Rising Motion Controllers for Physically Simulated Characters

by

Benjamin James Jones

B.S. Computer Science, B.S. Engineering Physics, Colorado School of Mines, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

July 2011

Abstract

The control of physics-based simulated characters is an important open problem with potential applications in film, games, robotics, and biomechanics. While many methods have been developed for locomotion and quiescent stance, the problem of returning to a standing posture from a sitting or fallen posture has received much less attention. In this thesis, we develop controllers for biped sit-to-stand, quadruped getting-up, and biped prone-to-stand motions. These controllers are created from a shared set of simple components including pose tracking, root orientation correction, and virtual force based control. We also develop an optimization strategy that generates fast, dynamic rising motions from an initial statically stable motion. This strategy is also used to generalize controllers to sloped terrain and characters of varying size.

Preface

Chapter 5 describes work included in "Locomotion Skills for Simulated Quadrupeds," published in ACM Transactions on Graphics 30(4), 2011, a collaboration with Stelian Coros, Andrej Karpathy, Lionel Reveret, and Michiel van de Panne. The image in Figure 2.4 is included from "Exploiting the global dynamics structure of whole-body humanoid motion–getting the knack of roll-and-rise motions" by Yasuo Kuniyoshi, Yoshiyuki Ohmura, Koji Terada, Akihiko Nagakubo, Shin'ichiro Eitoku, and Tomoyuki Yamamoto published in Robotics and Autonomous Systems 48(4), 2004, with permission from Yasuo Kuniyoshi. The images in Figures 2.1, 2.2, 2.3 and 5.3 are used with permission from Michiel van de Panne.

Table of Contents

List of Tables

List of Figures

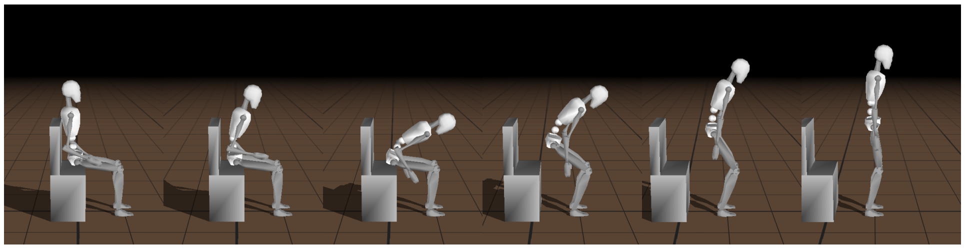
Chapter 1

Introduction

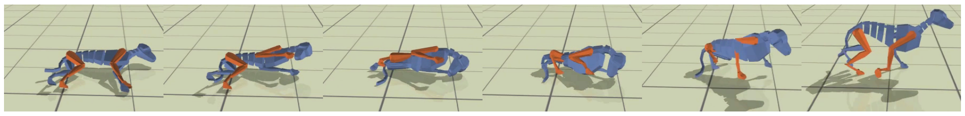
Generating motion for animated characters is an important and challenging problem for the film and games. There has been significant research in this area, which can be divided into kinematic methods and dynamic methods. Kinematic methods generate character poses over time but do not guarantee physically plausible motions, and often only have support for a predetermined set of interactions with other characters or the environment. Motion trajectories are often supplied by motion capture data, an expensive and labor intensive process.

Dynamic methods use the equations of motion to ensure that the generated motions are physically valid. At each frame, the controller computes actuation torques for each joint. This is a simplification for the aggregate effect of muscle activity in humans or animals. Forward simulation methods compute control torques online, while trajectory optimization techniques compute the entire control and motion sequence at once. Forward simulations methods are therefore better suited to changing environments or unpredicted interactions. Unlike kinematic methods, dynamic methods do not have direct control over character motion, as the computed control torques are combined with other physical constraints to generate the character motion. The increased realism of the motion is thus attained at the cost of more complex, indirect control of the movement as compared to kinematic methods.

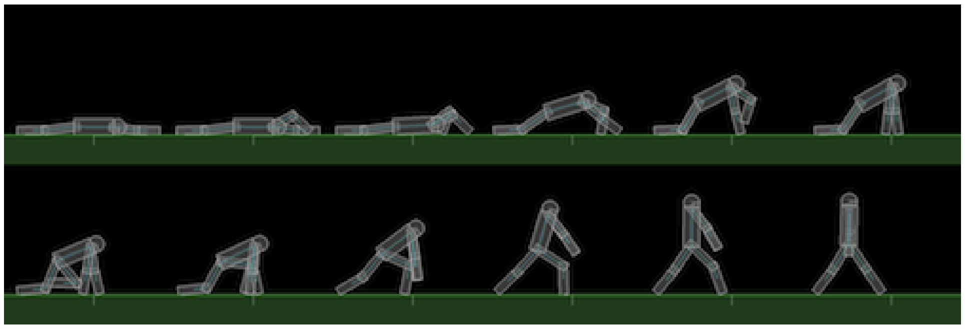
Much work on dynamic controllers for animation has focused on locomotion, with positive results. However, most legged characters are inherently unstable. For example, a humanoid, which has much of its mass concentrated in the upper body,



(a)



(b)



(c)

Figure 1.1: Sample results from Chapters 4, 5 and 6. Top to bottom: 3D biped sit-to-stand motion; 3D quadruped lateral decubitus-to-stand; 2D prone-to-stand.

will fall when pushed unless a corrective action such as stepping is taken. When external forces become too great or frequent, even the most robust controllers cannot prevent a fall. This thesis describes methods for recovering from such falls and generating other rising motions for physically simulated characters. Sample results are shown in Figure 1.1.

1.1 Contributions

The main contributions of this work are:

- Controllers developed from a common set of simple components, for several tasks:
  - 3D biped sit-to-stand motions
  - 3D quadruped sit-to-stand, prone-to-stand, and lateral decubitus-to-stand motions
  - 2D biped prone-to-stand motions
- An optimization framework for increasing the motion speed of fall recovery motions
- A continuation framework for generalizing the controllers to different environments and characters

1.2 Organization

This thesis is organized as follows: Chapter 2 reviews related work from character animation and robotics. Chapter 3 describes the core components used in all controllers developed for this work. Using these components, the next three chapters describe independent results for three different situations. Chapter 4 describes a sit-to-stand controller for a 3D humanoid. Chapter 5 describes controllers for a variety of sitting and standing motions for a 3D quadruped. Chapter 6 describes a family of controllers for standing up from the prone position for a 2D character. These are then optimized and extended to decrease motion duration and support a wider range of initial conditions and body sizes. Finally, Chapter 7 concludes and provides directions for future work.

Chapter 2

Related Work

Related work on character control can be divided into four main areas: kinematic animation, physically based animation, robot control, and biomechanics. Though their applications are often quite different, they share many strategies and tools.

2.1 Kinematic Methods

For generating motions, most kinematic methods rely on either hand created motions, painstakingly created by talented artists, or motion capture data recorded from actors. Motion graphs can be used to define allowable sequences of transitions between individual motion clips [10]. Motion graphs have been extended to generate parameterized motions that are more flexible and less repetitive, and to blend kinematic motion with physics based collision responses, e.g. [2, 13]. Any generated motion, however, is based on a captured performance, so motions similar to each desired motion must be recorded. A complete survey of kinematic based animation methods is outside the scope of this thesis.

2.2 Physically Simulated Character Animation

In computer animation, much work has been focused on locomotion. Many of the controller components used in this work have been combined to generate robust, stylized walking motions for a characters of arbitrary size, as shown in Figure 2.1[3] . Online optimization based approaches have also been applied to locomotion, e.g. [5]. Offline optimization of control parameters can generalize locomotion controllers to different environments, e.g. rough terrain or slippery surfaces [22, 23].

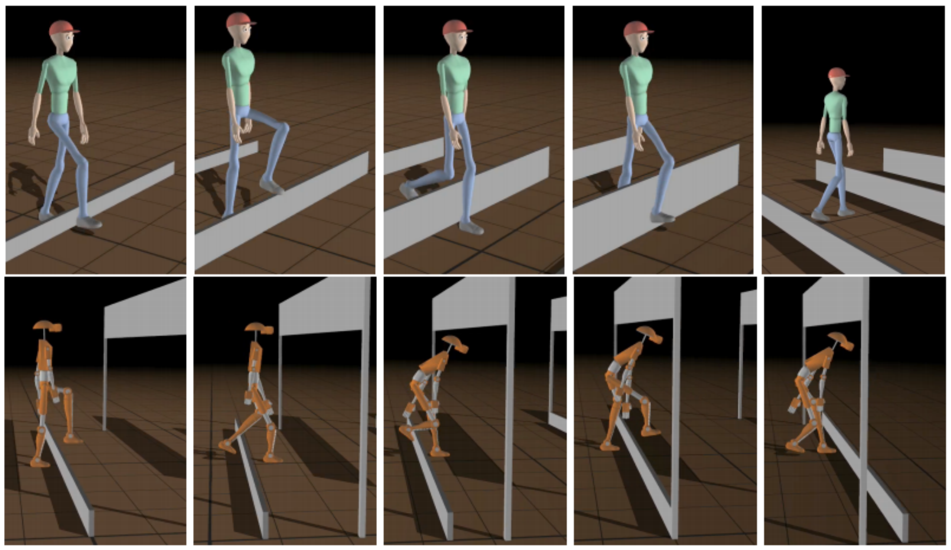


Figure 2.1: Example of flexible locomotion control applied to multiple characters [3]. Used with permission.

Some work has focused on controllers for contact rich motions including standing up. Lin and Huang [14] incorporate a dynamics simulation to link an arbitrary initial pose to a frame in a rising motion database, then transition to a kinematic motion. Faloutsos et al. [6] demonstrate moderate speed prone-to-stand and supine-to-stand motions, as well a a dynamic "kip" supine-to-stand motion. Their kip motion is shown in Figure 2.2. The controllers are based on pose tracking and timed state transitions, making it difficult to transfer controllers to new characters or environments.

Starting with kinematic data, Liu et al. [15] generate open loop dynamic controllers for contact rich motions including rolling and dynamic rising motions. This is an expensive offline process, however, and does not incorporate feedback into the generated controllers. Figure 2.3 shows a rolling motion retargeted to a physically simulated Asimo-like character.

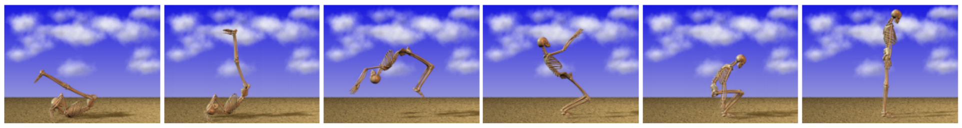


Figure 2.2: Example of dynamic supine-to-stand motion for a physics based character [6]. Used with permission.

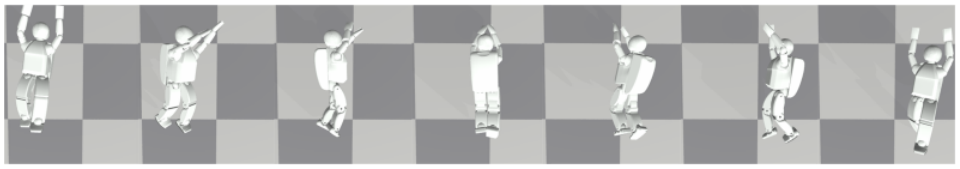


Figure 2.3: A kinematic rolling motion is retargeted to a physics based Asimo-like character [15]. Used with permission.

2.3 Robotics

The problem of standing up from a sitting, supine or prone position has been addressed in the robotics community by several researchers on a variety of hardware platforms. Kanehiro et al. successfully implemented prone-to-stand and supine-to-stand motion on the HRP-2P robot, which is 1.5 *m* tall and weighs 58 *kg*[9]. Similar controllers for a smaller 0.6 *m*, 2.3 *kg* RoboCup KidSize class humanoid robot have also been developed [21]. The motions, however are extremely slow, and require static stability throughout most of the motion.

Kuniyoshi et al. use an adult size humanoid robot and analyze the critical aspects of a highly dynamic getting up motion and use this information to find parameters to generate successful motions[12]. Transferring the motion from simulation to the robot was challenging and error prone, however, due to the dependence of the motion on difficult to simulate ground contact forces. Images of their robot performing this motion are shown in Figure 2.4. All of these previous works use control descriptions tied to a specific character and scenario, (e.g. a particular robot on flat ground), while the control descriptions described in this thesis are demonstrated to be applicable over a broader range of characters and environments.

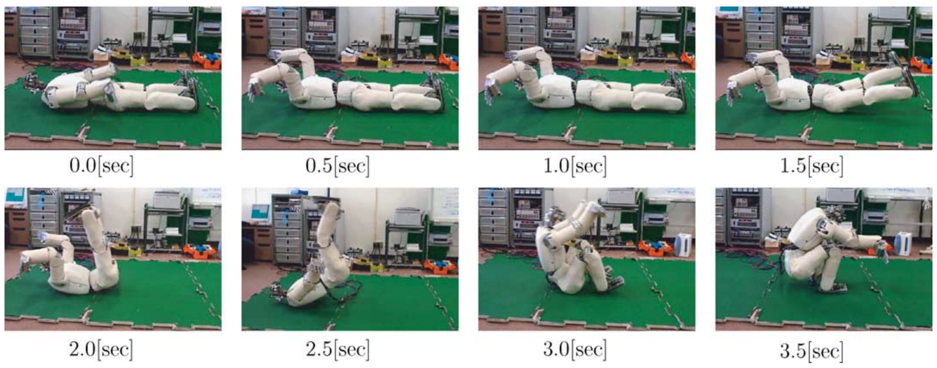


Figure 2.4: Dynamic supine-to-stand motion with the R. Daneel Study 1 robot[12]. Used with permission.

2.4 Biomechanics

Standing-up motions are well studied by the biomechanics community. Of particular importance to this work is the analysis of sit-to-stand motions. The dynamics and stability of these motions is well understood [18]. Muscle activity through the motion can be divided into three distinct phases: a forward lean of the upper body, an upward acceleration due to leg extension, and a deceleration phase [7, 11, 19]. The contact forces between the buttocks and feet are (indirectly) controlled by muscle activations to generate the forward and upward acceleration to stand. These forces are greatest just before seat-off, when the buttocks break contact with the seat, and the weight is supported entirely by the feet.

Chapter 3

Controller Components

The controllers developed in this thesis share several key components that have been shown to define a good vocabulary for designing controllers [3, 4]. At a high level, these consist of pose trajectories, root orientation correction, and virtual force trajectories. Pose tracking allows for approximate tracking of desired kinematic trajectories, while virtual force trajectories allow for the development of components that track in a cartesian reference frame. Root orientation corrections attempt to indirectly control the root orientation of the character because it cannot be controlled directly. Root orientation control modifies torques computed by the pose tracking control at the hip and shoulder joints. Torque contributions form virtual forces are simply summed to the output of pose tracking with root orientation correction. When combined as shown in Figure 3.1, these building blocks can be used to create flexible, robust controllers.

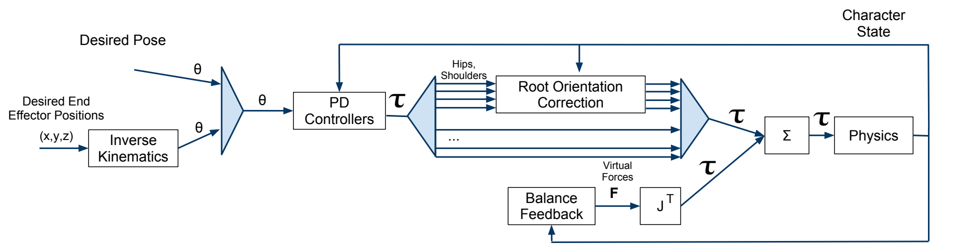


Figure 3.1: Overview of controller structure

3.1 Pose Tracking

The pose tracking controller takes as input the desired relative orientation of the bodies in each controlled joint and computes joint torques using proportional-derivative (PD) control. Each joint defines a parent and child link, and their relative orientation is used to compute the torque: $\tau = k_p * E_{angle} - k_d * \dot{E}_{angle}$ where E_{angle} is the deviation from the desired relative orientation between links. For one degree-of-freedom(DOF) joints (e.g. knees), the desired orientations are specified by a piecewise linear trajectory over time. For joints with more than one DOF, the desired orientation is represented by yaw, pitch, and roll trajectories or a single angle trajectory with a constant rotation axis. In both cases, the error terms are evaluated using quaternions to avoid numerical instabilities. It is sometimes convenient to control a link relative to the world coordinate frame or the character's heading instead of the parent link. In these cases the desired orientations are automatically converted to parent-relative orientations in the controller.

Inverse Kinematics

Inverse Kinematics (IK) is a method for determining the joint angles required to position the end effector of an articulated chain at a particular location in the world. For our humanoid characters, IK is used to specify the orientations of shoulders/hips and knees/elbows to position feet/hands in the world. This allows for more intuitive specification, as the trajectory of a hand or foot is simpler to reason about than the coupled rotation trajectories of hip and knee joints.

For humanoid characters, IK is used for the arms and legs. Both limbs are two link chains consisting of a three degree of freedom (DOF) ball and socket joint at the hip/shoulder, and a one DOF hinge joint at the knee/elbow. The two joint angles can be computed analytically to position the hand anywhere the arm can reach. To prevent numerical problems, if the end effector target is adjusted so it lies within the limb's reach. In two dimensions, there are two solutions, corresponding to which direction the elbow bends. For the arm, the solution chosen arbitrarily, while for the leg, it is chosen so that the knee bends in the anatomically correct direction. In three dimensions, there are an infinite number of solutions corresponding to the circle of rotations about the axis from the shoulder to the hand. The particular

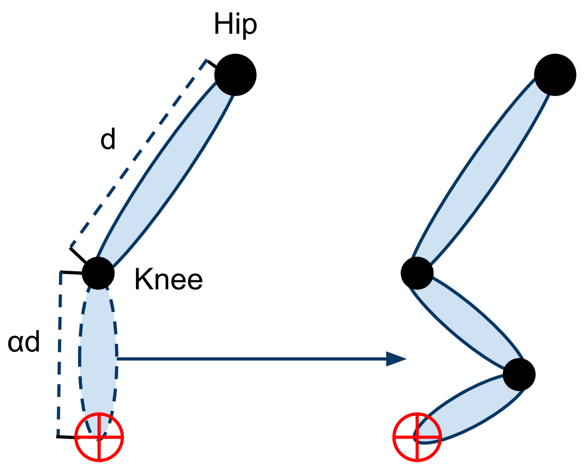


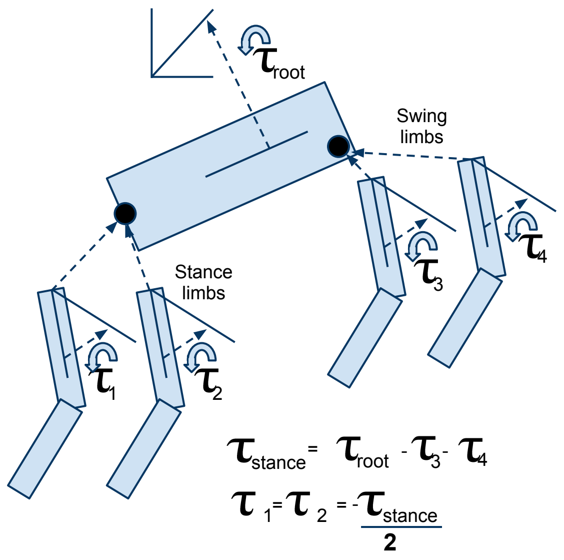
Figure 3.2: Two link IK is applied twice to solve for joint orientations of a quadruped limb.

solution for the legs is chosen such that the leg lies in a desired plane relative to the pelvis. For the arms, the solution is chosen such that the forearm is as vertical as possible, and the elbow is positioned away from the torso.

For the dog character, each leg has three links, so the above approach is still underconstrained. However, the shape of canine legs is used as a guide to add an additional constraint. The distance from the knee to end effector is specified to be some factor, α, times the femur length. With this constraint, two link IK can be applied twice to return the orientation of both leg joints as shown in Figure 3.2.

3.2 Root Orientation Correction

The characters examined in this work are all underactuated, meaning they cannot directly control their center of mass position or orientation. Global position and orientation must be controlled indirectly via control of the contact forces with the environment, which are discontinuous and potentially difficult to predict. Correcting for the orientation error in the pelvis with torques from the hips has been shown to adequately control the character's global orientation[17]. That approach is adapted for all controllers in this work.



τ_{root}
Swing limbs
Stance limbs
τ_1
τ_2
τ_3
τ_4
$\tau_{stance} = \tau_{root} - \tau_3 - \tau_4$
$\tau_1 = \tau_2 = -\frac{\tau_{stance}}{2}$

Figure 3.3: Root orientation correction with two swing and two stance limbs.

The pelvis is controlled via a PD controller, whose desired orientation is specified relative to the global coordinate frame, which computes the torques necessary to achieve the desired pelvis orientation, τ_{root}. The difference between this desired torque, and the torque from pose tracking of non-supporting limbs is computed as

$$\tau_{stance} = \tau_{root} - \sum_{\substack{swing \\ limbs}} \tau_{pose,i}$$

as shown in Figure 3.3. This torque replaces the PD torques computed for the stance hips, each hip exerting $-\tau_{stance}/n_{stance}$. In the prone-to-stand controller, more precise pose control is required at the stance hips, so the error torques are linearly blended with the torques computed by the pose controller rather than replacing them. Ideally, this corrective torque will induce a tangential contact force from the ground, applying a net torque to the character and correcting its global orientation. This feedback strategy works well for small perturbations, but interferes with pose tracking at the hips joints and can cause feet to slip when τ_{stance} is very different from the torques computed for the stance legs by pose control.

3.3 Virtual Forces

Virtual forces are a technique for generating coordinated joint torques along a chain of links that mimic the effect of an external force [1]. A virtual force acts from a **base link**, which is assumed to be stable, onto a **link of application**. The virtual force is the set of torques on the joints between the base and point of application generate at net force F on the link of application. For an infinitesimal displacement, the work done by an articulated chain is

$$W = F\Delta x = \tau\Delta q.$$

Forward kinematics relates Δx and Δq by the Jacobian,

$$\Delta x = J\Delta q.$$

Substituting and rearranging gives

$$\tau = J^T F.$$

In this thesis, virtual forces are used to model a character "pushing" the ground with an end effector, a technique which has been employed successfully for a variety of controllers and characters[3, 4, 16]. For these virtual forces, the base is the character's torso, and the point of application is the center of a hand or foot. Figure 3.4 shows a virtual force being applied at the foot using the pelvis as the base.

Virtual Forces are used to control a variety of aspects of the developed motions. As an example, Figure 3.5 shows a subset of the forces used for quadruped control. F_g is a gravity compensation virtual force, which counteracts gravity forces which would otherwise prevent PD tracking from actually reaching the desired pose. Virtual forces are added at the center of mass of certain links with the desired force $-mg$. The base link for these torques is one of the torso links (shoulders or pelvis). Other virtual forces regulate center of mass position in the coronal and sagittal planes (F_{cor} and F_{sag}). Finally, virtual forces are used to regulate leg frame height (F_h).

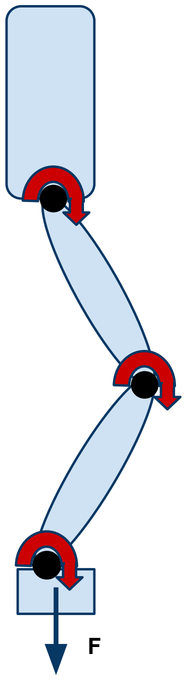


Figure 3.4: Example virtual force being applied to the foot with the pelvis as the base link. The torques mimic the application of the force, F, at the foot.

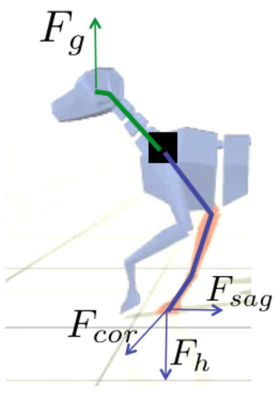


Figure 3.5: A subset of the feedback virtual forces used for quadruped controllers.

Table 3.1 outlines the components used in each chapter.

Table 3.1: Summary of controller components used in each chapter

Chapter	Pose Tracking		Root Orientation Correction	Virtual Forces	Optimization
	Direct	IK			
3D Sit to Stand	✓		✓	✓	
3D Quadruped Skills	✓	✓	✓	✓	
2D Prone to Stand		✓	✓	✓	✓

Chapter 4

Biped Sit-to-Stand Motions

A common human interaction with the environment is sitting down and standing up from chairs. This is an attractive situation to address because it is one of the simplest contact-rich motions that involves a support change and the motion has been well studied by the biomechanics community.

4.1 Character Description

The character is modeled as a 17-link articulated rigid body. The pelvis, torso, and head are modeled with 1 link each. The arms are modeled as three links each: humerus, radius/ulna, and hand. The legs are modeled as four links each: femur, tibia/fibula, foot, and toes. A schematic diagram of the character is shown in Figure 4.1.

4.2 Controller

The controller for this motion is implemented as a pose tracking controller with root orientation correction and a virtual force feedback loop. The joint trajectories are represented as a finite state machine with static target poses and timed transitions. The desired pose trajectory consists of the three phases described in the biomechanics literature. Once the sit to stand motion is triggered, forward momentum is generated by leaning forward at the hips and waist. Next, once the character is supported by its feet, the legs are extended by straightening the hips

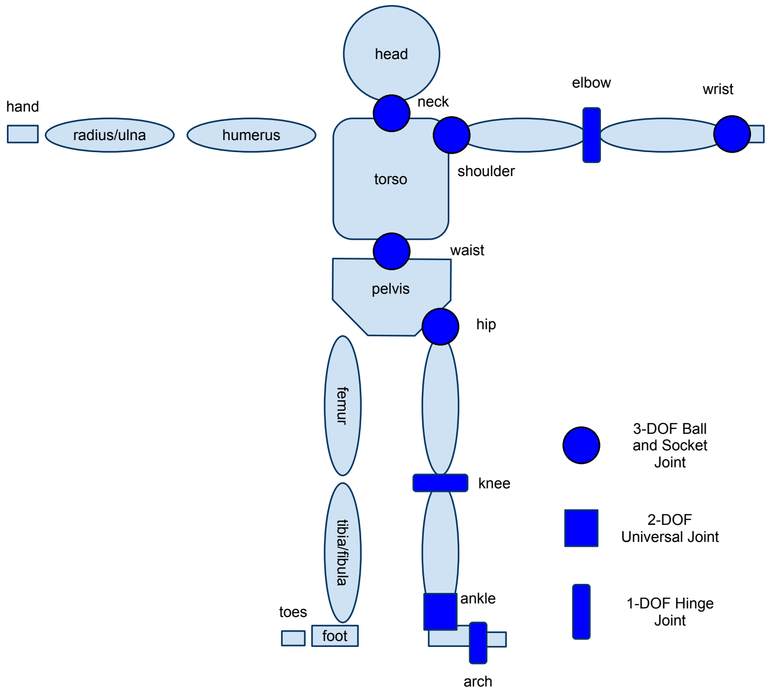


Figure 4.1: Schematic diagram of the biped character.

and knees, and the character accelerates upward. Finally, the character decelerates and the desired pose becomes a stationary standing pose. The ankle, shoulder and neck joint targets are held constant throughout the motion, while the wrists and arches are uncontrolled. To prevent the hands from colliding with the chair, the elbows are bent to 90° during the lean and lift phases. This sequence is illustrated in Figure 4.2, while the precise pose targets are described in Table 4.1. While the desired pose provides a coarse description of the desired motion, it is insufficient for dynamic control. It does not provide a means for balance correction.

To handle perturbations in the motion, root orientation correction is applied during the lift and standing phases, and a linear feedback system is added using virtual forces. Root orientation correction attempts to keep the pelvis completely vertical. To prevent falling to the side, the position of the planar projection of the **active center of mass** is controlled using virtual forces. The active center of mass

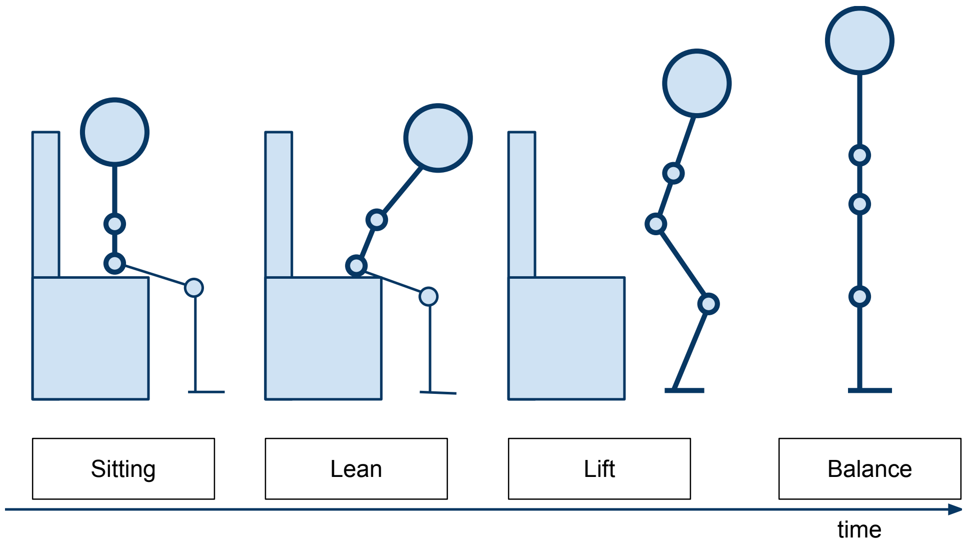


Figure 4.2: The phases of the sit to stand motion. Bolded limbs contribute to active center of mass computation.

Table 4.1: Control parameters for 3D biped sit-to-stand motion

Phase	Duration (*s*)	Desired Angle(°)				Root Orientation Correction	Virtual Forces
		Torso	Hip	Knee	Elbow		
Sitting	-	0	90	limp	limp		coronal
Lean Forward	0.9	90	90	90	limp		coronal
Lift Upward	0.5	0	0	10	90	✓	coronal, sagittal
Stand Balanced	-	0	0	10	limp	✓	coronal, sagittal

is defined as the center of mass of all bodies above the supporting limbs,

$$\vec{x}_{com} = \frac{\sum\limits_{\substack{supported\\bodies}} \vec{x}_i m_i}{\sum\limits_{\substack{supported\\bodies}} m_i}.$$

For example, when seated, the active center of mass is the center of mass of the torso, head, and arms, while when standing, it includes all links of the character. Virtual forces, using the supporting link as the base joint, are applied to the torso using independent PD controllers for the sagittal and coronal planes. In the coronal plane, the desired force is computed as

$$\vec{F} = k_p(\vec{x}_{com} - \vec{x}_{des})_{cor} - k_d\vec{v}_{com,cor},$$

where the $\vec{u}_{cor}$ is the horizontal projection of $\vec{u}$ in the coronal plane and *com* refers to the active center of mass. The force in the sagittal plane is computed analogously. The desired position position is the center of support. When sitting, this is the center of the pelvis. When standing, this is the midpoint of the feet. Without this control, the character may fall sideways out of the chair, and cannot maintain balance while standing.

4.3 Results

Open Dynamics Engine [20], a 3D articulated rigid body simulator, is used for simulation. Collision primitives are a combination of capsules, spheres, and boxes, as shown in Figure 4.3. The simulation is run at 2.0 *KHz* using a ground friction coefficient of 0.8.

This control representation was used to create a functional sit to stand controller which is capable of reliably standing from a chair. Images from the motion sequences are shown in Figure 4.4. Without modification, the controller works with chairs ranging from 45 *cm* to 65 *cm*, indicating that the control representation used was abstract enough to work effectively over a range of scenarios. When the chair height is below the effective range, the character does not generate enough momentum to stand, and falls backward into the chair. When the chair height is

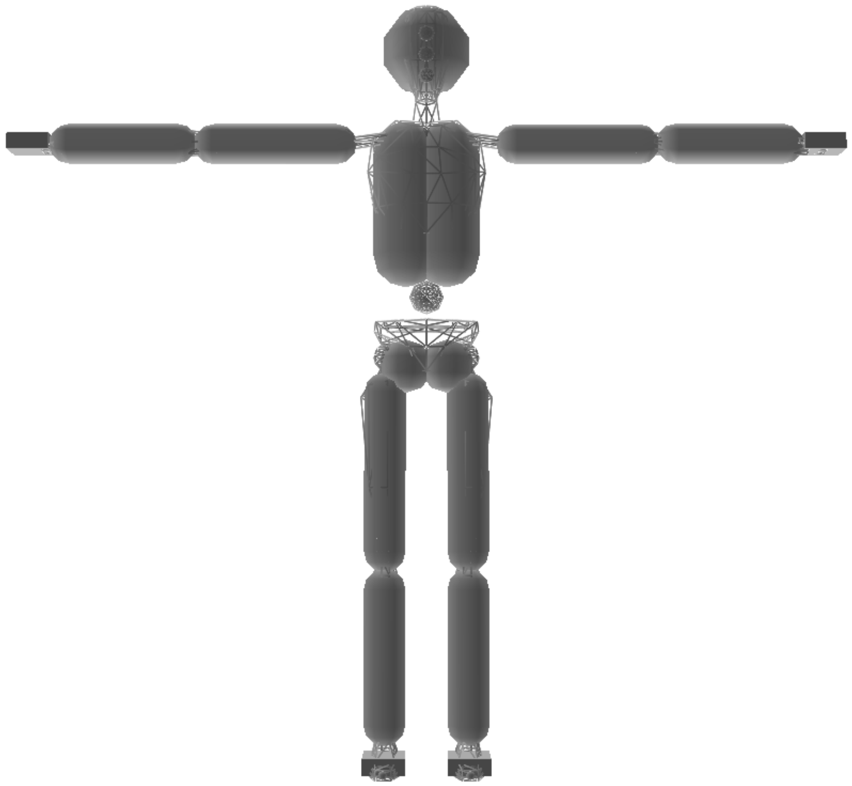
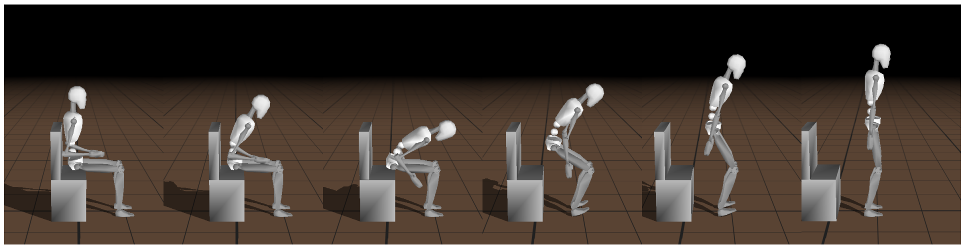
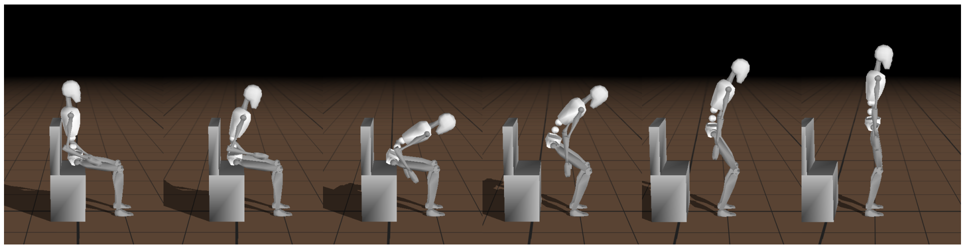


Figure 4.3: Collision primitives for the 3D biped model.

too great, the character does not balance properly sitting in the chair. The collision primitives of the pelvis are spheres, and without the feet providing support, the character is unstable, and tends to roll forward or backward, leading to a fall. More accurate collision handling, a more complex controller, or manual parameter tuning could likely cope with these failure cases.


(a)


(b)

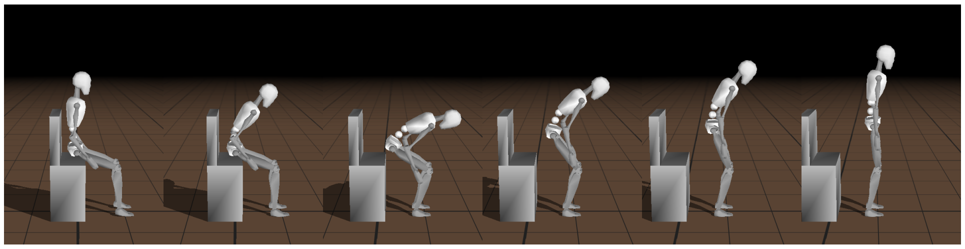

(c)

Figure 4.4: Sit to stand controller used unmodified on chair heights of 45, 50, 60 *cm*. The duration of all three motions is approximately 2 *s*.

Chapter 5

Quadruped Controllers

Being able to sit, lie down and get back up are important skills necessary to simulate realistic common interactions with dogs. As such, the controllers described in this section are designed to augment a skillset that includes a wide range of locomotion styles, hops and leaps for a simulated quadruped. Using the the components described in Chapter 3, controllers are designed for sit-to-stand, lie-to-stand, and lateral decubitus to stand motions. The simulations are again performed using Open Dynamics Engine with the quadruped model shown in Figure 5.1.

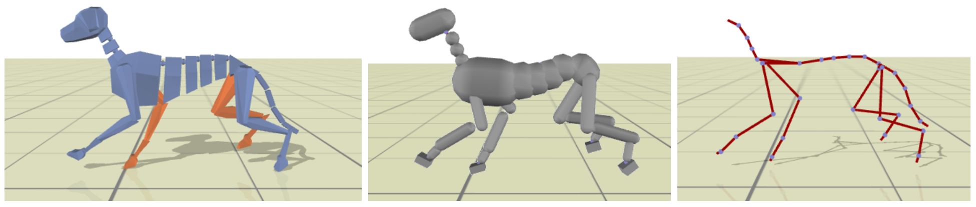

(a) Display mesh **(b)** Collision primitives **(c)** Joint hierarchy

Figure 5.1: Quadruped display and collision geometry.

5.1 Approach

5.1.1 Controller Structure

The controllers developed in this chapter are part of a large suite of controllers developed for the control of an agile simulated dog [4]. Many features of the controller are not used in this thesis, so we will describe only the pertinent aspects and refer the reader to Coros et al. [4] for the full details on the control of other capabilities, including a variety of gaits and leaping. The controller structure is based on a **gait graph**, which describes the desired height and orientation of the character's leg frames over time as well as a description of when each leg should be swinging or supporting the character and several other parameters of stepping. Based on the characters state, the controller translates a gait graph into a desired pose and a set of virtual forces at each timestep. The gait graph for one phase of the stand-to-lie motion is shown in Figure 5.2, indicating step timings and hip and shoulder height trajectories.

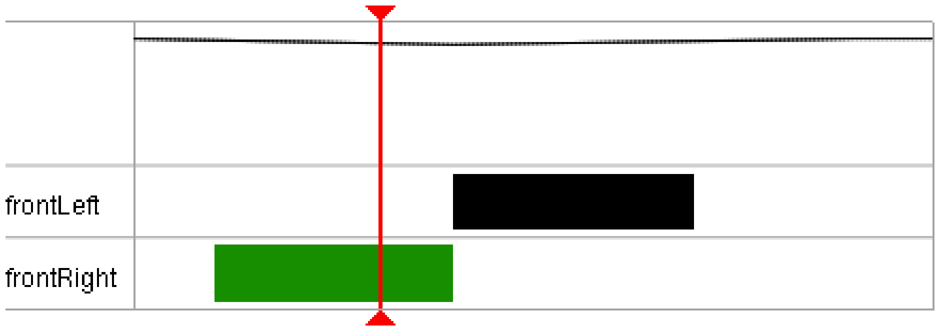


Figure 5.2: The gait graph for one phase of the stand-to-lie motion. The bars represent step timings, while the lines above represent hip and shoulder height trajectories.

The controller views the quadruped as two leg frames connected by a flexible spine, as shown in Figure 5.3. The front leg frame consists of the front legs and the spine link they are attached to, while the rear leg frame consists of the rear legs and pelvis. Using pose tracking, root orientation correction and virtual forces, and planted legs drive the spine links toward their desired orientations and heights. The spine links in between the pelvis and shoulders are also actuated to maintain the proper relative orientation between the pelvis and shoulders.

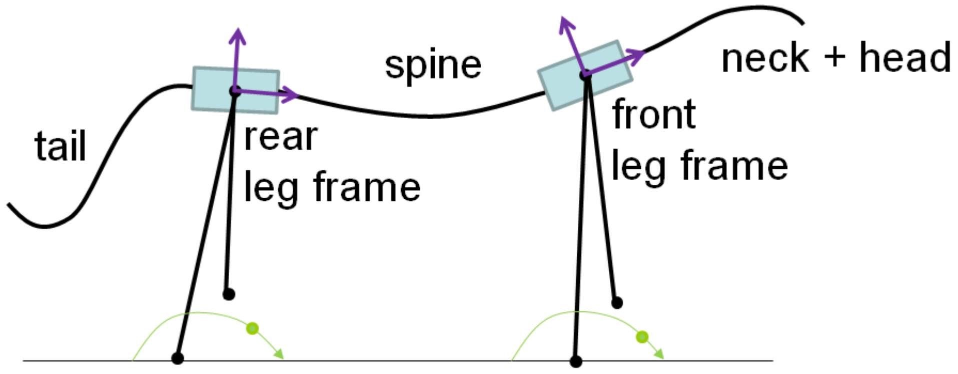


Figure 5.3: Abstract representation of the quadruped model as two leg frames connected by a flexible spine [4]. Used with permission.

Using this framework, the controllers described here were developed by specifying step timings for the legs and hip and shoulder height trajectories. Additional parameters such as shoulder orientation are also adjusted to make the motions more natural and robust.

5.1.2 Sitting

The sitting motion is composed of two components: stand-to-sit, and sit-to-stand controllers. The stand-to-sit controller has two phases with a timed transition between them. The gait graphs for each phase are:

1. Step forward (0.9 *s*)
   - Step rear feet forward, one at a time to 15 *cm* behind front feet, 6 *cm* wider than hips. Each step takes 0.3 *s*.
   - Decrease desired hip height linearly from 44 *cm* to 14 *cm* over 0.7 *s*.
2. Rest buttocks (0.3 *s*)
   - Decrease desired hip height linearly to 9 *cm* over 0.3 *s*.

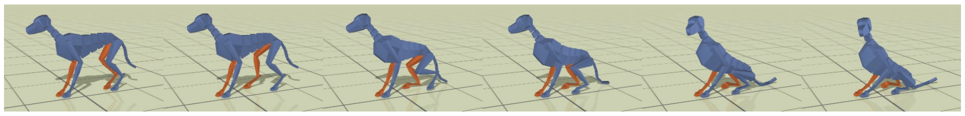


Figure 5.4: Stand-to-sit motion for the simulated quadruped.

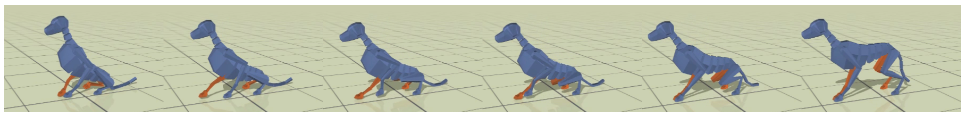


Figure 5.5: Sit-to-stand motion for the simulated quadruped.

The stand-to-sit motion is shown in Figure 5.4.

The sit-to-stand component steps forward with the front feet and lifts hips to standing height. Since the center of mass is so far back at the beginning of the motion, an extra virtual force and dipping motion at the shoulders is required to generate forward momentum. The gait graph for the sit-to-stand motion is:

Sit-to-stand (0.9 *s*)

- Step front feet forward to initial position relative to the rear feet. Each step takes 0.3 *s*.
- Lower desired shoulder height by 2 *cm*. Increase desired shoulder frame pitch in the sagittal plane linearly from 0° to 15° over 0.3 *s*.
- Apply a horizontal virtual force in the sagittal plane increasing linearly from 0 *N* to 100 *N* over 0.2 *s*, then decreasing to 0 *N* over 0.6 *s*.
- Increase desired hip height from 9 *cm* to 44 *cm* linearly over 0.4 *s*.

The sit-to-stand motion is shown in Figure 5.5.

To develop these controllers, we determine approximate step lengths and timings from videos of dogs performing these motions. For stand-to-sit, specifying a gait pattern with these stepping patterns is sufficient to create robust sitting motions. Because the controller automatically computes balance feedback, this process is similar to authoring a motion using keyframes. The beginning and end of each step must be specified as well as hip and shoulder height positions at key times. When the balance feedback fails, as it does during the sit-to-stand motion,

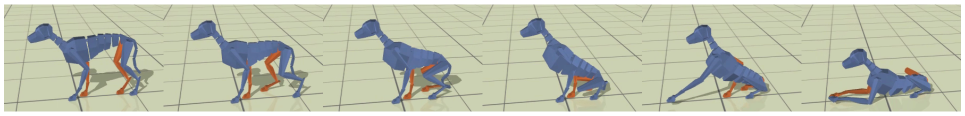


Figure 5.6: Stand-to-prone motion for the simulated quadruped.

this naive approach is insufficient. For the sit-to-stand motion, the naive gait graph results in the dog falling backward because the center of mass is too far behind the rear feet. Dipping the shoulders and applying an extra virtual force provide enough forward momentum to return the dog to standing. These situations require an extra manual balance specification on top of the keyframe-like specification.

5.1.3 Lying Down

The lying controller consists of a stand-to-prone and a prone-to-stand component. The stand-to-prone controller has three phases with timed transitions between them. The first two are identical to the stand-to-sit controller. From the sitting state, the following gait graph is used to transition to the prone position:

Sit-to-prone (0.8 *s*)

- Step front feet forward to 45 *cm* in front of and 4 *cm* to the side of the shoulders. Each step takes 0.3 *s*.
- While stepping the second foot, twist the shoulders in the coronal plane, lifting the shoulder of the stepping foot. Twist from $0°$ to $30°$ over 0.1 *s*, then return to level orientation over 0.2 *s*.
- While stepping, shift the desired center of mass position away from the swing leg by 5 *cm* in the coronal plane.
- Decrease desired shoulder height from 44 *cm* to 22 *cm* over 0.4 *s*.

The stand-to-prone motion is shown in Figure 5.6.

Once prone, the relative positioning of the feet is approximately the same as for quiescent standing. Therefore, the prone to stand component does not use any stepping, but just returns the desired hip and shoulder heights and orientation to their standing values. The gait graph is as follows:

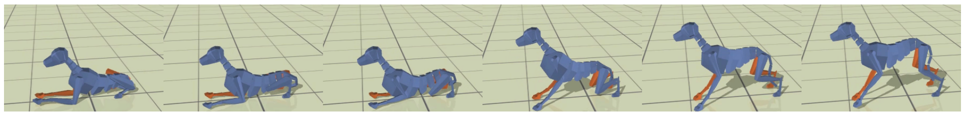


Figure 5.7: Prone-to-stand motion for the simulated quadruped.

Prone-to-stand (1.0 *s*)

- Increase desired shoulder height from 22 *cm* to 44 *cm* over 0.9 *s*.
- Increase desired hip height from 14 *cm* to 44 *cm* over 0.9 *s*.
- Increase the desired shoulder frame pitch angle in the sagittal plane from 0° to 15° over 0.3 *s*. Then, return is to horizontal over 0.7 *s*.

The prone-to-stand motion is shown in Figure 5.7.

Development of these controllers is analogous to the development of the sitting controllers. When stepping forward with the front legs, the character tends to tip over while stepping forward with the second front leg. To combat this, we add a shift of the coronal component of the desired center of mass and a twist of the coronal component of the desired shoulders orientation. These modifications lift the shoulder of the swinging leg enough to move the foot to the desired position. Like in the sitting controller, extra forward momentum is required to return to standing, so the same shoulder dipping strategy is employed.

5.1.4 Lateral Decubitus-to-Stand

To recover from unexpected falls, a get-up controller that enables the dog to get up from a lateral decubitus position, i.e., lying on one side, is also developed. When the dog has its feet under it, the trotting controller is extremely robust, and can recover gracefully from a wide range of scenarios. The strategy for this motion is therefore to position the feet under the character so that the trotting controller can be engaged.

Development of this controller is extremely difficult due to several factors. First, modeling a dog as a set of rigid capsules is not accurate in this case, causing bizarre simulation artifacts. Motions that a real dog is capable of are difficult or impossible to control using our simplified character model. Second, almost all of the feedback loops in the controller used for other quadruped motions expect the feet to be well planted below the character. This assumption is clearly not met in this case, and requires a different approach from the other motions. The controller for this motion is therefore heavily modified, and relies mainly on explicit pose tracking and hand specified virtual forces.

Much of the control comes from control of the character's spine. The desired orientations of the shoulders and pelvis are specified in terms of euler angle trajectories over time. The relative orientation between them is equal to the quaternion, $q_{rel} = q_{pelvis}^{-1} q_{shoulders}$. This quaternion can also be represented by a rotation α about a vector $\vec{v}$. Using this representation, each spine joint is driven by a PD controller to the orientation defined by a rotation α/n about $\vec{v}$ where n is the number of spine joints (5 for this character).

The motion is divided into two phases. First, the controller attempts to roll the character into a prone position. The hips and shoulder desired orientations begin horizontally while the legs are posed using IK such that the feet are as close as possible to the hip and shoulder joints. Next, the desired orientation of the shoulders is adjusted to rotate from $0°$ to $100°$ in the transverse plane, lifting the shoulders off the ground. Next, the shoulders are twisted $50°$ in the coronal plane toward the feet. Next, the desired orientation of the hips is linearly interpolated from $90°$ to $0°$ in the coronal plane. At this point, both the shoulders and hips are oriented correctly, the character's feet are underneath it, and the spine is curved to one side. The front feet are then controlled via IK so they make contact with the ground directly under the shoulders. Once both front feet are planted and the shoulders are approximately level, the controller proceeds to the second phase.

The second phase of the motion seeks to raise the hips and shoulders off the ground so that the trotting controller can be engaged. Since the hips and shoulders may not be completely vertical at this point, their desired orientations are interpolated from their orientations at the beginning of this phase to their fully upright position. For the shoulders, this transition takes $0.2\ s$, while for the hips, it takes $0.5\ s$. To straighten the spine, the desired shoulder orientation in the transverse plane is modified until the shoulders and hips point in the same direction. This occurs over $0.5\ s$.

Now that the front feet are planted, they can be used to support the shoulders.

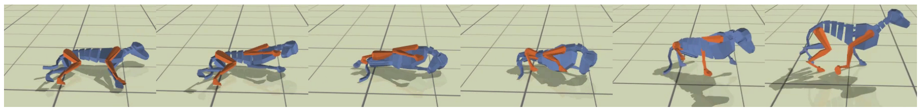


Figure 5.8: Fall recovery controller transition from lateral decubitus position to standing.

The desired shoulder and hip heights are increased linearly over 0.3 *s* . As this happens, the IK targets for the feet are adjusted so that the they are directly beneath the shoulders, and their effective lengths are equal to the desired height. The virtual force height correction term used in the controller is also linearly blended back in over 0.2 *s*. Finally, to generate forward momentum, virtual force is applied to the character center of mass in the sagittal plane. These forces are linearly increased to their maximum value over 0.2 *s*, and the remain constant until the trotting controller is engaged. The front feet provide a maximum of 350 *N* forward, and 100 *N* upward, while the rear feet apply 150 *N* forward and 50 *N* upward. Once the shoulders are above 25 *cm*, the character can transition to trotting returns to its steady state mode after several cycles.

The lateral decibutus-to-stand motion is shown in Figure 5.8.

5.2 Results

All motions in this chapter were authored by hand with manual tuning of control parameters. Due to the abstraction of the gait pattern, this was straightforward for the sitting and lying controllers. Developing the lateral decibutus-to-stand controller, however, was not straightforward, and many different strategies were attempted before the approach described above.

The sitting and lying controllers are extremely robust to disturbances. For instance, both controllers worked effectively while standing on or near an 8cm tall block as shown in Figure 5.9. The lateral decibutus-to-stand controller is effective for most falls on flat ground because before starting to rise, the character is driven to a standard pose. This controller is the least robust, however, often generating unnatural motions when transitioning to the trotting controller. This is not unexpected, as the motion is largely open loop, so disturbances are not corrected

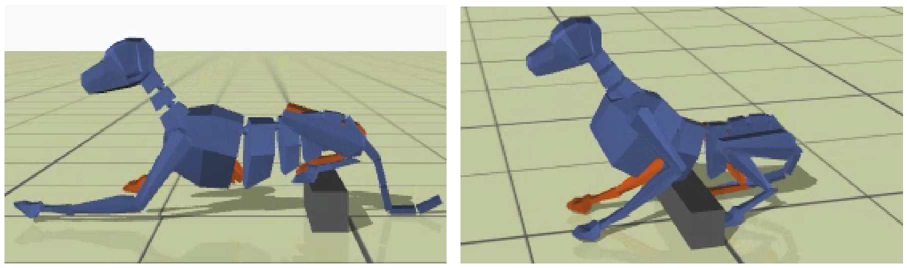


Figure 5.9: Generated controllers are robust to environmental disturbances like this 8cm block.

quickly. Feedback strategies that are effective even without reliable foot placement are a direction of future work.

Chapter 6

Biped Prone-to-Stand Motions

The final scenario we explore is a character rising from the prone position to a standing pose. In order to simplify the implementation and focus on the core features of the problem, this example is implemented in 2D. When falling, human characters often fall landing in a prone position, or roll to a prone position before standing up. Hence, this controller is a valuable asset for a more general fall recovery controller. This scenario presents a situation in which multiple support changes are required, and when performed by humans, produces motions which are not always statically stable. As a result, this scenario permits a wide range of complex, dynamic solutions.

The scenario tested consists of a 2D, 9-link character lying prone on flat ground and attempting to stand up onto two feet. The controller uses a support graph, similar to the gait graph in Chapter 5, to specify support change timings and the overall desired pose. An initial statically stable support graph for flat ground is used as a seed point for an optimization procedure which increases the speed of the motions. The faster motions output are then used in a continuation procedure to create a family of motions suitable for sloped ground and varying character size.

This controller is implemented using the JBox2D [8] 2D simulator. The character is modeled as a 9-link articulated rigid body with one torso link, two two-link legs, and two two-link arms. It is approximately 1.1 *m* tall and weighs 25 *kg* with most mass concentrated in the torso.

6.1 Controller Representation

The core representation of the controllers used in these experiments is an adaptation of the gait graph structure used in Chapter 5, which we refer to as a **support graph**[4]. The motion is described as a set of limb patterns and trajectories for the desired hip and shoulder heights during the motion as a function of the controller's phase. Limb patterns encode when a given limb is in stance mode, swing mode, or idle (limp) mode, and describe where the limb should be placed when stepping. They also describe whether the foot or the knee should be used as the end effector during a given period.

The core difference between gait graphs and support graphs is that gait graphs describe periodic motions, while support graphs describe single shot motions. In practice, this means that our controllers can't rely on certain types of feedback provided by gait graph based controllers, e.g. foot placements during the next step cycle. Instead, since the motions require lower momentum than fast locomotion, our controllers can pause phase advancement until it is safe to proceed. In this context, safe means that stance limbs are in contact with the ground, and swing limbs are not supporting weight.

The controller using the support graph uses the components described in Chapter 3. The limbs (arms and legs) are controlled using IK pose tracking with k_p and k_d equal to 35 $N\ m/rad$ and 8 $N\ m/rad$ respectively for all joints. When in stance mode, the end effector positions are chosen such that the attached body joint (hips or shoulders) is at its desired height, as evaluated from its height trajectory. When in swing mode, the desired end effector position is linearly interpolated horizontally from its takeoff position to its desired landing position, while its height is controlled by a separate trajectory. The landing position is specified in the limb's limb pattern as a horizontal offset from its parent joint (hip or shoulder). After a swing limb passes through half of its swing phase, its target landing position is recomputed to account for motion of the hip or shoulder joint while the limb is moving. It is common in prone-to-stand motions to use the knees as a support while rising. To exploit this, limb patterns track when limbs should be in kneel mode, using the knee as the end effector. The desired lower leg orientation is horizontal while in kneel mode. Since the distance from hip to knee is fixed, there is only one degree of freedom in the knee position, so the controller can guide the hip position in either the horizontal direction or the vertical direction, but not both. Our controller uses the desired hip height to determine the target angle at the hips, ignoring the desired horizontal hip position.

Root orientation correction is also used to ensure proper torso orientation. When arms and legs are both grounded, the root error is small, as the limb poses orient the root approximately correctly. However, once the arms stop supporting the character, root orientation correction must be used to maintain control. Because the pose tracking component is vital to height tracking, we do not replace the stance hip torques with the root error torque like described in section 3.2. Instead the root error torque is computed as

$$\tau_{error} = \tau_{root} - \sum_{\substack{hips,\\ shoulders}} \tau_i.$$

This error torque is clamped, $\tau_{clamped} \in [-\tau_{max}, \tau_{max}]$. Each stance hip torque is modified

$$\tau_{hip} \leftarrow \tau_{hip} - \tau_{clamped}/n_{stance}.$$

This approach balances the competing goals of regulating torso position and torso orientation.

Virtual forces are also used to provide feedback to the motion. To avoid large PD tracking gains, gravity compensation virtual forces are applied to each link of swing limbs. To control hip and shoulder height, a virtual force, computed via PD control, is added to planted limbs to push the hips or shoulders to the desired height. Finally, a corrective virtual force acts on the torso to drive its center to a weighted sum of the center of the lower limb for planted all planted limbs,

$$x_{des} = \frac{\sum_{\substack{planted\\ limbs}} w_i x_{lower,i}}{\sum_{\substack{planted\\ limbs}} w_i}$$

as shown in Figure 6.1. Initially all supporting limbs are weighted equally, but to avoid discontinuities, limbs that will be lifted soon have reduced weights. Limb

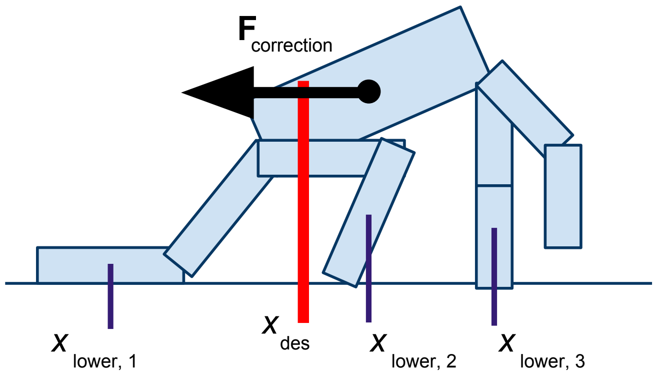


Figure 6.1: Sagittal corrective force to maintain balance.

weights are linearly interpolated from 1 to 0 over the 0.5 *s* before their expected lift time. The corrective force is distributed between the planted limbs according to these weights. Additionally, the desired vertical position of the lifting limb's end effector is increased from 0 *cm* to 4 *cm* over the 0.5 *s* before lifting. These modifications help ensure that limbs are not supporting weight when they are expected to lift.

This structure allows for intuitive design of controllers for relatively slow motions, however, faster motions are often desired. To automatically speed up slow controllers, we add a new structure, which we have named a time warp which controls the rate at which the support graph phase advances. It provides a one to one mapping of time to phase. Initially, time and phase are equal, however through the optimization procedure described below, this is adjusted so that the phase can advance faster or slower than time at during the motion. The time warp is represented as a piecewise linear trajectory of phase velocity, the time derivative of phase. This creates a piecewise quadratic mapping from time to phase. This representation allows motions to be sped up while preserving relative occurrences of events in the support graph.

Initial Support Graph

Using the support graph framework, we manually designed a controller for the prone-to-stand motion. The character begins lying with its arms stretched directly

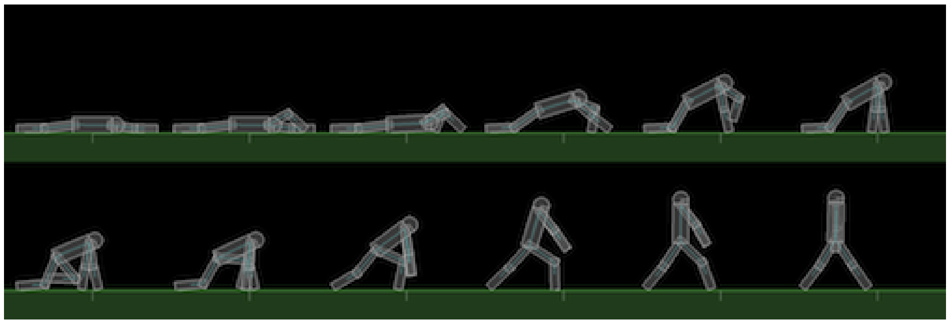


Figure 6.2: Selected frames from prone to stand sequence which requires 13.5 *s* to perform.

ahead. First, the hands are planted, one at a time 10 *cm* in front of the shoulders, each requiring 1 *s*. The legs transition from idle mode to kneeling stance mode, supporting the character with the knees. Using the hands and knees, the desired hip and shoulder heights are increased to 35 *cm* and 25 *cm* respectively. Then, the arms are lifted one at a time, and placed directly under the current shoulder position. Again, each step takes 1 *s*. Next, the character's right leg is lifted and planted 30 *cm* in front of the hip, now using its foot as the end effector. Finally, the left leg is also transitioned to standing mode, supporting the character with its foot. The desired hips and shoulder positions are increased to standing height while the arms transition into idle mode. The complete motion requires 13.5 *s* to achieve standing height. Selected frames from the motion are shown in Figure 6.2.

6.2 Optimization

Starting with initial hand designed support graphs, we use an optimization procedure to improve with respect to a given cost function (e.g. motion duration) and to adapt the controller to new terrains and characters. We employ a greedy stochastic search technique in the space of control parameter of a particular support graph. Because the parameterization is abstract and feedback loops are built into the controller, the simple optimization strategy works well.

The parameterization consists of the timings of all limb status changes, limb placement horizontal offsets, hip and shoulder height trajectory knot times and values (7 of each), and time warp control point values(10 total). The number and type of limb status changes is not changed during optimization. In total, there are 67 parameters modified during optimization. All optimizations begin with a single seed parameter vector, either hand designed or output from a previous optimization. At each iteration, several evaluations are performed (in practice, 16). Each evaluation consists of the character performing the motion described by the support graph with the given parameters, and returns a single cost value. Computing the cost value varies by the optimization type, and the cost functions used are described below.

Before each evaluation, a random subset of the parameters are perturbed with gaussian noise. The magnitude of noise for each parameter type is chosen by hand (e.g. time values all share $\sigma = 0.01\ s$). In practice we found them simple to choose due to the intuitive nature of the parameterization. After the evaluations, if one of the evaluations yields a lower cost than previously seen, it replaces the current seed. Because the parameterization used to represent the controllers is very abstract, the parameter space is smooth enough that a greedy, stochastic algorithm can be used. The optimization procedure is outlined in Algorithm 1.

6.2.1 Static-to-Dynamic Optimization

The first optimization seeks to increase the speed of the rising motion. The cost of an evaluation is defined as the time the character's shoulders are above a threshold height of 80 *cm*. To ensure that the character is still standing at the end, the shoulders are checked again several seconds after the expected completion time. We penalize motions if the shoulders are not above the threshold at this time.

Using this cost metric, the stochastic optimization procedure was able to shorten the motion from the original 13.5 *s* to 1.2 *s*. A comparison of the original and an intermediate optimized support graph is shown in Fig. 6.3. The optimization significantly increases the time warp phase velocity, accounting for most of the motion speedup. The four bars in the top section describe when each limb is in swing, stance, and idle mode. The red and blue curves indicate the desired hip and shoulder height trajectories. The vertical white line represents the current phase, while the vertical gray line shows the current time. The bottom section plots the phase velocity where the horizontal lines indicate 0 (phase stopped) and 1 (phase

```
**input**  : Parameter vector *seed*
**output**: Parameter vector optimized for cost
$n_{evaluations} \leftarrow 16$
bestCost $\leftarrow \infty$
parameters $\leftarrow$ Array of parameter vectors size $n_{evaluations}$
costs $\leftarrow$ Array of floats size $n_{evaluations}$
while not finished do
    for $i \in n_{evaluations}$ do
        parameters[i] $\leftarrow$ perturb(seed)
        cost[i] $\leftarrow$ evaluate(parameters[i])
    end
    bestIndex = argmin(cost)
    if cost[i] < bestCost then
        seed = parameters[i]
    end
end
return seed

function perturb(vec : Parameter vector) $\rightarrow$ Parameter vector
begin
    Constant Sigmas = Array of floats size of vec. Contains $\sigma$ for each
    index of vec.
    $n_{perturbations} \leftarrow$ integer(sampleGaussian($\mu$ = 0.2, $\sigma$ = 0.15)* len(vec))
    for $n_{perturbations}$ randomly selected indeces, i, of seed do
        vec[i] $\leftarrow$ vec[i] + sampleGaussian($\mu$ = 0, $\sigma$ = Sigmas[i])
    end
    return vec
end
```

Algorithm 1: Optimization procedure

and time are equal).

While most of the generated motions are reasonable, the fastest motions generated are quite unnatural and include violent collisions with the ground. Returning an infinite cost for evaluations with extreme contact forces would help mitigate these cases. Also, while the joints to have maximum torque limits, adding a torque minimization term to the optimization may also produce smoother motions at the expense of raw speed. The optimization increases phase velocity significantly to speed up the motion. A plot of motion completion time at each iteration is shown

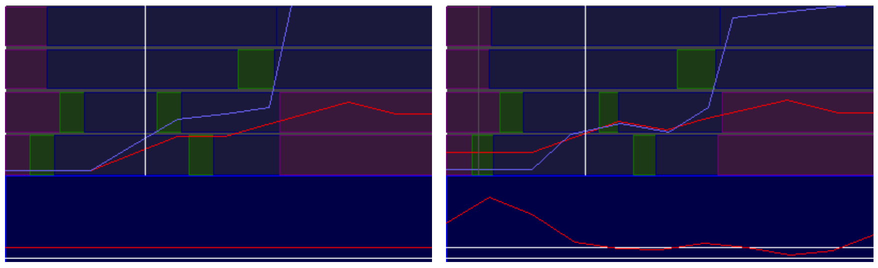

(a) (b)

Figure 6.3: Comparison of support graphs before and after time optimization.

in Figure 6.4.

The optimization may encounter local minima in the cost function, and require hundreds or thousands of evaluations to find an improved set of parameters. To guide the optimization away from these minima, after 80 evaluations with no improvement, the magnitude of noise is increased slightly every 16 evaluations until an improvement is found. A plot of the number of evaluations required per parameter improvement for one optimization run is shown in Figure 6.5.

6.2.2 Generalization for Terrain Slope

In order to generalize our controllers, a continuation optimization technique is used, gradually varying the ground slope. At each ground slope, we seek the most robust parameter vector, which we approximate by evaluating how many nearby controllers are also successful. The evaluation function is described in Algorithm 2.

Using this cost function, the optimization procedure is run until one parameter vector has an 80% or higher success rate. Then, the ground slope is increased by 1°, and the best cost is reset to infinity. Using a time optimized set of parameters as the initial seed, the procedure output controllers capable of getting up from terrains ranging from -12° to +17°. Frames from the uphill and downhill optimized motions are shown in Figure 6.6 and Figure 6.7, respectively. The initial seed, an output from a time optimization, does not meet the 80% threshold, but the optimization quickly improves robustness for the flat ground case until almost all perturbations are successful. For shallow slopes, few evaluations are necessary. For the steepest

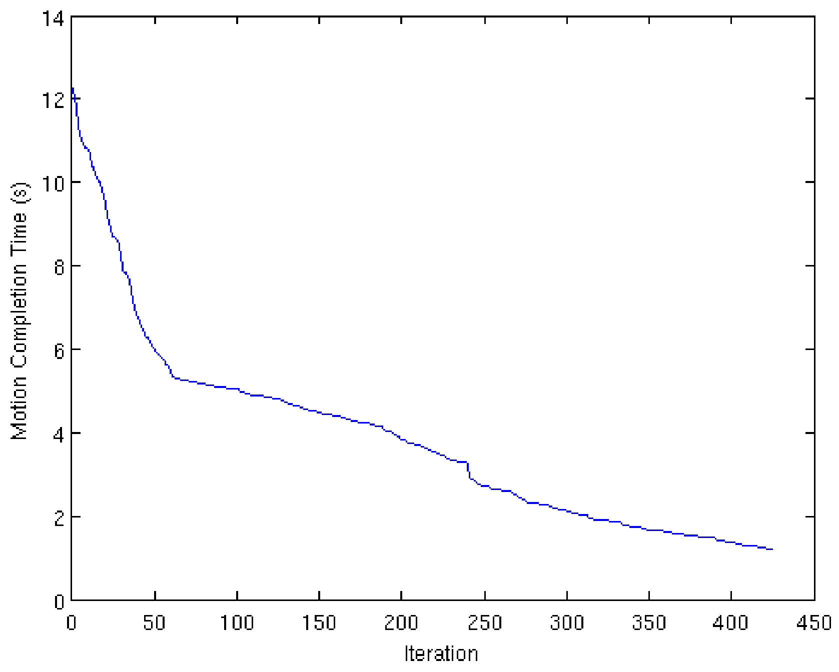


Figure 6.4: Prone to stand motion completion time compared to optimization iteration.

```
function evaluate($vec$ : $Parameter\ vector$) $\rightarrow \mathbb{R}$
begin
    $n_{perturbations} \leftarrow 16$
    $n_{successes} \leftarrow 0$
    for $i \in n_{perturbations}$ do
        if $performMotion(perturb(vec)) == success$ then
            $n_{successes} \leftarrow n_{successes} + 1$
        end
    end
    return $-n_{successes}/n_{perturbations}$
end
```

Algorithm 2: Cost function evaluation for continuation methods.

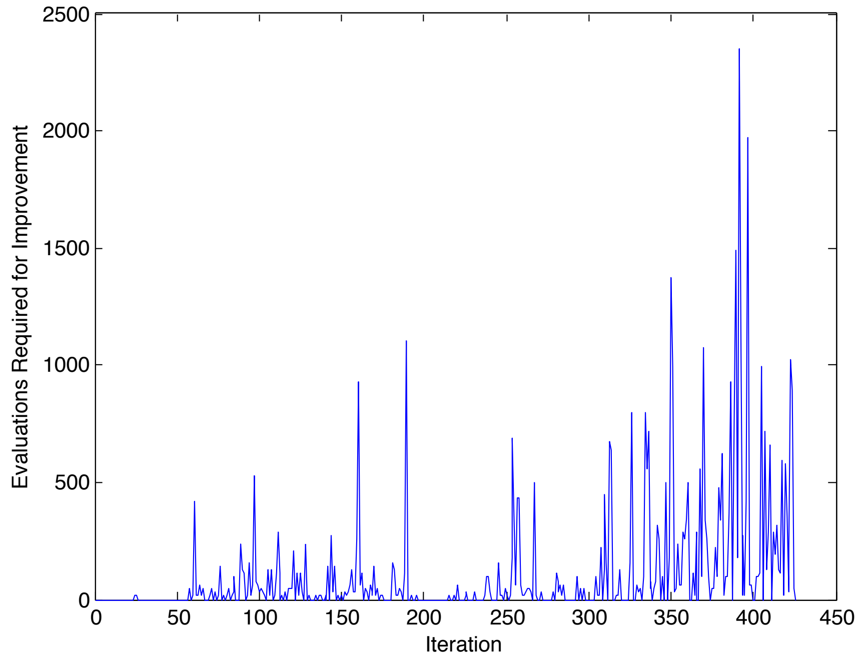


Figure 6.5: Number of evaluations required to find an improved set of parameters at each iteration of time optimization for the prone to stand motion.

slopes, thousands of evaluations are required to find an improvement.

6.2.3 Generalization for Character Size

Using the same procedure used for sloped terrain, our optimization procedure can generate parameters for characters of different body size. Instead of modifying the ground slope, the character's torso dimensions are scaled. The torso width and height are both multiplied by a factor, α, which is initialized at 1, and increased by 0.05 at each iteration. The torso is assumed to have a constant density, so this increases its mass and moment of inertia as well. This procedure generates controllers capable of prone-to-stand motions for characters with torso width and height 1.45 times larger, and torso mass $1.45^2 = 2.10$ times larger. Selected frames from the prone to stand sequence for a larger character are shown in Figure 6.8. The

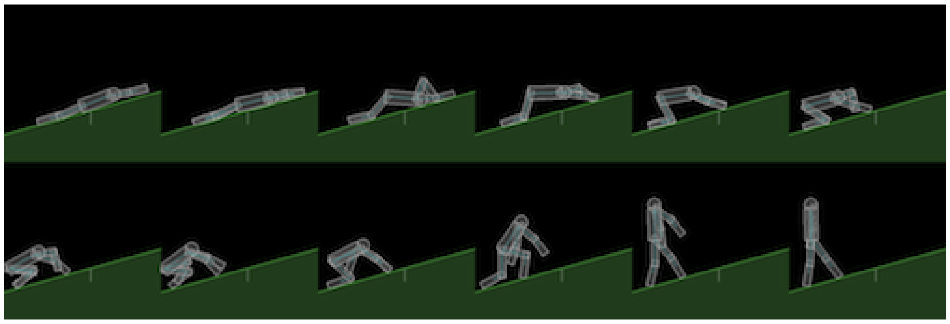


Figure 6.6: Selected frames from prone to stand sequence with parameters optimized for a steep, uphill slope.

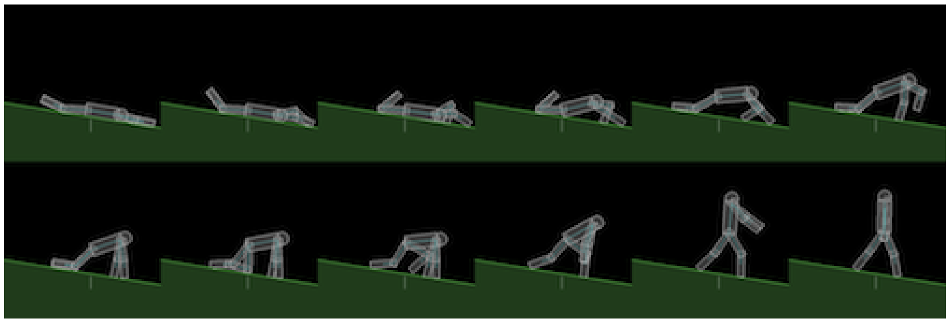


Figure 6.7: Selected frames from prone to stand sequence with parameters optimized for a steep, downhill slope.

computation effort required is similar to that of slope optimization, requiring few evaluations for small modifications, and thousands of evaluations for the largest characters.

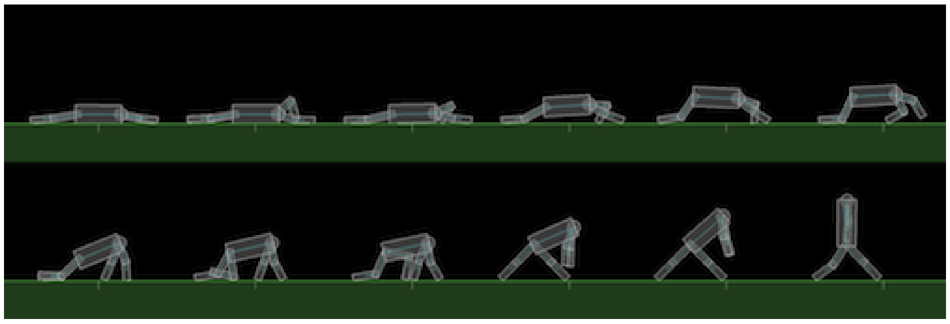


Figure 6.8: Selected frames from prone to stand sequence with parameters optimized for a character with 1.45 times greater dimensions and 2.10 times greater torso mass.

Chapter 7

Conclusions

Generating physics based rising motions is a difficult problem because it requires careful control of a character's body while simultaneously controlling set of discontinuous contact forces. This necessitates planning of supports and maintaing balance while leading the character toward a goal pose, all with a constrained set of control inputs. In addition, these motions contain a mix of discrete aspects, such as whether a limb is planted, and continuous aspects. Some dynamic motions motions, such as a kip, are not closely related to statically stable strategies, and are difficult to discover and perform. The controllers presented in this thesis solve some of these problem for a variety of characters and motions.

7.1 Discussion

The controllers developed in this work demonstrate that pose tracking, root orientation correction and virtual forces can be combined to generate robust and flexible controllers for rising motions. They are intuitive enough to admit controllers to be designed by hand, and are stable enough to be modified via optimization. In addition, an abstraction layer, such as gait graphs or support graphs, on top of these components makes it easier to design a broad range of controllers by sharing balance feedback rules that are common across actions.

7.1.1 Modeling Issues

All simulations in this work use an articulated rigid body model of characters, actuated by internal joint torques. However, especially for contact-rich motions, this approximation is flawed. When designing the controllers, especially for the quadruped, dealing with these simulation flaws was a major obstacle. Performing a more accurate physical simulation, however, is computationally taxing, and it unclear if it allows for simpler control strategies. Simulating the activation of all the muscles that act on each joint likewise adds additional control parameters, but may provide insight into how humans and animals move and balance.

7.1.2 Key Control Components

The main challenge for most controllers, whether for locomotion, standing up, or other skills, is controlling the center of mass position and orientation. Given the parent joint location, IK and pose control can effectively control limbs to perform most tasks, correcting for disturbances without trouble. Root control, however, requires coordination between many limbs, and correcting from errors is nontrivial, e.g. requiring a recovery step to maintain balance. Therefore planning support points is one of the most important aspects of any controller. For the controllers developed in this thesis, plant positions are mostly specified by hand as a relative offset to the torso, which works for these examples because reasonable support positions can be reliably estimated.

Balancing once support locations are known is another key aspect of designing robust controllers. Contact changes are necessary and frequent for the types of controllers in this work, so being able to balance in arbitrary configurations is a requirement of flexible controllers. We use simple feedback models, but other optimization based approaches may provide more robust balancing behaviors [5]. Especially for dynamic motions, a balance strategy must be anticipatory, planning for future contact configurations and considering character momentum.

7.2 Limitations and Future Work

The support graph implementation described was developed for two dimensional characters. The most immediate direction for future work is to extend the framework to a three dimensional character. This would likely require little more than an additional lateral balance feedback strategy, perhaps similar to the one used on the quadruped character.

A more significant limitation is that authoring these motions is still currently a manual process. Even when optimization is used, an initial seed point must be created by hand. While the controller abstractions described do help make the process easier, the controllers we generated were the product of manual trial and error. A question that commonly arises in animation is how can motion be generated automatically, while providing artists with ample stylistic control? For physics based characters, this is especially challenging, since physics imposes an extra set of constraints on possible motions. The optimization approach described in Chapter 6 could be adapted to find parameter vectors which match an artist's vision, similar to the approach used by Liu et al. [15], but this would require an expensive offline computation. Finding a balance between the ease of authoring motions and the control to stylize them remains an open problem.

For robotics applications, the style of motion is less important, and a completely automated approach to generation of motions is therefore another interesting avenue of future work. Heuristics might be developed which can generate a suitable sequence of supports or key poses, which would then be connected. Starting with an initial solution of connected, statically stable poses, the time warp based optimization method may be able to generate faster, dynamic motions or minimize expended energy among other goals.

The optimization strategy described here likewise has several limitations. The current strategy cannot modify the number and type of support changes for the limbs. For example, while timings and relative orderings can change, the left arm in the motions from Chapter 6 always begins in the idle state, transitions to swing state twice, stance state twice, and idle state once. This means that the optimization procedure described in this work cannot generate new motion strategies. The optimization could be extended to consider the addition or deletion of contact changes during the motion.

In addition, the optimization pipeline has potential improvements both in generality and efficiency. Currently, only one dimension of continuation can be computed at a time, e.g. slope *or* character size. However, applications will likely require a much higher degree of flexibility, so being able to efficiently explore the multidimensional space of controllers is an important direction for future work. The current naive, greedy, stochastic descent strategy could likely be improved to include the information from previous evaluations to guide the search.

Bibliography

[1] H. Asada and J. Slotine. *Robot analysis and control*. Wiley-Interscience, 1986.

[2] P. Beaudoin, S. Coros, M. van de Panne, and P. Poulin. Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 117–126. Eurographics Association, 2008.

[3] S. Coros, P. Beaudoin, and M. van de Panne. Generalized biped walking control. *ACM Transctions on Graphics*, 29(4):Article 130, 2010.

[4] S. Coros, A. Karpathy, B. Jones, L. Reveret, and M. van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):Article TBD, 2011.

[5] M. de Lasa, I. Mordatch, and A. Hertzmann. Feature-Based Locomotion Controllers. *ACM Transactions on Graphics*, 29(3), 2010.

[6] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260. ACM, 2001. ISBN 158113374X.

[7] H. Hirschfeld, M. Thorsteinsdottir, and E. Olsson. Coordinated ground forces exerted by buttocks and feet are adequately programmed for weight transfer during sit-to-stand. *Journal of neurophysiology*, 82(6):3021, 1999. ISSN 0022-3077.

[8] Jbox2D. Jbox2d. http://www.jbox2d.org.

[9] F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, S. Kajita, K. Yokoi, H. Hirukawa, K. Akachi, and T. Isozumi. The first humanoid robot that has the same size as a human and that can lie down and get up. In *Robotics and*

Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, volume 2, pages 1633–1639. IEEE, 2003.

[10] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *ACM SIGGRAPH 2008 classes*, pages 1–10. ACM, 2008.

[11] A. Kralj, R. Jaeger, and M. Munih. Analysis of standing up and sitting down in humans: definitions and normative data presentation. *Journal of biomechanics*, 23(11):1123–1138, 1990. ISSN 0021-9290.

[12] Y. Kuniyoshi, Y. Ohmura, K. Terada, A. Nagakubo, S. Eitoku, and T. Yamamoto. Embodied basis of invariant features in execution and perception of whole-body dynamic actions–knacks and focuses of roll-and-rise motion. *Robotics and Autonomous Systems*, 48(4):189–201, 2004.

[13] Y. Lee, K. Wampler, G. Bernstein, J. Popović, and Z. Popović. Motion fields for interactive character locomotion. In *ACM Transactions on Graphics (TOG)*, volume 29, page 138. ACM, 2010.

[14] W.-C. Lin and Y.-J. Huang. Rising from various lying postures. In *Computer Graphics International 2011 papers*. CGI, 2011.

[15] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu. Sampling-based contact-rich motion control. *ACM Transctions on Graphics*, 29(4):Article 128, 2010.

[16] J. Pratt, C. Chew, A. Torres, P. Dilworth, and G. Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research*, 20(2):129, 2001.

[17] M. Raibert and J. Hodgins. Animation of dynamic legged locomotion. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 349–358. ACM, 1991.

[18] P. Roberts and G. McCollum. Dynamics of the sit-to-stand movement. *Biological cybernetics*, 74(2):147–157, 1996. ISSN 0340-1200.

[19] M. Roebroeck, C. Doorenbosch, J. Harlaar, R. Jacobs, and G. Lankhorst. Biomechanics and muscular activity during sit-to-stand transfer. *Clinical Biomechanics*, 9(4):235–244, 1994. ISSN 0268-0033.

[20] R. Smith. Open dynamics engine. http://www.ode.org.

[21] J. Stückler, J. Schwenk, and S. Behnke. Getting back on two feet: Reliable standing-up routines for a humanoid robot. In *Proc. of The 9th International Conference on Intelligent Autonomous Systems (IAS-9), Tokyo, Japan*, pages 676–685. Citeseer, 2006.

[22] J. Wang, D. Fleet, and A. Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, page 73. ACM, 2010.

[23] K. Yin, S. Coros, P. Beaudoin, and M. van de Panne. Continuation methods for adapting simulated skills. *ACM Trans. Graph.*, 27(3), 2008.