-noise ratio for additive Gaussian noise

For β_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 17.2 Signal-to-noise ratio for additive uniform noise

For β_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 σ_β). The calculations in this section seem to imply that larger σ_β can only reduce the variance, overcoming errors or noise in the baseline estimator, $\tilde{b}$; this is a shortcoming 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 σ_β can increase the SNR. The derivation with a second-order Taylor expansion is left for an exercise (Problem 3).

17.3.2 Weight Perturbation with an Estimated Baseline

The weight perturbation update above requires us to evaluate the function J 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 [$J^{\alpha+\beta}$ and J^α], we replace the evaluation of J^α with an estimator, $b = \hat{J}(\alpha)$, obtained from previous trials? In other words, consider an update of the form:

$$\boxed{\Delta\alpha = -\frac{\eta}{\sigma_\beta^2}[J^{\alpha+\beta} - b]\beta} \tag{17.2}$$

The estimator can take any of a number of forms, but perhaps the simplest is based on the update (after the nth trial):

$$b[n+1] = \gamma J[n] + (1-\gamma)b[n], \quad b[0] = 0, 0 \leq \gamma \leq 1,$$

where γ parameterizes the moving average. Let's compute the expected value of the update, using the same Taylor expansion that we used above:

$$\begin{aligned} E[\Delta\alpha] &= -\frac{\eta}{\sigma_\beta^2} E\left[[J^\alpha + \frac{\partial J}{\partial \alpha}\beta - b]\beta\right] \\ &= -\frac{\eta}{\sigma_\beta^2} E\left[J^\alpha - b\right] E\left[\beta\right] + E\left[\beta\beta^T\right] \frac{\partial J}{\partial \alpha}^T \\ &= -\eta \frac{\partial J}{\partial \alpha}^T \end{aligned}$$

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 β, 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. Although it is left as an exercise for the reader (see Problem 2), the results are...

As we will see in a later section, if the evalution of J is stochastic, then the update with a baseline estimator can actually outperform an update using straight function evaluations.

17.4 THE REINFORCE ALGORITHM

The weight perturbation algorithm used only additive noise in the parameters. It also assumed that this noise was small and that J was smoothly differientable. More generally, we can minimize the expected value of a scalar deterministic function $y(\mathbf{z})$, where $\mathbf{z}$ is a vector random variable which depends on α through the conditional probability density function $f_{\mathbf{z}|\alpha}(\mathbf{z}|\alpha)$. Note that $\mathbf{z} = \alpha + \beta$, is a special case of this general form. Consider the gradient of the expected value of y:

$$\begin{aligned}\frac{\partial}{\partial\alpha}E[y] &= \int_{-\infty}^{+\infty} y(\mathbf{z})\frac{\partial}{\partial\alpha}f_{\mathbf{z}|\alpha}(\mathbf{z}|\alpha)dz \\ &= \int_{-\infty}^{+\infty} y(\mathbf{z})f_{\mathbf{z}|\alpha}(\mathbf{z}|\alpha)\frac{\partial}{\partial\alpha}\ln f_{\mathbf{z}|\alpha}(\mathbf{z}|\alpha)dz \\ &= E\left[y(\mathbf{z})\left[\frac{\partial}{\partial\alpha}\ln f_{\mathbf{z}|\alpha}(\mathbf{z}|\alpha)\right]^T\right],\end{aligned}$$

where the second line used the identity $\frac{\partial}{\partial z}\ln(z) = \frac{1}{z}$. Therefore, the update for the REINFORCE algorithm is:

$$\boxed{\Delta\alpha = -\eta\left[y(\mathbf{x}) - b\right]\left[\frac{\partial}{\partial\alpha}\ln f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha)\right]^T} \tag{17.3}$$

which will have the property that

$$E[\Delta\alpha] = -\eta\frac{\partial}{\partial\alpha}E[y].$$

This derivation generalizes our weight perturbation algorithm, removing the assumption on small β and allowing more general forms for $f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha)$. The caveat is that the update is only well-defined for probability densities which are smoothly differentiable in α.

EXAMPLE 17.3 REINFORCE with additive Gaussian noise

When $\mathbf{x} = \alpha + \beta$, $\beta \in N(0, \sigma^2)$, we have

$$f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha) = \frac{1}{(2\pi\sigma^2)^N}e^{\frac{-(\mathbf{x}-\alpha)^T(\mathbf{x}-\alpha)}{2\sigma^2}}$$

$$\ln f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha) = \frac{-(\mathbf{x}-\alpha)^T(\mathbf{x}-\alpha)}{2\sigma^2} + \text{terms that don't depend on } \alpha$$

$$\frac{\partial}{\partial\alpha}\ln f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha) = \frac{1}{\sigma^2}(\alpha - \mathbf{x})^T = \frac{1}{\sigma^2}\beta^T.$$

Therefore, the update

$$\Delta\alpha = -\frac{\eta}{\sigma^2}y(\alpha+\beta)\beta$$

will perform gradient descent (in expectation). Since

$$E[y(\alpha)\beta] = y(\alpha)E[\beta] = \mathbf{0},$$

the update from Equation 17.1 will also perform stochastic gradient descent on y.

17.4.1 Optimizing a stochastic function

To generalize our weight perturbation results, now consider the case where y is not a deterministic function of $\mathbf{x}$, but rather is described by the joint density function:

$$f_{y,\mathbf{x}|\alpha}(y,\mathbf{x}|\alpha) = f_{y|\mathbf{x}}(y|\mathbf{x})f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha).$$

Repeating the recipe above, we have:

$$\begin{aligned}\frac{\partial}{\partial\alpha}E\left[y\right] =& \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} y(\mathbf{x})f_{y|\mathbf{x}}(y|\mathbf{x})\frac{\partial}{\partial\alpha}f_{\mathbf{x}|\alpha}(\mathbf{x},\alpha)dxdy \\ =& \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} y(\mathbf{x})f_{y|\mathbf{x}}(y|\mathbf{x})f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha)\frac{\partial}{\partial\alpha}\ln f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha)dxdy \\ =& E\left[y(\mathbf{x})\left[\frac{\partial}{\partial\alpha}\ln f_{\mathbf{x}|\alpha}(\mathbf{x}|\alpha)\right]^T\right],\end{aligned}$$

Therefore, the weight perturbation algorithm derived still achieves stochastic gradient descent despite stochasticity in y.

17.4.2 Adding noise to the outputs

17.5 EPISODIC REINFORCE

Evaluating our control system many times per trial. Can we do better by performing multiple perturbation "tests" during a single trial? Stochastic processes are harder (but not impossible) to define in continuous time, and our control systems are (mostly) implemented on digital machines, so let's discretize time.

$$E_\alpha = \sum_{n=0}^{N} g(\mathbf{x}[n],\mathbf{u}[n]),$$

subject to

$$\mathbf{u}[n] = \pi_\alpha(\mathbf{x}[n],n), \mathbf{x}[n+1] = f(\mathbf{x}[n],\mathbf{u}[n],n), \mathbf{x}[0] = \mathbf{x}_0.$$

To match our previous derivation, let us rewrite this as

$$E_\alpha(\bar{\mathbf{x}},\bar{\mathbf{u}}) = \sum g(\mathbf{x}[n],\mathbf{u}[n]), \text{ and } P_\alpha(\bar{\mathbf{x}},\bar{\mathbf{u}})$$

where $\bar{\mathbf{x}} = \{x_0, x_1, ...x_N\}$ and $\bar{\mathbf{u}} = \{u_0, u_1, ...u_N\}$. The first task is to compute $\frac{\partial}{\partial\alpha}\log P_\alpha(\bar{\mathbf{x}},\bar{\mathbf{u}})$.

End up with a rule of the form

$$\Delta w = -\eta\sum_{n=0}^{N} g(x,u)\mathbf{e}(k),$$

where

$$\mathbf{e}(0) = 0, \mathbf{e}(k+1) = \left[\frac{\partial}{\partial\alpha}\log P_\alpha(x(k))\right]^T + \mathbf{e}(k).$$

17.6 **INFINITE-HORIZON REINFORCE**

17.7 **LTI REINFORCE**

17.8 **BETTER BASELINES WITH IMPORTANCE SAMPLING**

PROBLEMS

17.1. (MATLAB) *Weight perturbation on a quadratic cost.*

In this problem we will optimize a cost quadratic cost function using weight perturbation.

$$y(\alpha) = \frac{1}{2}\alpha^T \mathbf{A}\alpha, \text{ with } \mathbf{A} = \mathbf{A}^T$$

(a) Simulate the weight perturbation algorithm for $\mathbf{A} = \mathbf{I}_{2\times 2}$. Plot the values of α at each iteration of the algorithm on top of the cost function, y. Your results should resemble figure 17.1, which was made with these parameters. Try using additive Gaussian noise, and additive uniform noise. How does the performance of convergence depend on η? on σ_β^2?

(b) (*Signal-to-noise ratio.*) Estimate the signal-to-noise ratio of your weight pertubation update by holding α fixed and computing $\Delta\alpha$ for as many trials as are required for your SNR statistics to converge. Using $\mathbf{A} = \mathbf{I}_{N\times N}$, make a plot of the SNR for Gaussian and uniform additive noise as a function of N. Compare these plots with the theoretical predictions.

17.2. *Weight perturbation performance with a baseline estimator.*

17.3. *Weight perturbation - optimal noise calculations.*

Use a second-order Taylor expansion...

$$y(\alpha + \beta) = y(\alpha) + \frac{\partial J}{\partial \alpha}\beta + \frac{1}{2}\beta^T \mathbf{h}\beta,$$

where $\mathbf{h}$ is the Hermitian of the function evaluated at α:

$$\mathbf{h} = \frac{\partial}{\partial \alpha}\left[\frac{\partial J}{\partial \alpha}^T\right] = \begin{bmatrix} \frac{\partial^2 y}{\partial \alpha_1 \partial \alpha_1} & \cdots & \frac{\partial^2 y}{\partial \alpha_1 \partial \alpha_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y}{\partial \alpha_N \partial \alpha_1} & \cdots & \frac{\partial^2 y}{\partial \alpha_N \partial \alpha_N} \end{bmatrix}.$$

You may assume that the distribution on β_i is symmetric about the mean, which zeros all odd central-moments starting at $\mu_3(\beta_i) = 0$.

PART THREE

APPLICATIONS AND EXTENSIONS

PART FOUR

APPENDIX

CHAPTER A

Robotics Preliminaries

A.1 DERIVING THE EQUATIONS OF MOTION (AN EXAMPLE)

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:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i.$$

If you are not comfortable with these equations, then any good book chapter on rigid body mechanics can bring you up to speed[1]; for now you can take them as a handle that you can crank to generate equations of motion.

EXAMPLE A.1 Simple Double Pendulum

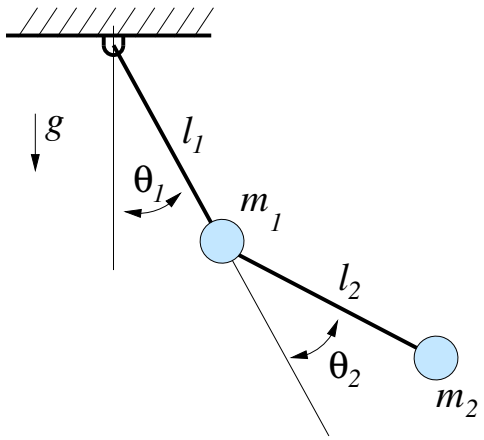


FIGURE A.1 Simple Double Pendulum

Consider the system in Figure A.1 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 $\mathbf{x}_1, \mathbf{x}_2$ to

[1] Try [27] for a very practical guide to robot kinematics/dynamics, [35] for a hard-core dynamics text or [85] for a classical dynamics text which is a nice read

denote the locations of m_1, m_2, respectively, the kinematics of this system are:

$$\begin{aligned}\mathbf{x}_1 = &\begin{bmatrix} l_1 s_1 \\ -l_1 c_1 \end{bmatrix}, \quad \mathbf{x}_2 = \mathbf{x}_1 + \begin{bmatrix} l_2 s_{1+2} \\ -l_2 c_{1+2} \end{bmatrix} \\ \dot{\mathbf{x}}_1 = &\begin{bmatrix} l_1 \dot{q}_1 c_1 \\ l_1 \dot{q}_1 s_1 \end{bmatrix}, \quad \dot{\mathbf{x}}_2 = \dot{\mathbf{x}}_1 + \begin{bmatrix} l_2(\dot{q}_1 + \dot{q}_2) c_{1+2} \\ l_2(\dot{q}_1 + \dot{q}_2) s_{1+2} \end{bmatrix}\end{aligned}$$

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 easily write the kinetic and potential energy:

$$\begin{aligned}T =& \frac{1}{2}\dot{\mathbf{x}}_1^T m_1 \dot{\mathbf{x}}_1 + \frac{1}{2}\dot{\mathbf{x}}_2^T m_2 \dot{\mathbf{x}}_2 \\ =& \frac{1}{2}(m_1 + m_2) l_1^2 \dot{q}_1^2 + \frac{1}{2} m_2 l_2^2 (\dot{q}_1 + \dot{q}_2)^2 + m_2 l_1 l_2 \dot{q}_1 (\dot{q}_1 + \dot{q}_2) c_2 \\ U =& m_1 g y_1 + m_2 g y_2 = -(m_1 + m_2) g l_1 c_1 - m_2 g l_2 c_{1+2}\end{aligned}$$

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:

$$\begin{aligned}&(m_1 + m_2) l_1^2 \ddot{q}_1 + m_2 l_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 l_1 l_2 (2\ddot{q}_1 + \ddot{q}_2) c_2 \\ &\qquad - m_2 l_1 l_2 (2\dot{q}_1 + \dot{q}_2) \dot{q}_2 s_2 + (m_1 + m_2) l_1 g s_1 + m_2 g l_2 s_{1+2} = \tau_1 \\ &m_2 l_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 l_1 l_2 \ddot{q}_1 c_2 + m_2 l_1 l_2 \dot{q}_1^2 s_2 + m_2 g l_2 s_{1+2} = \tau_2\end{aligned}$$

Numerically integrating (and animating) these equations in MATLAB produces the expected result.

A.2 THE MANIPULATOR EQUATIONS

If you crank through the Lagrangian dynamics for a few simple serial chain 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:

$$T = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q}) \dot{\mathbf{q}},$$

where $\mathbf{H}$ is the state-dependent inertial matrix. This abstraction affords some insight into general manipulator dynamics - for example we know that $\mathbf{H}$ is always positive definite, and symmetric[7, p.107].

Continuing our abstractions, we find that the equations of motion of a general robotic manipulator take the form

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

where $\mathbf{q}$ is the state vector, $\mathbf{H}$ is the inertial matrix, $\mathbf{C}$ captures Coriolis forces, and $\mathbf{G}$ captures potentials (such as gravity). The matrix $\mathbf{B}$ maps control inputs $\mathbf{u}$ into generalized forces.

EXAMPLE A.2 Manipulator Equation form of the Simple Double Pendulum

The equations of motion from Example 1 can be written compactly as:

$$\mathbf{H}(\mathbf{q}) = \begin{bmatrix} (m_1+m_2)l_1^2 + m_2l_2^2 + 2m_2l_1l_2c_2 & m_2l_2^2 + m_2l_1l_2c_2 \\ m_2l_2^2 + m_2l_1l_2c_2 & m_2l_2^2 \end{bmatrix}$$

$$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \begin{bmatrix} 0 & -m_2l_1l_2(2\dot{q}_1+\dot{q}_2)s_2 \\ m_2l_1l_2\dot{q}_1s_2 & 0 \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = g\begin{bmatrix} (m_1+m_2)l_1s_1 + m_2l_2s_{1+2} \\ m_2l_2s_{1+2} \end{bmatrix}$$

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 form of the dynamics assumed in equation 1.1.

Bibliography

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the Neural Information Processing Systems (NIPS '07)*, volume 19, December 2006.

[2] Yeuhi Abe and Jovan Popovic. Interactive animation of dynamic manipulation. Technical report, MIT CSAIL, feb 2006.

[3] R. McNeill Alexander. *Optima for Animals*. Princeton University Press, revised edition, 1996.

[4] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 113 – 120. IEEE, 1996.

[5] Araki, N.; Okada, M.; Konishi, Y.; Ishigaki, and H.;. Parameter identification and swing-up control of an acrobot system. *International Conference on International Technology*, 2005.

[6] J.P.Y. Arnould, D.R. Briggs, J.P. Croxall, P.A. Prince, and A.G. Wood. The foraging behaviour and energetics of wandering albatrosses brooding chicks. *Antarctic Science*, 8(3):229–236, 1996.

[7] H. Asada and J.E. Slotine. Robot analysis and control. pages 93–131, 1986.

[8] CG Atkeson and Stefan Schaal. Learning tasks from a single demonstration. *IEEE International Conference on Robotics and Automation (ICRA)*, 2:1706–1712, 1997.

[9] Christopher G. Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. International Conference on Robotics and Automation, 1997.

[10] D.N. Beal, F.S. Hover, M.S. Triantafyllou, J.C. Liao, and G. V. Lauder. Passive propulsion in vortex wakes. *Journal of Fluid Mechanics*, 549:385402, 2006.

[11] Hamid Benbrahim and Judy A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22:283–302, 1997.

[12] Berkemeier, M.D., Fearing, and R.S. Tracking fast inverted trajectories of the underactuated acrobot. *Robotics and Automation, IEEE Transactions on*, 15(4):740–750, Aug 1999.

[13] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[14] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Optimization and Neural Computation Series. Athena Scientific, October 1996.

[15] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Unpublished class notes edition, 2000.

[16] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics, 2001.

[17] A. Bicchi. Hands for dextrous manipulation and robust grasping: a difficult road towards simplicity. *IEEE Transactions on Robotics and Automation*, 16(6):652–662, December 2000.

[18] Antonio Bicchi. On the closure properties of robotic grasping. *International Journal of Robotics Research*, 14(4):319–334, 1995.

[19] Alec Brooks and Paul MacCready. Development of a wing-flapping flying replica of the largest pterosaur. In *AIAA/SAE/ASME/ASEE 21st Joint Propulsion Conference*. AIAA, July 1985.

[20] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, June 1999.

[21] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford Series in Electrical and Computer Engineering. Oxford University Press, third edition, Sept 10 1998.

[22] Stephen Childress. *Mechanics of swimming and flying*. Cambridge University Press, 1981.

[23] Chung Choo Chung and John Hauser. Nonlinear control of a swinging pendulum. *Automatica*, 31(6):851–862, June 1995.

[24] Michael J. Coleman. *A Stability-Study of a Three-Dimensional Passive-Dynamic Model of Human Gait*. PhD thesis, Cornell University, 1998.

[25] Steven H. Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307:1082–1085, February 18 2005.

[26] Steven H. Collins, Martijn Wisse, and Andy Ruina. A three-dimensional passive-dynamic walking robot with two legs and knees. *International Journal of Robotics Research*, 20(7):607–615, July 2001.

[27] John Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley, 2nd edition, January 1989.

[28] Michael H. Dickinson, Fritz-Olaf Lehmann, and Sanjay P. Sane. Wing rotation and the aerodynamic basis of insect flight. *Science*, 284(5422):1954–60, June 1999.

[29] Charles P. Ellington, Coen van den Berg, Alexander P. Willmott, and Adrian L. R. Thomas. Leading-edge vortices in insect flight. *Nature*, 384(19):626–630, December 1996.

[30] Isabelle Fantoni and Rogelio Lozano. *Non-linear Control for Underactuated Mechanical Systems*. Communications and Control Engineering Series. Springer-Verlag, 2002.

[31] Ila Rani Fiete. *Learning and coding in biological neural networks*. PhD thesis, 2003.

[32] IR Fiete, RH Hahnloser, MS Fee, and HS Seung. Temporal sparseness of the premotor drive is important for rapid learning in a neural network model of birdsong. *J Neurophysiol.*, 92(4):2274–82. Epub 2004 Apr 07., Oct 2004.

[33] R.J. Full and D.E. Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of Experimental Biology*, 202(23):3325–3332, 1999.

[34] Hartmut Geyer. *Simple models of legged locomotion based on compliant limb behavior*. PhD thesis, University of Jena, 2005.

[35] Herbert Goldstein. *Classical Mechanics*. Addison Wesley, 3rd edition, 2002.

[36] A. Goswami. Postural stability of biped robots and the foot rotation indicator (FRI) point. *International Journal of Robotics Research*, 18(6), 1999.

[37] Ambarish Goswami, Benoit Thuilot, and Bernard Espiau. Compass-like biped robot part I : Stability and bifurcation of passive gaits. Technical Report RR-2996, INRIA, October 1996.

[38] Kenneth C. Hall and Steven A. Pigott. Power requirements for large-amplitude flapping flight. *Journal of Aircraft*, 35(3), May 1998.

[39] Philip Holmes, Robert J. Full, Dan Koditschek, and John Guckenheimer. The dynamics of legged locomotion: Models, analyses, and challenges. *Society for Industrial and Applied Mathematics (SIAM) Review*, 48(2):207–304, 2006.

[40] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[41] Vanessa Hsu. Passive dynamic walking with knees: A point-foot model. Master's thesis, Massachusetts Institute of Technology, February 2007.

[42] David H. Jacobson and David Q. Mayne. *Differential Dynamic Programming*. American Elsevier Publishing Company, Inc., 1970.

[43] L.E. Kavraki, P. Svestka, JC Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[44] Daniel E. Koditschek and Martin Buehler. Analysis of a simplified hopping robot. *International Journal of Robotics Research*, 10(6):587–605, Dec 1991.

[45] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion, July 2004.

[46] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[47] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.

[48] Andrew L. Kun and W. Thomas Miller, III. Adaptive dynamic balance of a biped robot using neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 240–245, 1996.

[49] Xu-Zhi Lai, Jin-Hua She, Simon X. Yang, and Min Wu. Stability analysis and control law design for acrobots. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May 2006.

[50] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2000.

[51] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[52] Steven M. LaValle, James J. Kuffner, and Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[53] James C. Liao, David N. Beal, George V. Lauder, and Michael S. Triantafyllou. The Kármán gait: novel body kinematics of rainbow trout swimming in a vortex street. *Journal of Experimental Biology*, 206(6):1059–1073, 2003.

[54] Arun D. Mahindrakar and Ravi N. Banavar. A swing-up of the acrobot based on a simple pendulum strategy. *International Journal of Control*, 78(6):424–429, April 2005.

[55] Tad McGeer. Passive dynamic walking. *International Journal of Robotics Research*, 9(2):62–82, April 1990.

[56] Thomas A. McMahon and George C. Cheng. The mechanics of running: How does stiffness couple w/ speed. *Journal of Biomechanics*, 23(Supplement 1):65–78, 1990.

[57] W. Thomas Miller, III. Real-time neural network control of a biped walking robot. *IEEE Control Systems Magazine*, 14(1):41–48, Feb 1994.

[58] Simon Mochon and Thomas A. McMahon. Ballistic walking. *Journal of Biomechanics*, 13:49–57, 1980.

[59] Simon Mochon and Thomas A. McMahon. Ballistic walking: An improved model. *Mathematical Biosciences*, 52(3-4):241–260, December 1980.

[60] Jun Morimoto and Kenji Doya. Reinforcement learning of dynamic motor sequence: Learning to stand up. pages 1721–1726. IEEE/RSJ International, 1998.

[61] Richard M. Murray and John Hauser. A case study in approximate linearization: The acrobot example, April 1991.

[62] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., 1994.

[63] Eadweard Muybridge and Anita Ventura Mozley. *Muybridge's Complete Human and Animal Locomotion: All 781 Plates from the 1887 Animal Locomotion*. 3 Vols. Dover Publications, Incorporated, May 1979.

[64] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, 16, 2003.

[65] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall Incorporated, 3rd edition, August 1996.

[66] Barak Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–9, 1989.

[67] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[68] Marc H. Raibert. *Legged Robots That Balance*. The MIT Press, 1986.

[69] Stefan Schaal and Chris Atkeson. Robot juggling: An implementation of memory-based learning. (14):57–71.

[70] W Schultz. Getting formal with dopamine and reward. *Neuron.*, 36(2):241–63., Oct 10 2002.

[71] H. Sebastian Seung, Daniel D. Lee, 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*, 9:171–85, 2000.

[72] Sebastian Seung. The REINFORCE algorithm. 2001.

[73] Wei Shyy, Yongsheng Lian, Jian Teng, Dragos Viieru, and Hao Liu. *Aerodynamics of Low Reynolds Number Flyers*. Cambridge Aerospace Series. Cambridge University Press, 2008.

[74] Athanasios Sideris and James E. Bobrow. A fast sequential linear quadratic algorithm for solving unconstrained nonlinear optimal control problems, February 2005.

[75] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, October 1990.

[76] M. W. Spong. Underactuated mechanical systems. In B. Siciliano and K. P. Valavanis, editors, *Control Problems in Robotics and Automation*, Lecture notes in control and information sciences 230. Springer-Verlag, 1997.

[77] Mark Spong. The swingup control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1):49–55, February 1995.

[78] Mark W. Spong. Partial feedback linearization of underactuated mechanical systems. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 314–321, September 1994.

[79] Mark W. Spong. Swing up control of the acrobot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2356–2361, 1994.

[80] Mark W. Spong. Energy based control of a class of underactuated mechanical systems. In *Proceedings of the 1996 IFAC World Congress*, 1996.

[81] Mark W. Spong and Gagandeep Bhatia. Further results on control of the compass gait biped. In *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1933–1938, 2003.

[82] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 2nd edition, October 1998.

[83] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books, 1994.

[84] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[85] Stephen T. Thornton and Jerry B. Marion. *Classical Dynamics of Particles and Systems*. Brooks Cole, 5th edition, 2003.

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

[87] Michael S. Triantafyllou and George S. Triantafyllou. An efficient swimming machine. *Scientific American*, 272(3):64, March 1995.

[88] John Tsitsiklis and Ben Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

[89] Vance A. Tucker. Gliding flight: Speed and acceleration of ideal falcons during diving and pull out. *Journal of Experimental Biology*, 201:403–414, Nov 1998.

[90] Eric D. Tytell and George V. Lauder. Hydrodynamics of the escape response in bluegill sunfish, lepomis macrochirus. *The Journal of Experimental Biology*, 211:3359–3369, 2008.

[91] Nicolas Vandenberghe, Stephen Childress, and Jun Zhang. On unidirectional flight of a free flapping wing. *Physics of Fluids*, 18, 2006.

[92] Nicolas Vandenberghe, Jun Zhang, and Stephen Childress. Symmetry breaking leads to forward flapping flight. *Journal of Fluid Mechanics*, 506:147–155, 2004.

[93] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[94] Xin Xin and M. Kaneda. New analytical results of the energy based swinging up control of the acrobot. In *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC)*, volume 1, pages 704 – 709. IEEE, Dec 2004.

[95] Jun Zhang, Stephen Childress, Albert Libchaber, and Michael Shelley. Flexible filaments in a flowing soap film as a model for one-dimensional flags in a two-dimensional wind. *Nature*, 408(6814):835–838, December 2000.

[96] Kemin Zhou and John C. Doyle. *Essentials of Robust Control*. Prentice Hall, 1997.