Applied Optimal Control for Dynamically Stable Legged Locomotion

by

Russell L. Tedrake

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author .....
Department of Electrical Engineering and Computer Science
September 8, 2004

Certified by .....
H. Sebastian Seung
Professor, Department of Brain & Cognitive Sciences and Department of Physics
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Applied Optimal Control for
Dynamically Stable Legged Locomotion

by

Russell L. Tedrake

Submitted to the Department of Electrical Engineering and Computer Science
on September 8, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Online learning and controller adaptation will be an essential component for legged robots in the next few years as they begin to leave the laboratory setting and join our world. I present the first example of a learning system which is able to quickly and reliably acquire a robust feedback control policy for 3D dynamic bipedal walking from a blank slate using only trials implemented on the physical robot. The robot begins walking within a minute and learning converges in approximately 20 minutes. The learning works quickly enough that the robot is able to continually adapt to the terrain as it walks. This success can be attributed in part to the mechanics of our robot, which is capable of stable walking down a small ramp even when the computer is turned off.

In this thesis, I analyze the dynamics of passive dynamic walking, starting with reduced planar models and working up to experiments on our real robot. I describe, in detail, the actor-critic reinforcement learning algorithm that is implemented on the return map dynamics of the biped. Finally, I address issues of scaling and controller augmentation using tools from optimal control theory and a simulation of a planar one-leg hopping robot. These learning results provide a starting point for the production of robust and energy efficient walking and running robots that work well initially, and continue to improve with experience.

Thesis Supervisor:

H. Sebastian Seung
Professor, Department of Brain & Cognitive Sciences and Department of Physics

Thesis Committee:

Emilio Bizzi
Institute Professor, Massachusetts Institute of Technology

Leslie P. Kaelbling
Professor, Department of Electrical Engineering and Computer Science

Jovan Popovic
Assistant Professor, Department of Electrical Engineering and Computer Science

Jean-Jacques E. Slotine
Professor, Department of Mechanical Engineering & Information Sciences and
Department of Brain & Cognitive Sciences

Acknowledgments

First and foremost, I would like to thank a number of outstanding MIT undergraduates that I have been able to work with thanks to the MIT Undergraduate Research Opportunities Program (UROP). Ming-fai Fong worked on the mechanical design and construction of our first passive walker and the first version of our actuated walker. Derrick Tan made the robot autonomous for the first time by building the computer and batteries onto the robot. Andrew Baines is currently building the kneed version of the robot. Each of them has contributed immeasurably to the success of this project. In particular, I want to thank Teresa Weirui Zhang, a mechanical engineer who worked on this project for over a year. Teresa is largely responsible for the current mechanical design of the robot, which is in every way superior to it's predecessors.

I want to thank a number of my friends and colleagues that I have learned from and collaborated with over the last few years. Andrew Richardson, Max Berniker, and Dan Paluska have been a constant source for new ideas and for feedback on existing ideas. Matt Malchano and I talked about building an actuated version of a passive walker on the weekends, just because we thought it would be a fun project. It turned out to be the idea that fueled this thesis. Jerry Pratt has continued to give me a lot of great feedback over the phone, even now that he has graduated and moved into a faculty position of his own. Alan Chen helped with the linux installations on the robots. Josh Presseisen, who I originally met on the commuter rail, has been helping me design and build a cool body for the robot. I also want to thank all of the members of the Seung Lab, the former Leg Lab, the new Biomechatronics Lab, and the Bizzi Lab. These outstanding labs provided a community for me that was supportive, friendly, and constructive.

I've also received guidance and support from a number of faculty. First I want to thank my thesis committee: Emilio Bizzi, Leslie Kaelbling, Jovan Popovic, and Jean-Jacques Slotine. Their time and their feedback on this document was greatly appreciated. I also want to thank Gill Pratt, Hugh Herr, and Neville Hogan for the feedback and support on numerous occasions throughout my graduate career.

A very humble and special thank you goes to my advisor, Sebastian. Over the last 4 1/2 years, Sebastian has given me the support and the independence that I needed to thrive and taught me many of the mathematical tools that I missed out on during my undergraduate career as a computer programmer. The day that he suggested that we could buy some table-top machining equipment for the lab changed my entire research direction, and changed my life.

Most importantly, I want to thank my wife, Rachel, who has always been there for me. This thesis is dedicated to her.

Contents

List of Figures

Chapter 1

Introduction

The study of legged locomotion is an interdisciplinary endeavor, bringing together research from biomechanics, neuroscience, control theory, mechanical design, and artificial intelligence. It includes efforts to understand and to rehabilitate human and animal locomotion, as well as to replicate human and animal locomotion in machines. The focus of this thesis is on building legged robots, but many of the ideas contained here were inspired by what I have learned about biological motor control and motor learning.

Legged locomotion takes many forms. An insect or crustacean walking with six or more legs must solve a very different problem than a human walking with just two[1]. Besides the number of legs, the primary difference between these gaits is the issue of balance. Locomotion researchers distinguish between gaits that are *statically stable* and gaits that are *dynamically stable*. The *support polygon* is defined as the area on the ground which creates a bounding box around all of the contacts between the feet and the ground. In a statically stable gait, the center of mass of the robot is always directly over the support polygon. Consequently, without considering joint velocities or external forces, we would expect the animal or machine to always be in a balanced state. Hexapods can walk efficiently using a statically stable gait, but a statically stable gait for a biped is extremely slow. When humans walk, we use a dynamically stable gait: our gait is stable, but the ground projection of our center of mass occasionally leaves the support polygon.

1.1 The State-of-the-Art

Controlling a dynamically stable gait on a legged robot has proven to be a very challenging control problem. As the ground projection of the center of mass (CoM) leaves the support polygon it becomes increasingly difficult to apply corrective forces to the ground which accelerate the CoM of the robot back toward the statically stable configuration without having the feet rotate or slip. Imagine putting both of your feet together and leaning forward - you don't have to go very far before your feet roll up onto your toes and you begin to fall.

It is possible to measure how close we are to this critical point when our feet start rolling by looking at the ground reaction forces. The *center of pressure* (CoP) is defined as the weighted average of the ground reaction forces. Relative to any point, x, on the foot, the

[1] [Muybridge and Mozley, 1979] is an excellent pictorial reference of the huge variety of gaits in birds and mammals.

CoP is located at:

$$x_{cop} = \frac{\int_{x'}(x' - x)F(x')dx'}{\int_{x'} F(x')dx'}.$$

As we lean forward, the CoP moves from the middle of the foot toward the front of the foot. As soon as it reaches the front edge of the foot, the foot begins to roll.

The most successful bipedal robot to date is Honda Motor Company's humanoid robot, ASIMO. ASIMO's control system is based on tracking a trajectory which maintains the location of the CoP near the middle of the foot. This ensures that the robot can apply corrective ankle torques as it walks. Combined with a superb mechanical design and a traditional high gain adaptive trajectory tracking controller, this algorithm allows ASIMO to walk across a flat factory floor or even up stairs.

Despite these impressive accomplishments, this state-of-the-art robotic walking cannot compare to a human in measures of speed, efficiency, or robustness. ASIMO's top walking speed is 0.44 m/s [Honda Motor Co., 2004] and it cannot run, whereas humans don't even transition from walking to running until about 2.0 m/s [Alexander, 1996]. The gold medalist in the 100m sprint at the 2004 Summer Olympics in Athens finished in 9.85 seconds, averaging over 10 m/s; the marathon gold medalist in the same Olympics finished 26.2 miles in 2:10:55, averaging well over 5 m/s. ASIMO's large 38.4 volt, 10 amp-hour battery pack powers the robot for only 26 minutes [Honda Motor Co., 2004], which is roughly 20 times as inefficient as human walking [Collins et al., 2004]. There are no demonstrations of ASIMO walking on uneven or unmodelled terrain.

ASIMO's performance is limited because it is commanded to follow trajectories that conform to an overly restrictive measure of dynamic balance. ASIMO always walks with flat feet, but when humans walk, our CoP *does* reach the edge of our foot, and our feet *do* roll. The situation is even more exaggerated when we jog or run - no joint torques that we apply during the aerial phase will alter the ballistic trajectory of our CoM. What ASIMO is missing is the ability to plan a feasible trajectory through the uncontrolled portion of state space with the knowledge that control will be regained at the other side.

1.2 Machine Learning and Optimal Control

In lieu of good analytical solutions to the control problem, many researchers have turned to machine learning algorithms to *automatically* acquire a good control solution. These algorithms are normally formulated as an optimal control problem: the goal of control is the long-term optimization of a scalar cost function. This formulation is broad enough to capture (and potentially solve) the trajectory prediction problem that limits ASIMO's performance. Another major advantage of using machine learning is that the learning process can continue as the robot experiences changes in terrain or changes in dynamics.

Despite the promise, there are very few examples of learning systems that actually work on a real legged robot, and even fewer examples of a learning controller outperforming a non-learning one (see Chapter 3 for a review). Dynamic bipedal walking is difficult to learn for a number of reasons. First, walking robots typically have many degrees of freedom, which can cause a combinatorial explosion for learning systems that attempt to optimize performance in every possible configuration of the robot. Second, details of the robot dynamics such as uncertainties in the ground contact and nonlinear friction in the joints are difficult to model in simulation, making it unlikely that a controller optimized in a simulation will perform optimally on the real robot. Since it is only practical to run a small number of learning trials

on the real robot, the learning algorithms must perform well after obtaining a very limited amount of data. Finally, learning algorithms for dynamic walking must deal with dynamic discontinuities caused by collisions with the ground and with the problem of delayed reward - motor torques applied at one moment can affect the trajectory of the robot far into the future.

In order to successfully apply learning to a real bipedal robot, I built a robot that minimizes many of these difficulties. The robot pictured in Figure 1-1 is capable of dynamic walking, but has a total of only nine degrees of freedom and only four actuators. Furthermore, the mechanical design of this robot has been optimized so that the robot is actually capable of stable walking down a small ramp even when the computer is turned off. This dramatically improves the performance of the learning system by causing the robot to follow reasonable trajectories even when the learning policy is initialized to a blank slate.

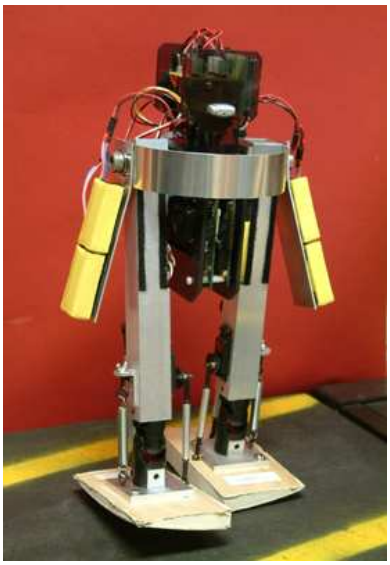

Figure 1-1: The Toddler robot

1.3 Passive Dynamic Walking

The mechanical design of this robot is based on the concept of passive dynamic walking. In the late 1980's, Tad McGeer [McGeer, 1990] introduced this class of walking robots that are capable of walking stably down a small decline without the use of any motors. Energy losses in these machines due to friction and collisions when the swing leg returns to the ground are balanced by the gradual conversion of potential energy into kinetic energy as the walker moves down the slope. The most impressive passive dynamic walker to date [Collins et al., 2001] has knees and arms, and walks with a surprisingly anthropomorphic gait. These machines provide an elegant illustration of how proper machine design can generate stable and potentially very energy efficient walking.

It wasn't until very recently that researchers have been able to harness the principles from passive dynamic walking to build actuated robots capable of walking on flat terrain. It was not clear how to design a control system around a machine that already solved a portion of the control problem in the dynamics. The robot in Figure 1-1 was designed by

adding a small number of actuators to a passive dynamic walker, and the control system was designed using machine learning. This turned out to be a winning combination; the robot achieves robust and efficient walking on a variety of terrains after only a few minutes of training on the physical robot.

1.4 Contributions and Organization of Chapters

The next chapter is a review of the analytical and numerical tools available for optimal control. Chapter 3 reviews legged robot control, with a focus on machine learning approaches, and presents a tool that will be used extensively throughout the thesis: step-to-step return maps.

In chapter 4, I present a thorough analysis of the most basic passive walking model: the rimless wheel. This analysis includes the first proof that the set of fixed points on the return map are globally exponentially stable, except for a set of measure zero. I then review the compass gait model, and work up to a model of my own 3D passive dynamic walker. This model provides a tool for designing the curvature of the feet for an experimental machine to produce an elegant gait with a desired step length and step frequency. The chapter concludes with the first experimental stability analysis of a physical passive walker.

In chapter 5, I present Toddler, the 3D actuated robot in Figure 1-1 which is based on the passive walking design from chapter 4. This is one of the first three robots, developed independently at the same time at different universities, capable of both passive walking down a ramp and powered walking on flat terrain [Collins et al., 2004]. This is a milestone that has been anticipated since the late 1980's, when Tad McGeer first presented his passive walking machines. This simplified robot captures the essence of dynamic walking but minimizes many of the complications associated with many degree-of-freedom systems.

Using the Toddler robot, I present the first learning system which is able to quickly and reliably acquire a robust feedback control policy for 3D dynamic walking from a blank slate using only trials implemented on the physical robot. I derive the actor-critic algorithm used on the robot, and highlight many of the tricks that allowed us to achieve such good performance. These include formulating the learning problem on the discrete return map dynamics and reducing the dimensionality through manual controller decomposition.

In chapter 6, I address the issues of scaling and controller augmentation using a simulation of a running robot: Marc Raibert's planar one-leg hopper [Raibert, 1986]. The original control system for the hopper is an elegant, intuitive controller which works well when the robot is hopping at a nearly steady state hopping trajectory, but which is unable to recover from large disturbances or awkward initial conditions. The derivation of this controller relied on a number of simplifying assumptions which can be interpreted as approximations to the optimal control solution. By applying a policy gradient reinforcement learning algorithm to a simulation of this robot, I have augmented the controller to dramatically increase the robot's region of stability. This work provides a fresh perspective on a famous control algorithm and a mechanism for systematically improving upon it. It also demonstrates the applicability of these learning techniques to a variety of legged robots.

Finally, I conclude in chapter 8 with a list of ongoing and future projects and a broad discussion of the implications of optimal control in the next few years of walking robots.

Overall, the thesis contributes a novel analysis of some of the simplest passive dynamic walking models and provides clear demonstrations of learning on two of the simplest legged locomotion systems, one for walking and one for running. The choice of these systems

allowed me to focus on the essence of the learning problem, instead of dealing with robot-specific implementation details. The examples demonstrate the power of using simulation, analytical methods, and real robots together to effectively study the problem of locomotion, and lay the ground-work for augmenting real robots with a learning component.

Chapter 2

Applied Optimal Control

2.1 The Optimal Control Problem

Consider a controllable, discrete time, dynamical system

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n),$$

where $\mathbf{x}_n$ represents the state of the system at time n, and $\mathbf{u}_n$ represents the control inputs. More generally, we can allow the system to be stochastic and write it as a Markov decision process (MDP):

$$Pr\{\mathbf{X}_{n+1} = \mathbf{x}_{n+1} | \mathbf{X}_n = \mathbf{x}_n, \mathbf{U}_n = \mathbf{u}_n\} = F(\mathbf{x}_{n+1}, \mathbf{x}_n, \mathbf{u}_n),$$

where F is a probability density function over the state $\mathbf{x}_{n+1}$.

The goal of optimal control is to specify a metric which evaluates the performance of the controlled system, and then to find the control inputs at each time, $\mathbf{u}_n$, which optimize this performance. The optimal control problem can either be formatted as *finite-horizon* or *infinite-horizon*.

2.1.1 Finite-Horizon Problems

Define an instantaneous scalar cost $g(\mathbf{x}, \mathbf{u}, n)$ and terminal cost $h(\mathbf{x})$. The *finite-horizon* cost is given by

$$C = h(\mathbf{x}_N) + \sum_{n=0}^{N-1} g(\mathbf{x}_n, \mathbf{u}_n, n),$$

subject to the dynamics given some distribution of initial conditions $P\{\mathbf{x}_0\}$. For our purposes, I will always assume that g and h are self-administered measures of cost, and that they are deterministic and known. If the control inputs are generated by a deterministic[1] policy

$$\mathbf{u}_n = \pi(\mathbf{x}_n, n),$$

[1]Often times, the optimal control action at each time is not unique, in which case a policy which selects stochastically among the optimal actions is still optimal. In partially observable domains, or in game theory where you have an intelligent opponent, it may actually be possible that a stochastic policy outperforms a deterministic policy. For the problems that we consider in this thesis, deterministic policies are sufficient.

then the expected *cost-to-go function*, or *value function*, is given by

$$J^{\pi}(\mathbf{x}, n) = E\left\{ h(\mathbf{X}_N) + \sum_{m=n}^{N-1} g(\mathbf{X}_m, \mathbf{U}_m, m) \Big| \mathbf{x}_n = \mathbf{x}, \mathbf{u}_m = \pi(\mathbf{x}_m, m) \right\}.$$

This expression can be written recursively as

$$\begin{aligned} J^{\pi}(\mathbf{x}, N) &= h(\mathbf{x}) \\ J^{\pi}(\mathbf{x}, n) &= g(\mathbf{x}, \pi(\mathbf{x}, n), n) + E\left\{ J^{\pi}(\mathbf{X}_{n+1}, n+1) \Big| \mathbf{x}_n = \mathbf{x}, \mathbf{u}_n = \pi(\mathbf{x}_n, n) \right\} \\ &= g(\mathbf{x}, \pi(\mathbf{x}, n), n) + \int_{\mathbf{x}'} J^{\pi}(\mathbf{x}', n+1) F(\mathbf{x}', \mathbf{x}, \pi(\mathbf{x})) d\mathbf{x}'. \end{aligned}$$

The goal of optimal control is to find the optimal policy, π^*, which minimizes the cost-to-go. Therefore [Bertsekas, 2000], the optimal cost-to-go is

$$\begin{aligned} J^*(\mathbf{x}, N) &= h(\mathbf{x}) \\ J^*(\mathbf{x}, n) &= \min_u \left[g(\mathbf{x}, \mathbf{u}, n) + \int_{\mathbf{x}'} J^*(\mathbf{x}', n+1) F(\mathbf{x}', \mathbf{x}, \mathbf{u}) d\mathbf{x}' \right]. \end{aligned}$$

and the optimal policy is

$$\pi^*(\mathbf{x}, n) = \arg\min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}, n) + \int_{\mathbf{x}'} J^*(\mathbf{x}', n+1) F(\mathbf{x}', \mathbf{x}, \mathbf{u}) d\mathbf{x}' \right].$$

These conditions on J^* and π^* are necessary and sufficient conditions for optimality; if we find a policy and corresponding cost-to-go function which satisfy these equations, then that policy is optimal.

An example of a finite-horizon problem is the execution of point-to-point reaching movements as studied in Emilio Bizzi's lab (i.e., [Li et al., 2001]). These experiments are often done with rhesus-macaque monkeys, which are given drops of juice as reward if they move a manipulandum from point A to point B in a specified amount of time. The monkey learns what motor commands (muscle activations) it needs to execute in order to maximize the reward (apple juice).

Another interesting example of a finite-horizon problem is the trajectory learning problem for neural networks. Consider the dynamical system

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n) = \tanh(\mathbf{u}_n),$$

and the parameterized control policy

$$\mathbf{u}_n = \pi_{\mathbf{W},\mathbf{b}}(\mathbf{x}_n, n) = \mathbf{W}\mathbf{x}_n + \mathbf{b}.$$

This is a common discrete time representation of a neural network. The trajectory learning problem is formulated [Pearlmutter, 1995, Williams and Zipser, 1989] as an optimal control problem where the goal is to find the $\mathbf{W}$ and $\mathbf{b}$ to minimize the cost

$$g(\mathbf{x}_n, \mathbf{u}_n, n) = \frac{1}{2}|\mathbf{x}_n - \mathbf{x}_n^d|^2, \text{ and } h(\mathbf{x}_N) = \frac{1}{2}|\mathbf{x}_N - \mathbf{x}_N^d|^2.$$

The "Backprop-Through-Time" algorithm [Pearlmutter, 1995] is an example of a policy-

gradient algorithm, which will be discussed section 2.3.

2.1.2 Infinite-Horizon Problems

In the infinite-horizon formulation, discounted formulation, we define a discount factor $0 \leq \gamma \leq 1$ and define the infinite horizon cost as

$$C = \sum_{n=0}^{\infty} \gamma^n g(\mathbf{x}_n, \mathbf{u}_n, n),$$

subject to the dynamics and some distribution on the initial conditions $\mathbf{x}_0$. For infinite-horizon problems, if the instantaneous cost is not a function of time and can be written as $g(\mathbf{x}_n, \mathbf{u}_n)$, then the optimal policy does not depend on time [Bertsekas, 2000], and can be written[2]

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

This property makes the infinite-horizon formulation more elegant, and is one of the primary reasons that we use it in this thesis. The cost-to-go is

$$\begin{aligned} J^{\pi}(\mathbf{x}) &= E\left\{ \sum_{n=0}^{\infty} \gamma^n g(\mathbf{X}_n, \mathbf{U}_n) \Big| \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_n = \pi(\mathbf{x}_n) \right\} \\ &= g(\mathbf{x}, \pi(\mathbf{x})) + \gamma \int_{\mathbf{x}'} J^{\pi}(\mathbf{x}') F(\mathbf{x}', \mathbf{x}, \pi(\mathbf{x})) d\mathbf{x}', \end{aligned}$$

and the optimal cost-to-go is

$$J^*(\mathbf{x}) = \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \gamma \int_{\mathbf{x}'} J^*(\mathbf{x}') F(\mathbf{x}', \mathbf{x}, \mathbf{u}) d\mathbf{x}' \right].$$

In the infinite-horizon, average reward formulation, the cost is written as

$$C = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} g(\mathbf{x}_n, \mathbf{u}_n),$$

subject to the dynamics and initial conditions on $\mathbf{x}$. The cost-to-go is

$$J^{\pi}(\mathbf{x}) = E\left\{ \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} g(\mathbf{X}_n, \mathbf{U}_n) \Big| \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_n = \pi(\mathbf{x}_n) \right\}$$

The classic example of an infinite-horizon problem is investing in the stock market. For robots, the infinite-horizon formulation allows us to reward or penalize a robot for accomplishing task without placing strict bounds on the time allowed for completion. Legged locomotion, including both walking and running, falls nicely into this formulation.

[2]For finite-horizon problems, the policy still depends on time even if the instantaneous cost does not.

2.1.3 Continuous Time Optimal Control

The optimality conditions for both finite and infinite horizon problems can be extended to the continuous time case. For the deterministic system,

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

the optimal cost-to-go must satisfy:

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

or

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

This famous equation is called the Hamilton-Jacobi-Bellman (HJB) equation. The optimal policy is simply:

$$\pi^*(\mathbf{x}, t) = \arg\min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t} \right].$$

Recall that for infinite-horizon problems, the dependence on t drops out, further simplifying the optimality conditions.

The HJB for the discounted case is a little more complicated. We define the continuous time, infinite-horizon, discounted cost as

$$C = \int_0^\infty e^{-\frac{t' - t}{\tau}} g(\mathbf{x}, \mathbf{u}) dt.$$

For this problem, the optimality conditions [Doya, 1999] are:

$$\begin{aligned} \frac{1}{\tau} J^*(\mathbf{x}) &= \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \right] \\ \pi^*(\mathbf{x}) &= \arg\min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \right] \end{aligned}$$

On most robots, the control equations are evaluated at discrete intervals, but these intervals may not be evenly spaced in time[3]. Understanding the continuous time optimal control equations allows us to generalize the discrete time updates to handle a variable amount of time between updates.

[3]On my robots, the control equations are evaluated as quickly as possible, and the interval between updates depends primarily on the CPU load.

2.1.4 Solving the Optimal Control Problem

In the next few sections I will quickly describe a number of tools for solving optimal control problems. For some simple systems, such as linear systems with quadratic cost functions, it is possible to find an optimal policy analytically. For more complicated systems, we must resort to computational tools that attempt to approximate an optimal control solution. A notable subset of these algorithms are the "reinforcement learning" algorithms, which attempt to solve the optimal control problem using trial and error. Reinforcement learning algorithms have the advantage that they can be applied to online learning on a real robot.

There are three fundamentally different approaches taken in reinforcement learning algorithms. In *policy gradient* algorithms, the control policy is parameterized by some vector $\mathbf{w}$, and the gradient $\frac{\partial}{\partial \mathbf{w}} E\{C\}$ is approximated directly and used to perform gradient descent. Policy gradient methods have become popular again recently because it is possible to prove convergence to a locally optimal policy. Unfortunately, they often suffer from slow convergence rates.

In the second approach, the policy is not represented explicitly. Instead, these *value function* algorithms attempt to learn the cost-to-go function, J. As we have seen, once we have the cost-to-go function, the policy is completely determined. These methods are potentially more efficient than policy gradient methods, but in general they are not guaranteed to convergence to even a locally optimal policy[4]. In fact, there are illustrations of the basic algorithms failing to converge on very simple problems.

The third approach combines the policy gradient methods with the value function methods, and are called *actor-critic* methods. These methods attempt to combine the strengths of each of the previous approaches: efficient learning and provable convergence.

2.2 Analytical Methods

For some systems, it may be possible to solve for, or at least approximate, the optimal controller analytically. The most famous example of this is the linear-quadratic regulator (LQR). Consider the linear system

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

and the quadratic cost function

$$g(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u},$$

where $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$ are constant linear matrices and $\mathbf{R}$ is symmetric and invertible. In the finite-horizon formulation, the HJB is

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

Try a solution for J^* with the form

$$J^*(\mathbf{x}, t) = \mathbf{x}^T \mathbf{K}(t) \mathbf{x},$$

[4]They *are* known to converge for problems with full-observable, finite, state and action spaces

where $\mathbf{K}$ is also symmetric. Then optimal policy would be

$$\begin{aligned}\pi(\mathbf{x}) &= \arg\min_{\mathbf{u}} \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + 2\mathbf{x}^T\mathbf{K}(t)(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \mathbf{x}^T\dot{\mathbf{K}}(t)\mathbf{x}\right] \\ &= -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{K}(t)\mathbf{x}\end{aligned}$$

Substituting $\mathbf{u}$ back into the HJB, we have

$$0 = \mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{x}^T\mathbf{K}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{K}(t)\mathbf{x} + 2\mathbf{x}^T\mathbf{K}(t)\mathbf{A}\mathbf{x} + \mathbf{x}^T\dot{\mathbf{K}}(t)\mathbf{x}.$$

With terminal cost $h(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x}$, this yields:

$$\begin{aligned}\mathbf{K}(T) &= \mathbf{Q} \\ \dot{\mathbf{K}}(t) &= -\mathbf{K}(t)\mathbf{A} - \mathbf{A}^T\mathbf{K}(t) + \mathbf{K}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{K}(t) - \mathbf{Q}\end{aligned}$$

For the infinite-horizon case, $\mathbf{K}(t)$ is a steady state solution to the equation above, which can be solved for $\mathbf{K}$ with some algebraic tricks. The linear quadratic regulator with Gaussian noise (LQRG) is an extension of LQR control to stochastic systems. Although most real-world systems are not linear, the infinite-horizon LQR and LQRG gain matrices are widely used in control system applications where it is possible to linearize a system around the set point. This is an important idea - performing *approximate* optimal control by acting optimally on a simplified model of the dynamics of the system.

2.3 Policy Gradient Methods

In policy gradient methods, we constrain the policy $\pi(\mathbf{x}, \mathbf{u})$ to come from some class of functions parameterized by a vector $\mathbf{w}$, which we will write $\pi_{\mathbf{w}}(\mathbf{x}, \mathbf{u})$. Common choices for the function class include neural networks, radial basis functions, and tabular function approximators. Our task is to compute the gradient

$$\frac{\partial}{\partial \mathbf{w}} E\{C(\mathbf{w})\},$$

and then perform standard gradient descent on the policy parameters.

We know the equations for the parametric form of the policy exactly. In some cases we know the dynamics of the environment exactly, and we can compute the policy gradient analytically. For the finite-horizon case, the gradient can be written as

$$\begin{aligned}\frac{\partial}{\partial w_i} E\{C(\mathbf{w})\} =& E\left\{\sum_{n=0}^{N} \frac{\partial C}{\partial \mathbf{x}_n}\frac{\partial \mathbf{x}_n}{\partial w_i} + \frac{\partial C}{\partial \mathbf{u}_n}\frac{\partial \mathbf{u}_n}{\partial w_i}\right\} \\ =& E\left\{\sum_{n=0}^{N} \frac{\partial C}{\partial \mathbf{x}_n}\left[\frac{\partial \mathbf{x}_n}{\partial w_i} + \frac{\partial \mathbf{x}_n}{\partial \mathbf{x}_{n-1}}\frac{\partial \mathbf{x}_{n-1}}{\partial w_i} + \frac{\partial \mathbf{x}_n}{\partial \mathbf{u}_{n-1}}\frac{\partial \mathbf{u}_{n-1}}{\partial w_i}\right] + \frac{\partial C}{\partial \mathbf{u}_n}\frac{\partial \mathbf{u}_n}{\partial w_i}\right\}\end{aligned}$$

and so on. This chain rule computation can be done efficiently using a forward and a backward pass using the BackPropagation-Through-Time [Pearlmutter, 1989], or using only a forward pass using Real-Time Recurrent Learning (RTRL) [Williams and Zipser, 1989].

In cases where we do not know the gradient of the dynamics of the environment, or if we don't believe that this complicated gradient calculation is biologically plausible, then we

can estimate the policy gradient numerically. Suppose that we add small Gaussian random vector $\mathbf{z}$, with $E\{z_i\} = 0$ and $E\{z_i z_j\} = \sigma^2 \delta_{ij}$, to $\mathbf{w}$ and compute the following value

$$\alpha(\mathbf{w}) = [C(\mathbf{w} + \mathbf{z}) - C(\mathbf{w})]\mathbf{z}.$$

For small $\mathbf{z}$, we can do a Taylor expansion to see that $\alpha(\mathbf{w})$ is a sample of the desired gradient:

$$\begin{aligned} E\{\alpha_i(\mathbf{w})\} =& E\left\{ [C(\mathbf{w}) + \sum_j \frac{\partial C}{\partial w_j} z_j - C(\mathbf{w})] z_i \right\} \\ =& E\left\{ z_i \sum_j \frac{\partial C}{\partial w_j} z_j \right\} = \frac{1}{\sigma^2} \frac{\partial}{\partial w_i} E\{C(\mathbf{w})\} \end{aligned}$$

The gradient descent update using this estimate of the gradient,

$$\Delta w_i = -\eta \alpha(\mathbf{w}),$$

does not necessarily move in the direction of the true gradient on every step, but can be shown to move in the direction of the true gradient on average:

$$E\{\Delta w_i\} \propto \frac{\partial}{\partial w_i} E\{C(\mathbf{w})\}.$$

Therefore, for small η, the algorithm is guaranteed to converge to a local minimum [5]. This algorithm, called "weight perturbation", and others like it (i.e., William's REINFORCE [Williams, 1992], [Baxter and Bartlett, 2001]), all try to estimate the gradient online and perform gradient descent on average.

The strength of the policy gradient methods is this guaranteed convergence to a locally optimal policy. Their major weakness is that it can be difficult to calculate the performance gradient of a real system analytically, and methods based on numerical estimates of the gradient can require a very large number of samples to acquire an accurate estimate.

2.4 Value Function Methods

Policy gradient methods are naive in the sense that they do not make use of some basic structure underlying the reinforcement learning problem. This structure is based on the *principle of optimality*, which can be stated as

> Let $\pi^* = [\pi(\mathbf{x}_0, 0), \pi(\mathbf{x}_1, 1), ..., \pi(\mathbf{x}_N, N)]$ be an optimal policy for the N-step finite horizon problem. Now consider the subproblem where we are at $\mathbf{x}'$ at time i and wish to minimize the cost from time i to time N:
>
> $$E\left\{ \sum_{n=i}^{N} g(\mathbf{x}_n, \mathbf{u}_n, n) | \mathbf{x}_i = \mathbf{x}' \right\}.$$

[5] Technically, C and the second derivatives of C with respect to w must be bounded, and η must be decreased to 0 to obtain true convergence [Bertsekas and Tsitsiklis, 1996, section 3.2].

The truncated policy $[\pi(\mathbf{x}_i, i), ..., \pi(\mathbf{x}_N, N)]$ is optimal for this subproblem.

The intuitive justification of this principle is that if we could find a better policy for the subproblem, then we could have also found a better policy for the complete problem. The principle of optimality tells us that it is possible to solve the complete reinforcement learning problem by iteratively solving more and more complicated subproblems.

2.4.1 Dynamic Programming

Dynamic programming is an efficient algorithm for solving the finite-horizon optimal control problem when you have finite states and finite actions, and a perfect model of the dynamics of the environment. It uses the principle of optimality to compute the optimal cost-to-go recursively backward, starting from the terminal cost:

$$\begin{aligned} J^*(\mathbf{x}, N) =& h(\mathbf{x}_N) \\ J^*(\mathbf{x}, n) =& \min_{\mathbf{u}_n} \left[g(\mathbf{x}_n, \mathbf{u}_n, n) + E\{J^*(\mathbf{X}_{n+1}, n+1) \Big| \mathbf{x}_n = \mathbf{x}\} \right] \end{aligned}$$

With a finite state and action space, this minimization can be done by searching over all possible $\mathbf{u}_n$ and $\mathbf{x}_{n+1}$.

To actually solve this problem, the algorithm needs to find the cost-to-go for every $\mathbf{x}$ before proceeding to the previous n. This can be computationally prohibitive as the state space becomes large, a problem known as the "curse of dimensionality". It is not a true reinforcement learning algorithm, as it not based on trial and error data and therefore is not suitable for online learning on a robot.

Unlike policy gradient methods, dynamic programming does find a *globally optimal* policy. For problems with small, finite state and action spaces, it is actually quite efficient; it solves the problem without repeating any computations as a policy gradient method would.

2.4.2 Value Iteration

The value iteration algorithm extends the basic dynamic programming algorithm to infinite horizon problems. The algorithm begins with an arbitrary initial estimate $\hat{J}(\mathbf{x})$ of J^*, which is stored in a lookup table for finite state spaces or in a function approximator for continuous state spaces. The estimate is refined by performing the update,

$$\hat{J}(\mathbf{x}) \Leftarrow \min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \gamma E\{\hat{J}(\mathbf{X}_{n+1}) | \mathbf{x}_n = \mathbf{x}, \mathbf{u}\} \right],$$

until convergence. In the synchronous version of the algorithm, this update is performed in a sweep over the entire state space. In the asynchronous version of the algorithm, the update is performed on the states visited during learning trials. Assuming that all states are visited with some non-zero probability during the learning trials, then both versions of the algorithm can be shown to converge, $\hat{J} \to J^*$, for lookup table representations as time goes to infinity. When $\hat{J}$ is stored in a function approximator, all guarantees of convergence disappear.

Unlike dynamic programming, the asynchronous value iteration algorithm forces us to select actions before the learning has converged. Our choice of actions must balance explo-

ration with exploitation. A typical action selection strategy is to select the greedy action

$$\mathbf{u} = \arg\min_{\mathbf{u}} \left[g(\mathbf{x}, \mathbf{u}) + \gamma E \left\{ \hat{J}(\mathbf{X}_{n+1}) \middle| \mathbf{x}_n = \mathbf{x}, \mathbf{u} \right\} \right],$$

with high probability, and to occasionally select a random action. For continuous action spaces, it is sometimes possible to solve for the optimal action given the current estimate of the value function. [Doya, 1999] observed that this is particularly easy if $\frac{\partial g(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$ is linear in $\mathbf{u}$.

2.4.3 Monte-Carlo and Temporal-Difference Learning

Another way to obtain the cost-to-go of the current policy, J^π, is through Monte-Carlo simulation. Recall that the definition of the cost-to-go for discounted, infinite-horizon problems is

$$J^\pi(\mathbf{x}) = E \left\{ \sum_{n=0}^{\infty} \gamma^n g(\mathbf{X}_n, \mathbf{U}_n) | \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_n = \pi(\mathbf{x}_n) \right\}.$$

This expectation can be approximated by running T trials and taking the average return.

Temporal difference learning, or TD(λ), is a way to combine value iteration and Monte-Carlo estimation. Define the n-step prediction of J^π as

$$\hat{J}^\pi_n(\mathbf{x}) = \sum_{m=0}^{n-1} [\gamma^m g(\mathbf{x}_m, \mathbf{u}_n)] + \gamma^n \hat{J}^\pi(\mathbf{x}).$$

Notice that the asynchronous value iteration algorithm performs the update $\hat{J}^\pi(\mathbf{x}) \Leftarrow \hat{J}^\pi_1(\mathbf{x})$, and that Monte-Carlo updating uses the update $\hat{J}^\pi(\mathbf{x}) \Leftarrow \hat{J}^\pi_\infty(\mathbf{x})$. The TD($\lambda$) algorithm uses a weighted combination of n-step predictions to update $\hat{J}^\pi$:

$$\hat{J}^\pi \Leftarrow (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{J}^\pi_n.$$

Therefore, TD(0) is simply value iteration, and TD(1) works out to be the Monte-Carlo update. The elegance of the TD update, though, is that it can be implemented very efficiently online using an eligibility trace [Sutton and Barto, 1998].

In general, we cannot prove that TD(λ) will converge $\hat{J}^\pi$ to J^π when $\hat{J}$ is represented in a function approximator. [Tsitsiklis and Roy, 1997] proved convergence for a class of linear function approximators. TD(1) can also be shown to converge [Bertsekas and Tsitsiklis, 1996] for arbitrary approximators, but there is significant experimental evidence that algorithms using $\lambda < 1$ work more quickly in practice [Sutton and Barto, 1998].

2.4.4 Q-learning

In some situations, it is not practical to solve the optimal policy by evaluating

$$\pi(\mathbf{x}) = \arg\min_{\mathbf{u}} [g(\mathbf{x}, \mathbf{u}) + \gamma E\{J^*(\mathbf{X}_{n+1}, n+1) | \mathbf{x}_n = \mathbf{x}, \mathbf{u}\}]$$

online. A common solution to this problem is to learn a Q-function instead of the value function. The Q-function is the expected cost-to-go from taking action $\mathbf{u}$ in state $\mathbf{x}$. This

is closely related to the cost-to-go function:

$$\begin{aligned} Q^\pi(\mathbf{x},\mathbf{u}) =& g(\mathbf{x},\mathbf{u}) + \gamma E\left\{J^\pi(\mathbf{X}_{n+1})|\mathbf{x}_n = \mathbf{x}\right\} \\ J^\pi(\mathbf{x}) =& Q^\pi(\mathbf{x},\pi(\mathbf{x})) \\ Q^*(\mathbf{x},\mathbf{u}) =& g(\mathbf{x},\mathbf{u}) + \gamma E\left\{J^*(\mathbf{X}_{n+1})|\mathbf{x}_n = \mathbf{x}\right\} \\ J^*(\mathbf{x}) =& \max_{\mathbf{u}} Q^*(\mathbf{x},\mathbf{u}) \end{aligned}$$

There are a variety of Q-learning algorithms which extend TD(λ) to learning a Q-function.

Learning a Q function instead of a cost-to-go function has some subtle implications. The disadvantage is that it increases the dimensionality of the function that you are trying to learn. One advantage is that you reduce the action selection problem to a lookup in the function approximator (instead of computing the argmin online). The primary advantage, though, is that it allows us to evaluate one policy while following another (called off-policy evaluation), because our update only changes our model for the current action. Off-policy evaluation can be desirable when the optimal policy does a poor job of exploring the state space.

2.4.5 Summary

In general, value function methods are very powerful for small problems with finite state and action spaces. They take advantage of the underlying structure in an optimal control problem, the HJB equation, and therefore have the potential to be much more efficient than a policy gradient algorithm.

When the value estimate has to be stored in a function approximator instead of a lookup table, the convergence guarantees of the algorithms are weakened dramatically. There are a handful of great success stories using this approach, including TD-Gammon by Tesauro [Tesauro, 1995], but there are also trivial examples of problems for which TD(λ) fails to converge (i.e., [Boyan and Moore, 1995]). The intuitive problem is that a small change in the parameters of the value estimate can produce large changes in the greedy policy. This causes instability in the update.

2.5 Actor-Critic Methods

Actor-critic methods attempt to combine the fast, efficient learning of value function methods with the guaranteed convergence of policy gradient methods. The goal of these algorithms, which have recently begun to receive more attention in the reinforcement learning literature, is to use a value estimate to reduce the variance of the gradient update.

The conventional [Sutton et al., 1999, Konda and Tsitsiklis, 1999] actor-critic update uses an acquired estimate of the Q-function to estimate a policy gradient. For a stochastic

policy $\pi_{\mathbf{w}}(\mathbf{x}, \mathbf{u}) = Pr\{\mathbf{U} = \mathbf{u}|\mathbf{X} = \mathbf{x}, \mathbf{w}\}$:

$$\begin{aligned}
\frac{\partial}{\partial w_i} E\{C(\mathbf{w})\} =& \frac{\partial}{\partial w_i} E\{J^{\mathbf{w}}(\mathbf{x}_0)|\mathbf{x}_0\} = \frac{\partial}{\partial w_i} E\{Q^{\mathbf{w}}(\mathbf{x}_0, \mathbf{u})|\mathbf{x}_0, \mathbf{u}\} \\
=& E\left\{ \frac{\partial}{\partial w_i} \int_u \pi_{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') Q^{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') du' \Big| \mathbf{x}_0 \right\} \\
=& E\left\{ \int_u \left[\frac{\partial \pi_{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}')}{\partial w_i} Q^{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') + \pi_{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') \frac{\partial}{\partial w_i} Q^{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') \right] du' \Big| \mathbf{x}_0 \right\} \\
=& E\left\{ \int_u \left[\frac{\partial \pi_{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}')}{\partial w_i} Q^{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') + \pi_{\mathbf{w}}(\mathbf{x}_0, \mathbf{u}') \frac{\partial}{\partial w_i} \gamma E\{J^{\mathbf{w}}(\mathbf{x}_1)|\mathbf{x}_1\} \right] du' \Big| \mathbf{x}_0 \right\} \\
=& \int_{\mathbf{x}} \sum_{n=0}^{\infty} \gamma^n Pr\{\mathbf{X}_n = \mathbf{x}|\mathbf{w}, n\} \int_u \frac{\partial \pi_{\mathbf{w}}(\mathbf{x}, \mathbf{u})}{\partial w_i} Q^{\mathbf{w}}(\mathbf{x}, \mathbf{u}),
\end{aligned}$$

where $J^{\mathbf{w}}$ and $Q^{\mathbf{w}}$ are short-hand for $J^{\pi_{\mathbf{w}}}$ and $Q^{\pi_{\mathbf{w}}}$, respectively. For most systems, these functions must also be estimated online using a function approximator. By properly parameterizing the function approximators for the actor, π, and the critic, $Q^{\mathbf{w}}$, relative to each other, it is possible to update the policy using this gradient estimate and prove that the algorithm performs stochastic gradient descent for arbitrary differentiable function approximators [Sutton et al., 1999]. The claim is that an estimate of the policy gradient using a learned Q function should have less variance than an estimate based on samples taken from only a single trajectory.

I use a slightly different actor-critic algorithm for the online optimization on the robot. That algorithm is described in detail in chapter 5.

2.6 Optimal Control Examples

The following examples are intended to elucidate the relationship between the instantaneous cost, the value function, the policy, and the resulting optimal trajectories.

2.6.1 Example 1: Linear, Second-Order System

The simplest linear, second-order, controllable system is

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

These dynamics could, for instance, represent pushing a one kilogram box with u Newtons of force on a horizontal frictionless surface. Because these dynamics are linear, we can solve quadratic cost functions analytically using the LQR results. For arbitrary cost functions, we can discretize the state and action spaces and apply value iteration to efficiently solve for the optimal policy. This works well for this problem because the system has only two state variables $(x, \dot{x})$ and one action variable (u).

Second-order systems move clockwise in state space. The cost-to-go function effectively rewinds the instantaneous cost function backward in time. Figure 2-1 shows the instantaneous cost, value function, and optimal policy for a quadratic regulator cost function on this simple system. The white line represents a sample trajectory with a cross at the initial conditions. The upper right and lower left corners of the policy are distorted by edge effects (the true solution is linear), but otherwise the value iteration algorithm has done an excel-

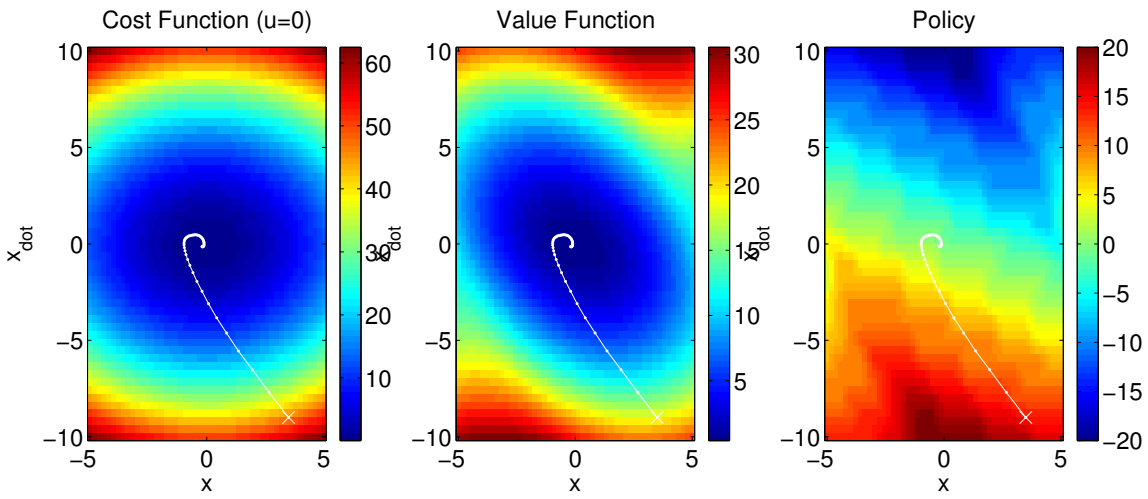

Figure 2-1: Example 1a: Infinite-horizon, discounted LQR. $\ddot{x} = u$, $g(x, \dot{x}, u) = \frac{1}{2}x^2 + \frac{1}{2}\dot{x}^2 + \frac{1}{10}u^2$, $\gamma = 0.9$.

lent job of matching the analytical LQR solution. If we were to remove the penalty on u in the cost function, then the optimal policy would be bang-bang control: apply maximum force toward the desired position, and then apply maximum braking force so that the state ends up exactly at the origin as quickly as possible.

In locomotion, the desired behavior is a stable limit cycle. What instantaneous cost function for this system would generate a limit cycle as the optimal behavior? Figure 2-2 shows one solution.

There is an interesting relationship between optimal control and stability theory. For finite-horizon problems, the value function can also serve as a Lyapunov function [Slotine and Li, 1990] for the system, since the cost-to-go monotonically decreases as the system trajectories evolve. Unfortunately, the same can not be said for infinite-horizon discounted value functions, as illustrated by the limit cycle example.

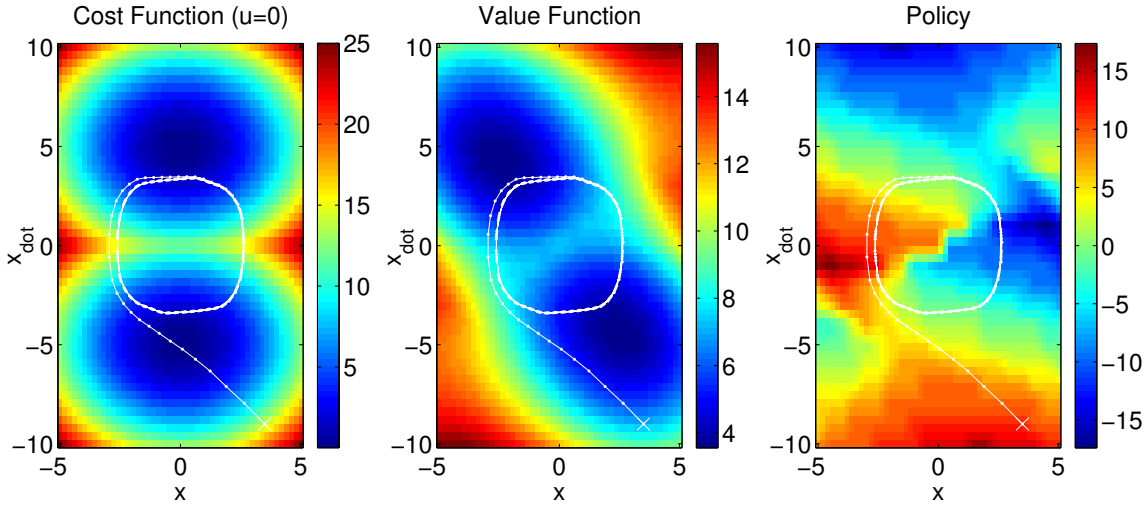

Figure 2-2: Example 1b: Optimal limit cycles. $\ddot{x} = u$, $g(x, \dot{x}, u) = \frac{1}{2}x^2 + \frac{1}{2}|\dot{x} - 5.0|^2 + \frac{1}{20}u^2$, $\gamma = 0.9$.

2.6.2 Example 2: Nonlinear, Second-Order System

Consider a simple pendulum:

$$ml^2\ddot{\theta} - mgl\sin(\theta) = u,$$

where m is a point mass, l is the length, g is gravity, and θ represents the state of the system. This system cannot be solved using LQR[6], even when the cost function is quadratic. However, since the state and action spaces are still small, we can once again apply the value iteration algorithm. Although the optimal value function resembles the value function for the linear case, the optimal policy, shown in figure 2-3, is clearly nonlinear.

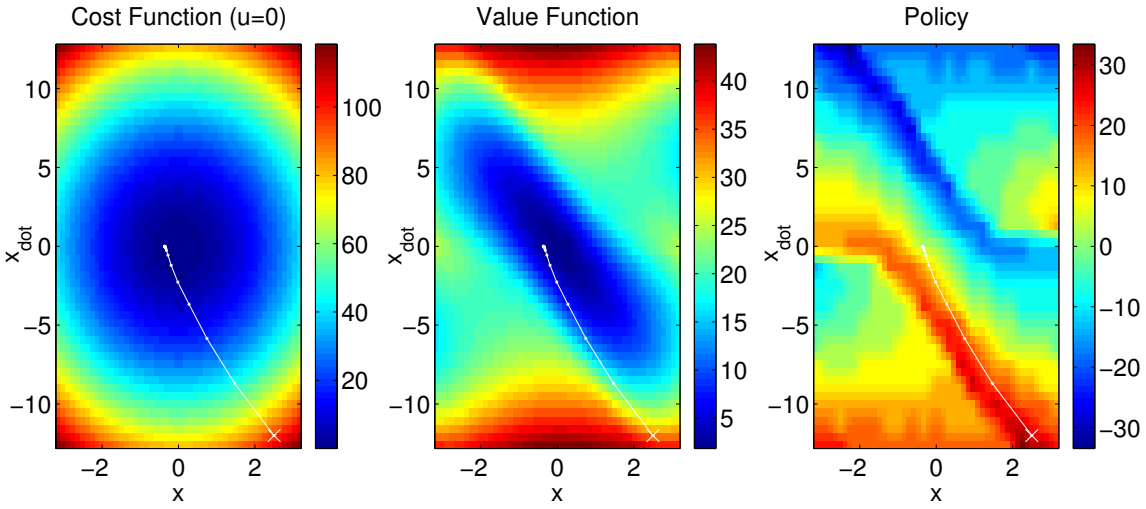

Figure 2-3: Example 2: Infinite-horizon, discounted non-linear quadratic regulator. $\ddot{x} = u$, $g(x, \dot{x}, u) = 4x^2 + \frac{1}{2}\dot{x}^2 + \frac{1}{10}u^2$, $m = 1, l = 1, g = 9.8, \gamma = 0.9$.

[6]Recall, however, that a common approach to deriving controllers for non-linear systems is to linearize the system over small regions and apply local linear controllers.

Chapter 3

Optimal Control for Legged Locomotion

3.1 Legged Robots vs. Robotic Arms

Robotic arms dramatically outperform human arms in measures of speed, accuracy, and repeatability, but walking and running robots can't compete with their biological counterparts in any of these areas. For animals, locomotion is one of the most basic and primitive of motor skills. Why is it that controlling robots with legs is so much more difficult than controlling robotic arms?

Most legged robots are under-actuated - they have fewer actuators than degrees of freedom. One of the simplest examples of an under-actuated system is the Acrobot [Spong, 1994], a two-link robotic arm with only a single actuator at the elbow. The equations of motion for this robot take the form:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \begin{bmatrix} 0 \\ \tau \end{bmatrix},$$

where $\mathbf{q} = [\theta_1, \theta_2]^T$. The zero on the right side of the equation makes it impossible to follow an arbitrary trajectory in $\mathbf{q}$. Despite many great success stories in robotic arm control, relatively little is known about controlling under-actuated systems like the Acrobot.

Now consider a seven-link planar biped, as illustrated in Figure 3-1. Even if the robot has an actuator at each joint, for a total of six, that may not be enough to completely control the seven or more degrees of freedom of the robot (angles of each of the links, and possibly the robot's location relative to the ground) at every point in time. The robot only becomes "fully" actuated if we assume that one foot is bolted to the ground. As described in the introduction, the trajectory following algorithm implemented on ASIMO [Hirai et al., 1998] assumes that the foot is rigidly attached to the ground, but also constantly monitors the ground reaction forces in order maintain the validity of that assumption.

There are also mechanical considerations which complicate legged robots. The forces and torques applied by the robot are often limited by both the walking surface and the actuators, because a locomoting robot must carry its own actuators and power supply. Typically, geared electric motors are used for walking and hydraulics or pneumatics are used on running robots. In fact, Sony's QRIO robot recently became the first biped to officially be able to both walk and run (using electric motors), although the running is not very impressive yet.

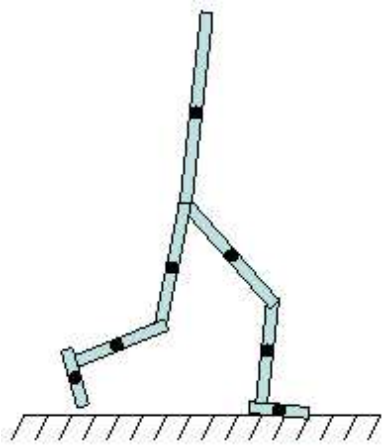

Figure 3-1: A seven link planar biped

3.2 Dynamically Stable Legged Robots

The fairly short history of dynamic legged robots has been punctuated by a handful of remarkable milestones. The first computer-controller quasi-dynamic walking gait on a biped was achieved in 1980 by Ichiro Kato and his group using artificial muscles [Kato et al., 1983]. In 1983, Marc Raibert demonstrated a planar one-legged hopping robot that could hop at desired velocity and jump over small obstacles [Raibert, 1986]. In 1990, Tad McGeer demonstrated a passive dynamic walker with knees that could walk stably down a small ramp without the use of any motors [McGeer, 1990]. In 1997, the world of robotics changed forever when Honda Motor Company announced that they had been developing a humanoid robot, P2. Honda's most recent humanoid robot, ASIMO, represents the current state-of-the-art in the field. Today, bipedal walking robots can be found in laboratories around the world. The abundance of different control systems for these robots can be sorted into a handful of categories.

The first and largest category are clones of the Honda humanoids, using a control system based on the center of pressure, trajectory tracking control used on ASIMO [Hirai et al., 1998]. These robots are sometimes described as walking robotic arms, because they attempt to rigidly follow a desired trajectory within the limits of the restrictive dynamic constraints. Robots in this category typically have high precision harmonic-drive, position-controlled actuators at each joint and flat feet equipped with pressure sensors. By assuming that the foot stays flat on the ground during the swing phase, the controller probably converges to the desired trajectory.

At the opposite end of the spectrum are the intuitive control algorithms. This includes work from the MIT Leg Laboratory used initially on the hopping robots and later extended to walking robots with a technique called virtual model control [Pratt and Pratt, 1998, Pratt and Pratt, 1999]. The controllers are called intuitive because they rely on injecting energy into the system and/or applying forces corresponding to our basic understanding of the biomechanics of movement. There are no automatic tools for deriving intuitive controllers, they often take a lot of manual tweaking and tuning, and there typically aren't any guarantees of their performance. Nevertheless, some of the most impressive demonstrations of both walking and running come from this group.

Both the intuitive controllers and the Honda-clones often make use of simplified mod-

els of walking and running dynamics (i.e., [Kajita et al., 1992, Schwind, 1998]). Many of these models come from the extensive biomechanics literature describing kinetic and kinematic properties of gait in humans and animals [McMahon, 1984, Alexander, 1996]. For the trajectory following robots, these models are useful in designing the desired trajectories.

In a category by themselves are the controllers based on coupling "neural" oscillators or pattern generators to a walking robot (i.e. [Taga, 1995]). There is definitive experimental evidence in lower vertebrates, and suggestive evidence in higher mammals, that pattern generators like these can be found in our spinal cords and are used to coordinate movement between multiple limbs. It is amazing that these generators can be tuned by hand to construct a detailed enough feedback response to allow dynamically stable walking, but it is not clear that this approach will scale to humanoids performing a wide class of movements.

The final, and often distinct, class of controllers are those acquired automatically using machine learning.

3.3 The Optimal Control Approach

Although there is a great deal of literature on learning control for dynamically stable legged robots, there are relatively few examples of learning algorithms actually implemented on the robot or which work quickly enough to allow the robot to adapt online to changing terrain. Much of the work that is done only in simulation depends on features of the simulation that are not available on the real robot, including perfect sensors and actuators, and the ability to simulate the robot from specific initial conditions millions of times. Some researchers attempt to learn a controller in simulation that is robust enough to run on the real robot [Morimoto and Atkeson, 2002, Tedrake and Seung, 2002], treating differences between the simulation and the robot as disturbances. Even the algorithms that are not directly suitable for implementation on real robots contain interesting ideas that are relevant for a variety of learning algorithms.

One of the early examples of online biped learning came from [Miller, 1994], who implemented online gait adaptation on a real 3D robot with servo motors using 'specialized' CMAC [Miller et al., 1990] function approximators (for right/left balance, front/back balance, and closed-chain kinematics). The output of the networks augmented an existing controller to define time-varying set-points for joint PD controllers. The learning algorithm was a heuristic update that contained a supervised learning term and a term that roughly resembled temporal difference learning. Training consisted of first learning to sway from side to side, then marching in place, then walking forward. Some other interesting ideas in this paper included a pause routine that was inserted every time that the robot took a bad step, and the fact that the PD gains for each actuator were turned up whenever the corresponding foot was in contact with the ground. The total training took about 1 hour. This work was extended from a static gait to a dynamic gait in [Kun and Miller, 1996].

[Westervelt and Grizzle, 2002] used a SQP (sequential quadratic programming) optimization in conjunction with a stability analysis for a five link planar biped. Using results based on a hybrid zero-dynamics formulation, they parameterized a class of policies using Bézier splines that were all provably asymptotically stable. The optimization attempted to minimize energy and "some other reasonable kinematic and dynamic constraints" within this class of functions. In a related approach, [Huber and Grupen, 1998] uses learning to switch between known control policies which are provably stable over some portion of the configuration space of their four-legged robot.

Although not directly related to legged locomotion, [Ng et al., 2003] presents some related work on learning control of an autonomous helicopter. The learning idea was to obtain a linear stochastic model of the helicopter dynamics using traditional model estimation algorithms. A policy for this model was optimized offline using the Pegasus algorithm [Ng and Jordan, 2000], which approximates the stochastic system with a deterministic one by using the same random number generator on each iteration of the policy gradient evaluation. Using a very simple policy gradient algorithm, Ng provided impressive demonstrations of helicopter control. I actually use an algorithm that, in retrospect, is very similar to Pegasus for the hopping results in Chapter 6.

A number of researchers have considered the problem of using machine learning to tune the parameters of a hand-designed controller. [Smith, 1998] introduces a method combining CMACs and eligibility traces to train a feedback controller online. The algorithm was applied to the control of a physical cart-pole system as well as a simulation of the Raibert 3D one-legged hopper and of a 3D biped. For the hopper, Smith's networks accepted a small number of inputs (speed,body angle, and slope of the ground) and output 2 of the gains for a modified Raibert controller. His reward function compared desired and actual speeds, and he reports good performance after only 20 iterations of the algorithm. Similarly, [Chew, 2000] used Q-learning with CMAC networks to find key parameters of swing leg strategy to augment the controller by [Pratt and Pratt, 1998]. One or two parameters were tuned at a time, and the policy was evaluated once per step. The objective function included a desired forward velocity and lateral stability. This controller reportedly produced 100 seconds of walking by the 16th trial.

Another common approach is to approximate an optimal solution by solving a reduced-order model of the system. [Larin, 1999] and [Seyfarth et al., 2002] reduce the dynamics of the planar one-leg hopper down to the spring-loaded inverted pendulum (SLIP) model, which consists of a point mass on a springy, massless leg. On this model, they could predict exactly the amount of thrust required to put the hopper back at the desired hopping height in one hop. [Maier et al., 1999] took a similar approach, but used radial basis functions (RBFs) to automatically acquire the controller. The network inputs were the leg angle of attack at touchdown and at lift-off, the x and y components of the COM velocity at touchdown and at lift-off, and the desired horizontal velocity. The output of the network is the angle of attack of the leg. The results presented in Chapter 6 of this thesis are similar, but are implemented on a complete model of the hopper instead of the reduced SLIP model.

Learning control has also been successfully implemented on Sony's quadrupedal robot AIBO (i.e., [Kohl and Stone, 2004]). This algorithm optimizes a parameterized open-loop trajectory for fast walking, using speed as the only objective function. The "policy gradient" algorithm used here is naive numerical gradient descent. This approach appears to work because AIBO is robust enough to reproduce trajectories with reasonable fidelity using only open-loop control (stability is not a concern), and because the experimenters automated the learning process with the robot walking back and forth across the field between beacons. Trajectory optimization as seen on AIBO has been studied extensively in the robot arm literature. It is the constraints of dynamic stability that distinguishes the formulation of the legged locomotion trajectory optimization problem from its predecessors.

Similar trajectory optimization routines have been described for bipeds by [Yamasaki et al., 2002] and [Channon et al., 1990], to name a few. [Yamasaki et al., 2002] used genetic algorithms to learn open-loop trajectory commands for PINO, a little hobby-servo based biped. They claimed that by tuning trajectories (parameterized by Fourier coefficients) they were able to compensate for the unmodelled dynamics of the servo motors to produce a smoother and

more efficient gait. [Channon et al., 1990] reduced gait optimization to fitting the coefficients of a polynomial for the feed forward trajectory of the swing foot and of the hip for a five link biped with the stance leg bolted to the ground. The remaining joint angles were computed using inverse kinematics. They used an interesting cost function that actually attempted to model some of the motor properties ("armature losses"). Their final plot shows the minimal energy for a given walking speed, that they claim correlates nicely with expectations. [Juang and Lin, 1996] took a similar approach using the Backpropagation-Through-Time algorithm.

One of the most principled examples of applying numerical optimal control to bipedal locomotion comes from [Hardt et al., 1999]. They develop a tool for efficiently computing the dynamics and gradients of a rigid-body mechanism, even through a discrete impulse model. They then apply an off-the-shelf optimal control solver (DIRCOL) to a simulation of a five-link biped model to solve for the feed-forward torques of each joint. The optimization attempted to minimize $\int \mathbf{u}(t)^T \mathbf{u}(t) dt$ while satisfying boundary constraints. The stated goal of this work was to model human walking, which presumably minimizes energy consumption.

There have been a number of other modelling studies of human locomotor "optimal" control. [Kuo, 1995] argued that LQRG control could explain the selection of control strategies that humans use in response to small perturbations to stable upright balance. [Todorov, 2002] argues that stochastic optimal feedback control can explain a wide variety of motor control and motor coordination tasks in biology. [Anderson and Pandy, 2001] used brute-force dynamic optimization packages to optimize energy consumption via the feed-forward muscle-activation trajectories for a three-dimensional, neuro-musculo-skeletal model of the human body. By enforcing only kinematic boundary conditions, they claim that their model reproduces the salient features of normal gait, and justifies the use of minimum metabolic energy per unit distance traveled as a measure of walking performance.

Finally, and most recently, [Vaughan, 2003, Vaughan et al., 2004] took an approach that is very similar to the one used in this thesis - learning a control strategy to actuate a simulation of a passive dynamic walker. They used genetic algorithms to tune both a open-loop central pattern generator and a neural network feedback controller. They also made an effort to evaluate the robustness of the learned controller to a variety of parameters, in an attempt to derive a controller that could potentially work on a real robot in the near future.

3.4 Return Map Analysis

During stable walking on flat terrain, the limit cycle trajectories of a legged robot can be analyzed using a Poincaré map, also known as a return map [Strogatz, 1994]. This tool has been used extensively in the legged robot literature (i.e., [McGeer, 1990, Koditschek and Buehler, 1991]). For an n-dimensional dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$, a Poincaré section S is a $n-1$ dimensional surface that the system crosses exactly once during each period. By integrating the dynamics forward from one intersection of S to the next, we can describe the discrete return map dynamics of the system as

$$\mathbf{x}_{n+1} = r(\mathbf{x}_n).$$

The advantage of doing this is that the limit cycle stability of the the periodic trajectory in $\mathbf{x}$ can now be studied by looking at the fixed point stability of the discrete system in $\mathbf{x}_n$.

As an example, consider the classical Van der Pol oscillator:

$$\ddot{x} + 0.2(x^2 - 1)\dot{x} + x = 0.$$

Figure 3-2 plots a sample trajectory of this dynamics and the associated return map. Fixed points of the return map are located at the intersection of the return map and the line of slope one. The local stability of this fixed point can be determined by an eigenvalue analysis of the return map dynamics linearized around this point.

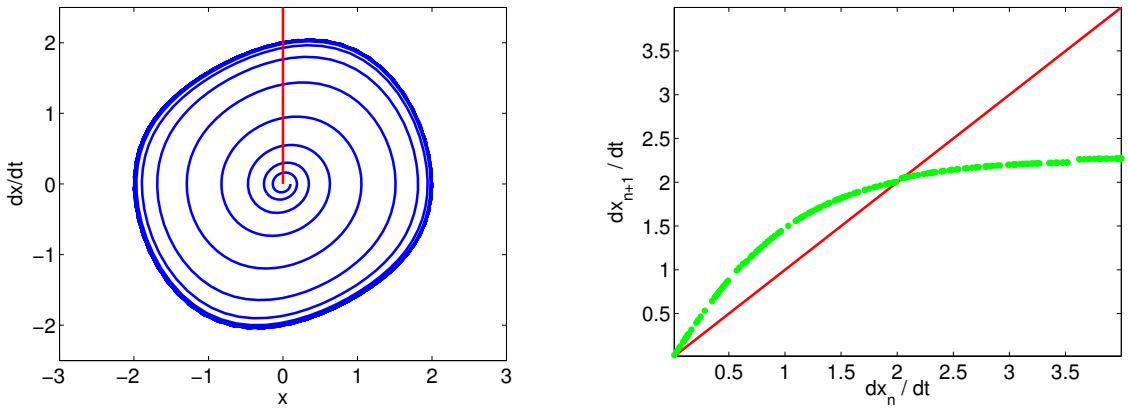

Figure 3-2: Return Map Analysis of the Van der Pol oscillator. On the left is a sample trajectory of the oscillator plotted in state space. The vertical line represents the Poincaré section. On the right is the associated return map plotted through that section. The diagonal line on this plot is the line of slope one. The points are return map data acquired by numerically simulating the oscillator dynamics.

Although it is rare that we are able to obtain an analytical form for the return map dynamics, r, I make extensive use of numerical return map analysis throughout this thesis. For a walking system, the return map is often taken at the point of collision between one of the feet and the ground, and is often called the step-to-step return map [McGeer, 1990]. This is a natural configuration to slice the trajectory because it simplifies the analytical analysis and because it is an event that is easy to measure using contact sensors that are present on most walking robots.

Chapter 4

Passive Dynamic Walking

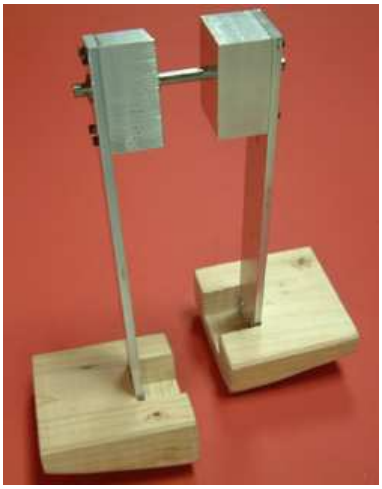

Figure 4-1: The 3D passive dynamic walker

In order to build an actuated walker capable of stable passive walking down a ramp, I first needed to understand the dynamics of passive walking. The passive dynamic walker shown in Figure 4-1 represents the simplest machine that I could build which captures the essence of stable dynamic walking in three dimensions. It has only a single passive pin joint at the hip. When placed at the top of a small ramp and given a small push sideways, the walker will rock onto a single stance leg, allowing the opposite leg to leave the ground and swing forward down the ramp. Upon landing, the robot rocks onto the opposite foot, and the cycle continues. This walker is morphologically equivalent to the Tinkertoy walker [Coleman and Ruina, 1998], except that on our robot the center of the radius of curvature of the feet is higher than the center of mass. Because of this, standing is a statically stable configuration.

The energetics of this passive walker are common to all passive walkers: the energy lost due to friction and collisions when the swing leg returns to the ground are balanced by the gradual conversion of potential energy into kinetic energy as the walker moves down the slope. The characteristics of the walker's gait are determined by the mass distribution and by the shape of the large curved feet. In order to understand this relationship, I began by analyzing two of the simplest walking models: the rimless wheel and the compass gait. By working up to a dynamic model of our curved foot robot, I was able to design the curvature

for the feet to tune the step length and step frequency of the passive gait.

In this chapter, I will present an analysis of these passive walking models, working up to the design and experimental stability analysis of the physical 3D passive dynamic walker pictured in Figure 4-1.

4.1 The Rimless Wheel

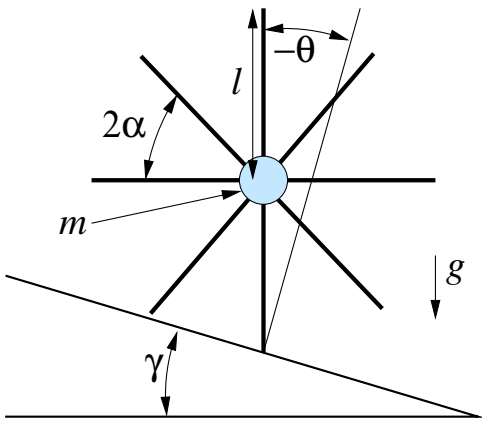

Figure 4-2: The rimless wheel

The most elementary model of passive dynamic walking, first used in the context of walking by [McGeer, 1990], is the rimless wheel. This simplified system has rigid legs and only a point mass at the hip as illustrated in Figure 4-2. To further simplify the analysis, we make the following modelling assumptions:

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

The most comprehensive analysis of the rimless wheel was done by [Coleman, 1998]. The analysis presented here simplifies and extends that work with a novel method for deriving the basins of attraction using contraction analysis [Lohmiller and Slotine, 1998].

4.1.1 Stance Dynamics

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

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

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

where

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

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 $\theta(t) = \alpha$, 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.

4.1.2 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:

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

Simulations of these dynamics reveal a stable limit cycle solution with a continuous phase punctuated by a discrete collision, as shown in Figure 4-3. 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.

4.1.3 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 $\gamma \leq \alpha$. For $\dot{\theta}_n > \omega_1$ this "step-to-step return map" is simply

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

Using the same analysis for the other cases, we can complete the return map. For $0 \leq \dot{\theta}_n < \omega_1$, we have

$$\dot{\theta}_{n+1} = -\dot{\theta}_n.$$

Using the convention that negative initial velocities incur a collision immediately, the thresh-

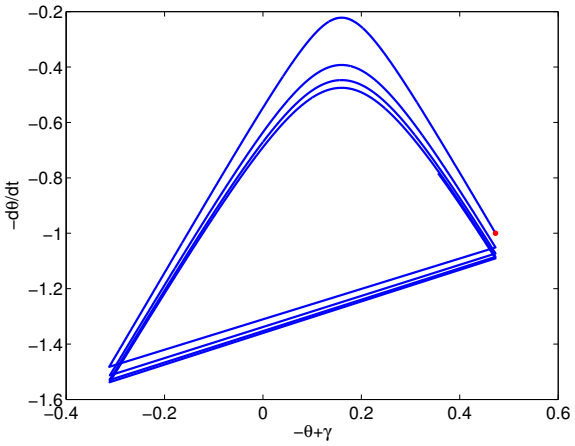

Figure 4-3: Limit cycle trajectory of the rimless wheel ($m = 1, l = 1, g = 9.8, \alpha = \pi/8, \gamma = 0.08$). Notice that the output coordinate system has been changed so that the trajectory can be compared directly to the models presented in the remainder of this chapter.

old for taking a step in the opposite direction is

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

For $\omega_2 < \dot{\theta}_n < 0$, we have

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

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

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

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 $\theta = -\gamma$ 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, ω_1 no longer exists because it is kinematically impossible to have the wheel statically balancing on a single leg. For this case, the region $0 \leq \dot{\theta}_n < \omega_1$ disappears, and the return map presented for $\dot{\theta}_n > \omega_1$ is valid for $\dot{\theta}_n \geq 0$. The negative half of the return map remains mostly intact, because ω_2 is simply the threshold for transfer onto the rear leg. The only difference is that the system is now well-defined at the point ω_2, and the conditions for the last region become $\dot{\theta}_n \leq \omega_2$.

4.1.4 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:

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

The limit cycle plotted in Figure 4-3 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.

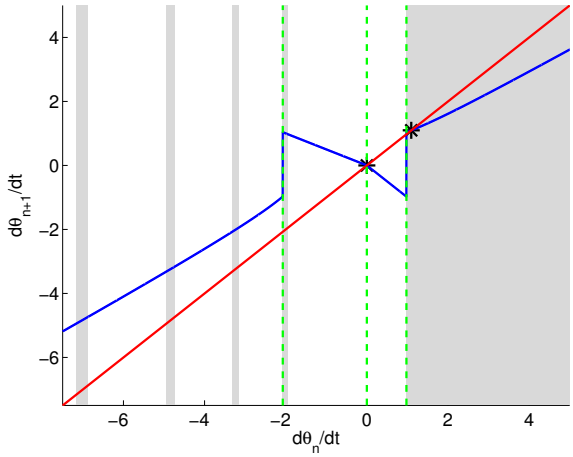

Figure 4-4: Return map, fixed points, and basins of attraction of the rimless wheel ($m = 1, l = 1, g = 9.8, \alpha = \pi/8, \gamma = 0.08$). The gray area is the basin of attraction of ω_{roll}, and the white area is the basin of attraction of ω_{stand}.

To understand the stability of these fixed points, we first observe that the return map dynamics are a contraction mapping [Lohmiller and Slotine, 1998]. Briefly, a system's dynamics are said to be contracting if initial conditions are forgotten exponentially. A sufficient condition for the discrete system $\mathbf{x}_{n+1} = f(\mathbf{x}_n)$ to be contracting is

$$\mathbf{A} = \frac{\partial f}{\partial \mathbf{x}}^T \frac{\partial f}{\partial \mathbf{x}} - \mathbf{I} < \mathbf{0}.$$

Intuitively, although the rimless wheel is modelled as conservative during the stance phase, foot collisions cause neighboring trajectories to converge, causing the dynamics to contract on the return map. For $\dot{\theta}_n > \omega_1$, we have

$$A = \cos^2(2\alpha)\frac{\dot{\theta}_n^2}{\dot{\theta}_n^2 + 4\frac{g}{l}\sin\alpha\sin\gamma} - 1 < 0.$$

For $\omega_2 < \dot{\theta}_n < 0$, we have

$$A = \cos^4(2\alpha) - 1 < 0.$$

For $\dot{\theta}_n < \omega_2$,

$$A = \frac{\dot{\theta}_n^2\cos^4(2\alpha)}{\dot{\theta}_n^2\cos^2(2\alpha) - 4\frac{g}{l}\sin\alpha\sin\gamma} - 1 < 0.$$

Finally, by convention the one-step dynamics for $0 \leq \dot{\theta}_n < \omega_1$ do not include a foot collision and therefore are not contracting. However, with the exception of the fixed point at $\dot{\theta}_n = 0$, points in this region are mapped in one step into $\omega_2 < \dot{\theta}_{n+1} < 0$, making the two-step dynamics contracting. Therefore, except for the undefined points ω_1 and ω_2, we have the global property that neighboring trajectories will converge and initial conditions will be exponentially forgotten.

Consider the parameter regime where both fixed points exist. First, observe that the region $\dot{\theta}_n > \omega_1$ is closed (trajectories inside the region will stay in that region). In other words, $\dot{\theta}_{n+1} > \omega_1$ when $\dot{\theta}_n > \omega_1$ and $\omega^*_{roll} > \omega_1$. Since the region is closed and contracting with a fixed point ω^*_{roll}, all trajectories in the region $\dot{\theta}_n > \omega_1$ converge exponentially to ω^*_{roll}. By iterating the return map backward, we can map out all of the initial conditions that will cause the system to enter this contracting region. This set of initial conditions (plotted in gray in Figure 4-4) define the basin of attraction of the rolling fixed point, and include a series of stripes with increasing width in the region $\omega_n < 0$. The boundaries of this basin of attraction are formed by the points ω_1, ω_2, and all initial conditions that lead to ω_1 and ω_2, all of which will eventually lead to the unstable fixed point of the continuous dynamics where $\theta = \dot{\theta} = 0$. For convenience, we will refer to the set of boundary points as B.

The remaining regions (outside of the basin of attraction of the rolling fixed point and not in B) form the basin of attraction of the standing fixed point, and are plotted in white in Figure 4-4. To see this, observe that this is by definition a closed region (if it where to ever leave the region, then it would have been colored gray). Since all trajectories in this region converge, they must converge to the only fixed point in that region, ω^*_{stand}.

When $\omega^*_{roll} < \omega_1$, only the standing fixed point exists. The region $\dot{\theta}_n > \omega_1$ is still contracting, but is no longer a closed set and no longer has a fixed point. Therefore all initial conditions $\dot{\theta}_0 \notin B$ will converge exponentially to ω^*_{stand}.

As γ approaches α, the width of the gray stripes increases until, when $\gamma > \alpha$, the white region and the standing fixed point cease to exist. For this regime, the system is well defined at all of the boundary conditions, and the fixed point ω^*_{roll} is globally exponentially stable.

This model was recently implemented in a physical robot which could control its speed by actively changing the leg length [Yan and Agrawal, 2004].

4.2 The Compass Gait

The rimless wheel models only the dynamics of the stance leg, and simply assumes that there will always be a swing leg in position at the time of collision. To remove this assumption, we take away all but two of the spokes, and place a pin joint at the hip. To model the dynamics of swing, we add point masses to each of the legs.

In addition to the modelling 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 [Goswami, 1999, Spong and Bhatia, 2003], 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 $\mathbf{q} =$

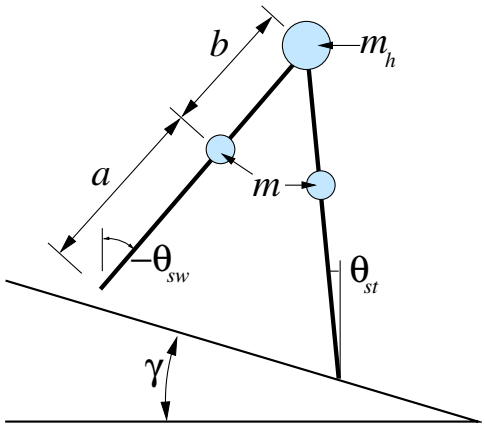

Figure 4-5: The compass gait

$[\theta_{sw}, \theta_{st}]^T$, we can write the dynamics as

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = 0,$$

with

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

and $l = a + b$. These equations come straight out of [Goswami et al., 1996].

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 4-6. 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 [Goswami et al., 1996].

The basin of attraction of the stable limit cycle is a narrow band of states surrounding

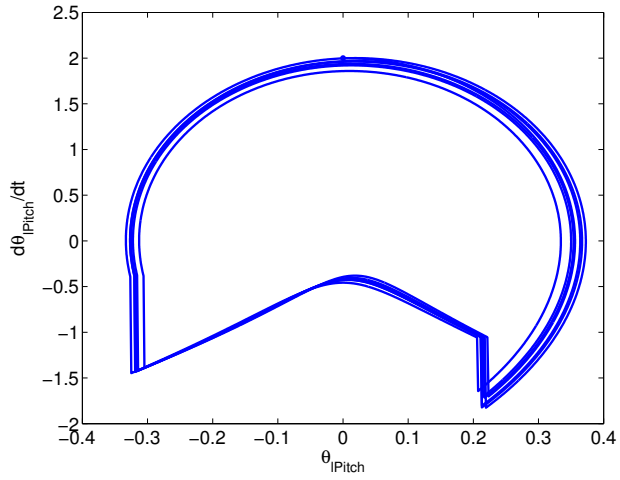

Figure 4-6: Limit cycle trajectory of the compass gait. (m = 5kg,m_h = 10kg,$a = b$ = 0.5m,ϕ = 0.03deg. $\mathbf{q}(0) = [0, 0, 2, -0.4]^T$). θ_{lPitch} is the pitch angle of the left leg, which is recovered from θ_{st} and θ_{sw} in the simulation with some simple book-keeping.

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.

4.3 The Curved Foot Model

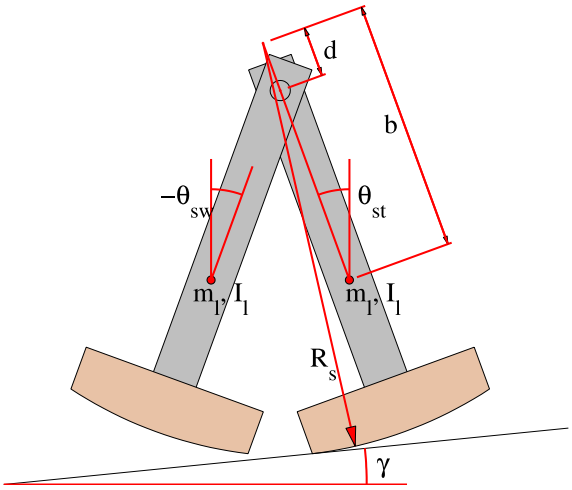

Figure 4-7: The curved foot model

To model the sagittal plane dynamics of the passive walker in Figure 4-1, we need to extend the compass gait model in two important ways. First, the addition of the large curved feet will dramatically increase the basin of attraction of the stable walking solution. We assume that one of the feet is always in contact with the ground at exactly one point. We also assume that the feet do not slip, but we neglect rolling friction. Second, we replace the point masses in the compass gait model with more realistic mass and moments of inertia for each link. A complete listing of the parameters of this new model can be found in Figure

4-7, and the dynamics are given by:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{0},$$

where $\mathbf{q} = [\theta_{st}, \theta_{sw}]^T$, and:

$$\begin{aligned}
H_{11} =& I_l + m_l b^2 + m_l d^2 + 2m_l R_s^2 - 2m_l R_s (b+d)\cos(\theta_{st} - \gamma) \\
H_{12} =& H_{21} = m_l (b-d)[d\cos(\theta_{st} - \theta_{sw}) - R_s \cos(\theta_{sw} - \gamma)] \\
H_{22} =& I_l + m_l (b-d)^2, \\
B_{11} =& m_l R_s (b+d)\sin(\theta_{st} - \gamma)\dot{\theta}_{st} + \frac{1}{2} m_l a (b-d)\sin(\theta_{st} - \theta_{sw})\dot{\theta}_{sw} \\
B_{12} =& m_l (b-d)[d\sin(\theta_{st} - \theta_{sw})(\dot{\theta}_{sw} - \frac{1}{2}\dot{\theta}_{st}) + R_s \sin(\theta_{sw} - \gamma)\dot{\theta}_{sw}] \\
B_{21} =& m_l (b-d)[d\sin(\theta_{st} - \theta_{sw})(\dot{\theta}_{st} - \frac{1}{2}\dot{\theta}_{sw}) - \frac{1}{2} R_s \sin(\theta_{sw} - \gamma)\dot{\theta}_{sw}] \\
B_{22} =& \frac{1}{2} m_l (b-d)[d\sin(\theta_{st} - \gamma) + R_s \sin(\theta_{sw} - \gamma)]\dot{\theta}_{st} \\
G_1 =& m_l g (b+d)\sin\theta_{st} - 2m_l g R_s \sin\gamma, \\
G_2 =& m_l g (b-d)\sin\theta_{sw}.
\end{aligned}$$

As with the compass gait model, we assume the swing foot can swing through the ground. Only the transfer of support as the swing leg becomes the stance leg is modeled as a collision. Because our model has large curved feet rather than point feet, some portion of the swing foot remains below the ground for the portion of the swing phase after the swing leg moves past the stance leg. Therefore, we are not able to model the time of collision as the time of the second impact. Instead, we couple this sagittal plane model to the frontal plane model discussed in the next section by estimating that a collision occurs once every half period of the frontal plane oscillations. At this moment, the collision is again modeled as an angular momentum conserving impulse:

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

where

$$\begin{aligned}
\Omega_{11}^- =& 2bd\cos(\theta_{sw} - \theta_{st}) - (b+d)R_s \cos(\theta_{sw} - \gamma) \\
& - 2bR_s \cos(\theta_{st} - \gamma) + 2R_s^2 + b^2 - bd \\
\Omega_{12}^- =& \Omega_{21}^- = (b-d)(b - R_s \cos(\theta_{sw} - \gamma)) \\
\Omega_{22}^- =& 0 \\
\Omega_{11}^+ =& (b-d)[d\cos(\theta_{st} - \theta_{sw}) - R_s \cos(\theta_{st} - \gamma) + (b-d)] \\
\Omega_{12}^+ =& -R_s(b-d)\cos(\theta_{st} - \gamma) - R_s(b+2d)\cos(\theta_{sw} - \gamma) \\
& + d^2 + 2R_s^2 + R_s b\cos(\theta_{sw} + \gamma) - b^2\cos(2\theta_{sw}) \\
& + d(b-d)\cos(\theta_{st} - \theta_{sw}) \\
\Omega_{21}^+ =& (b-d)^2 \\
\Omega_{22}^+ =& (b-d)(d\cos(\theta_{st} - \theta_{sw}) - R_s \cos(\theta_{st} - \gamma)
\end{aligned}$$

This simulation generates stable trajectories (see Figure 4-8) that are extremely similar to those generated by the compass gait (recall Figure 4-6), except that they are much more stable. Our dynamics do not model the edges of the feet, so our simulation actually models a passive walker shaped like two halves of an ellipsoid. Nevertheless, we have not been able to find any initial conditions from which the system does not return to a stable gait. Figure 4-8 was generated using the initial conditions with all variables set to zero and a slope of 0.027 radians, which is approximately the starting configuration that we use on our passive walker.

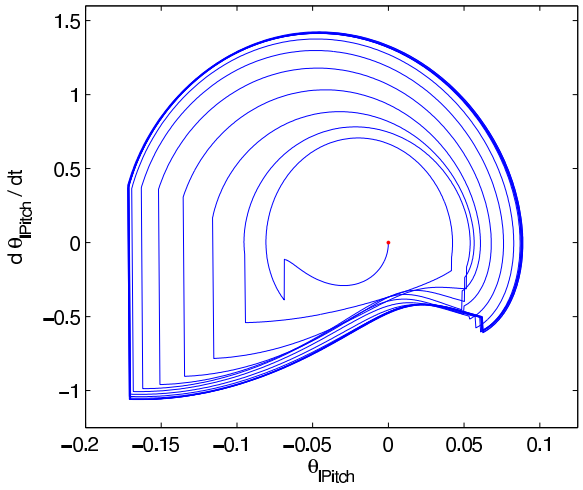

Figure 4-8: Limit cycle trajectory of the curved foot model. θ_{lPitch} is the pitch angle of the left leg, which is recovered from θ_{st} and θ_{sw} in the simulation with some simple book-keeping.

The step length and the forward velocity of the steady state gait can be tuned by adjusting the radius of curvature, R_s. Smaller radii cause the robot to fall forward more quickly. For the slope of 0.027 radians, a simulation of our model predicts that our robot will take steps of 2.56 inches (6.50 cm) and walk with an average forward velocity of 8.81 inches/second (22.38 cm/s).

4.4 The Frontal Plane Model

The mechanism[1] for achieving foot clearance on this walker is a gentle rocking motion in the frontal plane. To model these dynamics, we assume that the robot is always in contact with the ground at exactly one point and that the foot rolls without slipping. The equations of motion, in terms of body angle θ, for this planar model are given in three parts using the form

$$H(\theta)\ddot{\theta} + B(\theta, \dot{\theta})\dot{\theta} + G(\theta) = 0.$$

When $|\theta| > \phi$, the ground contact point is in the curved portion of one of the feet (the

[1]Other mechanisms for achieving foot clearance for the swing leg include bending at the knees, or cutting a hole in the ramp where the collision would have occurred.

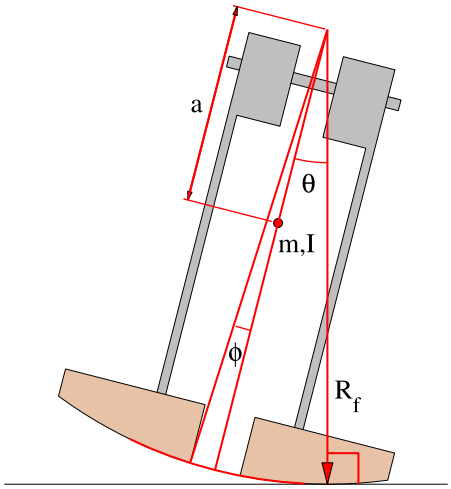

Figure 4-9: The frontal plane model

boundary condition on the outside of the foot is not modeled), and the dynamics are:

$$\begin{aligned} H(\theta) &= I + ma^2 + mR_f^2 - 2mR_f a\cos\theta, \\ B(\theta,\dot{\theta}) &= mR_f a\dot{\theta}\sin\theta, \\ G(\theta) &= mga\sin\theta. \end{aligned}$$

When $|\theta| \leq \phi$, the ground contact is along the inside edge of the foot. In this case, the dynamics are:

$$\begin{aligned} H(\theta) &= I + ma^2 + mR_f^2 - 2mR_f a\cos(\theta-\alpha), \\ B(\theta,\dot{\theta}) &= 0, \\ G(\theta) &= mg(a\sin\theta - R_f\sin\alpha). \end{aligned}$$

where $\alpha = \theta - \phi$ if $\theta > 0$, otherwise $\alpha = \theta + \phi$.

Finally, the collision of the swing leg with the ground is modeled as an inelastic (angular momentum conserving) impulse,

$$\dot{\theta}^+ = \dot{\theta}^- \cos\left[2\tan^{-1}\left(\frac{R_f\sin\phi}{R_f\cos\phi - a}\right)\right],$$

which occurs when $\theta = 0$.

A simulation of these dynamics produces a damped oscillation that will eventually result in the robot standing in place (energy lost on impact is not restored). Our primary concern for this model is the *frequency* of that oscillation. For a given mass and moment of inertia, we can change the frequency by changing R_f. The actuated version of the robot, presented in the next chapter, carries its mass very differently than the purely passive version of the robot, due to the added mass motors and sensors. By simulating this model, we were able to find very different radii for the feet in the frontal plane that allow different versions of the robot to both oscillate back and forth with the desired step frequency of approximately 1.4 Hz. The newest version of our actuated walker is considerably heavier because it carries the computer and batteries on board, so we reduced the desired step frequency to 0.8 Hz

for this robot.

4.5 Experiments

The planar models from the last two sections can be used as tools for designing the curvature of the feet to approximately tune the step frequency and step length of our robot. The coupling between these models is more complicated, and we are currently studying them in a simulation of the full 3D dynamics. The most important characteristic of this coupling is that energy from the sagittal plane stabilizes oscillations in the frontal plane. Consequently, we should expect to observe smaller steps than that predicted by the sagittal plane model.

Using the curvature of the feet in the frontal (R_f) and sagittal (R_s) planes determined from the planar models, we machined experimental feet on our CNC milling machine. The surfaces of the feet are given by:

$$z = \sqrt{R_f^2 - x^2} - R_f + \sqrt{R_s^2 - y^2} - R_s.$$

Using these feet, our robot produces stable periodic trajectories when placed on a small decline. Figure 4-10 demonstrates this stability with a sample trajectory of the machine walking down a slope of 0.027 radians. θ_{roll} is the roll angle of the robot body, in radians, which was simply called θ in the frontal plane model. This data was recorded from the actuated version of the robot with its ankle joints mechanically locked in place, because the passive robot is not equipped with the necessary sensors.

The limit cycle displayed in Figure 4-10 is fairly typical. Notice that the initial conditions for the robot are slightly outside the steady state trajectory, but that trajectories converge to a very reproducible cycle in roll and pitch. The robot has an uncompensated moment about the yaw axis - it will twist and turn whenever the point contact of the foot slips on the ground. This can be seen in the wandering trace of the yaw variable.

Upon close inspection, we determined that the majority of the noise visible in our unfiltered data is actually due to a mechanical vibration of the leg at approximately 10 Hz. For this reason, we have decided to low-pass filter our limit cycle plots at 8 Hz with a 4th-order Butterworth filter (sampling rate is 100 Hz).

Local Stability Analysis

The local stability of the passive walkers is traditionally quantified by examining the eigenvalues of the linearized step-to-step return map [McGeer, 1990], taken around a point in the period either immediately preceding or immediately following the collision. While we are currently designing a foot contact switch that will not interfere with the curved feet of the robot, the return map for this analysis is evaluated through the hyperplane $\theta_{roll} = 0$, when $\dot{\theta}_{roll} > 0$. The point in the cycle when the robot passes through the vertical position in the frontal plane is the expected point of impact.

The state of our passive robot is described by 4 variables (θ_{yaw}, θ_{lPitch}, θ_{rPitch}, θ_{roll}) and their derivatives, therefore the return map has dimension 7. $lPitch$ is short for left leg pitch and $rPitch$ is for the right leg pitch. In a simulation, we could find the eigenvalues of the return map numerically by perturbing the robot from the fixed point by ϵ in each dimension [Goswami et al., 1996]. Unfortunately, on the real robot we do not have accurate enough position control nor accurate enough sensors to perform this idealized analysis. To evaluate the eigenvalues of the return map experimentally on our robot, we ran the robot

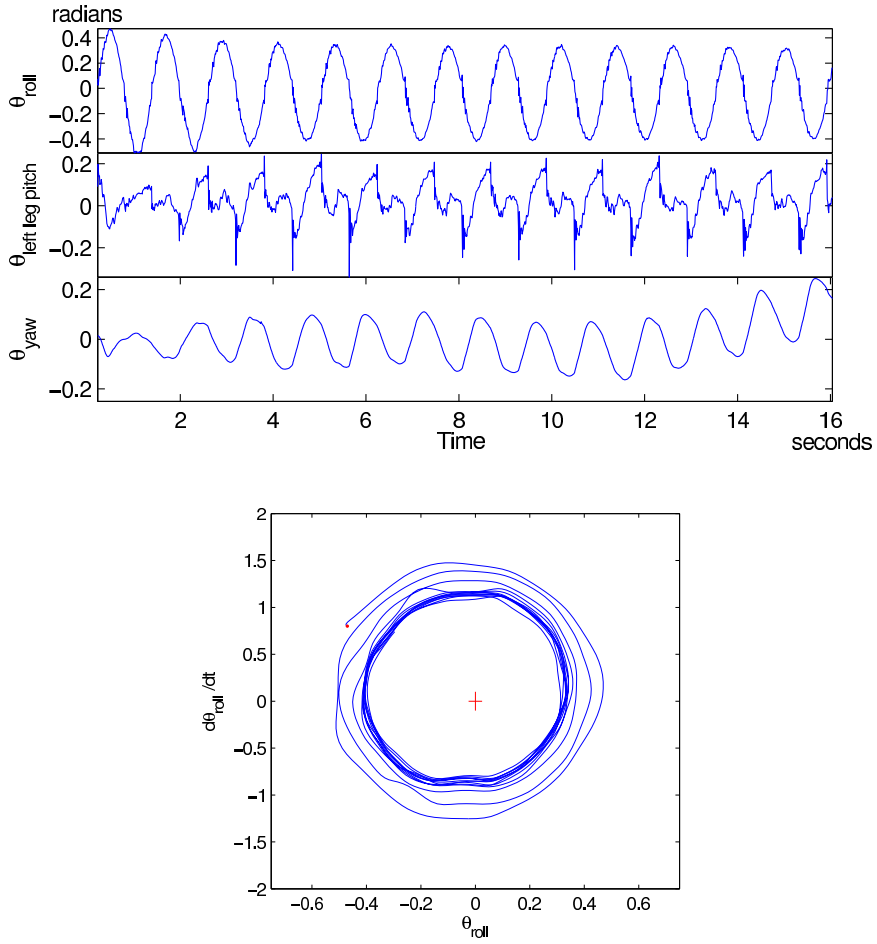

Figure 4-10: Passive dynamic walking experiments. The top figure plots the raw (unfiltered) yaw, pitch, and roll sensors for walking down a slope of 0.027 radians. The bottom figure is the state space plot of the resulting limit cycle in the roll axis, low-pass filtered at with a cut-off at 8 Hz.

from a large number of initial conditions and created the vectors $\mathbf{x}_j^i$, 7×1 vectors which represent the state of the system on the ith crossing of the jth trial. For each trial we estimated $\mathbf{x}_j^*$, the equilibrium of the return map. Finally, we performed a least squares fit of the matrix $\mathbf{A}$ to satisfy the relation

$$(\mathbf{x}_j^{i+1} - \mathbf{x}_j^*) = \mathbf{A}(\mathbf{x}_j^i - \mathbf{x}_j^*).$$

This was accomplished by accumulating the data from all trials into matrices

$$\begin{aligned}\mathbf{X} =& [\mathbf{x}_1^1 - \mathbf{x}_1^*, \mathbf{x}_1^2 - \mathbf{x}_1^*, \ldots, \mathbf{x}_2^1 - \mathbf{x}_2^*, \ldots] \\ \mathbf{Y} =& [\mathbf{x}_1^2 - \mathbf{x}_1^*, \mathbf{x}_1^2 - \mathbf{x}_1^*, \ldots, \mathbf{x}_2^2 - \mathbf{x}_2^*, \ldots]\end{aligned}$$

and computing

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}.$$

After 63 trials with the robot walking down a ramp of 0.027 radians, our linear approximation of the return map had the following eigenvalues: 0.88, 0.87, 0.75, 0.70, $0.34 \pm 0.11i$. 61 trials with on a slope of 0.035 radians produces very similar eigenvalues (0.88, 0.70, $0.43 \pm 0.01i$, $0.36 \pm 0.08i$, 0.21). We have also studied the associated eigenvectors, but find them difficult to interpret since they are sensitive to the units and scale of our data. The largest eigenvalue of 0.88 indicates that this system is locally stable.

The distribution of equilibrium trajectories was unimodal and narrow for both slopes (examined separately). We believe that most of the variance in the distribution of equilibrium trajectories can be accounted for by sensor noise, small disturbances, and changes in the effective ramp angle when the robot yaws to one side or the other.

Domain of Attraction

In practice, the robot can be initialized from a large range of initial conditions and can recover from relatively large perturbations. Because the return map has some eigenvalues close to 1, this recovery takes many steps. Walking trials are nearly always started with the robot tilted sideways to a non-zero θ_{roll} position but with θ_{yaw}, θ_{lPitch}, and θ_{rPitch} close to zero. It is not necessary to give the robot any initial forward velocity.

When started in this configuration, one of three things happen. If $|\theta_{roll}|$ is too small, approximately less than 1.25ϕ, then the robot will converge to a stable fixed point at $\theta_{roll} = \dot{\theta}_{roll} = 0$. If $2\phi < |\theta_{roll}| < \psi$, where ψ is the angle at which the center of mass of the robot is directly above the outside edge of the foot, then the robot returns to a stable gait. For larger $|\theta_{roll}|$, the robot falls over sideways. On our robot, $\phi = 0.03$ radians and $\psi = 0.49$ radians, which makes for a very large basin of attraction in this dimension. Compare this with the predictions of the compass gait model, which must be initialized much closer to the steady state trajectory in order to produce stable walking.

4.6 Adding Arms

The most recent versions of our passive walkers have arms that are mechanically coupled to the opposite leg, and therefore did not increase the number of degrees of freedom of the robot. These arms help to reduce the total moment in the yaw (causing the robot to turn) produced by the swinging leg. This trivial addition made the robot walk much straighter, and it also gave us a convenient location for mounting the batteries on the actuated version of the robot.

4.7 Discussion

The rimless wheel represents the simplest model of passive dynamic walking, and the only model for which we have a complete analytical understanding. Until we find an analytical expression for the return map of the compass gait[2], we must resort to numerical analysis techniques which include local stability analysis using eigenvalues and a more global analysis by plotting the basins of attraction of the limit cycle.

[2][Coleman, 1998] approximates the compass gait map by taking the limits as the mass of the legs goes to zero and linearizing around the fixed point.

The planar models that were constructed of the curved foot walker pictured in Figure 4-1 ignored rolling friction, assumed that the collisions with the ground were completely inelastic, and most importantly, ignored the dynamic coupling between the frontal and sagittal planes. Despite these assumptions, the models were accurate enough to predict the step frequency and step length of the walker with a high enough fidelity to use these models in the design process.

Simulation code for all of the models presented in this chapter are available at online at http://hebb.mit.edu/people/russt/ .

Chapter 5

Actuating a Simple 3D Passive Dynamic Walker

Ideas from passive dynamic walking are only beginning to have an impact on the way that fully actuated bipedal robots are designed and controlled (i.e., [Pratt, 2000, Spong and Bhatia, 2003]). To bridge the gap between passive and active walkers, a number of researchers have investigated the problem of adding a small number of actuators to an otherwise passive device ([Camp, 1997, van der Linde, 1998, Kuo, 2002, Wisse and van Frankenhuyzen, 2003]). There are two major advantages to this approach. First, actuating a few degrees of freedom on an otherwise passive walker is a way to capitalize on the energy efficiency of passive walking and the robustness of actively controlled systems. Second, by allowing the dynamics of the system to solve a large portion of the control problem, it may be possible to simplify the control problem that is solved by the actuators. In this chapter, I will present Toddler - one of the first robots capable of both passive walking down a ramp and actuated walking on the flat. Two other robots, developed independently at Cornell University [Collins et al., 2001] and at the Delft University of Technology in the Netherlands [Wisse and van Frankenhuyzen, 2003], also crossed this milestone, all at approximately the same time [Collins et al., 2004].

The passive dynamics of the robot can be thought of as an approximate solution to the optimal control problem, which works well for walking on a ramp, but works poorly on flat terrain. Using this philosophy, I will begin this chapter by formulating this optimal control problem on the step-to-step return map. Next, I will present two hand-designed controllers which improve the stability of the passive gait and produce stable walking on flat terrain. Finally, I will present a learning system which is able to quickly acquire a controller, using only trials implemented on the physical robot, that is quantifiably more stable than any controller I could design by hand.

5.1 Toddler

The passive dynamic walker described in the last chapter and shown on the left in Figure 5-1 represents the simplest machine that we could build which captures the essence of stable dynamic walking in three dimensions. It has only a single passive pin joint at the hip. We designed our learning robot by adding a small number of actuators to this passive design. The robot shown on the right in figure 5-1 has passive joints at the hip and 2 degrees of actuation (roll and pitch) at each ankle. The ankle actuators are position controlled servo

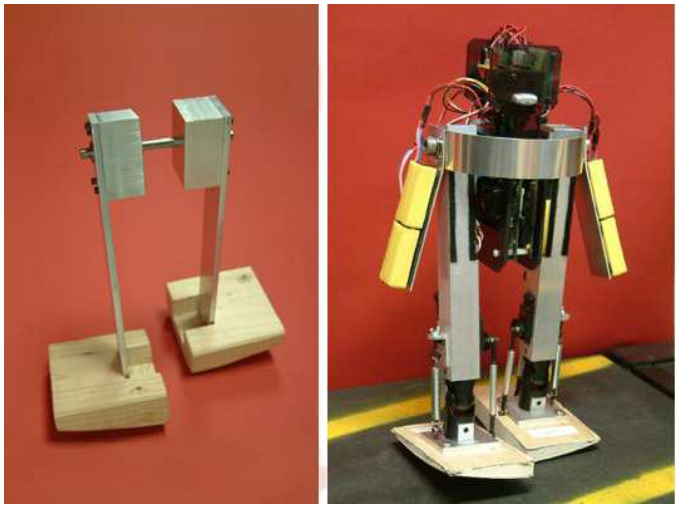

Figure 5-1: The robot on the left is a simple passive dynamic walker. The robot on the right is our actuated version of the same robot.

motors which, when commanded to hold their zero position, allow the actuated robot to walk stably down a small ramp, "simulating" the passive walker. The shape of the feet is designed using the planar walking models from the last chapter to make the robot walk passively at 0.8Hz, and to take steps of approximately 6.5 cm when walking down a ramp of 0.03 radians. The robot stands 44 cm tall and weighs approximately 2.9 kg, which includes the CPU and batteries that are carried on-board. This photo also illustrates the passive arms which are mechanically coupled to the opposite leg, and therefore do not add any degrees of freedom to the robot. We call this robot "Toddler" because the word is normally used to describe a child during the time that they are learning to walk, and this robot is primarily designed to investigate learning algorithms for dynamic walking. The name is also appropriate because the robot literally toddles back and forth when it walks.

When placed on flat terrain, the passive walker waddles back and forth, slowly losing energy, until it comes to rest standing still. In order to achieve stable walking on flat terrain, the actuators on our learning robot must restore energy into the system that would have been restored by gravity when walking down a slope.

5.2 The Optimal Control Problem

The goal of control is to apply torques at the ankles in order to produce a stable limit cycle gait that is invariant to small slopes. In total, the system has 9 degrees of freedom[1], and the equations of motion can be written in the form

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \tau + \mathbf{D}(t) \tag{5.1}$$

$$\dot{\mathbf{q}}^+ = \mathbf{\Omega}(\mathbf{q})\dot{\mathbf{q}}^- \tag{5.2}$$

[1]6 internal DOFs and 3 DOFs for the robot's orientation. We assume that the robot is always in contact with the ground at a single point, and infer the robot's absolute (x, y) position in space directly from the remaining variables.

where

$$\begin{aligned}\mathbf{q} =&[\theta_{yaw}, \theta_{lPitch}, \theta_{bPitch}, \theta_{rPitch}, \theta_{roll},\\ &\theta_{raRoll}, \theta_{laRoll}, \theta_{raPitch}, \theta_{laPitch}]^T,\\ \tau =&[0, 0, 0, 0, \tau_{raRoll}, \tau_{laRoll}, \tau_{raPitch}, \tau_{laPitch}]^T.\end{aligned}$$

H is the state dependent inertial matrix, **B** contains interaction torques between the links, **G** represents the effect of gravity, τ are the motor torques, **D** are random disturbances to the system, and $\mathbf{\Omega}$ represents the collision model. Our shorthand $lPitch$, $bPitch$, and $rPitch$ refer to left leg pitch, body pitch, and right leg pitch, respectively. $raRoll$, $laRoll$, $raPitch$, and $laPitch$ are short for right and left ankle roll and pitch. The robot uses a control policy

$$\mathbf{u} = \pi(\hat{\mathbf{x}}, t), \text{ with } \mathbf{x} = \begin{bmatrix}\mathbf{q}\\ \dot{\mathbf{q}}\end{bmatrix}. \tag{5.3}$$

The notation $\hat{\mathbf{x}}$ represents a noisy estimate of the state $\mathbf{x}$. The actual output of the controller is a motor command vector

$$\mathbf{u} = [u_{raRoll}, u_{laRoll}, u_{raPitch}, u_{laPitch}]^T,$$

which generates torques

$$\tau = h(\mathbf{x}, \mathbf{u}). \tag{5.4}$$

The function h describes the linear feedback controller implemented by the servo boards and the nonlinear kinematic transformation into joint torques.

To quantify the stability of our nonlinear, stochastic, periodic trajectory, we consider the dynamics on the return map, taken around the point where $\theta_{roll} = 0$ and $\dot{\theta}_{roll} > 0$. The return map dynamics are a Markov random sequence with the probability at the $(n+1)$th crossing of the return map given by

$$F_\pi(\mathbf{x}', \mathbf{x}) = P\{\hat{\mathbf{X}}_{n+1} = \mathbf{x}' | \hat{\mathbf{X}}_n = \mathbf{x}, \pi\}. \tag{5.5}$$

$F_\pi(\mathbf{x}', \mathbf{x})$ represents the probability density function over the state space which contains the dynamics in equations 5.1 - 5.4 integrated over one cycle. We do not make any assumptions about its form, except that it is Markov. Note that for this robot, the element of F_π representing θ_{roll} is the delta function, independent of $\mathbf{x}$. The stochasticity in F_π comes from the random disturbances $\mathbf{D}(t)$ and from sensor noise, $\hat{\mathbf{x}} - \mathbf{x}$.

The instantaneous cost that we would like to minimize uses a constant desired value, $\mathbf{x}^d$, on the return map:

$$g(\mathbf{x}) = \frac{1}{2}|\mathbf{x} - \mathbf{x}^d|^2. \tag{5.6}$$

This desired value can be considered a reference trajectory on the return map, and is taken from the gait of the walker down a slope of 0.03 radians; no reference trajectory is required for the limit cycle between steps. For a given trajectory $\hat{\mathrm{x}} = [\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N]$, we define the average cost

$$C(\hat{\mathrm{x}}) = \frac{1}{N}\sum_{n=0}^{N} g(\hat{\mathbf{x}}_n). \tag{5.7}$$

Our goal is to find the policy π which minimizes

$$\lim_{N\to\infty} E\left\{C(\hat{\mathbf{x}})\right\}. \tag{5.8}$$

By minimizing this error, we are effectively minimizing the eigenvalues of return map, and maximizing the stability of the desired limit cycle.

5.3 Hand-designed controllers

The optimal control problem can be solved using a variety of techniques. In the next sections, I will focus on learning a parameterized policy in a general function approximator, but in this section I will consider some hand-designed alternatives to learning control.

With nine degrees of freedom and only four actuators, the robot is clearly an under-actuated system, and there are no obvious methods for applying standard robotic manipulator control solutions. A solution to the optimal control problem will have to somehow use the ankle actuators to control (directly or indirectly) all 9 degrees of freedom. To solve this problem, we first focus our attention on stabilizing the oscillation in the frontal plane. The frontal plane model from the last chapter is a simplification of the dynamics of the robot on a ramp, but it is also a reasonable representation of the robot's dynamics when it is placed on a flat surface. The frontal plane model for the actuated version can be written as:

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

where $\mathbf{q} = [\theta, \theta_{la}, \theta_{ra}]^T$ and $\mathbf{u} = [0, u_{la}, u_{ra}]^T$. The abbreviations la and ra are short for left and right ankle, respectively. At each collision with the ground, the kinetic energy of the system, T, changes by:

$$\Delta T = \frac{1}{2}\dot{\mathbf{q}}^T \left[\mathbf{\Omega}(\mathbf{q})^T \mathbf{H}(\mathbf{q})\mathbf{\Omega}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\right]\dot{\mathbf{q}}.$$

In order to stabilize the oscillations, the control torques, $\mathbf{u}$, must restore the energy lost from these collisions.

5.3.1 Feed Forward Ankle Trajectories

The first actuation strategy that I experimented with use the idea of coupled oscillators. The mechanical system in the frontal plane is a damped oscillator. By producing a second oscillator in the control system which can apply forces to the robot, we hope that the control oscillator can entrain the dynamics of the mechanical oscillator - applying forces at just the right time to produce stable walking. This is a degenerate form of true oscillator-based control where we only consider a one-way coupling between the oscillators.

The oscillator we used is given by:

$$u_{ra} = \alpha \sin(2\pi\omega t)$$
$$u_{la} = -u_{ra},$$

where ω is the oscillator frequency and α is a control gain. Surprisingly, this oscillation was sufficient to stabilize the walking when ω was properly related to the natural stepping

frequency of the robot. The best entrainment occurred when the controller was a little slower than the passive step frequency ($\omega = 0.55$ Hz for our robot stepping at 0.8 Hz). The success of this simple controller is our first piece of evidence that mechanical designs based on passive dynamic walking can simplify control design for actuated walkers.

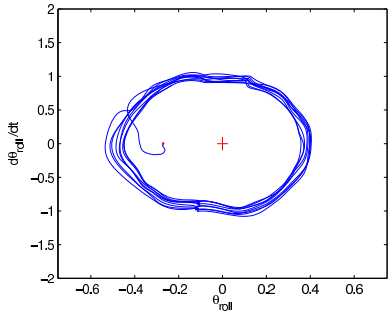

Figure 5-2: Limit cycle trajectory using the feed-forward controller

Using the techniques presented in the last chapter, we performed an online experimental local stability analysis by fitting a linear model to data collected on the return map. Unlike the passive robot, the actuated robot has 9 total degrees of freedom: 4 degrees of freedom in common with the passive robot, 2 extra degrees of freedom in each ankle, and an extra degree of freedom for the computer which hangs passively from the hip. The 4 degrees of freedom in the ankles are rigidly connected to our position controlled actuators and produce repeatable trajectories from step to step. For that reason, I don't even have the ankle potentiometers wired to the main computer, and I have left them out of the stability analysis. The return map data is recorded and fit to a linear model in 9 dimensions (5 states + 5 velocities - 1 Poincaré section).

The local stability analysis of the limit cycles generated by the feed forward controller suggests that this controller is more stable than the passive system. The eigenvalues of the return map evaluated from 89 trials on flat terrain were 0.80, 0.60, $0.49 \pm 0.04i$, 0.36, 0.25, $0.20 \pm 0.01i$, 0.01. Practically, this controller converges from a variety of initial configurations of the robot, but is very sensitive to disturbances in phase. The robot must be initialized in phase with the controller, and relatively small perturbations can knock it out of phase.

5.3.2 Feedback Ankle Trajectories

A more direct approach to stabilizing the roll oscillations is to build a controller which, on every cycle, injects exactly the amount of energy into the system that was lost during that cycle. Even simpler is the idea implemented by Marc Raibert's height controller for hopping robots ([Raibert, 1986, Buehler and Koditschek, 1988]): if we inject a roughly constant amount of energy into the system during every cycle, then system will settle into a stable oscillation with an amplitude that is monotonically related to the energy injected.

Our heuristic for injecting a roughly constant amount of energy on each cycle is implemented using a state machine with only two states. The right ankle is given by:

$$u_{ra} = \begin{cases} \alpha & \text{if } \theta_{roll} > \theta_1 \text{ and } \dot{\theta}_{roll} > \omega_1 \\ 0 & \text{otherwise,} \end{cases}$$

and the left ankle controller is symmetric. α is the desired ankle thrust position, θ_1 is the

roll threshold, and ω_1 is the angular roll velocity threshold. Typical values are $\alpha = 0.08$ radians, $\theta_1 = 0.1$ radians, and $\omega_1 = 0.5$ radians/second.

With this state machine, as the robot rolls from an upright position onto one foot, it crosses a threshold position and velocity at which the stance ankle is servoed by a small fixed amount, causing the robot to accelerate further in the direction that it was moving. As the robot is moving back toward the upright position, the ankle is servoed back to the original zero position, which further accelerates the center of mass toward the upright position. Both ankles are at their zero position when the robot is transferring support to the opposite foot in order to minimize the energy lost by the collision with the ground. The desired angle in the non-zero state is monotonically related to the resulting amplitude of oscillation.

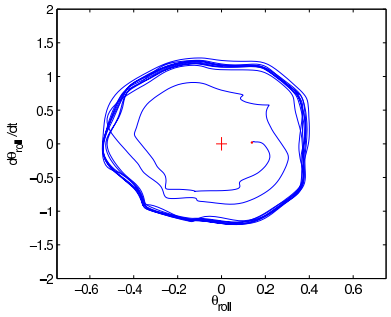

Figure 5-3: Limit cycle trajectory using the feedback controller. Notice that this trajectory demonstrates a gradual convergence to the limit cycle from small initial conditions.

The local stability analysis reveals that this controller converges more quickly than both the feed-forward controller and the purely passive device. After 58 trials on flat terrain, our linear return map analysis produced the eigenvalues: 0.78, $0.69 \pm 0.03i$, 0.44, $0.36 \pm 0.04i$, $0.13 \pm 0.06i$, 0.13. The controller is able to slowly recover from a range of perturbations, but requires many steps to return to steady state. It is not able to initiate walking when the robot is standing still, and never applies any breaking forces to slow the robot down and prevent it from falling over sideways.

5.3.3 Velocity Control

We control the dynamics of the sagittal plane independently using a simple velocity control algorithm. The forward velocity of the robot, regardless of the slope of terrain, depends on the location of the center of mass relative to the ground contact. When the center of mass is out in front of the ground contact point, the robot will lean forward. As soon as one leg leaves the ground, the passive joint at the hip allows it to swing forward, and the robot begins walking. The farther the center of mass is from the ground contact point, the faster the robot will move in that direction. On Toddler, the operator specifies the desired forward speed by joystick, and the corresponding placement of the center of mass is controlled by actuating the ankle pitch actuators. The heuristic makes it easy for Toddler to walk on flat terrain, and even up small inclines. For large changes in slope, the gains of the roll stabilizing controllers must also adapt.

The direction of the robot can also be controlled, to a limited degree, by differentially actuating the right and left ankles (either pitch or roll). The effectiveness of this direction control is limited, however, by the uncontrolled moment in the yaw axis generated by the

swinging leg. When the frictional contact between the robot and the ground does not balance this moment, then the robot turns. To minimize this effect, we have added rubber soles to the robot's feet to increase the surface contact with the ground. With the rubber soles, the robot walks well on the lab's linoleum flooring, but it still walks much better on a firm carpet or rubber surface.

5.3.4 Summary

To control this under-actuated system, we have developed two simple hand-designed control algorithms which stabilize the passive gait on a small slope or even on flat terrain. When walking down the linoleum hallways of MIT, which are not nearly as flat and even as they appear, the feed-forward controller needs to be restarted fairly frequently. The feedback controller rarely needs to be restarted on the linoleum, but does not adjust well to different surfaces, like carpet or wooden tiles. The velocity controller works well in a variety of situations, and will be used for the remainder of the experiments described in this thesis. The robot could walk faster by using a more sophisticated controller in the sagittal plane, potentially by pushing off with the toes at the end of every step, but this improvement has been left for future work.

5.4 The Learning Algorithm

To improve the robot's performance, and to investigate online learning algorithms on this simple platform, we replaced the hand-designed control policies with a deterministic feedback control policy stored in a linear function approximator. The function approximator is parameterized by the vector $\mathbf{w}$ and uses nonlinear features ϕ:

$$\mathbf{u} = \pi_{\mathbf{w}}(\hat{\mathbf{x}}) = \sum_i w_i \phi_i(\hat{\mathbf{x}}).$$

Before learning, $\mathbf{w}$ is initialized to all zeros, making the policy outputs zero everywhere, so that the robot simulates the passive walker.

The learning algorithm is an actor-critic algorithm (recall section 2.5) which makes small changes to the control parameters $\mathbf{w}$ on each step and uses correlations between changes in $\mathbf{w}$ and changes in the return map error to climb the performance gradient. This policy gradient update is augmented with an estimate of the value function, $J(\mathbf{x})$, which is used to reduce the variance of the policy update. Recall that the *value* of state $\mathbf{x}$ is the expected average cost to be incurred by following policy $\pi_{\mathbf{w}}$ starting from state $\mathbf{x}$:

$$J(\mathbf{x}) = \lim_{N\to\infty} \frac{1}{N} E\{\sum_{n=0}^{N} g(\mathbf{x}_n)\}, \text{ with } \mathbf{x}_0 = \mathbf{x}.$$

$\hat{J}_{\mathbf{v}}(\mathbf{x})$ is an estimate of the value function parameterized by vector $\mathbf{v}$. This value estimate is represented in another function approximator:

$$\hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}) = \sum_i v_i \psi_i(\hat{\mathbf{x}}). \tag{5.9}$$

The particular algorithm that we present here was originally proposed by [Kimura and Kobayashi, 1998]. We present a thorough derivation of this algorithm in the next section.

During learning, we add stochasticity to our deterministic control policy by varying $\mathbf{w}$. Let $\mathbf{Z}_n$ be a Gaussian random vector with $E\{Z_{i,n}\} = 0$ and $E\{Z_{i,n}Z_{j,n'}\} = \sigma^2\delta_{ij}\delta_{nn'}$. During the nth step that the robot takes, we evaluate the controller using the parameter vector $\mathbf{w}'_n = \mathbf{w}_n + \mathbf{z}_n$. The algorithm uses a storage variable, $\mathbf{e}_n$, which we call the eligibility trace. We begin with $\mathbf{w}_0 = \mathbf{e}_0 = \mathbf{0}$. At the end of the nth step, we make the updates:

$$d_n = g(\hat{\mathbf{x}}_n) + \gamma \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_{n+1}) - \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_n) \tag{5.10}$$

$$e_{i,n} = \gamma e_{i,n-1} + b_{i,n} z_{i,n} \tag{5.11}$$

$$\Delta w_{i,n} = -\eta_w d_n e_{i,n} \tag{5.12}$$

$$\Delta v_{i,n} = \eta_v d_n \psi_i(\hat{\mathbf{x}}_n). \tag{5.13}$$

$\eta_w \geq 0$ and $\eta_v \geq 0$ are the learning rates and γ is the discount factor of the eligibility trace, which will be discussed in more detail in the algorithm derivation. $b_{i,n}$ is a boolean one-step eligibility, which is 1 if the parameter w_i is activated ($\phi_i(\hat{\mathbf{x}}) > 0$) at any point during step n and 0 otherwise. d_n is called the one-step *temporal difference error*.

The algorithm can be understood intuitively. On each step the robot receives some cost $g(\hat{\mathbf{x}}_n)$. This cost is compared to cost that we expect to receive, as estimated by $\hat{J}_{\mathbf{v}}(\mathbf{x})$. If the cost is lower than expected, then $-\eta d_n$ is positive, so we add a scaled version of the noise terms, z_i, into w_i. Similarly, if the cost is higher than expected, then we move in the opposite direction. This simple online algorithm performs approximate stochastic gradient descent on the expected value of the average infinite-horizon cost.

5.5 Algorithm Derivation

The expected value of the average cost, C, is given by:

$$E\{C(\hat{\mathrm{x}})\} = \int_{\hat{\mathrm{x}}} C(\hat{\mathrm{x}}) P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} d\hat{\mathrm{x}}.$$

The probability of trajectory $\hat{\mathrm{x}}$ is

$$P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} = P\{\hat{\mathbf{X}}(0) = \hat{\mathbf{x}}(0)\} \prod_{n=0}^{N-1} F_{\mathbf{w}'}(\hat{\mathbf{x}}(n+1), \hat{\mathbf{x}}(n)),$$

where $F_{\mathbf{w}}$ is shorthand for $F_{\pi_{\mathbf{w}}}$. Taking the gradient of $E\{C(\hat{\mathrm{x}})\}$ with respect to $\mathbf{w}$ we find

$$\begin{aligned}
\frac{\partial}{\partial w_i} E\{C(\hat{\mathrm{x}})\} &= \int_{\hat{\mathrm{x}}} C(\hat{\mathrm{x}}) \frac{\partial}{\partial w_i} P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} d\hat{\mathrm{x}} \\
&= \int_{\hat{\mathrm{x}}} C(\hat{\mathrm{x}}) P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} \frac{\partial}{\partial w_i} \log P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} d\hat{\mathrm{x}} \\
&= E\left\{ C(\hat{\mathrm{x}}) \frac{\partial}{\partial w_i} \log P_{\mathbf{w}'}\{\hat{\mathbf{X}} = \hat{\mathrm{x}}\} \right\} \\
&= E\left\{ C(\hat{\mathrm{x}}) \left(\sum_{m=0}^{N-1} \frac{\partial}{\partial w_i} \log F_{\mathbf{w}'}(\hat{\mathbf{x}}_{m+1}, \hat{\mathbf{x}}_m) \right) \right\}
\end{aligned}$$

Recall that $F_{\mathbf{w}'}(\mathbf{x}', \mathbf{x})$ is a complicated function which includes the integrated dynamics

of the controller and the robot. Nevertheless, $\frac{\partial}{\partial w_i} \log F_{\mathbf{w}'}$ is simply:

$$\begin{aligned}
&\frac{\partial}{\partial w_i} \log F_{\mathbf{w}'}(\mathbf{x}'_{m+1}, \mathbf{x}_m) = \\
&\quad \frac{\partial}{\partial w_i} \log \left[P\{\hat{\mathbf{X}}' = \mathbf{x}' | \hat{\mathbf{X}} = \mathbf{x}, \mathbf{W}' = \mathbf{w}'\} P_{\mathbf{w}}\{\mathbf{W}' = \mathbf{w}'\} \right] \\
&= \frac{\partial}{\partial w_i} \log P_{\mathbf{w}}\{\mathbf{W}'_m = \mathbf{w}'\} = \frac{z_i(m)}{\sigma^2}
\end{aligned}$$

Substituting, we have

$$\begin{aligned}
\frac{\partial}{\partial w_i} E\{C(\hat{\mathbf{x}})\} &= \frac{1}{N\sigma^2} E\left\{ \left(\sum_{n=1}^{N} g(\hat{\mathbf{x}}_n) \right) \left(\sum_{m=0}^{N-1} z_{i,m} \right) \right\} \\
&= \frac{1}{N\sigma^2} E\left\{ \sum_{n=0}^{N} g(\hat{\mathbf{x}}_n) \sum_{m=0}^{n} b_{i,m} z_{i,m} \right\}.
\end{aligned}$$

This final reduction is based on the observation that $E\{g(\hat{\mathbf{x}}_n) \sum_{m=n}^{N} z_{i,m}\} = 0$ (noise added to the controller on or after step n is not correlated to the cost at step n). Similarly, random changes to a weight that is not used during the nth step ($b_{i,m} = 0$) have zero expectation, and can be excluded from the sum.

Observe that the variance of this gradient estimate grows without bound as $N \to \infty$ [Baxter and Bartlett, 2001]. To bound the variance, we use a *biased* estimate of this gradient which artificially discounts the eligibility trace:

$$\begin{aligned}
\frac{\partial}{\partial w_i} E\{C(\hat{\mathbf{x}})\} &\approx \frac{1}{N\sigma^2} E\left\{ \sum_{n=0}^{N} g(\hat{\mathbf{x}}_n) \sum_{m=0}^{n} \gamma^{n-m} b_{i,m} z_{i,m} \right\} \\
&= \frac{1}{N\sigma^2} E\left\{ \sum_{n=0}^{N} g(\hat{\mathbf{x}}_n) e_{i,n} \right\},
\end{aligned}$$

with $0 \leq \gamma \leq 1$. The discount factor γ parameterizes the bias-variance trade-off.

Next, observe that we can subtract any mean zero baseline from this quantity without effecting the expected value of our estimate [Williams, 1992]. Including this baseline can dramatically improve the performance of our algorithm because it can reduce the variance of our gradient estimate. In particular, we subtract a mean-zero term containing an estimate of the value function as recommended by [Kimura and Kobayashi, 1998]:

$$\lim_{N\to\infty} \frac{\partial}{\partial w_i} E\{C(\hat{\mathbf{x}})\} \approx \lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} g(\mathbf{x}_n) e_{i,n} - \sum_{n=0}^{N} \hat{J}_{\mathbf{v}}(\mathbf{x}_n) z_{i,n} \right\}.$$

The terms $\hat{J}_{\mathbf{v}}(\mathbf{x}_n) z_{i,n}$ have zero expectation because the value estimate of the state $\mathbf{x}_n$ is built up from previous trials, and is uncorrelated with the noise that is being injected on this particular step. Through some algebraic manipulation, we convert this equation into a

an efficient online update:

$$\begin{aligned}
\lim_{N\to\infty} \frac{\partial}{\partial w_i} E\{C(\hat{\mathrm{x}})\} &\approx \lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} g(\mathbf{x}_n) e_{i,n} - \sum_{n=0}^{N} \hat{J}_{\mathbf{v}}(\mathbf{x}_n) z_{i,n} \right\} \\
&= \lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} g(\mathbf{x}_n) e_{i,n} + \sum_{n=0}^{N} z_{i,n} \sum_{m=n}^{N-1} \gamma^{m-n} \left(\gamma \hat{J}_{\mathbf{v}}(\mathbf{x}_{m+1}) - \hat{J}_{\mathbf{v}}(\mathbf{x}_m) \right) \right\} \\
&= \lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} g(\mathbf{x}_n) e_{i,n} + \sum_{n=0}^{N} \left(\gamma \hat{J}_{\mathbf{v}}(\mathbf{x}_{n+1}) - \hat{J}_{\mathbf{v}}(\mathbf{x}_n) \right) e_{i,n} \right\} \\
&= \lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} d_n e_{i,n} \right\}
\end{aligned}$$

By this derivation, we can see that the average of the weight update given in equations 5.10-5.13 is in the direction of the performance gradient:

$$\lim_{N\to\infty} \frac{1}{N} E\left\{ \sum_{n=0}^{N} \Delta w_{i,n} \right\} \approx -\eta \lim_{N\to\infty} \frac{\partial}{\partial w_i} E\{C(\hat{\mathrm{x}})\}.$$

The value estimate update given in equation 5.13 is the well-known update for the temporal difference learning algorithm, TD(λ), with $\lambda = 0$. This update performs approximate gradient descent on the value parameters to minimize the squared temporal difference error:

$$\begin{aligned}
\frac{\partial}{\partial v_i} \frac{1}{2} d_n^2 =& \frac{\partial}{\partial v_i} \frac{1}{2} \left[g(\hat{\mathbf{x}}_n) + \gamma \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_{n+1}) - \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_n) \right]^2 \\
=& - d_n \left[\gamma \frac{\partial \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_{n+1})}{\partial v_i} + \frac{\partial \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_n)}{\partial v_i} \right] \\
\approx& - d_n \frac{\partial \hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_n)}{\partial v_i} \\
=& - d_n \psi_i(\hat{\mathbf{x}}_n).
\end{aligned}$$

By neglecting the contribution of the $\hat{J}_{\mathbf{v}}(\hat{\mathbf{x}}_{n+1})$ term to the gradient, this algorithm looses guarantees of convergence (it is no longer performing true gradient descent). Nevertheless, there is significant experimental evidence in the literature suggesting that this assumption, which makes the update a backup of future rewards, can significantly improve the rate of convergence in practice [Sutton and Barto, 1998].

5.6 Learning Implementation

As with the hand-designed controllers, we decompose the control problem for learning into the frontal and sagittal planes. The joystick velocity control in the sagittal plane was simple and effective, so we continue to use it here. Therefore, the goal of learning is to acquire a policy for the roll actuators to stabilize the oscillation in the frontal plane. The ideal control outputs change as the robot walks at different speeds, but we hope that the learning algorithm will adapt quickly enough to compensate for those differences.

The learning algorithm described in the last sections was chosen because it works on-

line, is robust to noise, and provably performs gradient descent on the performance gradient. Another feature of the algorithm is that it performs well even when we limit the representational power of the function approximators for the policy and the value estimate. The policy gradient update will attempt to find a local minimum in performance error within the function class of the approximator. The convergence of the algorithm does not depend on the quality of the value estimate, but the better our estimate, the faster the convergence.

The focus of this work is on an algorithm that works quickly enough to be implemented on the real robot. For that reason, we are willing to make some sacrifices on the optimality of the learned controller in order to improve the learning speed. This can be accomplished by constraining the feedback policy to be a function of only two variables: θ_{roll} and $\dot{\theta}_{roll}$. The choice of these two variables is not arbitrary; they are the only variables that we used when writing a non-learning feedback controller that stabilizes the oscillation. The policy is also constrained to be symmetric - the controller for the left ankle is simply a mirror image of the controller for the right ankle. With these reductions, the learned control policy has only two inputs and only a single output. The simplifications made in the value estimate are even more severe - we approximate the value as a function of only a single variable: $\dot{\theta}_{roll}$. This extremely low dimensionality allows the algorithm to very quickly learn a very coarse approximation to the real value function, and turns out to be good enough to significantly reduce the variance of our policy update and allow for very fast learning.

The control policy and value functions are both represented using linear function approximators of the form described in Equations 5.3 and 5.9, which are fast and very convenient to initialize. We use an overlapping tile-coding for our approximator basis functions: 35 tiles for the policy (5 in $\theta_{roll}\times$ 7 in $\dot{\theta}_{roll}$) and 11 tiles for the value function.

In order to make the robot explore the state space during learning, we hand-designed a simple controller to place the robot in random initial conditions on the return map. These initial conditions are selected from a random distribution that is biased according to the distribution of points that the robot has already experienced on the return map - the most likely initial condition is the state that the robot experienced least often. We use this controller to randomly reinitialize the robot every time that it comes to a halt standing still, or every 10 seconds, whichever comes first. This heuristic makes the distribution on the return map more uniform, and increases the likelihood of the algorithm converging on the same policy each time that it learns from a blank slate.

5.7 Experimental Results

When the learning begins, the policy parameters, $\mathbf{w}$, are set to $\mathbf{0}$ and the baseline parameters, $\mathbf{v}$, are initialized so that $\hat{J}_{\mathbf{v}}(\mathbf{x}) \approx \frac{g(\mathbf{x})}{1-\gamma}$. This is the total discounted cost that we would expect to receive if $\mathbf{x}$ was a fixed point of the dynamics. We begin the training with the robot walking in place on flat terrain using short trials with random initial conditions. During the first few trials, the policy does not restore sufficient energy into the system, and the robot comes to a halt standing still. Within a minute of training, the robot achieves foot clearance on nearly every step; this is the minimal definition of walking on this system. The learning easily converges to a robust gait with the desired fixed point on the return map within 20 minutes (approximately 960 steps at 0.8 Hz). Error obtained during learning depends on the random initial conditions of each trial, and is therefore a very noisy stochastic variable. For this reason, in Figure 5-4 we plot a typical learning curve in terms of the *average* error per step. Figure 5-5 plots a typical trajectory of the learned controller

walking on flat terrain. Figure 5-6 displays the final policy.

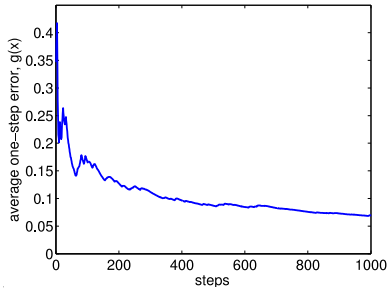

Figure 5-4: A typical learning curve, plotted as the average error on each step.

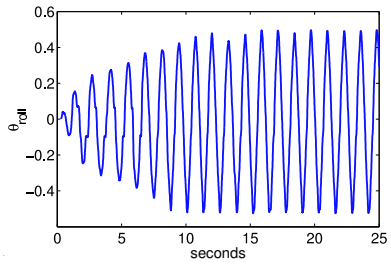

Figure 5-5: θ_{roll} trajectory of the robot starting from standing still.

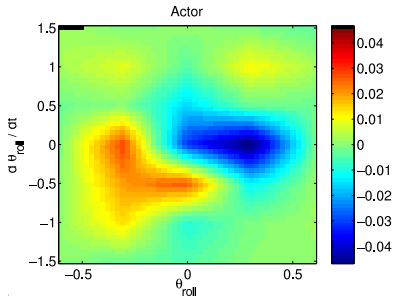

Figure 5-6: Learned feedback control policy $u_{raRoll} = \pi_{\mathbf{w}}(\hat{\mathbf{x}})$.

In Figure 5-7 we plot the return maps of the system before learning ($\mathbf{w} = \mathbf{0}$) and after 1000 steps. Recall that we are fitting a return map in 9 dimensions (5 states + 5 derivatives - 1 Poincaré section). In general, the projection of these dynamics onto a single dimension is difficult to interpret. The plots in Figure 5-7 where made with the robot walking in place on flat terrain. In this particular situation, most of the return map variables are close to zero throughout the dynamics, and a two dimensional return map captures the desired dynamics. As expected, before learning the return map illustrates a single fixed point at $\dot{\theta}_{roll} = 0$, which means the robot is standing still. After learning, we obtain a single fixed point at the desired value ($\dot{\theta}_{roll} = 1.0$ radians / second), and the basin of attraction of this fixed point extends over the entire domain that we tested. On the rare occasion that the robot falls over, the system does not return to the map and stops producing points on this graph.

Looking at Figure 5-7, we might guess that the controller has a single eigenvalue of 0.5.

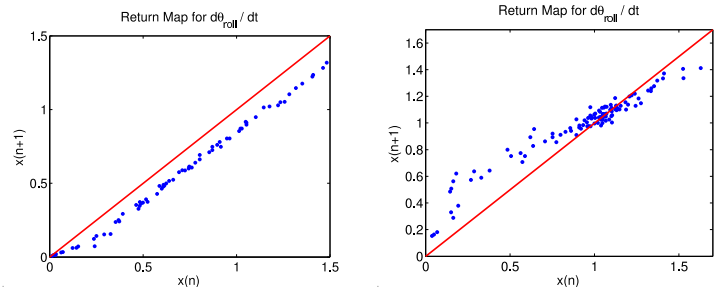

Figure 5-7: Experimental return maps, before (left) and after (right) learning. Fixed points exist at the intersections of the return map data and the diagonal line of slope one.

The linear least-squares eigenvalue analysis results for the learned controller, and all of the other controllers are:

Controller	Eigenvalues
Passive walking (63 trials)	$0.88 \pm 0.01i$, 0.75, $0.66 \pm 0.03i$, 0.54, 0.36, $0.32 \pm 0.13i$
Hand-designed feed-forward (89 trials)	0.80, 0.60, $0.49 \pm 0.04i$, 0.36, 0.25, $0.20 \pm 0.01i$, 0.01
Hand-designed feedback (58 trials)	0.78, $0.69 \pm 0.03i$, 0.36, 0.25, $0.20 \pm 0.01i$, 0.01
Learned feedback (42 trials)	$0.74 \pm 0.05i$, $0.53 \pm 0.09i$, 0.43, $0.30 \pm 0.02i$, 0.15, 0.07

All of these experiments were on flat terrain except the passive walking, which was on a slope of 0.027 radians. The convergence of the system to the nominal trajectory is largely governed by the largest eigenvalues. This analysis suggests that our learned controller converges to the steady state trajectory more quickly that the passive walker on a ramp and more quickly than any of our hand-designed controllers.

The results for walking in place on flat terrain were encouraging, so we tested the controller walking at different velocities and over different terrain. While it took a few minutes to learn a controller from a blank slate, adjusting the learned controller to adapt to slow changes in the joystick velocity command or to small changes in the terrain appears to happen very quickly - often within a few steps. While the non-learning controllers require constant attention and small manual changes to the parameters as the robot walks down the hall, on tiles, and on carpet. The learning controller easily adapts to each of these situations.

5.8 Discussion

Designing our robot like a passive dynamic walker changed the learning problem in a number of ways. Because each of the dynamic variables for the robot are coupled through the passive dynamics and because these dynamics already solve a large portion of the control problem, it was possible for us to learn a policy with only a single output which controlled a 9 degree of freedom system. Unlike most bipeds, most of the policies in this class of single output functions produce some form of stable behavior - whether it is walking or standing. With a relatively low cost for experimenting with different policy parameters on the real robot, we were able to blindly inject noise into our policy for the actor-critic update. Another

benefit of the passive dynamic design of this robot is that we were able to obtain stable periodic trajectories even when the policy was initialized to a blank slate. This allowed us to formulate the learning problem on the return map dynamics with the confidence that most trajectories will return to the Poincaré section.

The return map formulation of the learning problem was a major factor in the success of this algorithm, but it has a number of implications. The return map dynamics are modelled as a Markov chain that is parameterized by the policy parameters instead of as a Markov decision process (MDP). This distinction is necessary because we evaluate the policy many times within a single step, and we cannot directly apply a large variety of reinforcement learning algorithms which rely on the structure of an MDP (such as dynamic programming or value iteration). The return map formulation also has implications for the solution of the delayed reward problem. Our algorithm solves the delayed reward problem by accumulating the eligibility within a step and discounting eligibility between steps. Interestingly, our algorithm performs best with heavy discounting between steps ($0 \leq \gamma \leq 0.2$). This is probably a result of the particular choice of cost function, which implicitly rewards monotonic convergence to the fixed point on the return map, and the fact that the robot was walking on flat terrain where multi-step maneuvers were not required to improve stability.

Another important factor in the success of this algorithm is the variance reduction using a coarse estimate of the value function. Before implementing this component, the learning took so many trials that we were not confident that the policy changes were actually improving performance. To demonstrate the importance of the learned value estimate, we can run the learning with the policy parameters reset to zero, but the value estimate parameters in tact from the previous learning session. When we do this, the robot achieves good performance in just a few steps, and the learning actually converges in just one or two minutes.

The learning algorithm works well on this simple robot, but will the technique scale to more complicated robots? As the number of degrees of freedom increases, the actor-critic algorithm implemented here may have problems with scaling. The algorithm correlates changes in the policy parameters with changes in the performance on the return map. As we add degrees of freedom, the assignment of credit to a particular actuator will become more difficult, requiring more learning trials to obtain a good estimate of the correlation. This scaling problem is an open and interesting research question and a primary focus of our current research.

Although it was accomplished on a simple biped, this demonstration of online learning is important for a number of reasons. It suggests that principles from passive dynamic walking can and should be used on actuated robots. It provides a real demonstration of policy gradient learning implemented on a real robot with sensor noise and stochasticity in the environment, and experimental evidence for the importance of variance reduction using a value estimate. Finally, it provides a fresh idea on the solution to the delayed reward problem for walking robots - accumulating eligibility evenly over one step and discounting heavily between steps.

Chapter 6

The Planar One-Leg Hopper

In the early 1980's, Marc Raibert's Leg Laboratory built a series of extremely successful hopping robots [Raibert, 1986] which redefined people's expectations for robots executing extremely dexterous and dynamic movements [Koditschek and Buehler, 1991]. The first of these robots, a planar one-leg hopper, is a simple legged system which captures the essence of dynamic stability but avoids the complication of coordinating multiple legs or solving complex leg kinematics.

Due to the carefully designed dynamics of the robot, Raibert's group was able to design a surprisingly simple controller for stable hopping using only their engineering intuition. Twenty years later, this controller is still one of the most prominent controllers for robotic running, even though surprisingly little is understood about it analytically. The only real analysis was done by [Koditschek and Buehler, 1991], who provided more insight into the height controller by providing a Lyapunov function for a reduced physical model.

Raibert's original controller works well for steady-state hopping, but is not able to recover from large disturbances. In this chapter, I will present Raibert's original controller as an approximate solution to an optimal control problem, and then apply a policy gradient algorithm to systematically improve upon it. The learned controller is able to recover from a dramatically larger region of initial conditions than the original controller.

6.1 Planar One-Leg Hopper

The one-leg hopper has a total of ten state variables $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$, where $\mathbf{q} = [x_{ft}, y_{ft}, \theta, \phi, r]$ as defined in Figure 6-1, and two control variables $\mathbf{u} = [\tau_{hip}, F_{leg}]^T$ which are the torque at the hip and the linear force from the pneumatic spring in the leg. The equations of motion use the following constants: m and J are mass and moment of inertia of the body, m_l and J_l are the mass and moment of inertia of the leg, l_1 is the distance from the foot to the center of mass of the leg, l_2 is the distance from the hip to the center of mass of the body, and $R = r - l_1$.

The equations of motion for the hopper, taken with corrections from [Raibert, 1986],

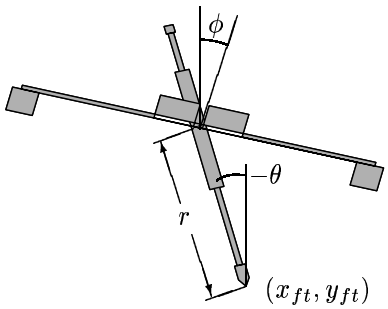

Figure 6-1: The planar one-legged hopper. (x_{ft}, y_{ft}) represents the position of the point foot in world coordinates. θ represents the absolute angle of the leg, ϕ represents the absolute angle of the body, and r represents the length of the leg.

are:

$$
\begin{aligned}
(J_l - m_l R l_1)\ddot{\theta}\cos\theta - m_l R\ddot{x}_{ft} &= \cos\theta(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}) \\
&\quad - R(F_x - F_{leg}\sin\theta - m_l l_1 \dot{\theta}^2 \sin\theta) \\
(J_l - m_l R l_1)\ddot{\theta}\sin\theta + m_l R\ddot{y}_{ft} &= \sin\theta(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}) \\
&\quad + R(m_l l_1 \dot{\theta}^2 \cos\theta + F_y - F_{leg}\cos\theta - m_l g) \\
(J_l + mRr)\ddot{\theta}\cos\theta + mR\ddot{x}_{ft} + mR\ddot{r}\sin\theta + mRl_2\ddot{\phi}\cos\phi &= \cos\theta(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}) \\
&\quad + RF_{leg}\sin\theta + mR(r\dot{\theta}^2 \sin\theta + l_2\dot{\phi}^2 \sin\phi - 2\dot{r}\dot{\theta}\cos\theta) \\
(J_l + mRr)\ddot{\theta}\sin\theta - mR\ddot{y}_{ft} - mR\ddot{r}\cos\theta + mRl_2\ddot{\phi}\sin\phi &= \sin\theta(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}) \\
&\quad - R(F_{leg}\cos\theta - mg) - mR(2\dot{r}\dot{\theta}\sin\theta + r\dot{\theta}^2\cos\theta + l_2\dot{\phi}^2\cos\phi) \\
J_l l_2 \ddot{\theta}\cos(\theta - \phi) - JR\ddot{\phi} &= l_2\cos(\theta - \phi)(l_1 F_y \sin\theta - l_1 F_x \cos\theta - \tau_{hip}) \\
&\quad - R(l_2 F_{leg}\sin(\phi - \theta) + \tau_{hip})
\end{aligned}
$$

The ground is modelled as a piecewise linear spring-damper system, with reaction forces given by

$$
F_x = \begin{cases} k_g(x_{ft} - x_{td}) - b_g\dot{x}_{ft} & \text{for } y_{ft} < 0, \\ 0 & \text{otherwise.} \end{cases}
$$

$$
F_y = \begin{cases} k_g y_{ft} - b_g\dot{y}_{ft} & \text{for } y_{ft} < 0, \\ 0 & \text{otherwise.} \end{cases}
$$

6.2 Raibert's Three-Part Controller

Raibert's original controller decomposed the problem into three parts, individually controlling forward velocity, body attitude, and hopping height. This was based on the observation that each of these variables can be controlled in an approximately independent way during different phases of the hopping cycle. The most important of these phases are flight, compression, and thrust.

The robot is built so that the moment of inertia of the body around the hip is much

larger than the moment of inertia of the leg. During the flight phase, this allows us to assume that hip torque causes the leg to move without altering the body attitude. The forward velocity is then controlled by the foot placement: using a symmetry argument to estimate the position on the ground that the foot should be placed for steady state hopping, a simple servo controller is used to move the leg to the correct angle.

When the leg is on the ground, we assume that the hip torque causes the body to rotate, but that the ground reaction forces keep the leg angle constant. Again, a simple servo controller was used to keep the body angle near the desired position. Finally, hopping height is controlled by injecting a fixed amount of energy during the thrust phase. The amount of energy injected is monotonically related to the resulting hopping height.

On the actual robot, these controllers were switched in and out using a simple state machine. It is possible to approximate Raibert's state-machine controller using an autonomous feedback policy $\mathbf{u} = \pi_{raibert}(\mathbf{x})$, as in [Buehler and Koditschek, 1988]. The robot is considered to be in the FLIGHT phase when the foot is above the ground ($y_{ft} > 0$), in the COMPRESSION phase when the foot is on the ground and the leg is compressing ($y_{ft} = 0$ and $\dot{r} < 0$), or in the THRUST phase any other time the leg is compressed ($r < r_{s0}$, where r_{s0} is the rest length of the spring in the leg). This allows us to write Raibert's control law in the following autonomous equations:

$$\tau_{hip}(\mathbf{x}) = \begin{cases} k_{fp}\left(\theta + \sin^{-}1(\frac{\dot{x}T_s}{2r} + \frac{k_{\dot{x}}(\dot{x}-\dot{x}_d)}{r})\right) + b_{fp}\dot{\theta} & \text{if FLIGHT} \\ k_{att}(\frac{\theta}{2} - \phi) - b_{att}\dot{\phi} & \text{if COMPRESSION or THRUST} \\ 0 & \text{otherwise.} \end{cases}$$

$$F_{leg}(\mathbf{x}) = \begin{cases} v_{retract} & \text{if FLIGHT} \\ v_{thrust} & \text{if THRUST} \\ 0 & \text{otherwise.} \end{cases}$$

$\dot{x}$ is the forward velocity of the center of mass of the body and $\dot{x}_d$ is the desired forward velocity. k_{fp},b_{fp}, and $k_{\dot{x}}$ are the gains of the foot placement controller, T_s is the duration of the last stance phase. Finally, v_{thrust} is a constant that is monotonically related to the hopping height. In my simulation, this controller works very nearly as well as the state machine controller.

As illustrated in Figure 6-2, when the robot is initialized with $|\phi| \gg 0$, these original control equations cannot prevent the robot from falling down. The large moment of inertia of the body serves to minimize the effect of the swing leg during the aerial phase of the steady state trajectory, but consequently complicates the problem of dealing with disturbances around the hip. The original control equations were based on an analysis of steady-state hopping conditions augmented with simple linear feedback, and they simply can't recover from these disturbances. [Vermeulen et al., 2000] reported improved robustness using a more comprehensive model of hip displacements, suggesting that a nonlinear controller may improve the situation.

6.3 Approximate Optimal Control

There is some discussion about "optimal" hopping in the literature, but always on simpler models of the robot dynamics. For instance, in [Seyfarth et al., 2002], they describe an optimal feed forward control law for hopping height in the spring-loaded inverted pendulum

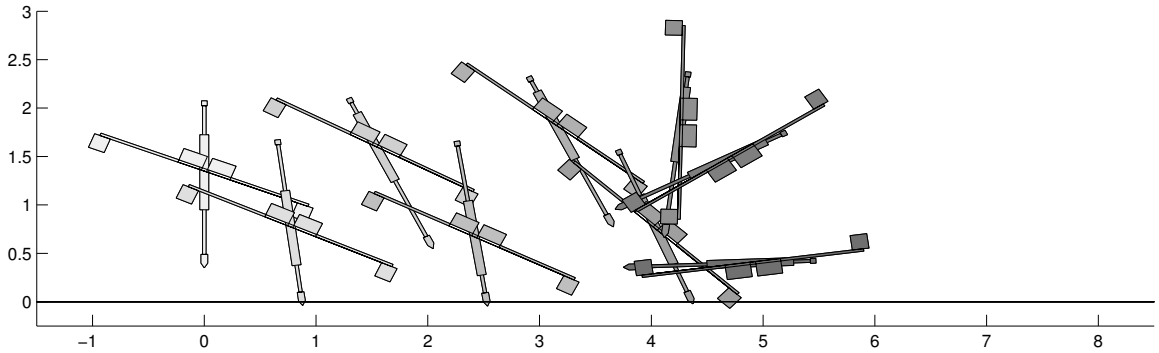

Figure 6-2: The original controller cannot recover from perturbations at the hip. The shading of the robot gets darker as time progresses.

model. Although Raibert never mentions optimization, I believe that his intuitive solution to the control problem can be interpreted as an approximation to an optimal controller.

The control objectives, as described by Raibert, are to regulate the hopping height (y), body attitude (ϕ), and forward speed ($\dot{x}$). The simplest cost function which quantifies Raibert's control objectives is

$$g(\mathbf{x}, \mathbf{u}) = \frac{1}{2}(y - y_d)^2 + \frac{1}{2}(\phi - \phi_d)^2 + \frac{1}{2}(\dot{x} - \dot{x}_d)^2,$$

where y_d is the desired (average) hopping height, ϕ_d is the desired body attitude, and $\dot{x}_d$ is the desired forward velocity.

The goal of the original control equations is to regulate these variables over time. In order to efficiently simulate many trials with the robot, we optimize the finite-horizon cost:

$$C = \int_0^T [\alpha g(\mathbf{x}, \mathbf{u}) - 1] dt,$$

where α is a constant scalar set to keep $\alpha g < 1$ and T is either the time that the robot falls down or 5 seconds, which ever comes first. By defining a finite-horizon time instead of an infinite-horizon discounted, we can evaluate the cost exactly in T seconds of simulation time. Cutting the simulations short when the robot falls over is another trick for dramatically reducing the computation time[1]. In order to ensure that simulations where the robot falls down are penalized more than those where the robot remains balanced, we optimize $\alpha g - 1$ instead of just g. 5 seconds is enough time for the hopper to land and hop at least 5 times.

Although the cost function is quadratic, the complicated equations of motion make it too difficult to solve the optimal control problem analytically. Instead, we will apply a policy gradient algorithm to improve the performance of the controller. We do this by augmenting the existing controller with a feedback controller stored in an artificial neural network parameterized by the vector $\mathbf{w}$:

$$\mathbf{u} = \pi_{raibert}(\mathbf{x}) + \pi_{\mathbf{w}}(\mathbf{x}).$$

[1]Not only do we simulate less time, but the equations of motion typically become stiff when modelling many successive collisions, forcing us to dramatically reduce the step-size. By avoiding these situations, we can keep the simulation time step at a reasonable number (typically 1e-4 seconds).

We use a small 3-layer neural network with 9 inputs, 25 hidden units, and 2 outputs. The inputs are $(y_{ft}, \theta, \phi, r, \dot{x}_{ft}, \dot{y}_{ft}, \dot{\theta}, \dot{\phi}, \dot{r})$, scaled and shifted linearly so that most inputs are within the range $[-1.0, 1.0]$. The outputs are (u_0, u_1), which are computed from the network using the standard sigmoid activation function and then linearly scaled and shifted from the range $[0, 1]$ to the desired actuator range. The parameter vector $\mathbf{w}$ contains the connection strengths between layers and the threshold biases for each unit, a total of 302 values.

This representation is very common for standard function approximation. In many of my simulations, I use a more local representation of the state space, as do [Miller et al., 1990, Schaal et al., 2001]. A global representation was chosen in this problem to deal with the large, continuous state space with a relatively small number of learning parameters, because the convergence rate of this policy gradient algorithm scales with the number of learning parameters.

6.4 Policy Gradient Optimization

The task is to find the control parameter vector $\mathbf{w}$ which minimizes the expected value of the finite-horizon cost $E\{C\}$ given some distribution of initial conditions. In order to quickly and efficiently run many simulations of the robot, I developed a simple distributed version of a policy gradient algorithm. In an idea that is in retrospect very similar to Pegasus [Ng and Jordan, 2000], my algorithm selected 50 to 100 random initial conditions, which I will call $\mathbf{X}_0$, from the distribution. It then did a policy evaluation sweep, simulating the robot with the same parameters from all of the different initial conditions in parallel on all of the lab's computers. The total value of the policy was estimated as the sum of the trials:

$$E\{C_{\mathbf{w}}\} \approx \frac{1}{N} \sum_{\mathbf{x}_0 \in \mathbf{X}_0} C_{\mathbf{w}}(\mathbf{x}_0).$$

In order to perform gradient descent on the total policy value, $E\{C_{\mathbf{w}}\}$, I implemented the Downhill Simplex method [Press et al., 1992, section 10.4]. This algorithm is just a fancy way of estimating a multi-dimensional gradient by sampling without having to resample every dimension on every iteration. $\mathbf{w}$ was always initialized to all zeros, so that they initial policy was simply Raibert's original controller.

Due to the large moment of inertia of the body relative to the hip, the robot is most sensitive to initial conditions where $|\phi| \gg 0$. For this reason, random initial conditions for ϕ were selected from a uniform distribution over $[-\pi, \pi]$ radians. Similarly, the internal angle between the body and the leg $(\phi - \theta)$ was selected from a uniform distribution over $[-\pi/2 + 0.25, \pi/2 - 0.25]$ radians. The remaining state variables were set to ensure that the center of mass of the robot was stationary at the point $x = 0$m, $y = 2.5$m. These random initial conditions correspond to the robot being dropped from 2.5m with arbitrary leg and body angles.

6.5 Results

After a very large number of simulations ($\approx 10^6$), the learning algorithm had converged on a surprising controller. Because our simulation did not model collisions between the body and the leg, the final controller recovered from large hip angles by spinning the body nearly 360 degrees around the hip to return to vertical (see Figure 6-3).

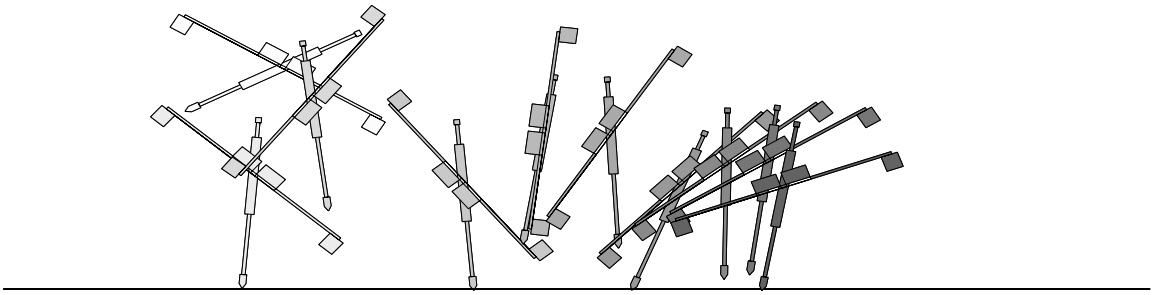

Figure 6-3: The controller augmented with a learning component is able to recover from an initial condition where both the leg angles are very far from vertical.

The goal of the optimization is to maximize the region of initial conditions from which the robot can recover. For comparison, Figure 6-4 plots the region of initial conditions for the three controllers. The initial conditions used for learning were taken as a uniform distribution from the range shown in this plot, also with some initial velocity.

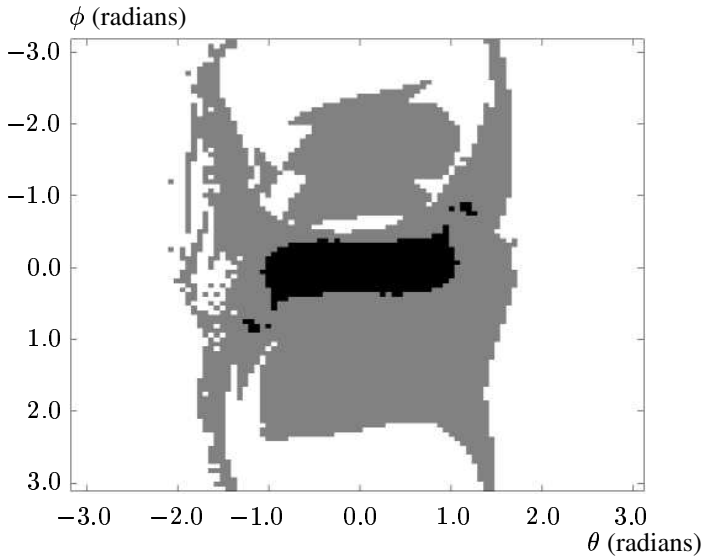

Figure 6-4: Regions of stability for the optimized controller, $\mathbf{u} = \pi_{raibert} + \pi_{\mathbf{w}}$, in gray, and for the original policy, $\mathbf{u} = \pi_{raibert}$, in black.

6.6 Discussion

Using brute force optimization techniques, we were able to train a very small neural network controller that notably expanded the region of stability of the hopping robot. This was accomplished by evaluating the robot from a relatively small sample (50-100) of random initial conditions pulled from a very broad distribution. Although we did not attempt to optimize any energetic criteria, nor to drive the robot along a specific trajectory, these could easily be taken into account by adding terms to the cost function. Furthermore, none of the algorithms presented here depend on the robot model and could be applied directly to a simulation of a different robot.

More importantly, this procedure was used to augment an existing controller. Because we were able to interpret the intentions of the original control equations in terms of optimal control, we were able to use a numerical algorithm in such a way that the performance could only improve. This particular policy gradient algorithm was optimized for efficient parallel computations, but the algorithm presented in the last chapter would have worked (more slowly).

Chapter 7

Conclusions and Future Work

Optimal control is a powerful framework for designing control systems for dynamically stable legged locomotion. It allows us to maximize the performance of our control algorithms during the design phase, and to derive rules for continually adapting the controller after the robot has left the laboratory. Finding truly optimal controllers for complicated legged systems is currently analytically and computationally intractable. In thesis, I have demonstrated that by combining existing ideas from legged robots with computational tools from the optimal control literature, we can derive controllers that outperform their non-learning counterparts for at least two of the simplest legged robots.

The Toddler robot was designed to be a minimal 3D bipedal walker. The robot is under-actuated, with only 4 actuators to control 9 degrees of freedom, making it difficult to analytically derive controllers with theoretical performance guarantees. The passive walking capabilities of the robot made it relatively easy to derive intuitive control strategies for stable flat-ground walking, but these controllers required many steps to return to the nominal trajectory after a small perturbation and could not recover from a large one. By applying an online actor-critic algorithm, I was able to quickly and reliably obtain a quantifiably more robust controller for walking on flat terrain. As the robot moves, the algorithm is constantly refining the policy allowing the robot to quickly adapt to small changes in the terrain.

Although it was accomplished on a very simple biped, this demonstration is important for a number of reasons. It suggests that passive walking principles used in the mechanical design of the robot can be used to improve the performance of our online learning algorithms by mechanically biasing the system toward a walking solution. It provides hope for policy gradient algorithms, which require us to perturb the policy parameters as the robot walks, by demonstrating that it was possible to perturb the parameters enough to estimate the performance gradient without visibly changing the walking gait. It also argues for the utility of value function methods, by demonstrating that even an extremely coarse estimate of the value function could be used to dramatically improve the performance of the learning system. Toddler also gives us some priceless hands-on experience with learning on a real walking robot.

7.1 Scaling Up

The major obstacle for applying these ideas to more complicated walkers is scaling the performance of the learning algorithm. As the number of degrees of freedom increases,

so will the number of trials necessary to acquire the controller. The algorithm correlates changes in the policy parameters with changes in the performance on the return map. As we add degrees of freedom, the assignment of credit to a particular actuator will become more difficult, requiring more learning trials to obtain a good estimate of the correlation.

The scaling of the reinforcement learning algorithms is an open and interesting research question and a primary focus of my current research. Most existing approaches to scaling in reinforcement learning involve hierarchical methods [Barto and Mahadevan, 2003]. I believe that there is also hope for improving the non-hierarchical methods. There is certainly room for improvement on the convergence and performance guarantees of the existing algorithms, and to come up with new algorithms. There is also a large body of domain specific knowledge in the robotics literature that can and should be exploited in our learning algorithms.

7.1.1 Controller Augmentation

One clear example of domain knowledge that can be exploited by a learning system is an existing control solution. If we already have a controller that is capable of producing stable periodic trajectories on the robot, then it would be naive to ask the learning system to start from a blank slate. The experiments presented in chapter 6 were the first test of these ideas, where we used a neural network to augment the existing control law for Marc Raibert's planar one-leg hopping robot. By re-examining Raibert's original three-part hopping controller in terms of approximate optimal control, we were able to train the neural network to systematically improve the controller. The augmented controller was able to recover from a much larger region of initial conditions than the original controller.

7.1.2 Toddler with Knees

Passive dynamic walking is another piece of domain specific knowledge that we have used in this thesis. To address scaling on the Toddler robot, we have built a new version of the robot with knees. The version of the robot described in chapter 5 has an efficient gait that is robust enough to accommodate small changes in terrain, but it cannot traverse obstacles. This limitation is primarily due to the mechanics of the robot, which can only achieve foot clearance by leaning sideways. The kneed version of the robot will have much more foot clearance and a much more anthropomorphic gait.

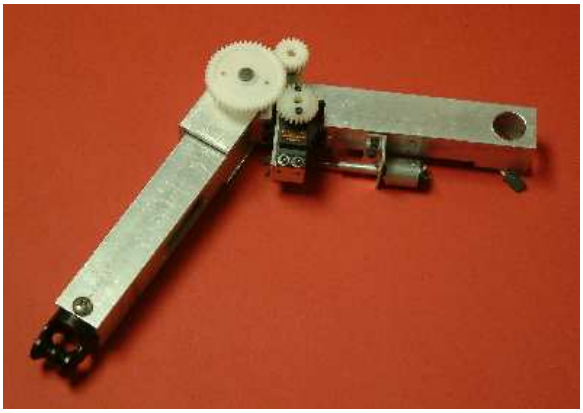

Figure 7-1: A prototype knee for the next version of the Toddler robot

In order to exploit the efficiency of the passive dynamics in the knees, we have developed

a simple clutch mechanism at the knee (see Figure 7-1) which will allow the knee actuator to apply torque when engaged and allow the knee to swing freely when disengaged. Figure 7-2 shows Toddler with the knees attached.

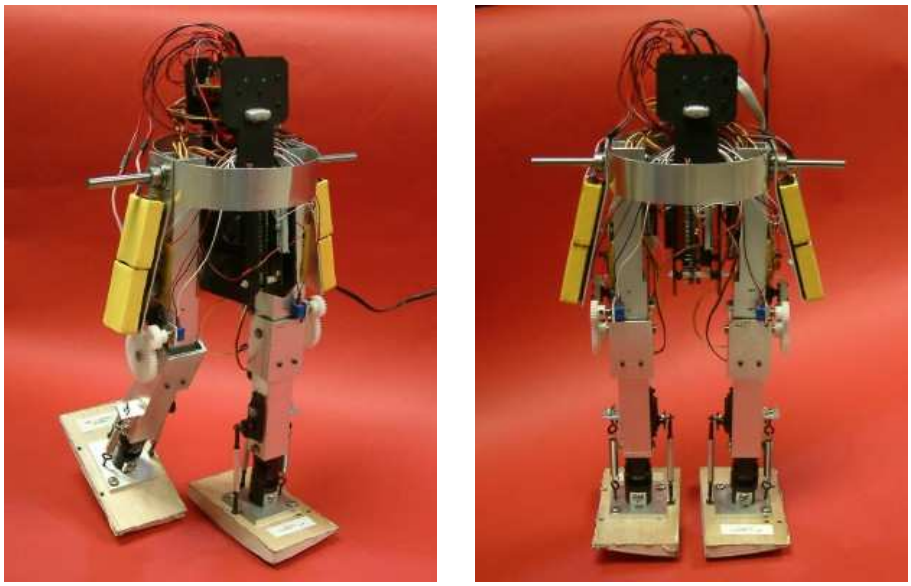

Figure 7-2: Toddler with knees

The new robot will also have foot contact sensors built into the ankles and actuators attached to the arms. The curvature of the feet is a very important component to both the stability and efficiency of the current robot. It is not trivial to build in a contact sensor in the foot without disturbing this geometry. Our solution is to add a small spring-loaded mechanism between the ankle and the foot that will close whenever the foot is on the ground. This sensor will improve our return map estimation as well as aid in the timing for the clutch mechanism on the kneed robot. The arm actuators will be added in series with the existing arm design, so that the basic mechanical coupling of each arm to the opposite leg will remain intact. When the motor is holding it's zero position, the arm will mimic the passive gait. When actuated, the arm will be provide active control over the yaw compensation, which will help to control the direction of the robot.

This version of the robot will not only walk better, but it will make a significantly more compelling argument that the learning ideas presented in this thesis can be applied to humanoid walking.

7.1.3 Future Applications

As we begin to address the scaling problem, we plan to apply these algorithms to more and more sophisticated robots, ultimately working up to a full scale humanoid robot like Honda's ASIMO [Honda Motor Co., 2004]. Figure 7-3 shows Pete Dilworth's new Protoceratops robot that is just beginning to walk using a hand-designed controller. This robot has 14 primary degrees of freedom (not including the animatronics in the head). We plan to test our controller augmentation ideas on this substantially more complicated robot in the coming months and hope to demonstrate online policy improvement.

The algorithms considered in this thesis could also be applied to physical simulations of walking characters for computer graphics. These simulations are a special case of the

Figure 7-3: Pete Dilworth's Protoceratops robot

robot problem where we have perfect knowledge of the robot dynamics and perfect sensors. Scaling is particularly important in this domain because animated characters typically have many more degrees of freedom than robots.

7.2 Scaling Down

The presentation of passive walking in chapter 4 provided a sufficient understanding of the passive dynamics for us to complete the mechanical design of our robot, but it also highlighted a number of basic results that are missing from the passive walking literature. It is very surprising how little we know about even the compass gait model. A stability analysis of the compass gait model that is as complete as the analysis of the rimless wheel would be an extremely important contribution to this literature. In the coming months, we also plan to use the compass gait as an even simpler system than Toddler for testing under-actuated learning and control strategies for walking.

In particular, we are currently working on an active control strategy using a novel model-based control design approach based on recent results on oscillator synchronization by [Slotine and Wang, 2003]. These results suggest that partial contraction analysis could be used as a tool to design the coupling terms between two oscillators, one being the physical robot and the other a simulated oscillation in the control program, in order to generate *provably* stable limit cycle behavior on a walking robot.

7.3 Closing Remarks

In the next few years, as robots become more robust and more efficient, we will see them leaving the laboratory environment and exploring our world. It is unlikely that these robots will be controlled by a learning system that acquired a controller from a blank slate, but online policy improvement will be a critical component for any of these real world systems. By combining learning techniques with existing robotic control solutions, we can produce robots that will work well initially and continue to improve with experience.

Bibliography

[Alexander, 1996] Alexander, R. M. (1996). *Optima for Animals*. Princeton University Press, revised edition.

[Anderson and Pandy, 2001] Anderson, F. C. and Pandy, M. G. (2001). Dynamic optimization of human walking. *Journal of Biomechanical Engineering (AMSE)*, 123(5):381–390.

[Barto and Mahadevan, 2003] Barto, A. G. and Mahadevan, S. (2003). Recent advances in hiearchical reinforcement learning. *Discrete Event Systems Journal*, 13:41–77.

[Baxter and Bartlett, 2001] Baxter, J. and Bartlett, P. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

[Bertsekas, 2000] Bertsekas, D. P. (2000). *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Optimization and Neural Computation Series. Athena Scientific.

[Boyan and Moore, 1995] Boyan, J. A. and Moore, R. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems (NIPS)*, 7.

[Buehler and Koditschek, 1988] Buehler, M. and Koditschek, D. E. (1988). Analysis of a simplified hopping robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 817–819.

[Camp, 1997] Camp, J. (1997). Powered "passive" dynamic walking. Master's thesis, Cornell University.

[Channon et al., 1990] Channon, P., Hopkins, S., and Pham, D. (1990). Simulation and optimisation of gait for a bipedal robot. *Mathematical and Computer Modeling*, 14:463–467.

[Chew, 2000] Chew, C.-M. (2000). *Dynamic Bipedal Walking Assisted by Learning*. PhD thesis, Massachusettes Institute of Technology.

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

[Coleman and Ruina, 1998] Coleman, M. J. and Ruina, A. (1998). An uncontrolled toy that can walk but cannot stand still. *Physical Review Letters*, 80(16):3658–3661.

[Collins et al., 2004] Collins, S., Ruina, A., Tedrake, R., and Wisse, M. (2004). Powered bipedal robots based on ramp-walking toys. *Submitted to Nature*.

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

[Doya, 1999] Doya, K. (1999). Reinforcement learning in continuous time and space. *Neural Computation*, 12:243–269.

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

[Goswami et al., 1996] Goswami, A., Thuilot, B., and Espiau, B. (1996). Compass-like biped robot part I : Stability and bifurcation of passive gaits. Technical Report RR-2996, INRIA.

[Hardt et al., 1999] Hardt, M., Kreutz-Delgado, K., and Helton, J. W. (1999). Modelling issues and optimal control of minimal energy biped walking. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*, pages 239–251. Professional Engineering Publishing Limited.

[Hirai et al., 1998] Hirai, K., Hirose, M., Haikawa, Y., and Takenaka, T. (1998). The development of honda humanoid robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1321–1326.

[Honda Motor Co., 2004] Honda Motor Co. (2004). http://world.honda.com/asimo/.

[Huber and Grupen, 1998] Huber, M. and Grupen, R. A. (1998). A control structure for learning locomotion gaits. In *7th International Symposium on Robotics and Applications*.

[Juang and Lin, 1996] Juang, J.-G. and Lin, C.-S. (1996). Gait synthesis of a biped robot using backpropagation through time algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1710–1715.

[Kajita et al., 1992] Kajita, S., Yamaura, T., and Kobayashi, A. (1992). Dynamic walking control of a biped robot along a potential energy conserving orbit. *IEEE Transactions on Robotics and Automation*, 8(4):431 – 438.

[Kato et al., 1983] Kato, T., Takanishi, A., Jishikawa, H., and Kato, I. (1983). The realization of the quasi-dynamic walking by the biped walking machine. In Morecki, A., Bianchi, G., and Kedzior, K., editors, *Fourth Symposium on Theory and Practice of Walking Robots*, pages 341–351. Warsaw: Polish Scientific Publishers.

[Kimura and Kobayashi, 1998] Kimura, H. and Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 278–286.

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

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

[Konda and Tsitsiklis, 1999] Konda, V. R. and Tsitsiklis, J. N. (1999). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014.

[Kun and Miller, 1996] Kun, A. L. and Miller, III, W. (1996). 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.

[Kuo, 1995] Kuo, A. D. (1995). An optimal control model for analyzing human postural balance. *IEEE Transactions on Biomedical Engineering*, 42(1):87–101.

[Kuo, 2002] Kuo, A. D. (2002). Energetics of actively powered locomotion using the simplest walking model. *Journal of Biomechanical Engineering*, 124:113–120.

[Larin, 1999] Larin, V. B. (1999). Control of a hopping machine. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*, pages 37–47. Professional Engineering Publishing Limited.

[Li et al., 2001] Li, C.-S. R., Padoa-Schioppa, C., and Bizzi, E. (2001). Neuronal correlates of motor performance and motor learning in the primary motor cortex of monkeys adapting to an external force field. *Neuron*, 30:593–607.

[Lohmiller and Slotine, 1998] Lohmiller, W. and Slotine, J. (1998). On contraction analysis for non-linear systems. *Automatica*, 34(6):683–696.

[Maier et al., 1999] Maier, K., Blickman, R., Glauche, V., and Beckstein, C. (1999). Control of legged planar hopping with radial basis function neural networks. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*, pages 133–141. Professional Engineering Publishing Limited.

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

[McMahon, 1984] McMahon, T. (1984). Mechanics of locomotion. *The International Journal of Robotics Research*, 3(2).

[Miller et al., 1990] Miller, III, W., Glanz, F., and Kraft III, L. (1990). CMAC: an associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567.

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

[Morimoto and Atkeson, 2002] Morimoto, J. and Atkeson, C. (2002). Minimax differential dynamic programming: An application to robust biped walking. *Advances in Neural Information Processing Systems*.

[Muybridge and Mozley, 1979] Muybridge, E. and Mozley, A. V. (1979). *Muybridge's Complete Human and Animal Locomotion: All 781 Plates from the 1887 Animal Locomotion*. 3 Vols. Dover Publications, Incorporated.

[Ng and Jordan, 2000] Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence*.

[Ng et al., 2003] Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, 16.

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

[Pearlmutter, 1995] Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228.

[Pratt, 2000] Pratt, J. (2000). *Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots*. PhD thesis, Computer Science Department, Massachusetts Institute of Technology.

[Pratt and Pratt, 1998] Pratt, J. and Pratt, G. (1998). Intuitive control of a planar bipedal walking robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

[Pratt and Pratt, 1999] Pratt, J. and Pratt, G. (1999). Exploiting natural dynamics in the control of a 3d bipedal walking simulation. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition.

[Raibert, 1986] Raibert, M. H. (1986). *Legged Robots That Balance*. The MIT Press.

[Schaal et al., 2001] Schaal, S., Atkeson, C., and Vijayakumar, S. (2001). Scalable techniques from nonparametric statistics for real-time robot learning. *Applied Intelligence*, 17(1):49–60.

[Schwind, 1998] Schwind, W. J. (1998). *Spring Loaded Inverted Pendulum Running: A Plant Model*. PhD thesis, University of Michigan.

[Seyfarth et al., 2002] Seyfarth, A., Geyer, H., Günther, M., and Blickhan, R. (2002). A movement criterion for running. *Journal of Biomechanics*, 35(5):649–655.

[Slotine and Li, 1990] Slotine, J.-J. E. and Li, W. (1990). *Applied Nonlinear Control*. Prentice Hall.

[Slotine and Wang, 2003] Slotine, J.-J. E. and Wang, W. (2003). A study of synchronization and group cooperation using partial contraction theory. Block Island Workshop on Cooperative Control, Kumar V. Editor, Springer-Verlag, 2003.

[Smith, 1998] Smith, R. (1998). *Intelligent Motion Control with an Artificial Cerebellum*. PhD thesis, University of Auckland, New Zealand.

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

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

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

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[Sutton et al., 1999] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems (NIPS)*.

[Taga, 1995] Taga, G. (1995). A model of the neuro-musculo-skeletal system for human locomotion. I. emergence of basic gait. *Biological Cybernetics*, 73(2):97–111.

[Tedrake and Seung, 2002] Tedrake, R. and Seung, H. S. (2002). Improved dynamic stability using reinforcement learning. In *5th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*. Professional Engineering Publishing Limited.

[Tedrake et al., 2004a] Tedrake, R., Zhang, T. W., Fong, M., and Seung, H. S. (2004a). Actuating a simple 3D passive dynamic walker. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

[Tedrake et al., 2004b] Tedrake, R., Zhang, T. W., and Seung, H. S. (2004b). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*.

[Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

[Todorov, 2002] Todorov, E. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5(11):1226.

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

[van der Linde, 1998] van der Linde, R. Q. (1998). Active leg compliance for passive walking. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2345.

[Vaughan et al., 2004] Vaughan, E., Di Paolo, E., and Harvey, I. (2004). The evolution of control and adaptation in a 3D powered passive dynamic walker. In *Proceedings of the International Conference on the Simulation and Synthesis of Living Systems (ALIFE)*. MIT Press.

[Vaughan, 2003] Vaughan, E. D. (2003). Evolution of 3-dimensional bipedal walking with hips and ankles. Master's thesis, Sussex University.

[Vermeulen et al., 2000] Vermeulen, J., Lefeber, D., and Man, H. D. (2000). A control strategy for a robot with one articulated leg hopping on irregular terrain. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*, pages 399–406. Professional Engineering Publishing.

[Westervelt and Grizzle, 2002] Westervelt, E. and Grizzle, J. (2002). Design of asymptotically stable walking for a 5-link planar biped walker via optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

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

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

[Wisse and van Frankenhuyzen, 2003] Wisse, M. and van Frankenhuyzen, J. (2003). Design and construction of Mike; a 2D autonomous biped based on passive dynamic walking. In *Proceedings of the International Symposium on Adaptive Motion of Animals and Machines*.

[Yamasaki et al., 2002] Yamasaki, F., Endo, K., Kitano, H., and Asada, M. (2002). Acquisition of humanoid walking motion using genetic algorithms: Considering characteristics of servo modules. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3123–3128.

[Yan and Agrawal, 2004] Yan, J. and Agrawal, S. K. (2004). Rimless wheel with radially expanding spokes: Dynamics, impact, and stable gait. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.