Synthesis of Stylized Walking Controllers for Planar Bipeds

by

Dana Sharon

B.Sc., University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

August 2004

Abstract

We present a new method for generating controllers for physics-based simulations of planar bipedal walking, targeted towards producing autonomous character behaviors for computer animation. A common criticism of physics-based character animation is the lack of personality and style in the final motion. We develop controllers that mimic a desired style as much as possible, while still subject to the laws of physics and to the realities of maintaining balance. The resulting simulated character can interact with and respond to its environment, unlike the original kinematically-specified desired motion.

Walking is a very challenging control task because of its dynamically unstable nature. Continuous balance control is required to prevent the character from falling over or reaching poses from which it cannot recover. The complexity of a walking controller is compounded by high dimensional continuous state and action spaces. Our controller implements a nearest-neighbor control policy with respect to a set of nodes embedded in the state space, and is parameterized by the locations of these nodes and their associated actions. We use greedy search to find an optimized controller given an intuitive objective function. Importantly, several shaping techniques are used to ensure that local minima in the optimization space define good, meaningful solutions.

We demonstrate the ability to generate stable walks of various styles, walks for bipeds of varying dimensions, and walks that are robust to unobserved terrain variations.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I thank my supervisor, Michiel van de Panne, for his guidance, enthusiasm, and financial support. Michiel's encouragement and unbelievable patience turned this roller coaster ride into a wonderful adventure. I am also grateful to my second reader, Robert Bridson, for his valuable feedback.

I thank my friends Leah Findlater, Matt Williams, and Karyn Moffatt for making it a joy to come to work. Special thanks to my partner, Nando de Freitas, for his continuous love and support.

This work was funded by NSERC and IRIS.

Dana Sharon

The University of British Columbia
August 2004

To my family: Michael, Miriam, Ideet, and David.

Chapter 1

Introduction

Computer graphics systems are becoming increasingly sophisticated, and are using ever-increasing levels of abstraction to model real and imaginary worlds. Instead of creating character motions by hand, we may wish to develop autonomous characters that are equipped with many basic skills, the most basic of which is walking. Developing something as seemingly simple as a physically-correct walking skill is, however, a daunting task. In this thesis we present novel methods for designing controllers that can walk with a particular style, as well as using a novel controller representation.

We begin by discussing the motivation behind bipedal locomotion research in Section 1.1. Section 1.2 details some of the challenges that researchers face when studying bipedal motion. In Section 1.3 we outline some of the goals we wish to achieve. We conclude this chapter in Section 1.4 with an outline of the thesis structure.

1.1 Motivation

Locomotion is a problem of interest to many fields of research, including robotics, animation, and biomechanics. While our particular interest is biased towards computer animation applications, the diverse range of fields with an interest in modeling locomotion, specifically walking gaits, is indicative of the importance of the problem.

Robotics

Anthropomorphic robots, or "humanoid robots" as they are sometimes called, have long been a fascination of mankind, first as captured by many a science fiction story, and more recently as embodied by a string of engineering marvels such as the Honda Asimo and the Sony QRIO robots. Because of the long-term challenges of building humanoid robots with even moderate capabilities, entertainment robotics has become the immediate driver of many current robotic efforts.

Roboticists can create other types of robots, such as wheeled robots or robots with more than two legs. These robots are more stable than bipeds but this stability comes at the cost of dexterity and flexibility. Bipeds are more versatile and can walk in the same environment as humans. Moreover, they can perform tasks that other robots cannot, such as stepping over large obstacles and walking up and down stairs.

Computer Animation

Building intelligent, autonomous characters for use in computer graphics is not all that different from building capable robots. It is a simpler problem in that it lets us easily experiment with the control problems without having to deal with the complexities of hardware. Instead, the biped remains a computer simulation and the goal might be to create a believable character for an animated movie or computer game. Intelligent, reactive motion is a key part of what makes a character believable and "alive".

Biomechanics

Bipedal locomotion models are of interest in biomechanics because they provide an understanding of human motion. This understanding can then help in developing prosthetics for people or might help in understanding how hip fractures from falls in the elderly occur and thus provide insights into how to reduce such injuries. Studying locomotion can also provide insight into the workings of the brain. While progress continues to be made, human motor control strategies continue to be poorly

understood. Working with simulations of a complex bipedal model provides a test-bed for newly proposed models of motor control.

Kinesiology

High fidelity models of human motion have applications in athletic training and injury rehabilitation. Computer simulations can help predict the feasibility of particular motions as well as their strength and timing requirements. Dance choreography is yet another application.

Engineering Evaluation

The control of biped locomotion also has applications in engineering and design, most notably for footwear and portable equipment [10]. Simulations can help answer questions such as "How effectively can a person walk or run uphill while wearing a backpack with this weight distribution?" While it might seem simpler to test this in person, a good model can make predictions for a wide variety of body shapes and sizes.

1.2 Challenges in Bipedal Walking Research

Bipedal walking control is a challenging research area. Given how effortlessly most of us walk, it is both frustrating and fascinating to discover how difficult it is to duplicate this effortless action. Walking is a dynamically unstable motion in which the character is first falling and then catching itself from falling with the next foot placement. The character must continuously control its balance in this fashion.

Walking systems are high-dimensional non-linear systems, and as such they are challenging to control. The literature on the control of walking is distributed across multiple fields, including computer animation, robotics, kinesiology and biomechanics. Coalescing information from these various disciplines is challenging because each discipline brings with it its own motivations and constraints, and therefore results in significantly different techniques. For example, in robotics, researchers must

ensure that their controllers work on actual physical robots. Torque limits at the joints are but one constraint that this introduces. Physical robots are expensive and thus the process of learning to walk cannot involve the robot falling down all the time. A recent trend is to build smaller robots that are robust to falls. Also, physical robots may suffer from imperfect knowledge about the state and any number of difficult-to-model effects such as joint friction and gear backlash. Robot walking controllers tend to produce slower, more artificial looking walks that ensure stability.

In computer animation the challenges lie elsewhere. Here, it might be important to produce a variety of natural looking motions, as the character will be walking in computer games or movies whose goal is to entertain people. In animated movies, the character needs to move in a believable way in order to maintain the required "suspension of disbelief". Creating a realistic character motion is an extremely challenging and time consuming task that requires a great deal of talent and skill even for kinematically defined motions. For dynamically simulated motions the challenge is even greater, although the solution is ultimately more general.

Repeatability of results is an issue in bipedal locomotion research. Most projects construct their own biped model with unique mass distributions, joint torque limits and so forth. It is unfortunate that the various techniques are rarely compared to one-another and their success often relies on manual tuning of various parameters. Without a proper comparison or benchmark, it is often hard to evaluate previous work.

Another problem that plagues this research area is the large number of parameters that are involved, including ground model parameters, muscle stiffness, and mass distribution of the character. Often these variables interact in unforseen ways and thus if even one parameter is wrong, a controller may not produce a successful walking motion.

Generalization and parameterizations with respect to different environments is another key challenge. Ideally, a walking controller, or set of controllers, would be able to handle any reasonable environment including obstacles, variable terrain and steps of various sizes. One cannot provide the controller rules for all possible situa-

tions it might encounter, given that the environment can have an infinite number of instantiations. Generalizing and classifying possible combinations of character and environment state is a challenging problem.

Many problems are specific to the approach used to generate controllers. For example, if machine learning or optimization techniques are used, one needs to evaluate the performance of controllers. It is not obvious what good performance criteria for walking should be. Criteria might include energy minimization, robustness, or similarity to a specific walk.

1.3 Research Goals

The general problem of walking control remains unsolved in many ways. The major goals for our work are as follows:

1. To develop controllers that apply to a variety of characters. Our controller should be able to take as input an arbitrary character model, including number of body parts, lengths, masses, and inertia information for each part, and control constraints such as joint strengths and joint limits.

2. To develop controllers that are robust to perturbations in the character state and can generalize to a variety of environments.

3. To develop controllers that mimic a specified desired style. The style may be physically infeasible in part or in whole. The controller should mimic the desired motion and at the same time maintain balance.

1.4 Thesis Organization

The remainder of this thesis is structured as follows. In Chapter 2, we present a selection of previous work relating to bipedal locomotion. We also describe related work on imitation-based learning. Chapter 3 documents our dynamic simulation system and gives the detailed specification of all our character models. Chapter 4 describes our controller representation and our method for optimizing the controller

to produce a stable walk of a desired style. We present several shaping techniques that are important for producing a tractable optimization problem. We present the results of our experiments in Chapter 5. We demonstrate controllers for distinct walking styles, for distinct characters, and for terrains of varying slopes. We conclude in Chapter 6 with a summary of contributions, limitations and directions for future work.

Chapter 2

Previous Work

Bipedal locomotion has been a challenging and popular research topic for decades. The published literature consists of many hundreds, if not thousands of papers. Interest has come from many different fields, such as computer animation, robotics and biomechanics. However, as discussed in the previous chapter, each field has its own specific motivations and goals. For example, the goals of entertainment robotics may clearly differ in significant ways from those of biomechanics. We begin by presenting an overview of kinematic and dynamic techniques tailored to computer animation in Section 2.1. We then provide an overview of control techniques drawn largely from the robotics literature in Section 2.2.

2.1 Animation-Specific Methods

Animation does not have to solve the real control problem. It can rely on kinematic techniques. The price to be paid is that the models of motion are not as general, and we do not necessarily learn much about real motion control in humans or robots.

2.1.1 Keyframing

Keyframing is at the heart of almost any commercial animation system. Motions are defined by a set of key poses or *keyframes* of a character at specific instants in time. The complete character motion is then computed by interpolating between

the keyframes, most commonly using splines. Keyframing grants the animator full control over the motion, both in terms of style and timing. However, it demands patience and skill on the part of the animator to incorporate realistic dynamics into the motion. The product of keyframing is a motion trajectory over time. This trajectory is not reusable in that the resulting motion is completely specific to the given situation and the given character.

2.1.2 Spacetime Constraints

Physics plays an important role in constraining how humans and animals move. Spacetime constraints were introduced in [49] as a means of automatically producing physically-realistic motion. The user specifies character attributes, as well as constraints on the desired motion, such as keyframes or foot placements that constrain the character at specific instants in time. Given an objective function related to control energy or motion smoothness, one can pose the animation as a trajectory optimization problem, with the goal of both satisfying the constraints and minimizing the objective function.

The spacetime constraints method has a number of limitations. The existence of analytic first and second derivatives is assumed, which is necessary to allow for the application of efficient local optimization techniques such as Sequential Quadratic Programming. Constraints can become cumbersome to specify even for simple motions and characters. It is often not clear what a suitable objective function should be. Another drawback is that the posed optimization problems may have many local minima and thus the local optimization method's success depends highly on the initial state, in this case an initial motion trajectory. Lastly, the optimization process can be prohibitively slow.

To partially address these issues, it is possible to use simplified dynamic models [31, 46], efficiently computed cost functions [11], and a priori knowledge of the specific problem [18, 22]. However, in the absence of initial motion capture data, spacetime constraints has not, to the best of our knowledge, been able to produce quality walking or running gaits for figures of human complexity. A number

of impressive results have been developed by using reference motions to initialize the optimization process [31, 36]. Nonetheless, spacetime constraints methods are trajectory based and output a set of poses over time. Such trajectories may not be reused if even minor changes are made to the environment and it has not been shown that the optimized trajectories are sufficiently accurate to drive a forward dynamics simulation even in the perfectly predictable environments assumed by the model. Lastly, trajectories can not be easily parameterized to enable generalization of the motion.

2.1.3 Motion Capture

Motion capture is among the most popular methods for producing walking motions. Here, an actor moves around in a studio and their 3D motions are directly recorded, usually as a set of joint angles sampled at 60 or 120 Hz. The raw motion capture data is once again specific to the given environment and the given actor.

By post-processing, the data may be adapted to satisfy new environmental, modeling, or timing constraints. For example, the data may be retargetted to characters of different dimensions or adjustments may be made to the speed and direction of the movement. A number of motion capture editing techniques have been developed to address these types of problems [12, 1, 30, 42]. However, similar to spacetime methods, the output of the motion capture process, including editing, is a trajectory of the degrees of freedom over time. This property limits the reusability and applicability of the data, as the character might be expected to exist in a continuously changing environment. For this situation, an online kinematic motion controller can be used to allow for a larger repertoire of motion and to create motions on the fly. In [4, 20, 34], novel motion is generated by cutting and pasting different motion sequences together. The possible points at which transitions can be made between motions are typically pre-computed and stored in a motion-graph data structure. These methods and others allow for the generation of novel motion sequences that can follow paths, accommodate for style and timing constraints, and can select among several kinds of captured motions, such as running and jumping.

The motion graph algorithms described above rely heavily on the richness of the recorded motion examples. More recent work looks at extracting parameterizations of motion from motion examples [7, 19, 28] and thus allows for interpolated motion styles instead of just re-sequencing of motion clips. However, the further one extrapolates from the given examples, the poorer the quality of the synthesized motion. Ultimately, one cannot record every possible motion type and every possible style, especially when the motion involves significant interaction with the environment or with other characters.

A glaring problem of motion capture techniques is that they are limited to real-world characters. As one cannot physically capture the motion of a character that is a figment of one's imagination, motion capture cannot make valid predictions about the feasible motions for imaginary creatures such as mech robots. The motion of many animals can also not be captured as they cannot be made to move in a desired way with motion capture apparatus attached to them. A kinematic model of motion is inherently limited in that there is no connection at all to the forces, muscles, sensory system, nervous system, and control strategies that play defining roles in how real humans and animals move.

2.1.4 Hybrid Kinematic/Dynamic Animation Methods

Motion capture methods can generate a rich repertoire of high-quality motion but in general they cannot interact with a continuously changing environment. Physically-based controllers, on the other hand, are more general and interactive, but do not yet produce high quality, stylized motion for complex human characters, as will be seen in the next section. Some research projects explore hybrid dynamic-kinematic controllers that can benefit from the advantages of both techniques, but where the dynamic control is mostly limited to upper body motion [27, 52, 53]. For example, [29] describes a hybrid gait controller in which the dynamic component is confined to falling and jumping motions.

2.2 Dynamic Methods

In physically-based modeling approaches to character animation, a character is usually specified as a set of rigid links connected together with actuated joints. The links each have different masses, lengths and moments of inertia. Motion is accomplished by generating torques at the various joints, and by applying ground reaction forces on links that are in contact with the ground. The torques and forces are used by the equations of motion to compute accelerations for all the character's degrees of freedom (DOF). The character's state, consisting of the DOFs and their time derivatives, is updated over time, given the known accelerations.

The challenge in making such forward-simulation methods work is deciding upon the required torques or muscle forces to apply, in order to generate a desired motion. This is a control problem and in this section we review approaches drawn largely from the robotics literature for solving this problem for walking gaits.

2.2.1 Zero Moment Point Methods

The Zero Moment Point (ZMP) was first described by [48] as the point on the stance foot at which the horizontal component of the moment created by the ground reaction forces vanishes. Equivalently, it can be shown that the ZMP coincides with the Center of Pressure (CoP) of the foot [13]. The sum of the ground reaction forces can be replaced by one force acting at the CoP. During the single stance phase of walking, the ZMP lies within the area of the supporting footprint. During double stance, the ZMP lies within the convex hull defined by both feet. These areas of support are often referred to as the *stability region*.

Many researchers have used the ZMP to define a motion trajectory that is guaranteed to be dynamically stable [15, 51, 50, 17, 41, 38]. As an example, in [38] this is done by first defining a walking pattern using several parameters such as walking stride length, hip height and walking phase periods. The walking pattern parameters are used to calculate initial trajectories for the hip positions, swing foot position, and joint angles over time. Given these, the walking pattern can be used

to compute a trajectory for the ZMP. If at any point along the trajectory, the ZMP is outside the stability region, then the motion is known to be unrealizable. In this case, the walking parameters can be adjusted to achieve a realizable ZMP trajectory. Thus, the ZMP is used as a criterion for the validity of motion trajectories.

Once joint trajectories with a realizable ZMP trajectory have been computed, a motion can be controlled to follow this desired ZMP trajectory [9, 14]. Motion capture data can also be used to create the desired ZMP trajectory [9]. The desired ZMP is distinguished from the actual ZMP that is generated from the motion trajectory using inverse dynamics. The motion trajectory is optimized by reducing the deviation of the desired ZMP from the actual ZMP.

In reality, due to disturbances, the robot will not be able to perfectly follow the desired motion trajectory. In this case the actual ZMP will not be the same as the desired ZMP and the robot may lose balance and fall. [14] present several strategies to correct for the deviation between ZMP's and thus regain balance. Adjusting the angles of the ankle joints can be used to shift the actual ZMP toward the desired one. The trajectory of the desired ZMP may also be changed to get closer to the actual ZMP. The above two control strategies will alter the ideal robot configuration thus changing the distance between the robot's upper body and the next footstep position. The next footstep is repositioned to avoid falling. This will change the stride length temporarily.

ZMP controllers are most popular in robotics because they can ensure robot stability in a controlled environment. However, these controllers are not robust to large unexpected changes in the environment, and may fail with even small pushes. The resulting motion is typically not energy efficient[1]. Moreover, the resulting motion is not natural looking, because the knees are usually bent during the entire motion in order to avoid kinematic singularities.

[1]This is due in large part to the use of high gain PD controllers.

2.2.2 Virtual Model Control

There is no intuitive way of setting the motion trajectory so that it produces a physically valid ZMP trajectory. The ZMP approach requires iterative modification of the desired motion trajectory. In response to these limitations, *virtual model control* [33] presents an intuitive method for controlling the locomotion of a character. The authors introduce virtual components that connect parts of the character to others, and parts of the character to the environment. These virtual components produced virtual forces that are used to guide the desired motion. Virtual forces are ultimately converted into realizable torques at the character's joints. For example, to induce the swing leg foot to move in a parabolic shape, a virtual spring and damper system is attached between the swing foot and the torso. The spring rest position changes as a function of the desired swing foot position. The forces calculated from the virtual spring and damper are translated into torques at the character's joints. Other virtual components may be used to induce a constant walking speed and body height. A finite state machine is used to induce a walking motion by activating and disabling different virtual components as a function of the current phase of the gait.

Such virtual components are intuitive to design and have trivial computational requirements. Moreover, this strategy is successful in controlling a character walking blindly over variable terrain. This method was tested on two small robots designed with lightweight legs in order to be easy to control. A drawback of this approach is that the virtual forces cannot necessarily be translated into actual torques during any given stage of the walk cycle and there is no notion of anticipation built into this model of control. Also, it is unclear how this method can be used to generate specific styles of walking.

2.2.3 Reinforcement Learning

The goal of Reinforcement Learning (RL) is to learn control strategies from delayed rewards [16]. Thus, for example, we would like a robot to learn how to walk by discovering which actions lead to falls and which ones lead to appropriate forward motion. In RL, a controller is known as an *action policy* or more simply, a *policy*.

The policy is a function that uses the current state of the character to decide on an appropriate action. After every action, the character's state is updated and the controller will receive positive or negative reinforcement, known as a *reward*, depending on some evaluation criteria. The controller needs to learn a policy that maximizes the expected future rewards.

Unfortunately, many RL methods suffer from the curse of dimensionality: the algorithms' time and memory needs grow exponentially with the dimensionality of the state. Another challenge is to find the right tradeoff between exploration (trying new actions) and exploitation (making decisions based on what worked before). Nevertheless, RL has been used to learn walking motions for simple, low dimensional, bipeds [24, 25, 43]. The approaches are not known to scale well to bipeds with higher number of DOFs.

One notable exception is presented in [39]. Here, RL was used to train a 19-DOF 3D biped to walk of flat ground and up and down slopes. The biped can adjust its stride length and direction. Careful preprocessing and decomposition of the control were used to help reduce the dimensionality of the problem in order to yield a tractable learning problem. However, this method has many parameters. Moreover it was tested on only one character and can produce only one style of motion.

The RL methods described above explicitly model the control system during learning. A *model* consists of a state-transition function and a reward function. Another approach to RL is to search for the policy directly, without learning a model of the system. A policy, or controller, can be represented, parameterized, and optimized in a variety of ways. Two such methods follow.

In one case, a character is augmented with sensors, detecting joint angles or contact, and actuators at the joints. A network connecting sensors and actuators can then be created and optimized to synthesize locomotion [45]. Such controllers require less knowledge about a unique representation of the character's state than other methods, as actions rely entirely on sensory information such as foot-ground contact. These controllers have produced interesting modes of locomotion for a

variety of simple 2D characters. However, the user has little or no control over the resulting motion, with speed being a typical optimization metric.

An alternative to sensor-actuator networks are banked stimulus-response controllers [5]. Here, a controller consists of a set of sensor-action rules. A Genetic Algorithm is used to globally optimize the controller parameters. The user may guide the search by providing an initial guess at the locomotion style and several controllers can be concatenated to generate a more complex motion repertoire. Although controllers have been shown to generate both 2D and 3D bipedal locomotion, finding an optimal controller is extremely slow in the 3D case. Similar to most dynamically-based controllers presented so far, the user has no control over the resulting style of motion. Furthermore, results are not robust and are given for flat ground only.

Imitation-Based Learning

The RL techniques presented so far attempt to learn a task from scratch. Others make use of prior knowledge about the task structure or hierarchy [23, 35] or from raw observations of human-captured performance [3, 26, 37]. These approaches are collectively known as *imitation-based learning* or *learning from demonstration*.

Bootstrapping the learning process with prior knowledge results in faster learning as the policy search can focus on areas of the search space that pertain to the task at hand. However, most techniques are limited to learning "one-time" motions such as the acrobat swing up task or weight lifting. Imitation based learning for a simple 5-link planar biped locomotion is demonstrated in [26]. Motion is, however, limited to flat terrain. Moreover, the imitation is reserved for learning the basic locomotion skill, rather than the style of locomotion presented in the demonstrations.

2.2.4 Other Specialized Controllers

Many of the specialized controllers begin by specifying the walking motion as a finite state machine with states such as double stance phase and swing phase. One such example is the work of [47], which associates each state with timing information and

a character pose. The controller will start at one state and will use Proportional Derivative (PD) control to drive the character to the pose associated with that state. After a fixed time, the controller will switch to another state. The pose and timing parameters associated with the states are not known in advance and are found by global and local search methods.

In [21], the authors control 3D simulated bipedal walking by introducing regulation variables to stabilize an open loop state machine. The user can control certain aspects of the motion such as speed and direction. The user can have limited control over style. The methods here are shown to work on a human and a non-human character. The method is not directly amenable to online control, however. The control for each actual step requires computing 4 trial simulations of that step in order to compute an online estimate of how parameter changes will affect the balance during the upcoming step.

2.3 Summary

We have presented some popular techniques used in the generation of bipedal locomotion. The literature exhibits an alarming trend of providing insufficient documentation of the capabilities and limitations of the controllers presented [32]. Moreover, new techniques are rarely compared to previous algorithms and most researchers work on substantially different platforms. We do not fully address these issues. However, we do test our method on a variety of characters and terrains.

Our controller representation has some similarities with the stimulus-response pairs used by Auslander et al [5], as will be shown in Chapter 4. Our approach differs significantly in that we allow the use of an example target motion to directly help define the motion style produced by a controller.

Our method also uses a form of imitation-based learning that is described in Section 2.2.3. However, we apply our strategy to more complex characters and motions. Moreover, we demonstrate more complete use of the motion example by

mimicking the style of the motion, thereby allowing the user greater control over the resulting motion.

Although some of the techniques described in this chapter can handle variable terrain and can work on various characters, most of the previous work assumes flat ground and work on limited range of characters. For example, motion capture techniques are mostly applicable to characters of a human-like structure. Dynamic techniques allow for interaction with the environment but most controllers for physically-simulated characters provide little or no control over resulting motion style. We address these issues in the following chapters.

Chapter 3

Dynamic Simulation

In this chapter we provide necessary background information related to our dynamic simulation environment. Figure 3.1 shows the main simulation loop and its key components. This chapter will discuss five of these components in some detail. The key variables in this figure are as follows: $x, \dot{x}, \ddot{x}, \alpha_d, \tau, F_g$.

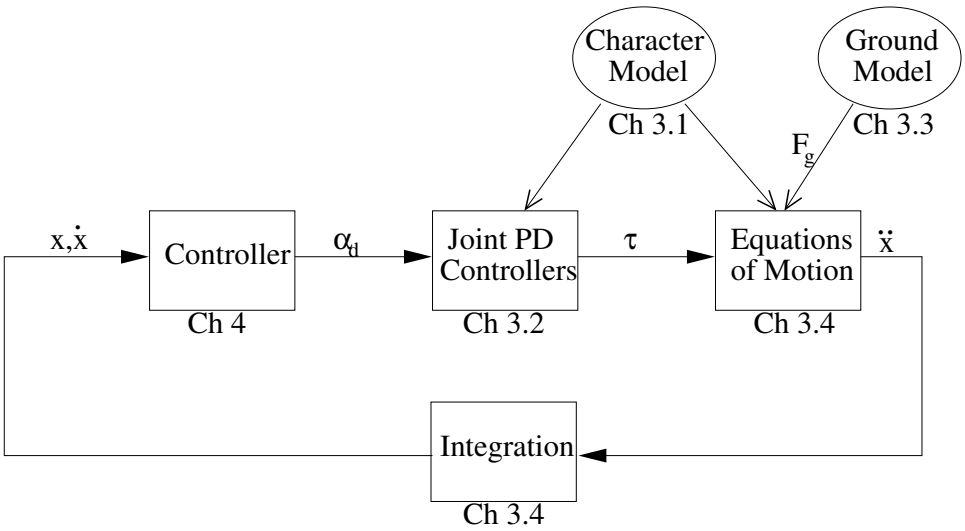


Figure 3.1: Simulation Loop

A summary description of the simulation loop operation is as follows. At each new time step, the controller takes the character's state, $\{x, \dot{x}\}$, as input, and outputs a set of desired joint angles, α_d. For now, we assume that the controller is a black box. A PD controller computes joint torques, τ, using the desired joint

angles, along with a specification of the character model. The ground reaction forces, F_g, are applied to the character at locations of contact with the ground. The character specification then serves to define the equations of motions that generate the character's acceleration, $\ddot{x}$, given the joint torques and the ground forces. The accelerations are integrated to update the character's state.

The structure of the remainder of this chapter is as follows. We describe our character models in Section 3.1. PD control in discussed in Section 3.2. The ground interaction model is described in Section 3.3. We discuss the equations of motions and the integration step in Section 3.4.

3.1 Character Modeling

We apply our control strategy to three planar biped models as shown in Figure 3.2. The human character in Figure 3.2(a) and the robot (mechbot) in Figure 3.2(c) both have 7 links and 6 pin-joints. The more complex human model in Figure 3.2(b) has 16 links and 15 joints. Each link has an associated set of defining parameters, including mass, length, center of mass (CoM) location, and moment of inertia. The values of these parameters can be found in Appendix A.

One of our goals was to test our algorithms on a variety of different characters. The 7-link human biped differs from the 7-link mechbot in several respects. First, the bipeds have different mass distributions and link lengths. The human has thigh, shin, and foot lengths and masses similar to those of an average adult human being. The mechbot model has much shorter thighs and shins. Second, the mechbot bends its knees inward. Last, the human's CoM is located directly above the character's hips. In contrast, the CoM of the mechbot is located ahead of its hips. These properties require different balancing and locomotion strategies. Our controllers are sufficiently general to adapt to these property changes.

The global location of a character in the world is given by the position of the root link. For the 7-link models presented above, the torso is the root. The 16-link character has its root located at the pelvis. The global position and orientation are

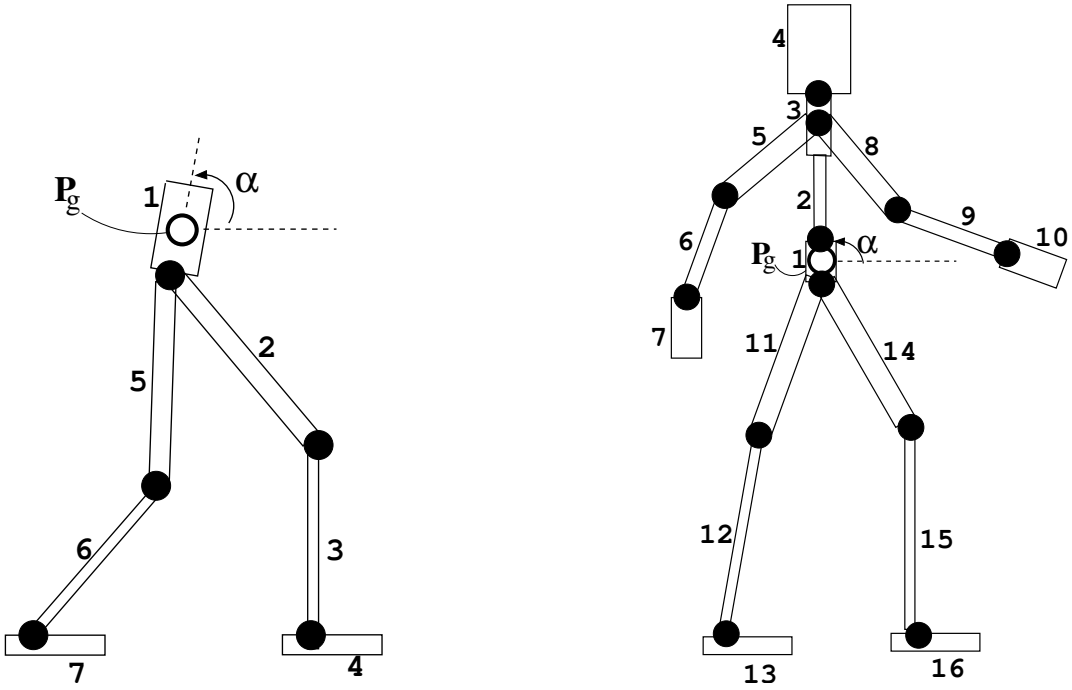


Figure 3.2: **Character Models** (a) 7-link human. (b) 16-link human. (c) 7-link mechbot.

given by P_g and α, respectively. Positions and orientations for all other links are given locally, with respect to their parent link. The origin of each local coordinate system is the point where the link attaches to its parent. The link CoM parameters in Appendix A are given with respect to the links' local coordinate systems.

3.2 Joint control

The character joints are actuated and can generate internal torques according to the following Proportional Derivative (PD) control equation:

$$\tau(t) = k_s(\alpha_d(t) - \alpha(t)) - k_d(\dot{\alpha}(t)), \tag{3.2.1}$$

where k_s and k_d are the spring and damper coefficients describing the joint's strength, α and $\dot{\alpha}$ are the angle and angular velocity of the joint at time t, and α_d is the current desired angle for the joint.

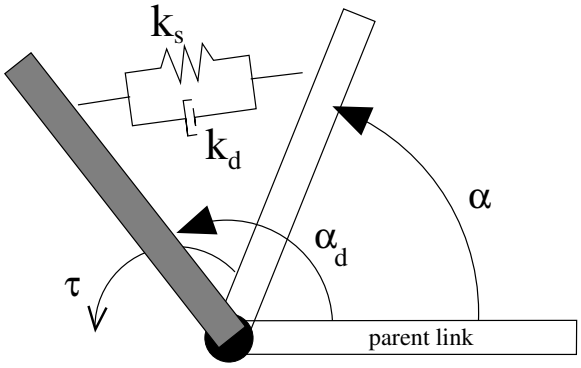


Figure 3.3: PD Controller.

Figure 3.3 illustrates a PD controller. The filled circle represents a joint between a link and its parent, represented by the white rectangles. α and α_d are the current and desired angles of the child link, respectively. Both α and α_d are given with respect to the parent link. The spring and damper system represents the PD controller. The controller drives the child link towards the desired target position given in gray. The joint's strength is modelled using the spring and damper

coefficients, k_s and k_d. The character's joint strengths and joint limits can be found in Appendix A.

If a joint angle limit is exceeded, an additional corrective torque is applied at the joint, computed as follows:

$$\tau(t) = k_1 \epsilon e^{k_2|\epsilon|} - k_3(\dot{\alpha}(t)) \tag{3.2.2}$$

$$\epsilon = \begin{cases} \alpha_{min} - \alpha(t), & \text{if } \alpha(t) < \alpha_{min} \\ \alpha_{max} - \alpha(t), & \text{if } \alpha(t) > \alpha_{max}, \end{cases} \tag{3.2.3}$$

where k_1 and k_2 are spring coefficients, k_3 is a damping coefficient, $\alpha(t)$ and $\dot{\alpha}(t)$ are the joint's angle and angular velocity at time t, and ϵ is the error between the joint angle, $\alpha(t)$, and the joint limit (either α_{min} or α_{max}). Thus, the magnitude of the corrective torque grows exponentially with the error, ϵ.

3.3 Ground Model

During simulation, we need to compute the ground reaction forces that arise during foot-ground contact. Generally, these forces are distributed along the foot length that is in contact with the ground. In practice, however, we cannot accurately model this continuous foot-ground interaction, and instead we apply forces at several key points along the foot.

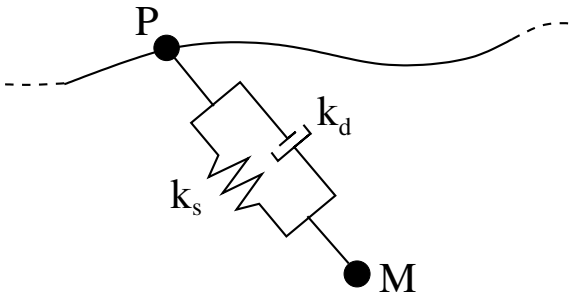


Figure 3.4: Ground Model.

The ground contact is modelled using a spring and damper system as shown in Figure 3.4. We define monitor points along the feet which will be tested for ground

contact. We record the position, P_i, at which the i_{th} foot monitor penetrated the ground as well as the current position, M_i, and velocity, $\dot{M_i}$, of the monitor point. An external force is computed that pushes the monitor point back towards the point of initial penetration of the ground. The external force is computed as:

$$F_i(t) = k_s(P_i - M_i(t)) - k_d(\dot{M_i}(t)), \tag{3.3.1}$$

where $F_i(t)$ is the external force applied at time t to the i_{th} foot monitor. k_s and k_d are the spring and damper coefficients, respectively. In our experiments, we typically use $k_s = 25,000N/m$ and $k_d = 2,500Ns/m$. Multiple monitor points can penetrate the ground simultaneously. Each such point is dealt with independently, using Equation 3.3.1.

3.4 Equations of Motion

A character's state is given by $\{x, \dot{x}, y, \dot{y}, \alpha_{1:j}, \dot{\alpha}_{1:j}, g_{left}, g_{right}\}$ and comprises both continuous and discrete variables. The continuous variables are the global position and velocity, $x, \dot{x}, y, \dot{y}$, the global orientation angle and angular velocity, $\alpha_1, \dot{\alpha}_1$, and the joint angles and velocities, $\alpha_{2:j}, \dot{\alpha}_{2:j}$, where the number of joints, j, depends on the character model. The discrete variables, g_{left}, g_{right} are booleans for the feet that are set to 'true' when the left or right foot, respectively, is in contact with the ground.

The equations of motion are derived using a reduced-coordinate Newton-Euler formulation. A full coordinate specification of the character's state would involve global positions and orientations for each link and would require constraints to ensure that the ends of connected links do not drift apart. Such a formulation can be costly in the absence of sparse matrix techniques and requires stabilization of the constraints. In a reduced, or *minimal coordinate* formulation, a global position and orientation is provided for one of the links, and the remaining links are specified using orientations relative to their parent links. For our application we have found the reduced coordinates formulation to be faster and more convenient. The accelerations

computed by the equations of motion are integrated using an Euler integration scheme.

3.5 Summary

In this chapter we presented the three character models controlled by our system. We also described how joint torques and ground reaction forces are computed and used to update the character's state. Next, we present our controller strategy.

Chapter 4

Learning Controllers for Walking

This chapter describes our approach for developing a controller for bipedal walking behaviors. Key choices include the controller representation to be used, the general optimization procedure to learn the free parameters of the controller, and, importantly, the choice of optimization function. In the remainder of this chapter, we discuss our choices, first in general terms, and then more specifically with respect to walking.

4.1 Controller Representation

Our controller is a mapping from an input state, ζ, to an output action, μ. The controller consists of a collection of nodes, $\omega_{1:n}$, that are used to determine a nearest neighbor control scheme. Each node ω_i consists of a center state, $\zeta \in I^m$, and a target action, $\mu \in O^p$, where I^m and O^p define the controller's *input-space* and *output-space*, respectively. During simulation, the center state parameters locate each node in space and are used to determine which node is *active*. The *active node*, ω_a, is defined as the nearest-neighbor to the current known state of the dynamically simulated character and is computed as

$$a = argmin_i(||\mathbf{w}^T(\zeta - \zeta_i)||_2). \tag{4.1.1}$$

where a is the index of the active node, ζ_i is the center state of the i_{th} node, ζ is a projection of the current full state[1] of the character onto the controller's input-space, and $\mathbf{w} \in R^m$ is a vector of weights for the various elements in ζ. The nature of the state projection as well as the weighting vector will be discussed in Section 4.3.1. Once the active node is found, its target action defines the controller's output, μ_a, which is used to drive the character.

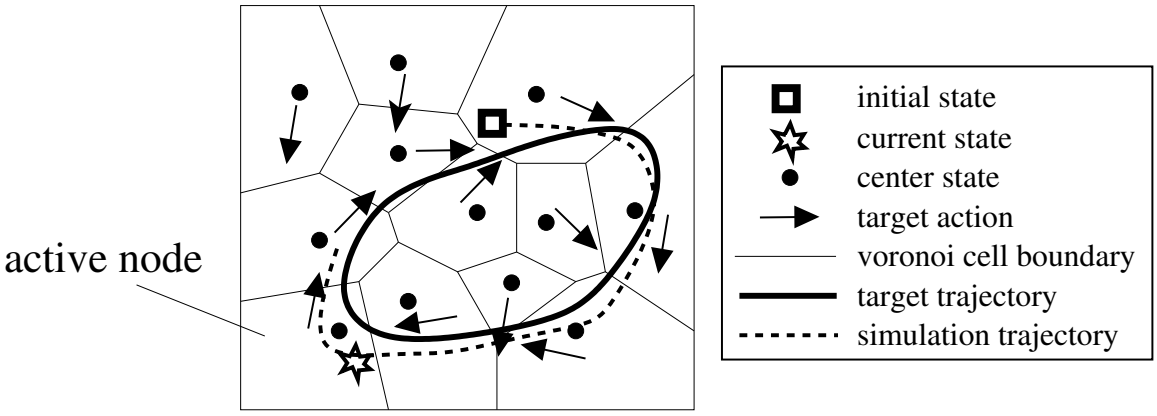


Figure 4.1: Abstraction of Controller Representation and Operation

Figure 4.1 offers an abstraction of our controller representation in operation. The thick solid line is a projection of a motion trajectory that we want our character to follow. The filled circles are the node center states, $\zeta_{1:n}$, and the arrows are an abstraction of the node target actions, $\mu_{1:n}$. The center states partition the input-space into a set of high-dimensional Voronoi cells. As can be seen from the figure, the cell actions generally drive the dynamical system, i.e., the moving simulated character, back onto the desired trajectory. The thick dashed line represents a simulated trajectory as guided by our controller. The simulation begins with the character in the position denoted by the square. The star is the current state of the simulation. A nearest neighbor search reveals that the bottom left node is active

[1] As discussed in Chapter 3, the character's full state is defined by the character's DOFs and their time derivatives.

(i.e., closest to the star). The controller will now use this node's target action to drive the simulation. Eventually, the system will leave this node's neighborhood region and another node will be activated.

The node centers and node actions are free parameters of the representation and are thus not known in advance. The values of these parameters are determined through an optimization, the goal of the optimization being to follow a given desired motion trajectory. An overview of the optimization process is given in the next section.

4.2 Controller Optimization

We employ a simple deterministic search (SDS) algorithm, also known as *coordinate search* [44], to find the optimal controller parameters, with respect to an evaluation criterion, or *cost function*, $C(\cdot)$. Our specific choice of cost function is documented in Section 4.3.3. The core SDS algorithm is summarized below in Figure 4.2. We let Θ denote the vector of all controller parameters, $[\zeta_{1:n}, \mu_{1:n}]$. In the absence of known derivative or gradient information we explore the corresponding parameter space by making systematic evaluations of $C(\Theta \pm \Delta\Theta_i)$ for each free parameter i in Θ. SDS uses a (local) hill climbing approach by iteratively updating Θ_{opt} if it yields a better cost function value.

We iteratively reduce the size of parameter perturbations, α. Using this coarse-to-fine method allows us to first explore larger neighborhoods in the parameter space and then refine our search once we get closer to our local minimum.

4.3 Walking Controller Realization

The controller strategy presented above is general enough to work on a variety of control problems. In [2], this controller was adapted to a significantly different control problem: steering cars and trucks with trailers, forwards and backwards around winding tracks. In order to apply this method to walking control, we adapted our controller representation and optimization in several respects. The following sub-

Input:

starting point Θ
perturbation step size α
decay factor β
$e_{best} \leftarrow C(\Theta)$

Algorithm:

repeat
  for each element i in Θ **do**
    $\Delta\Theta \leftarrow \alpha E$ $\{E \triangleq$ the Kronecker delta, $\delta_i\}$
    $e_{new} \leftarrow C(\Theta + \Delta\Theta)$
    if $e_{new} < e_{best}$ **then**
      $\Theta \leftarrow \Theta + \Delta\Theta$
      $e_{best} \leftarrow e_{new}$
    else
      $e_{new} \leftarrow C(\Theta - \Delta\Theta)$
      if$e_{new} < e_{best}$ **then**
        $\Theta \leftarrow \Theta - \Delta\Theta$
        $e_{best} \leftarrow e_{new}$
      end if
    end if
  end for
  $\alpha \leftarrow \beta\alpha$
until converged

Output:
$\Theta_{opt} \leftarrow \Theta$

Figure 4.2: Simple Deterministic Search algorithm (SDS)

sections provide controller alterations and implementation details that are specific to the generation of walking motions.

4.3.1 Controller Input and Output

The input of our walking controllers is the character's *pose* along with foot-ground contact information. A pose is defined by the character's angular DOFs, i.e., its global orientation along with its joint angles. Poses are thus easily extracted from the full character state, which is the collection of the character's DOFs and their

time derivatives, as discussed in Chapter 3. Using the pose instead of the full state significantly reduces the number of free parameters.

To find the active controller node, the current character pose is compared to the node center states, $\zeta_{1:n}$, as described in Section 4.1. The angles comprising the character pose or center states are each weighted differently using the weight vector, $\mathbf{w}$, in Equation 4.1.1. A joint angle weight is given by the associated mass of the lower link that the joint is attached to. For example, the ankle joint weight is set to the foot link mass and the hip joint weight is the mass associated with the thigh link. The weight for the global orientation is set to the mass of the torso.

Walking motion is naturally divided into phases - double stance, where both feet are in contact with the ground; and single stance, where only one foot is supporting the character, and the other foot is swinging from the back to the front. The nodes in our walking controller incorporate this phase information by supplementing the node center states with information about left and right foot contact. When comparing the state of the dynamic simulation with each node, the Left and Right foot contact state, as given by g_{left}, g_{right} (see Section 3.4), must be the same for the simulation and the node. Given the importance of foot-ground contact during walking, this phase information is an easy way to enforce particular nodes to be dedicated to particular phases of the walk cycle. We also take advantage of the left-right symmetry of these phases as we shall discuss in Chapter 5.

The output of a controller, expressed in Section 4.1 as a target action, μ, is a set of desired joint angles. Once an active node is found, we use PD controllers to drive the joints towards their desired angles.

4.3.2 Target Motion Trajectory

The goal of our controller is to mimic a style, as specified by a user-provided target motion trajectory. The target motion is defined by a set of keyframes as shown in Figure 4.3. Each keyframe specifies the character's pose at a given time. One of the legs is dashed to make the left and right legs distinguishable. Keyframes are linearly interpolated over time to produce the target motion trajectory. The time values in

Figure 4.3 show the time it takes to transition from one keyframe to the next. Other methods, such as motion capture, could also be used to acquire the desired motion. We shall see shortly how the target motion trajectory plays an important role in defining our cost function.

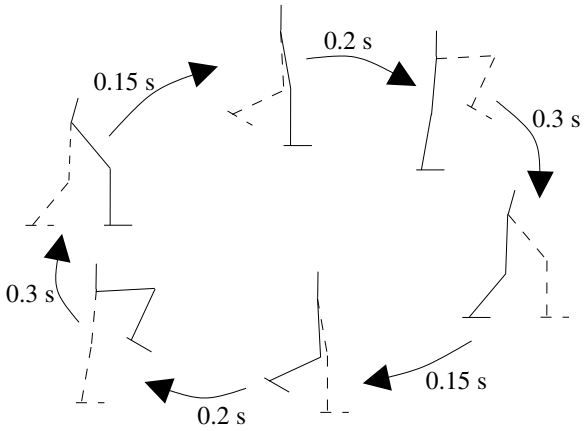


Figure 4.3: Keyframes.

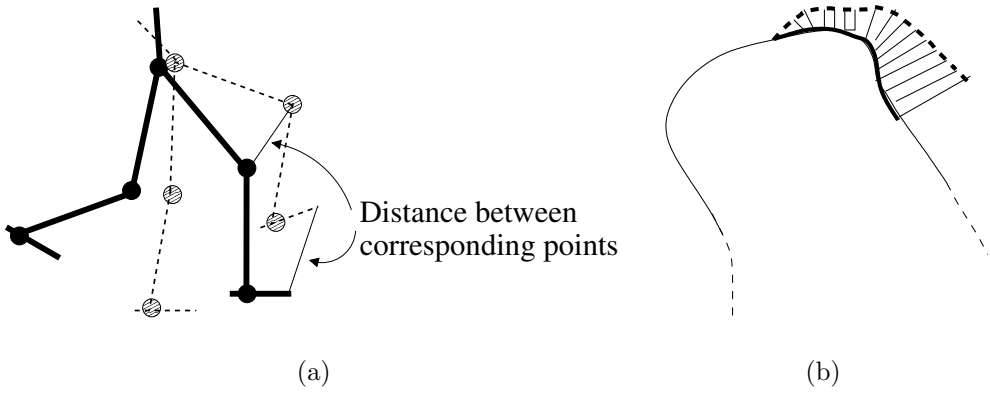
4.3.3 Cost Function

The choice of an appropriate cost function is important and challenging. Our controller has two potentially conflicting goals. On the one hand we wish our character to follow a desired trajectory. On the other hand, our character must maintain its balance while obeying the laws of physics, something that the target trajectory might not do. We design well-shaped cost functions for walking by using a form of imitation-based learning. By rewarding how well a controller mimics a desired target motion, we provide a convenient handle for the design of the desired style of walk. It also implicitly rewards balanced walking because actions that result in falls will cause larger deviations from the desired motion.

Our cost function, $C(\Theta)$, integrates the deviation of the physics-based walk simulation from a target walk cycle. At time t, the error, $\delta(t)$, between the dynamically simulated character and the kinematic target walk cycle is defined using a mass-distance metric integrated over all character link lengths:

$$\delta(t) = \sum_{i=1}^{M} \int_{L} \rho_i (x_i(t) - x_i'(t))^2 dx, \tag{4.3.1}$$

where M is the number of character links, L is the length of a link, $x_i(t)$ is a point on the i_{th} link at time t for the dynamically-simulated biped, $x_i'(t)$ is the location of the same point at the same time instant for the target walk cycle, and ρ_i is the associated density for the i_{th} link. The density is a link's mass per unit length and allows us to weight character links differently, assuming that errors at heavier parts of the body, such as the torso, are more important than errors at lighter parts, such as the foot. Figure 4.4(a) shows the distances at two points on the character.



(a) (b)

Figure 4.4: Cost Function: (a) Instantaneous motion tracking error. (b) An abstraction illustrating the error over multiple time steps.

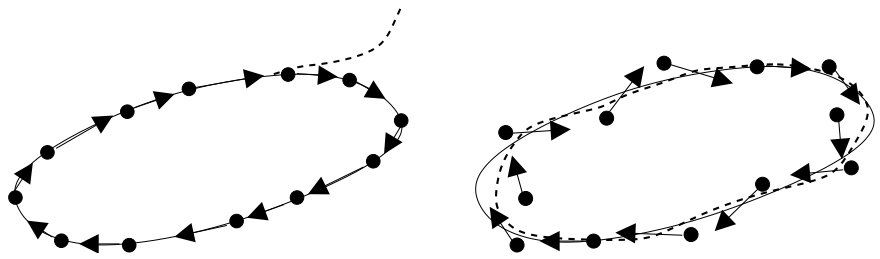
The dashed character shows the desired pose at time t. The solid character refers to the dynamic pose at time t. The error is computed by measuring the distance between corresponding points along the character's links. The knee and toe distances shown here are only illustrative examples. In practice, we approximate the continuous mass-distance integral by representing each link using two point masses, one placed at each end of the link. Each point mass is assigned half the weight of the link. Integrating this error over time gives us the cost function:

$$C(\Theta) = \int_{0}^{T_{sim}} \delta(t) dt \tag{4.3.2}$$

where T_{sim} is the duration of the simulation and $\delta(t)$ is as defined in Equation 4.3.1. The choice of T_{sim} is important and will be discussed further shortly. Figure 4.4(b) illustrates the cost function being integrated over time. The thick solid line is the desired trajectory for a period of time, T_{sim}. The dashed line is a dynamic simulation for the duration T_{sim}. The cost function is calculated by integrating the distance between these two trajectories over time.

4.3.4 Initialization

In local search algorithms such as ours, a successful optimization may largely depend on the parameter initialization. We initialize controller nodes to appropriately-spaced positions along the kinematic target trajectory, as shown in Figure 4.5(a). The number of nodes is chosen based on experience and generally depends on the complexity of the motion trajectory we wish to mimic. See Chapter 5 for choice examples and a more detailed discussion. Both center poses, ζ_i, and target poses, μ_i, lie directly on the trajectory. We assign $\mu_i = \zeta_{i+1}$, i.e., each node's target action will drive the character to the center state of the next node. Such an initialization is intuitive and it helps shape the controller optimization by focusing on those areas of the pose-space that are of interest, namely, those surrounding the desired trajectory. The dashed line in Figure 4.5(a) is an example simulated trajectory and indicates that the initial guess at the values of the controller parameters is suboptimal.



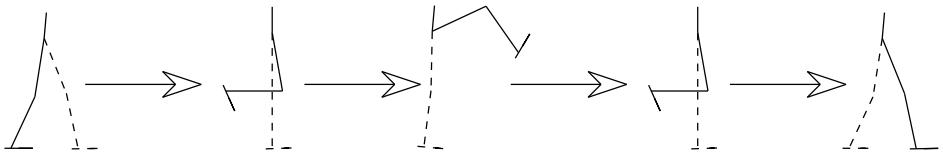
(a) (b)

Figure 4.5: Controller Initialization and Optimization: (a) Initial controller parameters. (b) Optimized controller parameters.

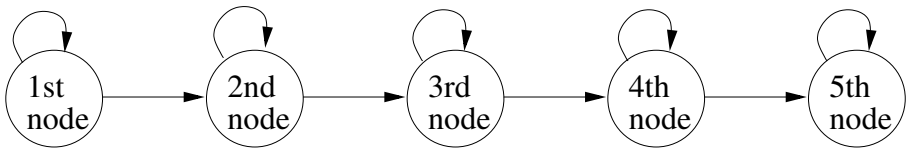
Figure 4.5(b) illustrates the same controller after optimization. We see a slight shift in some of the center and target poses. The dashed line representing the dynamic trajectory shows that we can adjust the node parameters, successfully tracing the desired trajectory to obtain balanced walking.

4.3.5 Connecting Nodes

Poses may repeat at different parts of a desired motion trajectory. For example, the motion in Figure 4.6(a) has identical poses at the second and fourth keyframes. If the simulated character is in this pose, our controller would not know wether to drive the character towards the third pose or the fifth pose. That is, we might have two controller nodes with similar or identical center states, but with significantly different target actions. To solve this problem, we create a directed graph of control nodes, as shown in 4.6(b). The controller's nearest neighbor search will only measure the character's pose against the center states of the current active node, and its successor nodes in the directed graph. Thus, when the character reaches the duplicate pose, there is no longer ambiguity as to which node to select, since only one of them will directly succeed the currently active node.



(a)



(b)

Figure 4.6: Duplicate Poses: (a) Keyframes with identical poses. (b) Directed graph associated with a controller.

4.3.6 Bootstrapping

We optimize over progressively longer durations, i.e., increasing T_{sim}. The optimized controller for a short walk simulation is used as the initial guess for the next iteration, which then optimizes for a longer walk simulation. Bootstrapping the optimization in this manner prevents the optimization process from getting trapped in errant local minima. For cases where a target trajectory is physically infeasible, the controller will follow the target trajectory during short term optimizations, but, over a longer time interval, will deviate as necessary in order to better follow the target trajectory in the long term. Figure 4.7 illustrates this progression. The thin solid curve represents part of the target trajectory we wish to mimic. The thick solid curve is the kinematic simulation after time T_{sim}, and the dotted curve is the dynamic simulation using the current controller after the same amount of time. Figure 4.7(a) shows that an initial controller fails to follow the target motion even for a short period of time. After optimizing, we generate better results, as seen in Figure 4.7(b). However, the optimized controller only works for the short period of time it was tested on. Figure 4.7(c) shows that if the simulation runs for a longer time, the current optimal controller fails. Figure 4.7(d) illustrates our controller after parameter optimization with a larger value of T_{sim}. Our mass-distance cost metric and the long time intervals over which it is evaluated distinguish our use of a target trajectory from other control methods which track a reference trajectory only very locally.

(a) (b) (c) (d)

Figure 4.7: Optimizing over Progressively longer Periods: Before (a) and after (b) optimization with short simulation evaluation period. Before (c) and after (d) optimization with longer simulation evaluation.

4.3.7 Introducing Disturbances

In order to adjust the node centers towards the regions of state-space that the character will visit, the character must indeed visit those regions during the optimization process. To ensure that the optimization routine sees a sufficient range of situations, we introduce small variations in the terrain to induce perturbations, even when the goal is to produce a stable walk on perfectly level terrain. This significantly enhances the stability of the final gaits.

4.3.8 Torso Controller

The torso plays an important role in both the balance of a walk and its style. Because of this, the torso is provided with a separate PD controller whose goal is to drive the torso to a desired global orientation which is given by the target trajectory.

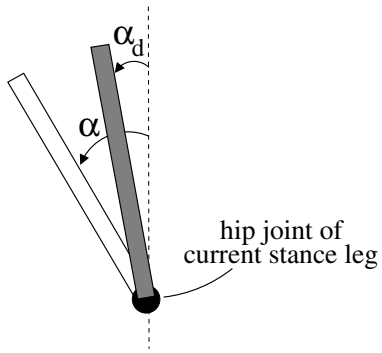


Figure 4.8: Torso Correction.

Figure 4.8 illustrates how the torso controller is used to correct the global orientation. α and α_d are the character's current and desired global orientations, respectively. These cannot be manipulated directly. Instead we add the torque in Equation 4.3.3 to the current stance hip, which pushes the torso towards the desired global orientation. Note that this stance hip torque is equal and opposite to a PD controlled external torque applied to the torso.

$$\tau = k_p(\alpha - \alpha_d) + k_d(\dot{\alpha}) \tag{4.3.3}$$

4.4 Summary

In this chapter we presented a nearest-neighbor controller strategy for generating walking motions. We use a hill climbing algorithm and several shaping techniques to optimize controllers that can stably mimic a desired style. In the next chapter, we demonstrate a number of results.

Chapter 5

Results

In this chapter we present results in the form of walking motions produced by our controller synthesis method. Our controller was tested for generality along several dimensions as shown in Figure 5.1. First, we ensure that our method can produce several styles of walking for a given character. We describe the style results in Section 5.1. Next, we test the generality of our method by applying our controller to different characters. Section 5.2 presents motion results for several characters. We experiment with terrains of various slopes to ensure that our controller is not restricted to level terrain. Section 5.3 describes the results of our uphill and downhill tests. Finally, we test our system for robustness using unobserved perturbations in the terrain. Results for this are given in Section 5.4. Experiments were performed on machines ranging from a dual Xeon 2.66GHz machine with 4GB of RAM to a 1GHz Pentium III with 512M RAM.

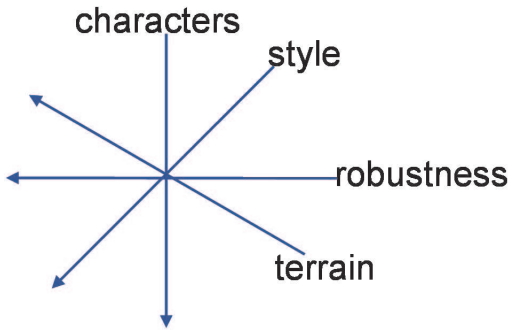
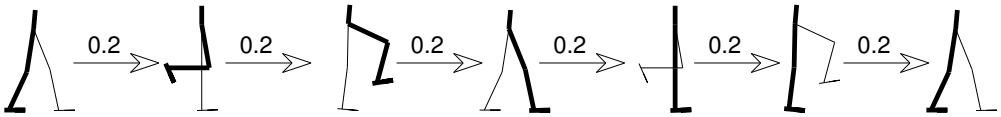


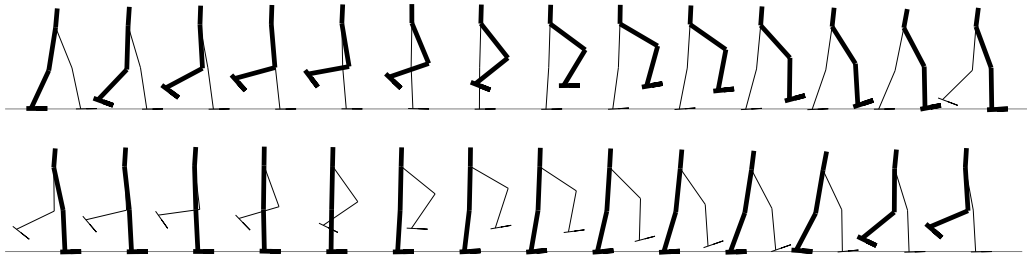
Figure 5.1: Controller Dimensions

5.1 Styles

We test a number of different walking styles for the 7-link human biped specified in Chapter 3. The first and simplest style is shown in Figure 5.2. The keyframes that define the desired walking style are shown in Figure 5.2(a). Each step is defined by 3 keyframes yielding a total of 6 keyframes for the full walk cycle. For clarity, one of the legs is drawn with a thinner line. All the walks are left-right symmetric, both in terms of the keyframes defining the target motion, as well as the controller nodes. The arrows between the keyframes annotate the transition time between the keyframes, in this case, 0.2 seconds.



(a)



(b)

Figure 5.2: Regular Walk

Figure 5.2(b) shows the first few steps of a walk generated by our optimized controller. Every 100th frame is drawn and successive frames are spaced out so as not to overlap. Notice that the style of walk is very close to the style specified by the keyframes in Figure 5.2(a). The controller has 12 nodes which are initialized along the target motion defined by the keyframes. Each node has 12 free parameters: 6 for the node center pose and 6 for the node action. Since the nodes are left-right

symmetric, we only optimize half of the nodes. In total, we thus have 6 nodes $\times$ 12 parameters per node = 72 free parameters requiring optimization.

For the optimization algorithm in Figure 4.2, we use an initial perturbation step size of $\alpha = 0.3$, and initial decay factor $\beta = 0.1$. We optimize for simulation duration times, T_{sim}, ranging from $0.5s - 1.2s$, with a step size of $\Delta t = 0.1s$. Once the optimization reached 1.2 seconds of simulation, a stable controller was found that produced the desired style. A controller is considered to be stable if it can produce a walking motion for at least 1000 simulation seconds without falling down.

Figure 5.3 shows the progress of our controller optimization routine by plotting the controller error against the number of iterations. The controller error is defined in Section 4.3.3 as the deviation of the dynamic character's motion from the desired kinematic motion. In this graph, one iteration is the process of perturbing one parameter and evaluating the resulting controller. Note the jumps in the error graph. These discontinuities result from the cost function changing during the optimization as longer values of T_{sim} are used. Since the cost is a function of T_{sim}, every increase of T_{sim} will result in a discontinuity in the cost function. Optimizing the controller in Figure 5.2(b) required 3,758 iterations, which took approximately 5 minutes of CPU time.

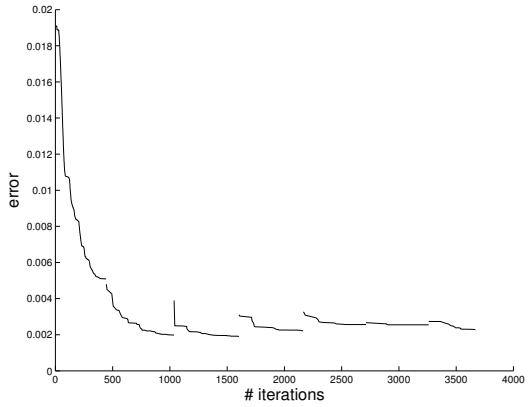
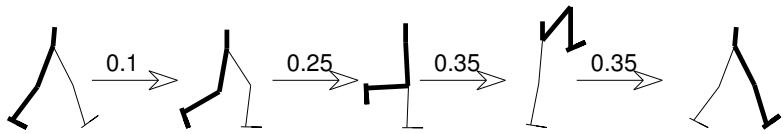


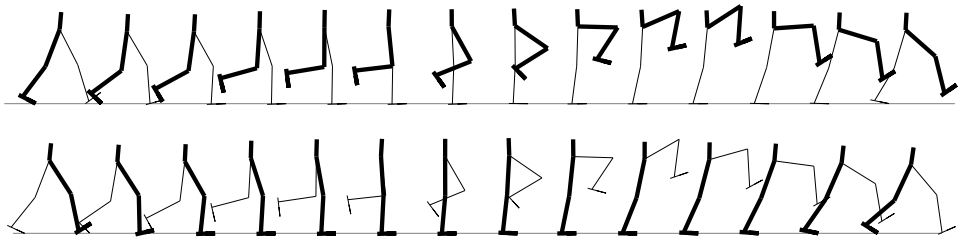
Figure 5.3: Regular Walk Error

A second style is shown in Figure 5.4. Figure 5.4(a) shows the keyframes defining the desired motion. This style differs from the first in several ways. The

swing leg is made to lift very high. The first two keyframes show that we desire the character to remain in double stance for 0.1 seconds while moving the weight from the back leg to the front in a specific style. A single step takes 1.05 seconds to complete, compared to 0.6 seconds of the previous style. Figure 5.4(b) shows a walking motion from our controller. The style matches the target motion closely, retaining the slow motion, the high swing leg lift and the slow weight shift from one leg to the next.



(a)

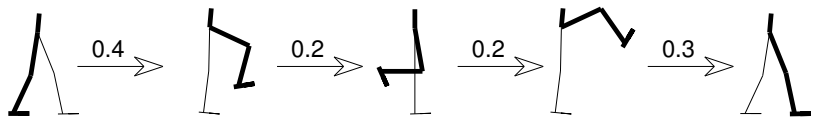


(b)

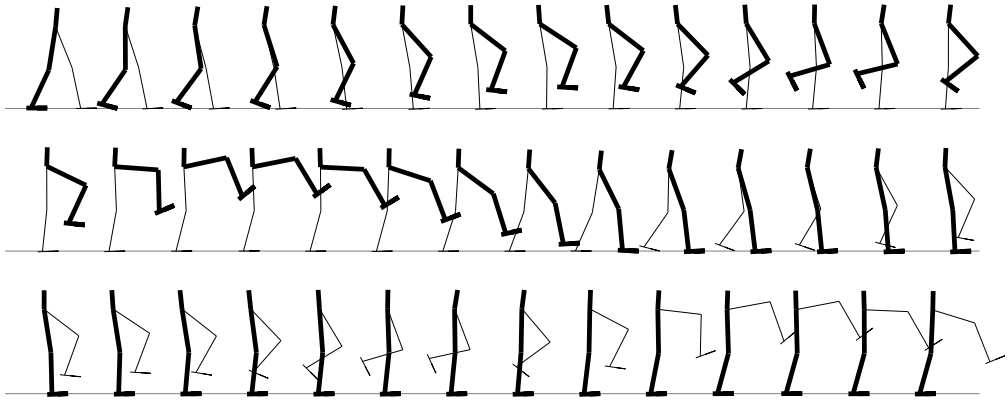
Figure 5.4: Extended Swing Walk

This controller was defined using 24 nodes. With left-right symmetry enforced, only 12 nodes need optimizing. As before, each node has 12 free parameters, resulting in a total of 144 free parameters. Optimization was performed for simulation evaluation times of $0.1s - 1.0s$, with simulation time increases of 0.1 seconds step sizes. A stable controller was found in under 10 minutes of CPU time, after 8,934 iterations.

A final style is shown in Figure 5.5. Figure 5.5(a) shows the desired motion keyframes. Notice that this walk exhibits a rather esoteric additional backwards swing of the swing leg. The swing leg swings forwards in the second keyframe, then



(a)



(b)

Figure 5.5: Double Swing Walk

backwards in the third keyframe, and once more forwards in the fourth keyframe before the footstrike. A single step takes 0.9 seconds. In Figure 5.5(b) we see the first few steps of the motion generated by our controller. The character preforms the double swing as requested by the keyframes.

Similarly to the previous style, this controller was defined using 24 nodes, 12 of them free to be optimized. The controller had 144 free parameters. Interestingly, it took only 450 iterations and 1.5 minutes to optimize this controller, even though the motion is likely much harder in that it requires delicate balance to succeed. We speculate that the fast convergence is likely due to a fortuitous initialization of the nodes. Center and target poses of each node were initialized to poses along the desired trajectory as defined by the keyframes. In this case, the initial controller parameters produced a good walk for the first 2 seconds of simulation. Thus, we begin the optimization using $T_{sim} = 2sec$, compared to previous styles where we

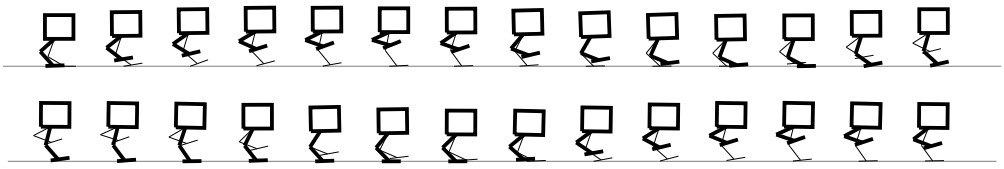
begin from $T_{sim} = 0.5sec$ or $T_{sim} = 0.1sec$. A stable controller was readily found with $T_{sim} = 2sec$.

5.2 Characters

We next test the ability of our method to control other characters. We test our method on a total of three characters, where one is the 7-link human already discussed in the previous section. We first discuss the 7-link mechbot character specified in Chapter 3.

Figure 5.6(a) shows the keyframes for a desired mechbot walk. Figure 5.6(b) show the walking result using our mechbot controller. The controller has 12 nodes, each of which has 12 free parameters, the same as for the 7-link human character. The optimization thus has 72 parameters to optimize. A stable controller was found after 22,420 iterations, using 2.4 hours of CPU time. The optimization routine ran simulations from 0.1 - 3.1 seconds. Surprisingly, this walk took much longer to optimize than for the 7-link human. We speculate this is because the much shorter stride length leads to a smaller region of support and thus a less stable walk.

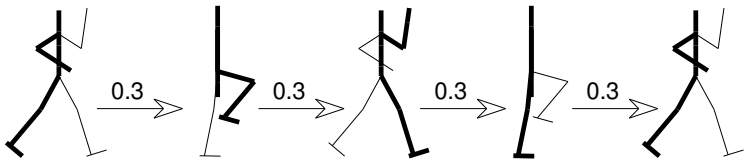
(a)



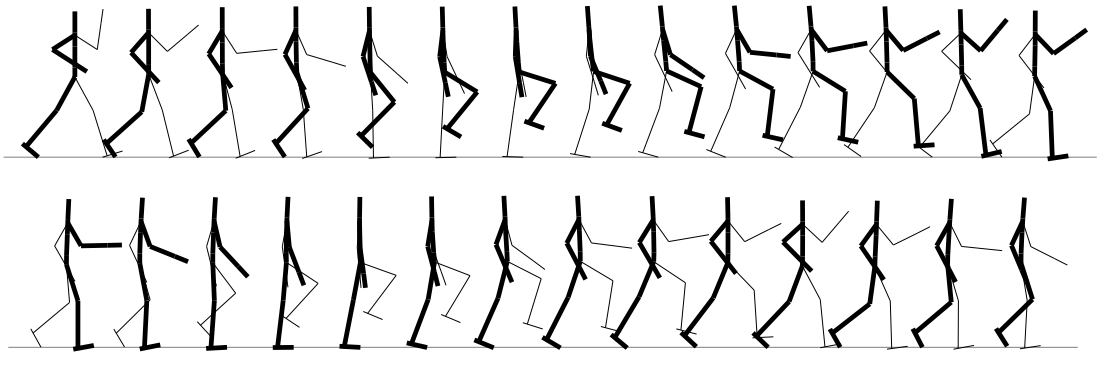
(b)

Figure 5.6: Bird Walk

The final character is the 16-link human biped described in Chapter 3. In Figure 5.7(a), we see the keyframes used to define the target walk. Note that the arms in the second and fourth keyframes are parallel to the character's upper body and are thus difficult to discern. Figure 5.7(b) shows a dynamic simulation using our controller.



(a)



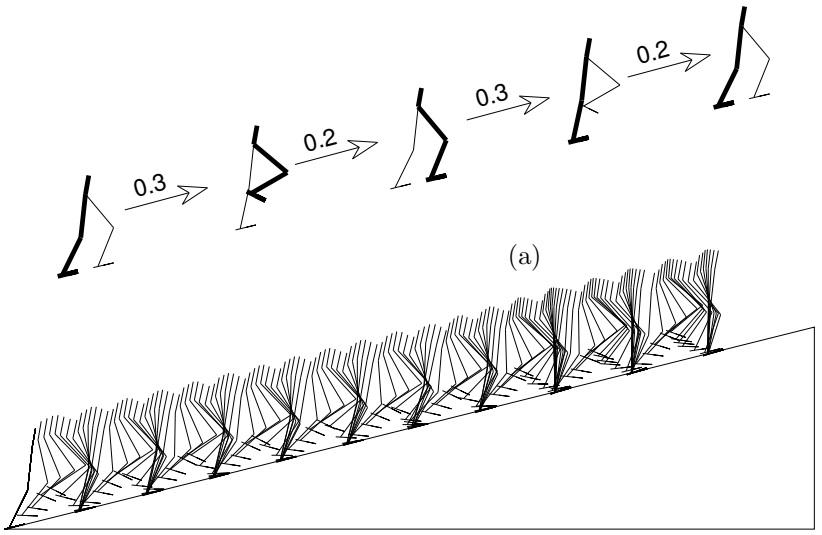
(b)

Figure 5.7: 16-Link Biped Walk

The controller uses 12 nodes. Each node's center pose or target pose contains 16 parameters - one for global orientation, and 15 for joint angles. However, we leave many of them constant, optimizing only for the hip, knee and ankle joints for the center and target poses. As a result, we have 12 free parameters for each node, the same as for the 7-link characters. This is because we assume that the lower body can react appropriately to the predictable upper body movements. A stable walk was discovered within 4,193 iterations, taking 30 minutes of CPU time.

5.3 Terrain

To ensure that we could control motion on sloped terrain, we test our method on the 7-link human biped walking both uphill and downhill. Note that we do not use the same controller as we did for the case of flat ground. Both of the following controllers are created and optimized for the specific slopes presented to them.



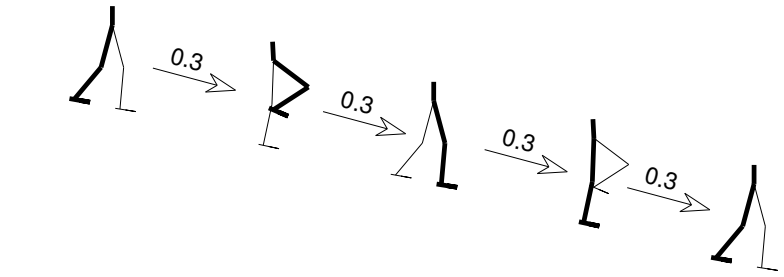
(a)

(b)

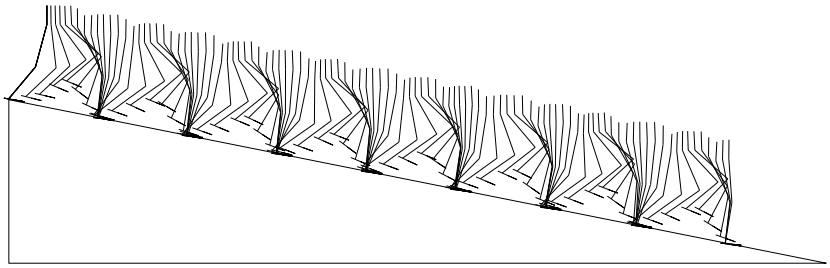
Figure 5.8: Uphill Walk

Figure 5.8(a) shows the keyframes used to define a desired walk up a hill. The hill had a slope of 22.5 degrees. The controller had 12 nodes, leaving a total of 72 parameters to be optimized. Left-right symmetry leaves 6 nodes, each having 12 free parameters. The optimization ran simulations over evaluation intervals of 0.5 - 15 seconds. An optimal controller was created within 5 hours, after 20,165 iterations. Figure 5.8(b) shows a dynamic simulation of our controller. Only one leg is drawn for clarity.

Figure 5.9(a) shows the keyframes used to define a desired downhill walk. The hill had a slope of 18 degrees. The optimization ran simulations from 0.5 - 10.9



(a)



(b)

Figure 5.9: Downhill Walk

seconds. An optimal controller was created after 59,728 iterations and 21 hours of CPU time. Figure 5.9(b) shows a simulation of our controller.

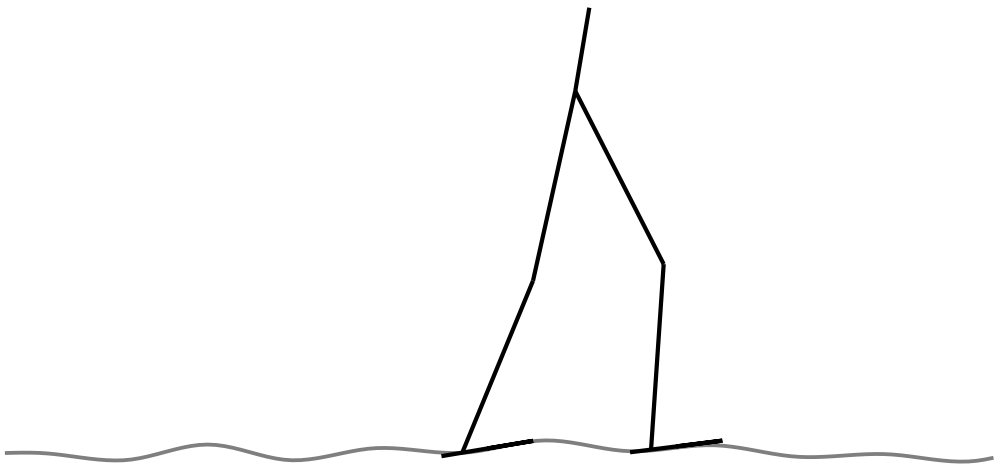
5.4 Robustness

The results presented thus far have produced stable walks on flat and sloped terrains. The stability arises from the stable basin of attraction created by the controller. If a character drifts off of the desired trajectory, the active node will drive the character back onto the limit cycle. However, if the character drifts too far, the small set of nodes will not be able to correct the character's motion. If our character is to be robust to sudden changes in its state or the state of the environment, we must require that the controller recover from larger deviations. Our hypothesis is that with additional controller nodes, we will be able to control the character in a larger region of the state space.

In order to test this assumption, we take the simplest controller presented in Section 5.1 and triple the number of nodes, yielding 36 nodes and 215 controller parameters to optimize. To ensure that the character would constantly be pushed off its limit cycle, we presented the character with bumpy terrain. The character is not aware in any way to the upcoming terrain, and must continuously recover from unexpected changes in state.

(a)

(b)



(c)

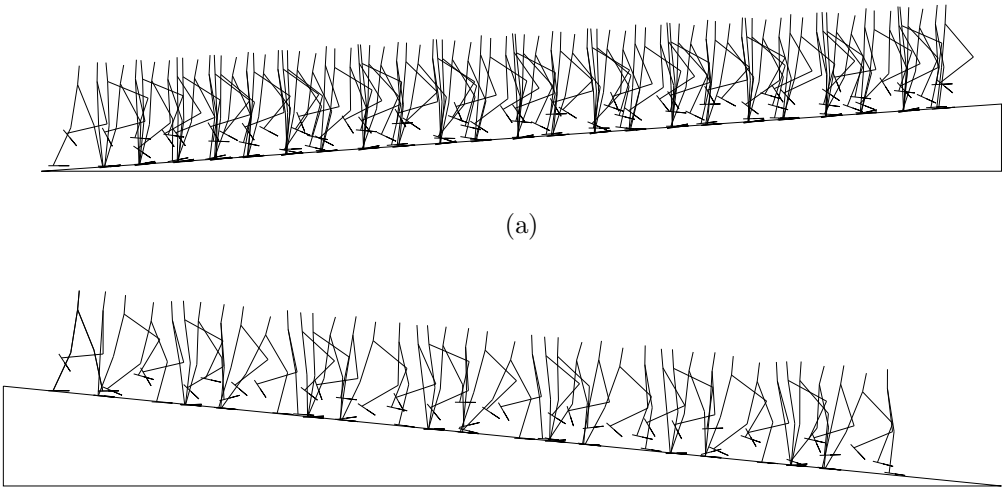
Figure 5.10: Bumpy Terrain Walk

We optimize the controller over heavily bumpy terrain with local slope gradients between 12.5% - 30%. At each stage, we present the controller with 20 randomly

generated terrains. We search at two resolutions; each parameter was perturbed by $\alpha = 0.3$ and $\alpha = 0.01$. After 130,000 iterations, requiring approximately one week of CPU time, we test our controller on terrain of various bumpiness. We define a controller to be successful if it can generate 70 seconds of walking over variable terrain without falling. Figure 5.10(a) shows the controller driving a character on a moderately bumpy terrain with gradients of 12.5% or less. Our controller succeeds at walking on 20 randomly generated terrains that were not seen during optimization. The controller's performance degrades smoothly with an increase in terrain gradient. Figure 5.10(b) shows a successful walk over a terrain having up to 20% gradients. However, our controller succeeds on only 70% of the 20 randomly generated terrains of this type. Figure 5.10(c) provides a closeup of the terrain.

There are various reasons for the limited controller capabilities. First, the search process does not present the controller with all possible situations during optimization. This is a challenging problem as there exist an infinite number of possible terrains. The explosion of possibilities worsens if combined with all possible character states on that terrain. Naturally, we cannot train the controller on all cases it might enter into. Second, the controller may have had too few nodes to allow for appropriate actions for the given range of situations. We use our experience to choose the required number of nodes. Moreover, there is a trade off between the number of nodes and the time it takes to optimize them.

We also test our robust controller on terrains with constant slope. The simple controller of Section 5.1 failed on slopes as small as 2%. Figure 5.11 shows the controller successfully walking uphill on a 7% slope, and downhill on a 10% slope. Notice that the steps in these results are irregular and thus do not appear as smooth as the ones in Figures 5.8 and 5.9. However, those controllers were specifically trained on those slopes. Here, the character continuously expects flat ground and must recover at each step.



(a)

(b)

Figure 5.11: Uphill and Downhill Walks with a Robust Controller

5.5 Summary

We have demonstrated our controller's ability to generate walks for a variety of user-specified styles, for a variety of characters, and for several terrain slopes. Moreover, a robust controller was presented that successfully walks over unobserved variations in terrain. In the next chapter we discuss our contributions as well as the known limitations of our method. We conclude with possible directions for future work.

Chapter 6

Conclusions

As shown in Chapter 5, our method is capable of generating controllers for various planar biped characters that walk in a desired style and are robust to unexpected state and environment changes. In this chapter we summarize our contributions (Section 6.1), point out known limitations (Section 6.2), and elaborate on future directions (Section 6.3).

6.1 Contributions

Compact Controller Representation

In Chapter 4 we described a controller representation that divides the state into Voronoi cells, and chooses an action based on which cell the simulation is currently closest to. By optimizing the free parameters in this compact controller representation, we were able to create walking controllers for various characters and walking styles.

An important property of our controller representation is that it does not suffer from the curse of dimensionality. The number of nodes does not have to increase with the character dimension. Particularly, for a well defined motion such as walking, we need not be concerned with all areas of state-space, but rather only poses close to the desired state-space trajectory. Our scheme could be classified as a semi-parametric representation [6].

Variety of Characters

Unlike much previous work, we present three different characters on which we test our method. The characters differ in both number of links and their mass distribution. We describe how these differences could lead to different control strategies for producing balanced walks. In Chapter 5 we show how our controller was general enough to accommodate for these differences.

Control of Walking Style

Unlike previous work on walking control, we provide the user the ability to control the style of the walk. We shape our cost function using a motion trajectory specified by the user and impose much weaker conditions on this desired trajectory than ZMP or other reference trajectory approaches. Our controllers are evaluated on how closely they could imitate the desired motion which also implicitly rewards balance. In Chapter 5 we show that our controller can generate several distinct, user-specified walking styles.

6.2 Limitations

The following is a discussion of the known limitations of our system. We note that most other walking techniques share similar limitations.

Optimization Time

If the controller we wish to generate is simple[1], an optimal controller can be found within a minute, after hundreds of iterations. However, for more difficult motions, our optimization routine can be painfully slow. For example, for the downhill walker, the routine took approximately 60,000 iterations to find a stable solution. An unavoidable reason for slowness is the high dimensionality of the problem. Exploring

[1]Deciding what makes a controller or walking motion simple is an open problem. In this context, a simple controller is one that does not require the character to be off balance for a long time interval during the motion.

a high-dimensional space is time consuming. Another reason may be the naive optimization algorithm used. The optimization perturbs one parameter at a time, and does not compute a gradient for the cost function.

Hand Tuned Parameters

Hand tuned parameters are a notorious problem in motor control methods. We manually tune tens of parameters including PD control coefficients for joints, ground, and joint limits. One of the implications of hand tuning is that our controllers are not as general as we would hope, since different characters require different parameter values.

Lack of Convergence Guarantees

Our optimization routine involves a cost function that changes during the course of the optimization, given that we optimize over progressively longer simulation times. For a constant simulation time, T_{sim}, we have convergence guarantee given that we use a simple hill-climbing method. However, no matter how large T_{sim} gets, we can never guarantee that the controller will operate successfully for a simulation time larger than T_{sim}.

A related problem is that we may chance upon a controller, $\Theta_{opt,1}$, early during the optimization, that happens to produce a stable indefinite walking motion, but we do not have a good way of discovering such immediate success. Since we can not determine if our controller is truly stable in the long run, we may continue optimizing, and produce a better $\Theta_{opt,2}$. Although $\Theta_{opt,2}$ is better than $\Theta_{opt,1}$ for the simulation time that the controller was optimized for, $\Theta_{opt,2}$ may still fail for longer periods of simulation. Thus, there is no way to identify the superiority of $\Theta_{opt,1}$ over the long run.

Unsuccessful Optimization

During our experiments, we have had unsuccessful optimizations. In these cases, our controller would get stuck in a bad local minimum. To solve these, we experimented with different hand tunable parameters as described in Section 6.2.

No Planning

Our controller is reactive in nature. It observes the character's state at a given time and decides on a proper action to take. This type of controller will fail in a more complex environment. For example, if the ground ahead contains holes or obstacles, our controller is incapable of planning ahead to adjust its behavior to accommodate upcoming changes.

Lack of Parameterization

Representations for motor control should ideally provide support for parameterization. It is natural to consider parameterizing a walking gait according to walking speed, stride length, and other stylistic parameters. We have not yet explored such parameterization of controllers.

Parameter Correlations

We assume that the input space for the controller has been defined in a compact, non-redundant fashion such that it can be used to compute meaningful distance metrics. If two or more dimensions of the state descriptor are highly correlated, our method cannot exploit this and may in fact be hampered by this.

Variable Terrain

The success of our method depends on the situations the controller has seen during training. It currently does not compare well with virtual model control methods [8] for blindly adapting to terrain variations. One possible solution is to consider better optimization methods that ensure the controller experiences more situations during

training. We speculate that the nearest neighbor approach will be able to generalize control to situations that are similar to those encountered during training, but will not succeed if a situation is entirely different.

6.3 Future Work

Better Optimization Algorithms

Since our cost function is a complex, discontinuous, non-linear function, we cannot compute an analytical gradient. We evaluate our cost function by running several simulations and integrating a cost function over time. Thus, we cannot take advantage of standard optimization routines that rely on first and second derivative information, such as Sequential Quadratic Programming. Moreover, we optimize tens to hundreds of parameters, which makes gradient approximation methods such as finite-differences prohibitively slow and unsuccessful. One possibility is to experiment with other gradient approximation methods, such as such as the Simultaneous Perturbation Stochastic Approximation (SPSA) method presented in [40]. The gradient approximation in SPSA requires only two cost function evaluations, as opposed to potentially hundreds, as in the finite-differences approach. However, in SPSA, one must define several learning rate parameters that are crucial to successful optimization and setting these correctly and consistently requires investigation.

It may also be possible to use the Voronoi cells to store value functions or Q functions and to therefore apply more efficient policy-iteration methods instead of direct policy search.

Multiple Samples

Our basic optimization method always begins simulations from a specific initial state. During the initial stages of optimization, we are simulating for very short periods, and thus run the risk of being trapped in a local minimum that will fail to control the walking motion for longer durations. One way to potentially alleviate this problem would be to run multiple simulations for every optimization episode.

Instead of evaluating the cost function from one initial state, one could evaluate it from multiple states that are distributed along the desired trajectory. Thus, parameter changes would not be accepted unless they provide reasonable solutions for the entire trajectory. We further speculate that controllers trained from multiple initial points will be more robust because they will have seen a larger range of situations.

Energy Efficiency

Currently, our cost function only takes into account the deviation of a simulated trajectory from a given target trajectory. There is no penalty for exerting excessive amounts of energy by applying unnecessarily large torques at the joints. This may lead to unnatural motions because we are most familiar with human and animal gaits that tend to minimize energy expenditure. We could choose to penalize large torques by adding a corresponding term to the cost function.

Adaptive Nodes

Currently, we choose the number of nodes based upon our experience. This is less than ideal, as we may end up with too few or too many nodes. If we have too few nodes, then we run the risk of not being able to find a suitable controller. If we choose too many nodes, our optimization routine will be unnecessarily expensive. A possible improvement is to add or remove nodes in some adaptive fashion. For example, if a node does not properly represent a region surrounding a trajectory, we could split it. If nodes are not being used, we can prune them. These are typical operations in semi-parametric fitting methods [6].

Dynamically Driven Motion Capture

In Chapter 2 we reviewed techniques that attempt to derive controllers from motion capture data. Our strategy takes a significant step towards solving this difficult problem because of our use of kinematic motion trajectories during controller optimization.

Appendix A

Model Parameters

link	name	mass (kg)	inertia	length (m)	COM(x)	COM(y)
1	torso	40	0.1333	0.2	0.00	0.0
2, 5	thigh	7	0.1234	.46	-0.23	0.0
3, 6	shin	4	0.0645	.44	-0.22	0.00
4, 7	foot	2	0.00807	.22	0.057	0.00

Table A.1: **Link Parameters for the 7-Link Human**

Joint	k_s	k_d	α_{min}	α_{max}
hip	900	90	-57	180
knee	900	90	-180	0
ankle	170	17	-115	-57

Table A.2: **Joint Parameters for the 7-Link Human**

link	name	mass (kg)	inertia	length (m)	COM(x)	COM(y)
1	torso	25	0.1875	0.3	0.05	0.05
2, 5	thigh	7	0.0233	.2	-0.1	0.0
3, 6	shin	4	0.0133	.2	-0.1	0.00
4, 7	foot	2	0.00666	.2	0.02	0.00

Table A.3: **Link Parameters for the mechbot Model**

Joint	k_s	k_d	α_{min}	α_{max}
hip	800	80	-115	180
knee	800	80	0	180
ankle	190	19	-180	-57

Table A.4: **Joint Parameters for the Mechbot Model**

link	name	mass (kg)	inertia	length (m)	COM(x)	COM(y)
1	pelvis	9.8	0.0817	0.1	0.00	0.0
2	lower torso	9.8	0.0395	0.22	0.0	.19
3	upper torso	15.4	0.0289	0.15	0.0	.075
4	head	5.6	.0226	0.22	0	.13
5, 8	upper arm	1.96	.0128	.28	0	-.1
6, 9	lower arm	1.12	.00732	.28	0	-.1
7, 10	hand	.42	.00079	.15	0	-.04
11, 14	thigh	7	.0933	.4	0	-.15
12, 15	shin	3.255	.0678	.5	0	-.2
13, 16	foot	1.015	.00409	.22	0	-.05

Table A.5: **Link Parameters for the 16-Link Human**

Joint	k_s	k_d	α_{min}	α_{max}
neck	100	10	-.785	.785
upper-to-lower back	500	50	-.785	.785
lower back -to- pelvis	500	50	-.785	.785
shoulder	100	10	-3.1	3.1
elbow	50	5	0	3.1
wrist	50	5	-.785	.785
hip	300	30	-57	180
knee	300	30	-180	0
ankle	170	17	-115	-57

Table A.6: **Joint Parameters for the 7-Link Human**

Bibliography

[1] Y. Abe, C. K. Liu, and Z. Popovic, *Momentum-based parameterization of dynamic character motion*, ACM SIGGRAPH / Eurographics Symposium on Computer Animation, 2004.

[2] K. Alton, Master's thesis, University of British Columbia, in preparation.

[3] R. Amit and M. Mataric, *Learning movement sequences from demonstration*, Proceedings of the IEEE International Conference on Development and Learning, June 2002, pp. 203–208.

[4] Okan Arikan and D. A. Forsyth, *Interactive motion generation from examples*, Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM Press, 2002, pp. 483–490.

[5] Joel Auslander, Alex Fukunaga, Hadi Partovi, Jon Christensen, Lloyd Hsu, Peter Reiss, Andrew Shuman, Joe Marks, and J. Thomas Ngo, *Further experience with controller-based automatic motion synthesis for articulated figures*, ACM Transactions on Graphics **14** (1995), no. 4, 311–336.

[6] C. M. Bishop, *Neural networks for pattern recognition*, Clarendon Press, 1995.

[7] M. Brand and A. Hertzmann, *Style machines*, Proceedings of ACM SIGGRAPH, 2000, pp. 183–192.

[8] C.-M. Chew, J. Pratt, and G. Pratt, *Blind walking f a planar bipedal robot on sloped terrain*, Proceedings of the International Conference on Robotics and Automation, May 1999.

[9] A. Dasgupta and Y. Nakamura, *Making feasible walking motion of humanoid robots from human motion capture data*, Robotics and Automation, vol. 2, 1999, pp. 1044–1049.

[10] Boston Dynamics, *www.bdi.com*, 2004.

[11] A. C. Fang and N. S. Pollard, *Efficient synthesis of physically valid human motion*, Proceedings of ACM SIGGRAPH, vol. 22, 7 2003, pp. 417–426.

[12] M. Gleicher, *Retargeting motion to new characters*, Proceedings of ACM SIGGRAPH, July 1998, pp. 33–42.

[13] A. Goswami, *Foot rotation indicator (fri) point: A new gait planning tool to evaluate postural stability of biped robots*, IEEE International Conference on Robotics and Automation, 1999, pp. 47–52.

[14] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenake, *The development of honda humanoid robot*, IEEE International Conference on Robotics and Automation, 1998, pp. 1321–1326.

[15] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, *Planning walking patterns for a biped robot*, IEEE Transactions on Robotics and Automation **17** (2001), no. 3, 280–289.

[16] L. Kaelbling, L. Littman, and A. Moore, *Reinforcement learning: a survey*, Artificial Intelligence Research **24** (1996), 237–285.

[17] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, *Biped walking pattern generation by using preview control of zero-moment point*, Proceedings of the IEEE International Conference on Robotics and Automation, September 2003, pp. 1620–1626.

[18] H. Ko and N. Badler, *Animating human locomotion in real-time using inverse dynamics, balance and comfort control*, IEEE Computer Graphics and Applications, vol. 16, March 1996, pp. 50–59.

[19] L. Kovar and M. Gleicher, *Flexible automatic motion blending with registration curves*, Proceedings of Symposium on Computer Animation, July 2003, pp. 214–224.

[20] L. Kovar, M. Gleicher, and F. Pighin, *Motion graphs*, Proceedings of ACM SIGGRAPH, 2002.

[21] J. F. Laszlo, M. van de Panne, and E. Fiume, *Limit cycle control and its application to the animation of balancing and walking*, Proceedings of ACM SIGGRAPH, August 1996, pp. 155–162.

[22] C. Liu and Z. Popovic, *Synthesis of complex dynamic character motion from simple animations*, Proceedings of ACM SIGGRAPH, July 2002, pp. 408–416.

[23] J. Morimoto and K. Doya, *Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning*, 17th International Conference on Machine Learning, vol. 1, 2000, pp. 623–630.

[24] J. Morimoto, G. Zeglin, and C. G. Atkeson, *Minimax differential dynamic programming: Application to a biped walking robot*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.

[25] Y. Nakamura, M. Sato, and S. Ishii, *Reinforcement learning for biped locomotion*, International Conference on Artificial Neural Networks, 2002, pp. 777–782.

[26] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, *Learning from demonstration and adaptation of biped locomotion with dynamical movement primitives*, Workshop on Robot Learning by Demonstration, IEEE International Conference on Intelligent Robots and Systems, 2003.

[27] A. Nakazawa, S. Nakaoka, K. Ikeuchi, and K. Yokoi, *Imitating human dance motions through motion structure analysis*, IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, October 2002, pp. 2539–2544.

[28] S. I. Park, H. J. Shin, and S. Y. Shin, *On-line locomotion generation based on motion blending*, SCA, 2002, pp. 105–111.

[29] A. Shapiro F. Pighin and P. Faloutsos, *Hybrid control for interactive character animation*, Pacific Graphics, 2003, pp. 455–461.

[30] N. S. Pollard and F. Behmaram-Mosavat, *Force-based motion editing for locomotion tasks*, Proceedings of the IEEE International Conference on Robotics and Automation, April 2000, pp. 663–669.

[31] Z. Popovic and A. Witkin, *Physically based motion transformation*, Proceedings of ACM SIGGRAPH, August 1999, pp. 11–20.

[32] J. Pratt, *Bipedal walking robots: Advancing the science through killer applications, replication and validation, standards and common platforms, and competition*, Proceedings of the FIRA Robot World Congress, May 2002, pp. 111–114.

[33] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, *An intuitive approach for bipedal locomotion*, International Journal of Robotics Research **20** (2001), no. 2, 129–143.

[34] K. Pullen and C. Bergler, *Motion capture assisted animation: Texturing and synthesis*, Proceedings of ACM SIGGRAPH, 2002, pp. 501–508.

[35] M.T. Rosenstein and A.G. Barto, *Robot weightlifting by direct policy search*, Proceedings of IJCAI 2001, pp. 839–844.

[36] A. Safanova, J. K. Hodgins, and N. S. Pollard, *Synthesizing physically realistic human motion in low-dimensional, behaviour-specific spaces*, Proceedings of ACM SIGGRAPH, 2004.

[37] S. Schaal, *Learning from demonstration*, Advances in Neural Information Processing Systems 9 (M. Jordan M. C. Mozer and T. Petsche, eds.), 1997, pp. 1040–1046.

[38] C.-L. Shih, Y. Z. Li, S. Churng, T. T. Lee, and W. A. Gruver, *Trajectory synthesis and physical admissibility for a biped robot during the single-support phase*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1990, pp. 1646–1652.

[39] R. Smith, *Intelligent motion control with an artificial cerebellum*, Ph.D. thesis, University of Auckland, New Zealand, 1998.

[40] J. C. Spall, *Introduction to stochastic search and optimization: Estimation, simulation, and control*, John Wiley and Sons, Hoboken, New Jersey, 2003.

[41] T. Sugihara, Y. Nakamura, and H. Inoue, *Realtime humanoid motion generation through zmp manipulation based on inverted pendulum control*, Proceedings of the IEEE International Conference on Robotics and Automation, May 2002, pp. 1404 – 1409.

[42] S. Tak, O-Y Song, and H-S Ko, *Motion balance filter*, Proceedings of Eurographics, vol. 19, 2000, pp. 437–446.

[43] R. Tedrake, T. W. Zhang, and H. S. Seung, *Stochastic policy gradient reinforcement learning on a simple 3d biped*, IEEE International Conference on Intelligent Robots and Systems, 2004.

[44] V. Torczon, *On the convergence of pattern search algorithms*, SIAM Journal on Optimization **7** (1997), no. 1, 1–25.

[45] M. van de Panne, *Sensor-actuator networks*, Proceedings of ACM SIGGRAPH, 1993, pp. 335–342.

[46] ———, *From footprints to animation*, Computer Graphics Forum **16** (1997), no. 5.

[47] M. van de Panne, R. Kim, and E. Fiume, *Virtual wind-up toys for animation*, Graphics Interface, 1994, pp. 208–215.

[48] M. Vukobratovic and D. Juricic, *Contribution to the synthesis of biped gait*, IEEE Transactions on Biomedical Engineering, vol. 16, 1969, pp. 1–6.

[49] A. Witkin and M. Kass, *Spacetime constraints*, Proceedings of SIGGRAPH, vol. 22, August 1988, pp. 159–168.

[50] R. Zhang and P. Vadakkepat, *An evolutionary algorithm for trajectory based gait generation of biped robot*, Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems, 2003.

[51] ———, *Motion planning of biped robot climbing stairs*, Proceedings of FIRA Robot World Congress, 2003.

[52] V. B. Zordan and J. K. Hodgins, *Tracking and modifying upper-body human motion data with dynamic simulation*, Eurographics, 1999, pp. 13–22.

[53] ———, *Motion capture-driven simulations that hit and react*, Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2002, pp. 89–96.