Proposal
Robust Optimal Walking on the Sarcos Humanoid Robot

Eric C. Whitman

April 2012

Robotics Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

Thesis Committee:
Christopher G. Atkeson, Chair
J. Andrew (Drew) Bagnell
Hartmut Geyer
Jerry Pratt

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thanks to the NSF for providing funding.

Keywords: Walking, Humanoid, Control

I am dedicated to completing my thesis.

Abstract

This thesis proposal focuses on using optimal control to generate a walking behavior for the Sarcos humanoid robot that is robust to both external disturbances and modeling error. We describe a walking controller that functions by coordinating multiple low-dimensional optimal controllers. We break a simplified model of the dynamics into several subsystems that have limited interaction. Each of the subsystems are augmented with coordination variables and we use a Dynamic Programming algorithm to generate optimal controllers for the augmented subsystems. We then use value functions to coordinate the augmented subsystems by managing tradeoffs of the coordination variables, producing an optimal controller for the simplified dynamic model. Inverse dynamics are then used to generate joint torques for the full rigid-body model of the robot. In a simulation based on the Sarcos robot, we demonstrate the robustness of this method to unexpected external disturbances such as pushes (both impulsive and continuous), trips, ground elevation changes, slopes, and regions where it is prohibited from stepping.

When implementing this controller on physical hardware (the Sarcos humanoid robot), we must also cope with significant modeling error. We produce stable walking by modifying our controller to include individual joint PD gains as well as modifying the swing leg subsystems to include acceleration as a state (and control jerk rather than acceleration). We also present two modifications to the Dynamic Programming algorithm, a multiple-model variant and a learning-based variant, that allow us to generate policies that are more tolerant of modeling error.

Acknowledgments

First and foremost, I would like to thank my advisor, Chris Atkeson, for all of his help and support throughout my time as a graduate student. Thanks are also due to the rest of my committee, Drew Bagnell, Hartmut Geyer, and Jerry Pratt for their advice and comments on this draft. I would also like to thank my entire lab group for their comments and feedback on all my research. Ben Stephens has been particularly helpful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Summary

This proposal is concerned with the problem of humanoid walking. The majority of the proposal focuses on work that has already been completed, and the final chapter briefly discusses what we intend to complete during the remainder of our thesis. Chapter 2 discusses the underlying Dynamic Programming algorithm that we use throughout the remainder of the document. We also consider two proposed improvements aimed at improving the robustness to modeling error: a multiple-model approach and combining the Dynamic Programming optimization with learning of the model. In Chapter 3, we propose a novel method for coordinating multiple controllers, which we apply to simulated humanoid walking. We also present results quantifying the robustness of our simulated walking to various types of perturbations. In Chapter 4, we discuss modifying the simulation controller to function on real hardware and present preliminary results for walking in place.

1.2 Motivation

Humanoid walking robots (as opposed to wheeled robots) have been promoted for several applications including rough terrain locomotion, working in environments built for humans, and working with people. To be effective in any of these roles, the system needs to robust to unexpected disturbances of many kinds. They must be capable of walking in unknown, unstructured environments. The robot should also be compliant, both for safety when working around humans and for stability in the face of an imperfectly sensed environment. It should also be able to place its feet in specific, desired locations. We aim to create a walking system that is both reactive, able to react instantaneously to disturbances or changing conditions, and deliberative, able follow a plan, avoid obstacles, and place its feet as necessary.

1.3 Types of Walking Robots

The robot's hardware typically dictates much about the style of controller that is used for walking. At one end of the spectrum, are passive dynamic walkers, which are able to walk with no actuation entirely due to the passive dynamics of the mechanical system [62] [20]. Some energy is inevitably dissipated in the impact at foot touch down, so unactuated robots can only walk down a slope, where gravity replaces the lost energy. There are also powered robots based on passive walkers [19], which use actuation to replace the lost energy and can walk on a flat surface. They typically use simple, possibly open-loop, controllers. The design goal for these controllers (and much of the mechanical design) is generally to increase the (typically very limited) stability, which is often expressed in terms of gait sensitivity [35]. For these robots, the controller has little or no control over the internal joint configuration of the robot, which is entirely determined by the dynamics.

At the other end of the spectrum are non-backdrivable robots. Most electric motors produce power at high RPM and low torque, which requires a high gear ratio (often achieved by harmonic

drives [57]) to produce the torques and speeds necessary for humanoid walking, especially in human-sized humanoids. Gear ratios amplify the motor inertia by the square of the gear ratio, which can easily dominate the dynamics for large gear ratios. Large gear ratio gear boxes can easily become non-backdrivable. Some of the most successful humanoid robots fall roughly into this category including Honda's ASIMO [34], HUBO [72], and the HRP series [48] [3]. These robots typically use high bandwidth feedback control to precisely track desired positions and velocities at individual joints. With precise control of the internal joint configuration, the primary dynamic concern is tipping over. Accordingly, many of the control strategies used focus on regulation of the Zero Moment Point (ZMP) [46] [94], often by pre-planning trajectories and following them precisely [38].

Somewhere in between these two extremes are compliant force (or torque) controlled robots. For such robots, the actuators produce a force that interacts with the rigid body dynamics rather than completely dominating it. This generally makes it more difficult to track precise desired positions, but compliance can be useful when interacting with an unknown environment. For robots with electric motors, compliance can be achieved either by use of series-elastic actuators [76] or by using backdrivable gearboxes. It is easier to make gearboxes backdrivable for low gear ratios, which necessitates either low torque requirements or special high-torque motors. The Sarcos Primus robot, which we use, achieves compliance by force control of hydraulic actuators. Hydraulic actuators do not require gearing to achieve the necessary joint torques, making it easier to achieve high performance force control. Unlike a physical spring, however, force control does have a bandwidth limit, and it may not be compliant on the time scales involved in an impact.

1.4 Simple Models

Humanoid robots often have a large number of joints, meaning that they have a high dimensional state space. High dimensional state spaces are difficult to deal with directly because of the "Curse of Dimensionality" [13]. Instead, many researchers choose to control a simplified

model that captures some aspect of the dynamics of the full system. This is related to the concept of “templates” and ”anchors” [30] introduced for studying biological systems. The simple models are easier to control and allow for greater physical intuition. Additionally, certain simple models are well studied and provide a convenient common ground between disparate hardware platforms. The Linear Inverted Pendulum Model (LIPM) [46] and compass gait model [33] [39] are commonly used simple models of walking that consider the motion of the center of mass.

Control of the simplified model only produces control of the aspects of the full system represented by the full model. This includes center of mass motion and footstep placement for the LIPM model. Inverse kinematics can then be used to find joint angles for the full model of the robot [94]. Joint torques, if needed, can be produced by computed torque methods.

1.5 Intended Thesis Contributions

The primary focus of my thesis will be on a novel optimal control method for humanoid walking and applying that method to the Sarcos humanoid robot as well as supporting technologies. The core of our approach is the coordination scheme built around the principle of **Instantaneously Coupled Systems**.

Our controller produces **flexible and robust walking in simulation**. It can start, stop, turn, and walk at controllable speed. It is robust to trips, low friction, slopes, abrupt changes in ground height, and pushes.

We have also produced two new versions of dynamic programming, a **multiple model variant and a learning variant**, that improve robustness to modeling error. For both variants, we have shown good results on the simple test problem of pendulum swing-up. We intend to use one of these versions to improve the robustness of our walking controller on hardware.

The end goal of the thesis will be to produce **stable walking on the Sarcos humanoid robot**. We have already applied a simplified version of the controller to the robot to achieve walking in place.

Chapter 2

Dynamic Programming

2.1 Introduction

Trajectory optimization [64] [41] can be used to generate a walking pattern offline. Feedback gains are then typically used to stabilize the system around the trajectroy. Such a controller is only optimal on the optimized trajectory and often simple feedback is only effective for a small region of state space near the nominal trajectory. If a disturbance or modeling error results in the system straying significantly from the nominal trajectory, a new trajectory can be generated starting from the current state [70]. Model Predictive Control and Receding Horizon Control continuously generate new trajectories from the current state regardless of whether the system is tracking accurately or not [102] [24]. The problem with these methods is that they require optimizing a new trajectory online. Optimizing a walking trajectory is generally computationally expensive, meaning that it can take a long time, resulting in low bandwidth control and a delayed response to perturbations. These methods often must employ approximations (such as linearity) to keep computation time short. Additionally, they may not be able to find globally optimal, or even acceptable, trajectories.

Another possibility is to prepare for for perturbations offline by generating a policy that is effective for a large region of state space. These methods generally suffer from the "Curse of

Dimensionality" [13] because high dimensional state spaces are very large and therefore difficult to fill. Trajectory libraries [58] [103] [28] are a common approach in which a large number of different trajectories are generated. A policy can then be implemented by choosing the action from the nearest trajectory or by interpolating betwen them [6]. It is sometimes possible to analytically compute the convergence region for a given trajectory [96], making it possible to know where more trajectories need to be added to the library and which regions are already covered.

Dynamic Programming (DP) [13] [17] [86] generates optimal controllers for a large region of state space. DP is a class of algorithms for solving Markov Decision Processes [14] [78]. It is based on the key observation that segments of an optimal trajectory are themselves optimal, as formalized by the Bellman equation [13],

$$V(\mathbf{x}) = \min_{\mathbf{u}} L(\mathbf{x}, \mathbf{u}) + \gamma V(f(\mathbf{x}, \mathbf{u})), \tag{2.1}$$

where $\mathbf{x}$ is the state, $\mathbf{u}$ is the control action, $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$ is the discrete time system dynamics, γ is a discounting factor slightly less than 1, and V is the value function (total cost to go). This observation makes it possible to reuse computation to efficiently solve for the value function, $V(\mathbf{x})$, and a deterministic policy, $\mathbf{u} = \pi(\mathbf{x})$ everywhere. Typically, a solution is achieved iteratively by using (2.1) to update the value and the $\mathbf{u}$ that achieved the minimum to update the policy,

$$\pi(\mathbf{x}) = \arg\min_{\mathbf{u}} L(\mathbf{x}, \mathbf{u}) + \gamma V(f(\mathbf{x}, \mathbf{u})). \tag{2.2}$$

There are many variants of DP, often based on how value and policy updates are ordered. In value iteration [13], the action is not stored during computation, but only computed as part of the minimization in (2.1). In policy iteration [37], a sweep of updating the policy at every state according to (2.2) is done once followed by multiple sweeps of updating the value with (2.1) until the value converges. This whole process is then repeated. In modified policy iteration [79], which we use a form of, the value update sweeps are not continued until convergence before doing another round of policy update.

If we have a continuous state space, we must approximate this process because it is impossible to store the policy and value function at all states. The most straightforward solution is to represent the continuous state space as a grid of discrete states. The grid can be regularly spaced (as we use in this thesis), or it can have adaptive spacing [67]. Alternately, the states can be randomly sampled [82] [8]. Additionally, $V(f(\mathbf{x}, \mathbf{u}))$ will not generally be available if the value function is only defined at some set of discrete points in a continuous space. It must therefore be estimated by interpolation or a local model of the value function. Local models can be Taylor series like, such as those produced by Differential Dynamic Programming [41] or fit to a set of state, value pairs [98].

Continuous action spaces pose a similar problem. Again, the most straightforward appraoch is to break the continuous space up into a grid of discrete points. However, for actions, the approximation can be avoided completely (at significant computational cost) by performing the minimization in (2.2) with a local optimizer. Random sampling of actions is an alternative for both discrete actions spaces [55] and continuous action spaces [5].

DP policies are globally optimal (up to the grid resolution), avoiding potential problems with local minima. They can handle both discrete decisions such as where to place a footstep as well as continuous decisions such as how much torque to apply. Additionally, DP is useful for optimizing transient responses to perturbations as well as optimizing a steady state periodic gait. In [61], DP on the Poincaré state was used to determine stride-level variables for walking. DP was used for continuous control of some joints in [99] and of all joints in [91].

DP also provides value functions, which give a measure of the cost-to-go from anywhere in the state space. Value functions are useful when coordinating multiple controllers (Section 3.2.2 for theory and Section 3.4.2 for practice). From just a policy, it is difficult to determine whether a decision is critically important or your optimal controller made it arbitrarily. Value functions provide an automatic way to determine which of multiple competing controller should have precedence; value is a natural currency for managing tradeoffs between controllers in an

optimal control setting.

2.2 Dynamic Programming Algorithm

We use a version of modified policy iteration (based on [5]) that generates time-invariant policies for systems with continuous state and action spaces. We represent the continuous state space by dividing it into a grid. Each grid point stores the current value estimate, $V(\mathbf{x})$, and the current best control vector, $\mathbf{u} = \pi(\mathbf{x})$ ($\mathbf{u} = \tau$ for the pendulum). Because we are interested in time-invariant policies, we do not store a value function for each time step, backing up from the final time. Instead, we store a single value function and continue to refine it until it converges. In practice, we generally stop policy generation once there are few changes per iteration. We do, however buffer our value function by always using the previous iteration's value when interpolating the terminal value and not updating the value function until the end of the entire iteration. Buffering makes it so that the order in which we update the points does not matter.

During an iteration, for each grid point, we use the Bellman equation to update the value function. Because we have a continuous action space, we cannot enumerate all possible actions. We have found that trying only a single random action and comparing it to the current best action [5] is a good tradeoff between finding the best action for the current estimate of our value function and completing a large number of iterations so that our value function accurately reflects our policy. Our Bellman value update therefore looks like

$$V(\mathbf{x}) = \min_{\mathbf{u} \in \{\mathbf{u}_0, \mathbf{u}_r\}} L(\mathbf{x}, \mathbf{u}) + \gamma V(f(\mathbf{x}, \mathbf{u})), \tag{2.3}$$

where $\mathbf{u}_0$ is the current best action, $\mathbf{u}_r$ is a random action drawn from the legal range. The policy is then updated to the corresponding action. Note that this is an extreme form of modified policy iteration that behaves much like value iteration. Convergence of DP with random actions is discussed in [5].

If the final state, $f(\mathbf{x}, \mathbf{u})$, is outside of the grid, we assign an infinite cost, $V(f(\mathbf{x}, \mathbf{u})) = \infty$.

Otherwise, we use multilinear interpolation [21] on the previous iteration's estimate of $V(\mathbf{x})$ to estimate the terminal value. Multilinear interpolation on a grid has the useful property that the value is continuous, even at grid cell boundaries. Simulating forward multiple time steps before evaluating the value function at the result can help minimize the effects of the approximation introduced by interpolating [54]. To ensure that we get a good estimate of the value function, if the final state is in a cell with the originating grid point as a corner, we simulate forward additional time steps (with the same $\mathbf{u}$, adding the appropriate one step cost, $L(\mathbf{x}_k, \mathbf{u})$, until we enter a cell not bordering the originating grid point. Additionally, if we enter a cell where some corners have infinite value and some corners have finite value, we continue to simulate until we reach a cell where either all corners have infinite value or all corners have finite value.

We can run this algorithm on multiple separate grids, even grids with different dimensionality. For example, the sagittal and coronal stance policies discussed in Section 3.4.1 have three grids each, a five-dimensional grid for double support, a four-dimensional grid for single support, and a two-dimensional grid for handling the transition between double support to single support and the decision of where to step. Multiple grids can also be useful in situations where variable grid resolution is required in various parts of the state space. This usually occurs because the second derivatives of the value function are very large in some region (requiring a finer grid), but not the whole space. Rather than wasting computation power by having a fine grid everywhere, we can have a coarse grid for most of the state space, and a fine grid just for the region that needs it. We do this, for example in the Swing-X and Swing-Y policy discussed in Section 3.4.1. For more extreme situations, such as near constraints, the necessary grid spacing for an accurate representation of the value function may be so fine that it is best to abandon the grid altogether and use an analytical value function (and policy) in this region.

The computational and memory costs of DP are linear in the number of grid points, and therefore exponential in the number of state space dimensions. This limits DP to problems with low dimensional state spaces (about 5 dimensions on a modern desktop computer), though in

some cases it is possible to treat higher dimensional problems by breaking them into multiple lower dimensional problems and coordinating the policies, as we will discuss in Section 3.2.

2.3 Robust Dynamic Programming

2.3.1 Related Work

Model based optimal control is a powerful tool that suffers from a major difficulty: it requires a model of the system. Developing such a model can be a difficult task. Even if you can find a good model structure, parameter measurement and sensor calibration can be inaccurate. Even if initially accurate, the model may change over time due to wear, temperature, weather, contact condition, or any number of unforseen influences. Additionally, you may wish to develop a single controller for a large number of instances of a manufactured product which have variations between them due to manufacturing inconsistencies or unequal change over time. In all of these cases, the controller must cope with a model that is not identical to the true system.

Optimal control methods can often be very sensitive to even small modeling errors. Optimizers are very good at finding and exploiting any possible advantage, even if that advantage is a modeling error. Optimal paths often lie along constraints, so even small errors can make them infeasible [4]. Methods that plan a long way into the future may rely on the first portion of their plan working precisely in order for the latter portion to perform well.

Most methods that aim to reduce the sensitivity of optimal control algorithms to modeling error fall into one of two general categories: (i) minimizing an expected cost over possible outcomes or (ii) minimax formulations that minimize the maximum cost (worst case) of possible outcomes.

For expected cost approaches, a distribution of possible outcomes is required. It can be either a discrete list of possibilities (with associated probabilities) or a continuous distribution. The distribution can take many forms, ranging from additive output noise to sets of models. Some

examples include additive or multiplicative state or control process noise on top of the nominal dynamics [97] [50], random variables in the process model [87], arbitrary state-dependent stochastic transition functions [104], and Markov decision processes (MDP) with known or even unknown transition probabilities [69].

Minimax methods are concerned with minimizing the worst case cost, so the probabilities of the potential outcomes are irrelevant and unnecessary. Again, there exists a large variety of ways to describe the set of possible outcomes. An H_∞ controller for linear systems can be found by solving a Riccati equation [26]. These controllers are robust to bounded input and process noise. Some examples for nonlinear systems include additive disturbances [65], discrete sets of models [9], and continuous sets of models [25]. Minimax formulations are generally able to provide stronger theoretical guarantees of robustness because they deal with worst case scenarios.

Most robust control algorithms are modifications of existing algorithms for deterministic problems. Differential Dynamic Programming (DDP) [65] [97] [50] and Model Predictive Control (MPC) [15] are popular techniques based on optimizing trajectories. Dynamic Programming (DP) is a class of algorithms that rely on the observation that any portion of an optimal trajectory is itself optimal. This leads to a situation where efficient computation can be achieved by reusing the solution to overlapping subproblems [13] [17] [86]. Most versions of DP produce control policies that are valid for large regions of the state space.

A popular approach to robust control law design is to optimize a policy by evaluating its performance in simulation on a distribution of possible models [9, 10, 12, 63, 71, 75, 85, 92, 100]. In this paper, we present Multiple Model Dynamic Programming (MMDP), which attempts to make a baseline implementation of DP more robust to modeling error by taking the expectation over a discrete set of multiple models. Importantly, we are not taking the expectation over a different model being randomly selected at each timestep; instead, we take the expectation over a single model being randomly selected once and used for all time.

2.3.2 Pendulum Swing-Up

We will demonstrate our algorithm on the problem of inverted pendulum swing-up. The controller must apply torques to a rigid pendulum in order to raise it to the inverted position and maintain it there. This is a good test problem because it involves both a dynamic travel component (getting to the inverted position) and a regulation component (remaining at the inverted position once there). The inverted pendulum is also one of the simplest nonlinear systems and a system about which we have good physical intuition, which makes it easier to interpret our results. The policy can also be severely limited by action-space constraints with it still remaining possible to achieve the goal from anywhere in the state space.

The pendulum is a second order system with one degree of freedom, so it has a 2-dimensional state space, $\mathbf{x} = \{\theta, \dot{\theta}\}$, where θ is the pendulum angle ($\theta = 0$ defined as upright), and the dot indicates a derivative with respect to time. It has a one-dimensional action space, $\mathbf{u} = \{\tau\}$, where τ is the control torque. The pendulum dynamics are given by

$$\ddot{\theta} = \frac{mLg\sin(\theta) + \tau}{mL^2}, \tag{2.4}$$

where m is the mass, L is the length, and $g = 9.81\mathrm{m/s^2}$ is the acceleration due to gravity. For the nominal system (we will perturb it later) the mass is 1 kg and the length is 1 m. We also constrain the control torque to $|\tau| \leq 1.5\mathrm{Nm}$, which requires the system to swing back and forth multiple times before reaching the goal.

The swing-up task is formally defined as minimizing the total cost, C, given by the integral,

$$C = \int_0^\infty L(\mathbf{x}, \mathbf{u})\, dt, \tag{2.5}$$

of the one step cost function,

$$L(\mathbf{x}, \mathbf{u}) = \theta^2 + 0.5\dot{\theta}^2 + \tau^2. \tag{2.6}$$

Since angles are topologically circular, we can represent the entire θ direction with a finite grid. However, we must bound the $\dot{\theta}$ dimension to the somewhat arbitrarily selected range of

$\pm 10\mathrm{rad/s}$. A grid resolution of at least about $\{50, 100\}$ is necessary to achieve swing-up. All experiments in this paper were performed with a resolution of $\{400, 900\}$ to give a good tradeoff between computation time (5 minutes on a PC using a 6-core processor with a clock speed of 3.33 GHz) and policy quality. This gives a resolution of $\{0.016$ radians, 0.022 radians/second$\}$ and a grid of 360,000 discretized states. It takes about 500 iterations to generate a controller that can achieve swing-up. We run all of our experiments for 5000 iterations to allow the policy and value function to converge very closely to the optimum.

Note that our DP algorithm requires the discrete time form of the pendulum dynamics, which we obtain by integrating (2.4):

$$\begin{bmatrix} \theta_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \theta_k + \dot{\theta}_k T + 0.5\ddot{\theta}_k T^2 \\ \dot{\theta}_k + \ddot{\theta}_k T \end{bmatrix} \tag{2.7}$$

where $T = 0.001\mathrm{seconds}$ is the time step, and $\ddot{\theta}$ is determined by (2.4).

2.3.3 Tests of Robustness

The goal of this paper is to produce steady state controllers, $\mathbf{u} = \pi(\mathbf{x})$, that are as robust as possible to modeling error. By this, we mean that we wish the controllers to be effective on a set of modified pendulum systems. We measure the effectiveness of a particular policy, $\pi(\mathbf{x})$, on a particular model by starting the system at $\mathbf{x} = \{\pi, 0\}$ (down), simulating it forward, and measuring (2.5). If the system does not reach the goal ($\mathbf{x} = \{0, 0\}$) and remain there within 60 seconds of simulation, we assume it never will and assign an infinite cost.

In order to test the robustness of our policies, we will modify the nominal model in four ways:

1. Varying the mass, m.
2. Varying the length, L.

3. Adding varying amounts of viscosity, ν. Equation (2.4) becomes

$$\ddot{\theta} = \frac{mLg\sin(\theta) + \tau - \nu\dot{\theta}}{mL^2}. \tag{2.8}$$

4. Misaligning the goal with gravitational up, representing an off-center mass or sensor miscalibration. An offset, θ_0 is added to (2.4), changing it to

$$\ddot{\theta} = \frac{mLg\sin(\theta - \theta_0) + \tau}{mL^2}. \tag{2.9}$$

For the final test, there is no state-action pair that results in both no acceleration and no cost, L, so the cost given by (2.5) will always be infinite. To accomodate this, we consider the task complete and stop integrating (2.5) after the system has remained near the goal for 15 seconds.

2.3.4 Modeling Error as Noise

Much work has been done on generating optimal controllers for stochastic systems by optimizing the expected value of the cost function. Modeling error is not actually stochastic, but these techniques can be used to increase robustness to modeling error by generally increasing robustness to unexpected dynamics.

For DP, we do not have an analytical expression for the value function, so we take the expectation of the stochastic dynamics by sampling from it, turning (2.3) into

$$V(\mathbf{x}) = \min_{\mathbf{u}\in\{\mathbf{u}_0,\mathbf{u}_r\}} \sum_{i=1}^{N} (L(\mathbf{x}, g_i(\mathbf{u})) + V(f_i(\mathbf{x}, g_i(\mathbf{u}))))p_i \tag{2.10}$$

$$\sum_{i=1}^{N} p_i = 1 \tag{2.11}$$

where g_i is an instantiation of the control noise, f_i is an instantiation of the process noise, and p_i is the probability of the ith sample. To avoid optimizing over the instantiations of the random noise, we must fix the g_i's, f_i's, and p_i's for all iterations, which essentially forces us to approximate the noise distribution as a sum of delta functions. In general, this can require a large number of samples (and correspondingly large computational cost) to adequately model the distribution.

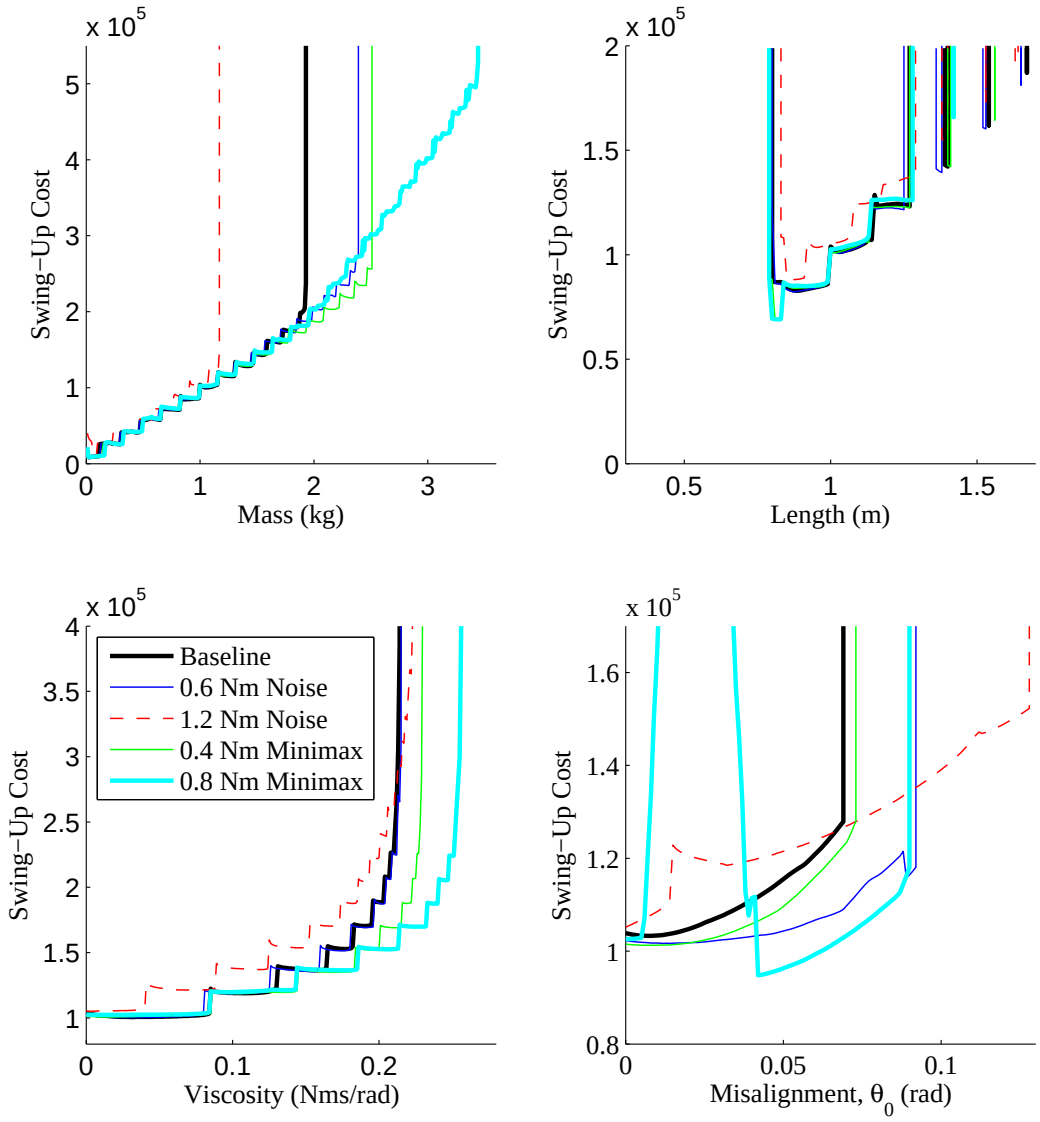

Figure 2.1: Robustness to model variations of mass, length, viscosity, and gravity misalignment. Results are shown for the baseline DP (Section 2.2), DP with small and large amounts of noise (Section 2.3.4), and DP with small and large minimax perturbations (Section 2.3.5).

For the case of additive noise, however, the Central Limit Theorem tells us that regardless of what distribution we use, after a large number of time steps, the total noise will have an approximately Gaussian distribution. This frees us to use a small number of samples and rely upon

summation over many time steps to produce a fuller distribution. For our swing-up example, we used no process noise ($f_i = f$), and additive control noise ($p_1 = p_2 = 0.5$, $g_1(\tau) = \tau - \Delta$ and $g_2(\tau) = \tau + \Delta$ where Δ is a parameter that controls the size of the noise). Note that additive control noise is identical to additive acceleration noise because $\partial\ddot{\theta}/\partial\tau = 1/mL^2$, which is constant.

Fig. 2.1 shows robustness results for stochastic DP compared to the baseline DP implementation. Discontinuities in the cost result from changes in strategy: using more or fewer back and forth swings to reach the upright position. In general, robustness to modeling error increases as the noise level increases. For all four tests, the large noise results in a significantly increased cost of swing-up. This is a general problem for methods aimed at improving robustness; the cost increases when minimizing it is no longer the sole goal of optimization.

2.3.5 Minimax Formulation

The unexpected dynamics introduced by modeling error are poorly represented by random noise because they are not independent at every time step. The errors introduced by modeling error often consistently push the system in the same direction, a situation which can get ignored by noise-based formulations as being extremely low probability. Another possibility for handling noise is to use a minimax formulation. Rather than minimizing the expected value, minimax algorithms attempt to minimize the maximum value or worst case scenario. In [65], a minimax version of DDP is developed and demonstrated on a walking robot.

We implement minimax DP much like stochastic DP, but instead of taking the expected value, we take the maximum, so the Bellman equation becomes

$$V(\mathbf{x}) = \min_{\mathbf{u}\in\{\mathbf{u}_0,\mathbf{u}_r\}} \max_i L(\mathbf{x}, g_i(\mathbf{u})) + V(f_i(\mathbf{x}, g_i(\mathbf{u}))). \tag{2.12}$$

We no longer require the Central Limit Theorem to approximate the full distribution. Instead, we rely upon the assumption that the worst case disturbance will be an extreme disturbance: either the maximum push in one direction or the maximum push in the other direction. This

assumption is an approximation for nonlinear systems, but in the case of short time steps, additive disturbances, and affine in controls, it is very nearly accurate. This again allows us to keep the computation managable by using only two samples. For the pendulum example, we again use no process noise ($f_i = f$), and additive control noise ($g_1(\tau) = \tau - \Delta$ and $g_2(\tau) = \tau + \Delta$).

Fig. 2.1 also shows results for the minimax version of DP. Both methods generally increase robustness to modeling error with increased magnitude of the disturbance until they reach a point of diminishing returns. This point can be quite abrupt, and it can be different for different types of error. For each method, we show results for two different disturbance sizes, one slightly before the point of diminishing returns on any of the tests, and another where performance has decreased on some tests but not on others. As expected, the minimax formulation generally outperforms the noise formulation, but it does well on different tests. Qualitatively, these trends can be summarized by noting that the minimax formulation tends to do better when pushing harder is necessary to overcome the unexpected modeling error (increased mass and viscosity), but the noise formulation deals better with small errors at inopportune times (misaligning the goal with gravitational up). Discontinuities in the cost result from changes in strategy: using more or fewer back and forth swings to reach the upright position.

2.3.6 Multiple Model Dynamic Programming

If the true system is deterministic, but different from your model, neither random nor worst-case disturbances are a good description of the true error. We are interested in the situation where there exists one true, deterministic model, but we do not know what it is. We approach this by generating a single policy, $\mathbf{u} = \pi(\mathbf{x})$, that is optimized to perform well on a set of N candidate models. We denote the dynamics of the ith model as $\mathbf{x}_{k+1} = f_i(\mathbf{x}_k, \mathbf{u}_k)$. This method only directly optimizes performance for the specific models it is given, but we have a reasonable expectation of good performance for a range of model space around each of the candidate models. If we know something about the type of modeling error we expect, we can

choose candidate models that cover the range of expected models. Formally, we wish to find the policy, π, that minimizes the total cost criterion,

$$C = \sum_{i=1}^{N} \sum_{x_0} \sum_{t=0}^{\infty} L(\mathbf{x}_t, \pi(\mathbf{x}_t)). \tag{2.13}$$

The innermost sum is trajectory cost and is the discrete form of (2.5). The middle sum is over trajectories started at each grid point. The outermost sum is over all N models.

For a single model, we have the convenient property that a single policy is optimal for all start states. Unfortunately, this property does not hold for the multiple model case. To show this, consider a policy that is optimal for a single start state. If we have a single start state, a state-indexed policy can control each of the models independently because they will travel through different states. We can therefore construct an optimal policy for multiple models with a single start state by copying the optimal policy along the optimal trajectory from each of the individual models' optimal policies. The optimal action at a given state will therefore depend on which model passes through it (and which corresponding optimal policy we must copy from). Since which model passes through a given point depends upon the start state, the optimal policy must also depend upon the start state.

We modify our DP algorithm to approximately minimize (2.13) by maintaining individual value functions, $V_i(\mathbf{x})$, for each of the N dynamic models (but only a single policy). To update a grid point, we pick the action by adding the value from each of the models:

$$\pi(\mathbf{x}) = \arg \min_{\mathbf{u} \in \{\mathbf{u}_0, \mathbf{u}_r\}} L(\mathbf{x}, \mathbf{u}) + \sum_{i=1}^{N} p_i V_i(f_i(\mathbf{x}, \mathbf{u})), \tag{2.14}$$

where p_i is again the probability of each model. We then update each value function individually according to

$$V_i(\mathbf{x}) = L(\mathbf{x}, \pi(\mathbf{x})) + V_i(f_i(\mathbf{x}, \pi(\mathbf{x}))). \tag{2.15}$$

As in the baseline implementation, we do many iterations of updating every point in the grid this way. This gives us a policy, $\pi(\mathbf{x})$, that is approximately optimized to perform well on all of the models, and a value function, $V_i(\mathbf{x})$, for each of the models given this policy.

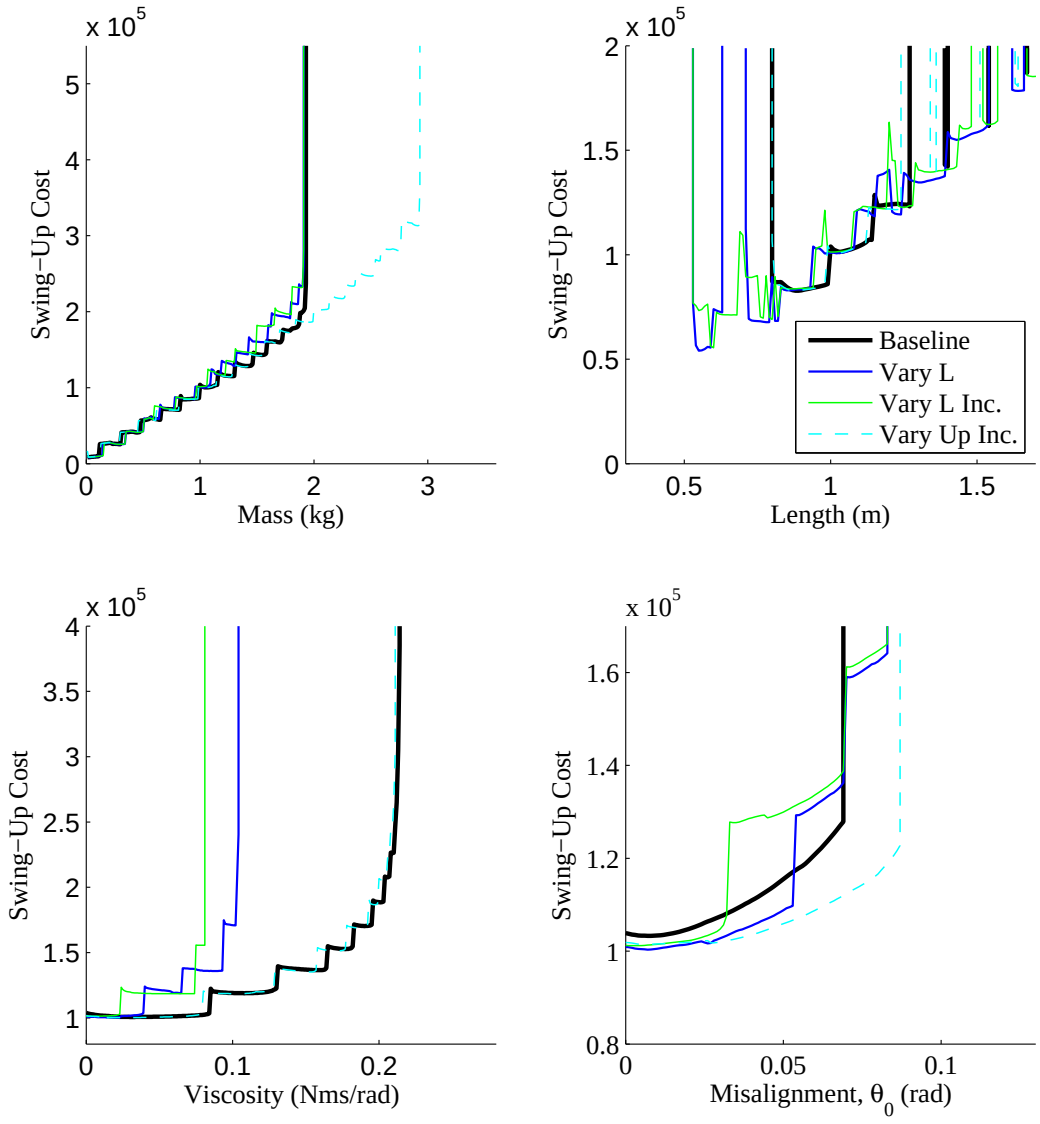

Figure 2.2: Robustness to model variations of mass, length, viscosity, and gravity misalignment. Results are shown for the baseline DP, MMDP with models of various length added all at once and added incrementally, and MMDP with models of various θ_0 added incrementally.

To analyze our update rule with respect to (2.13), we note that the value function, V, is

exactly equal to the innermost sum, so we can rewrite (2.13) as

$$C = \sum_{i=1}^{N} \sum_{x_0} p_i V_i(x_0). \tag{2.16}$$

We define $d_i(\mathbf{x})$ as the number of states for which if you start a trajectory there it will pass through state $\mathbf{x}$ for model i (and a given policy). If we make a single change to our policy at state $\mathbf{x}$, it will change the value at that state as well as at any state whose trajectory passes through it. All other values will remain the same. We can write the resulting change in total cost as

$$\Delta C = \sum_{i=1}^{N} p_i \left[\Delta V_i(\mathbf{x}) + d_i(\mathbf{x}) \Delta V_i(\mathbf{x})\right] \tag{2.17}$$

where ΔV_i is the new value minus the old value for model i. To ensure that this is decreasing, we must assume that

$$d_i(\mathbf{x}) = d_j(\mathbf{x}) \forall i, j, \mathbf{x} \tag{2.18}$$

which means that all models have an equal chance (given a random start state) to get into any given state. If this were not true, it would make sense to focus on minimizing the value for the models that are more likely to end up in that state, which our algorithm does not do. This assumption will not generally be true, but for many problems where the candidate models are similar, it will be approximately true.

Given (2.18), (2.17) simplifies to

$$\Delta C = (1 + d(\mathbf{x})) \sum_{i=1}^{N} p_i \Delta V_i(\mathbf{x}). \tag{2.19}$$

Given our update rule, we know that $\sum_{i=1}^{N} p_i \Delta V_i(\mathbf{x}) \leq 0$, so we have $\Delta C \leq 0$.

In practice, we generally have a nominal (or best guess) dynamic system and wish to increase the range of model space surrounding it for which the policy performs well. Either because the convergence rate is very slow or because of nonidealities such as (2.18) not holding, MMDP sometimes finds solutions that work well for the extreme cases of model, but not for the nominal case. A less drastic problem is gaps in the region of model space for which the policy succeeds.

To cope with this, we add models incrementally. This technique is similar to shaping, which was first invented in the field of psychology for training animals [74], but has since been used for machine learning and optimization [53]. Shaping is used to train animals to do complex tasks by first training them to do something simple, then training them to do successively closer approximations of the desired behavior. We start off by running DP on just the nominal model. We then add dynamic models that are very similar to the nominal case, and run a new optimization starting with the previously solved for policy and the previously solved for value function. We can then continue to add additional models progressively further from the nominal dynamics and rerunning the optimization. When new models are added, the value function can be initialized by copying the value function of the most similar model already in the training set.

Fig. 2.2 shows results for the MMDP. We took 21 pendulum models ($p_i = 1/N$) with lengths, L, ranging from 0.5 m to 1.5 m in 0.05 m increments and ran MMDP on them. The result was a drastically improved robustness to variation in the length. Swing-up was achievable for nearly the entire 0.5 m to 1.5 m range, but with a gap centered around 0.65 m.

We can compare this to the result when we add the same 21 models incrementally. We start with the nominal model ($L = 1.0$ m). After doing 5000 iterations of DP, we add the two adjacent models ($L = 0.95$ m and $L = 1.05$ m), initializing their value functions from the nominal models and then do another 5000 iterations of DP. We then add another 9 pairs of models, initializing each from the previous pair, and doing 5000 iterations of DP between each addition. By shaping the policy in this way and gradually building its robustness, we are able to cover the same range of modeling error, but without the gap centered at $L = 0.65$ m.

We also use the same method to target improving robustness to the gravitational angle by incrementally adding pairs of models with positive and negative θ_0's further and further from 0. The result is an improved robustness to varying this angle while maintaining a low cost. On the other hand, optimizing on this set of models does not improve performance with respect to changing the pendulum length. This illustrates a downside of MMDP: it often does not provide

improved robustness for model variations other than the type specifically targeted.

2.3.7 Discussion of MMDP

Because (2.18) will generally not hold, in practice MMDP is a heuristic method. The more qualitatively similar the models, the closer (2.18) will be to holding, and the closer MMDP will get to truly minimizing (2.13).

MMDP works well when you know something about the type of modeling error you have. It works best when you are able to provide it with models that span the range of possible systems. Attempts to generally increase robustness by providing models with multiple types of changes performed less well. This was at least partially due to the fact that in the higher dimensional model space, our sampling was much more sparse.

Another limitation of the MMDP approach is that it is computationally expensive to handle types of error that increase the dimensionality of the state space (e.g. time delays, filters, structural flexibility) even if the policy does not depend on these extra dimensions. MMDP must include these extra states in the grid and therefore pay the exponential computational and memory cost of the extra states. Methods that do not explicitly model the modeling error (such as the noise and minimax formulations) need not pay this cost. Methods that simulate entire trajectories (instead of single time steps as in DP) can avoid it as well because they need only fully explore the space that the controller depends on.

For a known, deterministic system, the optimal controller for any cost function where failure has an infinite cost will have the same basin of attraction, which is identical to the maximum feasible region. However, this says nothing about what happens when the system is different from what was expected. Robustness to modeling error can be very sensitive to the choice of cost function. Experiments support our intuition that in MMDP, the robustness to the type of model variation is relatively insensitive to choice of cost function because robustness is being directly optimized for. However, for types of modeling error other than what we were directly

optimizing for, we see just as much sensitivity to cost function choice as we see in the stochastic and minimax formulations.

The pendulum swing-up problem is an easy task, so even the baseline control algorithm achieves a moderate level of robustness to modeling error. In this case, the robust algorithm provides extreme robustness, which may seem unnecessary or excessive. In some cases, such as contact (e.g. shaking hands) or walking in sand or surf at the beach, this extreme level of robustness may be necessary. Additionally, for harder problems, the baseline algorithm may provide minimal robustness to modeling error, so the additional robustness provided by the robust versions of the algorithm will be necessary for controlling a real system.

As mentioned at the end of Section 2.3.4, there is a performance cost to achieving robustness. The optimal deterministic controller achieves the lowest possible cost if the system does follow the expected deterministic dynamics. Any other controller will therefore result in a higher cost, so if the robust algorithm generates a more robust controller, it must necessarily be different, and must necessarily have a higher cost when applied to the nominal plant. A downside of MMDP is that it does not necessarily provide benefits for model variations other than the type specifically targeted. For example, while the range of gravity misalignment angles that can be handled increases, the amount of viscosity that can be tolerated actually decreases.

2.4 Dynamic Programming with Learning

2.4.1 Introduction

There is a limit to the region of model space that can be stabilized by robust methods, and our robust method works best if we can parameterize the type of error we have. In most cases, including the one we are interested in, we only actually want to get good performance on a single dynamic system, we just do not know what it is. Finding policies for unknown systems is well studied in the field of Reinforcement Learning [42] [101]. Model Free Reinforcement Learning

attempts to find a policy directly without building a predictive model of the system [47] [11].

Model Based Reinforcement Learning methods, which first learn a model, then use the model to generate a policy are often more data efficient than model free methods [22]. Some version of Dynamic Programming is a popular choice for generating a policy from the model [78]. Methods that attempt to learn the parameters of a parametric model require the least data, but can perform poorly if the true model is not in the class of models considered. Our initial model is already obtained by fitting the parameters of a parametric model, so we are mostly interested in error that can not be adequately captured with a parametric model. Accordingly, we use a generic non-parametric function approximator to model our system. There are many possible non-parametric function approximators including [73], locally weighted learning [83], and neural networks [27]. We use Gaussian Processes are a stochastic function approximation method popular in Reinforcement Learning [80] [52] [23].

2.4.2 Method

We are interested in generating successful policies for low-dimensional, deterministic dynamic systems. In many cases, we have a model, but it is insufficiently accurate to generate a successful policy. We use Gaussian Processes (GP) to learn the model and dynamic programming to generate a policy. The main idea is to alternate between trials on the real system to collect more data and running dynamic programming to improve the policy. In practice, we often have a model of the system that is not accurate enough for for successful policy generation, which we wish to improve. We therefore write generic dynamics as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) + \Delta(\mathbf{x}, \mathbf{u}) \tag{2.20}$$

where f is our best guess at the dynamics and Δ represents the unknown dynamics. Rather than attempting to learn the model from scratch, we learn only the residuals, Δ. Starting from our best guess at the dynamics (as opposed to learning from scratch) should help us to learn more quickly by making it easier to get data in the regions of state space that we care about. Additionally, we

hope that it will capture some features of the dynamics and that Δ will be a simpler function to learn than the full dynamics.

In the specific case of dynamic systems of 2nd or higher order, we often have one state space dimension that is defined as a derivative of another. For these situations, we can assume some components of $\dot{\mathbf{x}}$ are known a priori and need not be learned. We can therefore eliminate them from the output of Δ (though not the input). In the pendulum example described in Section 2.3.2, errors in the dynamic model will affect the acceleration (derivative of velocity), but the derivative of the position will be the velocity (by definition) regardless of what happens with the dynamics. For a second order dynamic system, we could write the dynamics as

$$\ddot{\theta} = f(\theta, \dot{\theta}, \mathbf{u}) + \Delta(\theta, \dot{\theta}, \mathbf{u}), \tag{2.21}$$

where θ is a vector of generalized positions. This halves the output dimension of Δ, which reduces the computation, memory, and data requirements.

We start off by running the basic DP algorithm described in Section 2.2 using only f as the dynamics. Since our dynamics are only a guess, there is no need to run the algorithm all the way to convergence; we can stop once we stop seeing large changes. Once we have generated a policy, we run the system using that policy and collect data.

Next, we use that data to attempt to learn Δ. Any function approximation method will work, but we have found Gaussian Processes to be particularly convenient [81]. There are high quality off-the-shelf tools available for Gaussian processes and we can automatically select meta-parameters through evidence maximization [81]. Additionally, their ability to handle stochastic systems naturally gives us a lot of flexibility for more complex uses in future work.

We need to use the dynamics repeatedly as part of our iterative DP algorithm, and computing the function estimate for a GP with a lot of data can be computationally expensive. To speed this up, we represent Δ as a grid, which we only need to populate a single time. We use the same resolution and bounds for the state space dimensions of Δ as we used in our DP grid (where possible given memory usage and computation time). Use of this grid would limit us to low dimensional

state spaces because of the Curse of Dimensionality, but we already have a similar limit from our DP algorithm, so it adds no additional limitations. The Δ grid is higher dimensional than the policy and value grids used by DP because it must include the action dimensions as well, but we can often use lower grid resolutions for the Δ grid. The computation cost of populating the grid is small compared to the DP sweeps because it only needs to be populated once and DP requires thousands of sweeps.

We then iterate running DP, testing the system, and relearning a model with the new data. We run DP using f plus Δ as the dynamics. In each iteration, we use all of our data to learn, downsampling as necessary to keep the computation reasonable. Our data is generated by actually running the system, so we get data points strung out in trajectories. Adjacent points are mostly redundant, so significant downsampling has little effect on the learning. This procedure of iteratively improving the policy to get data closer to the test distribution is reasonably common [7] [1].

We leave a theoretical analysis of this algorithm to future work.

2.4.3 Results

We tested this algorithm on the pendulum swing up problem described in Section 2.3.2. We added viscosity to the dynamics, which are now given by (2.8), so that the dynamics would have a dependence on velocity as well as position. We use the parameters g = 9.81, m = 1.0, L = 1.0, and ν = 0.1. We start out with an extremely inaccurate model of the system: $\ddot{\theta} = 10\tau$, and only require 3 trials starting from the "down" position to achieve successful swing-up of the pendulum. On the first trial, it has a terrible model, so it just wiggles around near the down position. This gives it enough information to figure out how to pump energy into the system and the second trial gets most of the way to the goal. This gives it enough information to learn the model near the goal, and the third trial succeeds. Figure 2.3 shows the position of the three trials and Figure 2.4 shows the torque for the three trials. Figure 2.5 shows the policies generated at

each iteration.

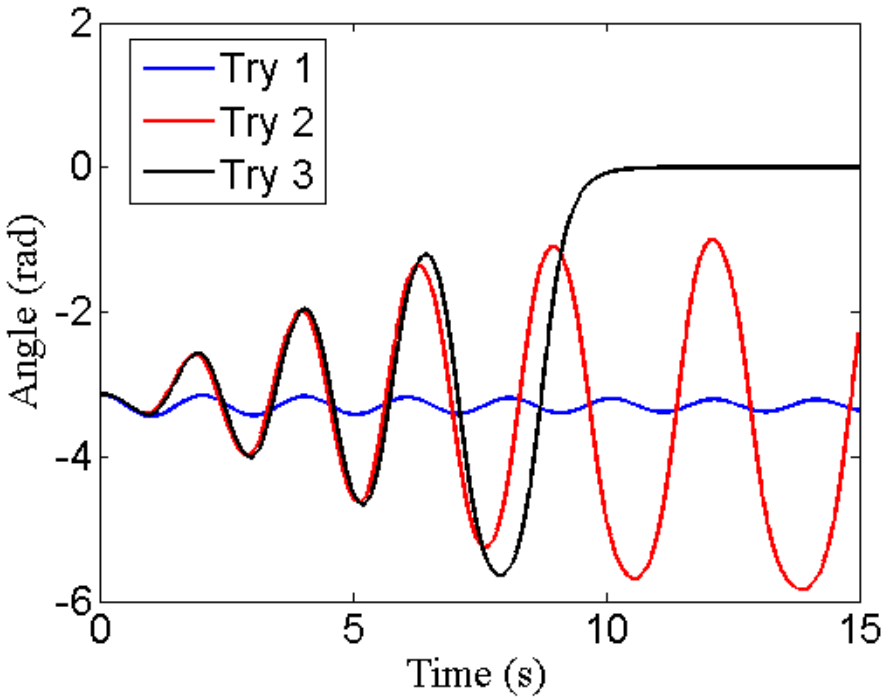

Figure 2.3: Plots of the position of the three trials used to learn the pendulum model well enough to achieve swing-up.

These results are encouraging because it took very few trials to achieve swing-up even without a good initial guess at the model. We expect that this algorithm will work well on any low-dimensional, deterministic system where we have access to the full state even if we know nothing about the dynamics to begin with.

2.4.4 Use of Uncertainty

If we ignore uncertainty, our method will suffer from model bias [84]; our DP solver will assume that the model is accurate when it is not. To prevent this, it is beneficial to use a probabilistic dynamics model [22]. In addition to providing a most likely regression of the learned function, they also produce a measure of uncertainty in the form of a standard deviation. There are several

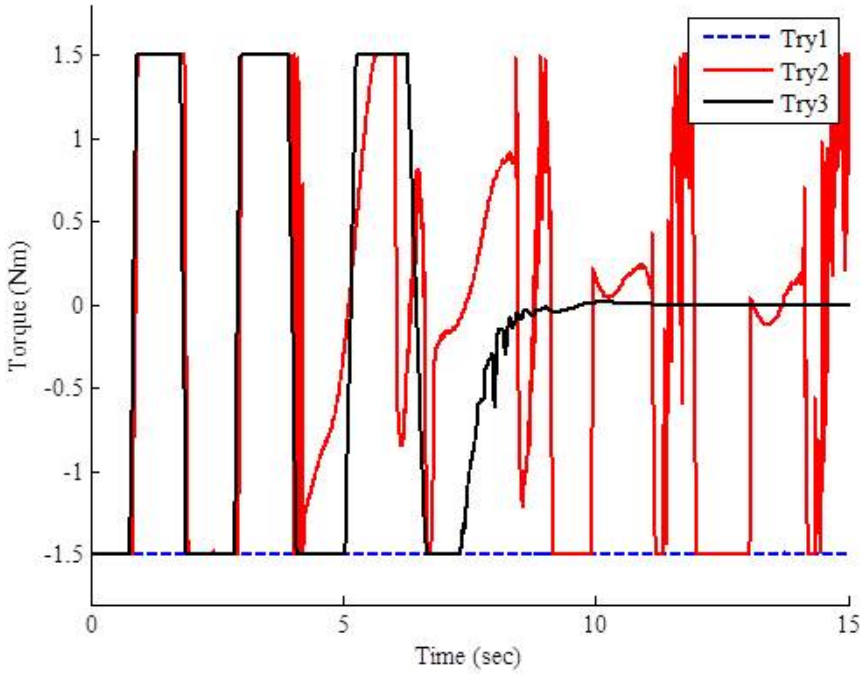

Figure 2.4: Plots of the torque for the three trials used to learn the pendulum model well enough to achieve swing-up.

ways to use this uncertainty that might be useful in various situations. The GP algorithm finds two different types of uncertainty. It assumes that the data is produced by some underlying function plus some additive noise. The size of this noise forms one type of uncertainty, and the uncertainty about the underlying function is the second. In various situations, it will make sense to use one or both types of uncertainty measurements.

The simplest way to make use of the uncertainty is to use either the expected value method described in Section 2.3.4 or the minimax method described in Section 2.3.5. For the minimax method, we can set the range as $\mu - a\sigma \leq \ddot{\theta} \leq \mu + a\sigma\}$ where μ is the most likely regression for δ and σ is the standard deviation. The constant a is a parameter that determines how conservative to be. For the expected value method, we can use this same range as our 2^n samples where n is the dimensionality of $\ddot{\theta}$. If we use only the additive noise, either of these methods might improve robustness in situations where the dynamics are not truly deterministic. If we also include

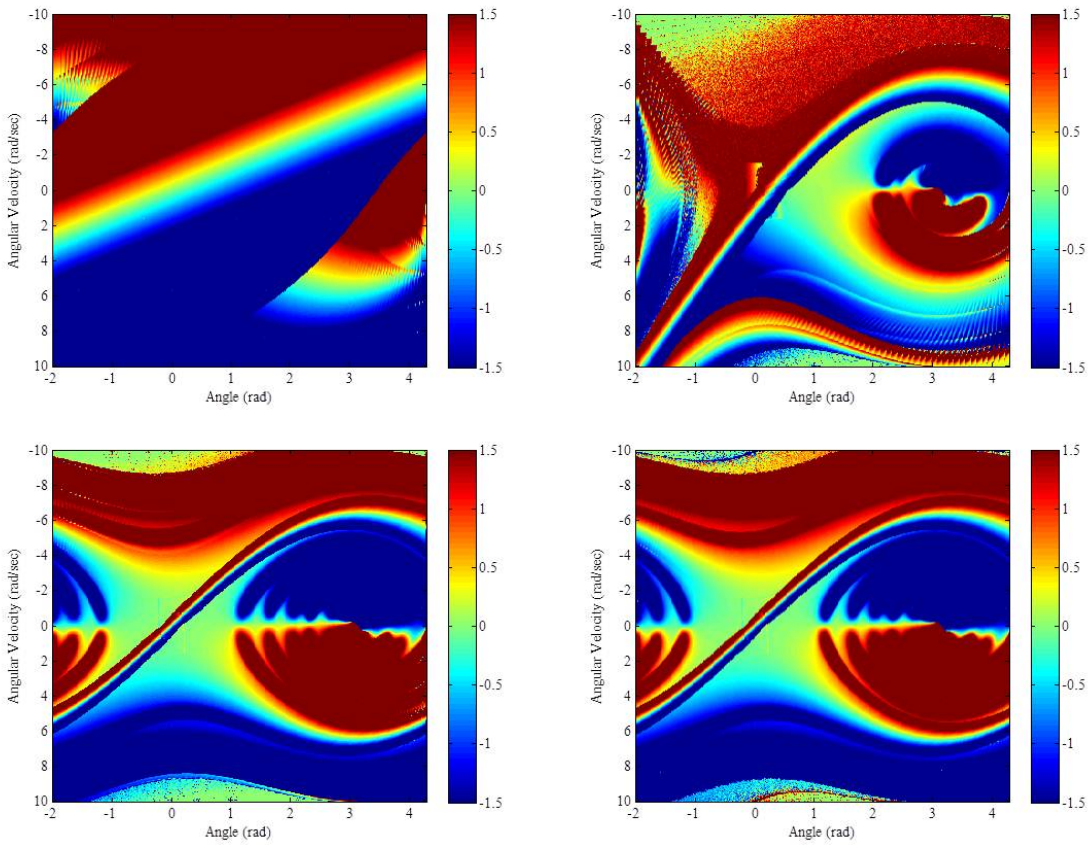

Figure 2.5: The policy generated for the first trial using the initial guess of a model (upper left), the policy generated after one trial of data collection (upper right), and the policy that achieved swing-up after two trials of data collection (lower left). For comparision, we also show the policy generated directly from the true dynamics (lower right).

the uncertainty about the underlying function, these methods will tend to decrease exploration and move the system to areas already explored. It may also achieve success faster by avoiding excessive exploration of unnecessary regions and handling the existing uncertainty better.

We are primarily interested in situations where we start out with a model that is already good enough to collect data near the region the optimal trajectory with pass through. For such problems, explicit exploration may be unnecessary and we may not need to address the well known Exploration vs. Exploitation tradeoff [93]. However, if we wish to extend this algorithm to a wider class of problems, we may need some mechanism to encourage exploration. The

baseline version of this algorithm can get stuck repeatedly going to the goal in an expensive manner even if a cheaper way exists; it can completely stop exploring once it finds the goal and never find the better path. One possible approach is to utilize the principle of "optimism in the face of uncertainty" as in R-MAX [18] and LSPI-R-MAX [57]. We would implement this as the opposite of the minimax formulation, a minimin approach. Rather than taking the worst case of the possible range, we take the best case. This only makes sense to do for the uncertainty about the underlying function. The effect of using uncertainty in this way would be to encourage exploration. The algorithm will tend to explore regions that would plausibly be good (where "plausible" is defined by the size of a) without wasting resources exploring regions that can not plausibly be useful. This may be important if we care about the quality of the solution rather than only success versus failure.

It may also make sense to combine the two approaches. We could use the minimin method for the uncertainty about the underlying function and the minimax (or expected value) method for the additive noise. However, combining the two approaches in this way relies on the ability of GP to distinguish between the two types of error, which it may not be able to do reliably.

Chapter 3

Simulated Walking

3.1 Introduction

A humanoid robot should be able to operate in the presence of large disturbances. We propose a method of control for bipedal walking that is capable of responding immediately to unexpected disturbances by modifying center of mass (CoM) motion, footstep location, and footstep timing. We use dynamic programming (DP) to design a nonlinear optimal controller for a simple model of a biped. DP suffers from the "Curse of Dimensionality", with storage and computation costs proportional to R^d, where R is the grid resolution and d is the dimension of the state. However, breaking the control design problem into parts greatly reduces the storage and computation costs. For example:

$$R^{d/2} + R^{d/2} << R^d. \tag{3.1}$$

By breaking the model into multiple subsystems of lower dimensionality, we are able to work with a higher-dimensional model than would otherwise be computationally feasible. To capture the coupling between the subsystems while keeping them low-dimensional, we augment the subsystems with additional coordination variables. We use dynamic programming to produce optimal policies and value functions for each of the augmented subsystems. Then, by using the value functions to manage tradeoffs between the coordination variables, we coordinate the sub-

system controllers such that the combined controller is optimal. Finally, we use the output of this high-level controller (CoM and swing foot accelerations) as the input to a low-level controller, which provides the joint torques necessary to produce those accelerations.

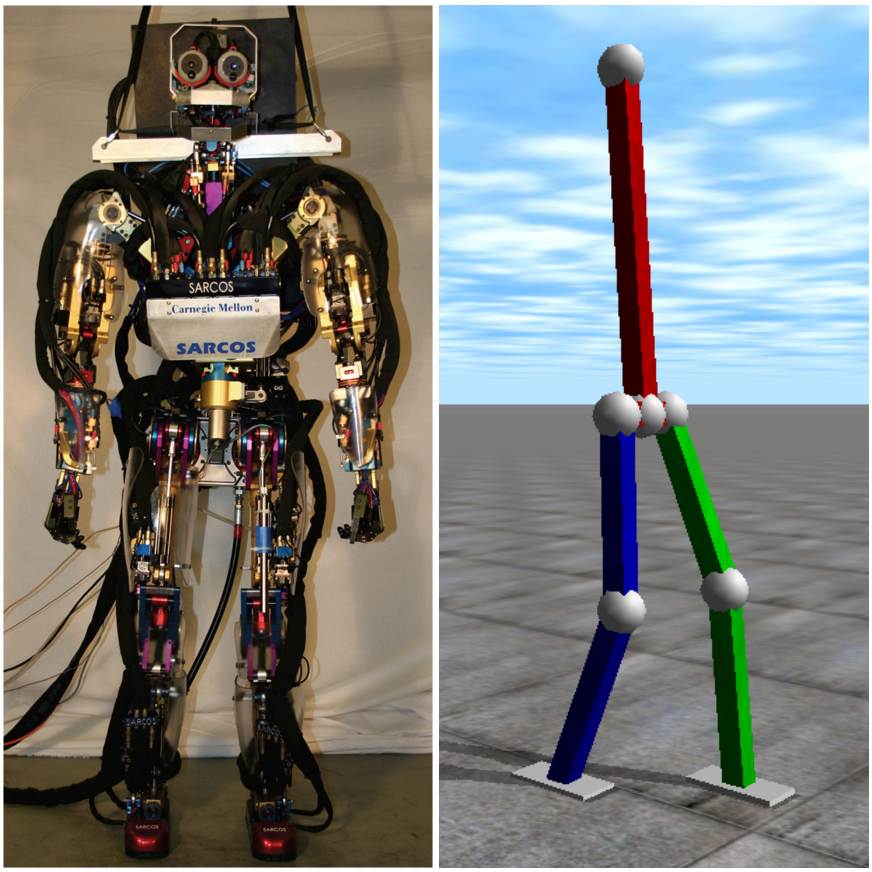

Figure 3.1: The Sarcos Primus hydraulic humanoid robot (left) and the simulation based on it (right).

3.1.1 Related Work

The central problem faced by walking controllers is managing reaction forces, which are constrained by friction and the requirement that the center of pressure (CoP) be within the convex hull of the region of support. Many walking controllers focus on CoM motion. A standard method of control is to first generate a CoM trajectory and then track that trajectory with inverse

kinematics [94]. Preview control of the CoP can be used to generate CoM trajectories [46]. By modifying the inverse kinematics for force control, it is possible to deal with small disturbances [29].

Unfortunately, even when tracking an optimal trajectory, the resulting controller is only optimal when near the desired trajectory, which is not the case following a significant unexpected disturbance. Due to constraints on reaction forces, linear independent joint controllers often can stabilize only a small region of state space. It is possible to frequently recalculate the CoM trajectory, taking into account the current robot state [70]. Model Predictive Control (MPC) and Receding Horizon Control (RHC) offer methods of generating trajectories online that continuously start from the current robot state [102].

For the system to recover from large disturbances, it is necessary to modify the reaction force constraints by adjusting the footstep placement or timing. One possible approach to this is trajectory libraries, where multiple trajectories are generated in advance and an appropriate one is used depending on the current robot state. Examples of trajectory libraries are given for standing balance in [58] and for walking in [103]. It is also possible to modify MPC so that it determines foot placement online [24]. In [66], the footstep timing is modified online in response to manually changed footstep locations.

Because of many walkings systems' high-dimensionality, which makes control difficult, it is common to model parts of a walking system as decoupled so that the lower-dimensional subsystems can be controlled separately [99], [105]. PD servos on individual joints is a very basic form of such decoupling. Unless coordination is handled carefully, the combined controller will be sub-optimal because the subsystem controllers lack the information necessary to make optimal decisions. We present a method of coordination that produces an optimal combined controller.

Angular Momentum

Many methods that control the LIPM, such as preview control[44] and some forms of model predictive control[102], focus on control of the zero moment point (ZMP). For many of these methods, it is either impossible or computationally expensive to extend them to work with a model that considers angular momentum or upper body rotation.

Due to disturbances and un-modeled dynamics, angular momentum and posture regulation are required, which can directly interfere with a controller based solely on the LIPM. Several authors have derived models of angular momentum for biped robots[44][32] and it has been shown that exploiting angular momentum can add significant stability to the system[77][88]. The subject of upper body angular momentum coordination and control for locomotion in position controlled humanoid robots has been considered[95][49].

Full body torque controllers based on force-based objectives such as desired COM acceleration and change of angular momentum have been presented[2][60][36][56]. Controllers such as these have achieved hip-strategy-like behaviors by making the angular momentum or posture objective less important than COM regulation. However, angular momentum and posture objectives have been mostly limited to regulation tasks.

In Section 3.2, we propose the concept of Instantaneously Coupled Systems (ICS) and demonstrate that our method for coordinating multiple optimal subsystem controllers is equivalent to an optimal controller for the full system. We then model walking as an ICS in Section 3.3 and describe our walking controller in Section 3.4. We discuss the capabilities of the controller in Section 3.5 and its robustness in Section 3.6. In Section 3.7 we propose a method for controlling torso rotation to aid in balancing, and in Section 3.8, we make a few concluding remarks.

3.2 Controlling Instantaneously Coupled Systems

For a certain type of system, which we call Instantaneously Coupled Systems (ICS), it is possible to construct an optimal controller by coordinating multiple optimal lower-dimensional controllers. First, subsystems are augmented with coordination variables, which provide enough information to account for coupling to other systems. Then, value functions are used to trade off the coordination variables. This is useful because it reduces an optimal control problem to several lower-dimensional optimal control problems, which can be solved more easily.

3.2.1 Instantaneously Coupled Systems

We define an instantaneously coupled system (ICS) as a dynamic system made up of a set of N lower-dimensional systems. The state of, $\mathbf{x}_f$, and input to, $\mathbf{u}_f$, the full system are given by the composition of the states of and inputs to the lower-dimensional systems,

$$\mathbf{x}_f = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \tag{3.2}$$

and

$$\mathbf{u}_f = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N\}. \tag{3.3}$$

The dynamics of each system evolve independently,

$$\dot{\mathbf{x}}_i = f_i(\mathbf{x}_i, \mathbf{u}_i). \tag{3.4}$$

At M specific instants, however, the systems may be coupled such that the dynamics of the subsystems instantaneously depend on the full state,

$$\mathbf{x}_i^+ = f_i^c(\mathbf{x}_f^-, \mathbf{u}_i), \tag{3.5}$$

where the superscripts $-$ and $+$ indicate before and after the coupling event.

The time of the coupling, t_j, is determined by some condition on the full state:

$$\Phi(\mathbf{x}_f(t_j)) = 0 \tag{3.6}$$

There can be one or multiple coupled instants. We only consider systems with a finite number, M, of coupled instants.

3.2.2 Obtaining the Optimal Policy

For an ICS with a cost function of the form

$$C = \int \sum_{i=1}^{N} L_i(\mathbf{x}_i(t), \mathbf{u}_i(t))\, dt + \sum_{j=1}^{M} \left(g(t_j) + h(\mathbf{x}_f(t_j))\right), \tag{3.7}$$

we can construct the optimal policy by finding the optimal policies and value functions for augmented versions of the subsystems and then combining them. Costs of coupling event times and state (g and h) are optional and are not used by the controller presented in this paper.

First, we define a coordination state, $\mathbf{x}_c$, as some set of features of the full state, $\mathbf{x}_c = \Theta(\mathbf{x}_f)$. The features, $\mathbf{x}_c$, are a compact means of communicating the essential information about the full state between the subsystems, and must be selected such that it is possible to:

I. Rewrite the coupling dynamics (3.5) as

$$f_i^c(\mathbf{x}_f, \mathbf{u}_i) = \tilde{f}_i^c(\mathbf{x}_i, \mathbf{x}_c, \mathbf{u}_i). \tag{3.8}$$

II. Rewrite the last term in (3.7) as

$$h(\mathbf{x}_f(t_j)) = \tilde{h}(\mathbf{x}_c(t_j)). \tag{3.9}$$

III. Rewrite (3.6) as the intersection of conditions on the low-dimensional systems

$$\Phi(\mathbf{x}_f(t)) = \Phi_1(\mathbf{x}_1(t), \mathbf{x}_c(t)) \cap ... \cap \Phi_N(\mathbf{x}_N(t), \mathbf{x}_c(t)). \tag{3.10}$$

It is always possible to choose $\mathbf{x}_c = \mathbf{x}_f$, but this method will be more useful if an $\mathbf{x}_c$ that is lower-dimensional than $\mathbf{x}_f$ can be found.

Next, we construct the decision space, $\mathbf{x}_d$, by composing t_j and $\mathbf{x}_c(t_j)$ from each of the coupled instants.

$$\mathbf{x}_d = \{t_1, \mathbf{x}_c(t_1), t_2, \mathbf{x}_c(t_2), ..., t_M, \mathbf{x}_c(t_M)\} \tag{3.11}$$

If we hold $\mathbf{x}_d$ constant, the subsystems are completely decoupled and the conditions from (3.10) are constraints:

$$\Phi_i(\mathbf{x}_i(t_j), \mathbf{x}_c(t_j)) = 0. \tag{3.12}$$

With the systems decoupled, we can individually optimize each one with respect to

$$C_i = \int L_i(\mathbf{x}_i(t), \mathbf{u}_i(t))\, dt, \tag{3.13}$$

the only part of (3.7) that depends on the ith system. It then remains only to optimize over all possible choices of $\mathbf{x}_d$ and select the best one.

To accomplish this, we augment the state of each of the subsystems with $\hat{\mathbf{x}}_d$,

$$\hat{\mathbf{x}}_d = \{\hat{t}_1, \mathbf{x}_c(t_1), \hat{t}_2, \mathbf{x}_c(t_2), ..., \hat{t}_M, \mathbf{x}_c(t_M)\} \tag{3.14}$$

$$\hat{t}_j = t_j - t \tag{3.15}$$

which has the trivial dynamics $\dot{\hat{t}}_j = -1$ and $\dot{\mathbf{x}}_c = 0$. This allows us to generate subsystem controllers that can apply the coupling dynamics (3.8) and know when to do so. We switch from the time of coupling in $\mathbf{x}_d$ to the time until coupling in $\hat{\mathbf{x}}_d$ to eliminate the dependence on time in our subsystem controllers. We then produce optimal (with respect to (3.13)) policies and value functions for each of the augmented systems subject to (3.12). Any method that produces both policies and value functions can be used, but we use dynamic programming.

Now, if we have an $\mathbf{x}_f$, we can hold each of the $\mathbf{x}_i$'s constant and get the value as only a function of $\mathbf{x}_d$. This allows us to rewrite (3.7) as only a function of $\mathbf{x}_d$, t, and $\mathbf{x}_f$:

$$C = \sum_{i=1}^{N} V_i(\mathbf{x}_d, t|\mathbf{x}_i) + k(\mathbf{x}_d) \tag{3.16}$$

where $k(\mathbf{x}_d) = \sum_{j=1}^{M} g(t_j) + \tilde{h}(\mathbf{x}_c(t_j))$. We then select the best decision state,

$$\mathbf{x}_d^* = \arg\min_{\mathbf{x}_d} C(\mathbf{x}_d, t, \mathbf{x}_f). \tag{3.17}$$

Having selected $\mathbf{x}_d$, we can look up each of the $\mathbf{u}_i$'s from the individual optimal policies and compose them to form $\mathbf{u}_f$ according to (3.3).

3.3 Walking as an ICS

To generate a walking controller, we first approximate walking as an ICS. Summing the forces and torques on the system gives us dynamics equations for the CoM

$$\mathbf{f}_L + \mathbf{f}_R + \mathbf{f}_g = m\ddot{\mathbf{c}} \tag{3.18}$$

$$(\mathbf{p}_L - \mathbf{c}) \times \mathbf{f}_L + \boldsymbol{\tau}_L + (\mathbf{p}_R - \mathbf{c}) \times \mathbf{f}_R + \boldsymbol{\tau}_R = \dot{\mathbf{l}} \tag{3.19}$$

where $\mathbf{c}$, $\mathbf{p}_L$, and $\mathbf{p}_R$ are the positions of the CoM, left and right feet, $\mathbf{f}_L$, $\mathbf{f}_R$, $\boldsymbol{\tau}_L$, and $\boldsymbol{\tau}_R$ are the reaction forces and torques generated at the feet, $\mathbf{f}_g = [0, 0, -g]^\mathsf{T}$ is the force of gravity, m is the mass, and $\mathbf{l}$ is the angular momentum. Since the absolute position is rarely relevant, it is useful to place the origin of the coordinate system at the stance foot so that the CoM location, $\mathbf{c}$, and the swing foot location, $\mathbf{p}_w$, are defined relative to the stance foot. During double support, the foot that will be in stance next is considered the stance foot. It is also useful to define the total reaction force and torque as follows:

$$\begin{aligned} \mathbf{f} &= \mathbf{f}_L + \mathbf{f}_R \\ \boldsymbol{\tau} &= \boldsymbol{\tau}_L + \boldsymbol{\tau}_R. \end{aligned} \tag{3.20}$$

During single support, the swing foot cannot generate reaction force, so one of the pairs of force and torque must be zero. If we then constrain our policy such that $\dot{\mathbf{l}} = 0$ and $\ddot{\mathbf{c}}_z = 0$, (3.18) and (3.19) simplify to the well-known Linear Inverted Pendulum Model (LIPM) [43] [45]. We further constrain the dynamics with $\dot{\mathbf{c}}_z = 0$ and $\mathbf{c}_z = h$ and write the LIPM dynamics as

$$\ddot{\mathbf{c}}_x = \mathbf{c}_x \frac{g}{h} + \frac{\boldsymbol{\tau}_y}{mh} \tag{3.21}$$

$$\ddot{\mathbf{c}}_y = \mathbf{c}_y \frac{g}{h} + \frac{\boldsymbol{\tau}_x}{mh}. \tag{3.22}$$

We model the swing leg as fully controllable and treat the acceleration of the swing foot, $\ddot{\mathbf{p}}_w$, as a control variable.

During double support, there is no swing foot to accelerate, but the horizontal CoM acceleration depends on how the weight is distributed between the two feet, which we define as

$$w = \frac{\mathbf{f}_{L,z}}{\mathbf{f}_{L,z} + \mathbf{f}_{R,z}}. \tag{3.23}$$

We assume that we can select w during double support such that

$$\ddot{\mathbf{c}}_x = \frac{\boldsymbol{\tau}_y}{mh} \tag{3.24}$$

$$\ddot{\mathbf{c}}_y = \frac{\boldsymbol{\tau}_x}{mh}. \tag{3.25}$$

Equations (3.24) and (3.25) are approximations because they require that both

$$w = \frac{\mathbf{c}_x - \mathbf{p}_{L,x}}{\mathbf{p}_{R,x} - \mathbf{p}_{L,x}} \tag{3.26}$$

and

$$w = \frac{\mathbf{c}_y - \mathbf{p}_{L,y}}{\mathbf{p}_{R,y} - \mathbf{p}_{L,y}}. \tag{3.27}$$

It is only possible to simultaneously satisfy (3.26) and (3.27) if the CoM is directly above the

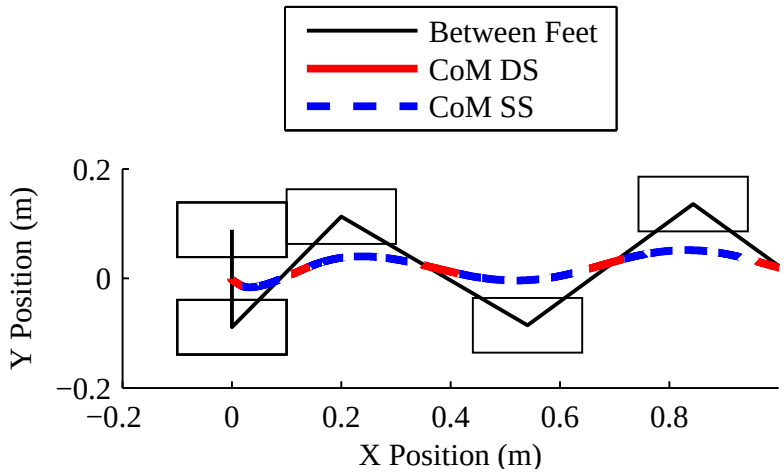

Figure 3.2: The CoM and footstep pattern of the walking simulation starting from rest. Note that during double support, the CoM is near the line between the two feet.

line between the two feet. However, this approximation is small because the CoM is usually near

this line during double support as shown in fig. 3.2, double support is brief, and the low-level controller can often fix some of the discrepancy by adjusting $\boldsymbol{\tau}$. This approximation is necessary because it allows us to decouple the sagittal and coronal dynamics, and it is useful because it allows us to calculate the CoM acceleration without knowing the position of both feet.

These dynamics constitute a 5 degree of freedom (DoF) ICS with a 10-dimensional state space (position and velocity for each DoF),

$$\begin{aligned}\mathbf{x}_f = \{&\mathbf{c}_x, \dot{\mathbf{c}}_x, \mathbf{c}_y, \dot{\mathbf{c}}_y, \mathbf{p}_{w,x}, \\ &\dot{\mathbf{p}}_{w,x}, \mathbf{p}_{w,y}, \dot{\mathbf{p}}_{w,y}, \mathbf{p}_{w,z}, \dot{\mathbf{p}}_{w,z}\}\end{aligned} \tag{3.28}$$

and a 5-dimensional action space (one for each DoF),

$$\mathbf{u}_f = \{\boldsymbol{\tau}_y, \boldsymbol{\tau}_x, \ddot{\mathbf{p}}_{w,x}, \ddot{\mathbf{p}}_{w,y}, \ddot{\mathbf{p}}_{w,z}\}. \tag{3.29}$$

We can then partition the state and action space into 5 subsystems, one for each DoF:

$$\begin{aligned}
\mathbf{x}_s &= \{\mathbf{c}_x, \dot{\mathbf{c}}_x\} & \mathbf{u}_s &= \{\boldsymbol{\tau}_y\} \\
\mathbf{x}_r &= \{\mathbf{c}_y, \dot{\mathbf{c}}_y\} & \mathbf{u}_r &= \{\boldsymbol{\tau}_x\} \\
\mathbf{x}_x &= \{\mathbf{p}_{w,x}, \dot{\mathbf{p}}_{w,x}\} & \mathbf{u}_x &= \{\ddot{\mathbf{p}}_{w,x}\} \\
\mathbf{x}_y &= \{\mathbf{p}_{w,y}, \dot{\mathbf{p}}_{w,y}\} & \mathbf{u}_y &= \{\ddot{\mathbf{p}}_{w,y}\} \\
\mathbf{x}_z &= \{\mathbf{p}_{w,z}, \dot{\mathbf{p}}_{w,z}\} & \mathbf{u}_z &= \{\ddot{\mathbf{p}}_{w,z}\}
\end{aligned} \tag{3.30}$$

where the subscripts, s, r, x, y, and z, refer to the sagittal stance, coronal stance, swing-x, swing-y, and swing-z subsystems.

The systems are only coupled during stance transitions (touch down and lift off). We choose a common state that describes the horizontal location of the swing foot, $\mathbf{x}_c = \{\mathbf{p}_x, \mathbf{p}_x\}$. In order to keep the decision state, $\hat{\mathbf{x}}_d$, low-dimensional, we consider only the next transition ($M = 1$) and make assumptions about all future transitions. This gives us a decision state of

$$\hat{\mathbf{x}}_d = \{t_t, x_{td}, y_{td}\} \tag{3.31}$$

where t_t is the time until transition, and $\{x_{td}, y_{td}\}$ is the location where the swing foot will touch down. For lift off transitions, x_{td} and y_{td} can be omitted. The stance subsystems assume

that subsequent transitions will have the nominal timing (0.1 second double support and 0.4 second single support), but that they will be able to select future touchdown locations. The swing subsystems assume that subsequent transitions will have nominal values from steady state walking. Fig. 3.3 shows t_t as a function of time for the walking simulation starting from rest and accelerating to steady state walking at 0.56 m/s. During single support, it is convenient to refer to t_t as time until touchdown, t_{td}.

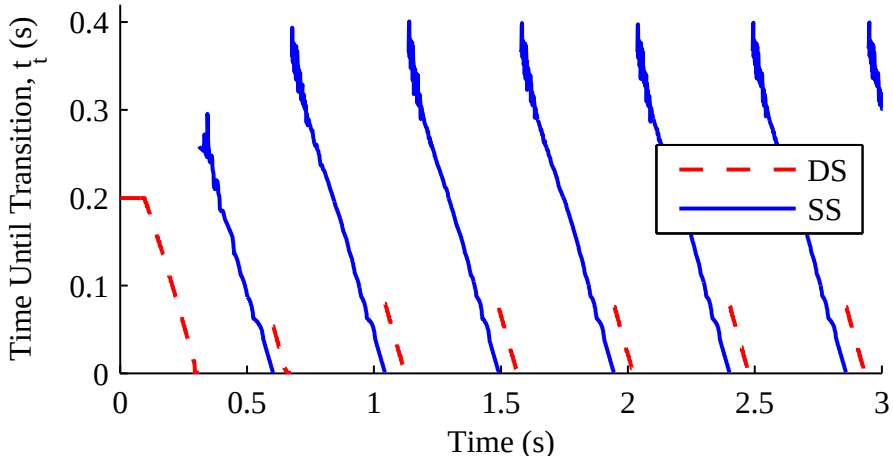

Figure 3.3: Time until transition, t_t versus time. To reduce computation, policies are only computed for $t_t < 0.2$ during double support.

This selection of $\mathbf{x}_c$ and the resulting $\hat{\mathbf{x}}_d$ allows our subsystem controllers to determine the optimal action for all possible choices of footstep timings and locations. The value functions can then be used to determine which choice of these variables is optimal for the full ICS.

We minimize the cost function

$$C = \int (w_1 \boldsymbol{\tau}_x^2 + w_2 \boldsymbol{\tau}_y^2 + w_3(\dot{\mathbf{c}}_x - v_{des})^2 + w_4(\mathbf{c}_y - w_h)^2 + w_5(\mathbf{p}_{w,z} - h_{fc})^2 + \ddot{\mathbf{p}}_{\mathbf{w}}^{\mathsf{T}} \mathbf{W}_6 \ddot{\mathbf{p}}_{\mathbf{w}})\, dt \tag{3.32}$$

subject to the constraint that

$$\mathbf{p}_w(t_{td} = 0) = \{x_{td}, y_{td}, 0\}. \tag{3.33}$$

The values w_1 through w_5 are weighting constants, $\mathbf{W}_6$ is a diagonal weighting matrix, w_h is

half the width of the hips, and h_{fc} is the nominal foot clearance height.

Note that (3.32) has the form of (3.7) and that (3.33) can be decoupled as in (3.10). This model of walking thus meets all the criteria of an ICS. If we omit the dimensions of $\hat{\mathbf{x}}_d$ that do not affect the dynamics, the original 10-dimensional system is equivalent to a coordinated set of one 3-dimensional system (swing-Z) and four 4-dimensional systems. In practice, we are able to use a change variables to reduce the swing-X and swing-Y policies from 4 dimensions to 3 dimensions and combine them (discussed in Section 3.4.1). We also add weight distribution to the state of the sagittal and coronal policies (discussed in the next Section) during double support. After these modifications, we have two 3-dimensional policies (one of which is used twice) and two policies that are 4-dimensional during single support and 5-dimensional during double support.

3.3.1 Non-ICS Modifications

The requirements for a system to be an Instantaneously Coupled System and therefore optimally coordinated by the method descibed in Section 3.2.2 are somewhat limiting. We make two modifications to the high-level walking model: we include weight distribution as a control during double support and we do not use the LIPM dynamics in the sagittal plane. These modifications result in the system not truly being an ICS, so our coordination is no longer optimal. Despite the loss of optimal coordination, the modifications significantly improve results by changing the restrictions on our walking controller.

Rather than allow the weight distribution between the two feet in double support, w, to be determined by the CoM position as described in (3.24) and (3.25), we wish to actively control the weight distribution to help with balance. This allows our controller to control center of mass location by adjusting how much of its total weight is on each foot, which moves the center of pressure. This is particularly helpful when starting from rest because it allows the controller to shift the center of mass towards the first stance foot during the initial double support phase.

Without shifting weight in this way, the system must step far out to the side of its first step when it begins walking from rest. Unfortunately, there is no way to coordinate the weight distribution within the ICS framework because it couples the sagittal and coronal sub-systems (they must have the same weight distribution) continuously during double support rather than just at the transition points. The run-time coordination between the two policies to determine the weight distribution during double support is discussed in Section 3.4.2.

The LIPM dynamics, (3.21) and (3.22), require that we do not accelerate the CoM vertically ($\ddot{\mathbf{c}}_z = 0$). For walking on flat ground, this implies a constant CoM height, $\mathbf{c}_z = h$. To achieve this, we must pick a height, h, that is low enough for the foot to reach the desired touch down locations. If we have this same CoM height during the middle of single support, the stance knee will be very bent, giving the appearance of a crouch-walk. If we then lower the height a bit more to add margin for the occasional abnormally long step required to recover from a perturbation, the problem becomes even more pronounced. This both looks unnatural and drastically increases the torque required at the knee. We therefore wish to allow the CoM to move up and down as the system walks. Unfortunately, the LIPM dynamics were the reason that sagittal and coronal plane motion decoupled completely. Clearly, the CoM must be at the same height for both sub-systems at all times. We allow the sagittal plane controller to control the CoM height and treat the effect of the vertical motion as a disturbance on the coronal plane, for which we continue to use the LIPM dynamics for generating a high-level policy. We consider the LIPM dynamics acceptable for high-level policy generation in the coronal plane because steps are much shorter in the y direction and the legs typically remain near vertical in the coronal plane.

We do not wish to increase the dimensionality of our system by adding $\mathbf{c}_z$ and $\dot{\mathbf{c}}_z$ as states in our system, so we set the CoM height to a function of x and d, $h(x, d)$, where x and d are the CoM location in the X direction and the distance between the feet, measured as shown in Fig. 3.4. Measuring torque around the CoM gives us

$$\dot{\mathbf{l}} = \boldsymbol{\tau}_y + \mathbf{f}_x h(x, d) - xw\mathbf{f}_z - (d + x)(1 - w)\mathbf{f}_z. \tag{3.34}$$

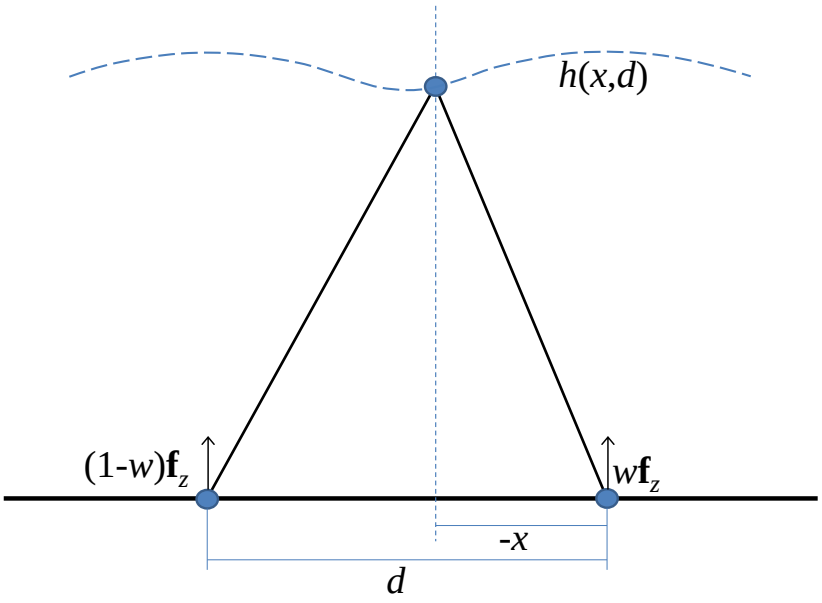

Figure 3.4: Diagram of the sagittal system not using LIPM dynamics. The fraction of the total vertical force, $\mathbf{f}_z$ on the right leg is given by w. Note that in the configuration drawn, x has a negative sign.

We have the constraint that $\dot{\mathbf{l}} = 0$. When $\mathbf{f}_z$ is not constant, the ZMP constraint does not give us a constant limit on $(\tau)_y$, so it is convenient to write our control in terms of center of pressure location on the foot. The relative CoP location, r is related to the torque by $\tau = r\mathbf{f}_z$ and constrained by $|r| \leq l_f$ where $l_f = 0.1$ is half the length of the foot. We know from Newton's Second Law that $\mathbf{f}_x = m\ddot{x}$ and adding gravity gives us that $\mathbf{f}_z = m(g + \ddot{h}(x,d))$. If we assume that $\dot{d} = 0$, we get that

$$\ddot{h}(x,d) = \frac{\partial^2 h}{\partial x^2}\dot{x} + \frac{\partial h}{\partial x}\ddot{x}. \tag{3.35}$$

Substituting all of these relations into (3.34) gives us

$$0 = m\ddot{x}h(x,d) + m(g + \frac{\partial^2 h}{\partial x^2}\dot{x} + \frac{\partial h}{\partial x}\ddot{x})(r + dw - d - x), \tag{3.36}$$

which we can solve for $\ddot{x}$ to get

$$\ddot{x} = \frac{(g + \frac{\partial^2 h}{\partial x^2}\dot{x})(x + d - dw - r)}{h(x,d) + \frac{\partial h}{\partial x}(r + dw - d - x)}. \tag{3.37}$$

In single support, $w = 1$ and (3.37) still holds with $dw - d = 0$ cancelling out. This removes any dependence on d, (so long as h does not depend on d), which makes sense because the second foot is not on the ground during single support.

During single support, $h(x, d)$ is based on the compass gait, with the hip moving in an arc around the stance foot and the CoM located a fixed offset, $a = 0.2$ above the hip:

$$h(x, d) = a + \sqrt{l_{SS}^2 + x^2}, \tag{3.38}$$

where l_{SS} is the constant length of the stance leg during single support. As the hip moves forward during double support, the front leg must get shorter (bend the knee more) and the back leg must get longer (straighten the knee). For this to be possible, we can not start double support with the back leg completely straight. For this reason, we keep the knee slightly bent during single support, making l_{SS} shorter than the full leg length. Note that this is necessary because our controller walks with flat footsteps, with no toe off. Humans (and robots that have toe off in their gait) lengthen their back leg during double support by lifting their heel off of the ground.

During double support, we wish to make a smooth transition between the preceding and following single support arcs. This is impossible with h being only a function of x and d because we do not know when/where touchdown occured or when/where liftoff will occur, so we attempt to make it as smooth as possible using nominal values. First, we find the expected double support region (the region the CoM will pass through in a nominal step), which has a length of the desired velocity times the nominal double support duration and is centered between the two feet. If the feet are outside this region (as they usually will be), we find the circle that is tangent to the two single support arcs as they enter this region (shown in Fig. 3.5). If the CoM is in the expected double support region, $h(x, d)$ is on that joining circle. If outside the expected region, $h(x, d)$ is on the tangent lines. If the feet are very close together and inside the expected DS region, we again find the circle that is tangent to both single support arcs as they enter the expected DS region. This time the circle's center will be below the ground (shown in Fig. 3.6). We then assign $h(x, d)$ to this circle regardless of whether or not it is in the expected DS region. In the

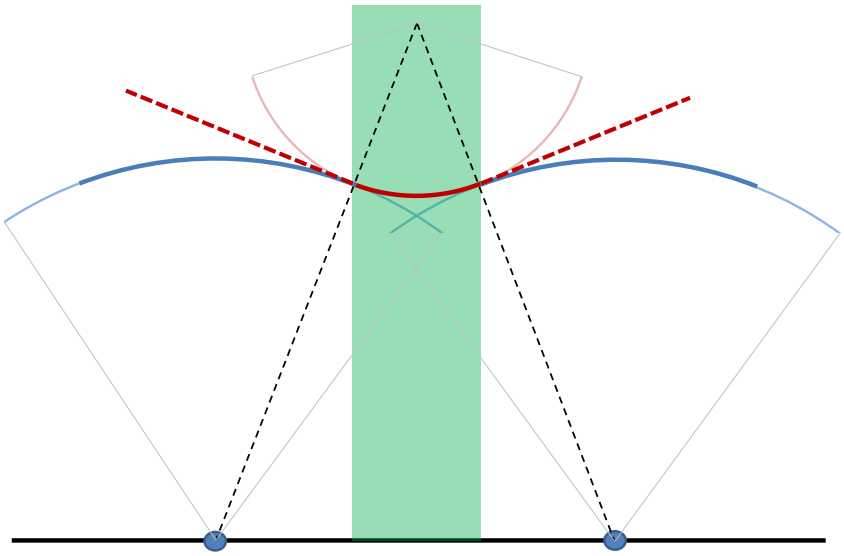

Figure 3.5: Diagram of $h(x, d)$. The green region is the expected double support region. The blue linse show the single support arcs. The red line shows $h(x, d)$ during double support with the joining arc solid and the tangent lines outside of the expected region dashed.

boundary case where the feet are right at the edge of the expected DS region, $h(x, d)$ will be a simple horizontal line during double support.

3.4 Walking Controller

We use the principle of an ICS to generate a walking controller for a simulated biped based on our Sarcos Primus System [16] [51] hydraulic humanoid robot with force-controlled joints. The simulation is of approximately human size (CoM is 1.0 m high when standing straight) and mass (78 kg). It is a 3D 5-link (torso and two 2-link legs) rigid body simulation with 16 degrees of freedom: 6 to locate and orient the torso as well as 3 at each hip and 1 at each knee. It is controlled by 12 torque controlled joints: 3 at each hip, 1 at each knee, and roll and pitch actuation between

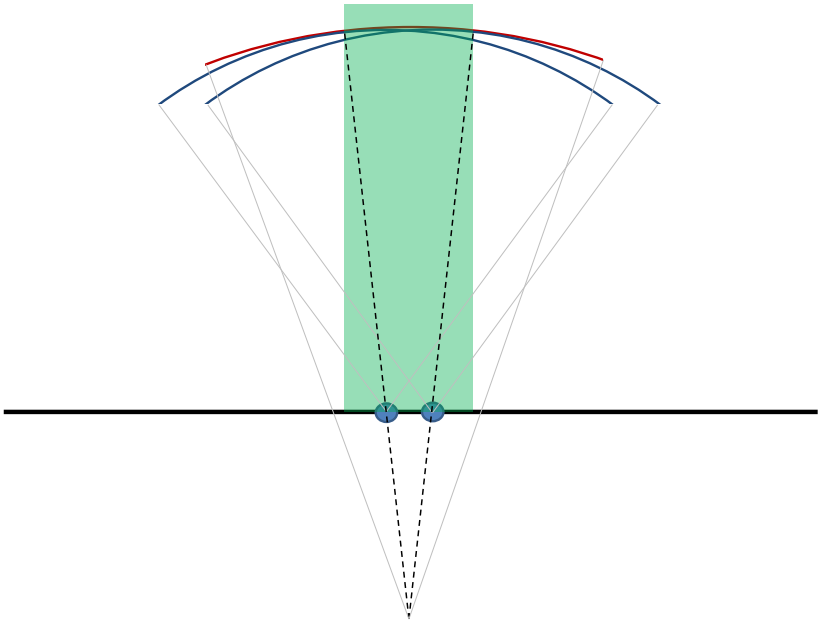

Figure 3.6: Diagram of $h(x, d)$ when the feet are inside the expected double support region (green region). The blue lines show the single support arcs, and the red line shows the double support arc when the feet are within the expected double support region.

each point foot and the ground. The CoP constraint of a finite-size foot is simulated by enforcing

$$\begin{aligned} |\boldsymbol{\tau}_x| &\leq w_f \mathbf{f}_z \\ |\boldsymbol{\tau}_y| &\leq l_f \mathbf{f}_z \end{aligned} \tag{3.39}$$

on each foot where $w_f = 0.05m$ and $l_f = 0.1m$ are approximately half the width and length of a human foot. Friction (coefficient of friction is $\mu = 1.0$) is modeled as a spring and damper between the foot and the ground. When the friction cone,

$$\frac{\sqrt{\mathbf{f}_x^2 + \mathbf{f}_y^2}}{\mathbf{f}_z} < \mu, \tag{3.40}$$

or yaw torque constraint,

$$\frac{\boldsymbol{\tau}_z}{\mathbf{f}_z} < \mu_r, \tag{3.41}$$

is violated, slipping is modeled by resetting the rest position of the spring and switching to a lower kinetic coefficient of friction ($\mu_k = 0.8$).

We use coordinated DP polices to produce an optimal controller for the ICS described in Section 3.3. This functions as our high-level controller, providing input CoM and swing foot accelerations to a low-level controller, which outputs joint torques.

3.4.1 Policy Generation

Policies and value functions are generated for each of the five subsystems using dynamic programming as discussed in Section 2.2. A discount factor of 0.9995 is used, which corresponds to costs fading to half importance after 1.4 seconds (nearly 3 steps).

Swing-Z Policy Generation

For the swing-z system, the dynamics are not affected by x_{td} or y_{td}, so it is sufficient to generate a policy on the 3-dimensional state space of $\{\mathbf{p}_{w,z}, \dot{\mathbf{p}}_{w,z}, t_{td}\}$ - denoted $\{z, \dot{z}, t_{td}\}$ here for convenience. Our only control action is $\dot{\mathbf{p}}_{w,z}$ - denoted $\ddot{z}$. We are controlling acceleration directly in the high-level walking system, so it has simple second order dynamics and t_{td} simply counts down. We wish to immediately lift the swing foot, hold it steady at a nominal height of $z_{\text{nom}} = 0.03\text{m}$, then reach the ground ($z = 0$) with a small nominal touchdown velocity of $\dot{z}_{td} = -0.04\text{m/s}$ when $t_{td} = 0$. We use a non-zero touchdown velocity to ensure that we get firm contact and to avoid numerical issues, but we keep it small to avoid large impacts. We divide the state space into a grid with minimum states of {0 m, -3 m/s, 0 s}, maximum states of {0.08 m, 3 m/s, 0.6 s}, and resolutions of {81, 301, 121}. To simplify analysis, where possible we select grid resolutions such that we have round numbers for the grid spacing. For this policy, for example, we have spacing of {0.001 m, 0.02 m/s, 0.005 s}. We use a cost function of

$$L(\mathbf{x}, \mathbf{u}) = 0.5 \left(\frac{\ddot{z}}{\ddot{z}_{\max}} \right)^2 + \left(\frac{z - z_{\text{nom}}}{z_{\text{nom}}} \right)^2, \tag{3.42}$$

where $z_{\max}$ is the maximum allowable acceleration. The first term minimizes acceleration, and the second term causes the foot to lift off and hold at the nominal height, z_{nom}. The weight, 0.5,

was selected manually by experimentation and the denominators are used to normalize terms that have different units.

For $t_{td} \leq 0.03$s, we use an analytical controller to select an action and produce a corresponding value function. An analytical function is necessary because the second derivatives of the policy and value function grow arbitrarily large as we approach touchdown, which would require an arbitrarily fine grid spacing to approximate accurately. Our analytic controller selects the single acceleration, $\ddot{z}$, to use from now until touchdown that minimizes the cost

$$C = 2t_{td}\ddot{z}^2 + 1000(\dot{z}_f - \dot{z}_{\text{nom}})^2, \tag{3.43}$$

subject to the constraint that it actually touch down ($z = 0$) within $T_{\text{slop}} = .00075$s (3/4 of a simulation time step) of the nominal t_{td}. The final velocity is given as $\dot{z}_f = \dot{z} + \ddot{z}t_{td}$. The constant 2 was selected to scale the cost of terminal acceleration relative to the cost of acceleration the rest of the time. Note that this is not normalized and is in the continuous rather than discrete time setting. We somewhat arbitrarily determined that a terminal acceleration of 10 m/s/s was about equally bad as missing the final velocity by 0.02 m/s, which tells us that the second constant should be 500 times larger than the first.

First we find the $\ddot{z}$ which minimizes (3.43) ignoring the constraint by expanding (3.43), which gives a quadratic in $\ddot{z}$, which can be easily minimized. Then we find the range of $\ddot{z}$ that satisfies the timing constraint by finding the $\ddot{z}$ that achieves $z = 0$ in $t_{td} + T_{\text{slop}}$ and the $\ddot{z}$ that achieves $z = 0$ at $t_{td} - T_{\text{slop}}$ and constrain our action to this range. We must also satisfy the constraint that $|\ddot{z}| \leq \ddot{z}_{\max}$. If these ranges do not overlap, the problem is not solvable and we assign an infinite value. Otherwise, we plug our $\ddot{z}$ into (3.43) to get the value. For much of the state space, it will not be possible to satisfy both constraints so we will get an infinite value. For most of the rest of the state space, the timing constraint will be active and the cost function only serves to determine whether we touch down as early as possible ($t_{td} - T_{\text{slop}}$) or as late as possible ($t_{td} + T_{\text{slop}}$). The value of T_{slop} was selected such that for every physical state $\{z, \dot{z}\}$, there is at least one integer number of time steps for which touchdown is feasible (some $\ddot{z}$ satisfies both constraint pairs).

Swing-X and Swing-Y Policy Generation

We use the same policy for both the Swing-X and Swing-Y sub-systems because they have the same dynamics and cost function (just with different names for the variables). From Section 3.3, we have that the state for the Swing-X system is $\{\mathbf{p}_{w,x}, \dot{\mathbf{p}}_{w,x}, x_{td}, t_{td}\}$. Because x_{td} has constant dynamics, we can reduce the state space of the policy by putting the origin at x_{td}, making the state $\{\mathbf{p}_{w,x} - x_{td}, \dot{\mathbf{p}}_{w,x}, 0, t_{td}\}$. We can drop the 0 from the state and for convenience, we denote the state as $\{p, \dot{p}, t_{td}\}$. We denote our control as $\ddot{p}$. We wish for our controller to smoothly move the foot to the target location ($p = 0$ with the change of coordinates) and arrive there when $t_{td} = 0$ with zero velocity ($\dot{p} = 0$). We will handle the destination requirements in a terminal analytic controller, so we need only minimize acceleration,

$$L(\mathbf{x}, \mathbf{u}) = \left(\frac{\ddot{p}}{\ddot{p}_{\max}} \right)^2, \tag{3.44}$$

in the grid.

Our grid must cover a larger region of state space for this policy than for the Swing-Z policy because the foot moves much farther in the x direction than in the z direction. To keep the computation reasonable, we use two grids: a coarse grid for the entire state space and a fine grid near the goal. The coarse grid has minimum states of {0 m, -4 m/s, 0 s}, maximum states of {1.2 m, 4 m/s, 0.6 s}, and resolutions of {121, 401, 121}. Because the dynamics and cost function are symmetric, we know that $\ddot{p} = \pi(p, \dot{p}, t_{td}) = -\pi(-p, -\dot{p}, t_{td})$. Therefore, we can cut our grid in half and only consider $p \geq 0$. If $p < 0$, we can return $-\pi(-p, -\dot{p}, t_{td})$, which is in our grid. We also have to include this transformation in our dynamics when computing the policy so that simulating forward one time step from, for example, $\{0\mathrm{m}, -1\mathrm{m/s}, 0.1\mathrm{s}\}$ does not end up off of the grid.

The smaller grid has minimum states of {0 m, -3.5 m/s, 0.03 s}, maximum states of {0.175 m, 3.5 m/s, 0.1 s}, and resolutions of {176, 701, 71}. The limits of 3.5 m/s and 0.175 m were found by starting at the origin and working backward for 0.1 s at the maximum acceleration, $\ddot{p}_{\max} = 35\mathrm{m/s/s}$. The two grids overlap for simplicity, but we only use the finer grid in the

region where it exists. We can stop the finer grid at 0.03 seconds because we use an analytic controller for $t_{td} \leq 0.03\text{s}$.

Our analytic controller is similar to the one used for the Swing-Z policy. It minimizes

$$C = 4t_{td}\ddot{p}^2 + 2000\dot{p}_f^2, \tag{3.45}$$

which is similar to (3.43), and subject to the constraint that $|p_f| \leq p_{\text{slop}}$ where p_f is the position, p, when $t_{td} = 0$ and $p_{\text{slop}} = 0.0005\text{m}$ allows just enough mismatch to ensure that a fine search (resolution 1 mm) of potential x_{td}'s (or y_{td}'s) will find at least one x_{td} (or y_{td}) that is feasible. The constants in (3.45) are half those in (3.43) because the coefficient of 2 in (3.42) is absent from (3.44).

As for the Swing-Z policy, the touchdown location constraint will be active for most of the state space, and the minimization will only serve to determine in which direction to miss the desired location by p_{slop}. To solve, we first do the minimization, then constrain it by the touchdown location constraint, then constrain it by the maximum acceleration constraint, $|\ddot{p}| \leq \ddot{p}_{\max} = 35\text{m/s/s}$.

Sagittal Policy Generation

During single support, the sagittal policy has a 4-dimensional state space: CoM position relative to the stance foot, CoM velocity, time until touchdown, and touchdown location relative to the stance foot, which we denote as $\{x, \dot{x}, t_{td}, x_{td}\}$. The only control action is the center of pressure location on the stance foot, $|r_x| \leq l_f$, where $l_f = 0.08\text{m}$ is half the length of the foot. For the dynamics, x_{td} is constant, t_{td} counts down, and we find $\ddot{x}$ according to (3.37), which we integrate one time step to update x and $\dot{x}$. For 0.5 m/s walking, we use a cost function of

$$L(\mathbf{x}, \mathbf{u}) = 2\left(\frac{\dot{s}}{v_{\text{des}}}\right)^2 + \left(\frac{r_x}{l_f}\right)^2, \tag{3.46}$$

where v_{des} is the desired walking speed in m/s. We can generate policies for walking slower by decreasing v_{des}, but more complicated cost functions appear to be necessary to walk faster stably.

We use a grid with minimum states of {-0.4 m, -0.3 m/s, 0 s, 0 m}, maximum states of {0.4 m, 1.5 m/s, 0.6 s, 0.7 m}, and resolutions of {81, 46, 121, 71}.

During double support, we have a 5-dimensional state space. Time until liftoff, t_{lo}, replaces time until touchdown, stance width, d, replaces x_{td}, and we add weight fraction on the front leg, w, giving us a state of $\{x, \dot{x}, t_{lo}, d, w\}$. For convenience, we measure the CoM position, x, relative to the center point between the two feet during double support. We also add a second action, $\dot{w}$. The dynamics are similar to single support except that we use $\dot{w}$ to update w. Note that we must first get the CoM position relative to the lead foot before we can use (3.37) to get the CoM acceleration. We must add a few extra terms to the cost function, (3.46), to get

$$L(\mathbf{x}, \mathbf{u}) = 2\left(\frac{\dot{s}}{v_{\text{des}}}\right)^2 + \left(\frac{r_x}{l_f}\right)^2 + 0.7\left(\dot{w}T_{\text{DS}}\right)^2 + \left(\frac{x}{v_{\text{des}}T_{\text{DS}}}\right)^2, \tag{3.47}$$

where $T_{\text{DS}} = 0.1\text{s}$ is the nominal duration of double support. The third term is added to limit $\dot{w}$, and the fourth term is added to keep the CoM near the middle of the stance region. Without this cost, the coordinated system tends to continue double support for too long and start dragging its back foot forward when the CoM moves far enough forward that the back leg gets completely straight. We use a grid with minimum states of {-0.2 m, -0.3 m/s, 0 s, 0 m, 0.1}, maximum states of {0.2 m, 1.5 m/s, 0.2 s, 0.7 m 0.9}, and resolutions of {41, 46, 41, 71, 17}. To enforce that $w = 0.9$ at the end of double support, when $t_{lo} \leq 0.01\text{s}$, rather than selecting $\dot{w}$ as an action, we use $\dot{w} = (0.9 - w)/t_{lo}$.

At the end of single support, when $t_{td} = 0$, we transition to double support by changing where x is measured relative to, setting $t_{lo} = T_{\text{DS}}$, $d = x_{td}$, and $w = 0.1$. If the distance between the hip and the foot that is touching down is greater than the length of the leg, we set the cost to infinite. Additionally, if this distance is within 2 cm of the straight length of the leg, we add a cost which increases linearly from 0 at 2cm of margin to 100 at no margin to encourage the controller to leave a little bit of margin.

At the end of double support, when $t_{lo} = 0$, we transition to single support. At this time, we select the length of the step as an action. To reduce memory costs in implementation, we

transition to a special liftoff dynamics for one step (representing zero time) between double support and single support. The liftoff dynamics have only a 2-dimensional state space, $\{x, \dot{x}\}$, and a single action, x_{td}. The grid has the same minimum state, maximum state, and resolution as the first two states of the single support grid. To transition from double support to the liftoff dynamics, we need only change the reference frame for x (from measuring from the center of stance to measuring from the new stance foot). To transition from the liftoff dynamics to single support, we set t_{td} to the nominal single support duration of 0.4 seconds and set the state x_{td} according to the action in the liftoff dynamics. There is no cost for the liftoff dynamics ($L(\mathbf{x}, \mathbf{u}) = 0$).

Coronal Policy Generation

The coronal policy is very similar to the sagittal policy. During single support, the state is identical, except with y instead of x: $\{y, \dot{y}, t_{td}, y_{td}\}$. When the left foot is in single support, the positive-y direction is measured to the right, and vice versa. The only action is the center of pressure location relative to the stance foot, r_y, which is constrained by $|r_y| \leq w_f$, where $w_f = 0.04\text{m}$ is half the width of the foot. We use the LIPM dynamics (3.22) to compute the CoM acceleration, $\ddot{y}$. Again, t_{td} counts down, and y_{td} is constant. We use the cost function

$$L(\mathbf{x}, \mathbf{u}) = \left(\frac{r_y}{w_f}\right)^2. \tag{3.48}$$

We use a grid with minimum states of {-0.05 m, -1 m/s, 0 s, 0 m}, maximum states of {0.4 m, 1 m/s, 0.6 s, 0.6 m}, and resolutions of {46, 45, 121, 61}.

During double support, t_{td} becomes t_{lo}, y_{td} becomes the stance width, d, and we add the fraction of the vertical force on the next single support leg, w, making the state $\{y, \dot{y}, t_{lo}, d, w\}$. We again add $\dot{w}$ as a second action, which controls the dynamics of w. We again use the LIPM dynamics to find $\ddot{y}$, but now we need to consider how much vertical force is on each foot, w, as well as the shift in CoP due to ankle torque, r_y to find the total CoP. To get the cost function, we

add a term limiting $\dot{w}$ to the single support cost function, producing

$$L(\mathbf{x}, \mathbf{u}) = \left(\frac{r_y}{w_f}\right)^2 + 0.7\left(\dot{w}T_{\mathrm{DS}}\right)^2. \tag{3.49}$$

As in the sagittal sub-system, we enforce that $w = 0.9$ at liftoff by assigning $\dot{w} = (0.9 - w)/t_{lo}$ when $t_{lo} \leq 0.01\mathrm{s}$. During double support, we measure the CoM position, y, relative to the next single support leg with the positive-y direction pointing towards the other foot. We use a grid with minimum states of {-0.05 m, -1 m/s, 0 s, 0 m, 0.1}, maximum states of {0.4 m, 1 m/s, 0.2 s, 0.6 m 0.9}, and resolutions of {46, 45, 41, 31, 17}.

To transition from single support to double support at touchdown, we need to shift the reference frame for y and flip the sign of $\dot{y}$. We also set $t_{lo} = T_{\mathrm{DS}}$, $d = y_{td}$, and $w = 0.1$.

To transition from double support to single support at lift off, we again use a special lift off dynamics as in the sagittal subsystem. Again, it is a 2-dimensional grid with step width for the next step, y_{td} as the only action. Here we have a cost on the step width to encourage reasonable step widths,

$$L(\mathbf{x}, \mathbf{u}) = 0.5\left(\frac{y_{td} - y_{td,\mathrm{NOM}}}{0.1}\right)^2 \frac{T_{\mathrm{DS}} + T_{\mathrm{SS}}}{\Delta t}, \tag{3.50}$$

where $y_{td,\mathrm{NOM}} = 0.22\mathrm{m}$ is the nominal touchdown width and Δt is the time step. The second fraction is necessary to scale this cost to compensate for the fact that it happens once per step rather than continuously every time step.

These five DP policies (three 3-dimensional and two 4-dimensional policies) are equivalent to a single 10-dimensional DP policy for the entire ICS. If we use a resolution of 100 states per dimension, the coordinated version uses 2.3×10^8 states as opposed to 1.0×10^{20} states for the equivalent single policy. Computing the DP policies is computationally intensive and can take on the order of a day for our 4-dimensional policies. They are computed before use, and this computation does not affect the run-time performance of the controller.

3.4.2 Policy Coordination

Double Support

At run time, we combine the value functions to obtain $\mathbf{x}_d^*$ as in (3.17). During double support, the $\arg\min$ operation is only a 1-dimensional search (we must only find the time until lift off), so it can be performed by a fine resolution brute force search. During single support, however, the search space is 3-dimensional(t_{td}, x_{td}, and y_{td}), so a brute force search would be too computationally expensive.

During double support, we also have to determine how to shift weight from the foot that used to be in single support to the foot that is about to be in single support. Both the sagittal and coronal policies have the fraction of weight on the lead leg w as a state and $\dot{w}$ as an action in double support. Averaging the two commanded $\dot{w}$'s would be the simplest solution, but this will do poorly if one of the policies absolutely needs a specific $\dot{w}$ and the other one can handle just about anything. To tell how important achieving the desired $\dot{w}$ is for a given policy, we must look at the value function.

We generally wish to minimize value, and we already have the value as a function of w, so we find the $\dot{w}$ that reduces the value the most while holding everything else constant, $\min \partial V/\partial t|_{x!=w}$. We can expand this to

$$\frac{\partial V}{\partial t} = \frac{\partial V}{\partial w}\frac{dw}{dt} + \frac{1}{T}L(\mathbf{x}, \mathbf{u}) \tag{3.51}$$

where the first term is the change in value due to changing the state and the second term is the cost of changing the state (T is the time step). If we drop all the terms in L that do not contain w this gives us

$$\frac{\partial V}{\partial t} = \left(\frac{\partial V_s}{\partial w} + \frac{\partial V_c}{\partial w}\right)\dot{w} + \frac{1}{T}(a_s + a_c)\dot{w}^2 \tag{3.52}$$

where a_s and a_c are the cost function weights for the sagittal and coronal policies. To minimize $\partial V/\partial t$, we take the derivative of this with respect to $\dot{w}$ and set it equal to zero:

$$\frac{\partial V_s}{\partial w} + \frac{\partial V_c}{\partial w} + \frac{2}{T}(a_s + a_c)\dot{w} = 0, \tag{3.53}$$

which we can solve for $\dot{w}$, giving us

$$\dot{w} = \frac{-2(a_s + a_c)}{T\left(\frac{\partial V_s}{\partial w} + \frac{\partial V_c}{\partial w}\right)}. \tag{3.54}$$

Parabolic Approximation Method

To speed up the search, we note that all five value functions depend on t_{td}, but that only 2 each depend on x_{td} and y_{td}, and that none of the value functions depend on both x_{td} and y_{td}. We wish to first find $x^*_{td}(t_{td})$ and $y^*_{td}(t_{td})$, so that we can then perform a 1-dimensional search over $V(t_{td}|x^*_{td}(t_{td}), y^*_{td}(t_{td}), \mathbf{x}_f)$.

To do this, we approximate the value functions (during pre-computation) in such a way that they can be added quickly and that $x^*_{td}(t_{td})$ and $y^*_{td}(t_{td})$ of the sums can be found analytically. For the coronal and swing-y value functions, we approximate the value function, $V(t_{td}, y_{td}|\mathbf{x}_i)$, with a series of parabolic approximations to $V(y_{td}|t_{td}, \mathbf{x}_i)$ for evenly spaced values of t_{td}. Each parabola is created by placing the vertex at the minimum of $V(y_{td}|t_{td}, \mathbf{x}_i)$ and using a point to either side to estimate the second derivative. Fig. 3.7 shows an example surface approximation. The sum of two parabolas is also a parabola, so two surfaces can then be added quickly by adding the parabolas, creating a new surface also represented by a series of parallel parabolas. The location of the vertex of each of these new parabolas gives us $y^*_{td}(t_{td})$, and the value at each vertex gives us $V_{C+Y}(t_{td}|y^*_{td}, \mathbf{x}_f)$, where V_{C+Y} indicates the value for the coronal and swing-y subsystems together.

To do this quickly at run-time, we must first pre-compute the parabolic approximations to $V(y_{td}|t_{td}, \mathbf{x}_i)$ for each of the policies. We compute the approximation (as shown in Fig. 3.7) for a grid of $\mathbf{x}_i$ with the same resolution as the main DP grid. At run time, when we look up the surface approximation, we use the actual $\mathbf{x}_i$ and multilinear interpolation on the grid of approximations. We have found that parameterizing the parabola as

$$y = a(x - h)^2 + k \tag{3.55}$$

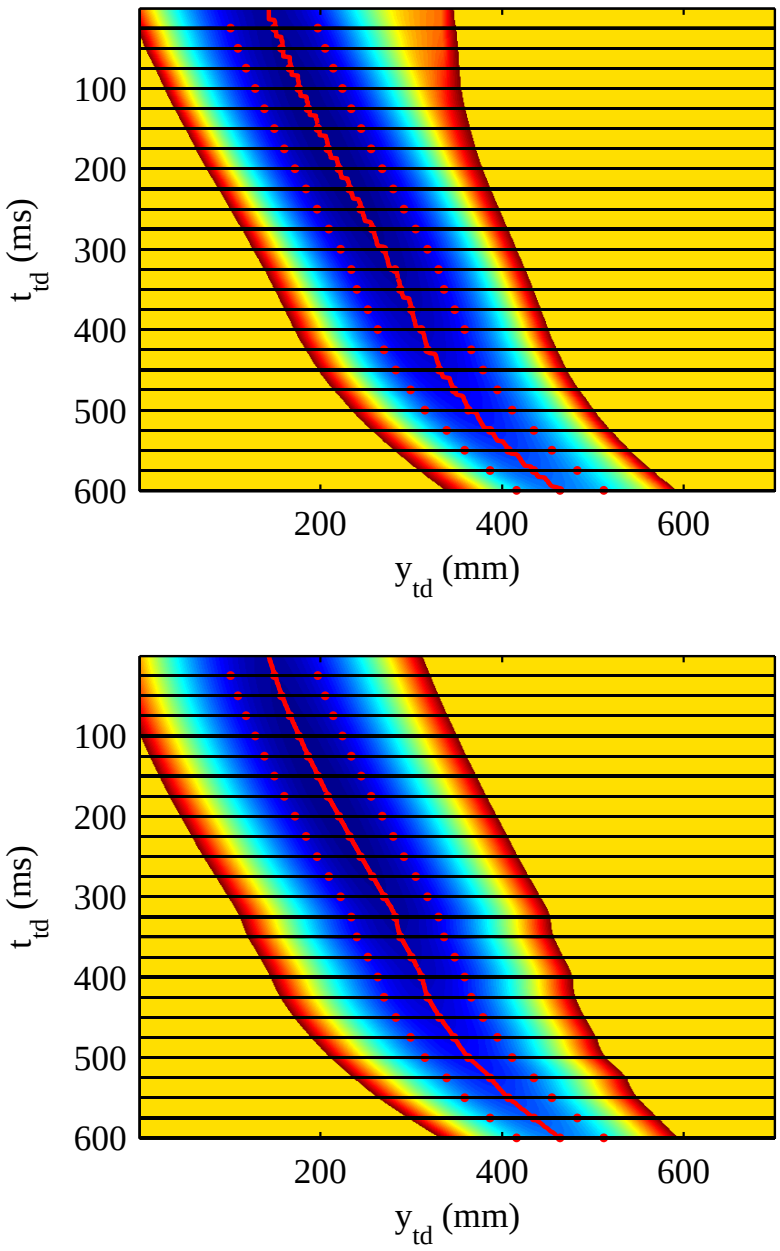

Figure 3.7: The coronal stance value function, $V(t_{td}, y_{td}|\mathbf{c}_y = 0.08, \dot{\mathbf{c}}_y = 0)$, from the DP tables (top) and from the parabolic approximation (bottom). The red line shows $y^*_{td}(t_{td})$. The dots show the points used to generate the parabolic approximation, and the horizontal black lines show the location of the parabolas.

and interpolating the parameters a, h, and k works better than parameterizing the parabolas as

$$y = ax^2 + bx + c \tag{3.56}$$

and interpolating the parameters a, b, and c because the former parameterization does a better job of capturing the things we care about: the location and value of the vertex. It is worth noting that these parabolic approximations are both much easier to compute and smaller to store than the original DP value function, so using them does not add significantly to our pre-computation requirements.

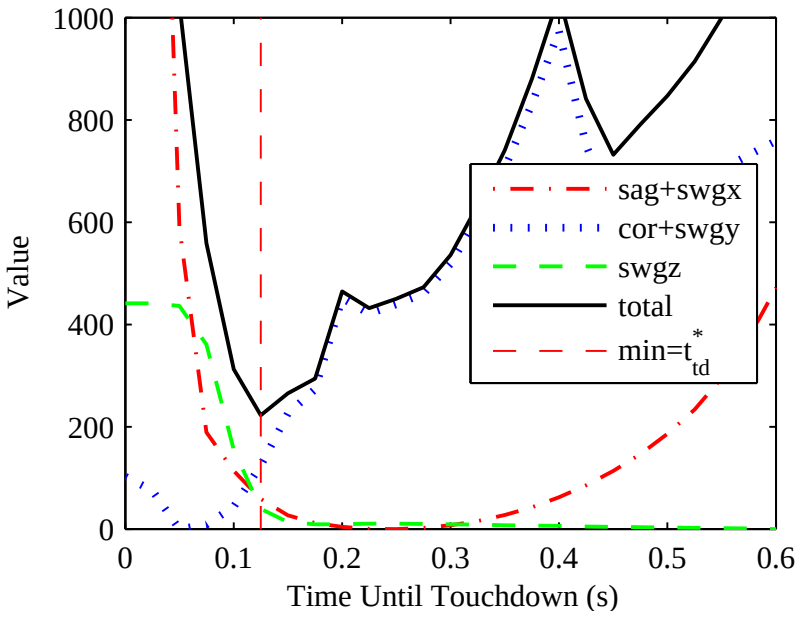

Figure 3.8: Value as a function of t_{td} following a push to the side. For ease of view, values are normalized so that the minimum is 0. Due to the push, the coronal policy wants to touch down soon, but it must compromise with the other policies to pick the best time, t^*_{td} for the full ICS.

The same is done with the sagittal stance and swing-x value functions, using x_{td} instead of y_{td}. With the value functions reduced to only a function of t_{td} as shown in Fig. 3.8, they can be efficiently added

$$V(t_{td}|x^*_{td}, y^*_{td}, \mathbf{x}_f) = V_{C+Y}(t_{td}|y^*_{td}, \mathbf{x}_C, \mathbf{x}_Y) + V_{S+X}(t_{td}|x^*_{td}, \mathbf{x}_S, \mathbf{x}_X) + V_Z(t_{td}|\mathbf{x}_Z) \tag{3.57}$$

to produce a value dependent only on t_{td}. This value function is represented as a series of values, $V(t_{td}|x^*_{td}, y^*_{td}, \mathbf{x}_f)$, with the same spacing as the parabolas in the surface approximation. We pick the point with the lowest value, then fit a parabola to it and its two neighbors. The location of the fit parabola's vertex gives us t^*_{td}. We can then plug t^*_{td} into the $x^*_{td}(t_{td})$ and $y^*_{td}(t_{td})$ functions we found earlier to look up x^*_{td} and y^*_{td}. Having determined $\hat{\mathbf{x}}_d$, we can now look up the appropriate controls, $\mathbf{u}_f$, from the individual policies.

The run-time computation for this method is extremely simple, and requires only about 6 microseconds of computation time on a 6 core 3.33 GHz Pentium i7 processor.

Unfortunately, this method relies on the structure of the value functions produced by dynamic programming; if they do not have the form we expect, then our approximations will not be sufficiently accurate to find a good solution. If, for example, we wish to handle varied terrain, we would modify the value function by using a non-zero h function in (3.7). For complicated terrain, this can make the final cost function arbitraril complex.

Optimization Method

To handle situations where we do not have the simple form of the value functions for which the convenient approximations work, we use a more generic method. We treat finding $\hat{\mathbf{x}}_d = \{t^*_{td}, x^*_{td}, y^*_{td}\}$ as a generic optimization problem and use the Nelder-Mead method [68] to find the optimum. To avoid local minima, we start optimizations from several points each time step (and select the result with the lowest value): four random points and the best guess based on the previous time step. The best guess is simply the same as the solution from the previous time step with t_{td} decremented by the timestep, except at the beginning of single support, when a default value is used. Even if the global optimum has a small basin of attraction, after several time steps, one of the random starts is very likely to start in it and find the global minimum. The controller is continuously updating its touchdown target, so it need not know where it will touch down at the beginning of the step so long as it figures it out quickly.

For the simple case of flat ground, but with some regions we are not allowed to step, there is still some problem structure that we can exploit to speed up the optimization and help find the global optimum step location as early in the step as possible. We implement the stepping requirement by modifying the h function in (3.7). We set $h(\mathbf{x}_f(t_j)) = \infty$ for $\mathbf{x}_f(t_j)$'s that correspond to stepping in illegal locations (where touchdown occurs at t_j) and $h(\mathbf{x}_f(t_j)) = 0$ for all other values of $\mathbf{x}_f(t_j)$. For any situation where the terrain imposes costs dependent on stepping location, we can expect an arbitrarily complicated dependence of the value on x_{td} and y_{td}, but still expect the same smooth behavior of $V(t_{td}|x_{td}, y_{td})$. Additionally, optimization will not work if it starts in an illegal region because it will not have a gradient (all points on the simplex will have the same value). We therefore start by randomly sampling only $\{x_{td}, y_{td}\}$ and checking to make sure it is a legal location. We resample new points until we find a legal stepping location. We then attempt to speed up the optimization and avoid local minima by picking the best (lowest value) t_{td} from a list of 19 choices. We use a list rather than even spacing so that we can have closer spacing for small t_{td}'s, where minima can have narrower basins of attraction. We then form a simplex with this point as one of the vertices and run the optimization. Choosing initial t_{td}'s in this way rather than randomly sampling does run the risk of systematically missing the global minimum because the nearest choice in the list is not the lowest of the list values. However, since we expect smooth behavior of $V(t_{td}|x_{td}, y_{td})$, this is unlikely. On the other hand, using this method often avoids local minima and therefore improves the chance of finding the global minimum early in the step.

The optimization method of coordination of policy coordination is more flexible than the parabolic approximation method, but it is also much slower. It takes an average of 586 microseconds on a 6 core 3.33 GHz Pentium i7 processor (compared to 6 microseconds for the parabolic approximation method). Fortunately, it is trivial to multithread; we can run the optimization from each of the five starting points in a different thread. This allows us to take advantage of multiple cores. In practice, multithreading decreased computation to an average of 325 microseconds,

which is reasonable as a component of a controller that we wish to run at 1 kHz.

In practice, we tend to get slightly worse robustness (can withstand slightly smaller pushes) with the optimization method. We believe this is because the parabolic approximation provides some smoothing to our value functions. They can be somewhat noisy because we stop optimizing our policies without them converging. Additionally, we use a course grid, which can result in weird effects near the border between where the value is infinite (recovery is impossible) and where it is finite.

3.4.3 Low-Level Control

The output of the high-level controller is the desired horizontal CoM acceleration, $\ddot{\mathbf{c}}_{x,des}$ and $\ddot{\mathbf{c}}_{y,des}$, as well as the desired swing foot acceleration $\ddot{\mathbf{p}}_{w,des}$. The objective of the low-level controller is to generate joint torques which will achieve these accelerations as well as enforce the constraints assumed by the high-level control, $\mathbf{c}_z = h$ and $\dot{\mathbf{l}} = 0$.

It is important to note that the high-level controller does not generate trajectories. Instead, it maps directly from system state to desired accelerations without maintaining any controller state. This allows it to react to perturbations and accumulated modeling error in real time, but it also means that we do not have desired positions or velocities, which precludes the use of traditional trajectory tracking techniques. In the place of trajectory tracking, we use a form of inverse dynamics to generate joint torques. The feedback gains of our controller are embedded in the gradients of the high-level DP policies.

We use PD controllers to enforce the $\mathbf{c}_z = h$ constraint and maintain a desired torso orientation, giving us $\ddot{\mathbf{c}}_{z,des}$ and the desired total moment. It is then straightforward to compute the desired total reaction force, $\mathbf{f}$, and torque, $\boldsymbol{\tau}$. During double support, we divide the total reaction force between the two feet while enforcing the CoP (3.39) and friction (3.40), (3.41) constraints

by minimizing

$$C = \frac{\mathbf{f}_{L,x}^2}{\mathbf{f}_{L,z}} + \frac{\mathbf{f}_{R,x}^2}{\mathbf{f}_{R,z}} + \frac{\mathbf{f}_{L,y}^2}{\mathbf{f}_{L,z}}\frac{\mathbf{f}_{R,y}^2}{\mathbf{f}_{R,z}} + a\left(\frac{\boldsymbol{\tau}_{L,x}^2}{\mathbf{f}_{L,z}} + \frac{\boldsymbol{\tau}_{R,x}^2}{\mathbf{f}_{R,z}} + \frac{\boldsymbol{\tau}_{L,y}^2}{\mathbf{f}_{L,z}}\frac{\boldsymbol{\tau}_{R,y}^2}{\mathbf{f}_{R,z}}\right). \tag{3.58}$$

This cost function has the useful property that it produces the same CoP offset for both feet, ensuring that there is as much margin as possible between the CoP and the edge of the foot.

We then use Dynamic Balance Force Control (DBFC) as presented in [90] to generate joint torques, $\boldsymbol{\tau}_j$. DBFC uses a weighted pseudo-inverse with regularization to solve

$$\begin{bmatrix} \mathbf{M}(\mathbf{q}) & -\mathbf{S} \\ \mathbf{J}(\mathbf{q}) & 0 \\ \epsilon_1 I & 0 \\ 0 & \epsilon_2 I \end{bmatrix} \begin{pmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\tau}_j \end{pmatrix} = \begin{pmatrix} N(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{J}(\mathbf{q})\hat{\mathbf{f}} \\ -\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} + \ddot{\hat{\mathbf{p}}} \\ 0 \\ 0 \end{pmatrix} \tag{3.59}$$

where $\mathbf{q}$ is a vector of generalized coordinates including 6 values specifying the position and orientation of the base and 12 joint angles, $\mathbf{M}(\mathbf{q})$ is the mass matrix, $\mathbf{J}(\mathbf{q})$ is the Jacobian of both feet, $\mathbf{S} = [0, \mathbf{I}]$ selects the actuated elements of $\mathbf{q}$, $\hat{\mathbf{f}} = [{\mathbf{f}_L}^\mathsf{T}, {\boldsymbol{\tau}_L}^\mathsf{T}, {\mathbf{f}_R}^\mathsf{T}, {\boldsymbol{\tau}_R}^\mathsf{T}]^\mathsf{T}$, and $\ddot{\hat{\mathbf{p}}} = [\ddot{\mathbf{p}}_L^\mathsf{T}, \ddot{\mathbf{p}}_R^\mathsf{T}]^\mathsf{T}$. The bottom two sets of equations provide regularization and $\epsilon_1 = 1.0 \times 10^{-5}$ and $\epsilon_2 = 1.0 \times 10^{-5}$ are small constants. It can be computed quickly, and the entire low-level controller requires only about 36 microseconds of computation time on a 6 core 3.33 GHz Pentium i7 processor.

Since we do not use PD joint torques in addition to the DBFC output, even small errors in the foot acceleration produced can accumulate over time. In order to more accurately match desired foot accelerations, we add an integrator on foot acceleration to our low level controller,

$$\ddot{\mathbf{p}}_{w,\mathrm{DBFC}} = \ddot{\mathbf{p}}_{w,\mathrm{HL}} + K_I \int (\ddot{\mathbf{p}}_{w,\mathrm{HL}} - \ddot{\mathbf{p}}_w)\, dt, \tag{3.60}$$

where $\ddot{\mathbf{p}}_{w,\mathrm{DBFC}}$ is the swing foot acceleration used by the low-level DBFC controller, $\ddot{\mathbf{p}}_{w,\mathrm{HL}}$ is the swing foot acceleration produced by our high-level policies, and K_I is the integral gain. The constant $K_I = 3.0$ was found by experimentation to work well. It gives a time constant shorter than a step, making it large enough to quickly correct for errors/disturbances but small enough

not to produce instability. The integrator is not necessary for stable walking, but it significantly improves the robustness to perturbations.

DBFC Modes

Dynamic Balance Force control, as described above, fails when the system is at or near a kinematic singularity (e.g. straight knees). Rather than simply avoiding these situations, we use a few modifications to the basic DBFC algorithm that each allow us to handle a specific situation.

During single support, we wish to keep the stance leg at a fixed length (fixed knee angle). The knee does have to be somewhat bent to allow for extension during double support (discussed in Section 3.3.1), but we do keep it mostly straight. Using the regular DBFC, we would use forward kinematics to get a desired CoM height, use feedback based on the actual CoM height to get a desired vertical foot force, and use DBFC to get joint torques. Essentially, we would be attempting to control knee angle by commanding a desired vertical force. Unfortunately, with the knee mostly straight, the Jacobian is nearly singular, so we have very little control over that vertical force. The result is that slight inaccuracies in our model or small perturbations can result in large errors. If what we really care about is knee angle, it is simpler to command it directly.

To directly command knee angle, we replace the regularizing equation for stance knee acceleration with a high gain PD controller that attempts to servo the stance knee to a fixed angle. It is contradictory to independently control stance knee angular acceleration and vertical force, so rather than commanding a desired force, we make the vertical force, $\mathbf{f}_{x,z}$, an additional variable. To do this, we must move the appropriate column of the Jacobian, $\mathbf{J}(\mathbf{q})$ from the right side of (3.59) to a new column in the large matrix on the left hand side.

We also have to handle two situations where the knee can become completely straight: 1) If the CoM moves too far forward during double support, the back knee can get completely straight. 2) If it attempts to step too far forward, the front knee can become completely straight. In either case, the result is a kinematic singularity and therefore a singular Jacobian, which causes DBFC

to fail. The policies are set up with appropriate constraints such that these situations do not occur during normal walking. However, if there is a large perturbation, they can happen, especially if the perturbation results in large torso rotations, which move the hip horizontally relative to the CoM.

If we find that the swing knee has become almost straight while attempting to reach its touch-down location, we rotate the three equations covering the translational acceleration of the swing foot (part of the second set of equations in (3.59)). We rotate the three equations from $\{x, y, z\}$ coordinates to coordinate system with one of the basis vectors pointing from the swing foot to the hip. We can then drop the equation for acceleration in that direction, leaving acceleration in the direction we can not control unspecified. We then directly control the swing knee by replacing a regularization equation with a PD controller on knee acceleration that servos it to a constant slightly bent angle (similar to what we did for the stance knee in single support). This way, the foot will touch down at the desired location as soon as the hip moves far enough forward that it can reach.

We use a nearly identical method to recover if the back leg becomes straight during double support. In that case, however, rather than servoing the knee to a fixed angle, we bend it until we are far enough from the kinematic singularity that normal control can resume.

3.5 Capabilities

To be useful, a walking system must be able to do more than walk in a straight line. It must be able to start from rest, walk at varrying speeds, turn, and stop. It also must be able to handle walking on complex terrain in addition to on flat ground. Our controller can avoid specified regions where it is not allowed to step. It can also handle slopes and step changes in ground height, but because we have only tested these sorts of terrain when the controller was unaware of them, we cover them in the following section on robustness.

3.5.1 Speed

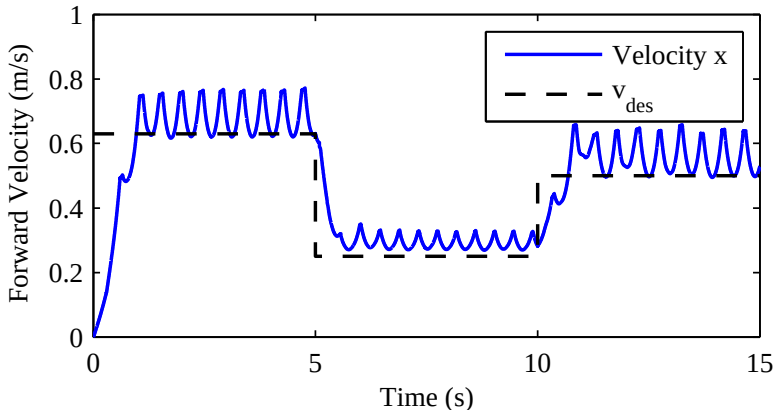

Figure 3.9: Forward speed as the sagittal stance policy is changed. Starts from rest with $v_{\text{des}} = 0.63$, switches to $v_{\text{des}} = 0.25$ after 5.0 seconds, and to $v_{\text{des}} = 0.5$ after 10.0 seconds.

Walking speed can be changed by switching the sagittal stance policy to one computed with a different v_{des}. The policies are global, so no transition is necessary, and the policies can be switched at any point during the step. Similarly, the system can start from rest and achieve steady state walking without switching policies. Fig. 3.9 shows how the velocity varies after changing policies.

If the system is walking slowly enough, it can stop simply by switching from a walking high level controller to a standing high level controller and continuing to run the same low level controller. If the switch is done during double support, it will simly stop, but if it is done during single support, it will stomp its swing foot as it tries to put weight on a foot that is not on the ground (without the system failing). However, if it is walking forward at any significant speed, it is necessary to slow down before stopping. To slow down, we switch to a sagittal policy generated with $v_{\text{des}} = 0.0$. This quickly brings the system to a stop and remains in an infinitely long double support phase, as shown in Fig. 3.10. The normal coordination procedure results in a permanent double support phase, though it does stand with its CoM slightly off center because the coronal policy thinks it is going to step again. To achieve balanced standing, it is necessary

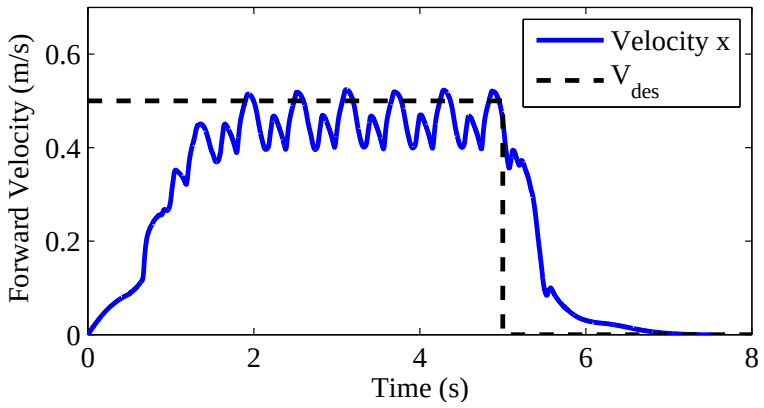

Figure 3.10: Forward speed as the system starts from rest, walks with $v_{\text{des}} = 0.5$, and stops by switching to a policy generated with $v_{\text{des}} = 0.0$. We switch to the stopping policy after 5.0 seconds, after which it takes about 1.5 steps to stop. During steady walking, the large humps in the speed occur during single support and the small humps during double support.

to switch to a standing controller once the system has come to a stop.

3.5.2 Turning

The walking controller is designed to always face and walk forwards. Therefore, you can turn, by simply changing which direction "forward" is. We rotate the coordinate frame that the robot state is measured in while still having the controller attempt to walk in the positive x direction. Practically this means rotating all vectors in the robot state (positions, velocities, and accelerations) and adding to the yaw euler angle, then reconstructing a new quaternian from the modified euler angles. Using this method, we can turn at up to about 1.5 radians/second while walking forward with $v_{\text{des}} = 0.5\text{m/s}$. If the robot were a point and it truly was walking at 0.5 m/s, this would correspond to walking in a circle of about 2/3 m diameter.

The system retains much of its robustness while turning. It can withstand a push of 38 Ns to the inside of its turn as compared to a 41 Ns push of the same direction and timing when walking forward. It can withstand a push of 21 Ns to the outside of its turn as compared to a 33 Ns push

of the same direction and timing when walking forward.

We also attempted to improve turning performance by adding centripetal acceleration to the CoM and the swing foot as offsets to what the normal controller outputs. Neither modification proved beneficial. In the case of adding centripetal acceleration to the CoM, we found decreased robustness, but for adding acceleration to the swing foot, we found that the system failed to walk entirely. It is possible that centripetal acceleration will play a bigger part and therefore these modifications will be more beneficial when walking at faster speeds.

We were able to get slightly improved robustness by rotating the swing foot to the average desired orientation during the next step rather than servoing it to the current heading. This way the stance foot matches the heading at the middle of the step rather than the beginning. This modification increased the size of the push that could be withstood from 21 Ns to 24 Ns for pushes to the outside of the turn without having any affect on pushes to the inside of the turn.

3.5.3 Terrain

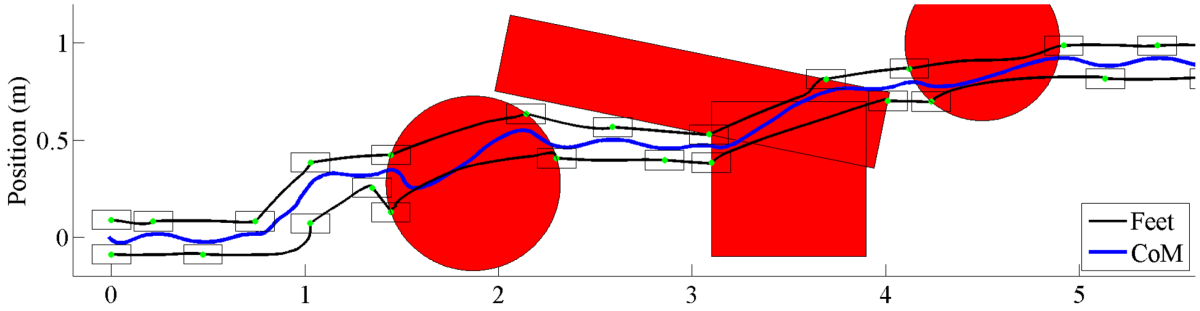

Figure 3.11: The footstep pattern as the system starts from rest, gets pushed (a 30Ns push to the system's left during the 4th step), and avoids obstacles (red regions). The red regions represent areas where the center of the foot (green dots) may not be placed, though the foot may overlap with the red region.

We are able to avoid stepping in regions that are specified as regions we are not allowed to step in. Switching from the coordination method based on parabolic approximations described

in Section 3.4.2 to the coordination method based on a generic optimization described in Section 3.4.2 gives us the flexibility to add an arbitrary cost function for where we are stepping. For the case of avoiding specified illegal regions, we add an infinite cost for stepping in the illegal regions. Fig. 3.11 shows the footstep pattern as the controller avoids obstacles. The largest gap that the system can step over is 48 cm measured, which corresponds to a step of 68 cm: the 48 cm gap plus the 20 cm length of the foot. This same method can also be used if we have a general terrain cost map. Such a cost map could potentially be generated from local features of the terrain, and Inverse Optimal Control could potentially be used to find the weights for these features [106].

3.6 Robustness

An important characteristic of any controller is its ability to reject perturbations. In particular, the size of the largest disturbance that does not cause the system to fail is a useful metric for systems where failure is well defined. One practical difficulty with using this as a metric of performance for walking is that there are many different types of disturbances. We discuss here the robustness of our controller to several types of disturbance.

3.6.1 Pushes

A major type of disturbance experienced by walking systems are pushes. We apply pushes to the torso center of mass, which is about 1.25 m high during walking (30 cm above the system CoM). Fig. 3.12 shows the effect of push angle and timing on the maximum survivable perturbation. Force perturbations lasting 0.1 seconds are administered to the torso CoM at various angles while the system is walking with a nominal speed of 0.5 m/s. Data is shown in Fig. 3.12 for perturbations beginning at increments of 0.1 seconds after the left foot lifts off. These pushes are shorter than most of the system dynamics, so we consider them to be essentially impulsive. We

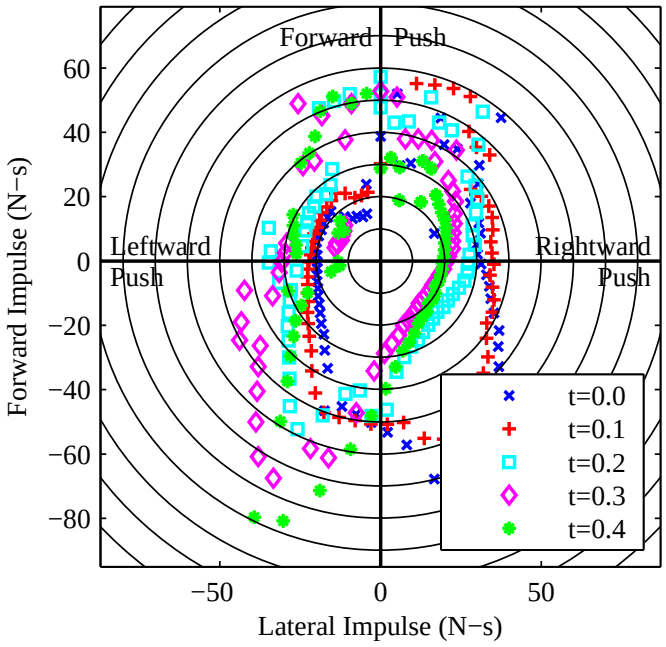

Figure 3.12: Polar plot of the maximum survivable 0.1 second push for our walking simulation as a function of angle and timeing. Data is shown for perturbations occuring at various times after left foot lift off. A point represents the maximum survivable perturbation in a given direction. Concentric circles are in increments of 10 Newton-seconds.

therefore measure their magnitude in impulse, Ns.

We also tested our system with more prolonged pushes lasting 3.0 seconds. Since the pushes last for multiple steps, the exact time within a step at which it begins is not important. Fig. 3.13 shows these results. Note that for the longer pushes, we measure the magnitude in force, N.

3.6.2 Slips

Our controller can walk on surfaces with a coefficient of static friction $\mu_s \geq 0.35$. Once slipping begins, we model friction as having a coefficient of kinetic friction, $\mu_k < \mu_s$. The exact value of this coefficient has almost no effect on whether slipping results in a fall: with $\mu_k = 0.8\mu_s$, we found that the minimum μ_s was 0.35, but with $\mu_k = 0.3\mu_s$, we found that the minimum μ_s was

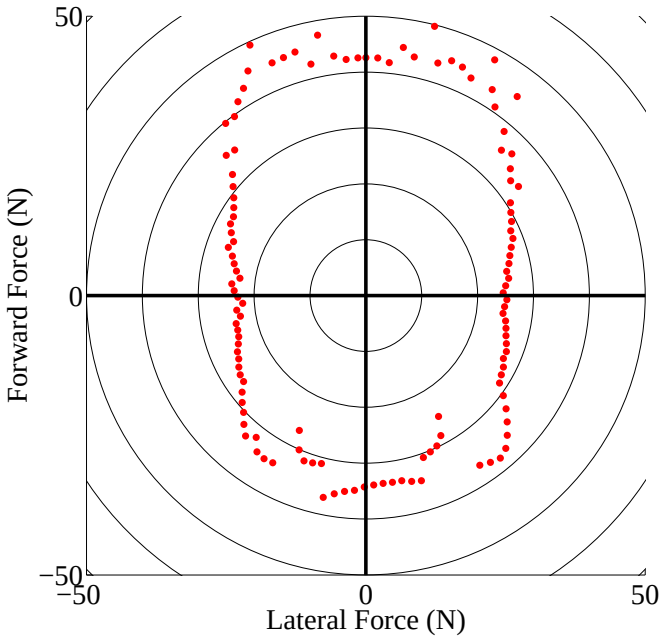

Figure 3.13: Polar plot of the maximum survivable 3.0 second push for our walking simulation as a function of angle. Data is shown for perturbations occuring at various times after left foot lift off. A point represents the maximum survivable perturbation in a given direction. Concentric circles are in increments of 10 Newton-seconds.

0.36.

It can tolerate small regions of even lower friction, but only very small regions. (We simulate changes in fricion by making μ dependent on the location of the center of the foot, without averaging over the entire foot.) Counter-intuitively, it can tolerate larger regions of low friction if the friction is lower: 2.5 cm for $\mu_s = 0.01$, 1.5 cm for $\mu_s = 0.1$, and less than 1 cm for $\mu_s = 0.15$. The reason for this unexpected behavior is that the failure mode is different from that found in human walking. When the foot is first put down in a low-friction area, it slides slightly forward during double support, which generally does not lead to failure. Then, during single support, it slides backward rapidly, which does result in failure. Once the foot starts slipping backward, it does not recover even if it gets onto a higher friction area. However, if the low-friction area is

very low friction, it will slide forward farther during double support, and be more likely to slip onto a higher-friction area before slipping in the other direction during single support.

It is likely that special sagittal and coronal policies generate with appropriate friction constraints would allow for successful very-low friction walking. Alternately, a simple traction-control controller decreasing the ground reaction force once slipping was detected would likely help somewhat.

3.6.3 Trips

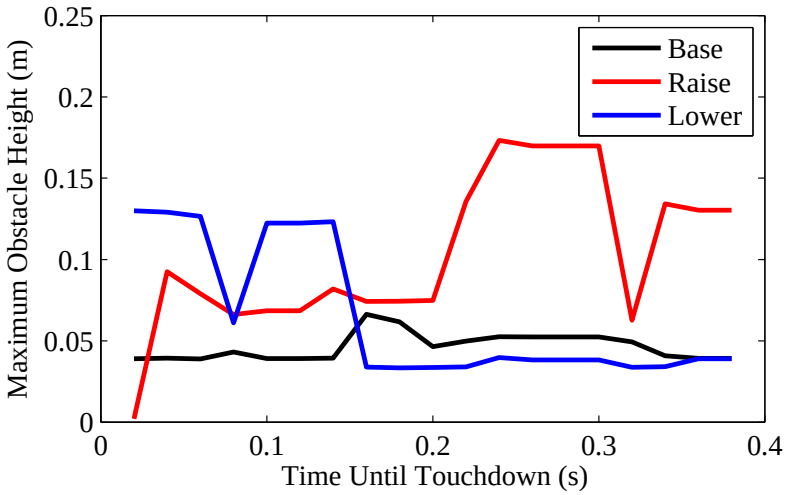

Figure 3.14: The largest tripping obstacle that our controller can handle as a function of when it is contacted during the step. Results are shown for the baseline controller, an explicit raising strategy, and an explicit lowering strategy.

We also tested our controller's response to tripping obstacles. The obstacles were wide enough to obstruct one, but not both, feet, of varying heights, and only 1.0 cm long (the foot must be lifted, moved forward one foot length plus 1.0 cm, then lowered). The controller does not know about the obstacles ahead of time - they are detected by the anamalous foot force, but we do assume it knows when it has raised its foot high enough to clear the obstacle. We also assume that the front face of the obstacle is frictionless, meaning the toe can slide up or down it

easily.

Fig. 3.14 shows the maximum height of tripping obstacles our controller can handle as a function of when in the step they are contacted. It shows result for the baseline walking strategy as described in Section 3.4. It also shows results for a simple explicit raising strategy and a simple explicit lowering strategy.

For the raising strategy, when the obstacle is detected (via anamalous foot forces), the high level controller immediately overrides the output of its Swing-X and Swing-Z policies. Instead, it commands zero acceleration in the x direction and a large upwards acceleration (25 m/s/s) in the z direction. Once the foot has cleared the obstacle vertically, the Swing-X policy is used as normal, but the Swing-Z policy acceleration is ignored in favor of servoing the foot to 0.5 cm above the obstacle. Once the foot has passed the obstacle horizontally, control returns completely to normal.

For the lowering strategy, we again override the Swing-X and Swing-Z policies, but use a large downwards acceleration (-25 m/s/s) in the z direction. Once the foot reaches the ground, we begin a normal double support phase. At the beginning of the lowered foot's next step, it uses the raising strategy to clear the obstacle.

Based on the results shown in Fig. 3.13, the lowering strategy works better towards the end of a step, and the raising strategy works better earlier in the step. We should therefore use the lowering strategy if an obstacle is detected with $t_{td} < 0.15\text{s}$ and use the raising strategy otherwise. This is qualitatively consistent with strategies observed in human walking [40] [59].

3.6.4 Steps Up/Down

We tested the system's response to unexpected changes in ground height to find the largest unexpected step up and step down it could handle. In both cases (up and down), we found that the largest step it could handle ws about 7 cm (though for stepping up, it depended somewhat on when in the swing phase it encountered the change in ground height).

When stepping down, the controller first becomes aware of the step when it does not make contact with the ground when it expects to. At this point, it moves the swing foot rapidly downward (It attempts to servo to 1.0 m/s downward velocity) until it hits the ground. It must find the ground so quickly because otherwise the CoM will move too far ahead of the foot, and it will fall forwards. Once the foot reaches the ground the normal controller resumes. The other foot assumes its starting position is the ground height (for purposes of the Swing-Z policy) until it passes the stance foot at the lower elevation, after which it switches to using the new ground height.

When stepping up, because we wish to handle steps that are larger than our normal ground clearance, we use the same raising strategy as was used for the tripping obstacles described in Section 3.6.3. Double support begins when it unexpectedly lands on the ground on the far side of the apparent obstacle. When stepping up, the swing foot immediately uses the higher ground elevation for its Swing-Z policy because too much ground clearance is safer than too little.

3.6.5 Slopes

The steepest constant slope that our controller could walk up for a large number of steps had a rise of 7.5 cm per horizontal meter (4.3 degrees from horizontal). We tested slopes in simulation only changing the height of the ground; we did not change the surface normal. This makes it a little bit like walking up steps, but without having to worry about where you put your feet. Changing the surface normal is problematic because our simulation is implemented without true rigid-body feet. It has point feet, and the ankle joints apply torque directly to the ground during stance. For walking on flat ground, this is nearly dynamically identical to having true rigid-body feet, but it makes changing the foot-ground interaction difficult.

Changing the surface normal should only affect the friction cone constraints, which are rarely active during normal walking. The ZMP constraints, which play a much bigger role in walking, are barely affected by non horizontal feet. The only effect is that the projection of the feet into

the horizontal plane is slightly smaller. On a real robot, non-vertical surface normals are more of a problem for state-estimation than from a dynamics perspective, but we use perfect state knowledge in our simulation.

3.7 Upper Body Rotation

The walking controller described above always attempts to maintain the torso in a vertical posture. Our policies are generated under the assumption that there will be no change in system angular momentum, $\dot{\mathbf{l}} = 0$. Most of the angular momentum in the system is in the torso, so we approximately obey this constraint by maintaining the torso at a fixed orientation. However, angular momentum and torso rotation can be used to aid in balance.

We have identified a mode of motion that rotates the upper body and translates the CoM without moving the CoP. We have also identified a point, which we refer to as the augmented CoM, which is unaffected by these motions, but follows LIPM-like dynamics. By performing a simple change of variables, we can use this point (instead of the CoM) with any LIPM-based control algorithm (including the DP policies described above). The key is that the identified mode functions as an additional source of control authority that looks mathematically similar to moving the CoP. Our DP policies can use this rotational mode of control the same way that they use variations of the CoP without any need for them to understand the difference. We generate versions of both the sagittal and coronal policies that replace the old ZMP constraint with a looser constraint that allows for the added effect of both rotating the torso and moving the ZMP. At run time, a lower level controller determines how much of the command to achieve by actually moving the CoP and how much to achieve by rotating the torso. In addition to giving us greater control authority, this modification allows us to predict the result of undesired torso rotations on CoM motion and compensate for it.

We found that this modification greatly improves robustness to pushes when walking with fixed footstep locations, but actually reduces robustness when walking normally, where the con-

troller is free to choose the footstep locations. During normal walking, most of the robustness to pushes comes from the ability to greatly modify the footstep location and take large steps in the direction the system was pushed. Balancing by torso rotation involves rotating the upper body so that the head moves in the direction of the push and the hip moves away, which prevents the system from taking a large step in that direction. The gains from rotating the torso are smaller than the gains from stepping farther in that direction, so our torso rotation strategy results in a net loss of robustness for normal walking. This result is not entirely surprising given human experience: humans will lean their bodies significantly when trying to walk on a balance beam or on stepping stones, but if pushed when walking on flat ground will keep their torso mostly upright and recover by stepping in the direction of the push.

3.7.1 System Model

We wish to modify the LIPM dynamics (3.21) and (3.22) to avoid the $\dot{\mathbf{l}} = 0$ constraint and model upper body rotation. It is often assumed that the majority of the mass is in the torso, and that we can approximately enforce $\dot{\mathbf{L}} = 0$ by maintaining a fixed torso orientation. However, it may not be possible to perfectly control the torso orientation. Undesired torso rotations may be caused by error in the model of the robot or the environment as well as by external pushes. These rotations can be particularly large in torque-controlled systems or systems with relatively low position gains. Any angular acceleration of the torso (or more generally, any change in angular momentum) without changing the COM acceleration will require some foot torque, which means some change in the COP. Conversely, if the COP of the robot is matched to that of a LIPM model, but there is a change in angular momentum, then the COM motion of the robot will not match that predicted by the LIPM model.

In order to model the interaction between upper body rotation and COM motion, we model the upper body as a single rigid flywheel, which rotates about the system center of mass. This model is known as the Linear Inverted Pendulum plus Flywheel Model (LIPFM) [77] with dy-

namics given by

$$\ddot{x} = \frac{g}{h}(x - z) - \frac{I\ddot{\theta}}{mh} \tag{3.61}$$

where I is the moment of inertia of the upper body about the system COM and θ is the angle of the upper body relative to the nominal. The upper body angle will also be subject to the kinematic constraint $\theta_{\min} \leq \theta \leq \theta_{\max}$. The LIPFM does not fully account for angular momentum, but many humanoid systems have a large fraction of their mass in the upper body (about 2/3 of the total mass in our system), so a large fraction of the angular momentum will be in the upper body. Additionally, while the lower body motion is highly constrained by walking, we are free to rotate the upper body as desired to aid in control.

We now perform a change of variables so that upper body rotation does not require any change in the COP. Additionally, it will put the LIPFM dynamics in the same form as the LIPM dynamics, allowing us to leverage the existing technology for controlling the LIPM.

Change of Variables

We start with the forward (rotational and translational) dynamics for the x-z plane,

$$\begin{bmatrix} m\ddot{x} \\ I\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -h & mg \end{bmatrix} \begin{bmatrix} F_x \\ -z \end{bmatrix} + mg \begin{bmatrix} 0 \\ x \end{bmatrix}, \tag{3.62}$$

where F_x is the horizontal ground reaction force. We then multiply by a change of variables matrix,

$$\mathbf{D} \begin{bmatrix} m\ddot{x} \\ I\ddot{\theta} \end{bmatrix} = \mathbf{D} \begin{bmatrix} 1 & 0 \\ -h & mg \end{bmatrix} \begin{bmatrix} F_x \\ -z \end{bmatrix} + \mathbf{D}mg \begin{bmatrix} 0 \\ x \end{bmatrix}. \tag{3.63}$$

where

$$\mathbf{D} = \begin{bmatrix} 1 & \frac{1}{h} \\ 0 & 1 \end{bmatrix}. \tag{3.64}$$

We now define the augmented COM acceleration, $\ddot{\tilde{x}} \equiv \ddot{x} + \frac{I\ddot{\theta}}{mh}$, then combine and simplify to get

$$\begin{bmatrix} m\ddot{\tilde{x}} \\ I\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & \frac{mg}{h} \\ -h & mg \end{bmatrix} \begin{bmatrix} F_x \\ -z \end{bmatrix} + \begin{bmatrix} \frac{mg}{h}x \\ mgx \end{bmatrix}. \tag{3.65}$$

Double integrating $\ddot{\tilde{x}}$ gives us the augmented COM position, $\tilde{x} = x + \frac{I\theta}{mh}$, which we substitute into (3.65) to get

$$\begin{bmatrix} m\ddot{\tilde{x}} \\ I\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & \frac{mg}{h} \\ -h & mg \end{bmatrix} \begin{bmatrix} F_x \\ -z \end{bmatrix} + \begin{bmatrix} \frac{mg}{h}\tilde{x} + \frac{gI\theta}{h^2} \\ mg\tilde{x} + \frac{gI\theta}{h} \end{bmatrix}. \tag{3.66}$$

Multiplying by the inverse of the 2x2 matrix and solving for the controls, F_x and z, gives the inverse dynamics,

$$\begin{bmatrix} F_x \\ -z \end{bmatrix} = \begin{bmatrix} 1 & \frac{-1}{h} \\ \frac{h}{mg} & 0 \end{bmatrix} \begin{bmatrix} m\ddot{\tilde{x}} \\ I\ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \tilde{x} + \frac{I\theta}{mh} \end{bmatrix}. \tag{3.67}$$

The benefit of the change of variables is that we now have a 0 in the lower right of the 2x2 matrix. This 0 means that changes in angular momentum without modifying $\ddot{\tilde{x}}$ have no effect on the COP.

If we define an augmented COP,

$$\tilde{z} = z + \frac{I\theta}{mh}, \tag{3.68}$$

we are able to put the LIPFM dynamics,

$$\ddot{\tilde{x}} = \frac{g}{h}(\tilde{x} - \tilde{z}), \tag{3.69}$$

in the same form as the original LIPM dynamics in (3.21).

Since (3.21) and (3.69) have the same form, control methods that work on the LIPM dynamics can also be used on the augmented LIPM dynamics, but by working in the space of $\tilde{x} = x + I\theta/mh$ and its derivatives instead of the space of x and its derivatives. Additionally, controllers will now be requesting a $\tilde{z}$, and (3.68) must be used to find z before applying control to the

system. The change of variables eliminated a $\ddot{\theta}$ term (by hiding it within $\dddot{\tilde{x}}$), but created a θ term. The angular position term is preferable from a modeling perspective because it changes more slowly and can be more easily measured.

Orientation Control

Since θ can be controlled independently of $\tilde{x}$, the θ term in $\tilde{z}$ can be used for more than canceling the effect of small undesired deviations from the desired posture. It can also be actively controlled to something other than 0 and treated as an additional control for the augmented COM. Torso orientation, θ, has the same effect on the system as does z, so a LIPM controller need not differentiate between them; instead, it can simply request a $\tilde{z}$. In addition, the controller can operate under a more lenient constraint (but with the same form) than the ordinary COP constraint, $|z| \leq z_{\max}$. For the LIPFM dynamics, the constraint is

$$|\tilde{z}| \leq z_{\max} + \frac{I_{\text{eff}}\theta_{\max}}{mh} \tag{3.70}$$

where $\theta_{\max}$ is the maximum allowable lean angle. Use of the more lenient constraint (and the torso rotation necessary to produce the larger requested $\tilde{z}$) makes it possible to plan more aggressive maneuvers that are impossible with a fixed-orientation torso. It also allows for improved feedback in response to unexpected disturbances.

There is also an additional constraint arising from the fact that while contact forces such as z can be changed quickly, torso angle will have a maximum velocity or acceleration. Such constraints become an issue if the requested $\tilde{z}$ rapidly changes. In situations where $\tilde{z}$ changes gradually or on systems where the internal dynamics (dependent on actuators) are much faster than the LIPM dynamics (dependent on g/h), this additional constraint can reasonably be ignored.

There are several options for controlling θ depending on the context. If the increased control authority is being used in a planning context for generating trajectories with large accelerations, then the planner will produce a time varying $\tilde{z}(t)$. In this case, optimization (for example, a

quadratic program) can be used to determine a time varying $\theta(t)$ which satisfies (3.70) as well as whatever hardware constraints exist on $\dot{\theta}$ or $\ddot{\theta}$. A cost on $\theta(t)$ can be added to the optimization to bias it towards upright postures that do not utilize torso rotation wherever possible. However, if (as we use it) the increased control authority is being used for improved responses to unexpected disturbances, then no expected $\tilde{z}(t)$ will be available (because the disturbances are unexpected).

In this case, we use a fixed relationship between $\tilde{z}$ and the desired torso orientation, θ_{des}, then use a high gain PD servo to track that desired angle. Immediately following a large disturbance, it will not be possible to obtain the requested $\tilde{z}$ until the torso has had time to slew, resulting in (3.69) being inaccurate. However, slewing happens relatively quickly, and rotating the torso at the maximum speed is the best thing to do in this situation. Therefore, the quality of our control does not suffer from neglecting this effect.

We have used a piecewise linear relationship with a dead zone to determine θ_{des} from $\tilde{z}$.

$$\theta_{des}(\tilde{z}) = \begin{cases} \text{if } \tilde{z} > z_{\max}/2 & : \quad (\tilde{z} - z_{\max}/2)mh/I_{\text{eff}} \\ \text{if } \tilde{z} < -z_{\max}/2 & : \quad (\tilde{z} + z_{\max}/2)mh/I_{\text{eff}} \\ \text{else} & : \quad 0 \end{cases} \tag{3.71}$$

The purpose of the dead zone is to prevent the upper body from rotating unnecessarily, which would both look unnatural and run the risk of exciting unmodeled dynamic modes. For ordinary small values of $\tilde{z}$, θ_{des} will remain at 0, and torso rotation will only be utilized when it becomes necessary to achieve large $\tilde{z}$'s. In simulation, the exact shape of this relationship has very little effect on the system robustness. However, smoother functions perform better on real hardware due to decreased jerk and less excitation of un-modeled higher-order dynamics.

3.7.2 Use in Practice

We use the method described here to add torso rotation to our controller without having to add dimensions to or otherwise complicate our DP policies. Many other existing systems that use the LIPM dynamics could use a similar modification. We use it for feedback control where we have a

system position and velocity and wish to generate desired accelerations and torques. However, it can also be used with pattern generation methods based on the LIPM dynamics. We first explain how it could be applied to open-loop pattern generation and then torque feedback control.

Pattern Generation

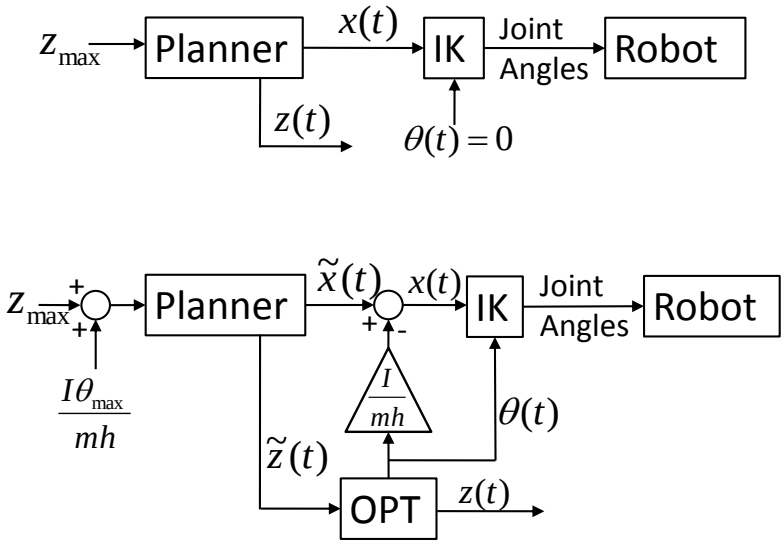

Figure 3.15: A flow chart of a simple generic system for humanoid walking pattern generation using standard LIPM dynamics (top) and a flow chart for that same system modified to use the LIPFM dynamics (bottom).

Fig. 3.15 shows how you would use this method for augmenting a simple generic motion planning system for a position controlled robot. The upper flow chart depicts the unmodified system, which uses a planner (e.g. preview control) to generate COM and COP trajectories. Then, an inverse kinematics solver uses the COM trajectory and a fixed upper body orientation, $\theta(t) = 0$, to generate joint angle trajectories, which are used to drive the robot.

The lower flow chart shows how this system would be modified to use the LIPFM dynamics as described in this paper. The same planner and inverse kinematics solver can be used without

modification, but the interaction between the two is modified. First, the planner is given the more lenient constraint given by (3.70). Now, the output of the planner is interpreted as $\tilde{x}(t)$ and $\tilde{z}(t)$. A simple trajectory optimization (denoted "OPT" in Fig. 3.15) is then used to break $\tilde{z}(t)$ into $z(t)$ and $\theta(t)$ such that (3.70) and any hardware constraints on θ or its derivatives are satisfied at all times. The $\theta(t)$ trajectory is then used in place of the fixed torso orientation given to the inverse kinematics solver. It is also used to compute $x(t)$ from $\tilde{x}(t)$. The inverse kinematics solver then determines joint angle trajectories for driving the robot.

The change of variables allows the planner to look at the extra capability given by upper body rotation as simply a larger virtual foot. Then, we post-process its output to achieve the extra control authority it requests through upper body rotation.

Torque Control

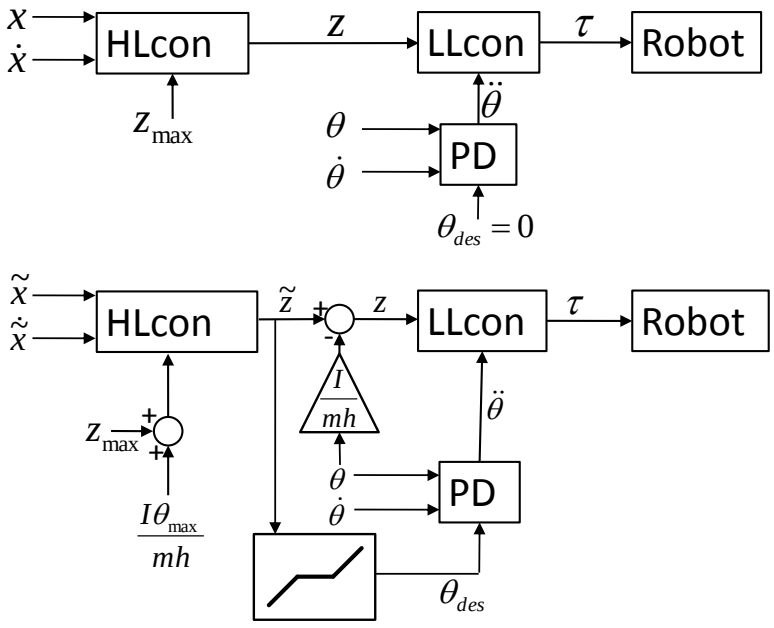

Figure 3.16: A flow chart of a simple generic system for torque control of a humanoid robot using standard LIPM dynamics (top) and a flow chart for that same system modified to use the LIPFM dynamics (bottom).

We use an analogous process to modify a LIPM-based controller for a torque controlled robot, as shown in Fig. 3.16. The upper flow chart depicts a generic LIPM-based controller. A high-level controller (coordinated DP policies in our case), denoted "HLcon" in Fig. 3.16, uses the current COM state, x and $\dot{x}$ to determine the desired COP, z. A PD servo is used to keep the torso angle at a fixed orientation, and a low-level controller (DBFC in our case), denoted "LLcon" in fig. Fig. 3.16, generates the joint torques necessary to achieve the desired z and $\ddot{\theta}$.

The lower flow chart shows how this system was modified to use the LIPFM dynamics. Again, the primary algorithms, the high-level and low-level controllers, remain unchanged. As for pattern generation, we modify the COP constraint according to (3.70). The inputs to the high level controller are now the augmented COM state, $\tilde{x}$ and $\dot{\tilde{x}}$, and the output is now interpreted as $\tilde{z}$. We use θ and $\tilde{z}$ to determine z by rearranging (3.68), which is passed to the low-level controller. The bottom box represents the function given in (3.71), and is used to pick a θ_{des} that makes $\tilde{z}$ achievable.

3.7.3 Results

Fig. 3.17 shows results for a robustness to pushes experiment similar to the one depicted in Fig. 3.12 but with the torso rotation control modification described here and fixed footstep locations. We used sagittal and coronal policies computed with the old ZMP constraint replaced by the weaker constraint given by (3.70). The system shows considerably less robustness here than in Fig. 3.12 because it is walking with fixed footstep locations. The fact that the recoverable regions are nearly rectangular can be interpreted as indicating that the sagittal and coronal systems really are mostly decoupled. The improvement is larger in the coronal direction because we start to run into kinematic constraints in the sagittal direction even when walking with fixed footstep locations; the hip moves too far for away from the desired touchdown location and the swing foot cannot reach it.

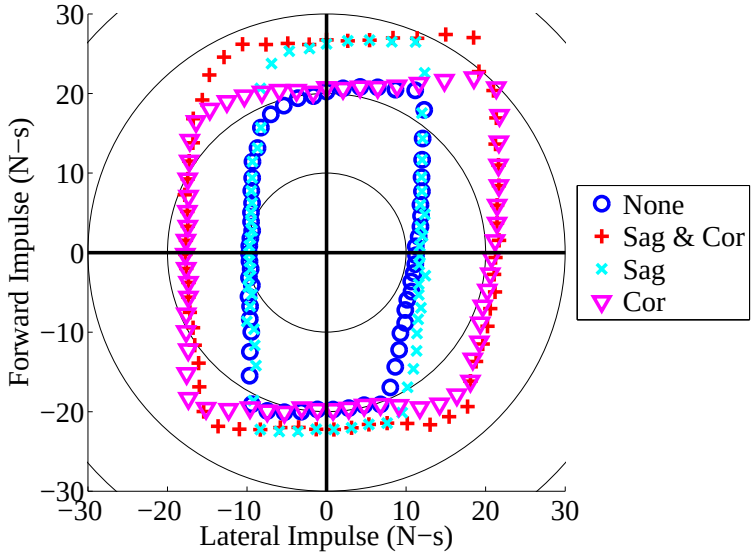

Figure 3.17: Polar plot of the maximum survivable perturbation of our walking simulation as a function of push angle. A point represents the maximum survivable perturbation in a given direction. Pushes occur midway through right single support. Concentric circles are in increments of 10 Newton-seconds. Data is shown for the unmodified system, the system modified in both the sagittal and coronal planes, only the sagittal, and only the coronal plane.

3.8 Conclusion

By avoiding pre-planned trajectories, our controller can react immediately to unexpected perturbations. It selects a new time and location of touchdown at every control cycle (currently 400 Hz). In the sense that it avoids tracking a planned trajectory, our method resembles MPC. However, to compute the control in real time, MPC approaches typically use linear dynamics and fixes the step timing. We, on the other hand, avoid any requirement of linearity by performing our optimization iteratively offline. However, offline computation forces us to consider all possible states, which subjects us to the "Curse of Dimensionality" and constrains the dimensionality of our dynamics. Our coordination scheme does let us handle higher-dimensional systems by

breaking them into lower-dimensional subsystems, but it requires that the full system be an ICS. This, in turn, requires that both the dynamics and cost function be capable of being decoupled (except at transitions).

One major advantage of this control framework is its flexibility: Coordination is not dependent on any particular dynamic model, cost function, or constraints. While we must take care to maintain low dimensionality and the ability to decouple our system, we remain free from any further restrictions - for instance, linearity - on the dynamics or cost function. In fact, many of the motion's characteristics can be controlled by adjusting the cost functions, and the variables considered can be changed by combining or adding additional subsystems.

We have defined an Instantaneously Coupled System and demonstrated the equivalence of coordinated policies that are optimal for the subsystems to a single controller that is optimal for the full ICS. We apply this theory to walking and present a walking controller for a simulated biped. Our controller optimizes center of mass motion as well as footstep timing and location, and it can react in real time to perturbations and accumulated modeling error. We also present standing balance experiments on a force-controlled humanoid robot.

Chapter 4

Robotic Walking

4.1 Introduction

Our simulated walking was accomplished on a simulation based on the Sarcos Primus hydraulic robot, and it was designed with the eventual goal of walking on the real robot in mind. However, the simulation controller does not perform well unaltered on the actual robot. The difference in performance is largely due to an inaccurate dynamic model of the robot, inaccurate sensor calibration, and inaccurate state estimation. In this chapter, we will discuss the robot itself, the changes we made to the simulation controller in response to the difficulties associated with working on real hardware, and some preliminary results for walking in place.

4.2 Robot Description

All of our hardware experiments are done with the Sarcos Primus humanoid robot [16] [51] shown in Figure 4.1. It is a hydraulicly powered, force controlled humanoid robot massing about 95 kg and about 1.7 m tall. When standing straight, its CoM is about 1.0 m above the ground. Its feet are flat rectangular aluminum plates (about 0.25 cm long and 0.12 cm wide) with damping rubber glued to the bottom.

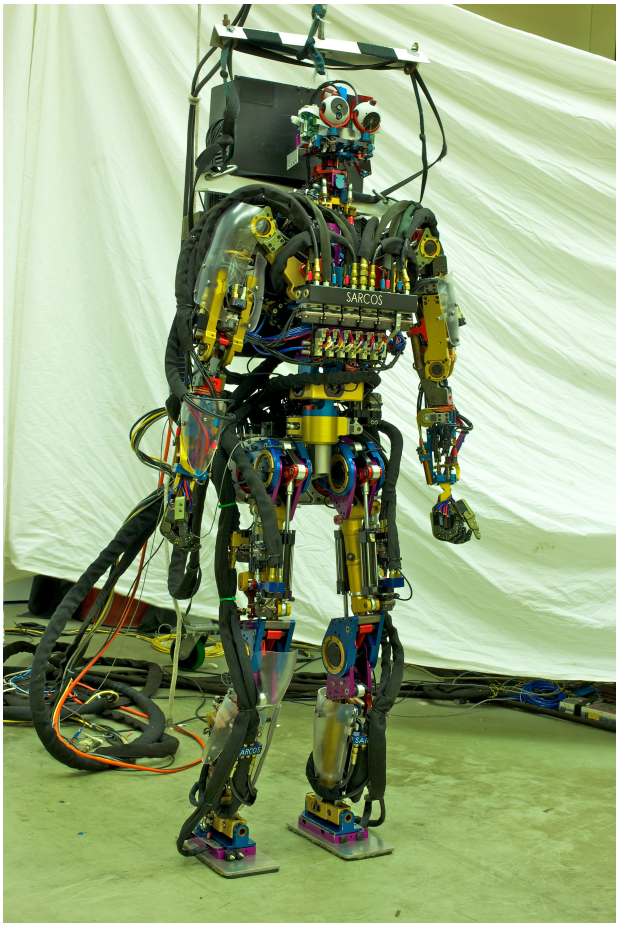

Figure 4.1: The Sarcos Primus humanoid force-controlled robot.

4.2.1 Mechanical

The primary actuation is a 3000 psi hydraulic system powered by an offboard pump. It has 34 hydraulicly auctuated joints: 7 in each leg, 7 in each arm, 3 in the torso, and 3 in the neck. We only actively control 12 of those joints for the work in this thesis: 2 at each hip, 1 at each knee, 2 at each ankle, and a shank rotation joint with its axis parallel to the shank. The remaining joints are servoed to fixed angles. Using these joints gives us the ability to position our feet with the full 6 degrees of freedom relative to the torso without having any redundancy. All other joints are servoed with high gains to fixed angles. The robot also has electrically actuated pan and tilt for each eye, and 12 pneumatically actuated joints in the hands, which we do not use in this thesis.

4.2.2 Sensing

Every joint has position sensing by means of an analog potentiometer. We also have force sensing at every joint. For the majority of joints, the force sensing is done by strain gauges glued to a rigid link in the hydraulic drive chain. However, for a few joints, we use commercial load cells to sense force. Each foot has a 6-axis force/torque sensor. We also have an IMU attached to the pelvis near the right hip. There are cameras in the eyes, but we do not use them in this work.

4.2.3 Computation

Our primary computation is done on an offboard computer with an 8-core Pentium processor. It receives sensor information and transmits joint force commands to the robot at 400 Hz. In the robot's backpack are small micro-computers (one for each joint) that perform the high speed force control on individual joints at 5000 Hz. They also have small position and velocity gains that help to smooth control on the short time scale between new commands from the offboard computer.

4.3 Major Controller Changes

We can use the same controller for standing both in simulation and on the real robot. Unfortunately, for more complex activities, such as walking, the nonidealities of the real hardware become more important. While the walking controller described in Section 3.4 was designed with an eye towards coping with an inaccurate dynamics model, there are several nonidealities on the real robot that make control difficult, some of which need to be specifically addressed. The robot has inaccurate state estimation, which can among other things, make it difficult to determine when the foot will reach the ground. There are also significant delays in the dynamics, caused in part by state estimation, but also contributed to by the hydraulic valves and by software filters. These software filters are necessary because the robot has higher order dynamics; where

the simulation is a pure second order system, the robot has higher order dynamics that can be be easily excited. During single support, our dynamics model of the swing leg is currently inaccurate. We believe this is because the mass distribution model was fit using data taken only with both feet on the ground. Improving the model to fix thiw will be part of the work of this thesis.

4.3.1 Weighted-Objective Inverse Dynamics

For controlling the real robot, we replace the Dynamic Balance Force Control low level controller discussed in Section 3.4.3 with Weighted-Objective Inverse Dynamics adapted from [89]. To use the DBFC controller, we first determine the desired contact forces and torques at each foot. Then DBFC uses a pseudo-inverse to find the least squares solution to a set of equations to get the joint torques and accelerations. It is important to realize that it had the same number of variables as it had equations with large weights (not counting the regularization equations). Essentially, it functioned as a perfectly determined system (same number of equations and variables) with some regularization rather than as an overdetermined system. For the Weighted-Objective Inverse Dynamics, we combine the two steps by simulataneously solving for the joint accelerations, joint torques, and contact forces/torques. We also wish to handle constraints such as the ZMP constraints, friction cone constraints, torque limits, and joint limit constraints, so we solve using off-the-shelf Quadratic Programming software [31]. We implement joint-limit constraints by constraining the joint acceleration when near the limit.

In addition to rearranging the 60 equations (18 dynamics equations, 12 foot acceleration equations, 18 acceleration regularization equations, and 12 torque regularization equations) in (3.59), we add many new equations. We add 1 equation for the vertical weight distribution between the two feet, 3 equations for CoM acceleration, 3 equations for the total moment around the CoM, 12 equations to regularize the contact forces, 3 equations to control torso angular acceleration, 2 equations for the CoP, and 12 equations to accelerate towards the reference pose produced by the inverse kinematics, q_{IK} (described in the next Section). The equations for track-

ing the reference pose look like

$$\ddot{\mathbf{q}}_i = k_p(\mathbf{q}_{\mathrm{IK},i} - \mathbf{q}_i) + 2\sqrt{k_p}((\dot{\mathbf{q}}_{\mathrm{IK},i} - \dot{\mathbf{q}}_i) - \mathbf{q}_{g,i}, \tag{4.1}$$

where the subscript i indicates the ith joint, $\mathbf{q}_{\mathrm{ref}}$ is the reference pose, $\mathbf{q}_g$ is the joint acceleration induced by gravity, and k_p is a proportional gain. The $2\sqrt{k_p}$ term ensures that this acceleration is critically damped.

This gives us a system of 42 variables and 96 equations, which we write as

$$\mathbf{A}\begin{pmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\tau}_j \\ \mathbf{f} \end{pmatrix} = \mathbf{b}, \tag{4.2}$$

where $\mathbf{A}$ is a 96 by 42 matrix and $\mathbf{b}$ is a 96-vector. Even without the regularization equations, this is an overdetermined system. In order to modify the relative importance of each equation, we multiply both sides of (4.2) by a diagonal weighting matrix, $\mathbf{W}$, which multiplies each row of $\mathbf{A}$ and $\mathbf{b}$ by the corresponding weight,

$$\mathbf{WA}\begin{pmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\tau}_j \\ \mathbf{f} \end{pmatrix} = \mathbf{Wb}. \tag{4.3}$$

We have selected the actual weights manually through experimentation. To turn this into a QP, we construct $\mathbf{A}^\mathsf{T}\mathbf{WWA}$ as the quadratic cost matrix and $-b^\mathsf{T}\mathbf{WA}$ as the linear cost matrix.

For standing balance, either this controller or the low-level controller based around DBFC (Section 3.4.3) work well, but choosing the contact forces in the same optimization as joint torques and accelerations produces somewhat better responses to perturbations because a larger range of tradeoffs can be considered. The Weighted Objective Inverse Dynamics is much more flexible, making it more successful for more complicated activities such as walking. We do not need to structurally change the controller to cope with different situations as we had to for DBFC (Section 3.4.3), though (manual) adjustment of the weights is sometimes necessary. Additionally, this framework allows us to easily use more of the robot's joints actively if we need to.

4.3.2 Inverse Kinematics

We have found that in addition to the torques produced by our inverse dynamics, having PD controllers on individual joints with low gains helps to produce smooth motion. This is especially important for the swing leg during walking because we are unable to accurately achieve desired swing foot accelerations from inverse dynamics alone (likely due to an inaccurate mass distribution model). Unfortunately, our walking controller (Section 3.4) goes directly from system state to desired accelerations and torques without producing desired positions and velocities. To get the desired positions and velocities, we define virtual points representing the desired foot and CoM positions and velocities, then use Inverse Kinematics (IK) to find the desired joint positions and velocities.

We use virtual points to represent the desired positions of both feet in the x, y, and z directions as well as the CoM in only the x and y directions (we use a constant desired CoM height), for 8 total degrees of freedom. For each degree of freedom, we treat the point as a generic second order system to which we apply the desired acceleration, a_{des} (produced by the high level DP policies). To keep the points near the true system in the face of persistent acceleration errors, we also integrate (with gain k_I) them towards the measured position, p_{meas} and measured velocity, p_{meas}. The update equation for the virtual points' position, p, and velocity, v, therefore looks like

$$p_{i+1} = k_I T p_{\text{meas},i} + (1 - k_I T)(p_i + v_i T + 1/2 a_{\text{des},i} T^2) \tag{4.4}$$

$$v_{i+1} = k_I T v_{\text{meas},i} + (1 - k_I T)(v_i + T a_{\text{des},i}), \tag{4.5}$$

where the subscript i indicates the time step and T is the duration of a time step. Using (4.4) and (4.5), v will not actually be the derivative of p. In order to supply the inverse kinematics solver with something consistent, we therefore use the effective velocity $v_{\text{eff},i+1} = (p_{i+1} - p_i)/T$. We also do not allow the foot virtual points to go below $z = 0$. Note that if the robot actually achieves the desired velocity, these virtual points will lie directly on the measured positions and have the measured velocities. Figure 4.6 in Section 4.5 shows the motion of the virtual CoM relative to the true CoM for swaying.

Our inverse kinematics solver (adapted from the one described in [89]) maintains a reference pose, $\mathbf{q}_{\text{IK}}$. During each time step, we take the desired foot and CoM positions and velocities as well as the desired torso orientation and angular velocity and solve for the best velocities, $\dot{q}_{\text{IK}}$. We then update the positions $\mathbf{q}_{\text{IK},i+1} = \mathbf{q}_{\text{IK},i} + T\dot{\mathbf{q}}_{\text{IK},i}$. We can then use $\mathbf{q}_{\text{IK}}$ and $\dot{\mathbf{q}}_{\text{IK}}$ as desired positions and velocities for low gain individual joint PD controllers. We also use it as a reference pose within the inverse dynamics.

To solve for $\dot{\mathbf{q}}_{\text{IK}}$, we set up a quadratic program and use an off-the-shelf QP solver. The setup is similar to that used for the Inverse Dynamics described above, except that the only variables are the generalized velocities (6 base coordinates and 12 joint angular velocities) in $\dot{\mathbf{q}}_{\text{IK}}$. We again set up an overconstrained system with manually selected weights on the equations as in (4.3).

4.3.3 Jerk-Based Policies

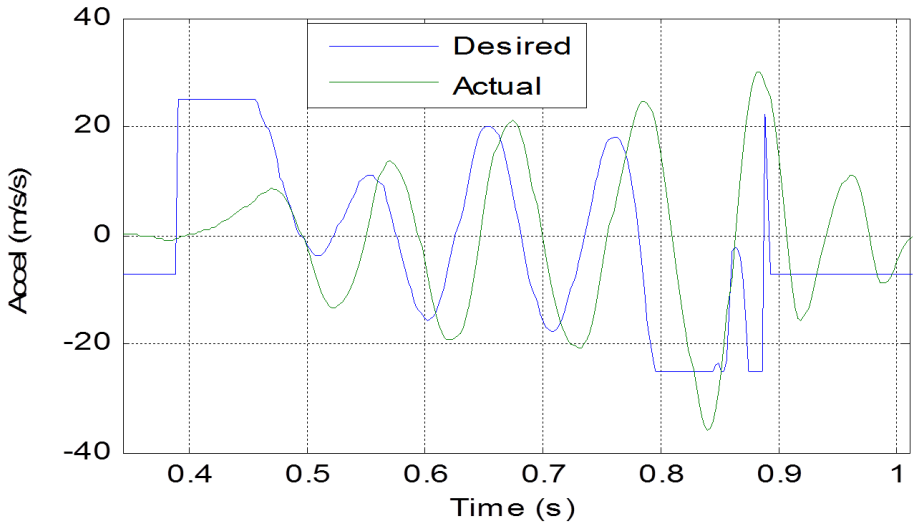

Figure 4.2: Plot of the desired and measured swing foot acceleration in the z direction for one step. Measured accelerations are obtained by filtering and double integrating potentiometer data.

One of the major difficulties with controlling walking on the Sarcos humanoid is control of

the swing foot. If we have the robot walk in place using the swing leg policies described in Section 3.4.1, we get oscillation. Figure 4.2 shows the measured and desired acceleration for the swing foot in the z direction. There is a slow initial rise and a large amplitude oscillation at about 9 Hz. Additionally, there is a delay of about 23 ms from the desired to the measured acceleration.

The delay has at least three component causes. There is a slight delay between desired accelerations and desired torques because of a filter on the output torques. The purpose of this filter is to avoid exciting higher order dynamics of the mechanical system. We can increase the cutoff frequency of this filter, but avoiding abrupt changes in desired torque is important. We also have a delay of about 9 ms between desired torques and measured torques caused by the physical characteristics of the hydraulic valves and our low level force control. Finally, we have a significant delay between actual system acceleration and measured acceleration due to state estimation.

Unmodeled delays can cause oscillations, and simple simulations have shown that the oscillations we see in Figure 4.2 are consistent (in frequency and amplitude) with what would be caused by a 23 ms delay. However, changes to the cost function used for policy generation which mitigated the impact of delays in simulation had little effect on the real hardware, indicating that delays are not the sole cause of oscillation.

To prevent large oscillations in commanded swing foot acceleration, we penalize its derivative, jerk. To support this change, we increase the order of our policies by one from acceleration-based to jerk-based. We add acceleration as a 4th state (in addition to the already existing, time until touchdown, position, and velocity), and we replace acceleration as the control input with jerk. Jerk-based swing foot policies can be coordinated in the same way as the original acceleration-based policies within the ICS framework. When running the policy on the robot, we do not attempt to measure the actual acceleration. Instead, we have acceleration as a controller state that starts at 0 m/s/s at the beginning of swing and evolves ideally according to the commanded jerk. We then pass this acceleration on to the inverse dynamics, which considers a

second order model of the system. From the point of view of the rest of the system, the jerk-based policies still produce a desired acceleration, but now we penalize its rate of change.

To keep computation reasonable, we must reduce the resolution of the state space grid for the other dimensions when adding an additional state space dimension. We partially mitigate the effect of this by also reducing the range of the grid so that the grid cells are closer to their original size. We now use a grid with minimum states of $\{0$ m, -0.8 m/s, -20 m/s/s 0 s$\}$, maximum states of $\{0.06$ m, 0.8 m/s, 20 m/s/s, 0.6 s$\}$, and resolutions of $\{61, 81, 61, 121\}$.

For the swing-Z subsystem, we replace the original cost function, (3.42) with

$$L(\mathbf{x}, \mathbf{u}) = \left(\frac{\ddot{z}}{\ddot{z}_{\max}}\right)^2 + \left(\frac{\dddot{z}}{\dddot{z}_{\max}}\right) + 2\left(\frac{z - z_{\text{nom}}}{z_{\text{nom}}}\right)^2, \tag{4.6}$$

where $\ddot{z}_{\max}$ is still 25 m/s/s and $\dddot{z}_{\max}$ is 300 m/s/s/s. We also add jerk to the analytic controller that takes over for $t_{td} \leq 0.04$s. We increase this limit from 0.03 s in the acceleration-based case because of the more limited control authority and decreased grid resolution. We now replace (3.43) with

$$C = \int_0^T 4\dddot{z} + 4\ddot{z}\, dt + 3000(\dot{z}_f - \dot{z}_{\text{nom}})^2, \tag{4.7}$$

and find the constant jerk, $\dddot{z}$ that minimizes (4.7). The constant coefficients are selected for the same reasons described in Section 3.4.1. We use an integral rather than simply multiplying by T as in (3.43) because acceleration, $\ddot{z}$, is not constant. We also loosen the touchdown timing constraint somewhat and now require that touchdown actually occur ($z = 0$) within 0.003 seconds of the nominal time. The looser constraint is necessary because of both the reduced control authority of the jerk-based policies and the slower time steps when controlling the robot (1 ms for simulation and 2.5 ms for the robot). We make a similar modification to the combined Swing-X/Swing-Y policy.

Figures 4.3 and 4.4 compare the response of the acceleration-based and jerk-based polcies to a 20 ms delay in simulation. The jerk-based policy is less affected by the delay.

Unlike simply adjusting the cost function, this improvement translated well to operating on hardware. Figure 4.5 compares the performance of the acceleration-based policy and jerk-based

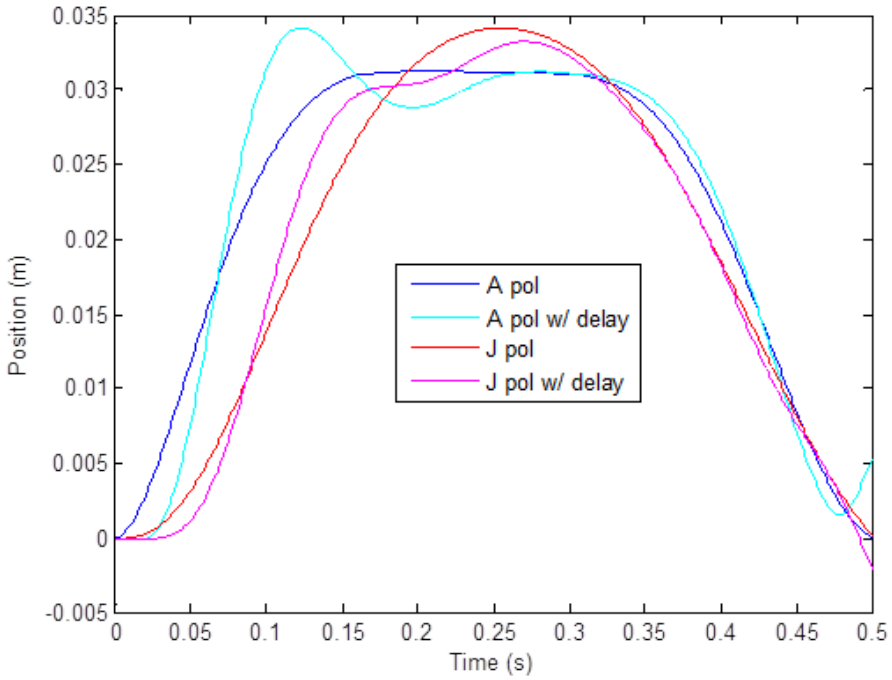

Figure 4.3: Comparison of the acceleration-based and jerk-based swing-z policies showing the position that results with and without a 20 ms delay.

policy for walking in place on the Sarcos humanoid robot. We still have trouble accurately placing the swing foot, but jerk-based policies eliminate much of the oscillation.

4.4 Integral Control

Modeling errors can cause persistent errors, especially with our emphasis on low gains. However, our hardware, while difficult to model, is very repeatable. What happened in the past is generally an accurate indicator of what will happen in the future and integral control can be a very effective means of compensating for modeling error.

Our state estimator generally considers whichever way the torso is facing to be "forward". To maintain a desired posture, we define the direction the feet are facing (average of the two feet) as the desired heading and use a PID controller to generate desired torso angular accelerations.

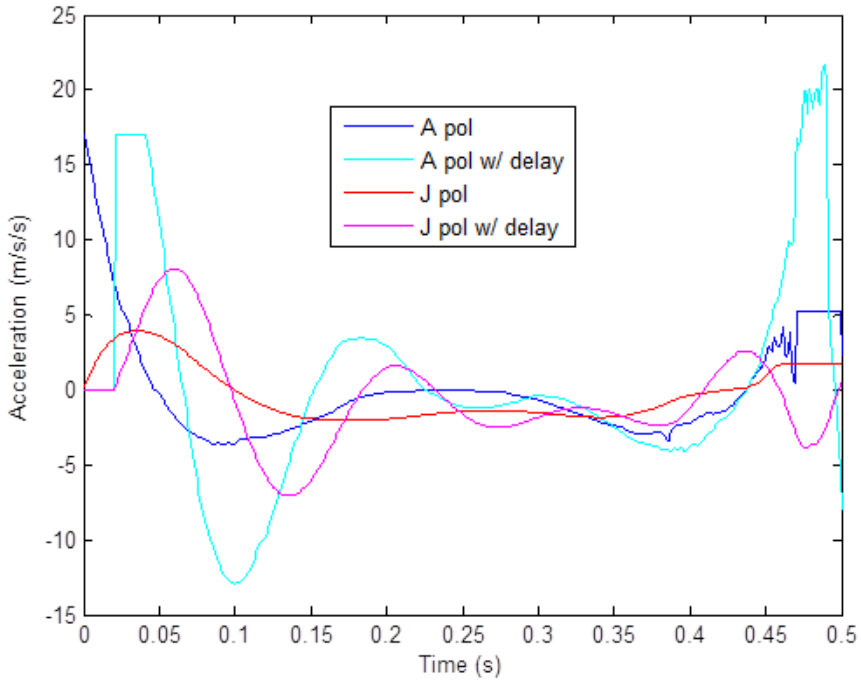

Figure 4.4: Comparison of the acceleration-based and jerk-based swing-z policies showing the acceleration that results with and without a 20 ms delay.

Unfortunately, even slight modeling errors can result in significant steady state error using only PD gains, but the addition of an integral term results in forward-facing standing.

4.4.1 Virtual Forces and CoM Offsets

A problem with using integral terms in this way is that once we have reached steady state, we have an integral in the high level controller continuously balancing a modeling error. The inverse dynamics will be attempting to produce the acceleration of the integral term but actually producing zero acceleration due to modeling error. If what we actually want is zero acceleration, this works well for achieving the feature we are integrating. However, in a complicated, interconnected system such as a many-linked humanoid robot, it can cause other problems. For example, if the inverse dynamics believes that the torso will be accelerating and it wants to keep

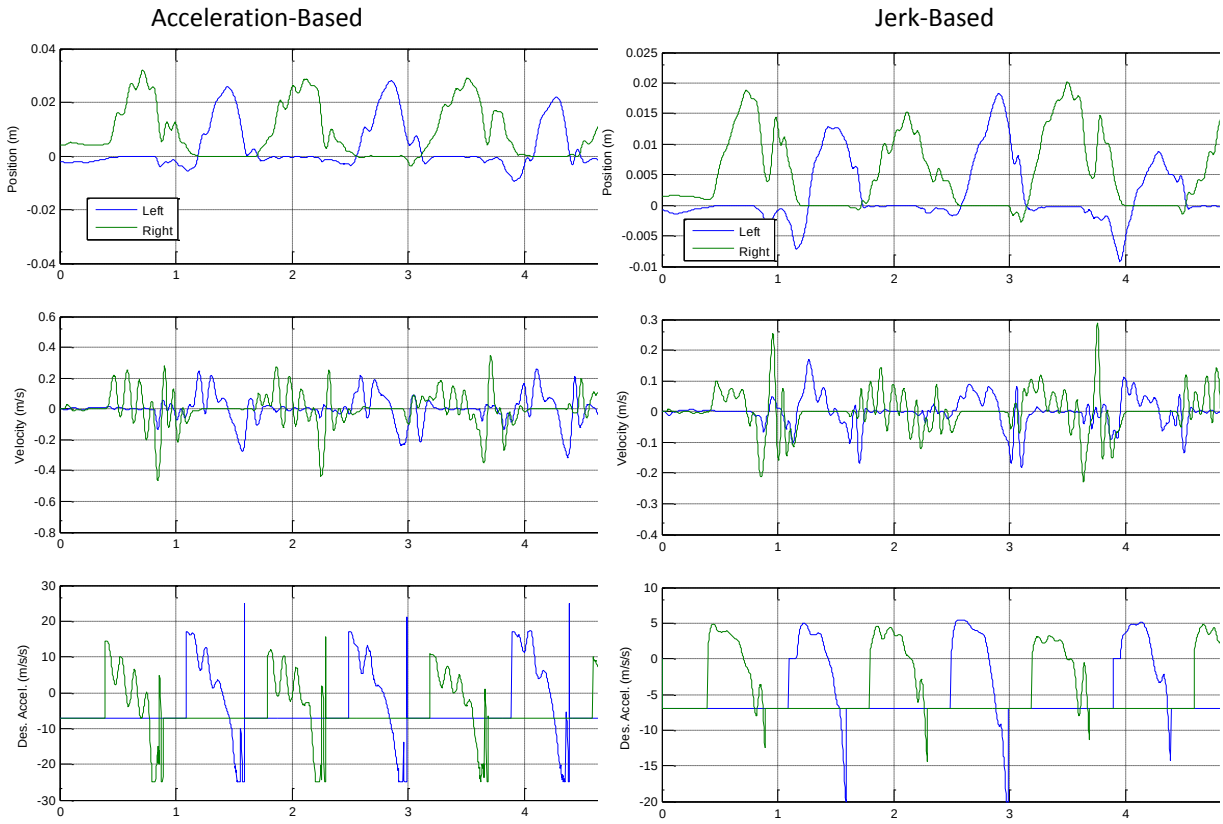

Figure 4.5: Comparison of the acceleration-based and jerk-based swing-z policies for walking in place on the Sarcos humanoid robot.

its foot in the same place, it will accelerate its hip in the opposite direction to counteract the torso acceleration. If, however, the torso does not accelerate, but the hip does, we end up with foot acceleration. In particular, this can make control of the swing foot more difficult.

For this reason, we prefer to instead use integral control to modify the dynamic model. This way, the solved for torques, accelerations, and contact forces remain consistent. This requires that we find some parameters to modify in the dynamic model that reasonably capture the type of error that is actually occuring. We choose to add virtual torques, τ_{virt}, on the torso to the dynamic model, changing the dynamics equations used by the inverse dynamics to

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{S}\tau - \mathbf{J}^{\mathsf{T}}\mathbf{F} = -\mathbf{N}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{J}_h^{\mathsf{T}}\tau_{\text{virt}}, \tag{4.8}$$

where $\mathbf{J}_h^\top$ is the hip Jacobian for the virtual torques. These are identical to the normal dynamic equations except for the addition of the final term. The virtual torques can directly represent the torques that the tether hoses put on the robot, but they can also generally represent any modeling error that tends to rotate the torso. We compute each component of τ_{virt} individually by integrating the commanded angular acceleration,

$$\tau_{\text{virt},i} = k_I \int \ddot{\theta}_i \, dt. \tag{4.9}$$

By imagining a virtual force causing the error, the low level inverse dynamics controller pushes back against it and corrects for it, without expecting unrealistic accelerations.

To correct translational motion in the horizontal plane, we use offsets to the CoM location. We know that if the CoM is not accelerating, the CoP should be directly under the CoM. We have a measurement of the CoP force/torque sensors, so we know where the CoM actually is in the horizontal plane. We therefore integrate the error to generate offsets, δ, to the CoM within our dynamics model.

$$\delta_x = k_I \int \mathbf{p}_x - \mathbf{c}_x \, dt \tag{4.10}$$

$$\delta_y = k_I \int \mathbf{p}_y - \mathbf{c}_y \, dt \tag{4.11}$$

where $\mathbf{p}$ is the location of the CoP and $\mathbf{c}$ is the location of the CoM. We use these offsets by adding them to the CoM location generated by forward kinematics.

Running either the CoM offset integrators or virtual torque integrators continuously can be problematic. Both sets of integrators requires (in their current form) that there be no acceleration for them to do the right thing. If we run the integrator continuously with a large k_I, these dynamic effects will cause problems. Additionally, we may run into resonances as it interacts with with other components of the controller. If we use a low k_I on the other hand, the dynamic effects will cancel out, but it will be inconvenient because we will have to wait a long time for the integrator to achieve steady state. The solution to this is to only run the integrators intermittently and save the results. We run the integrators until they achieve steady state, then save the CoM offsets and

virtual torques and continue to use them as constants. In practice we do this at least once a day and often more frequently to compensate for frequent changes in the system.

4.5 Swaying

Swaying was a convenient first step towards walking. By swaying, we mean moving the CoM laterally and shifting weight so that most of it is on a single foot, but without actually lifting the feet. We also want to accomplish the swaying without using time-indexed trajectories so that it can adjust the timing of the motion in response to external disturbances. Initially focusing on this allowed us to test many of the types of things we would need to do for walking, but without doing everything all at once. The goal of these experiments was primarily to accurately achieve desired CoM accelerations.

We accomplished swaying by using a simple PD controller for CoM motion in the forward direction and a DP policy for CoM motion in the lateral direction. The policy has the CoM position, y, and velocity $\dot{y}$ as states and CoM acceleration, $\ddot{y}$ as its only action. We used a somewhat complicated cost function to produce stable periodic motion with a desired amplitude of $a = 0.08\text{cm}$ and a period of $t = 2\text{s}$.

$$L(\mathbf{x}, \mathbf{u}) = 2(E_1(\mathbf{x}) - 1)^2 + (E_2(\mathbf{x}) - 1)^2 + p^2 + \ddot{y}^2, \tag{4.12}$$

where E_1 and E_2 are two different measures of "orbital energy", and p is the CoP location relative to the center point between the feet. Orbital energy based on an elipse in position-velocity space is given by

$$E_1(\mathbf{x}) = \frac{y}{a}^2 + \frac{\dot{y}T}{32a}^2 \tag{4.13}$$

and orbital energy based on an elipse in velocity-acceleration space is given by

$$E_2(\mathbf{x}) = \frac{\dot{y}T}{32a}^2 + \frac{\ddot{y}T^2}{32a}^2 . \tag{4.14}$$

With only E_1 in the cost function, the policy can choose to remain stationary at $y = a$ with little cost. The combination of E_1 and E_2 in the cost function results in stable periodic motion. Figure

4.6 shows the result of applying this controller on the Sarcos robot. The IK CoM is determined by the method of virtual points described in Section 4.3.2, which experiments showed to be more effective (in terms of achieving desired acceleration) than the alternatives of putting the IK CoM on the true CoM or as a stationary point centered between the feet.

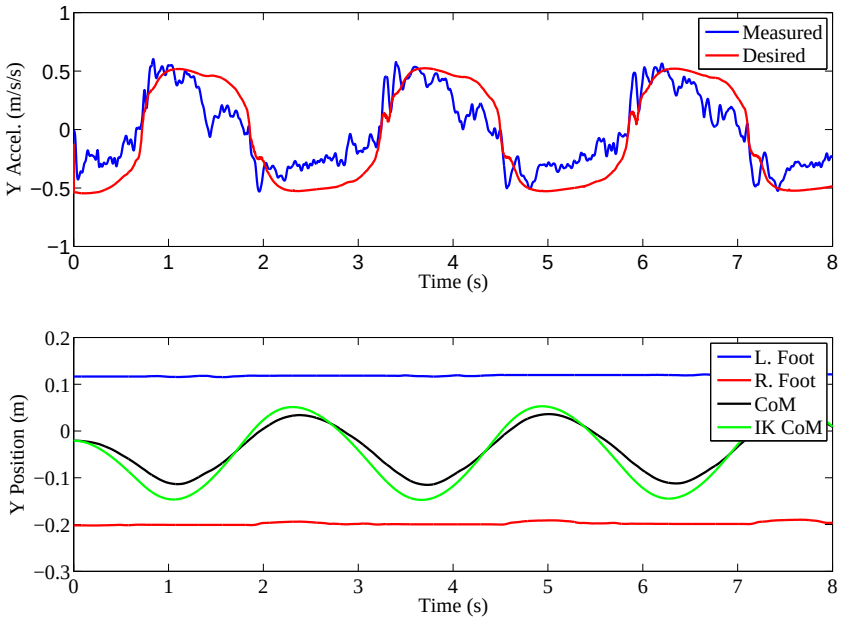

Figure 4.6: Lateral motion during swaying experiments. Accelerations are plotted on top and position on bottom.

The primary source of error in achieving the desired acceleration was failure to achieve the desired CoP. The primary cause of that failure was in turn the failure to distribute weight between the two feet as expected. It is difficult to accurately put the desired fraction of weight on each leg because the result is sensitive to small modeling errors near the straight-knee kinematic singularity. However, we can compensate for this error when trying to obtain the desired CoP. The CoP (in the x direction) is related to the contact forces by

$$0 = \mathbf{f}_{z,L}(\mathbf{l}_x - \mathbf{p}_x) + \mathbf{f}_{z,R}(\mathbf{r}_x - \mathbf{p}_x) - \tau_{y,L} - \tau_{y,R}, \tag{4.15}$$

where $\mathbf{l}$, $\mathbf{r}$, and $\mathbf{p}$ are the locations of the left foot, right foot, and CoP. We model the effect of

modeling error on the vertical force ase

$$\mathbf{f}_{z,L} = \mathbf{f}_{z,L,C} + \Delta\mathbf{f}_{z,L}, \tag{4.16}$$

where $\mathbf{f}_{z,L,C}$ is the commanded force and $\Delta\mathbf{f}_{z,L}$ is the error caused by modeling error. Substituting (4.16) into (4.15) and rearranging gives

$$(\Delta\mathbf{f}_{z,L} + \Delta\mathbf{f}_{z,L})\mathbf{p}_x - \Delta\mathbf{f}_{z,L}\mathbf{l}_x - \Delta\mathbf{f}_{z,R}\mathbf{r}_x = \mathbf{f}_{z,L}(\mathbf{l}_x - \mathbf{p}_x) + \mathbf{f}_{z,R}(\mathbf{r}_x - \mathbf{p}_x) - \tau_{y,L} - \tau_{y,R}. \tag{4.17}$$

We can find excellent predictions of $\Delta\mathbf{f}_{z,L}$ and $\Delta\mathbf{f}_{z,R}$ by filtering the recent history, and we can therefore use (4.17) rather than (4.15) in the inverse dynamics to more accurately achieve the desired CoP.

Unfortunately, the compensating terms interact with the control and we get an approximately 7 Hz oscillation with a large amplitude resulting in instability unless we use extremely slow filters for $\Delta\mathbf{f}_{z,L}$ and $\Delta\mathbf{f}_{z,R}$. If, however, we damp out the oscillations by multiplying the compensating terms (the entire left half side of (4.17)) by a coefficient less than 1, we can use faster filters. Figure 4.7 shows results using a coefficient of 0.75 and a cutoff frequency of 5 Hz for the filters. The expected vertical force closely matches the actual. The result is that we accurately track the desired CoP and therefore accurately achieve the desired CoM acceleration.

This compensation works well with both feet on the ground. However, during walking double support is very brief and the weight shifts very quickly, requiring faster filters. As a result, the compensation has a negligable effect on performance while walking, so we remove it for simplicty. We only use it when performing tasks that have both feet on the ground for extended periods of time.

4.6 Preliminary Walking in Place

We have also implemented a preliminary walking in place controller. It uses a collection of coordinated policies for a high level controller. Joint torques are generated by the inverse dynamics

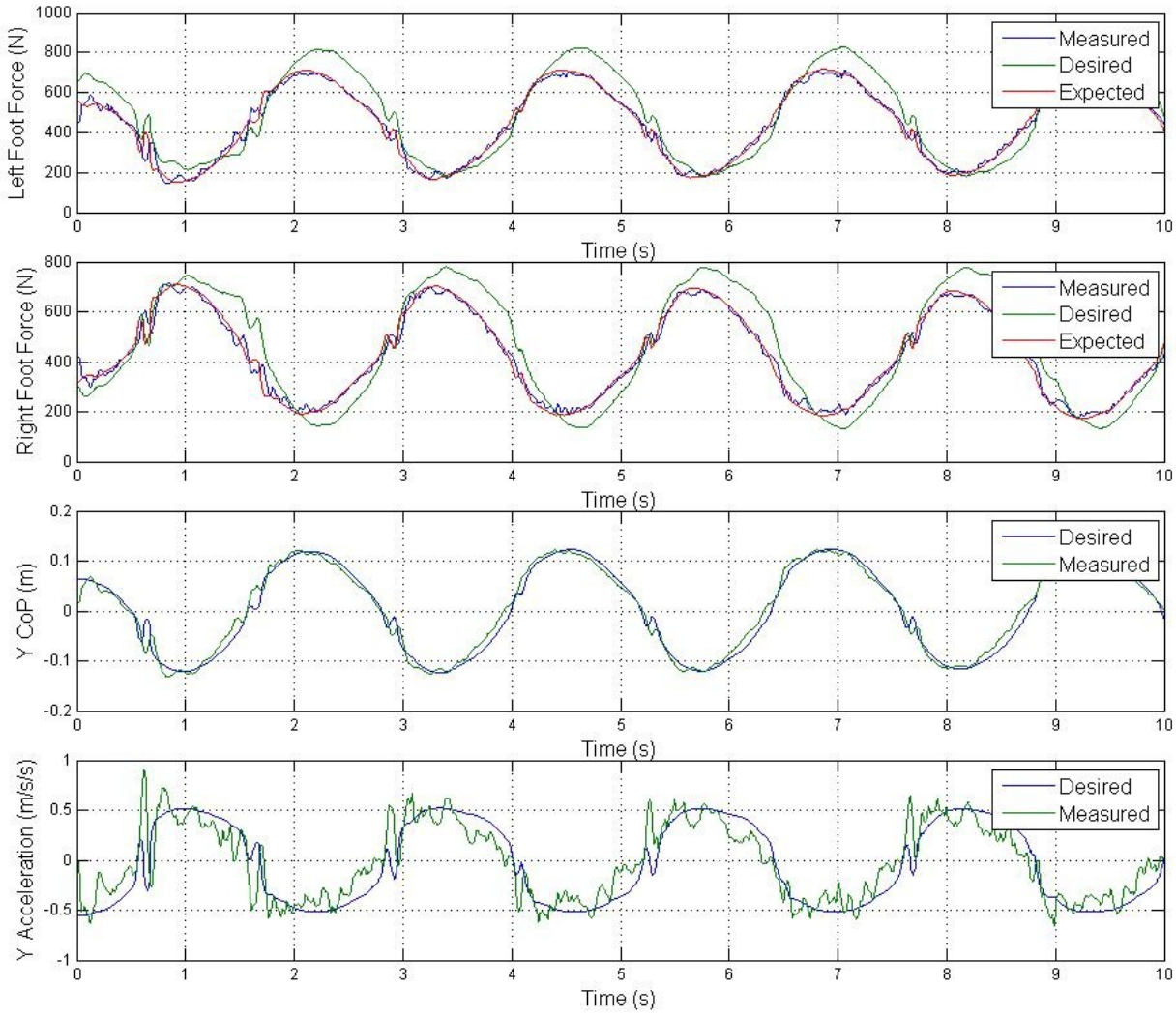

Figure 4.7: Swaying while compensating for inaccurate vertical forces.

algorithm described in Section 4.3.1 plus individual joint PD gains supported by the inverse kinematics algorithm described in Section 4.3.2. The current version simplifies coordination by using fixed step timing rather than re-optimizing it continuously. The steps run off of a clock at a slightly slow cadence: 0.2 second double support and 0.5 second single support. The first double support is increased to 0.4 seconds to allow for a shift onto the stance foot. This reduces coordination to two (completely decoupled) one-dimensional optimizations for touchdown location in the x and y directions during single support and weight distribution (accomplished as described

in Section 3.3.1) during double support. For swing foot policies, we use the jerk-based policies described in Section 4.3.3. We use the same coronal plane stance policy as in simulation, except with the nominal timing adjusted to match the slightly slower cadence. The coronal policy has no notion of absolute location in the global reference frame, so it is only neutrally stable with respect to sidestepping. To keep the system in place as it walks, we add a small proportional gain on absolute position to the desired CoP. We use the state estimator's odometry to get absolute position, which does drift slowly.

For walking in place, we use an analytic policy rather than a policy generated by DP for control of the CoM motion in the forward direction. If we do not specifically address it, we have more trouble with movement in world coordinates in this direction because of coupling between the vertical motion of the foot and the forward motion of the foot. If we simply try to stay in one place, the robot will slowly walk forward. A small proportional gain on position added to the CoP is insufficient to keep it in one place. To remain stationary, we select the footstep location such that for the next footstep (in 0.7 seconds), the capture point will be at the desired CoM location, which we assume to be the origin without loss of generality. Note that if the the capture point is already at the origin, then placing the foot there will result in the system coming to rest at the origin and all subsequent capture points and steps being there. Therefore, this is theoretically a deadbeat controller. We compute the desired footstep location analytically, by starting with the LIPM dynamic equation

$$\ddot{x} = \frac{g}{h}(x - p) \tag{4.18}$$

where h is the height, x is the CoM location, and p is the CoP location. If we solve the differential equation, we get

$$x(t) = C_1 e^{\sqrt{\frac{g}{h}}t} - C_2 e^{-\sqrt{\frac{g}{h}}t} \tag{4.19}$$

and its derivative

$$\dot{x}(t) = C_1\sqrt{\frac{g}{h}}e^{\sqrt{\frac{g}{h}}t} + C_2\sqrt{\frac{g}{h}}e^{-\sqrt{\frac{g}{h}}t}, \tag{4.20}$$

where C_1 and C_2 are unknown constants depending on the initial conditions. If we add the initial

conditions (2 equations) and the constraint that the capture point be at the origin on the next step (1 equation), we get a system of 5 equations and 5 unknowns: C_1, C_2, x_1, $\dot{x}_1$, and p, where x_1 is the CoM position at the next step. We can then solve this system to find where to step now

$$p = \frac{e^{\sqrt{\frac{g}{h}}T}(x_0\sqrt{\frac{g}{h}} + \dot{x}_0)}{\sqrt{\frac{g}{h}}(e^{\sqrt{\frac{g}{h}}T} - 1)}, \tag{4.21}$$

where T is the time until the following touchdown. To coordinate this with the swing foot policy, we need a value function to specify how important it is that we actually put the foot at p. We construct a parabola with its vertex at $\{p, 0\}$ and a constant quadratic term to act as the value function.

One of the primary reasons for simplifying step timing was that inaccurate measurements of foot height makes it difficult to predict when touchdown will occur. The foot often hits the ground (as detected by the force plates) when the state estimator still believes that it is significantly above it. The inability to predict touchdown causes other problems as well. For example, attempting to apply large lateral ground contact forces before the foot is firmly placed can result in rapid motion and a drastically misplaced foot. Accordingly, while the nominal time until touchdown runs off of the clock and simply counts down, we actually switch to double support when we detect touchdown with the force plates. We consider touchdown to have occured after we detect vertical forces of greater than 100 N for 5 consecutive time steps, after which we immediately switch to the beginning of double support, regardless of what the clock says. If we reach $t_{td} = 0$ before touchdown is detected, we remain at $t_{td} = 0$ until we detect touchdown. For the stance policies this means nothing special. For the horizontal swing policies, we attempt to achieve zero velocity to avoid slipping at touchdown. For the vertical swing policy, we continue to move downwards rapidly, ensuring that we do not touchown too late. Figure 4.8 shows results for walking in place.

As in simulation, we can stop the robot simply by switching to a standing controller from the walking controller. The transition goes much more smoothly if done during double support, especially early in double support. If done during single support, the swing foot stomps violently

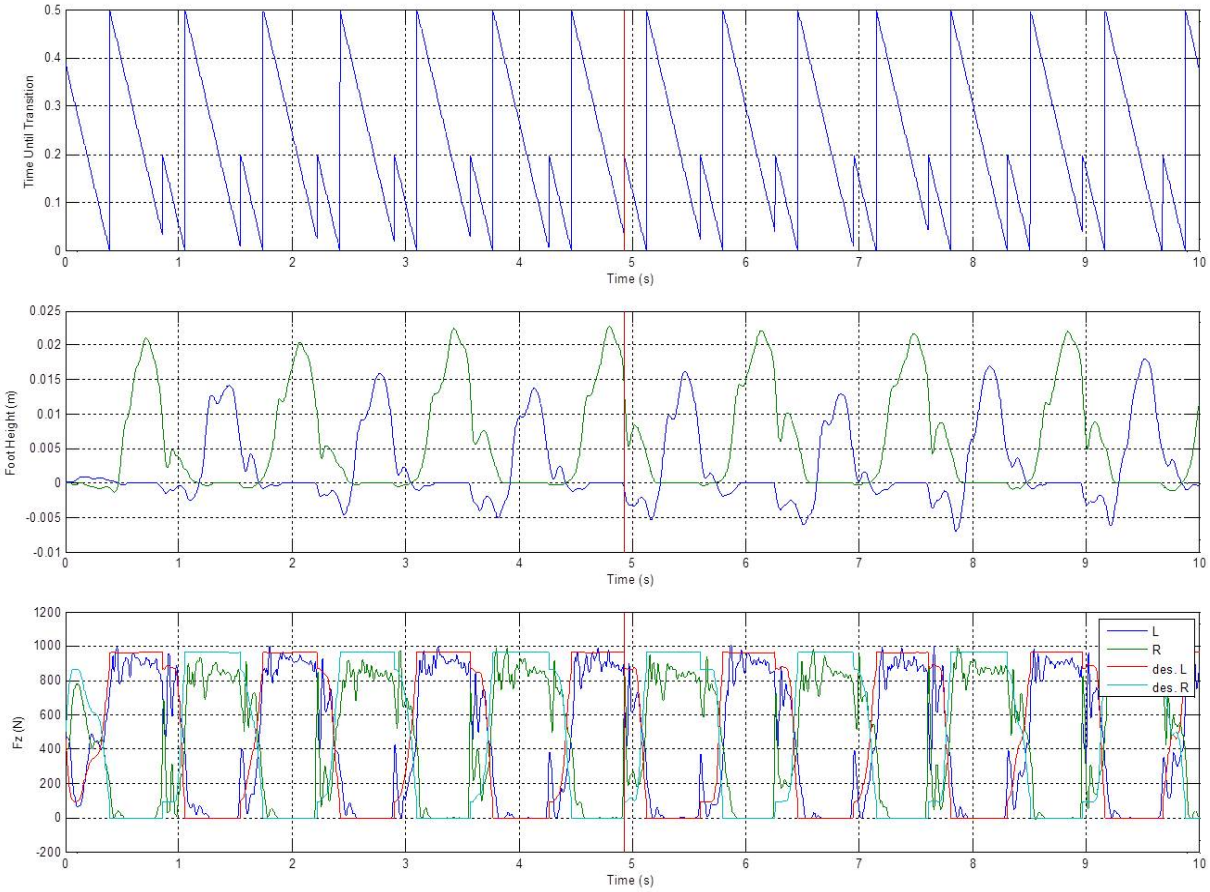

Figure 4.8: Walking in place. Time until transition (top), foot heights (middle), and vertical forces (bottom).

as the robot tries to abruptly put weight on it.

Chapter 5

Future Work

The remainder of my thesis will focus on improving walking both in simulation and on hardware. In simulation, this will entail generating faster, more human-like walking as well as improving the controller's ability to handle rough terrain. One possible improvement to the simulated walking is to add arms to the simulation and use arm swing to help stabilize yaw. Another possible improvement is produce walking that uses heel strike and and toe off. This will require breaking the sagittal policy at least (and possibly other policies as well) into additional phases besides just double support and single support with different contact constraints. It will also require upgrading the ground contact model to support non-flat feet.

For walking on the Sarcos humanoid robot platform, the main goal will be to produce forward walking at a reasonable speed that is stable enough to be pushed without falling. We expect that current efforts to improve the system model and sensor calibration will be helpful. For walking in place, we would like to improve our coordination strategy to allow for continuously optimizing footstep timing. We would also like to improve control of the swing foot to increase placement accuracy and ground clearance. Moving forward slowly should require simply substituting a DP-generated policy for the analytic policy currently in use. We will address the problems that arise as we try to move faster when it becomes clearer what they are.

We also hope to achieve improved robustness by changing our underlying Dynamic Program-

ming. We intend to more thoroughly analyze the theoretical performance of the Multiple Model Dynamic Programming algorithm. We believe that combining DP with learning shows more promise than the multiple model dynamic programming for our situation. We will therefore continue work on that method, including analyzing the theoretical performance of the algorithm and determining the best way to use uncertainty. Specifically, we hope to apply it to our swing leg policies. We currently see repeated undesired motion on every step - for example, the foot tends to move forward and back on every step rather than remaining in place in the forward direction. We hope that our high level controller can learn a model of the physical system plus the low level controller in order compensate for consistent errors. We also expect that there will be other ways to use learning to improve robustness, either by improving our system model or by learning aspects of the controller directly.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 1–8, 2005. 2.4.2

[2] Yeuhi Abe, C Karen Liu, and Z Popovic. Momentum-based Parameterization of Dynamic Character Motion. *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 194–211, 2004. 3.1.1

[3] K. Akachi, K. Kaneko, N. Kanehira, S. Ota, G. Miyamori, M. Hirata, S. Kajita, and F. Kanehiro. Development of humanoid robot HRP-3P. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 50–55, 2005. 1.3

[4] Yaman Arkun and George Stephanopoulos. Studies in the synthesis of control structures for chemical processes: Part IV. design of steady-state optimizing control structures for chemical process units. *AIChE Journal*, 26:975–991, 1980. 2.3.1

[5] Christopher G. Atkeson. Randomly sampling actions in dynamic programming. In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007. 2.1, 2.2, 2.2

[6] Christopher G. Atkeson and Jun Morimoto. Nonparametric representation of policies and value functions: A trajectory-based approach. In *NIPS*, pages 1611–1618, 2003. 2.1

[7] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, pages 12–20, 1997. 2.4.2

[8] Christopher G. Atkeson and Benjamin J. Stephens. Random sampling of states in dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(4):924–929, 2008. 2.1

[9] V. Azhmyakov, V.G. Boltyanski, and A.S. Poznyak. On the dynamic programming approach to multi-model robust optimal control problems. *American Control Conference*, pages 4468–4473, 2008. 2.3.1

[10] D. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *IEEE International Conference on Robotics and Automation*, 2001. 2.3.1

[11] J. Andrew Bagnell, Andrew Y. Ng, Sham Kakade, and Jeff Schneider. Policy search by dynamic programming. In *in Advances in Neural Information Processing Systems*, page 16. MIT Press, 2003. 2.4.1

[12] F. J. Bejarano, A. Poznyak, and L. Fridman. Min-max output integral sliding mode control for multiplant linear uncertain systems. In *American Control Conference*, pages 5875–5880, 2007. 2.3.1

[13] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. 1.4, 2.1, 2.1, 2.3.1

[14] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6(4):679–684, 1957. 2.1

[15] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, volume 245 of *Lecture Notes in Control and Information Sciences*, pages 207–226. Springer Berlin / Heidelberg, 1999. 2.3.1

[16] Darrin C. Bentivegna, Christopher G. Atkeson, and Jung-Yup Kim. Compliant control of a compliant humanoid joint. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2007. 3.4, 4.2

[17] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. ISBN 1886529132. 2.1, 2.3.1

[18] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003. 2.4.4

[19] John Camp. Powered "passive" dynamic walking. Master's thesis, Cornell University, August 1997. 1.3

[20] Steven H. Collins, Martijn Wisse, and Andy Ruina. A three-dimensional passive-dynamic walking robot with two legs and knees. *I. J. Robotic Res.*, 20(7):607–615, 2001. 1.3

[21] Scott Davies. Multidimensional triangulation and interpolation for reinforcement learning. In *NIPS*, pages 1005–1011, 1996. 2.2

[22] Marc P. Deisenroth and Carl E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, June 2011. 2.4.1, 2.4.4

[23] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72:1508–1524, 2009. 2.4.1

[24] Holger Diedam, Dimitar Dimitrov, Pierre-Brice Wieber, Katja Mombaur, and Moritz Diehl. Online walking gait generation with adaptive foot positioning through linear model predictive control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008. 2.1, 3.1.1

[25] Moritz Diehl and Jakob Bjornberg. Robust dynamic programming for min-max model predictive control of constrained uncertain systems. *Automatic Control, IEEE Transactions on*, 49(12):2253–2257, dec. 2004. 2.3.1

[26] Geir E. Dullerud and Fernando Paganini. *A course in robust control theory: A convex approach*. Texts in Applied Mathematics. Springer, New York, NY, 2000. 2.3.1

[27] Silvia Ferrari and Robert F. Stengel. Smooth function approximation using neural net-

works. *IEEE Transactions on Neural Networks*, 16(1):24–38, 2005. 2.4.1

[28] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. on Robotics*, 21(6):1077–1091, December 2005. 2.1

[29] Y. Fujimoto and A. Kawamura. Proposal of biped walking control based on robust hybrid position/force control. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2724–2730. IEEE, 1996. 3.1.1

[30] R.J. Full and D.E. Koditschek. Templates and anchors: Neuromechanical hypotheses of legged locomotion on land. In *The Journal of Experimental Biology*, volume 202, pages 3325–3332, 1999. 1.4

[31] Luca Di Gaspero. Quadprog++. 4.3.1

[32] A Goswami and V Kallem. Rate of change of angular momentum and balance maintenance of biped robots. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volume 4, pages 3785–3790, Honda Res. Inst., Mountain View, CA, USA, 2004. 3.1.1

[33] Ambarish Goswami, Bernard Espiau, and Ahmed Keramane. Limit cycles in a passive compass gaitbiped and passivity-mimicking control laws. *Auton. Robots*, 4(3):273–286, July 1997. 1.4

[34] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of honda humanoid robot. *IEEE International Conference on Robotics and Automation Proceedings*, 2:1321–1326, 1998. 1.3

[35] Daan G. E. Hobbelen and Martijn Wisse. A disturbance rejection measure for limit cycle walkers: The gait sensitivity norm. *IEEE Transactions on Robotics*, 23(6):1213–1224, 2007. 1.3

[36] A. Hofmann, M. Popovic, and H. Herr. Exploiting angular momentum to enhance bipedal center-of-mass control. In *2009 IEEE International Conference on Robotics and Automation*, pages 4423–4429. Ieee, May 2009. ISBN 978-1-4244-2788-8. doi: 10.1109/ROBOT.2009.5152573. 3.1.1

[37] R.A. Howard. *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960. 2.1

[38] Qiang Huang, Kazuhito Yokoi, Shuuji Kajita, Kenji Kaneko, Hirohiko Arai, Noriho Koyachi, and Kazuo Tanie. Planning walking patterns for a biped robot. *IEEE Transactions on Robotics and Automation*, 17:280–289, 2001. 1.3

[39] Fumiya Iida and Russ Tedrake. Minimalistic control of a compass gait robot in rough terrain. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 3246–3251. IEEE Press, 2009. 1.4

[40] Eng J. J., Winter D. A., and Patla A. E. Strategies for recovery from a trip in early and late swing during human walking. *Experimental Brain Research*, 102:339–349, 1994. 3.6.3

[41] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Prgogramming*. American Elsevier Pub. Co., New York, 1970. 2.1, 2.1

[42] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. 2.4.1

[43] S. Kajita and K. Tani. Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1405–1411, April 1991. 3.3

[44] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Resolved momentum control: humanoid motion planning based on the linear and angular momentum. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, 2003. 3.1.1

[45] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d Linear Inverted Pendulum Model: A simple modeling for biped walking pattern generation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 240–246, November 2001. 3.3

[46] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1620–1626, September 2003. 1.3, 1.4, 3.1.1

[47] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002. 2.4.1

[48] K. Kaneko, F. Kanehiro, S. Kajita, K. Yokoyama, K. Akachi, T. Kawasaki, S. Ota, and T. Isozumi. Design of prototype humanoid robotics platform for HRP. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2431–2436, 2002. 1.3

[49] D. Kaynov, P. Soueres, P. Pierro, and C. Balaguer. A practical decoupled stabilizer for joint-position controlled humanoid robots. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3392–3397, October 2009. doi: 10.1109/IROS.2009.5354431. 3.1.1

[50] David Andrew Kendrick. *Stochastic control for economic models*. McGraw-Hill, second edition, 2002. 2.3.1

[51] Jung-Yup Kim, Christopher Atkeson, Jessica Hodgins, Darrin Bentivegna, and Sung Ju Cho. Online gain switching algorithm for joint position control of a hydraulic humanoid robot. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2007. 3.4, 4.2

[52] J. Ko, D. J. Klein, D. Fox, and D. Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 742–747, 2007. 2.4.1

[53] George Konidaris and Andrew Barto. Autonomous shaping: knowledge transfer in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine learning*, ICML '06, pages 489–496, 2006. 2.3.6

[54] R.E. Larson. *State increment dynamic programming*. American Elsevier Pub. Co., 1968.

2.2

[55] Steven M. Lavalle. From dynamic programming to rrts: Algorithmic design of feasible trajectories, 2002. 2.1

[56] Sung-hee Lee and Ambarish Goswami. Ground reaction force control at each foot : A momentum-based humanoid balance controller for non-level and non-stationary ground. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3157–3162, 2010. 3.1.1

[57] Z. Li, W. W. Melek, and C. Clark. Decentralized robust control of robot manipulators with harmonic drive transmission and application to modular and reconfigurable serial arms. *Robotica*, 27(2):291–302, March 2009. 1.3, 2.4.4

[58] Chenggang Liu and Christopher G. Atkeson. Standing balance control using a trajectory library. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009. 2.1, 3.1.1

[59] Schillings A. M., Van Wezel B.M. H., Mulder T. H., and Duysens J. Muscular responses and movement strategies during stumbling over obstacles. *Journal of Neurophysiology*, 83:2093–2102, 2000. 3.6.3

[60] Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Transactions on Graphics*, 28(3):1, 2009. ISSN 07300301. doi: 10.1145/1531326.1531386. 3.1.1

[61] Thijs Mandersloot, Martijn Wisse, and Christopher G. Atkeson. Controlling velocity in bipedal walking: A dynamic programming approach. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2006. 2.1

[62] Tad McGeer. Passive dynamic walking. *Int. J. Rob. Res.*, 9(2):62–82, mar 1990. 1.3

[63] M. McNaughton. CASTRO: robust nonlinear trajectory optimization using multiple models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 177–182, 2007. 2.3.1

[64] K. Mombaur, J.P. Laumond, and E. Yoshida. An optimal control model unifying holonomic and nonholonomic walking. In *Proceedings of the IEEE International Conference on Humanoid Robots*, 2008. 2.1

[65] Jun Morimoto, Garth Zeiglin, and Christopher G. Atkeson. Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings of Intl. Conference on Intelligent Robots and Systems*, 2003. 2.3.1, 2.3.5

[66] Mitsuharu Morisawa, Kensuke Harada, Shuuji Kajita, Shinichiro Nakaoka, Kiyoshi Fujiwara, Fumio Kanehiro, Kenji Kaneko, and Hirohisa Hirukawa. Experimentation of humanoid walking allowing immediate modification of foot place based on analytical solution. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3989–3994, October 2007. 3.1.1

[67] Remi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49, Numbers 2/3:291–323, 2002. 2.1

[68] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer*

Journal, 7(4):308–313, 1965. 3.4.2

[69] Arnab Nilim, Laurent El Ghaoui, and Vu Duong. Robust dynamic routing of aircraft under uncertainty. In *Proceedings of Digital Avionics Systems Conference*, 2002. 2.3.1

[70] Koichi Nishiwaki and Satoshi Kagami. High frequency walking pattern generation based on preview control of ZMP. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2667–2672, May 2006. 2.1, 3.1.1

[71] R. H. Nyström, J. M. Böling, J. M. Ramstedt, H. T. Toivonen, and K. E. Häggblom. Application of robust and multimodel control methods to an ill-conditioned distillation column. *Journal of Process Control*, 12:39–53, 2002. 2.3.1

[72] Ill-Woo Park, Jung-Yup Kim, Jungho Lee, and Jun-Ho Oh. Mechanical design of humanoid robot platform khr-3 (kaist humanoid robot-3: Hubo). In *Proc. IEEE/RAS Int. Conf. on Humanoid Robots*, pages 321–326, 2005. 1.3

[73] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Comput.*, 3(2):246–257, 1991. 2.4.1

[74] Gail B Peterson. A day of great illumination: B. F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, 82:317–328, 2004. 2.3.6

[75] Y. Piguet, U. Holmberg, and R. Longchamp. A minimax approach for multi-objective controller design using multiple models. *International Journal of Control*, 72(7-8):716–726, 1999. 2.3.1

[76] G. Pratt and M. Williamson. Series elastic actuators. In *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 399–406, Los Alamitos, CA, USA, 1995. IEEE Comput. Soc. Press. 1.3

[77] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami. Capture Point: A Step toward Humanoid Push Recovery. In *Proceedings of the International Conference on Humanoid Robots*, pages 200–207. IEEE, December 2006. ISBN 1-4244-0199-2. doi: 10.1109/ICHR.2006.321385. 3.1.1, 3.7.1

[78] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 1994. 2.1, 2.4.1

[79] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978. 2.1

[80] Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *NIPS*, 2003. 2.4.1

[81] Carl Edward Rasmussen and Chris Williams. *Gaussian processes for machine learning*. the MIT Press, 2006. 2.4.2

[82] John Rust. Using randomization to break the curse of dimensionality. *Econometrica*, pages 487–516, 1997. 2.1

[83] Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Appl. Intell.*, 17(1):49–60, 2002.

2.4.1

[84] Jeff G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *in Neural Information Processing Systems 9*, pages 1047–1053. The MIT Press, 1996. 2.4.4

[85] J. Shinar, V. Glizer, and V. Turetsky. Solution of a linear pursuit-evasion game with variable structure and uncertain dynamics. In S. Jorgensen, M. Quincampoix, and T. Vincent, editors, *Advances in Dynamic Game Theory - Numerical Methods, Algorithms, and Applications to Ecology and Economics*, volume 9, pages 195–222. Birkhauser, 2007. 2.3.1

[86] Jennie Si, Andrew G. Barto, Warren B. Powell, and Don Wunsch, editors. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004. 2.1, 2.3.1

[87] Marc C. Steinbach. Robust process control by dynamic stochastic programming. *Proceedings in Applied Mathematics and Mechanics (PAMM)*, 4(1):11–14, 2004. 2.3.1

[88] Benjamin Stephens. Humanoid Push Recovery. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2007. 3.1.1

[89] Benjamin Stephens. *Push REcovery Control for Force-Controlled Humanoid Robots*. PhD thesis, CMU, 2011. 4.3.1, 4.3.2

[90] Benjamin J. Stephens. Dynamic balance force control for compliant humanoid robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. Available online at www.cs.cmu.edu/˜bstephe1/papers/iros10.pdf. 3.4.3

[91] Mike Stilman, Christopher G. Atkeson, James J. Kuffner, and Garth Zeglin. Dynamic programming in reduced dimensional spaces: Dynamic planning for robust biped locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2399–2404, 2005. 2.1

[92] H. T. Su and T. Samad. Neuro-control design: Optimization aspects. In O. Omidvar and D. L. Elliott, editors, *Neural Systems For Control*, pages 259–288. Academic Press, 1997. 2.3.1

[93] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. 2.4.4

[94] A. Takanishi, T. Takeya, H. Karaki, and I. Kato. A control method for dynamic biped walking under unknown external force. In *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, volume 29, pages 795–801. IEEE, 1990. 1.3, 1.4, 3.1.1

[95] Toru Takenaka, Takashi Matsumoto, Takahide Yoshiike, Tadaaki Hasegawa, Shinya Shirokura, Hiroyuki Kaneko, and Atsuo Orita. Real time motion generation and control for biped robot -4th report: Integrated balance control. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1601–1608. Ieee, October 2009. ISBN 978-1-4244-3803-7. doi: 10.1109/IROS.2009.5354522. 3.1.1

[96] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *Int. J. Rob. Res.*, 29(8):1038–1052, July 2010. 2.1

[97] Evangelos Theodorou, Yuval Tassa, and Emo Todorov. Stochastic differential dynamic programming. In *Proceedings of American Control Conference*, 2010. 2.3.1

[98] Emanuel Todorov and Yuval Tassa. Iterative local dynamic programming. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 90–95, 2009. 2.1

[99] Michiel van de Panne, Eugene Fiume, and Zvonko G. Vranesic. A controller for the dynamic walk of a biped across variable terrain. In *Proceedings of the 31st Conference on Decision and Control*, pages pp. 2668–2673, December 1992. 2.1, 3.1.1

[100] A. Varga. Optimal output feedback control: a multi-model approach. In *IEEE International Symposium on Computer-Aided Control System Design*, pages 327–332, 1996. 2.3.1

[101] Sethu Vijayakumar, Tomohiro Shibata, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Autonomous Robot*, 2003. 2.4.1

[102] Pierre-Brice Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2006. 2.1, 3.1.1, 3.1.1

[103] Pierre-Brice Wieber and Christine Chevallereau. Online adaptation of reference trajectories for the control of walking systems. *Robotics and Autonomous Systems*, 54(7):559–566, 2006. 2.1, 3.1.1

[104] Wee Chin Wong and Jay H. Lee. Postdecision-state-based approximate dynamic programming for robust predictive control of constrained stochastic processes. *Industrial & Engineering Chemistry Research*, 50(3):1389–1399, 2011. 2.3.1

[105] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: simple biped locomotion control. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 105, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1275808.1276509. 3.1.1

[106] Matt Zucker, J. Andrew Bagnell, Christopher G. Atkeson, and James Kuffner. An optimization approach to rough terrain locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010. 3.5.3