Pre-Computation for Controlling Character Behavior in Interactive Physical Simulations

by

Marco Jorge Tome da Silva

Sc.B., Brown University (2001)
S.M., Massachusetts Institute of Technology (2008)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
August 6, 2010

Certified by .....
Jovan Popović
Associate Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Pre-Computation for Controlling Character Behavior in Interactive Physical Simulations

by

Marco Jorge Tome da Silva

Submitted to the Department of Electrical Engineering and Computer Science
on August 6, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The development of advanced computer animation tools has allowed talented artists to create digital actors, or characters, in films and commercials that move in a plausible and compelling way. In interactive applications, however, the artist does not have total control over the scenarios the character will experience. Unexpected changes in the environment of the character or unexpected interactions with dynamic elements of the virtual world can lead to implausible motions. This work investigates the use of physical simulation to automatically synthesize plausible character motions in interactive applications.

We show how to simulate a realistic motion for a humanoid character by creating a feedback controller that tracks a motion capture recording. By applying the right forces at the right time, the controller is able to recover from a range of interesting changes to the environment and unexpected disturbances. Controlling physically simulated humanoid characters is non-trivial as they are governed by non-linear, non-smooth, and high-dimensional equations of motion. We simplify the problem by using a linearized and simplified dynamics model near a reference trajectory.

Tracking a reference trajectory is an effective way of getting a character to perform a single task. However, simulated characters need to perform many tasks form a variety of possible configurations. This work also describes a method for combining existing controllers by adding their output forces to perform new tasks. This allows one to reuse existing controllers. A surprising fact is that combined controllers can perform optimally under certain conditions.

These methods allow us to interactively simulate many interesting humanoid character behaviors in two and three dimensions. These characters have many more degrees of freedom than typical robot systems and move much more naturally. Simulation is fast enough that the controllers could soon be used to animate characters in interactive games. It is also possible that these simulations could be used to test robotic designs and biomechanical hypotheses.

Thesis Supervisor: Jovan Popović
Title: Associate Professor

Acknowledgments

Research is an uncertain endeavor, and this uncertainty can be a great source of stress for a graduate student. As I take a moment to reflect on my graduate studies, I am happy to say that I felt way more happiness than stress during my time here. This relation holds not because of any extraordinary research skills on my part but because of the tremendous amount of help I got from the people around me.

Jovan was a master motivator, confidence builder, and adviser. Initially, I struggled to hit my stride in graduate school and struck out on a few initial ideas. At just the right time, Jovan gave me a needed push in the right direction. He pushed me to pursue linear quadratic regulators as an avenue for interactive physical simulation of characters. After getting the ball rolling, Jovan kept me going with a combination of exuberance and encyclopedic knowledge of the field. Jovan's positive attitude made it easier to feel like progress was being made. Anytime I did feel stuck at a roadblock, Jovan always helped me find a way around each obstacle. I want to thank Jovan for inspiring me to pursue the work in this thesis.

When Jovan left MIT, I asked Fredo to be a local adviser. Given that I was not close to finishing at the time, I felt it was important to have somebody at MIT who would help me with any issues that might arise (a special thank you to Bill Freeman for helping to establish this new working arrangement). Fredo proved to be very helpful indeed. The first thing he did was to give me confidence to continue to pursue my work instead of starting over on some other topic. He then provided me with invaluable insights on the work on linear Bellman combination described in this thesis. Every student in the graphics group was amazed with Fredo's capacity to provide feedback on projects he was not directly involved. I feel privileged to have worked directly with Fredo on a research project.

In graduate school, I learned more outside of the classroom than in it. One labmate in particular was instrumental in helping me complete the work in this thesis. Yeuhi Abe and I spent countless hours exchanging ideas on physically-based character animation. He also worked directly on some of the work in this thesis. He developed

the initial implementation of the quadratic programming approach that is vital to accurate tracking of motion capture data. He also helped me pre-process the motion data used in the experiments. Yeuhi and I taught each other a lot about our field in countless discussions that ultimately had to come to an end with a phone call from my wife!

Much of the work in this thesis is related to and takes inspiration from work in robotics. My local robotics expert was Russ Tedrake. Russ had significant input on the linear quadratic tracking approach described in this paper. I also took Russ' class on under-actuated robotics. Linear Bellman combination started as a final project in his class and his early feedback helped shape that project in its formative stages.

A stereotypical depiction of graduate student life is that of a grizzled hermit toiling in solitude. I think it holds for a lot of students but thankfully not for the students in the graphics group at MIT. I had never had such a friendly relationship with co-workers. Beyond making graduate school fun, the graphics group also helped me complete my work. The gave me early feedback on papers and brainstorming sessions. They helped me collect data. Tom Buehler helped me produce videos to demonstrate the methods in this work. Britton Bradley helped me navigate the peculiars of MIT's inner workings.

My interest in research was stoked as an undergraduate by David Laidlaw. Under his guidance, I was able to complete a couple of modest research projects. The satisfaction that came with this work stuck in my memory and ultimately made me come back to graduate school. David also had a great role in helping me get into MIT. In addition to writing a recommendation letter, he helped me land my first job. The work I did there was enough to get Jovan interested in my application.

Returning to graduate school after four years of working was not an easy decision. I am indebted to my wife, Lisa, for pushing me to pursue this challenge. Lisa put up with a lot while I toiled away in lab. I spent a large portion of the first two years in grad school sick with one form of cold or another. Lisa was always there to bring me back to health with love, kindness, and patience. That patience served her well while I spent countless nights and weekends working. I am eternally grateful for her

unwavering support during this endeavor.

Finally, I want to thank my family. One reason I chose to study at MIT was its proximity to my family. I would never have gotten to this point without their tireless efforts to support me. It is a lot easier to focus on graduate studies when you know that your family is behind you every step of the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of advanced computer animation tools has allowed talented artists to create digital actors, or characters, in films and commercials that move in a plausible and compelling way. Remarkably, the technology has reached a point where digital characters routinely share the screen with real actors. Even in completely digital productions, artists exploit pseudo-physical principles to maintain an illusion of life and engage the viewer [47].

In interactive applications such as video games or training simulators, however, the artist has limited control over the final animation (Figure 1-1). Typically, a game-engine uses *kinematic* techniques to stitch together or blend pre-animated clips to respond to high-level direction provided by a user or computer AI. As a result, animation in games can be quite repetitive. Worse, characters in interactive scenarios can quickly get into situations that were not explicitly planned for by the animator. Unexpected changes in the environment of the character or unexpected interactions with dynamic elements of the virtual world can lead to implausible motions. For example, consider the situation where an animator has created or captured a single animation of a character walking on flat ground (Figure 1-2). In an interactive setting, that character may encounter varied terrain or interact with other simulated objects. Kinematic methods fail to do a convincing job of adapting the animator's work to these new situations.

This work investigates the use of physically-based simulation to animate char-

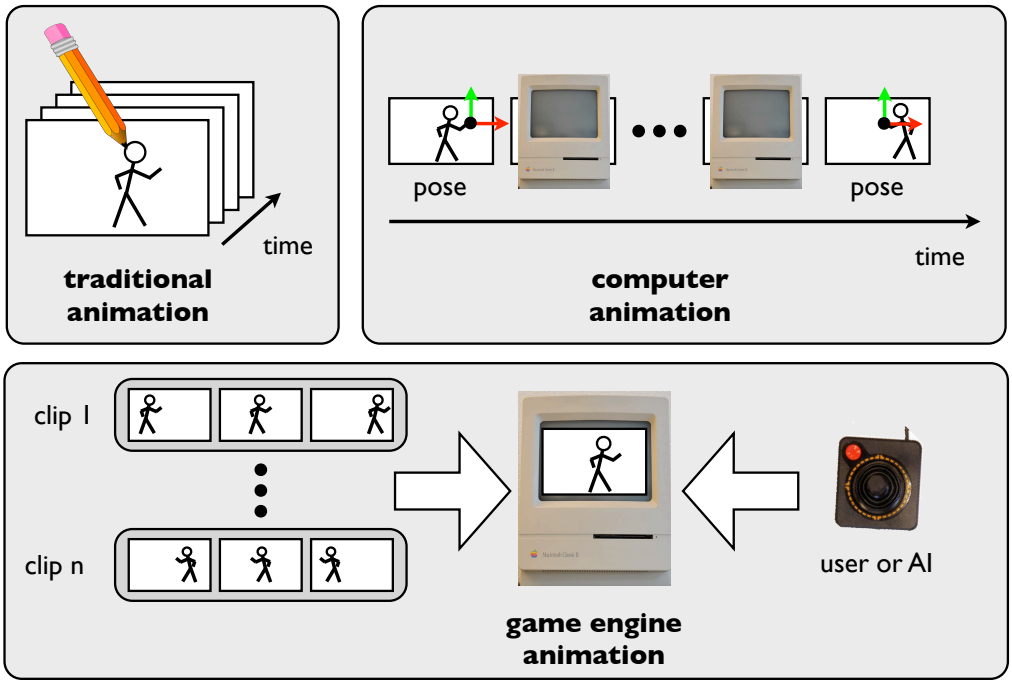

Figure 1-1: By drawing everything that's seen on the screen a traditional animator has total control over the final animation. The computer animator gives up some control to save a lot of effort. The computer animator creates key poses through a user interface and the computer interpolates those poses using an algorithmic scheme. In game animation, these animations, which may originate from motion capture recordings, are packaged into clips. A game engine than stitches these clips together or blends them to match high level input from a user or AI to produce the final animation.

acter behaviors and aims to address some of the limitations of kinematic character animation used in today's interactive applications. Consider again, our walking character example. Now, instead of simply playing back the motion, we instead simulate the motion by applying forces to the character's limbs. Since motion is generated procedurally using physics, the character will interact consistently and plausibly with changes in terrain and dynamic disturbances. Furthermore, we should be able to vary the motion by, as an example, changing the character's mass properties. It is worth noting that many of these same advantages are already being exploited for the animation of *passive* dynamics systems such as fluids, cloth, and rigid bodies [35, 13, 12] in films and interactive applications. Character behaviors on the other hand are not often simulated as it is difficult to simulate a character that does something besides fall.

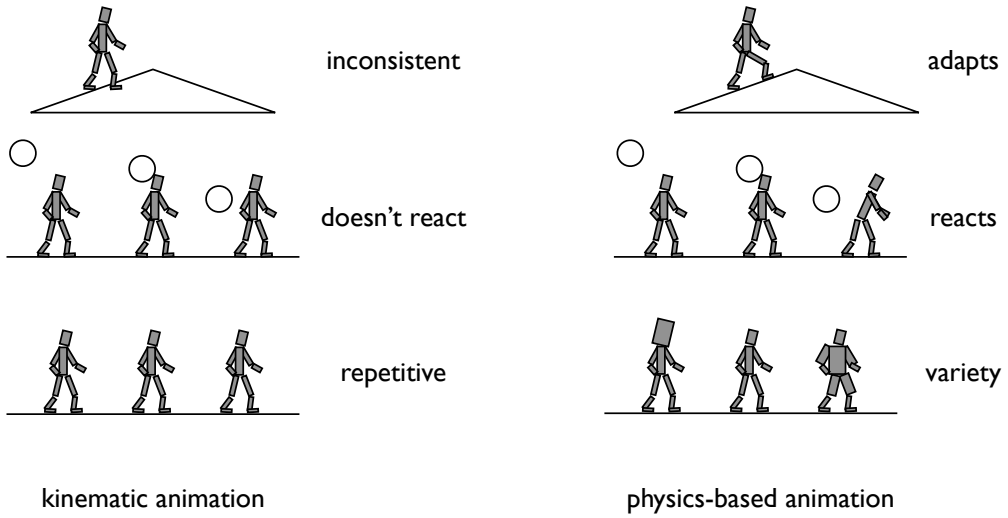

Figure 1-2: Physically-based character animation has a number of advantages over standard kinematic methods. A physically-simulated character adapts to changes in the environment, reacts to dynamic disturbances, and offers an infinite range of possible motions. This figure illustrates this point with some examples. Trying to replay a motion that was animated walking on flat ground will lead to an implausible result if the motion is replayed on a slope. If you simulate a ball hitting the character, the character won't react if you are simply replaying a kinematic method. Finally, using pre-animated motions leads to repetitive motion, whereas, you could easily get new motions in a physics based approach simply by changing the mass properties of your character.

Simulating a character behavior presents a number of challenges, some of which are familiar to researchers in robotics. Characters, like many robots, are governed by non-linear, possibly non-smooth equations of motion. As opposed to passive systems, these equations of motion are functions of the character's state *and* a control input. In order to get a character, or a robot, to move in a useful way as opposed to just fall down, someone has to algorithmically specify a controller that uses this control input to coordinate the character's movement (Figure 1-3). Typical characters are under-actuated, meaning they have more degrees of freedom than control inputs. Under-actuation makes it difficult to design robust and efficient controllers. In addition, interesting characters in graphics applications are typically very high-dimensional. Even a simplified linked rigid body model of a humanoid character has on the order of fifty degrees of freedom, which, when you include velocities, leads to a hundred dimensional state space. In graphics, the problem can be even worse as it's not

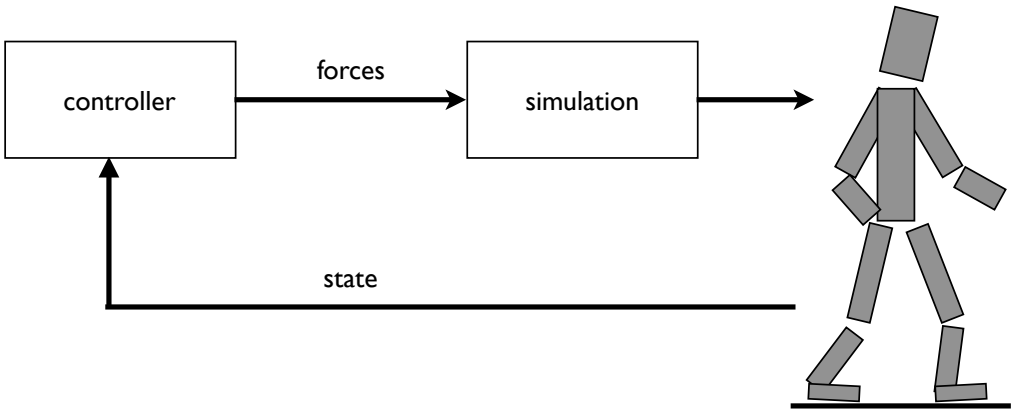

Figure 1-3: Simulating a character behavior requires a controller. A feedback controller examines the current state of the character to compute forces which, when fed into simulation, help the character achieve some goal. Designing robust controllers for high-dimensional, non-linear dynamical systems is difficult. The task is even harder when you also want the controller to produce life-like motions.

unusual to have characters with thousands of degrees of freedom; characters modeled as deformable objects or fluids. Finally, whereas robotics is mostly concerned with robustness, the style of the motion is important for graphics applications. For a physically-based approach to be useful to graphics, an animator should retain some say as to how the character moves. Beyond graphics, matching a particular style of motion is also important for biomechanics applications if the goal is to explain the motion of a particular biological creature.

This work approaches the problem of controlling a character from an optimal control perspective. Many existing approaches to controlling characters in graphics rely on hand-crafted control schemes. In contrast, optimal control formulates an optimization problem that searches for controls that minimize some metric of controller performance while satisfying constraints [88]. This approach has many advantages over more manual approaches that mainly rely on user intuition to control the character. One advantage is that it is easier for a user to specify a motion via constraints than it is to achieve those constraints by hand-tuning a parameterized controller. Also, by incorporating look-ahead knowledge of where the character is going, an

optimal control approach can achieve life-like and efficient motions. Unfortunately, solving non-linear, non-convex optimal control problems is computationally expensive and brittle. As a result, optimal control has mainly been used as a tool for *offline* animation and not in interactive applications [88, 60, 68, 53].

The contribution of this work is to show how one can use pre-computation and some simplifying assumptions to apply optimal control principles to controllers in interactive applications. First, we show how to derive controllers from motion capture trajectories by pre-computing them from a simplified optimal control problem. Then, we show how existing controllers can be combined to be reused for new tasks. A surprising result is that combined controllers can be optimal for new tasks under certain conditions. Key to this result is an alternative formulation of the optimal control problem which leads to a linear condition for optimality. Beyond, combination this work also shows that this alternative formulation allows for analytical and more efficient solution techniques.

Using these methods, we were able to simulate a variety of humanoid character behaviors including stepping and jumping motions. The resulting motion is often as life-like as motion capture. Since we simulate the motion, we were able to make interesting changes to the environment such as varying the terrain and introducing other dynamic objects. In addition, we were able to combine controllers to interpolate controller goals, allow for multiple goals, and expand the region of state space from which we could successfully simulate. As a concrete example, we combined stepping controllers to take steps of different lengths.

The characters we simulate typically have many more degrees of freedom than robot systems and move much more naturally. Simulation is fast enough that the controllers could soon be used to animate characters in interactive games. This will improve realism and reduce the burden on animators and data acquisition. In addition, our approach to combining controllers often outperforms simpler approaches and could be used to create controllers that traverse constrained environments such as an environment where a character must step in certain regions of the floor.

In nature, people, animals, even bacteria have evolved highly robust strategies for

moving around their environments. These strategies exploit the laws of physics to make efficient use of the organism's resources and rely on feedback from the environment to keep the organism safe. Finally, many organisms show a remarkable ability to modify a strategy over time to improve performance and respond to changes. Identifying movement strategies is a fundamental scientific challenge with broad implications for many fields including biomechanics, neuroscience, and artificial intelligence. Reproducing a motion in simulation is one way to identify a movement strategy and is a first step to designing agile robots that move as efficiently as biological creatures. While automatic synthesis of character motion is a grand goal of computer animation, this work is also a step towards a general simulation toolbox that can be used to answer questions of importance beyond computer animation.

1.1 Organization

This work first presents some background material on simulating character behaviors and on relevant concepts from optimal control theory. Then, the exposition is organized as follows.

Pre-computing Feedback Policies Chapter 3 of this proposal describes work on pre-computing feedback controllers for legged character motions. By linearizing the dynamics of the character around a given trajectory, we can simplify the task of staying close to that trajectory.

The Linear Bellman Equation Chapter 4 reviews an optimal control formulation that leads to a linear condition for optimality called the linear Bellman equation.

Solving the Linear Bellman Equation Chapter 5 discusses ways to solve the linear Bellman equation analytically and numerically. We show when the linear Bellman equation can be solved analytically. We also show that solving the linear Bellman equation has certain numerical advantages over solving standard Bellman equations for the same optimal control problem.

Combining Controllers Chapter 6 describes work on adding pre-computed policies together to achieve new tasks such as interpolating existing goal states or reaching a goal state from multiple starting points. Using the linear Bellman equation, we show under what conditions combined controllers are actually optimal for new tasks.

Chapter 2

Background

This chapter introduces some necessary background on animation using physical simulation and optimal control. In physically based animation, a *character* is modeled as a dynamical system whose state varies in time according to differential equations. Below, we will describe in detail characters modeled as linked rigid bodies. We will see that the equations of motion depend on a control input vector. This control needs to be set somehow for the character to do something useful. As described in the second part of this chapter, optimal control theory is one way to derive useful controllers for characters. We focus on optimal control as it is the starting point for the methods to be described later in this work. This chapter offers only a brief overview, but there are good textbooks that cover rigid body dynamics [33] and optimal control theory [19].

2.1 Physically Simulating Character Behavior

In computer graphics, the pose a character takes on the screen is a function of its internal degrees of freedom called its kinematic state $\boldsymbol{q} \in \mathbb{R}^n$. A point $y \in \mathbb{R}^3$ on the character can be described using a forward kinematics function $y = g(\boldsymbol{q})$. In an animation, the kinematic state varies with time $\boldsymbol{q}(t)$ causing the character to move. In traditional computer animation, the animator controls this movement by specifying a spline for each component of $\boldsymbol{q}$. The kinematic state can also be specified using

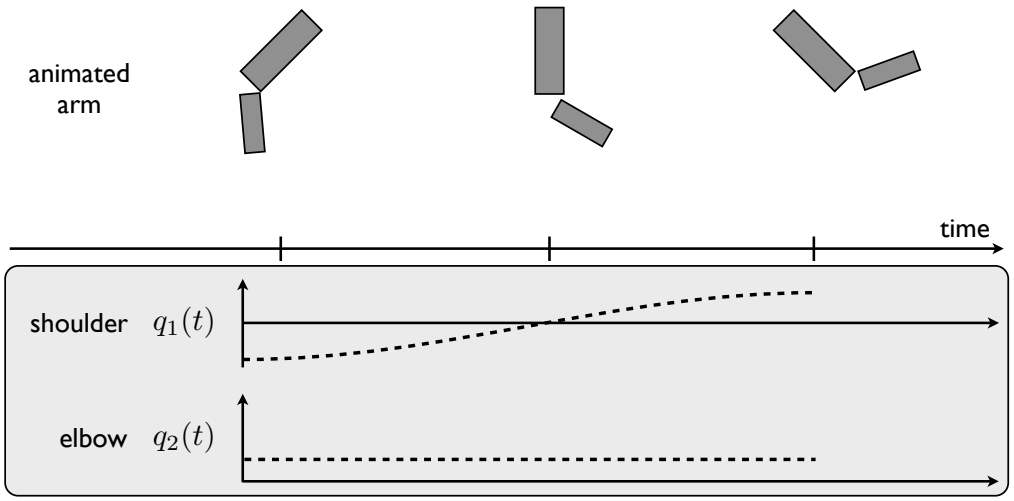

Figure 2-1: The pose a character takes on the display is a function of its kinematic state. Here we see an arm modeled as two rigid links. To animate the arm, an artist specifies a function for each degree of freedom of the arm over time. This function can be defined using a smooth spline or can come from motion capture data.

motion capture data (Figure 2-1).

In physically-based animation, the kinematic state of the character varies in time according to second order differential equations. By stacking the kinematic state and velocity of each degree of freedom into a single vector

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix},$$

we can state the general equations of motion for any dynamical system as a first-order system $\dot{x} = f(x)$. To get the state of a *passive* dynamical system at some time, we integrate this equation forward from an initial condition

$$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \int_0^t f(\boldsymbol{x}(\tau))\,\mathrm{d}\tau.$$

The system is deemed passive because once the initial conditions are given the only thing that determines where it will be at some future time are the dynamics equations f. Depending on the numerical integration scheme used and the dynamical system being simulated, the particulars of implementing this equation vary. Though the

simulation of passive dynamical systems is still an active area of research, this general setup is regularly used to simulate fluids, cloths, deformable bodies, and other passive systems in computer graphics applications [35, 13, 14].

2.1.1 Linked Rigid Bodies

The characters in this work, typically bipedal humanoid characters, are modeled as linked rigid bodies. For example, we create links for the head, forearm, upper leg, and so on giving each link inertial properties that roughly approximate the inertial properties of a real person. Specifying inertial properties requires a little bit of extra work to model the character than would normally be the case. Nevertheless, it is fairly easy to set these properties given the shape of the link and assuming a uniform density for the character. Then, the links are joined together using constraints that model the joints of a real person. Modeling skeleton geometry and connectivity is a requirement for kinematic animation as well.

One could represent the state of the character as the global position, orientation, linear, and angular velocity of each rigid link. This would require twelve times the number of links numbers for a three-dimensional character. Since the links are constrained together by joints, however, we can represent the state using many fewer numbers. The parameterization used to describe the degrees of freedom of the character, called generalized coordinates, determine the equations of motion via Lagrangian mechanics. The recipe's first step is to write down the Lagrangian of your system as a function of the generalized coordinates and their time derivatives. The next step is to apply the stationary action principle which yields the equations of motion in generalized coordinates.

More formally, the Lagrangian of a system is defined as the kinetic energy minus the potential energy

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = K(\boldsymbol{q}, \dot{\boldsymbol{q}}) - P(\boldsymbol{q}).$$

The stationary action principle from classical mechanics yields the following equations

of motion for a passive system

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\delta L}{\delta \dot{\boldsymbol{q}}_i} - \frac{\delta L}{\delta \boldsymbol{q}_i} = 0.$$

One can model a controlled system by introducing generalized torques $\boldsymbol{u}_i$ with the relation

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\delta L}{\delta \dot{\boldsymbol{q}}_i} - \frac{\delta L}{\delta \boldsymbol{q}_i} = \boldsymbol{u}_i$$

indicating the instantaneous change in generalized momentum.

To make the recipe more concrete, we can derive the equations of motion for a bead on a circular wire in generalized coordinates. We will assume the wire has a unit radius and is centered at a point one unit above the ground. Though the bead lives in a three-dimensional space, it only has one-degree of freedom since it is constrained to live on a wire. This degree of freedom is aptly described using the angular position of the bead θ. The kinetic energy of the bead is

$$K(\theta, \dot{\theta}) = \frac{1}{2} m \dot{\theta}^2$$

where m is the mass of the bead. The potential energy is

$$P(\theta) = mg(1 + \sin(\theta))$$

where g is the constant of gravitational acceleration. Applying our recipe, we get

$$m\ddot{\theta} - mg\cos(\theta) = u.$$

The bead is a simple oscillatory system, much like a pendulum. Note that, if we are given the control input u we can solve for the angular acceleration of the bead $\ddot{\theta}$ and vice versa.

More generally, the kinetic energy of a linked rigid body can be written as

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^\top \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ maps generalized velocities $\dot{\boldsymbol{q}}$ to momentum. Plugging this into the Lagrangian recipe gives us

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \frac{\delta P(\boldsymbol{q})}{\delta \boldsymbol{q}} = \boldsymbol{u}. \tag{2.1}$$

This equation is the equivalent to $f = ma$ from Newtonian physics. The first term $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}$ is mass times acceleration. The gradient of the potential $\frac{\delta P(\boldsymbol{q})}{\delta \boldsymbol{q}}$ are the generalized forces due to gravity. The middle term $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ are forces that arise due to rotational effects. These forces are balanced on the right hand side by our control forces $\boldsymbol{u}$.

To forward simulate, we must integrate the system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$. First, we must solve for the generalized accelerations $\ddot{\boldsymbol{q}}$ given the controls $\boldsymbol{u}$ in Equation (2.1). This involves inverting the inertial matrix $\boldsymbol{M}$. A straightforward forward dynamics algorithm does just that, giving a running time of $O(n^3)$. In practice, we use recursive $O(n)$ algorithms [32]. It is often useful to solve the inverse dynamics problem, i.e. compute the control forces given the generalized accelerations. Again, a straightforward implementation of inverse dynamics gives a $O(n^2)$ algorithm but a recursive formulation can be applied to give a $O(n)$ algorithm [32].

2.1.2 External Forces

The forces on the right hand side of Equation (2.1) are generalized forces. If we want to simulate applying a force $\boldsymbol{f}$ to the character at some point, we have to map a three-dimensional force into a generalized force $\boldsymbol{u}$. This mapping can be derived by relating the work done by the point force with the work done by the generalized forces. These two quantities must be equal by the principle of virtual work: $\boldsymbol{f}^\top \delta \boldsymbol{x} = \boldsymbol{u}^\top \delta \boldsymbol{q}$. Defining the Jacobian of the contact point $\boldsymbol{J}(\boldsymbol{q}) = \frac{\delta \boldsymbol{x}}{\delta \boldsymbol{q}}$, we see that $\delta \boldsymbol{q} = \boldsymbol{J}(\boldsymbol{q})\delta \boldsymbol{x}$ giving the desired mapping

$$\boldsymbol{u} = \boldsymbol{J}(\boldsymbol{q})^\top \boldsymbol{f}.$$

With external forces, Equation (2.1) becomes

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \frac{\delta P(\boldsymbol{q})}{\delta \boldsymbol{q}} = \boldsymbol{u} + \sum_i \boldsymbol{J}_i(\boldsymbol{q})^\top \boldsymbol{f}_i. \tag{2.2}$$

2.1.3 Contacts

In simulation, external forces arise due to collisions between objects. The simulation's job is to find contact forces that keep the objects from interpenetrating and resolve any existing penetration. In addition, the simulation can model phenomena like friction and restitution using appropriately chosen contact forces. There are a number of methods for computing contact forces in the literature [12, 58]. In practice, many simulators skip forces and directly apply impulses to velocities to model contact [73, 6]. Efficient and accurate contact resolution remains an active area of research [45]. Here, we will focus on two contact models used in this work: penalty forces and analytical contact forces.

Penalty-based Contact

A penalty force contact model creates imaginary springs between colliding objects. The simulation checks designated vertices to see if they are penetrating another object. If so, a virtual spring is anchored to the colliding point $\boldsymbol{p}_c$ and a nearby point on the object $\boldsymbol{p}_o$. The contact force is then computed as

$$\boldsymbol{f}_c = k_s(\boldsymbol{p}_o - \boldsymbol{p}_c) - k_d(\dot{\boldsymbol{p}}_o - \dot{\boldsymbol{p}}_c)$$

and applied as an external force to each object as described in the previous section. The penalty method is very intuitive and scales well to complex geometry with many contacts. It is not often used in practice, however, due to a number of drawbacks. First, some amount of interpenetration is unavoidable. Depending on how the constants k_s and k_d are set and the time step, there will be more or less penetration. Increasing the stiffness of the penalty spring reduces penetration but may impact the stability of the simulation. Finally, it is not straightforward to incorporate Coloumb

friction using a penalty based method. In recent years, researchers have extended the simple penalty method described here to include frictional dynamics and improve stability [91].

Analytical Contact

Analytical contact methods use constrained optimization to insure that no interpenetration occurs. Here, we briefly describe an analytical contact model [12] and show how to incorporate a Coloumb friction model.

If a point comes into contact with some geometry, it either must accelerate away from the geometry in the normal direction or a non-zero force is needed to make the acceleration zero in the normal direction. If we denominate the acceleration of the point $\boldsymbol{a}$ and the contact force as $\boldsymbol{f}_c$, we require $\boldsymbol{a} \cdot \boldsymbol{n} \geq 0$, $\boldsymbol{f}_c \cdot \boldsymbol{n} \geq 0$, and, since constraint forces should do no work, $(\boldsymbol{a} \cdot \boldsymbol{n})(\boldsymbol{f}_c \cdot \boldsymbol{n}) = 0$. The normal of the contact is denominated by $\boldsymbol{n}$.

Recalling our definition for the Jacobian of a point, we see that the velocity of a point on the character can be expressed as $\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$. Differentiating this by time, we get an expression for the acceleration of the contact point

$$\boldsymbol{a} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}. \tag{2.3}$$

From Equation (2.2), we see that, at any instant in time, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{f}_c$ are related by an affine expression. Substituting this into Equation (2.3) and introducing some simplifying variables we get

$$\boldsymbol{a} = \boldsymbol{A}\boldsymbol{f}_c + \boldsymbol{b}.$$

Putting this expression into our constraints, we see that two of the constraints are linear and one constraint is quadratic in the unknown contact forces $\boldsymbol{f}_c$. Therefore, we can find the contact forces numerically by solving a quadratic program (QP) where we make the quadratic constraint the objective and retain the other linear conditions

as constraints

$$\begin{aligned} \min_{\boldsymbol{f_c}} \quad & (\boldsymbol{f}_c \cdot \boldsymbol{n})((\boldsymbol{A}\boldsymbol{f_c} + \boldsymbol{b}) \cdot \boldsymbol{n}) \\ \text{subject to} \quad & (\boldsymbol{A}\boldsymbol{f_c} + \boldsymbol{b}) \cdot \boldsymbol{n} \geq 0 \\ & \boldsymbol{f}_c \cdot \boldsymbol{n} \geq 0. \end{aligned}$$

In practice, it's more efficient to find the contact forces by formulating an equivalent linear complementarity problem [12]. We'll use QP's again later in this work so we'll stick to this formulation.

The Coloumb friction model states that the ratio of any tangential friction forces to the contact force in the normal direction must be less than some constant coefficient of friction μ:

$$\frac{||\boldsymbol{f}_c - (\boldsymbol{f}_c \cdot \boldsymbol{n})\boldsymbol{n}||}{\boldsymbol{f}_c \cdot \boldsymbol{n}} \leq \mu.$$

Geometrically, this means that the contact force lies within some cone around the contact normal. The ratio constraint is a non-linear expression so it cannot directly be used as a constraint in a QP formulation. However, a simple trick allows us to model Coloumb friction with a linear constraint. We can approximate the friction cone with a polyhedral pyramid. Any vector in that pyramid can be expressed as a positive linear combination of four basis vectors. Mathematically, we express the contact force in a friction cone basis $\boldsymbol{f}_c = \boldsymbol{V}\boldsymbol{\lambda}$ where we require $\boldsymbol{\lambda} \geq 0$. This trick was first used in graphics for offline optimizations [30].

2.2 Control

At this point, all the tools needed to simulate a character modeled with passive linked rigid bodies have been described. Simply set the control to zero $\boldsymbol{u} = 0$ and run a forward dynamics algorithm from some initial condition. However, under gravity, all that the character will do is fall down. Many of today's interactive games tout this feature as rag-doll physics. Getting the character to do something besides falling,

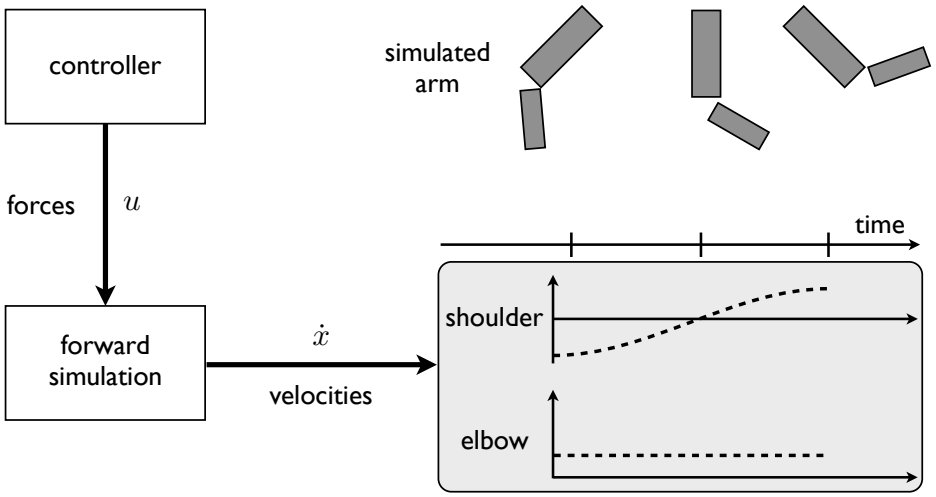

Figure 2-2: To simulate an arm that does something besides swing aimlessly, you need a controller that feeds meaningful forces to a simulation. The simulation then computes forward dynamics to update the state of the arm. Controlling an arm is fairly straightforward if you can exert a torque at the shoulder and elbow.

requires setting the control input in a meaningful way (Figure 2-2).

A character described by Equation (2.1) is *fully-actuated*; it has as many degrees of freedom as control inputs ($\ddot{\boldsymbol{q}} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^n$). Controlling fully-actuated characters is a pretty straightforward endeavor since we can easily compute generalized forces that give us a desired acceleration from Equation (2.1). This is the principle behind many industrial robots, for example. The control problem gets a little more interesting if we want to maximize some performance measure or perform multiple tasks simultaneously [4].

Unfortunately, interesting characters, biological creatures, and robots are *under-actuated*; they don't have as many control inputs as degrees of freedom in some or all configurations ($\ddot{\boldsymbol{q}} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^m$ with $m < n$). This means that we can no longer simply compute torques given desired accelerations. Later, we'll see that we can exploit contact forces in Equation (2.2) to achieve desired accelerations in a least squares sense, but first, let's review some existing approaches to controlling under-actuated characters in computer graphics.

2.2.1 Control in Computer Graphics

Automatic synthesis of virtual characters through physically based simulation is a long standing goal of computer animation and robotics. Indeed, the first SIGGRAPH conference featured work that modeled a two-dimensional human to be used as a crash test dummy and proposed it could be used for other activities such as walking [86]. As we have described active movements like walking, however, require a control algorithm to apply the right forces at the right time. Finding these controllers has proven difficult, though, and researchers in robotics and computer animation have spent decades picking off various types of motion on various character and robot models.

Initial control designs tried to cancel the effects of dynamics as much as possible in order to simplify the control problem. For example, the earliest controllers for legged creatures remained balanced by never allowing the projected center of mass to leave the base of support and by moving slowly enough so that dynamics did not have an impact on control [55, 65]. This static approach to balance restricts the style of output simulations to slow, often unnatural motions, a critical restriction for the application of computer animation.

An important advance was the ability to control legged creatures with active balance which allowed the simulation of more dynamic motions such as running, hopping, and agile walking [59, 65]. Later these approaches were extended to create animations of humanoids executing specific tasks [66, 39, 89]. Yin and colleagues recently introduced a robust balancing mechanism for walking and running [94]. However, these hand-crafted feedback controllers are difficult to tune to achieve a desired motion style. For example, it would be difficult to manually tune such a controller such that it produced a motion matching a given motion capture sequence.

Deriving controllers that mimic reference motions such as motion capture sequences is an active area of research in robotics and graphics. For robotics, it is believed that human motions are more energy efficient. For graphics, starting from a reference motion makes it easier to direct the style of the motion. A key difficulty

in tracking a reference motion is that the motion cannot be tracked exactly while still maintaining balance. Modifications are needed to maintain balance because of dynamic inconsistencies between the system that generated the reference motion and the system being simulated and due to disturbances from the dynamic simulation environment. Earlier approaches based on tracking either avoided the issue of balance [95, 90, 92] or were restricted to tracking standing motions [96, 3]. Recently, researchers have been able to simulate balanced motions by pre-computing closed-loop control policies with randomized search algorithms [60, 70, 71].

Optimal control provides a more automatic approach to producing stylized character motion. Trajectory optimization, an optimal control tool, has been applied in *offline* simulations to synthesize character motion [88, 63, 30, 74, 68, 53]. Trajectory optimization involves expensive and finicky numerical computation. Furthermore, solutions are feed-forward policies valid only for the specific initial condition and task specified in the optimal control problem. These limitations make trajectory optimization an inappropriate approach to controlling simulated characters in interactive virtual worlds where the environment is dynamic. This thesis examines ways the tools of optimal control can be adapted for use in interactive applications.

2.3 Optimal Control

Optimal control is the study of mathematical and algorithmic methods for computing controllers that optimize a well-defined performance metric. Every controller has some task it is trying to get the character to perform. Many times this task is implicit. The starting point for an optimal control approach, however, is an explicit description of the task in the form of an objective function. For example, an objective function might try to minimize the amount fuel used to get a spaceship to the moon. For the purposes of animation, it is much easier to describe a task with an objective function than it is to find control forces that make a high-dimensional, non-linear, under-actuated character model achieve that task from a given initial state. Below we review optimal control theory and tools and review its application to animation.

Optimal control is a rich topic, and here, we focus on small pieces that are relevant to the rest of this work.

2.3.1 Measuring Performance

An optimal control policy minimizes an objective function of the form

$$g(\boldsymbol{x}_T) + \int_0^T \ell(\boldsymbol{x}, \boldsymbol{\pi}, t)\, \mathrm{d}t, \tag{2.4}$$

where $\ell(\boldsymbol{x}, \boldsymbol{\pi}, t)$, is a loss function applied at every time instant, and the desired state at the end is described by the terminal cost function, $g(\boldsymbol{x})$. The controller is defined by a *policy* $\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{x}, t)$. Typical loss functions penalize muscle exertion (some norm on the policy) or deviation from some example trajectory. Typical terminal cost functions describe some desired pose and velocity for the character. Note that here we have assumed a finite duration for the simulation T. There are variants that, for example, search for optimal controllers over infinite durations. This work focuses on finite-horizon problems.

The time evolution of the character state is constrained by the equations of motion for the system. We will focus on controlling systems with control-affine, stochastic dynamics:

$$\mathrm{d}\boldsymbol{x} = [\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}]\mathrm{d}t + \boldsymbol{C}(\boldsymbol{x})\mathrm{d}\boldsymbol{\omega}, \tag{2.5}$$

where $\boldsymbol{a} \in \mathbb{R}^d$ and $\boldsymbol{B} \in \mathbb{R}^{d\times m}$ are generally non-linear functions of state, and $\boldsymbol{u} \in \mathbb{R}^m$ is the control action. The Brownian noise, $\boldsymbol{\omega}$, can be used to model physical disturbances and modeling error with variance $\boldsymbol{C}\boldsymbol{C}^\top$. Characters modeled as linked rigid bodies are control-affine as we can see from Equation (2.2).

2.3.2 Hamilton-Jacobi-Bellman Equations

Optimal policies produce trajectories which minimize the expected value of the objective function in Equation (2.4), but the policy is also optimal for sub-intervals. This is known as the principle of optimality, due to Richard Bellman [17], and it leads to a

necessary and sufficient condition for the optimality of a given policy. The condition is that optimal policies satisfy a partial differential equation. Here, we will briefly re-derive the condition. First, define an optimal value, or cost-to-go, function as the minimum cost to get from the current state to the goal

$$v(\boldsymbol{x}, t) = \min_{\pi} E[g(\boldsymbol{x}_T) + \int_t^T \ell(\boldsymbol{x}, \boldsymbol{\pi}, t)\, \mathrm{d}t]$$

By the principle of optimality, we know that the minimum cost to get from where the character is now to the goal is equal to the minimum possible cost of being where the character is now plus the minimum cost of where the character ends up. Mathematically, this gives us a recursive definition of the value function

$$v(\boldsymbol{x}, t) = \min_{\boldsymbol{u}} \left(h\ell(\boldsymbol{x}, \boldsymbol{u}) + E[v(\boldsymbol{x} + h(\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}) + \epsilon, t + h)]\right)$$

where h is a time step and ϵ is zero-mean Gaussian random variable with variance $h\boldsymbol{C}\boldsymbol{C}^\top$. Finally, note that we have a boundary condition at the final time that $v(\boldsymbol{x}, T) = g(\boldsymbol{x}_T)$. We can simplify the value function definition using a second-order Taylor-series expansion. Setting $\Delta = h(\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}) + \epsilon$, we see that

$$v(\boldsymbol{x} + \Delta, t + h) \approx v(\boldsymbol{x}, t + h) + \Delta^\top v_{\boldsymbol{x}}(\boldsymbol{x}, t + h) + \frac{1}{2}\Delta^\top v_{\boldsymbol{x}\boldsymbol{x}}(\boldsymbol{x}, t + h)\Delta.$$

Using this approximation to compute the expected value on the right hand side of the recursive expression for the value function, we get

$$\begin{aligned} E[v(\boldsymbol{x} + \Delta, t + h)] = v(\boldsymbol{x}, t + h) &+ h(\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u})^\top v_{\boldsymbol{x}}(\boldsymbol{x}, t + h) \\ &+ \frac{1}{2} Tr(h\boldsymbol{C}\boldsymbol{C}^\top v_{\boldsymbol{x}\boldsymbol{x}}(\boldsymbol{x}, t + h)) \end{aligned}$$

where second order expressions in h have been dropped. With this expected value expression, we have

$$\frac{v(\boldsymbol{x},t)-v(\boldsymbol{x},t+h)}{h}=\min_{\boldsymbol{u}}(\ell(\boldsymbol{x},\boldsymbol{u},t)+(\boldsymbol{a}(\boldsymbol{x})+\boldsymbol{B}(\boldsymbol{x})\boldsymbol{u})^{\top}v_{\boldsymbol{x}}(\boldsymbol{x},t+h) + \frac{1}{2}Tr(\boldsymbol{C}\boldsymbol{C}^{\top}v_{\boldsymbol{x}\boldsymbol{x}}(\boldsymbol{x},t+h))).$$

Taking the limit as $h \to 0$ gives us a partial differential equation

$$-v_t(\boldsymbol{x},t)=\min_{\boldsymbol{u}}(\ell(\boldsymbol{x},\boldsymbol{u},t)+(\boldsymbol{a}(\boldsymbol{x})+\boldsymbol{B}(\boldsymbol{x})\boldsymbol{u})^{\top}v_{\boldsymbol{x}}(\boldsymbol{x},t)+\frac{1}{2}Tr(\boldsymbol{C}\boldsymbol{C}^{\top}v_{\boldsymbol{x}\boldsymbol{x}}(\boldsymbol{x},t))) \quad (2.6)$$

which is known as the Hamilton-Jacobi-Bellman (HJB) equation. Note that deterministic problems can be handled by this equation as well; the last term falls out since $\boldsymbol{C}=0$.

2.3.3 Solving HJB Equations

The HJB equation is a non-linear pde with a boundary condition in time. Normally, it does not have an analytical solution, so one must discretize it and solve it numerically. Common algorithms for doing this are known as value iteration and policy iteration in the reinforcement learning literature and are variants of dynamic programming [19]. Later, we'll see that certain HJB equations have an alternate formulation that allow you to apply linear solution methods.

The main difficulty in solving the HJB equation is that the number of discretization points you need to represent the value function grows exponentially with the dimensionality of the state space. This is affectionately known as the curse of dimensionality. This curse makes it impractical to solve HJB equations directly for complex, high-dimensional characters. Fortunately, there are work-arounds.

2.3.4 The Minimum Principle

Another way to attack the optimal control problem is to use calculus of variations to minimize the objective function. This line of attack leads to Pontryagin's mini-

mum principle which gives a set of necessary conditions for optimal trajectories [78]. More importantly it leads to numerical tools that avoid the curse of dimensionality. The price for this improvement is that the Pontryagin minimum principle is not a sufficient condition for optimality. This means that you may find a trajectory that is an extremal trajectory of the objective function but is not necessarily the optimum trajectory. For concreteness, consider how the discovery of the Fosbury flop greatly improved the performance of athletes in the high jump. For years, high jumping had been stuck in local minima using inferior jumping methods.

An optimal trajectory is an extremal trajectory of a variational problem consisting of the objective function augmented by Lagrangian constraints. This observation leads to the following necessary conditions for optimal trajectories

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.7}$$

$$-\boldsymbol{\lambda}(t) = \ell_{\boldsymbol{x}}(\boldsymbol{x}(t), \boldsymbol{u}(t)) + f_{\boldsymbol{x}}^{\top}(\boldsymbol{x}(t), \boldsymbol{u}(t))\boldsymbol{\lambda}(t) \tag{2.8}$$

$$\boldsymbol{\lambda}(T) = g_{\boldsymbol{x}}(\boldsymbol{x}(T)) \tag{2.9}$$

$$\boldsymbol{u}(t) = \arg\min_{\boldsymbol{u}^*}(\ell(\boldsymbol{x}(t), \boldsymbol{u}^*) + f(\boldsymbol{x}(t), \boldsymbol{u}^*)^{\top}\boldsymbol{\lambda}(t)). \tag{2.10}$$

There are a number of interpretations of $\boldsymbol{\lambda}$. It can be simultaneously viewed as the gradient of the value function, the adjoint variables of a constrained optimization, or the generalized momentum of a system. For now, we will just call it the adjoint state. The Pontryagin minimum principle gives ODE's governing the forward time evolution of the state and the backwards time evolution of the adjoint state. This two-point boundary value problem can be solved using a variety of numerical tools [20]. The methods can broadly broken down into two classes: direct methods and indirect methods. Direct methods try to directly minimize the objective function whereas indirect methods try to solve the boundary value problem given by the Pontryagin conditions.

Direct Methods

A common direct approach, known as direct transcription [20], is to represent the state and control trajectories as piecewise polynomials. At some number of points on the time interval, called collocation points, a defect equation is formulated that insures that the equations of motion are satisfied at those points (the defect is zero). Given state and control trajectories, the objective function is numerically integrated. This allows the continuous optimization problem to be formulated as a discrete constrained non-linear program.

An advantage of direct transcription is that it is straightforward to include additional constraints on the state or control trajectories. Furthermore, the programmer's job is made easier by not having to specify the adjoint equations. Due to these advantages, most applications of optimal control in graphics have used direct transcription. The biggest drawback to direct transcription is that computing the gradient of the objective function with respect to the state and control variables is extremely expensive. If the number of unknowns in the linear program is n, then computing the gradient is of the objective function is $O(n^2)$.

Indirect Methods

Indirect methods discretize the Pontryagin conditions and try and find state, control, and adjoint state trajectories that satisfy the discretized problem. This approach is cheaper since we do not need to compute the gradient of the objective function, but, in practice, indirect methods are very hard to use [20]. The programmer must specify the adjoint equations, and it is very difficult to add additional constraints on the state trajectory. Finally, in addition to specifying initial guesses for the state and control vectors, one must specify an initial guess for the adjoint variables. The adjoint state does not have an intuitive physical interpretation making this difficult. As a result, indirect methods are not very robust.

The Adjoint Method

The adjoint method is a direct method that uses the adjoint equations to cheaply evaluate the gradient of the objective function [56]. The gradient of the objective function can be evaluated as

$$\ell_{\boldsymbol{u}}(\boldsymbol{x}(t), \boldsymbol{u}(t)) + f_{\boldsymbol{u}}^{\top}(\boldsymbol{x}(t), \boldsymbol{u}(t))\boldsymbol{\lambda}(t). \tag{2.11}$$

This allows us to use a simple gradient descent algorithm to optimize the objective function. Given an initial guess for the control trajectory, forward simulate a state trajectory using the forward simulator of your choice. Next, integrate Equation (2.8) backwards in time to obtain a legal adjoint state trajectory. You can then use the gradient calculation from Equation (2.11) to update the control trajectory and repeat until the objective stops improving. While the programmer must specify the adjoint equations to use the adjoint method, there is no need to specify an initial guess for the adjoint state. In this work, we use the adjoint method to synthesize physically valid and locally optimal motions. In practice, we couple the basic method described here with a conjugate gradient descent algorithm and a line search.

2.3.5 Optimal Control in Computer Graphics

Value and policy iteration have recently applied to kinematic animation techniques in computer animation. Researchers pre-compute policies to choose appropriate motion clips in response to user input [54, 83]. These methods allow characters to avoid obstacles and turn responsively. A key to getting these methods to work was keeping the model of the character's state space low-dimensional using tricks like decoupling certain degrees of freedom. Similar approaches have been used in robotics to apply value functions to controlling simple legged robots [77, 23].

As mentioned above, trajectory optimization, an optimal control tool, has been applied in *offline* simulations to synthesize linked-rigid body character motion [88, 63, 30, 74, 68, 53]. The tools of optimal control have also been applied to control more elaborate characters. For example, the adjoint method was used to generate

animations of characters consisting of fluids [56]. In work not covered by this dissertation, my colleagues and I coupled the adjoint method with a reduced order model of deformable body dynamics to synthesize physically plausible animations [16].

2.4 Summary

The kinematic state of a physically-simulated character varies in time according to differential equations of motion. In this chapter, we sketched a derivation of the equations of motion for linked rigid bodies in generalized coordinates using Lagrangian mechanics. We saw how to apply external forces and how to compute contact forces analytically and using a penalty-based method.

Getting a physically-simulated character to do something besides falling down requires setting the control input, i.e. the character's muscle forces, in a meaningful way. Controlling fully-actuated characters is fairly straightforward, but most characters are under-actuated. Most prior controllers for interactive under-actuated characters were crafted by hand. Optimal control provides automatic, offline, tools for creating controllers. Now that we have reviewed some concepts from optimal control, we'll build on them to create controllers for interactive characters.

Chapter 3

Interactive Simulation of Stylized Human Locomotion

Consistent interaction between characters, objects, and the surroundings can be difficult to achieve in dynamic environments. Simulation produces valid motions automatically but simulating human motion requires a control system that computes internal muscle forces. Without such forces, a simulated human would simply fall to the ground. The human form and the unstable dynamics of its bipedal locomotion make it difficult to find muscle forces that maintain balance, particularly in complex environments with unexpected disturbances. Its use in computer animation is further complicated by the need to generate motions that are comparable to recorded motion data. This has been difficult to achieve and, except for rag-doll effects, human simulation is rarely used in interactive animation systems.

This chapter describes controllers for interactive simulation of stylized human locomotion. This control design precomputes a balance strategy for the given style using a simplification of Equation (2.6) that provides an automated analysis of linear time-varying systems. By tailoring the balance strategy in this manner, a controller preserves the style better than a more cautious strategy. For example, a controller can restrict the center of mass to be directly above the feet at all times, but in doing so it can only accomplish slow robotic-like motions. In contrast, the controller in this chapter can reproduce a variety of locomotion styles.

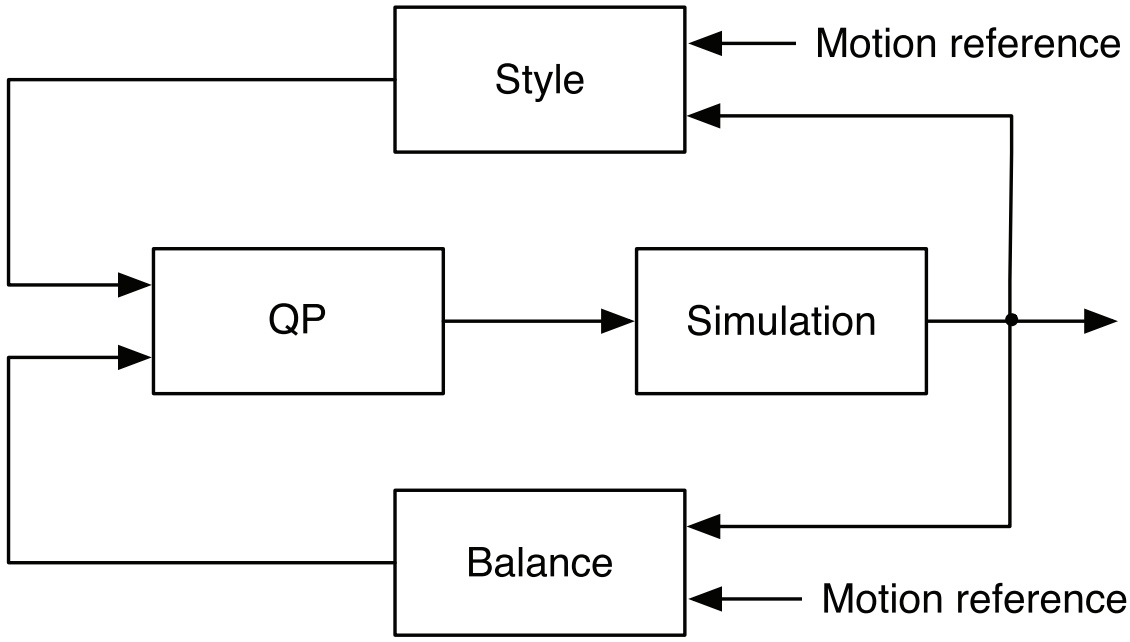

Figure 3-1: Automated analysis of linear time-varying systems leads to a balance feedback policy that accounts for the under-actuated dynamics in the human motion. With balance taken care of, a control system can then reproduce a given locomotion style by tracking every body joint. Quadratic programming combines the two feedback terms to compute control forces needed to simulate stylized locomotion in new environments.

The automatic pre-computation, which typically completes in less than two minutes, determines a linear time-varying state feedback that prescribes desired accelerations for the three largest body segments: the two legs and the torso. Simultaneously, the second feedback loop tracks individual joint angles to compute the accelerations needed to preserve the given style. As shown in Figure 6-1, a reference motion guides both the style and balance feedback. The style feedback aims to preserve the nuances of the motion, while the balance feedback seeks to adapt the motion of three balance-critical segments. The control algorithm computes a final set of forces by maintaining a desired tradeoff between the balance and style feedback.

Interactive animation systems could use this approach in addition to traditional kinematic solutions. Physically based simulations automatically produce motions that are geometrically and physically consistent. When paired with a control system that produces high-quality motion, these simulations can animate lifelike human mo-

tion in dynamic environments, which is difficult to accomplish with kinematics alone. Furthermore, this process transforms a single recorded motion, valid for one environment only, into a general purpose action that can be used in many other settings or even composed with other actions to create versatile characters in games and other interactive applications.

3.1 Related Work

The ability to remain balanced affects the styles of motion a character can achieve [85]. The earliest controllers for legged creatures remained balanced by never allowing the projected center of mass to leave the base of support and by moving slowly enough so that dynamics did not have an impact on control [55, 65]. This static balancing approach restricts the style of output simulations to slow, often unnatural motions.

An important advance was the ability to control legged creatures with active balance which allowed the simulation of more dynamic motions such as running and hopping and more agile walks [59, 65]. Later these approaches were extended to create animations of humans executing specific tasks [66, 39, 89]. These approaches simplified control design by decoupling the problem into different regulation tasks, which were often understood by analyzing simple dynamics approximations. The controller we describe also approximates the full dynamics to make it possible to compute optimal feedback for the approximation. However, rather than designing these policies by hand, it precomputes feedback gains automatically. It also incorporates motion-capture data to ease the generation of more natural motions.

An impressive array of motions can be simulated using manually designed feedback controllers. These controllers can be composed to perform multiple tasks [22, 29]. However, these control policies are task, model, and environment specific. Changes in task, character model, or simulation environment require careful redesign of a new feedback policy and there is a nonintuitive relationship between controller parameters and simulation output. Dynamic scaling rules and randomized search can help [38] but it remains difficult for an artist to achieve a desired motion style using manually

designed feedback controllers. In our approach, much of the control policy is computed automatically by exploiting an explicit model of the character dynamics. In addition, we take advantage of the fact that we know the desired motion ahead of time by precomputing a balance policy tailored to that motion. This reduces the number of manually adjusted parameters and improves the quality of simulated motion.

Others also take the approach of tracking a reference motion to improve the style of simulated motions. Earlier approaches based on tracking either avoided the issue of balance [95, 90, 92] or were restricted to tracking standing motions [96, 3]. Yin and colleagues introduced a very effective balancing mechanism for SIMple BIped COntrol (SIMBICON), which they also coupled with feedback error learning to track cyclic walking motions [94]. The same balancing mechanism can also be coupled with quadratic programming to track non-cyclic motions without relying on feedback error learning [25]. This chapter explores another form of balance through subtle body adjustments instead of swing-leg placement. It is more complex to implement, but reduces parameter tuning and generates motions that are more similar to the given reference, including possibly non-cyclic motions such as standing, stepping, and transitions between the two.

Optimal control has been shown to produce stylistic human motions in *offline* simulations [88, 63, 30, 74, 68, 53]. It is conceivable that offline trajectory optimization could be used to precompute a sequence of control actions that reproduce a motion style. However, these computations are finicky and slow. In interactive systems, they must be recomputed frequently because they are easily invalidated with unexpected changes in the dynamic environment. In fact, even minor accumulation of integration errors will disrupt such a policy after only a few simulation steps [48]. Laszlo and van de Panne apply limit-cycle control to automate the derivation of closed-loop controls from a given reference motion [48]. Approximating the step to step return map dynamics has the advantage of incorporating collision effects and time-varying dynamics into the feedback policy. The balance policy in this chapter relies on optimization and time-varying feedback to better match the given reference motion.

Recently, researchers have been able to simulate balanced motions by precomput-

ing closed-loop control policies with randomized search algorithms [60, 77, 70, 71]. However, it is not known how to extend these approaches to higher dimensional search spaces, an important limitation given the large number of degrees of freedom for a 3D human character. Instead of relying on a brute-force search, our pre-computation approximates the dynamics of the character with a discrete-time linear time-varying system (LTV) by differentiating system dynamics along the reference motion. This allows us to automate our analysis and to precompute balance policies efficiently. LTV approximations have previously been exploited for simple walking systems using differential dynamic programming [41, 10, 76]. In this work, we show how to apply controls derived for simple walking systems to more realistic 3D characters and how to incorporate desired motions from motion data.

3.2 Balance

A balance strategy determines the style of motion that can be reproduced in simulation. Conservative strategies may maintain balance by restricting the center of mass to be above the foot support region at all times. This is a safe strategy but it produces slow robotic motions. A regular walk requires excursions beyond these regions of support and more stylized motions may require even more delicate balancing. Manual design of a balance strategy is possible but tedious, and it requires some knowledge of the expected terrain. Our goal is to devise an inexpensive control approach that tailors itself to the provided reference motion automatically. In this section, we describe the pre-computation of a balance policy for a given reference motion.

3.2.1 Balance As Optimal Tracking

Maintaining balance requires careful manipulation of all limbs to adjust the location of the overall center of mass. We formulate balance control as an optimization that tracks the provided reference motion. Consider a dynamical system with n degrees of freedom and m actuators. A reference motion for that system is a sequence of states

$\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_T$, where each state $\bar{\boldsymbol{x}}_t \in \mathbb{R}^{2n}$ includes generalized positions and velocities. The current tracking error is the difference between the current and desired state $\Delta \boldsymbol{x} = \boldsymbol{x} - \bar{\boldsymbol{x}}$. An idealized optimal controller, then, finds the sequence of control actions, $u \in \mathbb{R}^m$, that minimizes the total tracking cost:

$$\arg\min_{u} \quad \Delta \boldsymbol{x}_T^\top Q_T \Delta \boldsymbol{x}_T + \sum_{t=1}^{T-1} \Delta \boldsymbol{x}_t^\top Q_t \Delta \boldsymbol{x}_t + \boldsymbol{u}_t^\top R_t \boldsymbol{u}_t \tag{3.1a}$$

$$\text{subject to} \quad x_{t+1} = F(x_t, u_t). \tag{3.1b}$$

The system function F expresses the discrete dynamics of the system. We use the discrete expression instead of the continuous form, $\dot{x}(t) = f(x(t), u(t))$ because numerical integrators fundamentally assume constant control force $\boldsymbol{u}(s) = \boldsymbol{u}_t$ over finite intervals $s \in [t, t + \delta t]$ of short duration:

$$F(x_t, u_t) = x_t + \int_t^{t+\delta t} f(x(s), \boldsymbol{u}_t)\, \mathrm{d}s \tag{3.2}$$

The tracking and control costs are determined by the (semi) positive definite matrices, $Q_t \geq 0 \in \mathbb{R}^{2n \times 2n}$ and $R_t > 0 \in \mathbb{R}^{m \times m}$, that express the desired compromise or the optimal performance. For example, if Q weights the tracking error across all joints equally, then the optimal control will reduce tracking errors in the wrist motion as vigorously as it corrects errors in the upper leg motion. While this choice preserves the style of hand and leg motion alike, it can also lead to poorly balanced motion because the legs and the upper body play a more critical role in bipedal locomotion. One possible approach to this issue is to apply inverse reinforcement learning [1]. Later, we will describe our approach which is to manually reduce the dimensionality of the walking system.

3.2.2 Simplified Optimal Tracking

Interactive computation of the control policy according to Equation (3.1) remains impractical. It is a non-convex optimization problem because of the nonlinear dy-

namics of the character. These problems can be solved with smooth optimization but they require a good initial guess and are still too slow for interactive simulation [20]. Moreover, the solution would only be valid for a single initial condition. In interactive animation systems, we would also need to compute controls fast enough to reject any new disturbances.

Linear Time-Varying Approximation Our solution approximates the nonlinear dynamics of the character model with a discrete-time linear time-varying system (LTV):

$$\Delta \boldsymbol{x}_{t+1} = A_t \Delta \boldsymbol{x}_t + B_t \Delta \boldsymbol{u}_t. \tag{3.3}$$

The time-varying system matrices $A_t \in \mathbb{R}^{n \times n}$ and $B_t \in \mathbb{R}^{n \times m}$ are computed by linearizing the nonlinear system dynamics along the desired reference trajectories. Given the system function G for the simplified dynamics (cf. Equation (3.2)), the time-varying system matrices are the Jacobians $A_t = \frac{\partial G}{\partial \boldsymbol{x}}(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t)$ and $B_t = \frac{\partial G}{\partial \boldsymbol{u}}(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t)$ of G with respect to the state and control.

We annotate sections of each reference motion to delineate different contact states. An associated LTV is constructed for each single support phase. In a simple walk cycle for example, there would be an LTV system for the right step and another for the left (Figure 3-3). In our work, the LTV systems do not span collision events. In principle, linear approximations could differentiate through contact transitions. For example, Popović and colleagues do so for rigid body simulations with frictionless impacts [62]. We take a simpler approach by approximating each single support phase (e.g. alternating left and right steps for walking). During double support, the control relies on another feedback mechanism (§3.3.1) to stabilize the character.

Feedback Policy The LTV approximation allows us to compute the optimal feedback policy for a problem with quadratic cost functions. We apply this idea to derive

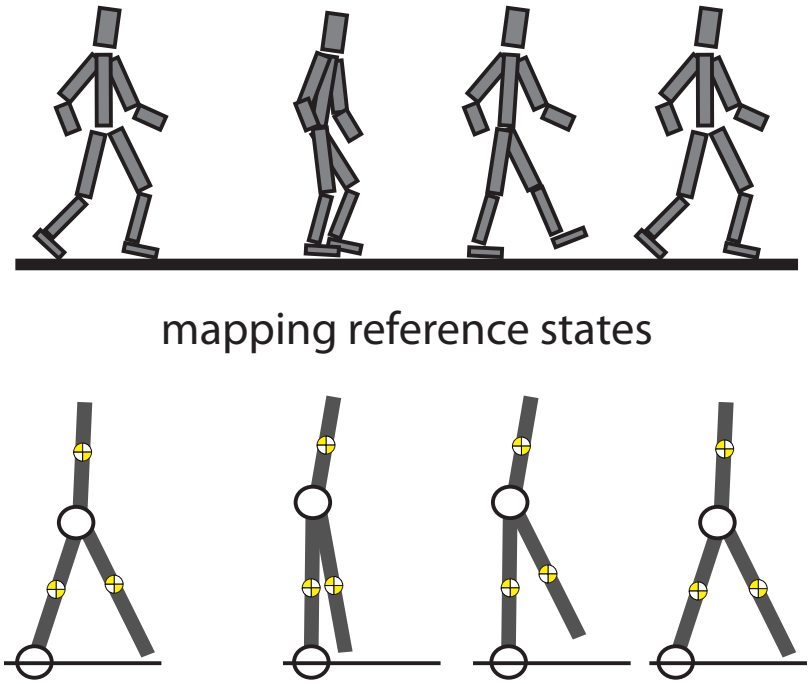

Figure 3-2: The state of the character is geometrically mapped to the approximate model. The user annotates sections of the reference motion corresponding to different contact states. A simple model is constructed for each single support section of the reference motion and the motion is retargeted geometrically to produce a sequence of desired states for the simple model. At runtime, the same mapping is used to determine the current tracking error for the three-link model.

balance control from this approximation of the original formulation:

$$\min_{\Delta \boldsymbol{u}} \quad ||\Delta \boldsymbol{x}_T||^2 + \sum_{t=1}^{T-1} c||\Delta \boldsymbol{x}_t||^2 + ||\Delta \boldsymbol{u}_t||^2 \tag{3.4}$$

$$\text{subject to} \quad \Delta \boldsymbol{x}_{t+1} = A_t \Delta \boldsymbol{x}_t + B_t \Delta \boldsymbol{u}_t, \tag{3.5}$$

The time window T is determined by the duration of the corresponding single support phase in the reference motion. The scaling constant c determines the tradeoff between tracking fidelity and additional control effort. We found a value of $c = 0.1$ yielded the best results in our experiments, but many other values will yield motion of similar quality.

The solution to this problem is a linear function of the tracking error, $\Delta \boldsymbol{u}_t = K_t \Delta \boldsymbol{x}_t$, which is known as the Linear Quadratic Regulator (LQR) [72]. The feedback

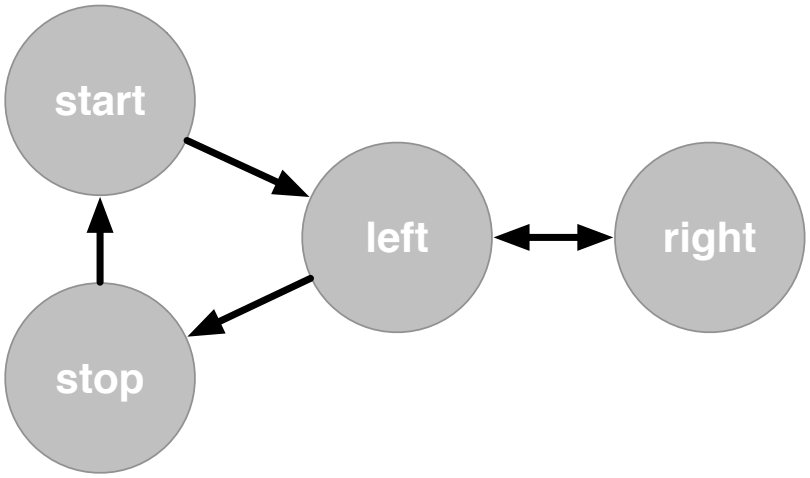

Figure 3-3: The balance control approximates dynamics of each single support phase with a linear time-varying system. Dwell times are determined manually by timing corresponding phases in the reference motion. The balance control illustrated in this diagram alternates between LTV systems for left and right step with the option to pause and restart.

gains K_t are precomputed efficiently by numerically integrating Riccati equations [43, 21, 72]. This solution is optimal for the given LTV system. Differential dynamic programming is closely related to LQR and has been exploited for simple walking systems [41, 10, 76].

Here we briefly re-derive LQR for a linear time-invariant system with no noise. A linear time-invariant system has dynamics of the form $A\boldsymbol{x} + B\boldsymbol{u}$. If we assume a quadratic objective of the form $\frac{1}{2}\boldsymbol{x}^\top Q\boldsymbol{x} + \frac{1}{2}\boldsymbol{u}^\top R\boldsymbol{u}$, we can carry out the minimization on the right hand side of Equation (2.6) analytically. Differentiating with respect to control, we see that the minimizer is

$$\boldsymbol{u} = -R^{-1}Bv_{\boldsymbol{x}}.$$

Plugging this solution into the HJB equation, we get

$$-v_t(\boldsymbol{x}, t) = \frac{1}{2}\boldsymbol{x}^\top Q\boldsymbol{x} + v_{\boldsymbol{x}}^\top B^\top R^{-1}Bv_{\boldsymbol{x}} + (A\boldsymbol{x} - BR^{-1}v_{\boldsymbol{x}})^\top v_{\boldsymbol{x}}.$$

If we guess that our value function is quadratic $v = \frac{1}{2}\boldsymbol{x}^\top P\boldsymbol{x}$, we can derive an ordinary differential equation for the Hessian of the value function P. This ODE is the Riccati

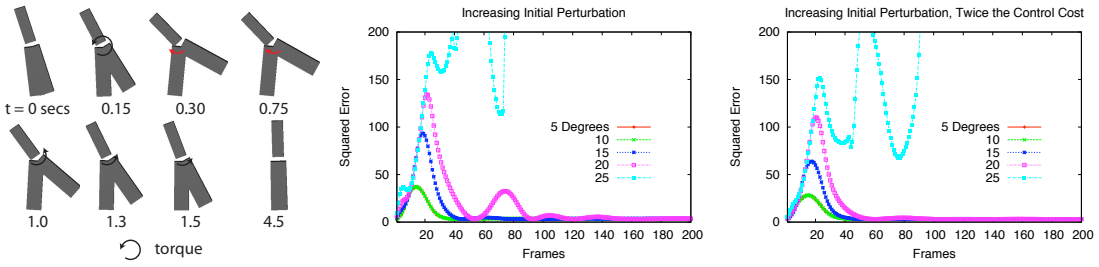

Figure 3-4: To prevent a fall, the LQR balance controller employs a flywheel strategy, rapidly swinging the upper body in the direction of the fall and the swing leg in the opposite direction. The strategy is depicted using snapshots from a simulation in the leftmost image. The arrows indicate the direction and relative magnitude of the applied torque. For this particular model, the controller can recover from an initial error of about 20 degrees which corresponds to the projected center of mass being approximately 36 centimeters away from the base of support. The time evolution of the squared tracking error of this system is shown for various initial perturbations in the plots on the right. Doubling the control cost slightly improves performance but in either case the end result is the same.

equation mentioned above

$$-\dot{P} = Q + A^\top P + PA - PBR^{-1}B^\top P.$$

Note that A and B can vary in time without changing the derivation here. Also, since the value function is equal to the terminal cost function at the final time, we have the boundary condition in time that P must equal the Hessian of the terminal cost function at the final time.

3.2.3 Tracking a Simple Model

Control designers often make simplifying assumptions to make control design easier, its analysis simpler, and its use more broadly applicable [65, 48]. Full and Koditschek argue that simple models play an important role in neuromechanical hypotheses of motion control [36]. Others have used simple models to simplify high-dimensional non-convex optimizations [63, 30, 68]. We, too, will make similar approximations for the stepping motions we are interested in controlling. Specifically, we use a simple three-link model (Figure 3-5) that suppresses many details while preserving only those

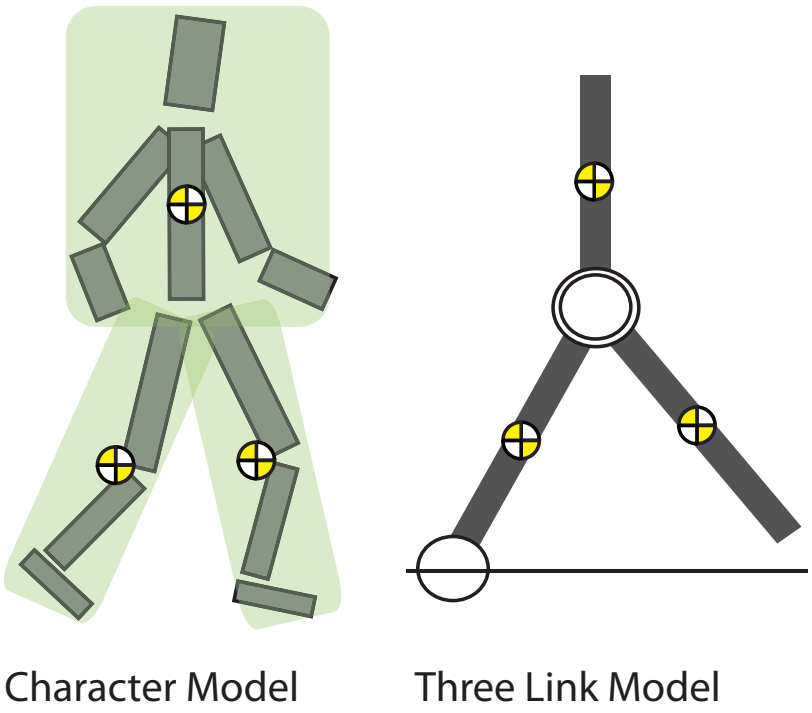

Figure 3-5: A simple three link model is constructed from the geometric and inertial properties of the character model. The inertial properties of each simple link are created by summing the inertial properties of corresponding links on the simple character. The base link is attached to the ground with a three-dof ball joint. Two three-dof ball joints connect the other two links to the base link. The yellow dots represent centers of mass, which are in agreement with the detailed character model.

aspects of dynamics that are most relevant for balance control.

The three-link model is under-actuated like the full human character with its mass distributed over the three-links. These are important features for recreating human motion strategies which manipulate total linear and angular momentum to maintain balance [2, 42, 64]. The base link approximates the contact between the foot and the ground with an unactuated ball joint. The next link is attached to the base with a ball joint like the upper body in the human figure. This allows it to speed up and slow down by leaning forwards and backwards. Lastly, the third link is attached to the upper body with a ball joint just like a swing leg whose motion is used to anticipate the next contact. We then adapt the same control formulation in Equation (3.1) using the state of the simple model and the system function derived from the dynamics for the three-link model: $\dot{x}(t) = g(x(t), u(t))$, where $x \in \mathbb{R}^{18}$ consists of three rotations

and three angular velocities, and $u \in \mathbb{R}^6$. The inertial properties for the dynamics of the simple model are constructed automatically by accumulating inertial properties of corresponding links in the full model.

We calculate the reference trajectory, $\bar{\boldsymbol{x}}$, using a simple geometric mapping from the character to the simple model (see Figure 3-2). The ankle of the supporting leg of the character is mapped to the base joint of the simple model. Then, the location of the center of mass of the support leg relative to the ankle is used to determine the angle of the base link on the simple model. This angle determines the location of the "hip" on the simple model. The relative locations of centers of mass of the upper body and swing leg are similarly used to determine the orientation of the remaining links on the simple model. The relative velocities of the centers of mass on the full character determine the angular velocities of the simple model.

Advantages The simple model makes it much more computationally feasible to precompute feedback policies. We use iterative optimization to estimate the matching control forces $\bar{\boldsymbol{u}}$ for the mapped reference motion [52]. The optimization usually converges in less than five iterations and the total time needed to compute the LTV matrices typically completes in less than two minutes. This step is typically much more costly for an unreduced model. Using a simplified model also simplifies computation of LTV approximations. Multi-body simulators typically solve linear complementarity programs to compute frictional contact forces [12, 5] which are non-smooth mappings of the current state. Instead, our simple model approximates this contact with an unactuated joint whose action is a smooth function of the state. Second, the computation of the optimal feedback gains is faster. The computation of LQR control is $O(Tn^3)$ where T is the number of time samples. Since the simple model has fewer degrees of freedom (9 versus 60), the computation of optimal feedback gains is faster. Third, the simple model reduces the memory needed to store the optimal feedback gains. The optimal gain matrix requires $O(Tn^2)$ space, which makes it less practical to store feedback gains for the full human model, especially for many reference motions. The simple model, on the other hand, requires only a small fraction of that

space.

We have now described the pre-computation portion of our controller which finds a set of time varying gain matrices to stabilize the motion of an approximation of our character. What makes our approach unique is that we make the entire analysis automatic and its application efficient by exploiting the information in provided reference motions. In the next section, we describe a didactic example that illustrates one application of our control: standing balance recovery.

3.2.4 Example: Balance Recovery

We illustrate the behavior of LQR control on a balance recovery task for our simple three-link model. Pratt and colleagues develop another balance strategy for an equivalent under-actuated model [64] and many other approaches are also discussed in the literature. Unlike most of this prior work, our approach relies on fully automated analysis with linear approximations and optimal state feedback. And we will use the same systematic approach to reduce the need for manual tuning when controlling more complex stylized motions.

The task for our control is to recover balance by returning to the "inverted-pendulum" configuration, an unstable equilibrium point for the model. The reference motion is a single pose indicating the inverted-pendulum configuration. The reference forces are zero in this case because the desired pose is a static equilibrium point. Since there is no particular time the pendulum should return to the equilibrium point, the optimal feedback forces are determined by the solution to an infinite-horizon problem:

$$\min_{\Delta \boldsymbol{u}} \quad \sum_{t=1}^{\infty} \Delta \boldsymbol{x}_t^\top Q \Delta \boldsymbol{x}_t + \Delta \boldsymbol{u}_t^\top R \Delta \boldsymbol{u}_t \tag{3.6}$$

$$\text{subject to} \quad \Delta \boldsymbol{x}_{t+1} = A \Delta \boldsymbol{x}_t + B \Delta \boldsymbol{u}_t, \tag{3.7}$$

where the discrete-time approximation is time-invariant because the linearization is around a single point instead of a trajectory.

The feedback control derived from this formulation is quite effective. For a model

designed with rough human-like inertias and dimensions, the controller recovers for a range of leaning angles up to the maximum lean of 20 degrees. This maximum lean corresponds to a deviation of 36 centimeters between the projected center-of-mass and the base of the support leg.

The precise manner of recovery is controlled by the tracking Q and control R costs, with a very wide range of possible settings. We show one time-lapse sequence of recovery in Figure 3-4 for the case when the velocity errors are permitted by setting their costs to zero while both the position and control are weighed equally by setting their cost matrices to the identity. The controller recovers by rapidly swinging the upper body in the direction of the fall while simultaneously swinging the non-stance leg in the opposite direction. Once the center of mass is safely over the support leg, the model slowly returns to its balanced equilibrium.

3.3 Applying Pre-Computed Controls

We apply the pre-computed linear feedback policy indirectly. First, we use it to compute desired accelerations for the full character model. Then, we formulate a quadratic program that allows us to compute torques that achieve those accelerations given the current configuration of the character and its contact points. Applying the pre-computed linear feedback policy directly does not work well. The pre-computed balance gains are tailored to a given reference motion in order to emulate the motion of a three link walker with point feet. Our character has more links and block feet. To resolve this, we use a second style feedback control to compute accelerations that greedily try and track the original reference motion. These two feedback policies, balance and style, produce different versions of the best action to take. We arbitrate between these two suggestions using the quadratic program which finds the feasible forces that best achieve some weighted combination of the two suggestions. Note that, the resulting controller is a non-linear function of character state making it hard to evaluate its robustness analytically. In the results section of this chapter, we describe some ways to measure robustness.

3.3.1 Style Feedback

The style feedback determines the joint accelerations a_{s} needed to track the reference motion:

$$a_{\mathrm{s}} = \ddot{\bar{\boldsymbol{q}}} + k_1 D(\bar{\boldsymbol{q}}, q) + k_2(\dot{\bar{\boldsymbol{q}}} - \dot{q}), \tag{3.8}$$

where the comparison function D uses quaternion arithmetic to compute the angular acceleration needed to eliminate the difference between current and desired joint orientation [14]. This feedback acceleration tracks the root of the full human figure. So, although it primarily maintains style, it also plays a role in maintaining balance. In a standing motion, for example, any motion of the root produces accelerations that dampen this motion.

If desired joint accelerations are met with properly chosen control forces, the error on each joint will behave as a spring-damper mechanism. The stiffness k_1 and damping k_2 gains determine the rise, settling time, and the overshoot for the error-cancellation curve. We keep things simple in our experiments with gain settings for a critically damped spring $k_2 = 2\sqrt{k_{\mathrm{s}}}$, leaving only one parameter value k_{s} for each joint. The values change for each human figure, but in many cases, the same values are used for all motions with that character model (Table 3.1).

3.3.2 Balance Feedback

The balance feedback determines the stable motion of the simple model using the precomputed state feedback gains K_t. We compute this control by mapping the full state of the character $(q, \dot{q})$ onto the reduced state of the simple model $\boldsymbol{x}$ as described in Section 3.2.2. The balance control is then a function of measured deviations $\Delta\boldsymbol{x}$ and precomputed reference forces: $\boldsymbol{u}_t = \bar{\boldsymbol{u}}_t + K_t \Delta\boldsymbol{x}_t$. To map this motion back onto the full character, we use forward dynamics of the simple model to compute the center-of-mass accelerations a_{b} for all three limbs in the simple model given the new $\boldsymbol{u}_t$. The balance control for the full character will then seek to find joint accelerations that match center-of-mass accelerations of the simple model.

3.3.3 Quadratic Programming

Our control computes the final forces for the simulation with quadratic programming (QP). The QP solution arbitrates between style and balance feedback by finding joint torques that are close to a weighted combination of the two suggested accelerations. This is done carefully to ensure that computed torques are consistent with the feasible ground reaction forces:

$$\min_{u,\ddot{q},\lambda} \quad w_{\mathrm{s}}\|a_{\mathrm{s}} - \ddot{q}\|^2 + w_{\mathrm{b}}\|a_{\mathrm{b}} - \dot{J}_{\mathrm{c}}\dot{q} - J_{\mathrm{c}}\ddot{q}\|^2 \tag{3.9a}$$

$$\text{subject to} \quad \ddot{q} = f\left((q,\dot{q}), u + J_{\mathrm{p}}^{\top} V\lambda\right) \tag{3.9b}$$

$$J_{\mathrm{p}}\ddot{q} + \dot{J}_p\dot{q} = 0,\ \lambda_i \geq 0. \tag{3.9c}$$

The objective function is a weighted sum of the style and balance feedback terms. The balance feedback terms are desired accelerations for three points on the body and are active during single support. The Jacobian matrix J_{c} relates the acceleration of these points to the generalized configuration of the character. The constraints maintain a physically consistent relationship between joint accelerations $\ddot{q}$, control torques $\boldsymbol{u}$, and contact forces $V\lambda$ by enforcing the contact dynamics equations at the current time instant [3]. Note that $\boldsymbol{u} \in \mathbb{R}^{n-6}$ is the control signal for the character and not the simple balance model.

The manipulator equations (3.9b) encode the dynamics of the linked rigid body structure representing the character. These equations can be derived by Euler-Lagrange mechanics and can be evaluated efficiently [32]. For a fixed position and velocity, the manipulator equations express a linear relationship between applied forces and acceleration. Similar expressions are exploited in computed torque methods to control industrial robots [51]. Our formulation follows a more general formulation [3] that also includes unilateral constraints on ground reaction forces and mixed objectives.

The remaining constraint equations (3.9c) prevent the ground reaction forces from accelerating the contact points. The expression $J_{\mathrm{p}}^{\top} V\lambda$ maps external ground reaction

forces expressed in a discretized friction cone basis, V, into generalized torques using the Jacobian for the contact points J_{p} [30]. The inequality constraint ensures that ground reaction forces do not pull on the character ($\lambda_i \geq 0$).

The behavior of the QP is controlled by the relative weights of the objective function, w_{b}, and w_{s}. In practice, it is not too difficult to find values of these parameters that work and similar values typically work for many motions on the same character. We use a recursive implementation of linked rigid body dynamics [32] and SQOPT to solve QP problems [37].

Adjustment with PD. Following our formulation in previous work [25], the controller adjusts the QP solution with a low gain proportional-derivative (PD) component. The desired pose (i.e. set point) of the PD component is the current pose of the reference motion. The gains for the PD component were set to gains used in previously published work [71, 94] and then further reduced by multiplying with a factor smaller than one. Using lower gains allows the control to be less stiff which leads to more stable simulations. The values used are listed as the "Strength" parameters in Table 3.2. At run time, the gains for a joint are scaled by the composite rigid body inertia of all child links [96, 25].

The PD adjustments accomplishes two tasks. First, it guides the character through contact transitions. The QP's model of dynamics assumes a fixed contact configuration while we would like to track motions that make and break new contacts. Second, the PD component adjusts the QP solution at each simulation step. The QP step typically takes two to three milliseconds on a Pentium 4 2.8 Ghz processor. To allow the simulation to run faster, the QP is solved every ten to one hundred simulation steps. The PD component is calculated at each time step allowing it to react to disturbances instantly.

3.4 Results

The new controller produces high-quality motions for a large variety of reference motion styles. In many instances, simulations are almost indistinguishable from input motions. Moreover, it succeeds in the presence on unexpected disturbances, allowing it to generate physically valid motions even in the environments that are quite different from the one used during motion capture. Our comparisons with prior work reveal that customized balance strategy yields motions that better match the style of a given reference motion. The final results and these comparisons are best seen in video [26].

3.4.1 Style

The automatic derivation of our controller simplifies simulation of stylized human motions. We demonstrate this versatility by simulating motions acquired from several different sources: the dataset assembled by Sok and his co-workers [71]; motions generated by the SIMBICON controller [94]; and our own dataset of stylized motions recorded in a motion-capture studio. Time-lapse snapshots of some example simulations are shown in Figure 3-8.

Changing styles with our controller is straightforward. Once a good set of controller parameters have been found for one motion, those parameters typically work for many reference motions. Listed are the parameter settings for each of the styles simulated by our controller. Table 3.1 lists several of the styles simulated by our controller and the parameters that had to be changed. The first two datasets required very little tuning; the same parameter setting worked for all motions in many different environments. Simulating our own stylized motions required some tuning, but this still reduced to changing only two real-valued parameters.

Highly stylized motions require more expressive simple models. For example, we were not able to track a motion where the subject swayed his arms and upper body from side to side to imitate a charging monkey. The simple model used in our experiments so far does not capture that balance strategy because it lacks arms, which

Motion	k_s	w_b
2D Data		
Walk	500	0.1
Uphill	500	0.1
Backwards	500	0.1
March and Walk	500	0.1
3D SIMBICON		
Walk	1000	0.1
Downhill	1000	0.1
3D Mocap		
Sideways Steps	1200	0.2
Monty Python	1200	0.01
Monty Python See-Saw	1200	0.01
Stop and Start	1200	0.01
Dance	1800	0.01
Sumo Stomp	1200	0.01
Turn	1200	0.01

Table 3.1: Changing styles with our controller is usually straightforward. Once a good set of controller parameters have been found for one motion, those parameters typically work for many reference motions. Listed are the parameter settings for each of the styles simulated by our controller. The style feedback parameter is k_s. Finally, w_b determines the weight of the balance feedback policy. The style weight, w_s, was fixed at 0.5 for every result.

play a critical role for the balance of this motion. We expect that adding additional links to our simple model would address this mismatch.

The balance policy used by a controller determines which types of motions can be simulated by that controller. In the extreme case, a controller that cannot balance will simply fall down which is probably not the style intended. The controller can be better at reproducing the reference style by customizing its balance policy for the given reference motion. We compared the tracking quality with two recently proposed controllers [25, 71]. Although some amount of tracking error is unavoidable (due to many differences between the simulation and the real world), the controller with the heuristic balance policy exhibits an irregular gait [25]. In contrast, the controller with a tailored balance policy yields motions that are smoother and more faithful to the original reference motion. Direct visual comparison also shows reduced oscillation for a backwards walk when compared to the results presented by Sok and colleagues [71].

2D Model					
Link	Strength	Mass	Inertia		
head	150	3	0.011		
upper arm	200	2	0.022		
lower arm	150	1	0.009		
torso	N/A	10	0.176		
thigh	200	7	0.121		
shin	200	5	0.077		
foot	200	4	0.019		
3D SIMBICON					
Link	Strength	Mass	Ix	Iy	Iz
hip	N/A	16.61	0.0996	0.1284	0.1882
trunk	30	29.27	0.498	0.285	0.568
head	1.5	6.89	0.0416	0.0309	0.0331
upper arm	1.5	2.79	0.00056	0.021	0.021
lower arm	1.5	1.21	0.00055	0.0076	0.0076
hand	0.1	0.55	0.05	0.2	0.16
thigh	9	8.35	0.145	0.0085	0.145
shin	3	4.16	0.069	0.0033	0.069
foot	0.3	1.34	0.0056	0.0056	0.00036
3D Mocap					
Link	Strength	Mass	Ix	Iy	Iz
hip	N/A	12.8	0.174	0.148	0.12
trunk	100,400	17.02	0.335	0.167	0.281
head	30,120	4.067	0.02	0.0176	0.0248
clavicle	40,100	2.51	0.0111	0.0111	0.00691
upper arm	40,100	1.42	0.0133	0.0014	0.0132
lower arm	30,120	0.57	0.003	0.0003	0.003
hand	30,120	0.09	0.00008	0.00002	0.0001
thigh	400,1600	9.013	0.204	0.0356	0.208
shin	400,1600	3.91	0.0585	0.0087	0.06
foot	100,400	0.29	0.0002	0.0004	0.0003
toe	5,20	0.23	0.0002	0.0002	0.0002

Table 3.2: Model parameters used for our results. The joint stiffness values are low when compared to uses of these models in other controllers. This is because much of the control effort is generated by the QP component of our controller. The units are as follows: newtons per radian for the gains, kilograms for the mass, and kilogram meters cubed for the inertias. For the model used to track motion capture results, inertial properties were generated by integrating a fixed density over geometric models of each limb. The dimensions of the character were chosen to match the dimensions of our capture subject. Also, for this model, we used two sets of gains for the PD component of the controller as indicated by comma-separated strength values.

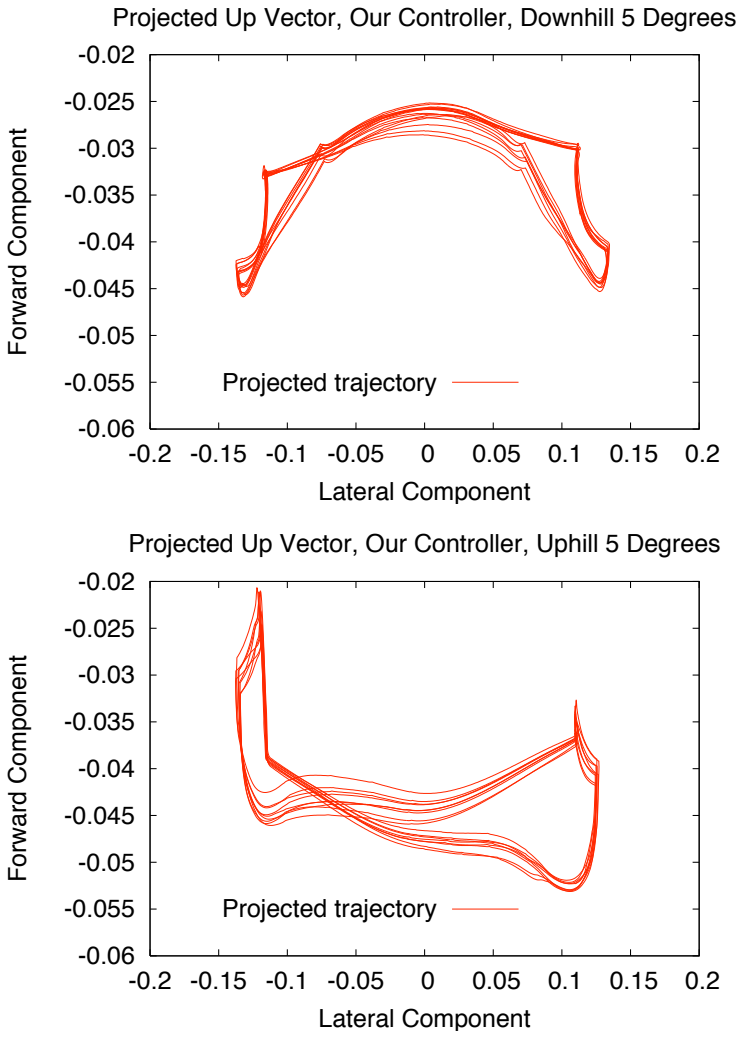

Figure 3-6: We plot the horizontal components of the root link's up vector. The x axis is the lateral component and the z axis is the forward component (the character is walking down negative z). The first plot was created using our controller on a slope of five degrees tracking the output of a manually designed PD controller on flat ground. The second plot was created using our controller on an uphill slope of five degrees but with the same parameters as the downhill plot. Note that in the uphill plot, the up vector has a more negative forward component in the upright phase (lateral component equal to 0) indicating that the character is leaning forward as compared to the downhill plot. In either case, the character achieves a stable gait.

3.4.2 Balance

An effective balance policy allows the controller to adapt the motion to new environments. In practice, the animator can change the terrain or the environment to produce numerous variations from a single reference motion. We adapted reference motions to sloped terrain, steps, and moving platforms. These animations would be difficult to create using traditional key framing tools or spacetime optimization. In addition to varying the terrain, the animator can introduce external disturbances.

As a test case, we manually tuned a PD controller which balances through a foot placement strategy to walk on flat ground [1]. The same controller also works on a downhill slope of two degrees but fails on slopes of three degrees or higher. To walk down steeper slopes, one could readjust the target poses of the manual controller and its gains. However, our controller does this automatically. Given the reference output from the successful simulation on flat ground, our approach can derive controllers that successfully simulate the same motion on slopes of up to five degrees. This experiment demonstrates that controller adapts to new ground slopes even without any user intervention (see Figure 3-6).

Our controller was not designed to handle higher level motion planning tasks. For example, it does not know how to adapt a normal walk to walk up large steps. Applying our controller would result in the character stubbing its toe and falling. One approach to this problem is to build a set of primitive actions with our controller and to compose them together at runtime using the surrounding feedback. Our approach in this chapter allows us to build such simple actions and we leave their automatic run-time composition for future work.

3.4.3 Composing Controllers

A library of motions can be used to derive many controllers, which can be composed together to produce complex actions. Similar ideas have been proposed in the past [29, 94, 71], but the approach in this chapter may make it simpler to derive such

[1] A SIMBICON controller produces stable walks on steeper slopes by departing from the original gait in a different way, namely by increasing step lengths and forward velocity[94].

controls for a wider range of motions. Moreover, the new control produces high-quality motions that could be used along with kinematic motion graphs [46, 7]. we used the action graph in Figure 3-7 to jump between normal and marching styles including transitions between starting and stopping. In fact, even our simple walks are transitions between controllers for the left and right leg. Currently, the allowable transitions are specified manually. In the future, we hope to construct these action graphs automatically by precomputing optimal transitions based on character state as has been done for kinematic controllers [54, 83].

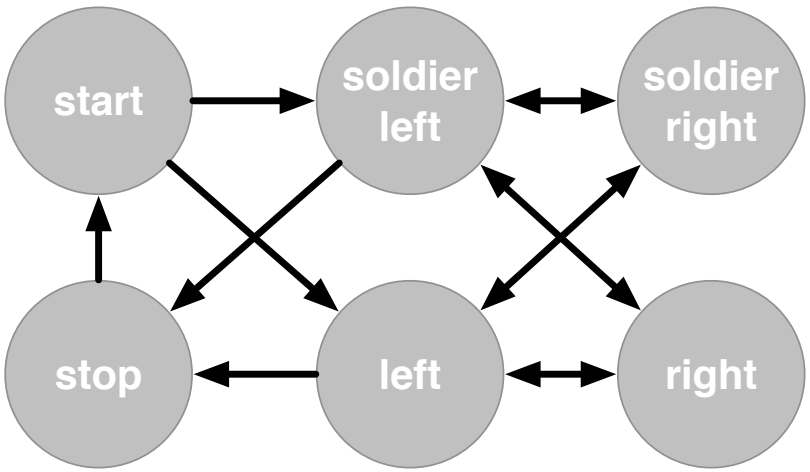

Figure 3-7: An action graph consisting of controllers for normal and marching styles of walking. Currently, allowable transitions are found manually. An interesting area of future work is constructing action graphs automatically by precomputing optimal transitions based on character state.

3.4.4 Experimental Setup

Physical Modeling and Simulation The motions were simulated using three different physical character models. The physical properties of these models are provided in Table 3.2. These models were constructed either to match the properties used from previous papers [71, 94] or to match our motion capture subject. We set bone lengths to match the subjects dimensions and computed inertial properties by integrating a density over a polygonal human-like mesh [87]. The joint stiffness gains are significantly lower than those used in previous controllers since most of the control

effort is generated from the QP component of our controller. This makes the control less stiff which leads to more stable simulations. The motions were simulated in Open Dynamics Engine (ODE) with scenes created in DANCE [69].

Foot Clean up The raw mocap data is noisy, particularly for toes and ankles which are essential for stepping motions. We cleanup these motions to ensure that contact between the feet and the flat ground is consistent. First, a constant offset is applied to the ankle joints to align the feet. It is computed using a reference frame where the feet are assumed to be flat. Next motion edits are performed to constrain the character's toes to be flush against the ground when in contact. We use a "per-frame" kinematic fix with hierarchical B-spline smoothing, similar to the approach employed by [49]. We manually annotate the segments of the motion with toe contacts and then compute the global transformation of the toe by projecting its position and orientation in the first frame of each segment onto the ground plane. A fast, analytic IK then adjusts the ankle, knee and hip joints minimally so that they connect with the positioned toe segments. The result of this per-frame adjustment is smoothed over the adjacent frames with a hierarchal B-spline. The displayed reference motions have all been fixed in this manner, except for the motions used in the previous work [71], which were used as is.

3.5 Summary

A control system that reproduces stylized locomotion has two major advantages. First, it can adapt motion-capture data to physically consistent dynamic environments. Second, it can produce large repertoires of simple actions. Both of these features are needed to create versatile characters in games and other interactive applications. In this chapter, we demonstrated such a control system for control of human stepping motions.

We showed that automatic analysis of linear time-varying systems enables efficient pre-computation of balance strategies. These strategies can be combined with a sim-

ple style feedback to simulate stylized human stepping motions. A possible additional application of our approach is combining it with existing controllers to improve their stability. For example, in the results section, we took a PD controller and extended its ability to walk on steeper slopes. Further investigation is needed to determine whether this additional feedback can improve stability of other controllers.

Large disturbances require higher level planning and more significant deviation from the reference motion. For example, a push might require taking several steps to retain balance. In this chapter, we demonstrate manual composition of a few actions stabilized by our control system. In the future, higher-level planners should be developed to accomplish similar tasks automatically to improve robustness and respond to user input. Using a complex graph of simple actions, a higher-level planner could monitor the current state of the character, its environment, and the provided user input to simulate versatile characters in rich physically simulated environments.

The pre-computed balance policy was derived by making some simplifying assumptions that allowed us to massage the HJB equation (2.6) into something that could be tractably computed. Furthermore, by focusing on controlling the character near a trajectory, we were able to represent a high dimensional value function while avoiding the curse of dimensionality. In the coming chapters, we'll see that related assumptions can help us turn the non-linear HJB equation into a linear equation. This will make it easier to solve the HJB equation, and will allow us to combine solutions together. We can use trajectory-based representations of value functions to do this for high-dimensional problems.

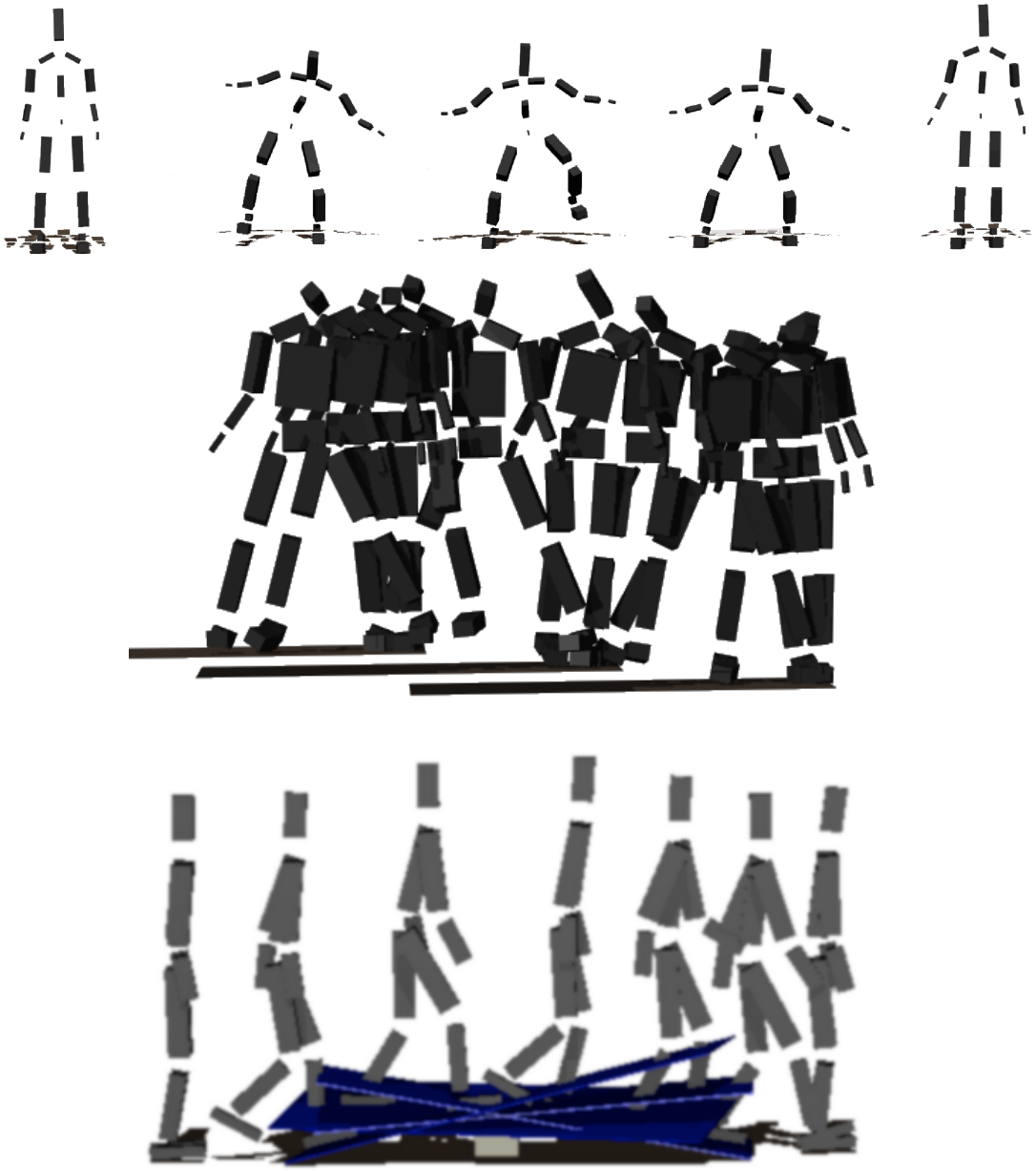

Figure 3-8: Some examples of the styles of motion simulated by our controller. Starting from motion capture makes it easy to simulate motions like a sumo maneuver. We can simulate reference motions in new environments, such as adapting a sideways stepping motion or a turning motion to walk down steps. The backwards walk is adapted to walk over a dynamic seesaw. The entire sequence is simulated creating a realistic motion that would be difficult to animate by hand, motion capture, or synthesize kinematically.

Chapter 4

The Linear Bellman Equation

In the previous chapter, we saw that assuming linear dynamics and quadratic loss functions allowed us to turn the HJB PDE into a Riccati ODE. This allowed us to efficiently pre-compute feedback policies for characters near reference trajectories. In this chapter, we review a slightly more general set of assumptions and a change of variables which transforms the non-linear HJB PDE into a linear PDE. In subsequent chapters, we will see that this linearity enables us to solve optimal control problems more easily and also allows us to combine existing solutions together.

Here we summarize the derivation of the linear HJB equation for finite horizon control problems. Note that the derivation here is a special case where noise acts on the system as if it were acting on control inputs. More general derivations where noise acts in different subspaces are possible [80] . The linear form of the Bellman equation was originally discovered in the context of computational physics [40]. Holland showed that the non-linear HJB equation was equivalent to the Schrodinger equation using a change of variables.

Suppose we have a control-affine system as in Equation (2.5) and a loss function

$$\ell(\boldsymbol{x}, \boldsymbol{u}, t) = q(\boldsymbol{x}, t) + \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{u}. \tag{4.1}$$

This loss function is more general than the strictly quadratic loss functions used in the last chapter. The penalty on control effort is quadratic but the position penalty

$q(\boldsymbol{x}, t)$ can be any positive function. Recall that the optimal value function satisfies the HJB equation

$$-\dot{v}(\boldsymbol{x}, t) = \min_{\boldsymbol{u}} \{\ell(\boldsymbol{x}, \boldsymbol{u}, t) + \mathcal{L}(v(\boldsymbol{x}, t), \boldsymbol{u})\} \tag{4.2}$$

$$v(\boldsymbol{x}, T) = g(\boldsymbol{x}). \tag{4.3}$$

For control-affine, stochastic systems, the operator $\mathcal{L}$ can be thought of as taking an expectation of the value function at an infinitesimal point in the future given the control action. Its functional form is

$$\mathcal{L}(v(\boldsymbol{x}, t), \boldsymbol{u}) = (\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u})^\top \nabla v(\boldsymbol{x}, t) + \frac{1}{2}\mathrm{tr}[\boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla^2 v(\boldsymbol{x}, t)].$$

Given the form of the loss function in Equation 4.1, the right hand side of the HJB Equation 4.2 can be minimized analytically with the policy

$$\boldsymbol{u} = -\boldsymbol{B}(\boldsymbol{x})^\top \nabla v(\boldsymbol{x}, t)$$

This control law is plugged into Equation 4.2 to yield

$$\begin{aligned} -\dot{v}(\boldsymbol{x}, t) = q(\boldsymbol{x}) &- \frac{1}{2}\nabla v(\boldsymbol{x}, t)^\top \boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla v(\boldsymbol{x}, t) \\ &+ \boldsymbol{a}(\boldsymbol{x})^\top v(\boldsymbol{x}, t) \\ &+ \frac{1}{2}\mathrm{tr}[\boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla^2 v(\boldsymbol{x}, t)] \end{aligned}$$

which is a non-linear partial differential equation as there are terms which are quadratic in the value function. We now use the change of variables

$$z(\boldsymbol{x}, t) = \exp\left(-v(\boldsymbol{x}, t)\right),$$

and observe that the transformed HJB equation is

$$\begin{aligned}\dot{z}(\boldsymbol{x},t) = q(\boldsymbol{x})z(\boldsymbol{x}) &- (\boldsymbol{a}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x},t) \\ &+ \frac{1}{2}\mathrm{tr}[\boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla^2 z(\boldsymbol{x},t)] \\ &+ \frac{1}{2\,z(\boldsymbol{x},t)}\nabla z(\boldsymbol{x},t)^\top \boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x},t) \\ &- \frac{1}{2\,z(\boldsymbol{x},t)}\nabla z(\boldsymbol{x},t)^\top \boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x},t).\end{aligned}$$

Note that the non-linear terms cancel, leaving a *linear*, second-order PDE. To solve, one can discretize the PDE into the form

$$z_t = M z_{t+\delta t}$$

where M can in general vary with time. In the next chapter, we'll see that solving control problems in the linear space is often advantageous. In addition, linearity will enable a new application: combination of existing controllers.

Chapter 5

Solving Linear Bellman Equations

Optimal value functions for continuous dynamical systems satisfy the Hamilton-Jacobi-Bellman (HJB) equation, a non-linear partial differential equation (PDE). This PDE can be solved using value iteration [19]. However, as we saw in the previous chapter, the HJB equation can be solved using an equivalent linear PDE for a large class of control problems. Linear PDEs can be solved with a number of stable and highly accurate numerical methods. Surprisingly, there has been no exploration of how to solve linear Bellman PDEs directly either in closed-form or with finite-difference or spectral numerical schemes. Instead, these linear Bellman PDEs have been solved using Monte Carlo integration [44]. The integration can be accelerated using importance sampling but solutions are only valid for a single initial condition. Recently, a linearly solvable Markov Decision Process [79] was shown to be equivalent to the continuous problem in the limit of infinitely fine grids [80]. This formulation produces solutions over regions of state space but requires the use of stochastic control policies.

In this chapter, we explore closed-form, finite-difference, and spectral methods for solving linear Bellman equations. In particular, we identify a subclass of problems for which there is a closed-form solution. The solution is computed through the use of Fourier series and highlights the connection of the Bellman equation to the heat equation. PDEs associated with control problems for non-linear systems such as a three-wheel omnidirectional autonomous vehicle, cannot be solved in closed-form.

However, we show how to discretize these PDEs with finite-difference and spectral methods. Depending on the particulars of the PDE, some methods will converge faster and be more accurate than value iteration.

Solutions to linear Bellman equations generate optimal control policies for a variety of dynamical systems. As a proof of concept, we generate policies for a one dimensional particle. This example has a closed-form solution which allows us to verify the accuracy of various numerical solution methods including value iteration. We show that a standard value iteration scheme for this problem is less accurate and more expensive than the easy to implement implicit schemes used to solve the linear Bellman form of the equation. We repeat this experiment for a second order dynamical system and show that a simple finite-difference scheme outperforms value iteration. Finally, we generate navigation feedback policies for an autonomous vehicle with three omnidirectional wheels. In three-dimensional problems, memory resources can be quickly exhausted when using fine grids. It's beneficial in this case to be able to use larger time steps while not having to sacrifice spatial accuracy.

5.1 The Linear Bellman Equation

Optimality for a control problem involving a stochastic dynamical system is described by a non-linear HJB equation. This equation describes the backwards time evolution of the optimal cost function. With quadratic control costs and certain noise models, a change of variables leads to a linear Bellman equation [34, 44, 79, 80]. This linear PDE describes the backwards time evolution of the optimal reward function:

$$z_t(\boldsymbol{x}, t) = q(\boldsymbol{x}, t)z(\boldsymbol{x}, t) - \mathcal{L}(z(\boldsymbol{x}, t)) \tag{5.1}$$

$$z(\boldsymbol{x}, T) = \exp(-g(\boldsymbol{x})) = h(\boldsymbol{x}) \tag{5.2}$$

where the dependence on z is *linear* [34, 44, 79, 80] and

$$\mathcal{L}(z(\boldsymbol{x}, t)) = \boldsymbol{a}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x}, t) + \frac{1}{2}\mathrm{Tr}(C(\boldsymbol{x})C(\boldsymbol{x})^\top \nabla^2 z(\boldsymbol{x}, t)).$$

The boundary condition of the PDE is given by the terminal cost function $g(\boldsymbol{x})$ transformed into the linear Bellman space by exponentiation. The optimal value function can be recovered from the solution to the PDE by $v(\boldsymbol{x}, t) = -\log\big(z(\boldsymbol{x})\big)$. The optimal policy then descends the gradient of the optimal value function as closely as possible,

$$\boldsymbol{u}^* = -\boldsymbol{B}(\boldsymbol{x})^\top \nabla v(\boldsymbol{x}) = \boldsymbol{B}^\top \nabla \log(z(\boldsymbol{x})) = \frac{\boldsymbol{B}^\top \nabla z(\boldsymbol{x})}{z(\boldsymbol{x})}.$$

The PDE described by Equation 5.1 is a Fokker-Planck equation typically found in physics applications [67]. The PDE resembles a convection-diffusion equation with an extra reaction term. The convection term at each grid point depends on the passive dynamics (control equal to zero) of the system. The standard diffusion term depends on the directions that control actions can change the system. The reaction term depends on the position cost.

Typically, non-linear HJB equations are solved in cost space using value iteration. For the class of problems which lead to a linear Bellman equation, the standard value iteration algorithm solves a non-linear advection equation with an explicit time stepping scheme. This leads to stability problems when using large time steps with fine spatial grids. In the linear reward space, it is easy to apply implicit schemes that allow for larger time steps. Furthermore, as described in the next section, certain special cases can be solved in closed-form.

5.2 A Closed-Form Solution

The linear Bellman equation is a convection-diffusion equation. As with the heat equation in physics, the linear Bellman equation can be solved in closed-form when the convection and diffusion coefficients are constant. This allows us to derive a closed-from solution for control problems of the form

$$\min_{\boldsymbol{\pi}} E\left[g(\boldsymbol{x}) + \int_0^T c + \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{u} \mathrm{d}t\right]$$

where c is a constant. The terminal cost function g need not be quadratic, making this a more general solution than the related linear quadratic regulator problem [19].

In this case, the linear Bellman becomes $-z_t = az_x + \frac{1}{2}B^2 z_{xx}$ where the position cost is assumed to be zero. Also, we stick to the one-dimensional case for now but the solution can generalize to many dimensions. The solution is found by assuming a Fourier series solution $z = \sum \hat{z}_i(t)e^{ikx}$. Plugging this guess into the PDE yields

$$-\sum \dot{\hat{z}}_i(t)e^{ikx} = a\sum ik\hat{z}_i(t)e^{ikx} - \frac{1}{2}B^2\sum k^2\hat{z}_i(t)e^{ikx}.$$

By orthogonality of the wave functions e^{ikx}, we have

$$-\dot{\hat{z}}_i = (aik - \frac{1}{2}B^2k^2)\hat{z}_i,$$

which has the solution

$$\hat{z}_i(t) = \hat{z}_i(0)e^{(\frac{1}{2}B^2k^2 - aik)t}.$$

This solution has an intuitive behavior. Starting from the final time and going backwards, the terminal reward function $\exp(-g(x))$ is translated and diffused, where the higher frequencies are attenuated more rapidly. One can get a solution for the case where $q(x) = c$ is constant by linearity. The effect is extra frequency-independent smoothing of the terminal cost function.

It is rare for optimal control problems to have closed-form solutions. While the assumptions here are fairly restrictive, this solution method could be useful as an approximation for controlling high-impedance robots. It could also be useful for local approximations of control problems. In this chapter, we use the closed-form solution to analyze different numerical solutions as an alternative to value iteration.

5.3 Numerical Solutions

For non-linear dynamical systems and more general position costs, linear Bellman PDEs must be solved numerically with a time-stepping method. Fortunately, linearity

allows us to apply any number of well-studied numerical tools. For example, stability analysis of finite-difference schemes state that the convergence of an explicit method requires a specific relationship between the time-step and grid size. For schemes that solve the non-linear HJB, like value iteration, evaluating convergence is more complicated. Furthermore, implicit methods which are unconditionally stable with respect to time-step and grid size are readily applicable to the linear Bellman form of the problem. Implementing an implicit value iteration scheme, in contrast, would be difficult and inefficient as it would require solving a non-linear equation.

Even in the linear case, there are issues that can arise depending on the solution method and the particulars of the equation being solved. This is true especially in the case when diffusion terms are lacking. This can make it difficult to solve advection using higher-order linear methods. In this chapter, we evaluate a number of finite-difference schemes and a pseudospectral method combined with explicit and higher-order implicit time-stepping schemes.

5.3.1 Time-Stepping

Since the PDE is linear, the derivative operator can be expressed as a matrix $\mathbf{M}$ making it easier to describe each time-stepping scheme. Recall that the linear Bellman equation runs backwards in time, so that in each scheme below, z^{i+1} is known and we are solving for z^i. The time step is h.

Forward Euler (FE)

$$z^i = z^{i+1} - h\,\mathbf{M}z^{i+1}.$$

In practice, this time stepping method is difficult to use. In addition to being only first order accurate, the scheme is always unstable when coupled with negative-definite derivative operators such as those that arise from central differencing schemes. With symmetric differencing schemes, one must take care to restrict the time step to be smaller than the CFL condition for the particular dynamical system [50].

Implicit Euler (BE)

$$z^i = z^{i+1} - h\,\mathbf{M}z^i.$$

This time stepping method is unconditionally stable but only first order accurate [50].

Crank-Nicolson (CN)

$$z^i = z^{i+1} - \frac{1}{2}h(\mathbf{M}z^i + \mathbf{M}z^{i+1}).$$

Crank-Nicolson is second-order accurate and unconditionally stable implicit scheme [50].

Implicit Runge-Kutta 3 (IRK3)

$$z^i = (\frac{1}{3}\mathbf{I} + \frac{2}{3}\hat{\mathbf{M}}(\frac{3}{4}\mathbf{I} + \frac{1}{4}\hat{\mathbf{M}}\hat{\mathbf{M}}))z^{i+i}$$

where $\hat{\mathbf{M}} = \mathbf{I} - h\mathbf{M}$. This update is a third order and implicit Runge-Kutta 3 method which was combined with pseudospectral estimates of the spatial derivatives.

The implicit time stepping schemes can be performed efficiently by first doing an LU decomposition of the update matrix and using back substitution for each update.

5.3.2 Spatial Derivatives

The linear Bellman equation includes first and second derivatives of the transformed value function. Here we evaluate three finite-difference and spectral methods: an upwind method, central differences, or a Fourier spectral method. Each method has its own accuracy and stability requirements. The upwind method computes the first derivative by applying the two-point stencil $\frac{1}{\Delta x}(-1, 1)$ in the direction of the advection field. Upwind can be overly diffusive and is only first order accurate. However, this artificial diffusion can also prevent strong oscillations from developing when advection terms in the linear Bellman are not coupled with real diffusion terms. The central

difference method computes the first derivative by applying the three-point stencil $\frac{1}{2\Delta \boldsymbol{x}}(-1, 0, 1)$ and the second derivative with the three-point stencil $\frac{1}{\Delta \boldsymbol{x}^2}(1, -2, 1)$. Central-difference operators are second- order accurate but non-symmetric, so they must be coupled with implicit time stepping schemes. We also evaluate a matrix based Fourier spectral method. The differencing matrix consists of columns of the form $(\ldots, -1/2\cot(2\Delta \boldsymbol{x}/2), 1/2\cot(1\Delta \boldsymbol{x}/2), 0, 1/2\cot(1\Delta \boldsymbol{x}/2), \ldots))$ (see [82] for a reference on spectral methods). This method can give very accurate results when coupled with a high order time-stepping scheme. However, even a simple upwind method can yield superior results to value iteration.

5.4 Results

We evaluate our closed-form and numerical solution methods on a set of three test cases: a one-dimensional particle example, a two-dimensional brick on ice problem, and a three-dimensional robot navigation example.

5.4.1 One-Dimensional Particle

The simplest possible example that can be controlled using the linear Bellman formulation is a one-dimensional particle with the dynamics

$$\mathrm{d}\boldsymbol{x} = (1 + u)\mathrm{d}t + \mathrm{d}\boldsymbol{\omega}.$$

where $\boldsymbol{x}$ is the position of the particle and u is the control, in this case a commanded velocity. The cost function used is

$$2\,x(T)^2 - x(T)^4 + \int_0^T 1 + \frac{1}{2}\,u(t)^2\,\mathrm{d}t.$$

With this periodic cost function and dynamics, the closed-form solution can be used to verify numerical results. The linear Bellman PDE for this problem is

$$-z_t = z_x + \frac{1}{2} z_{xx} - z \tag{5.3}$$

$$z(x, T) = 2\, x(T)^2 - x(T)^4 \tag{5.4}$$

$$z(-1, t) = z(1, t). \tag{5.5}$$

5.4.2 Brick on Ice

For a two-dimensional problem, we study a second order system

$$\mathrm{d} \begin{pmatrix} x \\ v \end{pmatrix} = \left(\begin{pmatrix} v \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \right) \mathrm{d}t + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathrm{d}\boldsymbol{\omega},$$

where x is the position of the brick, v is the velocity, and u is the lateral force on the brick. In this case, control cannot directly alter the position of the brick. The cost function is

$$100\, x(T)^2 + 10\, v(T)^2 + \int_0^T 100\, x(t)^2 + 10\, v(t)^2 + \frac{1}{2}\, u(t)^2 \, \mathrm{d}t.$$

This cost function places more emphasis getting to the position zero than velocity zero. The linear Bellman PDE for this problem is

$$-z_t = v z_x - (100\, x(t)^2 + 10\, v(t)^2) z \tag{5.6}$$

$$z(x, T) = 100\, x(T)^2 + 10\, v(T)^2 \tag{5.7}$$

$$z(x \in \partial\Omega, t) = 0. \tag{5.8}$$

5.4.3 Three Omni-Wheeled Robot

The third example examines the control of a three wheeled robot as shown in Figure 5-1. Each wheel can move in any direction though certain directions are faster than others. We assume that the velocity of the wheels can be controlled directly leading

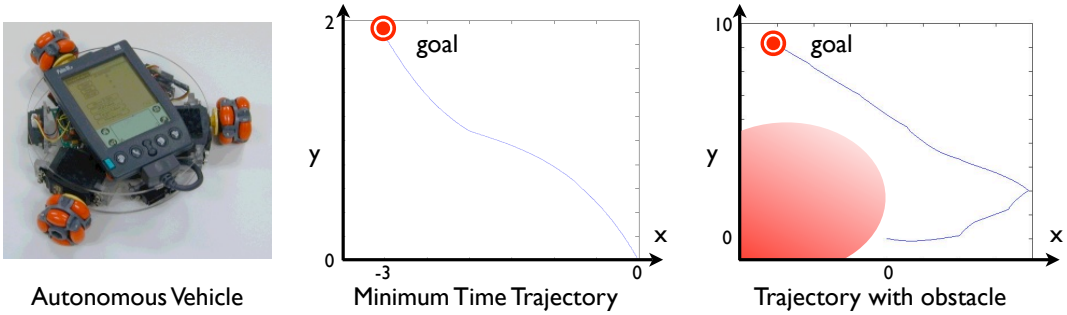

Figure 5-1: Solutions to the linear Bellman equation can be used to generate optimal trajectories for a three-wheeled vehicle shown here on the left (image from http://www.cs.cmu.edu/∼reshko/PILOT/). The wheels can move in any direction but some directions are faster than others. This causes the vehicle to move along circular paths to get to the goal as quickly as possible as shown in the center plot. In the figure on the right, the vehicle first moves to the right to allow itself room to avoid the obstacle.

to the following first order system dynamics:

$$\mathrm{d}\boldsymbol{x} = \frac{2}{3}\begin{pmatrix} -\sin(\theta_1) & -\sin(\theta_2) & -\sin(\theta_3) \\ \cos(\theta_1) & \cos(\theta_2) & \cos(\theta_3) \\ 1/2 & 1/2 & 1/2 \end{pmatrix}(u\,\mathrm{d}t + \mathrm{d}\boldsymbol{\omega})$$

where θ_1 is the orientation of the first robot with respect to the horizontal axis, $\theta_2 = \theta_1 + 120$, and $\theta_3 = \theta_1 + 240$. The state is three dimensional consisting of the position and orientation of the robot. The control is the commanded velocity of each wheel.

In this case, two cost functions are used:

$$\frac{1}{2}\left((x(T)-\bar{x})^2 + (y(T)-\bar{y})^2 + \int_0^T (x(t)-\bar{x})^2 + (y(t)-\bar{y})^2 + \boldsymbol{u}(t)^\top \boldsymbol{u}(t)\,\mathrm{d}t\right)$$

where $\bar{x}$ and $\bar{y}$ are the desired position of the robot (there is no desired orientation), and

$$\frac{1}{2}\left((x(T)-\bar{x})^2 + (y(T)-\bar{y})^2 + \int_0^T 1 + \boldsymbol{u}(t)^\top \boldsymbol{u}(t)\,\mathrm{d}t\right).$$

Note that we also modify this objective function to penalize certain states where there are obstacles. For long time periods, this cost function approximates the minimum

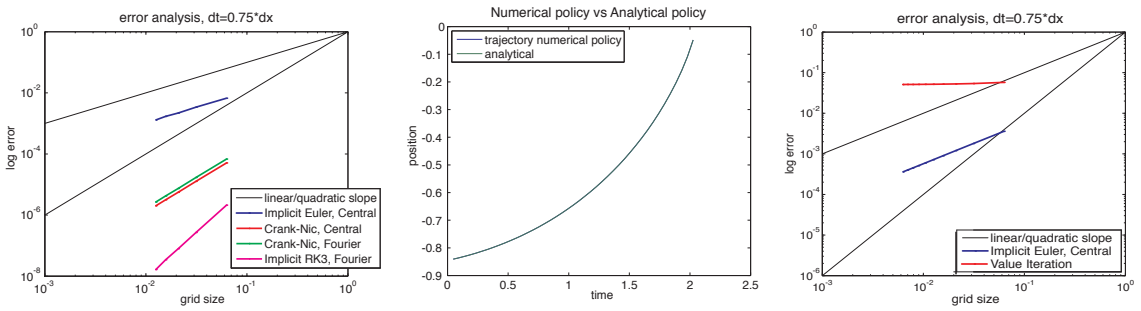

Figure 5-2: On the left is an error analysis of various numerical schemes for the one-dimensional particle example. These plots were generated by comparing the numerical solution at the final time (in this case the reward function at time 0) with the closed-form solution using the $\ell - \infty$ norm. The middle is the trajectory of a one-dimensional particle under the numerically computed (IRK3 spectral) and closed-form policies. The two trajectories are indistinguishable even though there are some small errors in the numerically computed policy. For this plot, the grid size is 0.0126, though similar results are obtained for grids as coarse as 0.3927 units per grid point (500 versus 16 grid points). The figure on the right compares value iteration of the non-linear HJB with an implicit method for solving the linear Bellman equation. Shown is the largest error in the value function for the particle test case after half a second. The error for value iteration is shown in red while the error for the implicit Euler method is shown in blue. For equal time steps and grid sizes, solving in reward space is more accurate than using value iteration in cost space.

time problem. The minimum time problem determines the quickest way to get to a desired location while using as little velocity as possible. Interestingly, all possible solutions for this problem have been enumerated through analysis of the minimum principle [11]. We will use the linear Bellman equation to generate trajectories for the robot that avoid obstacles. The linear Bellman equation in this case is

$$\begin{aligned} -z_t =& \frac{2}{9}((\sin^2(\theta_1) + \sin^2(\theta_2) + \sin^2(\theta_3))z_{xx} \\ & - 2(\sin(\theta_1)\cos(\theta_1) + \sin(\theta_2)\cos(\theta_2) + \sin(\theta_3)\cos(\theta_3))z_{xy} \\ & - (\sin(\theta_1) + \sin(\theta_2) + \sin(\theta_3))z_{x\theta_1} + (\cos^2(\theta_1) + \cos^2(\theta_2) + \cos^2(\theta_3))z_{yy} \\ & + (\cos(\theta_1) + \cos(\theta_2) + \cos(\theta_3))z_{y\theta_1} + \frac{3}{4}z_{\theta_1\theta_1}) - z. \end{aligned} \tag{5.9}$$

5.4.4 Error Analysis

The one-dimensional test problem has constant drift, diffusion, and reaction terms. This problem can be solved in closed-form and the infinity norm was used to evaluate numerical solution methods. A plot of the error for various numerical methods is shown in Figure 5-2. Not surprisingly, the most accurate method is the third-order spectral method. Interestingly, the Crank-Nicolson time stepping with spectral approximation of the spatial derivatives had a slightly larger error constant than the central difference version. Computationally, the central difference method is much less expensive than the spectral method because the spectral method involves inverting a dense matrix.

In the end, the computed solution generates a feedback policy to control the dynamical system. Errors in the solution result in higher costs of trajectories as given by the objective function for the problem. However, the feedback policies are typically robust to these errors as shown in Figure 5-2. While the system may end up slightly farther from the goal, it does not veer wildly off course due to small errors in the solution.

5.4.5 Comparison to Value Iteration

An alternative to solving the linear Bellman equation in reward space is to solve the original non-linear HJB problem in cost space using value iteration. However, with the linear form it is easier to determine the accuracy and stability of the numerical approach. Furthermore, the simplest implicit time stepping scheme often outperforms value iteration in accuracy and computational performance. We demonstrate these properties in Figures 5-2 and 5-3. Shown are two experiments demonstrating the performance and accuracy advantages of solving the linear Bellman form of the control problem over using value iteration to solve the non-linear HJB form. Value iteration is less accurate and stable because it uses an explicit update rule. It would not be straightforward to implement an implicit version of value iteration. Also, value iteration is computationally more expensive because it requires computing the optimal

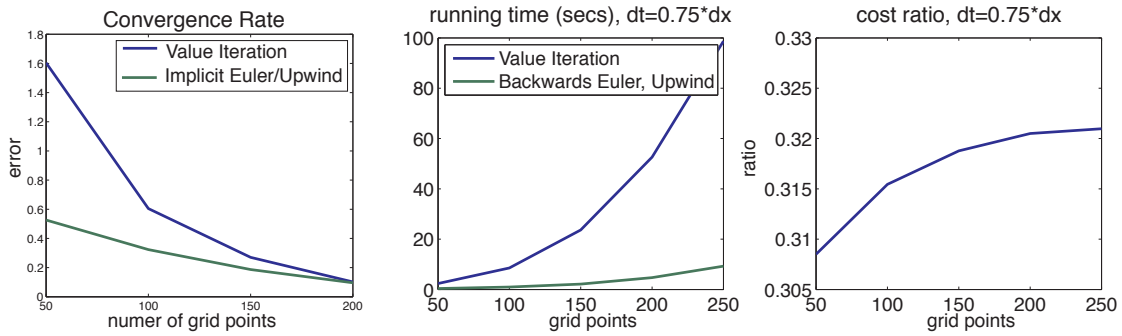

Figure 5-3: The left figure plots the convergence rates of value iteration and the implicit Euler method with upwind differencing for the brick problem. The middle figure compares the running time of value iteration of the non-linear HJB versus an implicit method solving the linear Bellman form for the brick on ice problem. Solving the linear Bellman is considerably faster than value iteration. Finally, the rightmost figure shows the ratio of the total cost of the policy computed by solving the linear Bellman over the total policy cost as computed by value iteration. For the same grid size and time step, the linear Bellman policy is less than a third of the cost of the policy computed by value iteration.

policy at each time step, simulating it for each grid point, and interpolating the value function at each result. Finally, the policies that value iteration computes for a given grid size are more costly than the corresponding policies computed by solving the linear Bellman form of the problem.

5.4.6 Navigation policies

Two objective functions were used to create controllers for the three-wheeled vehicle. One controller tried to reach a point as quickly as possible while minimizing squared velocity. The other tried to reach a point while avoiding a costly region of state space (costly because there is an obstacle there for example). Figure 5-1 shows the trajectories of the three wheeled robot under these policies. As predicted by analysis of the Pontryagin minimum principle [11] for the minimum time problem, the vehicle moves in a sequence of circular arcs and shuffles. In three-dimensions, memory requirements become an issue when storing time-varying policies. It is important that the solution method is as accurate as possible with the limited resources available. With the linear form of the control problem, it is easier to apply more accurate methods.

5.5 Discussion

The continuous linear Bellman equation describes (after a non-linear transformation) the reverse time evolution of the optimal value function. It is a convection-diffusion-reaction equation. The passive dynamics of the system determine the advection terms, the control directions determine the diffusion terms, and the cost function determines the reaction term. Under the PDE, the terminal cost function is advected and blurred. Near the terminal time, the optimal value function is a slightly warped and blurred version of the terminal cost function. Far from the terminal time, it is hard, in general, to predict what the optimal value function will look like. However, with the linear Bellman formulation, one can apply many stable and accurate methods to compute this numerically. The value function then gives optimal control policies.

This chapter has shown that linearity allows one access to several useful theoretical and numerical tools for solving stochastic optimal control problems. For dynamical systems with constant convection, diffusion, and constant running terms there is closed-form solution. This solution may prove useful for local approximations to more complicated control problems. For more general problems, the PDE can be solved efficiently using standard finite-difference or spectral methods. Solving the linear PDE provides a faster and more accurate alternative than value iteration. The linear PDE is not applicable to all control problems, however, and value iteration is more generally applicable to problems without control affine dynamics, quadratic control penalties, or bounds on control.

As in value iteration, a key limitation of solving the linear Bellman equation using standard grid techniques is that it is limited to low dimensional problems. Humanoid robots have tens of degrees of freedom. These problems cannot be solved on a grid. Based on the accuracy of the spectral method presented with high-order time stepping, it is conceivable that one could go beyond three dimensions. An interesting direction would be to use sparse grid representations like particle methods coupled with model reduction on the system dynamics to solve even higher dimensional problems. Particle methods are ideally suited to solving advection PDEs.

Another consequence of linearity is that we can apply superposition to add solutions of the linear Bellman equation together. In the next chapter, we will exploit this to combine pre-computed controllers to perform new tasks at almost no computational cost. We will introduce a method for approximating solutions for high-dimensional problems.

Chapter 6

Linear Bellman Combination for Control of Character Animation

Physically-based animation of active characters requires controllers that find forces that achieve the goals of the animator. While controllers exist for a variety of human activities [94, 24, 39, 96, 71] and even complex, non-human activities [56, 31, 18], creating a controller that achieves the animator's goal usually involves either intensive manual tuning or expensive computational optimization.

Given the difficulty of creating individual controllers for given tasks, a goal of this work is to reuse pre-computed controllers for new tasks by combining them. For example, if two controllers can reach different goals, we want to blend them to reach end states in between. Simple linear combination of controllers is not sufficient and can lead to unpredictable results due to the complex relationship between control forces and the resulting animation. In this chapter, we present a new method for combining controllers that naturally weights each component controller by its relevance for the current state and goal. Combined controllers can interpolate goals or coordinate between multiple controllers. Coordination between controllers trained for different parts of state space can broaden the domain that can be handled. Coordination between controllers trained for different goals can yield a controller with multiple acceptable outcomes.

Our approach is based on the control formulation where controllers (equivalently,

control policies) are measured by an objective function which quantifies how well they achieve the goals of the animator [19, 78, 88]. For example, this function can emphasize reaching a particular state at a particular time or tracking a given trajectory for the character. In computer animation, improving a controller with respect to this objective is difficult because there are often many control parameters. Also, the relationship between these parameters and the resulting value of the objective function is non-linear, due to the cost functions used to describe the task and the dynamics that govern the motion of the character. This relationship is encapsulated by a partial differential equation, the Hamilton-Jacobi-Bellman (HJB) equation, whose solution defines the optimal *value* function [17]. The optimal value function maps a character state to the lowest possible cost of reaching the goal. The best policy seeks to follow the gradient of the optimal value function. Non-linearity of the HJB equation is the reason tuning, optimizing, or combining controllers is so difficult.

Our approach, which we call linear Bellman combination, derives from the linear form of the Bellman equation as described in the previous chapters. We've shown that the linear form is advantageous for solving optimal control problems. In this chapter, we observe that linearity allows us to apply the superposition principle to combine temporally aligned optimal policies for similar tasks with different boundary conditions. The resulting controller is provably optimal for the new task under some conditions. In practice, we need to combine controllers that might not be optimal. Still, linear Bellman combination performs accurately and robustly and we provide empirical evidence that it outperforms alternatives. Furthermore, it is simple to use and can be applied to many control schemes as it treats controllers as a black box, using only their outputs to compute combinations.

Evaluating a controller using linear Bellman combination requires representations of each component controller's policy and value function. Often these mappings are lacking and the common approach for low-dimensional systems is to tabulate the policy and value function. In this chapter, we also describe a method capable of controlling character motions with many more degrees of freedom. We demonstrate linear Bellman combination on value functions which are approximated around sam-

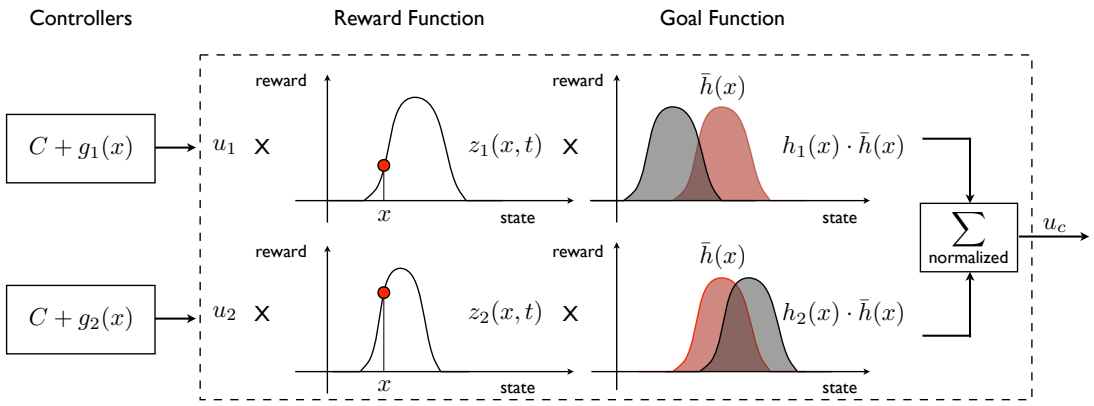

Figure 6-1: **Overview.** Linear Bellman combination creates a new controller using pre-computed controllers optimized for related tasks of finite duration. The tasks share a common *integrated* cost, $\int_0^T q(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{u} \mathrm{d}t$, but have different terminal costs $g_i(\boldsymbol{x})$. The output of each controller, $\boldsymbol{u}_i$, is scaled by a reward function, $z_i(\boldsymbol{x}, t) = \exp\big(-v(\boldsymbol{x}, t)\big)$ and by the amount the controller's terminal reward function, $h_i(\boldsymbol{x}) = g_i(\boldsymbol{x})$, overlaps the desired terminal reward function, $\bar{h}(\boldsymbol{x})$. These weighting factors are normalized and the resulting control action is a sum as in Equation 6.6.

ples from example trajectories.

Control-inspired approaches are used in many areas of graphics, offering many potential applications of the linear Bellman equation. For this approach, we focus on several animation tasks. As an example application, we demonstrate taking steps of varying length and jumps of different heights. Controllers that bring a gymnast to an upright position on a high bar from different initial conditions are combined to work from a range of initial conditions. We use coordination to combine controllers for walking on flat and hilly terrain. We show that linear Bellman combination outperforms simple linear blending and other tracking approaches. The main limitation of the presented formulation is that combined controllers must cover the same duration of time. In the conclusion, we sketch possible extensions to address this limitation.

6.1 Related Work

The goal of our work is to combine pre-computed controllers to perform new tasks. Others have explored parameter tuning for specific control schemes to achieve new tasks such as adapting running controllers from larger characters to smaller charac-

ters [38], modifying jumping controllers [61], or walking under various environmental constraints [93]. In contrast, we treat existing controllers as a black box which we do not modify directly. These existing controllers may be hand-crafted [89] or computer optimized [88]. Rather than manipulate the parameters of the controller, our method achieves new tasks by combining the output of each component controller.

Some prior work has explored interpolation of existing controllers in order to achieve new tasks [84]. Yin et al [93] observed that simple linear interpolation of SIMBICON [94] parameters can be used to adapt the walking controller to conditions that are "in-between" the conditions expected by example controllers. This observation was expanded upon to build a controller capable of taking different step lengths to avoid holes in the terrain [24]. In this chapter, we show that simple linear interpolation of feedback controllers can be suboptimal both in terms of the goal and the amount of control effort being exerted. In some cases, sub-optimality results in visual imperfections while in others it results in failure to accomplish the task. In practice, this can be overcome by placing optimized examples closer and closer together, but these optimized examples are difficult to obtain.

Controllers can also be sequenced to create composite controllers that achieve new tasks [22, 29]. In our approach, multiple component controllers may be active at any one time. The total control effort is a weighted sum of the output of each component controller. In contrast to a sequencing approach, combining controllers allows us to achieve tasks that are not executed by any of the component controllers.

Sok et al [71] and Erez et al [28] also interpolate example controllers to achieve robust execution of biped locomotion maneuvers. These approaches rely on hand-crafted metrics to determine the blend weights to place on each example controller. In contrast, the metric used to weight component controllers in our approach is automatically computed given a mathematical description of the current goal. This allows our composite controller to look ahead to determine the best course of action to achieve this goal.

Linear Bellman combination builds on the observation that the non-linear HJB equation can be linearized under a change of variables [34]. The same general idea

also appears in the study of stochastic diffusion processes [40]. Recently, this linearity has been exploited to derive methods for computing low-dimensional optimal value functions using a power method [79] and importance sampling [44]. In our work, we observe that linearity of the HJB equation allows one to apply linear superposition to combine existing controllers for new tasks for a variety of systems. Concurrent with our work, Todorov also explored linearity for the composition of control laws and applied it to a first-exit control problem of a discrete two-dimensional particle [81].

For high-dimensional characters, linear Bellman combination uses multiple quadratic expansions of the value function in a non-parametric representation to track multiple trajectories. A related non-parametric representation was proposed by Atkeson et al [9]. In our case, example trajectories need not be optimized for the same task. In addition, rather than switch between example controllers discretely using whichever example is closest to the current state, controls are smoothly interpolated using the value function. Some recent work [27, 15] has adapted feed-forward controllers using linear quadratic regulators (LQR) to track a single example trajectory. Earlier work on this topic was pursued by Brotman and Netravali in graphics [21] and Atkeson in robotics [8]. However, LQR controllers are only valid near the example trajectory. We show that linear Bellman combination can coordinate multiple controllers to improve controller robustness.

6.2 Linear Bellman Combination

Linear Bellman combination produces a new controller by combining existing control policies. For example, given two policies for stepping near and far, the combined policy can step in between. A control policy $\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{x}, t)$ defines the actions (forces) $\boldsymbol{u} \in \mathbb{R}^m$ needed to drive the system from the current state $\boldsymbol{x} \in \mathbb{R}^d$ and time t to some goal in the future. Liner Bellman combination blends a set of control policies

$\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n$ to produce a combined policy:

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \alpha_1(\boldsymbol{x}, t)\boldsymbol{\pi}_1(\boldsymbol{x}, t) + \cdots + \alpha_n(\boldsymbol{x}, t)\boldsymbol{\pi}_n(\boldsymbol{x}, t).$$

Setting these coefficients, α_i, such that the resulting controller achieves a well defined goal can be difficult given the non-linear dynamics of the systems typically controlled in physically based animation. Linear Bellman combination enables a principled derivation of these coefficients.

Linear Bellman combination uses carefully computed time-varying coefficients,

$$\alpha_i(\boldsymbol{x}, t) = \frac{w_i z_i(\boldsymbol{x}, t)}{\sum_{i=1}^{n} w_i z_i(\boldsymbol{x}, t)}, \tag{6.1}$$

that can—under certain conditions—generate *optimal* control actions for a well defined task. The blending formula weights each component policy by how appropriate it is for the current state and goal (see Figure 6-1). The reward function, $z_i(\boldsymbol{x}, t)$, indicates how appropriate the policy is for the current state, while the fixed (in-time) weights, w_i, determine the goal of the controller. The time-varying reward weights are computed automatically given the controller's value function. The fixed weight parameters are also computed automatically given a new desired goal function. The resulting control can either interpolate goal functions or coordinate the component policies. Coordination combines controllers that share the same goal but were created for different regions of state space. Coordination can also create a goal function with multiple acceptable outcomes.

The blending formula is derived from a linear form of the Bellman equation. In this section, we review the Bellman equation to associate solving this partial differential equation with the computation of optimal control policies. We then state the linear form of this partial differential equation and apply the linear superposition principle to derive Linear Bellman combination. Finally, we discuss the conditions for which this combination yields an optimal control policy.

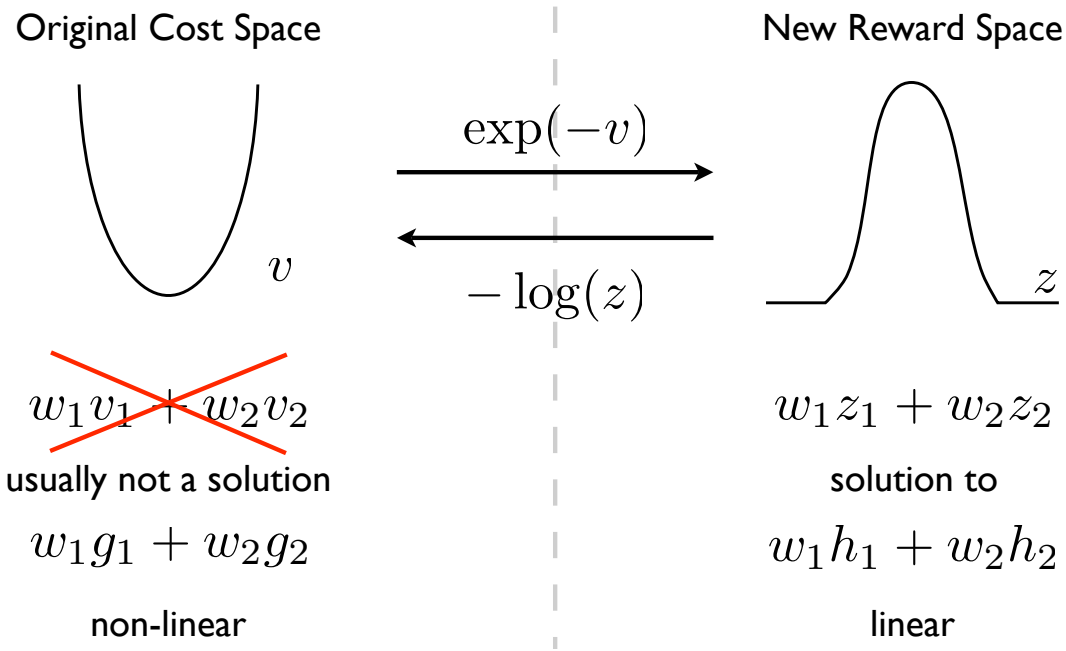

Figure 6-2: **Cost Space-Reward Space.** The HJB equation is a non-linear partial differential equation describing the time derivative of the cost-to-go function. Non-linearity means that linear combinations of optimal value functions are *not* a new optimal value function. However, using a non-linear mapping from cost to rewards space leads to a linear PDE for the time evolution of a reward function. In this space, linear combinations are solutions to an optimal control problem.

6.2.1 Linear Bellman Equation

As illustrated in previous chapters, using an exponential mapping that maps costs v into rewards $z = \exp(-v)$, the HJB equation can be transformed into the following form:

$$\mathrm{D}_t[z(\boldsymbol{x},t)] = q(\boldsymbol{x},t)z(\boldsymbol{x},t) - \mathcal{L}(z(\boldsymbol{x},t)) \tag{6.2}$$

$$z(\boldsymbol{x},T) = \exp\big(-g(\boldsymbol{x})\big) = h(\boldsymbol{x}) \tag{6.3}$$

where the dependence on z is *linear* [34, 44, 79, 80] and the differential operator no longer depends on control:

$$\mathcal{L}(z(\boldsymbol{x},t)) = \boldsymbol{a}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x},t) + \frac{1}{2}\mathrm{tr}[\boldsymbol{C}(\boldsymbol{x})\boldsymbol{C}(\boldsymbol{x})^\top \nabla^2 z(\boldsymbol{x},t)].$$

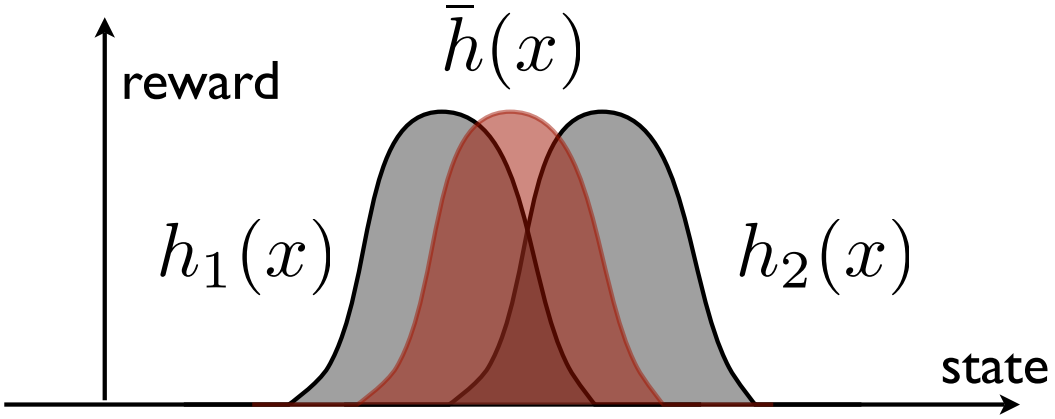

Figure 6-3: **Projecting Terminal Cost Functions.** Since we restrict terminal cost functions, $g(\boldsymbol{x})$, to be quadratic, we have Gaussian reward functions, $h(\boldsymbol{x})$, after exponentiating. To compute the fixed blend weights, w_i, we must project the desired terminal reward function, $\bar{h}(\boldsymbol{x})$ onto the example terminal rewards associated with each component controller, $h_1(\boldsymbol{x})$ and $h_2(\boldsymbol{x})$.

The core observation of this chapter is that linearity allows us to apply superposition to derive a combined controller that solves a new task (see Figure 6-2). Suppose we have several solutions to Equation 6.2 $z_i(\boldsymbol{x}, t)$ for different boundary conditions, i.e. terminal reward functions $h_i(\boldsymbol{x})$. Then, by superposition we also have the solution to the same PDE with the terminal reward function

$$h(\boldsymbol{x}) = \sum_{i=1}^{n} w_i h_i(\boldsymbol{x}). \tag{6.4}$$

The solution is

$$z(\boldsymbol{x}, t) = \sum_{i=1}^{n} w_i z_i(\boldsymbol{x}, t). \tag{6.5}$$

To get back to standard value function space, we apply the inverse transformation, $v(\boldsymbol{x}, t) = -\log z(\boldsymbol{x}, t)$. Applying this transformation, we see that superposition automatically gives a solution to the HJB equation with the terminal condition, $g(\boldsymbol{x}) = -\log\big(w_1 h_1(\boldsymbol{x}) + \cdots + w_n h_n(\boldsymbol{x})\big)$. Later, we will describe how to determine the weights w_i by projecting a desired terminal cost function onto the example terminal cost functions.

6.2.2 Time-Varying Coefficients

To derive the coefficients of the linear Bellman combination of Equation 6.1, note that the optimal policy for control problems with control-affine dynamics and loss functions that are quadratic in control actions is $-\boldsymbol{B}(\boldsymbol{x})^\top \nabla v(\boldsymbol{x}, t)$ [19]. This policy descends the value function in the steepest direction allowed by the current configuration of the system. Using this policy, we see that Equation 6.5 leads to

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \boldsymbol{B}(\boldsymbol{x})^\top \nabla \log \left(\sum_{i=1}^{n} w_i z_i(\boldsymbol{x}, t) \right).$$

Taking the derivative of the log and applying the chain rule leads to:

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \frac{1}{z(\boldsymbol{x}, t)} \sum_{i=1}^{n} w_i z_i(\boldsymbol{x}, t) \boldsymbol{\pi}_i(\boldsymbol{x}, t), \tag{6.6}$$

6.2.3 Summary

In summary, given a set of control policies, linear Bellman combination creates a new policy that optimizes a new terminal reward function which is a weighted combination of example terminal reward functions. The combined policy weights each component policy by how inexpensive it is as indicated by its reward function z_i and by its relevance to the goal indicated by the weight w_i.

The optimality of policies described by Equation 6.6 depends on several factors. First, the form of the linear PDE in Equation 6.2 must be shared by each solution in order for superposition to hold. Changing the dynamics of the system changes the PDE as does changing the loss function. In addition, the component policies must also be optimal for the combination to be optimal. In practice, it is difficult to compute optimal policies for high-dimensional systems. However, our experiments show that linear Bellman combination is more effective than naive blending approaches even when the component policies are known to be suboptimal.

The presented formulation was derived assuming continuous system dynamics. In many simulation engines, character dynamics can be discontinuous as a result of

contact with the environment. In experiments involving contact, we used penalty based contact forces to make our dynamics as smooth as possible. The disadvantage of this is that contact constraints cannot be enforced exactly. Incorporating discrete jumps in character state into the linear Bellman formulation in order to allow for exact constraint enforcement requires further study.

6.3 Coordination and Interpolation

Given a set of optimal controllers and their associated value functions, linear Bellman combination is optimal for a task with the following terminal cost function:

$$g(\boldsymbol{x}) = -\log\left(\sum_{i=1}^{n} w_i h_i(\boldsymbol{x})\right).$$

This allows us to create controllers which interpolate the terminal cost functions of the component controllers. It also allows us to create goal functions with multiple acceptable outcomes (multiple minima). The fixed blend weights w determine whether the combined controller is interpolating terminal cost functions or coordinating between multiple policies. These weights can either be set automatically given a new terminal cost function or manually. This section describes these weight settings and the behavior of the resulting control policy.

6.3.1 Control Interpolation for New Tasks

One application of linear Bellman combination is interpolating existing controllers. For example, this can be used to combine a controller for a long step with a controller for a short step to generate steps of intermediate length. Given a desired terminal cost function $\bar{g}$, we can automatically find the weights w_i such that g approximates $\bar{g}$ as well as possible. Working in reward space, this can be accomplished by projecting the function $\bar{h} = \exp(-g)$ onto the potentially non-orthonormal basis functions, h_i (see Figure 6-3). Non-orthonormal projections are carried out by solving a linear system $Aw = b$ where $A_{i,j} = g_i \cdot g_j$ and $b_i = g_i \cdot \bar{g}$. For low-dimensional systems,

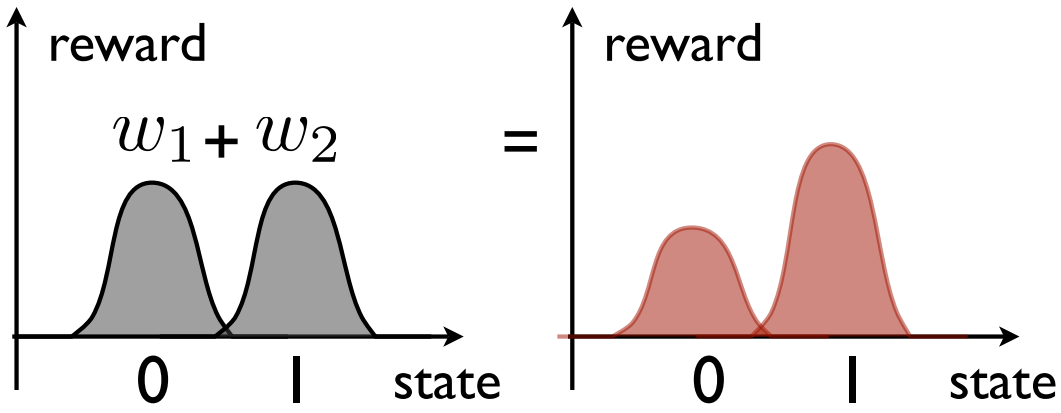

Figure 6-4: **Unintuitive Projection.** Depending on how example terminal cost functions are sampled, weighted combinations may produce unintuitive new terminal cost functions. For example, if the examples are too far apart, a weighted combination will produce a terminal reward function with two acceptable outcomes. Here a terminal cost function centered on zero and another centered on one are scaled such that weighted combinations produce local minima in the reconstructed terminal cost function.

the functions can be sampled at points in state space. For higher-dimensional systems, we restrict terminal cost functions to be quadratics (which are Gaussians after exponential transformation) and compute their inner product analytically.

The terminal cost functions of the component policies may form a poor basis for representing the desired terminal cost function, $\bar{g}$ as depicted in Figure 6-4. A good indicator is if the weights computed from projection are negative and do not come close to summing to one. The quadratic cost functions should share the same scale factor which is inversely proportional to the square of the distance between each minimum point. Otherwise, it may not be possible to smoothly interpolate points in state space as potential targets for the combined controller. Note that this scale factor also has an effect on the stiffness of the resulting control policy, as steeper cost functions lead to larger control actions. This can be mitigated by scaling control costs in the objective function.

6.3.2 Coordinating Multiple Controllers

Linear Bellman combination can also coordinate the operation of multiple controllers. Coordination is useful for two types of tasks as depicted in Figure 6-5. The first is a task where there is a set of acceptable outcomes. For example, in a subway car, you can balance by holding onto the wall of the car either in front of you or behind you. Coordination is also useful for tasks where controllers share a single desired outcome but are optimized for different parts of state space. As an example, consider continuing a walk after taking a longer or shorter step. In this case, the value function for each component controller would be the same if the component controllers were actually optimal, but this might not be the case due to approximations that are valid in some localized neighborhood of state space. Coordination will smoothly choose the controller with the best chance for success.

Coordination between component controllers is achieved by equally emphasizing the goal of each controller $w_i = 1$. If the assumptions for optimality are satisfied as described above, then the resulting controller $\boldsymbol{\pi}$ is optimal for an objective function with a terminal cost:

$$g(\boldsymbol{x}) = -\log\left(\sum_{i=1}^{n} \exp\big(-g_i(\boldsymbol{x})\big)\right).$$

First, we consider the case where all the terminal cost functions are equal, i.e. $g_i = g$ for all i. For high-dimensional systems, it is typically only possible to represent policies that cover a portion of state space. For example, we may have feedback controllers optimized to accomplish the swing up task from different initial conditions of a simple gymnast character. Combination can be used to arbitrate between each of these policies.

Next, consider the case where the g_i's are quadratics with minimums at different spots in state space:

$$g_i(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^\top \boldsymbol{Q}(\boldsymbol{x} - \bar{\boldsymbol{x}}_i).$$

Then, the combined terminal cost g has at most n minimum points. This means that

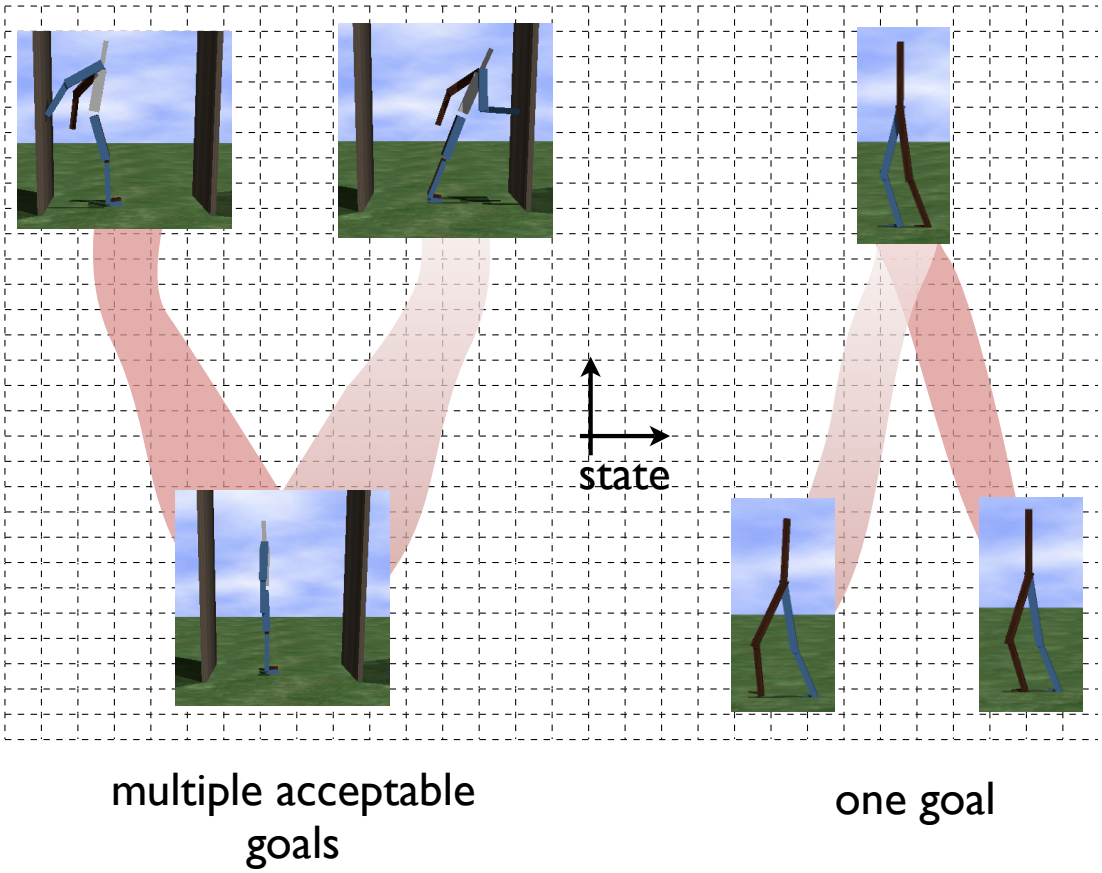

Figure 6-5: **Coordinating Multiple Controllers.** Linear Bellman controllers can track multiple trajectories at once, smoothly choosing whichever is appropriate at the given state and time. For example, we may have multiple controllers that start a step in different configurations but end in the same state. Or we may have controllers that start in the same state but end up in an acceptable outcome.

the combined policy will try to reach the goal of one of the policies depending on which ever is the most convenient given the current state of the character. As a concrete example, consider the case where a character is pushed forward or backwards and must grab onto something to maintain balance. In this case, the character should activate a controller that reaches forwards or backwards based on the push. The results section describes such an example.

6.4 Approximating the Value Function

Linear Bellman combination requires policies $\boldsymbol{\pi}_i$ and their associated value functions v_i. In practice, we often start with policies without value functions. To get a value function, we first specify an objective function as in Equation 2.4. Once the objective is identified, we can represent the value function for each controller in a manner that allows for the interpolation necessary in Equation 6.6. The details of the representation depend on the dimensionality of the system being animated.

6.4.1 Defining the Objective

Each component controller is assumed to be optimal with respect to an objective function of the form of Equation 2.4. Specifying an objective of this form requires making several choices if the objective function is not known. There is the scale of the control cost relative to the other terms, the form of the state penalty q and the form of each terminal cost function g_i.

The control penalty determines the stiffness of the policy. Decreasing the control penalty results in more control effort (i.e. stiffer feedback control) but gets the character closer to the terminal condition. Overly stiff control has an adverse effect on simulation stability and can lead to undesirable reactions to disturbances. However, placing a large cost on control effort will compromise the accuracy of the controller at achieving the final goal state.

The position penalty $q(\boldsymbol{x})$ is not used for some tasks, i.e. it is simply set to zero. For other tasks, excluding the position cost leads to undesirable feedback strategies that stray far from the example motion. Since the position cost must be consistent for each controller, we set q to be a quadratic penalty centered on some average trajectory or pose. This approach is similar to many other position penalties used in trajectory optimization approaches to character animation [63, 68, 74].

The terminal cost term is free to vary for each controller. We use quadratic functions of the form

$$g_i(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^\top \boldsymbol{Q} (\boldsymbol{x} - \bar{\boldsymbol{x}}_i),$$

where $\bar{\boldsymbol{x}}_i$ is the state of the system at the end of the ith controller's execution. As described in the previous section, restricting terminal cost functions in this way is important for determining what the composite controller is trying to accomplish and allows us to take inner products analytically. The scale of the terminal cost function $\boldsymbol{Q}$ affects the accuracy of the control and determines how closely examples must be sampled in order to allow for smooth interpolation.

While we specify the objective function manually, there exist algorithms for automatically determining objective functions given example trajectories [53, 1]. A generalization of this process to multiple trajectories would be a nice complement to other applications of linear Bellman combination.

6.4.2 Sampling the Value Function

Once an objective function is defined for each controller, the construction of each v_i depends on the dimensionality of the problem. There are two cases: low-dimensional systems of dimension less than five, and high-dimensional systems with tens of state variables. We discuss each case in turn.

Low-Dimensional Systems. For low-dimensional systems, we discretize the problem by constructing a regular grid over the area of interest in state space. Then, dynamic programming [19] solves for the optimal policy and value function at each point in this grid. At run-time, the value function is evaluated in the grid using barycentric interpolation of the values at grid points[75]. Low-dimensional optimal control has been used in character animation [54, 83].

High-Dimensional Systems. For high-dimensional systems, memory costs prevent us from building a grid in state space due to exponential growth of storage requirements. Instead, we generate an example trajectory using an execution of the controller from an initial condition. We then optimize this trajectory using an adjoint method [56] or differential dynamic programming (DDP) [41]. The output of the optimization is a state and control trajectory $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{u}}_t$ that define a value function

along the trajectory:

$$v_i(\bar{\boldsymbol{x}}_i, t) = g_i(\bar{\boldsymbol{x}}_i, T) + \int_t^T q(\bar{\boldsymbol{x}}_i, s) + \frac{1}{2}\bar{\boldsymbol{u}}_i(s)^\top \bar{\boldsymbol{u}}_i(s)\,\mathrm{d}s.$$

These samples may not equal the globally optimal value function in general if the control actions are suboptimal.

Given the set of value function samples along the optimized trajectory, we linearize the dynamics about that trajectory and use differential dynamic programming (DDP) [41] to compute a quadratic approximation for each trajectory:

$$\hat{v}_i(\boldsymbol{x}, t) = v_i(\bar{\boldsymbol{x}}_i, t) + \boldsymbol{p}_i(t)^\top \Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^\top \boldsymbol{P}_i(t)\Delta\boldsymbol{x} \tag{6.7}$$

where the vector $\boldsymbol{p}_i(t)$ and matrix $\boldsymbol{P}_i(t)$ are computed by solving standard Ricatti equations [41], and $\Delta\boldsymbol{x} = \boldsymbol{x} - \bar{\boldsymbol{x}}_i$. This local approximation is valid within some neighborhood of the optimized trajectory. This is an efficient representation as there are large regions of the state space which do not represent realistic states for the character. We considered other interpolation schemes such as radial basis functions but the advantage of this approach is that it avoids setting or determining interpolation parameters.

The approximation in Equation 6.7 is used with the linear quadratic regulator $\boldsymbol{K}_i$ [41] to compute the combined controller as

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \sum_{i=1}^{n} \alpha_i(\boldsymbol{x}, t)(\bar{\boldsymbol{u}}_i(t) + \boldsymbol{K}_i(t)\Delta\boldsymbol{x}),$$

where

$$\alpha_i(\boldsymbol{x}, t) = \frac{w_i \exp\big(-\hat{v}_i(\boldsymbol{x}, t)\big)}{\sum_{j=1}^{n} w_j \exp\big(-\hat{v}_j(\boldsymbol{x}, t)\big)}.$$

The exponential remapping used to compute α maps large costs to small rewards which can lead to precision errors with the normalization step. Also, other local approximations to the policy can be used instead of a control tape. For example, the optimization that generates the value function samples may optimize the parameters

of a feedback policy rather than a feedforward control tape. These policies can be blended using Equation 6.6.

6.5 Results

There are two applications our control formulation supports: controller interpolation and coordination. In this section, we evaluate the performance of both applications in several different simulation scenarios. We also provide empirical evidence that more naive combination schemes can lead to undesirable animation outcomes.

6.5.1 A Low Dimensional Example

To provide intuition for how linear Bellman combination operates, we first demonstrate it on a low-dimensional, non-linear toy system. The dynamics of the system are described by

$$\dot{\boldsymbol{x}} = \sin(4\pi x) + \boldsymbol{u}.$$

One can think of this system as describing a particle with no momentum on a rolling landscape. The non-linear system is first-order, so the controls, $\boldsymbol{u}$, can directly set the velocity of the system, $\dot{\boldsymbol{x}}$.

The task in this case is to minimize the amount of control used and to arrive at a certain location at the end of the simulation. In other words, we want to find policies of the form

$$\boldsymbol{\pi} = \arg\min_{\boldsymbol{\pi}} \frac{1}{2}||\boldsymbol{x}_T - \bar{\boldsymbol{x}}_T||^2 + \int_0^T \frac{1}{2}||\boldsymbol{u}_t||^2 \, dt.$$

Since the system is one-dimensional, we can represent the value function directly using a grid over a region of state space and find the optimal policy using dynamic programming. Two solutions are shown in Figure 6-6. The first policy, $\boldsymbol{\pi}_1$, drives the particle towards $\bar{\boldsymbol{x}}_T = 1.35$. The second policy, $\boldsymbol{\pi}_2$, drives the particle towards $\bar{\boldsymbol{x}}_T = 0.35$. Finding these policies takes about half a minute in Matlab.

Once we have solved for optimal policies we can then use them to drive the particle to points in the interval $[0.35, 1.35]$. Given a new desired state in this interval,

we project a Gaussian centered on that state onto the Gaussians associated with the terminal cost functions for each example policy. This yields a new terminal cost function which, after exponential remapping, is a weighted combination of the example terminal cost functions, as shown in Figure 6-6.

The combined policy as computed by Equation 6.6 is approximately optimal for the new terminal cost function. Without having to solve a new optimization problem, we have combined our existing controllers to create a new policy that outperforms any other policy when measured according to the objective function. For the terminal cost function shown in Figure 6-6, $\bar{\boldsymbol{x}} = 1.2$, a simple policy which linearly blends the two example policies drives the particle to $\boldsymbol{x}_{simple,T} = 0.82$ and has a total cost of 70,076 when the particle starts at the origin. The combined policy drives the particle from the origin to $\boldsymbol{x}_{c,T} = 1.23$ and has a final cost of 63,574. Note that the optimal policy sacrifices a little accuracy in the final position of the particle to save on exerted control effort. The advantage of optimality can vary depending on the desired end goal. The costs for various desired states are given in Figure 6-7.

6.5.2 Control Interpolation for New Tasks

Linear Bellman combination can be used to interpolate control tasks. Here we demonstrate this application using several animation tasks: diving, stepping, and jumping. We also highlight some important factors that go into the construction of an effective combined policy.

Diving. Optimizing any diving motion using trajectory optimization can be a time-consuming task. We can use blending to produce multiple diving examples from just two optimized solutions. In this case, one solution does a reverse dive that under rotates ending up short of the desired vertical position while the other dive over rotates and goes past vertical. Blending the resulting feedback policies creates dives that under rotate more or less or hit vertical.

In Figure 6-8, we compare linear Bellman combination for interpolation with a naive approach that averages the output of the each component controller using fixed

weights. This approach fails to produce a vertical dive. The simple approach can be made to work by tuning a set of time-varying blend weights. Linear Bellman combination produces blend weights automatically given the desired terminal cost function.

An important aspect of linear Bellman combination is that it uses less control than standard proportional derivative control. For graphics, this is important not only for simulation stability, but also for aesthetics. An overly stiff controller does not react to disturbances. As an example, we added random forces to the diver. Using linear Bellman combination, one can see the effects of this added force and watch how the control compensates for it. With a standard proportional-derivative dive controller, the effects of the applied disturbances are not immediately visible, although the final outcome is altered. With PD control, disturbances are invisible even when the applied forces are scaled by a factor of four.

Stepping. It is important while walking to be able to avoid stepping on obstacles [24]. Linear Bellman combination can create a stepping controller which can take steps of various lengths. For this application, two example steps, a short step and a long step, are combined to interpolate the step lengths. A complete walk can be constructed by sequencing step controllers together.

Jumping. The height of a jump is determined by the speed of the character at the moment of takeoff. Once in the air, the character cannot adjust its momentum to fly higher or further. As such, we use linear Bellman combination to interpolate takeoff controllers to jump over obstacles in the scene or to grab onto objects at different heights. Linear Bellman combination interpolates desired takeoff conditions more accurately than a naive approach. To test this, we created a tracking controller that lands a jump for a given initial condition. We then used this initial condition as the target for the linear Bellman combination. The resulting jump lands successfully. A weighted average controller using the same weights fails to land as it falls outside of the region of competence for the tracking controller.

Objective Function. In each of these cases, we found it important to include a term in the integrated cost of the objective which penalized deviations from some form of average trajectory. In other words, each example minimizes an objective of the form

$$(\boldsymbol{x}(T) - \bar{\boldsymbol{x}}_i(T))^\top \boldsymbol{Q}_T (\boldsymbol{x}(T) - \bar{\boldsymbol{x}}_i(T)) + \int_0^T (\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t))^\top \boldsymbol{Q} (\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t)) + \frac{1}{2} \boldsymbol{u}(t)^\top \boldsymbol{R} \boldsymbol{u}(t) \, \mathrm{d}t.$$

If the integrated cost term is not included in the objective, the resulting controller will make significant departures from the example trajectories to try and meet the terminal goal. It may also employ feedback strategies that result in undesirable motions.

6.5.3 Coordinating Multiple Controllers

Some recent work has used linear quadratic regulators (LQR) to track a single example trajectory [27, 15]. However, these tracking controllers are often valid only near the example trajectory. If a perturbation takes the character outside of this neighborhood, the tracking controller will fail to recover. Our solution to this problem is to combine many tracking controllers that cover different parts of the state space.

Simple Gymnast Example. We first demonstrate this idea using a high bar gymnastics task. In this case, the gymnast has two links and can only exert torques at the hip (this model is known as the acrobot model in robotics). Its state space consists of two joint angles and two joint velocities. The goal is to build a controller that brings the gymnast to a hand-stand position on the high bar while minimizing exerted torque. We build such a controller by combining multiple tracking controllers using Equation 6.6 with fixed weight one to create a combined policy $\boldsymbol{\pi}$. The tracking controllers track example trajectories which solve the swing up task for different initial conditions. These solutions are synthesized using an adjoint method solving

an objective of the form

$$g(\boldsymbol{x}) + \int_0^T \frac{1}{2}\boldsymbol{u}(t)^\top \boldsymbol{u}(t)\,\mathrm{d}t,$$

resulting in two feedforward trajectories $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ which solve the swing up task from two different initial conditions.

Figure 6-9 compares several approaches to solving the swing up task for an initial condition which does *not* have an associated pre-computed policy, $\boldsymbol{x}(0) = (-105, -15, 0, 0)$ where the angles are in degrees. The first attempt uses only a single tracking controller which tracks a solution initialized at $\boldsymbol{x}(0) = (-90, 0, 0, 0)$. The second attempt also uses a single tracking controller but this one is initialized at $\boldsymbol{x}(0) = (-110, -20, 0, 0)$. The third approach simply sums the results of the two tracking controllers. None of these approaches come as close to the ideal handstand as a controller created using linear Bellman combination of the two tracking policies. The hand-stand position is an unstable equilibrium point of the character. It is important to come as close as possible so that a stabilizing controller can easily maintain that position.

Diving. We use a diving example to illustrate that combining multiple controllers can outperform using a single tracking controller. We construct a controller using linear Bellman combination of three components which enter the water vertically for three different initial angular velocities: 0.85, 1, and 1.15, radians per second. A tracking controller is constructed from the dive that starts with an initial angular velocity of 1 radian per second. We sampled a number of different initial angular velocities for the diver. The combined controller does a better job recovering to a vertical pose on entry than a single tracking controller, see Figure 6-10.

Push Recovery. The idea of coordination can also be used to build simple push recovery controllers. We created a controller which chooses to reach forwards or backwards to prevent a fall based on an initial push. In Figure 6-11, we plot the blend weight of a coordinating controller indicating which controller has the most influence at each point in time. In the simulation depicted, the arm is tugged back

and forth causing changes in the blend weight.

Stepping. Coordination was also used to combine two SIMBICON [94] walkers. One walker was tuned to walk on flat ground, while the other was tuned to walk on an incline. When placed on an incline of ten degrees, the flat ground walker does not fall, but fails to advance up the hill. We constructed a combined controller that walks on flat and hilly terrain. As the combined walker encounters a hill, the blend weights shift to weight the hill component more heavily.

6.5.4 Implementation Details

Simulation. The examples involving linked rigid bodies were simulated using our own minimal coordinates simulation engine. For the examples involving contacts, we used a penalty-based method with parameters as described in Yin et al's SIMBICON simulation [94].

Physical Models. The two-link gymnast is a point-mass model with unit masses and unit length limbs. The diver is a seven link, planar character with physical parameters as described by Sok et al [71]. A version of the diver with both sets of extremities was used for the broad jump examples that land. The step example uses the same 2D model as SIMBICON [94]. The vertical leap examples use the the same 3D model as described in Limit Cycle Control [48].

Optimization. Linked rigid body examples were optimized using one of two methods. One method was a conjugate gradient descent algorithm where the derivative of the objective function with respect to the control parameters is calculated using the adjoint method. The other method was differential dynamic programming. We found that differential dynamic programming consistently converged in fewer iterations but sometimes settled in a different local minimum than the adjoint method. The optimizations often needed a decent initial guess to converge to good solutions. The diving and jumping examples were initialized using PD controllers similar to those

described by Wooten et al [89]

It is important to note that, as opposed to low-dimensional systems, high-dimensional systems require local optimization methods that avoid the curse of dimensionality by finding the solution for a single initial condition. However, it is not guaranteed that the resulting policy is optimal. This violates the assumptions required for optimality as described in §6.2, so in practice, all high dimensional examples are using sub-optimal policies.

6.6 Discussion

This chapter introduces linear Bellman combination as a method for joining control policies to derive new control strategies. Combination can interpolate two policies to obtain a strategy that accomplishes a different goal. Or, it can coordinate two policies to obtain a strategy that combines different functionalities. For example, coordination can combine policies designed for two different preconditions (or outcomes) to derive a composite strategy valid for either precondition (or outcome).

The combined strategies are modular because they only blend the control forces of constituent policies. The individual building blocks could therefore be arbitrary control schemes. Furthermore, the combination amortizes the cost and the difficulty of generating individual control policies. Policies can be painstakingly crafted once and then repurposed many more times in various combinations. With some assumptions, the combination can even combine optimal policies to compute another *optimal* policy for a different terminal cost.

When many related optimal control problems must be solved, there is an advantage to using linear Bellman combination over direct non-linear optimization. Even on simple problems, direct optimization can be too costly to compute controllers in real-time applications. Linear Bellman combination computes a new controller essentially instantly. However, linear Bellman combination is not intended as a replacement for direct non-linear optimization in *offline* applications. An alternative for real-time applications is model predictive control. Model predictive control may

be preferable to combination when feasible as in autonomous control of aircraft [57] or character models over short time horizons [76, 25].

One difficulty in applying linear Bellman combination to high-dimensional characters is that value functions are required to add controllers together. Obtaining a value function for a low- dimensional system can be done using standard techniques and one can be certain that linear Bellman combination will yield near-optimal controllers. Getting value functions for high-dimensional systems is not straightforward for a couple of reasons. First, you may not know what objective function a controller is trying to optimize a priori. Second, there is no standard method for representing high-dimensional value functions. Currently, we use a local approximation of the value function around example trajectories from the controller and a manually specified objective function to construct an approximate value function. This approach has the advantage that it is correct for optimal controllers of linear dynamical systems. Unfortunately, the systems we wish to control are non-linear. An interesting area of future work would be to automatically infer value-function approximations given observations of previously successful trajectories. This would allow us to learn control strategies from observations of human performances recorded by motion capture. Another useful addition to this work would be a mathematical tool for quantifying the error due to the local approximation of the value function.

A major limitation of linear Bellman combination, as presented here, is that the component controllers must share the same duration. We would like to devise a strategy for indexing component controllers using non-time based indices in order to blend policies of different duration. One could use any monotonic feature of the motion to reindex time such as the angle between the ankle and hip in a walk. In addition, it should be possible to extend combination to first-exit policies where the policy terminates when a certain state region is first reached.

In this chapter, combination was computed assuming a shared integrated cost between all component controllers. This assumption is easily lifted, however. If the position cost $q(\boldsymbol{x})$ is expressed in a finite linear basis $q(\boldsymbol{x}) = \sum_{i=1}^{k} c_i b_i(\boldsymbol{x})$, then solutions for each basis function $b_i(\boldsymbol{x})$ can be pre-computed and combined as they

are when terminal cost functions are expressed in a finite linear basis. An interesting area of future work is to exploit this added flexibility to combine controllers that are optimized for specific tasks rather than combining controllers for generic tasks as we have done here.

Linear Bellman combination complements the trajectory based trackers described earlier. While the trajectory trackers allow one to adapt motion capture trajectories to new regions of state space, they will eventually fail far from the reference motion. Combination can smoothly switch between trajectory tracking controllers to cover larger regions of state space. In addition, we can use interpolation to track to new terminal states that were not in any reference motion.

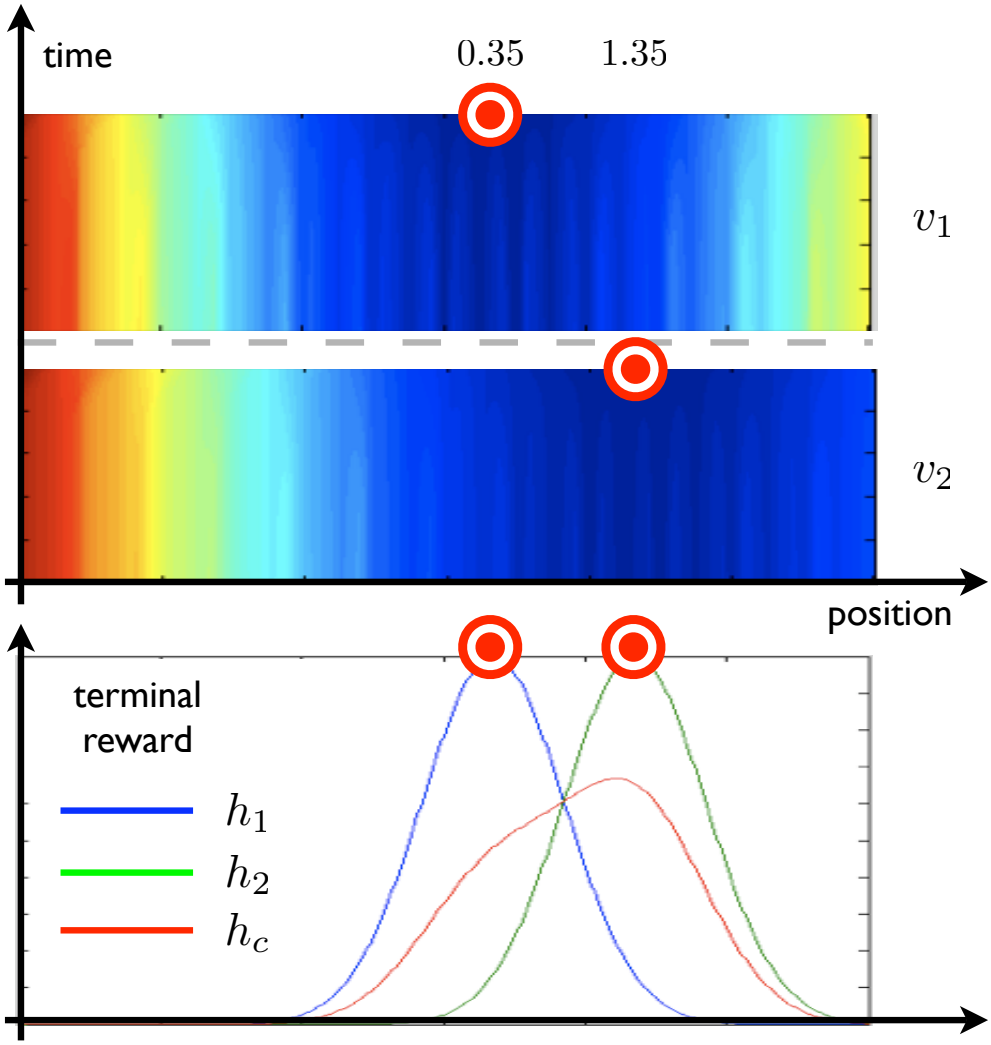

Figure 6-6: **Toy example- sine particle.** Shown on the top half of the figure are the time-varying optimal value functions for two control problems involving a one-dimensional particle with non-linear dynamics. The optimal policies descend the gradient of the value function to drive the particle to the indicated target. The plots on the bottom half are the exponentially remapped terminal cost functions. In blue is $h_1(\boldsymbol{x}) = z_1(\boldsymbol{x}, T) = \exp\big(-v_1(\boldsymbol{x}, T)\big)$, the remapped terminal cost function for a control problem where we want the particle to land at $\bar{\boldsymbol{x}}_T = 0.35$ at the final time. This corresponds to the top row of the top value function image. In green is the remapped terminal cost function for a control problem where we want the particle to end up at $\bar{\boldsymbol{x}}_T = 1.35$. This corresponds to the top row of the second value function image. In red is $h_c(\boldsymbol{x}) = w_1 h_1(\boldsymbol{x}) + w_2 h_2(\boldsymbol{x})$, the combined terminal cost function. The combined policy as described in Equation 6.6 solves this control problem without having to solve an expensive dynamic programming problem.

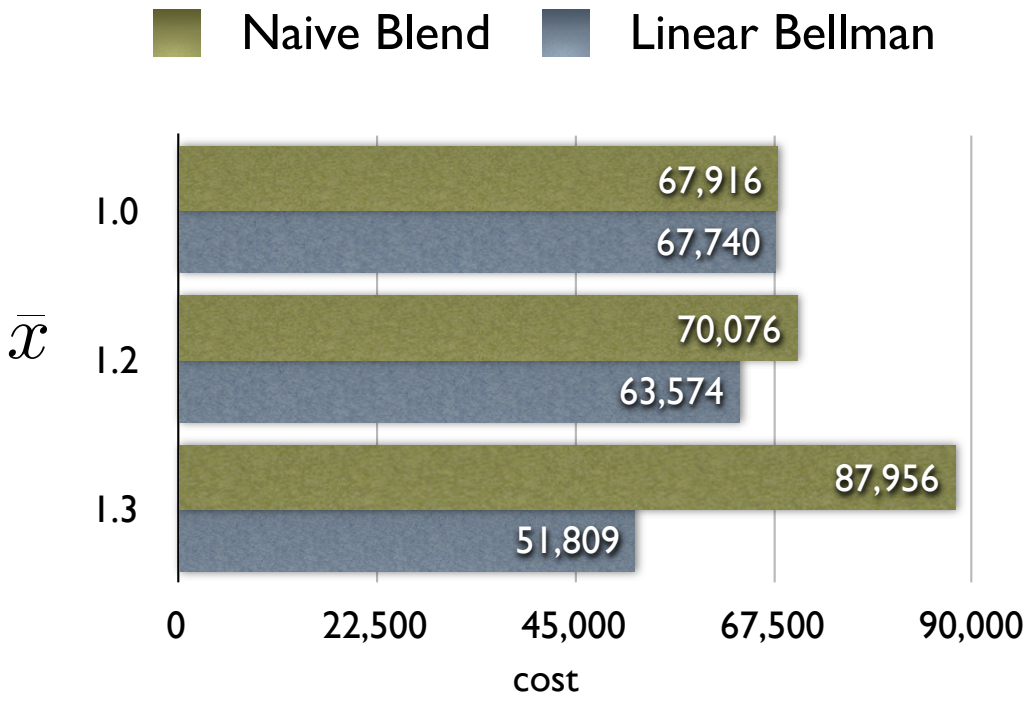

Figure 6-7: **Total cost comparison.** Shown are the total cost of policies that drive a 1D particle with sinusoidal dynamics to the desired state, $\bar{\boldsymbol{x}}$. Linear Bellman combination always give a lower cost although the size of the advantage depends on the desired state.

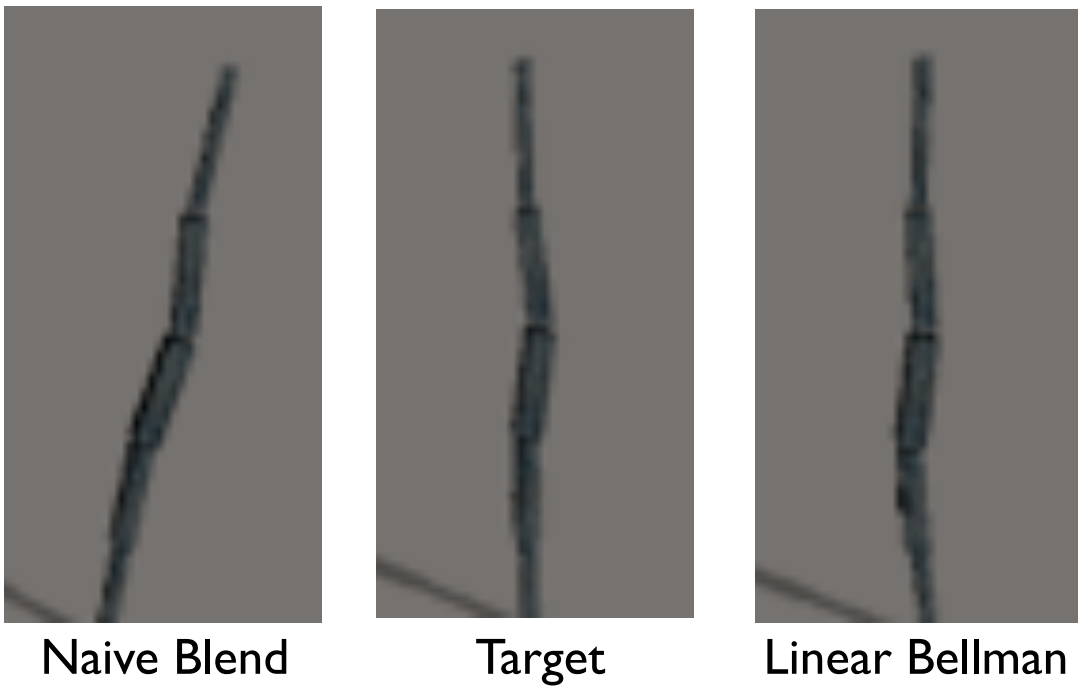

Figure 6-8: **Naive Blend vs Linear Bellman.** Shown are the final frames of two dive simulations. The simulation on the right used linear Bellman combination to compute time-varying weights to combine a dive controller that under rotates and a dive controller that over rotates. The fixed blend weights w_i needed to produce a vertical diver were automatically computed by projecting a goal function centered on the vertical pose onto the goal functions of each component controller. Using these fixed blend weights in a naive weighted average fails to produce a vertical dive.

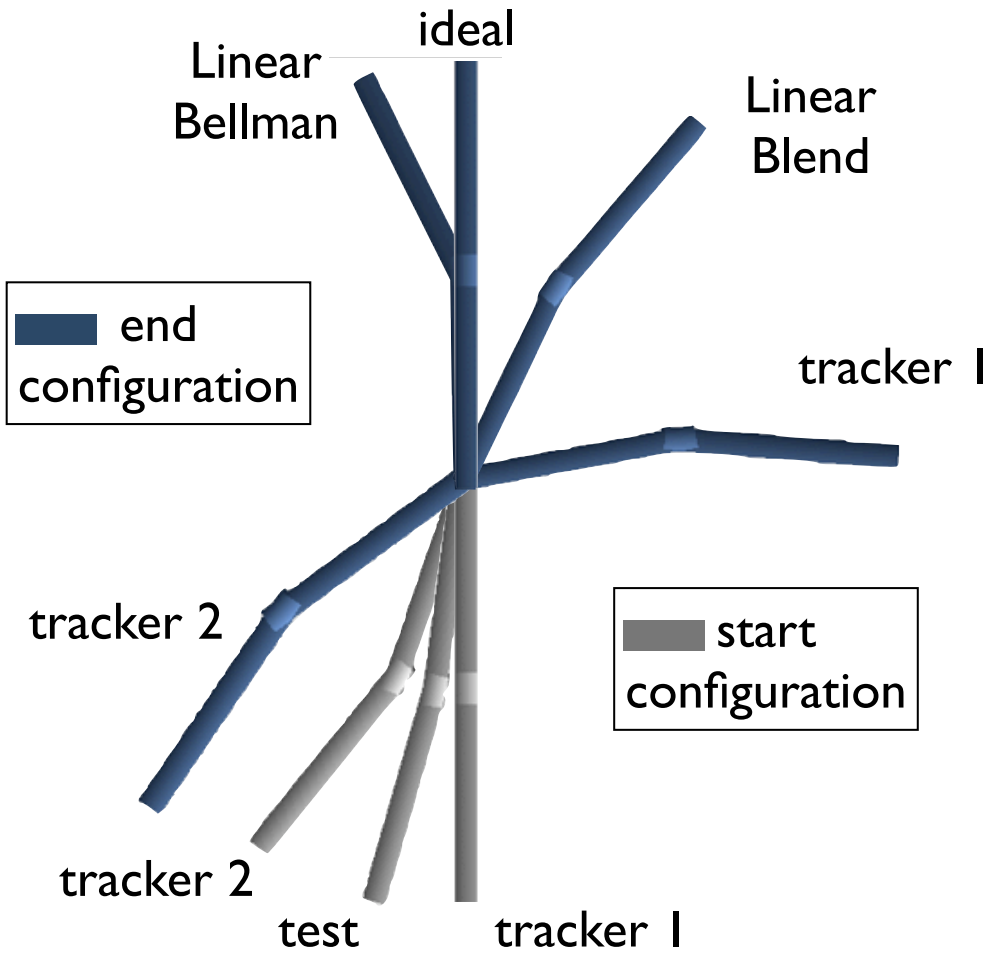

Figure 6-9: **Tracking Multiple Swing Up Trajectories.** Here we compare the final position of the simple gymnast under four policies. Tracker 1 and 2 are LQR policies used to track optimized trajectories for the swing up task starting from the indicated locations. Simulations starting from the test location using these trackers fail to come near the upright position. A simple linear blend fairs better but is not as close as a controller created using linear Bellman combination. The results of all these trackers can be improved by placing examples closer to the test configuration.

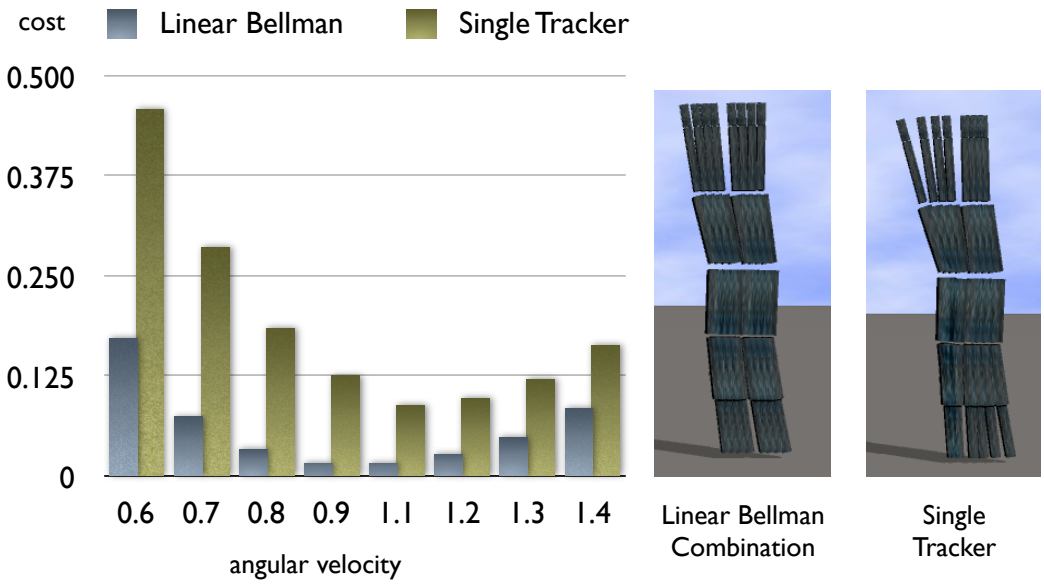

Figure 6-10: **Diving Coordination.** Shown are the final poses of a diver starting from eight different initial angular velocities. The poses on the left were controlled using linear Bellman combination of three dive controllers. The poses on the right were simulated using a single tracking controller. The single tracking controller does not reach a vertical configuration for some initial conditions and has an inferior objective score in each case.

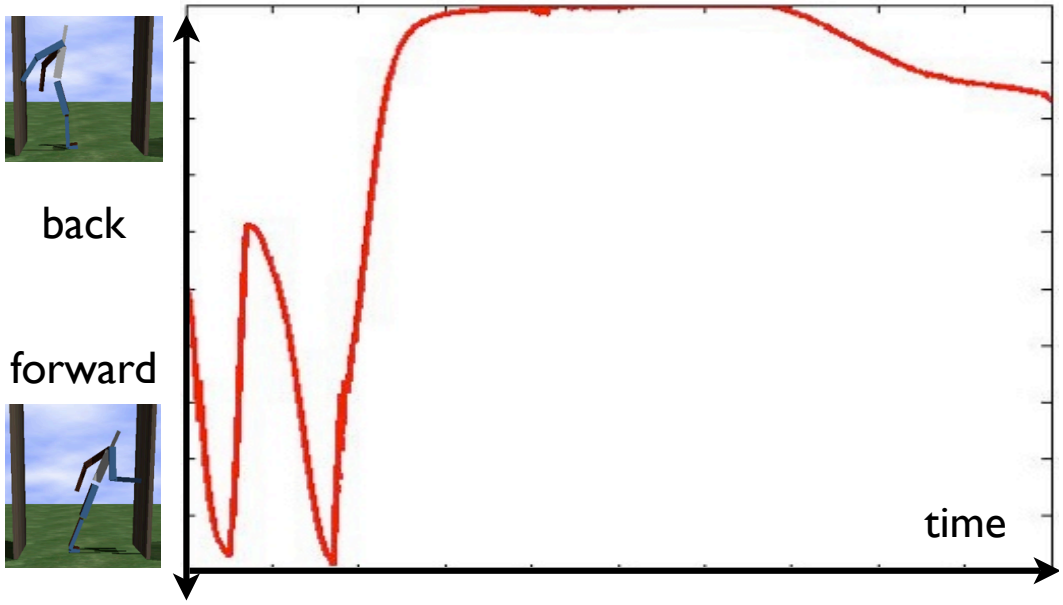

Figure 6-11: **Balance Coordination.** Shown is the time varying blend parameter of a coordinating controller created using linear Bellman combination of a controller which catches itself falling forwards and another falling backwards. In this simulation, the arm of the character was tugged forwards and backwards causing the combined controller to vacillate before settling on the backwards strategy.

Chapter 7

Conclusions and Open Problems

This work has presented methods for controlling characters in physical simulations. These control methods allow even a novice animator to easily produce realistic character behaviors that would be difficult for expert animators to create. More importantly, even the most expert animator cannot possibly account for every possible scenario a character will encounter in an interactive simulation. The control methods in this work can adapt the animator's input to new scenarios, reducing the burden on the animator while producing more convincing results.

The key to controlling characters in interactive simulations is making effective use of pre-computation. While moving around the world may seem natural, controlling non-linear, high-dimensional, and under-actuated characters is challenging. This challenge is even harder in animation because the animator wants to dictate the style of the character's movement. Optimal control has been shown to be an effective tool for producing stylized animations in *offline* applications. In this work, we have shown how optimal control methods can be applied to online control problems by using pre-computation.

The Hamilton-Jacobi-Bellman equation, the starting point for each of the methods in this work, provides a systematic method for computing optimal controllers, but is typically intractable to solve directly. Depending on the assumptions we make, we can derive tractable optimal control formulations suitable for pre-computation. For example, by assuming we are close enough to a nominal trajectory, we can use a linear

approximation to the character's dynamics. This, coupled with quadratic tracking penalties, allows us to pre-compute tracking policies. We also saw that a slightly more general set of assumptions leads to a linear Bellman equation. This let us compute optimal control solutions efficiently using linear PDE solvers. Furthermore, linearity allows you to combine pre-computed solutions together at nearly no cost.

7.1 Optimality

While solving the HJB equation can give you optimal controllers, in practice many challenges conspire to limit your ability to guarantee optimality. The main challenge is the high-dimensionality of characters, but there are other challenges as well. For example, character dynamics are sensitive to changes in contact configuration. To deal with this, we described how to use quadratic programming to adapt LQR policies to the current contact configuration. While we observed this approach to be much more robust that applying LQR forces directly, it's hard to analyze the robustness of this non-linear control scheme.

The main road-block to using the HJB equation to derive controllers for character behaviors is the curse of dimensionality. Our approach has been to approximate the value function locally along example trajectories. It can be very difficult to get optimal example trajectories or even good example trajectories. Another drawback to using local approximations is that we do not, as of yet, have a good measure of how good those approximations are away from the trajectory. There are discretization errors for low-dimensional problems as well, but at least there is a well-defined relationship between grid resolution and solution accuracy. For example, when solving the linear Bellman equation on a grid, we can predict the order of the errors in the solution given the particular integration scheme we are using. There is no equivalent tool for trajectory based representations of the value function. Such a tool would be immensely useful as it would tell us where new samples of the value function are needed.

7.2 Tracking Motion Capture

Tracking a recorded motion trajectory is a good strategy for getting realistic looking motions. However, a single motion does not tell you much about why a particular motion looks the way it does. This makes it hard to deviate significantly from that motion. Even using combination, you must capture many motions to handle a large number of tasks very accurately. A better approach would be to look at one or more motion trajectories and learn some physical principles that lead to this type of motion. Such an approach could potentially generalize better than a tracking based approach and has been proven to work well for offline animation [53]. Furthermore, while analyzing data from a tracking simulation could be useful, ultimately a more fundamental approach is needed to explain why animals move the way they do.

7.3 Beyond Animation

The focus of this dissertation has been to apply control to animating characters in interactive simulations. Ultimately, it is unclear whether animators, noted control freaks, will be comfortable with the possibility of the character getting into an infinite number of states. However, there are a number of other potential applications for this work. In particular, the methods described could be adapted to controlling real robots. One difficulty will be the discrepancy between the controller's model of physics and the actual physics governing the motion of the robot. In particular, it would be interesting to combine the methods described here with a systems identification approach. Beyond robotics, our tools can be used in biomechanics to generate data for analysis but also to test different hypotheses of motion.

7.4 Closing Remarks

Physically-based animation already allows artists to create movies that would otherwise be impossible to animate using manual tools. Coupling control with physical simulation will extend this benefit to characters in movies and games, help us design

real-world autonomous characters, and answer questions about how real creatures move. This dissertation has argued that an effective way to use control is to start from an idealized optimal control perspective. At first glance, such an approach may seem intractable. However, by applying domain appropriate assumptions, we can often arrive at a tractable and practical solution.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine learning (ICML)*, volume 69, page 1. ACM, 2004.

[2] A. Abdallah, M.; Goswami. A biomechanically motivated two-phase strategy for biped upright balance control. In *International Conference on Robotics and Automation (ICRA)*, pages 1996–2001, 2005.

[3] Yeuhi Abe, Marco da Silva, and Jovan Popović. Multiobjective control with frictional contacts. In *Symposium on Computer Animation (SCA)*, pages 249–258, 2007.

[4] Yeuhi Abe and Jovan Popović. Interactive animation of dynamic manipulation. In *Symposium on Computer Animation (SCA)*, 2006.

[5] M. Anitescu, F. Potra, and D. Stewart. Time-stepping for threedimensional rigid body dynamics, 1998.

[6] M. Anitescu and F.A. Potra. A time-stepping method for stiff multibody dynamics with contact and friction. *International J. Numer. Methods Engineering*, 55(7):753–784, 2002.

[7] Okan Arikan and David A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, 2002.

[8] Christopher G. Atkeson. Using local trajectory optimizers to speed up global optimization in dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, volume 6, pages 663–670, 1994.

[9] Christopher G. Atkeson and Jun Morimoto. Nonparametric representation of policies and value functions: A trajectory-based approach. In *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 1611–1618. Cambridge, MA, 2002.

[10] Christopher G. Atkeson and Jun Morimoto. Nonparametric representation of policies and value functions: A trajectory-based approach. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1611–1618. MIT Press, Cambridge, MA, 2003.

[11] D.J. Balkcom, P.A. Kavathekar, and M.T. Mason. Time-optimal trajectories for an omni-directional vehicle. *Int. Jrnl. of Robotics Research*, 25(10):985–999, 2006.

[12] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics (Proceedings of SIGGRAPH 94)*, Annual Conference Series, pages 23–34. ACM SIGGRAPH, 1994.

[13] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1998. ACM.

[14] Jernej Barbič. *Real-time Reduced Large-Deformation Models and Distributed Contact for Computer Graphics and Haptics*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2007.

[15] Jernej Barbič and Jovan Popović. Real-time control of physically based simulations using gentle forces. *ACM Transactions on Graphics*, 27(5):163:1–163:10, 2008.

[16] Jernej Barbič, Marco da Silva, and Jovan Popović. Deformable object animation using reduced optimal control. *ACM Trans. on Graphics (SIGGRAPH 2009)*, 28(3), 2009.

[17] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[18] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. Tracks: toward directable thin shells. *ACM Transactions on Graphics*, 26(3):50:1–50:10, 2007.

[19] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3 edition, 2007.

[20] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, Philadelphia, PA, 2001.

[21] Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, pages 309–315, 1988.

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

[23] K. Byl and R. Tedrake. Approximate optimal control of the compass gait on rough terrain. *International Conference on Robotics and Automation (ICRA)*, pages 1258–1263, 2008.

[24] Stelian Coros, Philippe Beaudoin, KangKang Yin, and Michiel van de Panne. Synthesis of constrained walking skills. *ACM Transactions on Graphics*, 27(5):113:1–113:9, 2008.

[25] M. da Silva, Y. Abe, and J. Popović. Simulation of human motion data using short-horizon model-predictive control. *Computer Graphics Forum*, 27(2), 2008. In press.

[26] Marco da Silva. Interactive simulation of stylized human locomotion. http://hdl.handle.net/1721.1/42003, 2008.

[27] Marco da Silva, Yeuhi Abe, and Jovan Popović. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics*, 27(3):82:1–82:10, 2008.

[28] T. Erez and W.D. Smart. Bipedal walking on rough terrain using manifold control. *International Conference on Intelligent Robots and Systems (IROS)*, pages 1539–1544, 2007.

[29] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, pages 251–260, 2001.

[30] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22(3):417–426, 2003.

[31] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Transactions on Graphics*, 23(3):441–448, 2004.

[32] R. Featherstone and D. E. Orin. Robot dynamics: Equations and algorithms. In *International Conference on Robotics and Automation (ICRA)*, pages 826–834, 2000.

[33] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, New York, NY, 1998.

[34] Wendell H. Fleming. Exit probabilities and optimal stochastic control. *Applied Mathematics and Optimization*, 4:329–346, 1978.

[35] Nick Foster and Dimitri Metaxas. Practical animation of liquids. In *Graphical Models and Image Processing*, pages 23–30, 1996.

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

[37] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for SQOPT 5.3: A fortran package for large-scale linear and quadratic programming. Technical Report NA 97–4, University of California, San Diego, 1997.

[38] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 153–162, 1997.

[39] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 71–78, 1995.

[40] C. Holland. A new energy characterization of the smallest eigenvalue fo the schrödinger equation. *Communications on Pure and Applied Mathematics*, 30:755–765, 1977.

[41] D.H. Jacobson and D.Q. Mayne. *Differential Dynamic Programming*. Elsevier, New York, 1st edition, 1970.

[42] 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. 2:1644–1650, 2003.

[43] R. E. Kalman. Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana*, 5:102–119, 1960.

[44] Hilbert J. Kappen. Linear theory for control of nonlinear stochastic systems. *Physical Review Letters*, 95(20):200–204, 2005.

[45] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics (SIGGRAPH Asia 2008)*, 27(5):164:1–164:11, 2008.

[46] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, 2002.

[47] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 35–44, 1988.

[48] Joseph F. Laszlo, Michiel van de Panne, and Eugene L. Fiume. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of SIGGRAPH 96*, Annual Conference Series, pages 155–162, 1996.

[49] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, Annual Conference Series, pages 39–48. ACM SIGGRAPH, 1999.

[50] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations Steady State and Time Dependent Problems*. SIAM, Philadelphia, PA, 2007.

[51] F.L. Lewis, C.T. Abdallah, and D. M. Dawson. *Control of Robot Manipulators*. Macmillan, New York, 1st edition, 1993.

[52] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*, pages 222–229, 2004.

[53] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics*, 24(3):1071–1081, 2005.

[54] James McCann and Nancy Pollard. Responsive characters from motion fragments. *ACM Transactions on Graphics*, 26(3):6:1–6:7, 2007.

[55] R.B. McGhee. Vehicular legged locomotion. In G.N. Saridis, editor, *Advances in Automation and Robotics*. JAI Press, 1983.

[56] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23(3):449–456, 2004.

[57] Mark B. Milam. *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems*. PhD thesis, Caltech, 2003.

[58] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California Berkeley, Berkeley, CA, 1996.

[59] H. Miura and I. Shimoyama. Dynamic walk of a biped. *International Journal of Robotics Research*, 3(2):60–74, 1984.

[60] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, pages 343–350, 1993.

[61] Nancy S. Pollard and Fareed Behmaram-Mosavat. Force-based motion editing for locomotion tasks. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 663–669, 2000.

[62] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*, Annual Conference Series, pages 209–218. ACM SIGGRAPH, 2000.

[63] Zoran Popović and Andrew P. Witkin. Physically based motion transformation. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, 1999.

[64] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *International Conference on Humanoid Robots*, pages 200–207, 2006.

[65] Marc H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, MA, 1986.

[66] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, Annual Conference Series, pages 349–358. ACM SIGGRAPH, 1991.

[67] Hannes Risken. *The Fokker-Planck equation: methods of solution and applications*. Springer, New York, NY, 2nd edition, 1989.

[68] Alla Safonova, Jessica Hodgins, and Nancy Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics*, 23(3):514–521, 2004.

[69] Ari Shapiro, Petros Faloutsos, and Victor Ng-Thow-Hing. Dynamic animation and control environment. In *Proceedings of Graphics Interface (GI)*, pages 61–70, 2005.

[70] Daniel Sharon and Michiel van de Panne. Synthesis of controllers for sylized planar bipedal walking. In *International Conference on Robotics and Automation (ICRA)*, pages 2387–2392, 2005.

[71] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. *ACM Transactions on Graphics*, 26(3):107:1–107:9, 2007.

[72] Robert F. Stengel. *Optimal Control and Estimation*. Dover Books on Advanced Mathematics, New York, NY, 1994.

[73] D.E. Stewart and J.C. Trinkle. Dynamics, friction, and complementarity problems. In M.C. Ferris and J.S. Pang, editors, *Complementarity and Variational Problems*, pages 425–439. SIAM, 1997.

[74] Adnan Sulejmanpasić and Jovan Popović. Adaptation of performed ballistic motion. *ACM Transactions on Graphics*, 24(1):165–179, 2005.

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

[76] Yuval Tassa, Tom Erez, and William Smart. Receding horizon differential dynamic programming. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1465–1472. MIT Press, Cambridge, MA, 2008.

[77] Russell L Tedrake. *Applied Optimal Control for Dynamically Stable Legged Locomotion*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[78] E. Todorov. *In Bayesian Brain: Probabilistic Approaches to Neural Coding, Doya K (ed)*. MIT Press, 2006.

[79] E. Todorov. Linearly-solvable markov decision problems. *Advances in Neural Information Processing Systems*, 19:1369–1376, 2006.

[80] Emanuel Todorov. A new mathematical framework for optimal choice of actions. http://www.cogsci.ucsd.edu/∼todorov/papers/framework.pdf. Unpublished manuscript, 2008.

[81] Emanuel Todorov. Compositionality of optimal control laws. http://www.cogsci.ucsd.edu/∼todorov/papers/primitives.pdf. Unpublished manuscript, January 15 2009.

[82] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. SIAM, Philadelphia, PA, 2000.

[83] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Transactions on Graphics*, 26(3):7:1–7:7, 2007.

[84] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Synthesizing parameterized motions. 5th Eurographics Workshop on Simulation and Animation, 1994.

[85] P. B. Wieber. On the stability of walking systems. In *International Workshop on Humanoid and Human Friendly Robotics*, pages 1–7, 2002.

[86] K. D. Willmert. Occupant model for human motion. In *SIGGRAPH '74: Proceedings of the 1st annual conference on Computer graphics and interactive techniques*, pages 46–46, New York, NY, USA, 1974. ACM.

[87] David A. Winter. *Biomechanics and Motor Control of Human Movement*. John Wiley and Sons, Inc., New York, 2nd edition, 1990.

[88] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 159–168, 1988.

[89] W. L. Wooten and J. K. Hodgins. Simulating leaping, tumbling, landing and balancing humans. *International Conference on Robotics and Automation (ICRA)*, pages 656–662, 2000.

[90] K. Yamane and Y. Nakamura. Dynamics filter - concept and implementation of online motion generator for human figures. *Transactions on Robotics and Automation*, 19(3):421–432, 2003.

[91] K. Yamane and Y. Nakamura. Stable penalty-based model of frictional contacts. pages 1904 –1909, may 2006.

[92] K. Yin, M. Cline, and D. K. Pai. Motion perturbation based on simple neuromotor control models. In *Pacific Conference on Computer Graphics and Applications (PG)*, pages 445–449, 2003.

[93] KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Continuation methods for adapting simulated skills. *ACM Transactions on Graphics*, 27(3):81:1–81:7, 2008.

[94] Kangkang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON: Simple biped locomotion control. *ACM Transactions on Graphics*, 26(3):105:1–105:10, 2007.

[95] Victor B. Zordan and Jessica K. Hodgins. Tracking and modifying upper-body human motion data with dynamic simulation. In *Computer Animation and Simulation*, 1999.

[96] Victor B. Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Symposium on Computer Animation (SCA)*, pages 89–96, 2002.