UNIVERSITY OF CALIFORNIA
RIVERSIDE

Simulation and Control of Humanoid Rolling and Falling Behaviors

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

David Frederick Brown

December 2012

Acknowledgments

First I would like to thank my advisor, Victor Zordan for the advice, guidance, and many opportunities he has provided for me. Without his help this thesis would not have been possible. I thank Adriano Macchietto for sharing his experience and providing much of the basis for this work. I would also like to thank the other members of UCR's graphics lab, especially Nam Nguyen and Nkenge Wheatland, who made it a great place to work and study. Finally I would like to thank my parents Lora and Jim, and my sister Julie for their constant support.

ABSTRACT OF THE THESIS

Simulation and Control of Humanoid Rolling and Falling Behaviors

by

David Frederick Brown

Master of Science, Graduate Program in Computer Science
University of California, Riverside, December 2012
Dr. Victor Zordan, Chairperson

Controlling physically based characters is an ongoing problem in animation. Motions that involve a large of changing contacts with the environment are especially problematic. In this thesis we present techniques for controlling contact rich motions, specifically rolling and falling motions. For rolling we present a method for controlling the full body orientation of a multibody character. We use this to describe a roll for the character to perform in terms of the character's goal orientation, linear velocity and angular velocity. We then use a quadratic programming controller to compute the torques to best achieve these goals. To deal with the many and constantly varying contacts we introduce a heuristic that selects only the most important contact points to be active in the quadratic program. Falling is similar except it requires time varying orientation and position goals for the character. We automatically compute these goals by defining a function to evaluate the quality of a fall and use sampling based optimization to find the best way to control the character for a given fall. We demonstrate our rolling controller for several different rolls in different environments and our falling controller on a variety of different falls.

Contents

List of Figures

Chapter 1

Introduction

Physically simulating character animations is useful for a variety of disciplines. It allows animators to include the subtle artifacts of physical motion in characters without needing to perform lots of tedious tweaking. It allows the animations of characters in video games to convincingly react to external stimuli for a more immersive experience. It can also help robotics researchers perform experiments that would be too difficult or dangerous to try with a real robot. Falling and rolling motions are a particularly suited for physical simulation because they involve such complex interactions with the environment.

In this thesis we present techniques for simulating and controlling humanoid characters in falling and rolling motions. For falling we use a stochastic global trajectory optimization algorithm to find the best parameters for some general controllers according to several quality metrics. We also introduce a method for controlling characters performing contact rich tasks such as rolling and tumbling. Both of these techniques use the same low level momentum based quadratic program controller controller [27].

1.1 Motivation

Humanoid animation and simulation has been a popular field of research in character animation for the past several decades. We have reached the point where it is possible to create robust controllers for common tasks such as balancing and walking. We have also seen these paired with insightful path planning algorithms allowing characters to navigate complex and dynamic environments. However most of these tasks the contact points and configuration are relatively simple and known beforehand. These methods are not as helpful for motions that require a large number of unpredictable contact points such as rolling. In addition, even the most robust character or robot is bound to fail under some extreme circumstances. In such cases we must design a way for the character to fail believably and, in the case of robots, safely. Our work is predominantly focused on an animation solution suited for interactive applications, however this work could also be applied to robotics and kinesiology.

Animation

It can be difficult to hand animate a falling or rolling character because the dynamics of a multibody and the collisions with the ground can be quite complicated. The animations are also very dependant on the topology the character is falling on. As such it would be very useful to be able to pragmatically generate these animations for different characters. These animations could then be used as is, or tweaked as necessary to fulfill other constraints of the animation.

With interactive animations the problem is much more acute because there is essentially impossible to create animations for every possible fall configuration. Many video games use “rag doll” physics to animate fallen characters, however this makes the characters completely passive when falling instead of trying to perform some kind of recovery during the fall.

Robotics

With robots, the primary concern is ensuring the robot does not damage its self or any objects or people around it when falling. Robotics research often uses simulations to test algorithms more quickly and without the cost of acquiring and setting up a real robot. Testing falling control and algorithms makes damaging the robot much more likely. Simulations can also be used to run tests much more rapidly than on a real robot. Our method could be used to refine a falling control algorithm until it is safe enough to be tested on a real robot.

Kinesiology

Athletes in fast paced competitive sports are bound to fall at some point during training or competitions. Our system could be used to develop better falling techniques for actual humans to help avoid injury during such sports.

1.2 Contributions

The main contributions of this thesis are a system for controlling contact rich rotational motions such as rolling and a trajectory optimization controller for falling motions. We demonstrate the rolling controller with various types of rolls on different surfaces. For the falling controller we generate several falls with different objectives and control techniques.

With our falling system we can experiment with a variety of measures for the quality of a fall such as the impact force, which body part was hit, and where the character fell. We then use these measures to evaluate falls using a variety of control techniques such as controlling the character's pose while falling, center of mass position and angular momentum while falling. Using different combinations of these quality measures and control techniques we are able to generate a variety of convincing fall animations.

We also present a novel rolling controller that is able to generate convincing and physically accurate rolling animations. We use a momentum controller to regulate the motion of the character and a contact management system that decides which contacts the system should use to control the character. The combination of these two makes for a rolling controller that produces high quality animations and can operate in a variety of environments.

1.3 Thesis Organization

In Chapter 2 we summarize related research work to ours and topics that we build upon. This includes research in animation, robotics and biomechanics. In Chapter 3 we go over the dynamics of our simulation and the low level momentum controller we use. Chapter 4 introduces our control system for contact rich rotational motions. Chapter 5 introduces our method for optimizing a trajectory for a falling character. In Chapter 6 we review the results of both methods and in Chapter 7 contains concluding remarks.

Chapter 2

Related Work

Physical modeling and simulation of humanoids has been used in a variety of fields and topics of research. Most relevant to our work is research in the areas of physics based character animation, robotics, and biomechanics. In Section 2.1 we review the relevant research in computer animation, including physics based character animation, contact management, and sampling. In Section 2.2 we look at fall detection and damage reduction research in robotics. Finally in Section 2.3 we examine a few related works in biomechanics.

2.1 Animation

There are many subtle details in humanoid motion that can be very difficult to recreate with an animated character. While we can now create photorealistic models of humans, animating these characters often breaks our suspension of disbelief. A common method for creating realistic animations is to use some form of motion capture and there is a large body of research on that topic [28, 29]. Motion capture editing techniques make it possible to edit and adapt recorded animations to new situations and characters.

However these editing techniques can also result in unrealistic motions. Modeling the physics of the character ensures that the resulting animations are physically consistent. They also enable us to generate new animations that can react to and interact with the environment.

2.1.1 Task Controllers

One technique for creating physically simulated animations is to design controllers that is able to perform specific tasks. These tasks are usually smaller parts of an overall animation. Examples include balancing, walking, and reaching for an object. Techniques for doing this include designing a controller by hand, using a mathmatical model of the dynamics to create a controller, and using a hybrid of kinematic control and dynamic simulation.

Hand Designed Controllers

Walking is one of the more common applications these controllers in the literature. An early example of this can be found in [59] where the author divides a task such as walking into a parameterized finite state machine where each state is a part of the full animation. However Zeltzer's system was purely kinematic with no simulation. Bruderlin and Calvert [5] present a simulation of a walking human using a hand designed controller. They model the stance leg as an inverted double pendulum and the swing leg as a regular double pendulum. Their system also has several parameters to adjust the speed and appearance of the walking motion.

Raibert and Hodgins developed controllers for locomotion simulations including biped walking, quadruped walking and a plainer kangaroo [39]. Their system also used state machines where each state uses hand tuned proportional-derivative (PD) controllers were used to compute torques for each of the character's joints.

Since hand crafting these controllers is so time consuming, there has been a variety of research into stringing together multiple task controllers to create longer more complex animations. Wooten demonstrates a technique for doing this by simulating acrobatic motions such as leaping, tumbling, and landing [53]. He designs controllers where the exit state for one task leaves the character in a valid initial state for another. Doing so he is able to combine controllers for a single acrobatic maneuver into longer routines.

Faloutsos et al. also use the combining controllers technique [10]. They use composable controllers including walking, jumping, rolling over, and getting up. The system treats each controller as a black box and can be queried to see how suitable it is for the task at hand. Each must also be able to specify it's range of possible starting states and ending states, and to evaluate if any failure has occurred. A higher level system can use these to determine which controllers should be used to achieve a specific task. They also use a supervised learning technique for determining the starting conditions of controllers.

Coros et al. used a combination of hand designed controllers, dynamics modeling, and optimization to design a quadruped controller [6]. They used virtual forces to compute the torques necessary to control the height and velocity of the quadruped and PD controllers to drive the character to a particular pose in a particular phase of the gait. Finally they used stochastic optimization to tune the parameters of their controller to improve the speed and robustness of the controller and to try and match some example data of dog locomotion. They also design controllers for leaping, sitting, lying down, and getting up.

Dynamics models for Control

Hand designing controllers for specific tasks can produce convincing and robust controllers. However it can be very tedious to design a new controller for every task in a particular animation. One way to alleviate this is to come up with a model for the dynamics that enables you to control features of a character, such as the center of mass acceleration or end effector position, instead of raw joint torques. Controlling higher level tasks such as balancing or walking is easier using these features than if you had to control individual joint torques. Another benefit of these methods is they generally do not depend on the skeletal structure of the character and so controllers designed using them can often be used on a variety of characters.

One such method is using Jacobian transpose control to apply virtual forces [45, 38, 36]. This control method allows us to apply virtual forces (or torques) to a particular point where we may not have a way of directly applying a force. For example to balance a character we may want to apply a force to the character's center of mass. Using Jacobian transpose control we can apply a virtual force to the center of mass that is realized through the contact points of the supports. Yin et al. use virtual torques in their SIMBICON walking controller [57]. They use one virtual torque to keep the character's torso upright and another to position the swing leg which maintains the balance of the character. Coros et al. use a variety of virtual forces in their quadruped controller [6] for gravity compensation and velocity control.

Another method is to perform multi-objective control using a local quadratic optimization as introduced by Abe et al. [2]. We will cover this method in detail in Section 3.2. Briefly the method models the character's dynamics as a quadratic program with linear constraints. Weighted objectives such as center of mass position and end effector position can be added to the system. The system is then solved using a constrained quadratic optimization solver to get the resulting joint torques of the character. Abe et al. used this method to create a robust balancing character that could reach its arms while staying in balance. Macchietto et al. extended this technique to use linear and angular momentum objectives [27] which improved the balancing robustness and resulted in more realistic responses to external pushes. De Lasa et al. used a slightly modified version of this method with prioritized objectives to create general jumping and walking controllers [8]. They showed their controllers could be generalized to characters with different body shapes and morphologies.

Hybrid Controllers

Simulation based techniques can produce very lifelike animations, however they tend to give animators less control over the resulting animation. There are control methods that attempt to combine the best of both simulation based techniques and pure kinematic control. Often these techniques perform a simplified version of the simulation or constrain it in some way to conform to some kinematic data.

Zordan et al. track motion capture data in a physical simulation using PD controllers to track joint angles and a balance controller to keep the character upright [60]. Undisturbed, the character follows the motion capture data closely, however when the character encounters an external force it reacts in a physically realistic way. Nguyen et al. use a similar technique to create reactive animations in real time using motion capture for input [33]. Shapiro et al. [42] use a system similar to [10] but which uses both kinematic and dynamic controllers. The character switches to a dynamic controller to react to unexpected contacts and then can switch back to a kinematic controller when it is sufficiently close a kinematic controller's starting state. Tang et al. use a somewhat similar technique to generate falling animations [48]. When a character begins to fall their system uses simulation to calculate the trajectory of the fall and determine a good motion to transition to after falling. Then a physically generated falling motion is used to create the transition from the current pose to the post fall motion clip.

2.1.2 Sampling and Genetic Algorithms

Sampling based techniques and genetic algorithms have been used fairly extensively in computer animation for everything from choosing parameters for a controller [6] to creating entire virtual creatures [43]. These techniques are appealing because they are conceptually simple and can produce good results. However they can be very time consuming if the search space is large and the quality of the results can be very dependant on the initial conditions as well as the objective function.

Ngo and Marks use a genetic algorithm to generate gaits for simple two dimensional creatures [32]. They used a set of learned stimulus-response functions to control the character. Their work showcases the potential of such techniques in that they were able to generate some interesting gaits for various creatures and even one that resembled human walking while only using longest distance traveled as an objective. However the animations are not particularly natural looking and to obtain the human walking gait they had to tweak their objective function to be the average distance travelled by the feet instead of the position of the center of mass. Gritz and Han use a genetic algorithm to generate a gait for a Luxo lamp in three dimensions [16]. They make their controller more robust by evaluating their fitness function over several trials with different initial conditions. They also evaluated the style of the motion by penalizing falling over and moving after reaching the goal, while rewarding the speed at which the goal was reached. This results in more natural looking animations and enables the gait to be repeated for longer animations.

Sims used a genetic algorithm to evolve both creature morphology and control for swimming, walking, jumping, and following a light source [43]. The resulting creatures developed a wide variety of strategies for performing their task, some of which resembled features of real animals such as fins. However the system is mostly automatic and gives very little control over the resulting creatures and behaviors.

Instead of generating unique motions, Sok et al. take existing motion capture data or key-framed animations and modify them so they can be physically simulated using PD controllers in two dimensions [44]. They use a sliding optimization window to improve performance and are able to chain several optimized controllers together. Similarly, Liu et al. use a sampling based approach to simulate contact rich animations including rolling [26]. They use 3D motion capture data as input and use sampling to determine the paramaters of PD controllers that will successfully perform that animation in a simulation.

Wampler and Popović use a hybrid approach of gradient based local optimization and gradient free global optimization to generate animal gaits [49]. Since changes in contact times causes discontinuities they use the gradient free optimization to manipulate the contact times and positions and then use the gradient-based optimization to find the best solution with a given contact profile.

Wang et al. use a sampling based optimization technique to improve the SIMBICON walking controller and adapt it to different characters and environments [50]. They use Covariance Matrix Adaptation [18] to determine parameters for the walking controller that minimize energy exertion, angular momentum and head movement. This produces a more natural walk that includes human walking features like active toe-off.

Yin et al. also build upon the SIMBICON controller but instead they adapt it to perform more extreme tasks such as stepping over obstacles, stepping up large steps, walking on steep inclines and pushing heavy objects [56]. They also compare several different optimization techniques including a gradient method using derivative approximations, a hybrid stochastic and gradient based method, and a stochastic search method with different parameters. In their results the stochastic methods generally require fewer function evaluations but the hybrid approach results in lower objective function values and better looking motions.

2.1.3 Contact Management

Dealing with contacts continues to be a topic of research in computer animation. Contact forces are usually the only external forces a character has control over when performing a task, and the position of those contacts affects how the character can leverage them to achieve its goals. When performing a task a character must decide not only what to do with the contacts it has, but also if it should change its contact configuration. One of the main difficulties with this is that when contacts break or new contacts appear it causes a discontinuity in the amount of control the character has, so changing contacts are problematic for gradient based optimization techniques.

Even in just resolving contact forces in multibody simulation is not a straightforward problem. There can be many solutions and possibly no solution for a given constraint configuration [4]. The usual technique of resolving contacts is formulating a Linear Complementarity Problem who's solution gives the contact forces and accelerations for a given point in time [3]. However LCP algorithms only find one of solution to the contact force problem. If there are more than one solution they have no way of evaluating which one might be most suitable for a particular situation. This becomes a problem when you are using an optimization based controller where one solution could be better than others.

Another problem is when using quadratic optimization based controller such as in [2] the contact constraint is generally expressed as bilateral constraints to keep the problem convex. However this prevents the controller from ever breaking contact. Tan et al. [47] select which contacts should be used for their soft body controller using an iterative pivoting algorithm. Performance is another reason for wanting to exclude some contacts from the control formulation. With rigid bodies the number of contact points tends to be relatively limited in number, however if using soft contacts such as Jain et al. [20] the number of contacts can be much larger.

2.2 Robotics

Fall management is a very important topic in robotics. Although preventing falls should be a primary goal, it is still inevitable that a humanoid robot in uncertain environments will fall occasionally. Therefor the robot must have a strategy for minimizing the damage to its self and its surroundings in the event of a fall. To employ some falling strategy the robot must also be able to detect when it is falling and usually the sooner a fall is detected, the more that can be done to mitigate it. In this section we will review the relevant research on fall control and detection.

2.2.1 Fall Control

A common strategy for mitigating the damage to a robot from a fall is the Ukemi strategy introduced by Fujiwara et al. [13] and inspired by martial arts falling techniques. The technique involves crouching during the fall to minimize the amount of induced angular momentum induced by the fall. The robot also orients its self to put shock absorbers on likely impact points of the robot and during the fall the robot will orient its self so these points absorb most of the impact. Fujiwara et al. demonstrate the effectiveness of this method on a real robot in [14]. Fujiwara et al. also develop a specific strategy for forwards falls on the knee in [15].

Other works have used simplified models of the dynamics of a robot to come up with fall strategies. Ogata et al. verify the effectiveness of the Ukemi strategy of contraction and expansion using a linear inverted pendulum [34]. They calculated an optimal trajectory of contraction and expansion for the inverted pendulum that minimized the velocity at impact. Fujiwara et al. use a more complicated triple and quadruple inverted pendulum to design fall controllers where the knee impacts first [12, 11]. They use this model to perform an optimization that minimizes several weighted parameters including the impact force at the knee and the arms, and the angular momentum at impact.

Ruiz-del-Solar et al. created a robot fall simulation that evaluates falls based on the impact force at the joints and other fragile points on the robot [41]. They then used this system to manually optimize the joint positions during the fall to minimize these forces. They develop several strategies this way and then test them on a real robot.

During a fall a robot may need to consider not only damage to its self but also damage to its surrounding environments. Although the robot may not be able to prevent the fall it can try to at least fall in a safe direction. Yun et al. and Nagarajan and Goswami control a robot's falling direction by taking steps and shaping the robot's inertia during the fall [58, 31]. Stepping is used to change the robot's support region so it will fall in a particular direction. If stepping fails the robot can also shape its inertia to produce an angular momentum in the desired direction. Lee and Goswami also use inertia shaping, but instead use it to ensure a specific part of the robot (in this case a backpack) make contact with the ground first [25].

2.2.2 Fall Detection

For a robot to react to a possible fall it must first detect that it is falling. A commonly used technique is to look at an indicator of the overall stability of the robot. One indicator is the center of mass. If the center of mass projected on the ground the robot is in balance. However in some tasks such as walking the center of mass is rarely directly over the support even when the robot is not in danger of falling. Another more robust indicator is the Zero Moment Point (ZMP). The ZMP is the support point where a reaction force applies no moment about the horizontal axes. [35] use the estimated position of the zero moment point (ZMP) to detect instability in a robot and decide whether to take a step or perform some shock reducing motion. The shock reduction motion they use attempts to minimize the velocity of the center of mass at impact and is similar to the crouching used in the Ukemi strategy

There are also more complex models for detecting robot stability. Pierre-Brice reviews several methods of stability detection and then introduces a method for computing a viability kernel within which the robot will not fall [37]. Renner and Behnke design use an attitude sensor in the chest of the robot to detect instability [40]. They create a model of attitude readings under usual circumstances and use sensor readings that deviate from this as an indicator of instability. Kalyanakrishnan and Goswami attempt to predict humanoid falls early by examining the robot's linear and angular momentum [21, 22]. They use machine learning techniques to model when the character is going to fall using the robot's momentum and support configuration as features. Similarly [23] use Principal component analysis over the state of the robot to generalize good states of a robot and use deviation from those good states to predict a fall.

2.3 Biomechanics

Much of the biomechanics research on falling is on what kind of damage is caused by falling and what usual human reactions are to falling. A large portion is also devoted to falls by elderly people since falling can be more devastating for them. Hsiao and Robinovich studied what kind of movements people perform to protect themselves when falling [19]. They had their subjects stand on a mattress and then abruptly translated the mattress to simulate a slip. They found that subjects forwards and to the side tended to impact with their wrist first, while when falling backwards the wrist and pelvis impacted at nearly the same time.

Kim and Ashton-Miller compared the upper body movements of young and old subjects when falling forwards [24]. Although both the old and young subjects broke their fall with their hands, the young subjects reacted faster to unexpected falls and had lower maximum impact force. Groen et al. studied the techniques of martial artists performing sideways falls and how they helped reduce impact force [17]. They found that compared with simply blocking the fall with your arm, the martial arts techniques reduced the impact force significantly. They concluded that this was due to reduced impact velocity and rolling after impact.

Chapter 3

Dynamics and Control

Before describing our method of simulating contact rich motions we must first introduce the simulation and control system it is built upon. The system is split into a controller that calculates joint torques for the character at each time step and a forward simulator that uses those forces to step the character forward in time. We use a standard multibody forward simulation system as described in [9]. The controller is based on the one used in [27] and [8].

In Section 3.1 we will review the mechanics of contacts in this system. In Section 3.2 goes over the system's dynamics and introduces the low level controller used in our system. Finally in Section 3.3 we describe our method for measuring and controlling the full body orientation of the character.

3.1 Contact Mechanics

Contacts with the environment apply an external force the character. This force can be manipulated by our joint actuations and forms the basis of our control. We only consider static friction in our contacts because dynamic friction makes the optimization

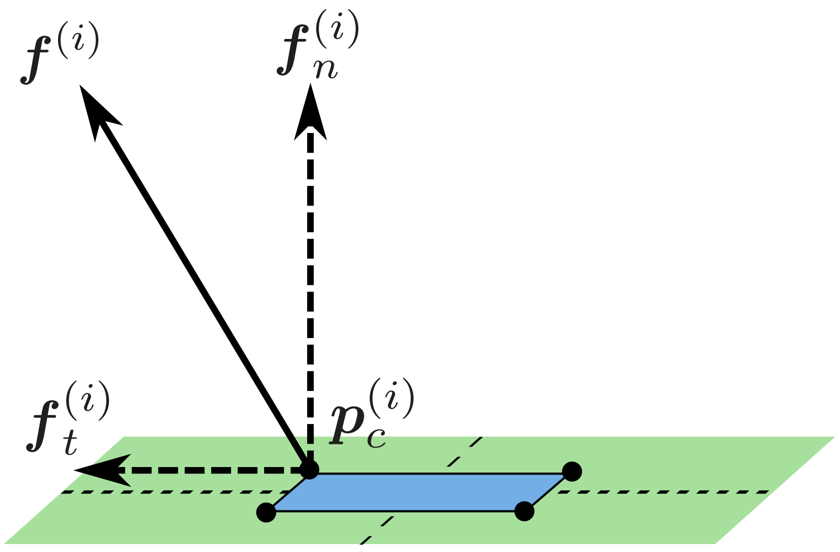


Figure 3.1: An example contact point with the contact force decomposed into normal and tangential components

we use non-convex. Each contact point $\boldsymbol{p}_c^{(i)}$ can exert a contact force $\boldsymbol{f}^{(i)}$. We split this contact force into its tangential component $\boldsymbol{f}_T^{(i)}$ and its normal component $\boldsymbol{f}_n^{(i)}$ as shown in Figure 3.1. Since contacts cannot perform work me must enforce the zero work (complementarity) constraint $\boldsymbol{f}_n^{(i)^t}\dot{\boldsymbol{p}_c}^{(i)} = \mathbf{0}$. We can express this in joint velocities $\dot{\boldsymbol{q}}$ using the contact force Jacobian $\boldsymbol{J}_c^{(i)}(\boldsymbol{q})$.

$$\boldsymbol{J}_c^{(i)}(\boldsymbol{q})\dot{\boldsymbol{q}} = \dot{\boldsymbol{p}_c}^{(i)} \tag{3.1}$$

Since we are limited to static friction we can use the constraint $\dot{\boldsymbol{p}_c}^{(i)} = \mathbf{0}$ to enforce the complementarity constraint. This forces the character to always maintain contacts, however in contact rich motions we often need to break contact. To overcome this we express the complementarity constraint as a highly weighted objective instead of a constraint in our optimization as described in Section 3.2 and we use a heuristic for deciding when to enable and disable specific contacts in the optimization (described in Section 4.3).

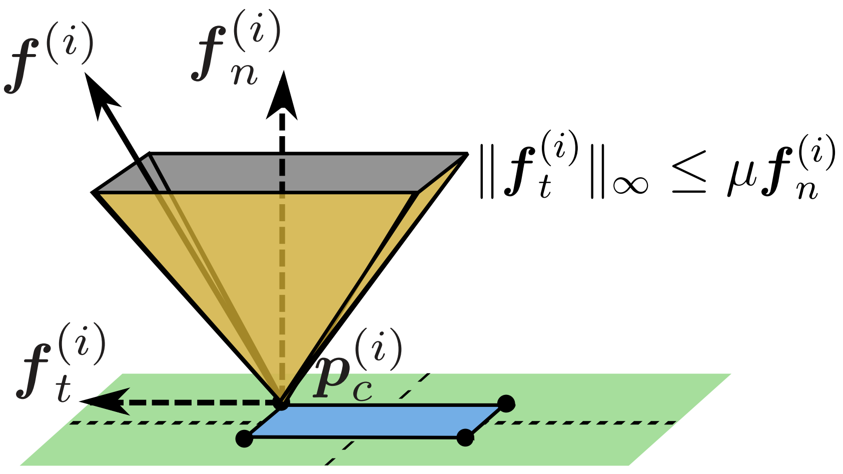


Figure 3.2: The friction pyramid constraint for a contact point

There are two other constraints on contact forces in addition to the complementarity constraint. Unlike bilateral joints, a contact can only pushes the body away from it and not pull it back, implying that the normal component of the contact force must be positive: $\boldsymbol{f}_n^{(i)} \geq 0$. Furthermore since we are dealing with no-slip contacts, Coulomb friction limits the magnitude of the tangential component of the force based on the magnitude of the normal component: $\|\boldsymbol{f}_t^{(i)}\| \leq \mu \boldsymbol{f}_n^{(i)}$ where $\mu \geq 0$ is the coefficient of friction for the contact. This constrains the force to be in a friction cone centered on the normal component of the contact force. We use a linear approximation of this constraint $\|\boldsymbol{f}_t^{(i)}\|_\infty \leq \mu \boldsymbol{f}_n^{(i)}$ as in [2] forming a friction pyramid as shown in Figure 3.2.

3.2 Low Level Controller

The equations of motion of a multibody system can be written as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\tau} \end{bmatrix} \tag{3.2}$$

Where $\boldsymbol{M}$ is the generalized inertia matrix, $\boldsymbol{C}$ is the generalized bias force (combining Coriolis, centrifugal and external forces) and $\boldsymbol{\tau}$ is the generalized force. The $\mathbf{0}$ in the vector on the right hand side represents the unactuated degrees of freedom of the multibody's root.

Our controller uses the momentum control method from [27] which in turn uses the multi-objective optimization method of [2]. The controller uses a constrained quadratic optimization to minimize several weighted objectives (described in Section 3.2.1) while upholding the dynamics and contact constraints. If n is the number of degrees of freedom of the character and m is the number of contact points then the optimization state is

$$\boldsymbol{x} = \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\tau} \\ \boldsymbol{f} \end{bmatrix} \tag{3.3}$$

Where $\ddot{\boldsymbol{q}} \in R^n$ are the generalized joint accelerations, $\boldsymbol{\tau} \in R^{n-6}$ are the joint torques and $\boldsymbol{\lambda} \in R^{3m}$ are the contact forces.

The quadratic optimization has the form

$$\frac{1}{2}\sum_{k=1}^{p} \|\boldsymbol{A}_k\boldsymbol{x} - \boldsymbol{b}_k\|^2_{\boldsymbol{W}_k} \tag{3.4}$$

where p is the number of optimization objectives, $\boldsymbol{A}_k$ and $\boldsymbol{b}_k$ are the objective's linear and constant terms, and $\boldsymbol{W}_k$ is the objective weighting matrix. There are three constraints to this quadratic program. The first is

$$\boldsymbol{f}_N^{(i)} \geq 0 \qquad i = 1, \ldots, m \tag{3.5}$$

This ensures that contact force is unilateral and can only push the character away from the ground. The second constraint also relates to contact forces

$$\|\boldsymbol{f}_T^{(i)}\|_\infty \leq \mu \boldsymbol{f}_N^{(i)} \qquad i = 1, \ldots, m \tag{3.6}$$

This ensures that the contact forces uphold Coulomb friction remain within the approximate friction cone. The last constraint is Equation 3.2 which ensures the contact forces, joint torques and joint accelerations are consistent with the dynamics.

As mentioned in Section 3.1 we do not express the complementarity constraint as a hard constraint in our optimization. Instead it is included as a highly weighted objective. This keeps the character from performing some extreme exertion to maintain a contact that really should just be broken. This objective has the form

$$\frac{1}{2}\|\dot{\boldsymbol{p}}_c\|^2_{\boldsymbol{W}_v} \tag{3.7}$$

where $\boldsymbol{W}_v$ is the weight matrix for the complementarity objective and $\dot{\boldsymbol{p}}_c$ is the concatenation of all the contact point velocities. The full optimization can now be stated as

$$\begin{aligned}
\min_{x} \quad & \frac{1}{2}\sum_{k=1}^{p}\|\boldsymbol{A}_k\boldsymbol{x} - \boldsymbol{b}_k\|^2_{\boldsymbol{W}_k} + \frac{1}{2}\|\dot{\boldsymbol{p}}_c\|^2_{\boldsymbol{W}_v} && (3.8)\\
\text{subject to} \quad & \boldsymbol{f}_N^{(i)} \geq 0, && (3.9)\\
& \|\boldsymbol{f}_T^{(i)}\|_\infty \leq \mu \boldsymbol{f}_N^{(i)}, \qquad i = 1, \ldots, m && (3.10)\\
& \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \end{bmatrix} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) && (3.11)
\end{aligned}$$

3.2.1 Controller Objectives

Since we use a semi-implicit integration method, all the objective functions have also been modified to be semi-implicit. We use the semi-implicit Euler integration method:

$$\begin{aligned}
\dot{x}_{t+h} &= \dot{x}_t + h\ddot{x}_t \\
x_{t+h} &= x_t + h\dot{x}_{t+h} && (3.12)
\end{aligned}$$

Pose Tracking

The pose tracking objective tries to get the character to match a specific pose or some pose in a linear blend between two poses. Since the linear momentum and angular momentum objectives together only specify 6 degrees of freedom, we need the tracking objective to guide the optimization to natural a natural looking solution. Generally this objective is weighted lower than the momentum objectives. The optimization tries to meet the momentum goals as much as possible then as a less important goal it tracks some given pose.

We use a force based tracker to give us compliant tracking at low optimization frequencies. Prior quadratic optimization-based controllers [2, 27, 7] have used acceleration based trackers. We find the force based formulation to be more compliant when dealing with passive contacts.

Let quantities with ˆ represent desired tracking values and $\boldsymbol{K}_s$, $\boldsymbol{K}_d$ represent the spring and damping gains respectively. The tracker objective has the form

$$\boldsymbol{\tau} = \boldsymbol{K}_s \left(\hat{\boldsymbol{q}}_t - \boldsymbol{q}_{t+h}\right) + \boldsymbol{K}_d \left(\hat{\dot{\boldsymbol{q}}}_t - \boldsymbol{q}_{t+h}\right) \tag{3.13}$$

Substituting Equation 3.12 and rearranging the terms we get

$$\left(h^2 \boldsymbol{K}_s + h\boldsymbol{K}_D\right) \ddot{\boldsymbol{q}} + \boldsymbol{\tau} = \boldsymbol{K}_s \left(\hat{\boldsymbol{q}}_t - \boldsymbol{q}_t\right) + \boldsymbol{K}_d \left(\hat{\dot{\boldsymbol{q}}}_t - \dot{\boldsymbol{q}}_t\right) - h\boldsymbol{K}_s \dot{\boldsymbol{q}}_t \tag{3.14}$$

Which we can represent in the optimization form as

$$\boldsymbol{A}_{\text{track}} = \begin{bmatrix} h^2 \boldsymbol{K}_s + h\boldsymbol{K}_D & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \tag{3.15}$$

$$\boldsymbol{b}_{\text{track}} = \boldsymbol{K}_s \left(\hat{\boldsymbol{q}}_t - \boldsymbol{q}_t\right) + \boldsymbol{K}_d \left(\hat{\dot{\boldsymbol{q}}}_t - \dot{\boldsymbol{q}}_t\right) - h\boldsymbol{K}_s \dot{\boldsymbol{q}}_t \tag{3.16}$$

This formulation specifies one specific pose for the character to track. We may instead want to track different poses based on the other objectives in the system. One way to do this is to provide two poses and let the optimization choose some linear (slerp) blend between those two poses to track. To do this we introduce two more optimization variables $\alpha^{(1)}$ and $\alpha^{(2)}$ that will be the blend values for the two poses $\hat{\boldsymbol{q}}^{(1)}$ and $\hat{\boldsymbol{q}}^{(2)}$. To do this we must extend the optimization state to be

$$\boldsymbol{x} = \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\tau} \\ \boldsymbol{f} \\ \alpha^{(1)} \\ \alpha^{(2)} \end{bmatrix} \tag{3.17}$$

We also add three more constraints

$$\alpha^{(1)} \geq 0 \tag{3.18}$$

$$\alpha^{(2)} \geq 0 \tag{3.19}$$

$$\alpha^{(1)} + \alpha^{(2)} = 1 \tag{3.20}$$

These ensure that the blend values only choose blends that are between the two poses. Lastly we modify the objective function by first pulling out and negating the terms from from Equation 3.16 that depend on the desired pose

$$\boldsymbol{u}^{(1)} = -\left(\boldsymbol{K}_s\left(\hat{\boldsymbol{q}}_t^{(1)} - \boldsymbol{q}_t\right) + \boldsymbol{K}_d\left(\hat{\dot{\boldsymbol{q}}}_t^{(1)} - \dot{\boldsymbol{q}}_t\right)\right) \tag{3.21}$$

$$\boldsymbol{u}^{(2)} = -\left(\boldsymbol{K}_s\left(\hat{\boldsymbol{q}}_t^{(2)} - \boldsymbol{q}_t\right) + \boldsymbol{K}_d\left(\hat{\dot{\boldsymbol{q}}}_t^{(2)} - \dot{\boldsymbol{q}}_t\right)\right) \tag{3.22}$$

We then append them to $\boldsymbol{A}_{\text{track}}$ and remove them from $\boldsymbol{b}_{\text{track}}$ accordingly:

$$\boldsymbol{A}_{\text{track}} = \begin{bmatrix} h^2\boldsymbol{K}_s + h\boldsymbol{K}_D & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{u}^{(1)} & \boldsymbol{u}^{(2)} \end{bmatrix} \tag{3.23}$$

$$\boldsymbol{b}_{\text{track}} = h\boldsymbol{K}_s\dot{\boldsymbol{q}}_t \tag{3.24}$$

Now the $\alpha^{(1)}$ and $\alpha^{(2)}$ optimization variables allow the system to track any pose along the blend between $\hat{\boldsymbol{q}}_t^{(1)}$ and $\hat{\boldsymbol{q}}_t^{(2)}$. It is possible to extend this to more than two poses, however it is much less intuitive to understand what a blend between three or more poses looks like and two poses was sufficient for our experiments.

Feature tracking

We use the feature tracking control described in [8]. A feature $\boldsymbol{y}$ can be any algebraic function of the generalized coordinates:

$$\boldsymbol{y} = f(\boldsymbol{q}) \tag{3.25}$$

The derivatives of this function are specified with its Jacobian $\boldsymbol{J}^y$:

$$\dot{\boldsymbol{y}}_t = \boldsymbol{J}_t^y \dot{\boldsymbol{q}}_t \tag{3.26}$$

$$\ddot{\boldsymbol{y}}_t = \boldsymbol{J}_t^y \ddot{\boldsymbol{q}}_t + \dot{\boldsymbol{J}}_t^y \dot{\boldsymbol{q}}_t \tag{3.27}$$

Since we are doing semi-implicit tracking we approximate the post integration feature value and its time derivative using Equation 3.12:

$$\dot{\boldsymbol{y}}_{t+h} = \boldsymbol{J}_t^y (\dot{\boldsymbol{q}}_t + h\ddot{\boldsymbol{q}}_t) \tag{3.28}$$

$$\boldsymbol{y}_{t+h} = \boldsymbol{y}_t + h\boldsymbol{J}_t^y \dot{\boldsymbol{q}}_{\boldsymbol{t}} + h^2 \dot{\boldsymbol{J}}_t^y \ddot{\boldsymbol{q}}_t + O(h^2) \tag{3.29}$$

We specify a desired value for this feature $\hat{\boldsymbol{y}}$ and its derivative $\hat{\dot{\boldsymbol{y}}}$ We can then compute a desired acceleration for this feature using a proportional-derivative (PD) controller

$$\hat{\ddot{\boldsymbol{y}}}_t = k_p \left(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_{t+h}\right) + k_d \left(\hat{\dot{\boldsymbol{y}}}_t - \boldsymbol{y}_{\dot{t+h}}\right) \tag{3.30}$$

Where k_p is the proportional term and k_d is the derivative term. We normally set $k_d = 2\sqrt{k_p}$. We then set up the objective function to minimize the difference between the desired feature acceleration $\hat{\ddot{\boldsymbol{y}}}_t$ and the actual feature acceleration $\ddot{\boldsymbol{y}}_t$.

$$E(\boldsymbol{x}) = \|\hat{\ddot{\boldsymbol{y}}}_t - \ddot{\boldsymbol{y}}_t\|^2 \tag{3.31}$$

Transforming this into our optimization form we have

$$\boldsymbol{A}_{\text{feature}} = \begin{bmatrix} \left(\boldsymbol{I}_{3x3} + h^2 k_p + h k_d\right) \boldsymbol{J}_t^y & \boldsymbol{0} \end{bmatrix} \tag{3.32}$$

$$\boldsymbol{b}_{\text{feature}} = k_p \left(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t\right) + k_d \left(\hat{\dot{\boldsymbol{y}}}_t - \dot{\boldsymbol{y}}_t\right) - h k_p \dot{\boldsymbol{y}}_t - \left(\boldsymbol{I}_{3x3} + h^2 k_p + h k_d\right) \dot{\boldsymbol{J}}_t^y \dot{\boldsymbol{q}}_t \tag{3.33}$$

Center of mass tracking

The center of mass tracking objective tries to match the character center of mass position, velocity, and acceleration to some specified values. This is done by regulating the character's linear momentum time derivative $\dot{\boldsymbol{L}}_t$ using a semi-implicit proportional-derivative (PD) controller with a feed-forward term. The output of the PD-controller is

$$\hat{\dot{\boldsymbol{L}}}_t = m \left(k_p \left(\hat{\boldsymbol{c}}_t - \boldsymbol{c}_{t+h}\right) + k_d \left(\hat{\dot{\boldsymbol{c}}}_t - \dot{\boldsymbol{c}}_{t+h}\right) + \hat{\ddot{\boldsymbol{c}}}_t\right) \tag{3.34}$$

Where c_t is the center of mass at time t, $\hat{c}_t$ is the desired center of mass at t, k_p is the proportional gain value and k_d is the derivative gain value. This is essentially the same as our standard feature tracking except it has a feed forward term ($\hat{\ddot{\boldsymbol{c}}}_t$) and is multiplied by a constant (m). As such its optimization form is similar

$$\boldsymbol{A}_{\text{cm}} = \begin{bmatrix} \left(\boldsymbol{I}_{3x3} + h^2 m k_p + h m k_d\right) \boldsymbol{J}_t^c & \boldsymbol{0} \end{bmatrix} \tag{3.35}$$

$$\boldsymbol{b}_{\text{cm}} = m \left(k_p \left(\hat{\boldsymbol{c}}_t - \boldsymbol{c}_t\right) + k_d \left(\hat{\dot{\boldsymbol{c}}}_t - \dot{\boldsymbol{c}}_t\right) + \hat{\ddot{\boldsymbol{c}}}\right) - h m k_p \dot{\boldsymbol{y}}_t - \left(\boldsymbol{I}_{3x3} + h^2 m k_p + h m k_d\right) \dot{\boldsymbol{J}}_t^y \dot{\boldsymbol{q}}_t \tag{3.36}$$

where $\boldsymbol{J}_t^c$ is the Jacobian relating the change in joint positions $\boldsymbol{q}$ to changes in the center of mass.

Angular Momentum Tracking

The final feature we control is angular momentum. Angular momentum control is obviously vital for rotational motions such as rolling or flipping, but it is also important for everyday motions such as walking and balancing. When walking we swing our arms to keep our angular momentum at zero, and when a balancing person is pushed they minimize their linear momentum by inducing some angular momentum to stay in balance. The angular momentum ($\boldsymbol{H}$) is tracked just like the other features. We start with an objective function for the optimization

$$E_{\boldsymbol{H}}(\boldsymbol{x}) = \|\hat{\dot{\boldsymbol{H}}}_t - \dot{\boldsymbol{H}}_t\|^2 \tag{3.37}$$

Then we define an equation for $\hat{\dot{\boldsymbol{H}}}_t$ using a PD controller and some specified desired angular velocity $\hat{\boldsymbol{\omega}}_t$ and angular acceleration $\hat{\dot{\boldsymbol{\omega}}}_t$

$$\hat{\dot{\boldsymbol{H}}}_t = \boldsymbol{I}(\boldsymbol{q}) \left(k_d \left(\hat{\boldsymbol{\omega}}_t - \boldsymbol{\omega}_t \right) + \hat{\dot{\boldsymbol{\omega}}} \right) \tag{3.38}$$

where $\boldsymbol{I}(\boldsymbol{q})$ is the moment of inertia of the multibody. Note that there is no positional term as there was in the linear momentum case. The center of mass is a useful measure of the overall position of a multibody, however there is no such simple measure for overall orientation of a multibody. To perform a good rotational motion however we need such a measure. For example if the character is rolling forward we would want to control the orientation of the sagittal plane with respect to the world to keep the rolling character "upright" and also rolling in the same direction with respect to the world. This value which we call angular excursion and denote with $\boldsymbol{\theta}$ will be discussed in more detail in Section 3.3.

With this measure for full body orientation we can add a proportional term to Equation 3.38:

$$\hat{\dot{\boldsymbol{H}}}_t = \boldsymbol{I}(\boldsymbol{q}) \left(k_s \left(\hat{\boldsymbol{\theta}}_t - \boldsymbol{\theta}_t \right) + k_d \left(\hat{\boldsymbol{\omega}}_t - \boldsymbol{\omega}_t \right) + \hat{\dot{\boldsymbol{\omega}}}_t \right) \tag{3.39}$$

The control of the features angular momentum, linear momentum and body position along with pose tracking allow us to guide a character through a wide variety of activities. As shown in [2] and [27] it is able to very robustly balance a character, and [8] showed that with some higher level planning it is able to drive characters performing various locomotion tasks. The controller is also independent of the morphology of the character.

3.3 Angular Excursion

As mentioned in Section 3.2.1, we need a measure for full body absolute orientation. There is no simple analog to center of mass for the orientation of a multibody. We could look at just the orientation of the trunk, but this excludes orientation information about the rest of the limbs. This could cause problems when, for example, the character tucks their legs in to perform some rotational motion like a dive. We could also try generalizing center of mass to orientation by averaging the orientation of each body part multiplied by it's moment of inertia, however unlike mass, the moment of inertia contribution by each body part varies based on the pose of the character.

To solve this problem we use an approach similar to that of [55] where they attempted to keep the total integral of the angular momentum at zero. We instead use the integral of the character's full body angular velocity (that is, the inverse of the full body moment of inertia multiplied by it's angular momentum). We use the integral of angular velocity so that the rotation value is not susceptible to changes in the character's inertia. For example an ice skater performing an Axel jump might tuck their arms in to increase their angular velocity though their angular momentum is constant. We call this value the angular excursion.

The angular excursion can be defined mathematically as

$$\boldsymbol{\theta} = \int_0^T \boldsymbol{I}(\boldsymbol{q}_t)^{-1} \boldsymbol{H}_t \,\mathrm{d}x \tag{3.40}$$

We estimate the value of this integral using Riemann sums

$$\boldsymbol{\theta} \approx h \sum_{i=1}^{N} \boldsymbol{I}(\boldsymbol{q}_i)^{-1} \boldsymbol{H}_i \tag{3.41}$$

This measure of orientation takes into account the current pose so it remains accurate even if the character undergoes large changes in inertia. However since this estimate is accumulated over time, the error will also accumulate. This can cause a drift in the orientation with respect to the character. Our examples are relatively short (20 seconds or less) so this issue did not cause noticeable problems. Another issue is it is possible to change our apparent orientation without any angular velocity. A well known example is a cat being able to reorient its self to land on its feet. An option for dealing with this is to have a set of poses with known orientations and when the character is close to one of these poses re-initialize the orientation based on the known orientation of the pose.

Chapter 4

Contact Rich Rotational Motions

In this chapter we describe our system for controlling contact rich rotational motions. This includes mainly rolling motions though each component could be applied to different animations. To perform a roll we need to control the orientation of the character about each axis independently as well as using the many varying contact points to keep the character rolling forwards.

This system combines the momentum control method described in [27] with a novel contact selection scheme to enable automatic contact transitions in motions where contacts are constantly changing. The orientation control uses the angular momentum and angular excursion control described in Sections 3.2 and 3.3. We will explain how this is used to control a rolling character in Section 4.1. The contact management system we use selects a subset of the contacts to actively use in the control of the character and is described in Section 4.3.

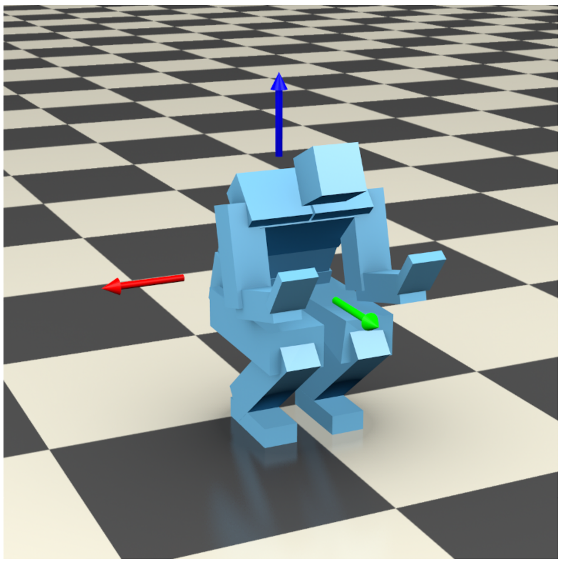
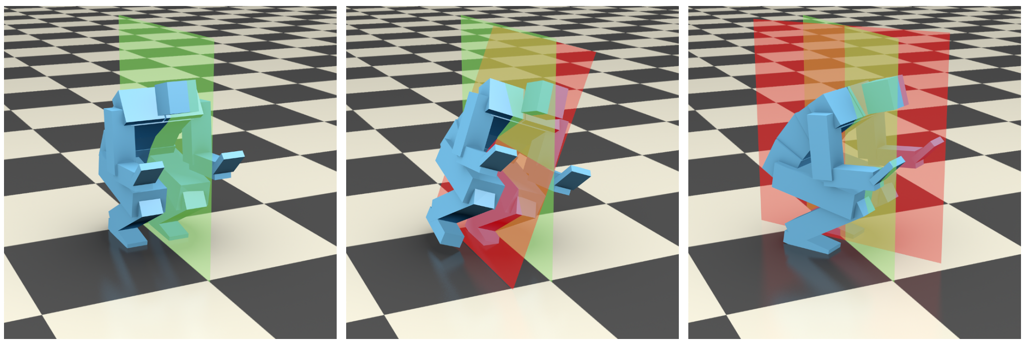


Figure 4.1: The axes we use for the rolling character. The red, green and blue arrows represent the X, Y and Z axes respectively

4.1 Rotation Control

The rolling controller we use is conceptually quite simple. The high level description is to maintain the orientation of the character's sagittal plane while having a constant forwards linear velocity a constant angular velocity about the sagittal axis. If the Z-axis is the up direction and the character is facing in the positive Y direction and wants to roll forwards then this corresponds to the goals of having zero angular excursion about the Z and Y axes while having a positive linear velocity in the Y direction and a negative angular velocity about the X axis. This can be seen in Figure 4.1.

The reason for having a forward velocity and a angular velocity are because that is the direction we want the character to roll and the axis we wan to roll about (for backwards rolling we would simply have the velocities in the opposite directions). We also want to maintain the excursion about the axes we aren't rotating about to keep the character "upright" while rolling. We want the character to roll forwards but not fall over



(a) Correct excursion (b) Incorrect Y excursion (c) Incorrect Z excursion

Figure 4.2: Differences between the desired (green) and actual (red) orientation of the sagittal plane for a forward roll.

sideways or turn so that it starts rolling on its side. These two situations are shown in Figure 4.2.

4.2 Pose Control

We can give the character momentum goals that tell it to roll, but if it is not in a suitable pose it will likely fail or produce an unnatural looking motion. To perform a rolling motion we need to give the character a relatively round shaped pose. A rounded pose will make rolling possible, however it will also needs the freedom of movement to apply some force to induce and control the rolling motion. For humans one option is to have the character in a tight squatting pose with the hands held up near the head. This allows the character to push with its legs in the upright portion of the roll and then to push with its hands when rolling over the head.

A single pose may be enough to perform a successful roll of the weight for the pose goal is not too high, however when a human rolls they usually go through several different poses over the course of the roll. We use two different methods to account for this. One way is to specify several poses for the rolling motion and let the optimization choose the most appropriate one using the blend pose tracker described in Section 3.2.1. This gives the optimization more freedom in the pose it chooses for a particular time in the roll.

The other method we use is to take an example roll animation and index each frame based on the excursion of the character in the animation. Then when simulating a roll we choose the pose that is closest to the simulated character's current excursion. This is useful if we want to simulate a particular style of roll from an example. We can also use the blend tracker in this case by choosing two poses, one slightly ahead in the animation and one slightly behind. This gives the character some freedom to adapt its pose while still following the style of the example animation.

4.3 Contact Management

As described in Section 3.1, the low level controller we use is only able to deal with fixed contact points. Furthermore the contact constraints try to maintain the set of contacts at the current time step. For motions such as walking where the contacts are fairly predictable we can simply disable contacts when we know they are no longer necessary. However in motions where contact changes are more unpredictable pose a problem.

To solve this problem we use the fact that in the case of many contacts, only a small subset of the contacts may be actually necessary to achieve a particular change in momentum. In fact if we ignore torque about the vertical axis then any momentum change can be achieved with just three contact points. Those three points are whichever points enclose the center of pressure necessary to achieve the desired change in momentum.

We could use the desired linear and angular momentum change given in Equations 3.34 and 3.38 to calculate the center of pressure and then only use the minimum set of contacts that enclose that point.

However humans do not do this, they tend to distribute force applied over as many contact points as possible. So instead of using this minimum set we use a heuristic that gives an importance value to each contact point. Then in the control optimization step we only enable the most important contacts (note that this is only during the control optimization, during the forward dynamics simulation all contact points are used). We experimented with several different heuristics and found that the specific heuristic used did not make a substantial different as long as it was reasonable and threw out clearly unhelpful contacts.

The first heuristic we tried was selecting contacts based on how far they were from the desired center of pressure. The idea being that close contact points can contribute more to the desired motion of the character than further contact points. One downside of this heuristic is that the center of pressure we calculate is only based on the momentum objectives of the character and does not take into account other objectives like the desired pose.

This led us to try another simple heuristic which was to simply run a single step of the multi-objective controller with all contact points and use the magnitude of the contact forces it produces as the importance value of each contact. Since the optimization contains a regularization term it won't apply a force at a contact point unless it is truly beneficial to do so. This heuristic naturally takes into account all the objectives since it's running the full optimization. However this method is very slow since solving the quadratic optimization takes the majority of the time, and the more contact points that are included the slower it is. We ended up using this method for all our examples due to its simplicity, however when comparing it with the center of pressure distance method the differences were small.

Chapter 5

Fall Trajectory Optimization

In this chapter we describe our method for creating physically simulated falling animations. The system is built on the low level controller described in Section 3.2 to create a general parameterized falling controller. We then use a stochastic optimization to minimize a set of weighted error measures to come up with the parameters to produce a high quality fall. We induce a fall by applying a large external force to the character over a short period of time. In Section 5.1 we describe the general controller we use in our experiments. Section 5.2 describes the different objective functions we use to evaluate the quality of a fall. Then in Section 5.3 we describe the stochastic optimization technique we use to evaluate the parameter values for the fall controller.

5.1 Fall Controller

During a fall there are a variety of prevention and mitigation strategies that can be employed. Several examples from robotics are reviewed in Section 2.2.1 and some from biomechanics are reviewed in Section 2.3. In this work we focus on damage mitigation for unrecoverable falls. Example damage mitigation motions from previous works include bracing with your hands or other shock absorbing parts [19, 24, 13, 25], lowering your center of mass to reduce impact velocity [13, 34], as well as modifying the fall direction to avoid obstacles [58, 31]. Our fall controller attempts to generalize these strategies so that we can achieve any one of them by adjusting the parameters of one general controller. We can then use a global optimization to find the best parameters for a fall in a particular situation.

The general falling controller works on top of the low level controller by specifying trajectories for the controllable features exposed by the low level controller. The trajectories are parameterized as cubic splines with a fixed number of control points. The parameters are then the values of each degree of freedom of the spline's control points as well as the end time of each spline. We could have splines for every feature controllable by the low level controller, however that would give us many parameters (more than the degrees of freedom of the character) and we would be unlikely to find a good solution in that space. Instead we parameterize some features depending on the type of fall we want to generate. The features we control are the center of mass, the angular excursion, and the pose.

The center of mass trajectory is a useful generalization of many motions. Following a specific center of mass trajectory is also at the core of many fall strategies such as the Ukemi strategy [34]. The Ukemi strategy only controls the center of mass in vertical direction and the direction of the fall so we could use a 2 dimensional spline for those two direction. However if we are trying to avoid obstacles we would need to control the full 3 degrees of freedom of the center of mass. We could similarly control the center of mass velocity or acceleration however center of mass position is more intuitive and the positional feedback of controlling the center of mass gives more precise control. We can similarly control the angular excursion, angular velocity, and angular acceleration of the character with splines.

Controlling the pose is somewhat more problematic than the center of mass and angular excursion. Controlling a spline for each degree of freedom of the pose would make finding a good solution take far too long and likely reduce in unrealistic looking animations. Instead we use a set of poses and the blend pose tracker to allow the system to vary its pose while not exceeding human joint limits. This gives the system some full body pose control, however when falling we often need more control over specific body parts, such as the arms, to break a fall. So in addition to the blend pose tracker we use splines to control some individual joint angle targets.

One final component of the fall controller is the initial "reflex phase" described in [1]. When humans receive an unexpected push they induce an angular momentum to rotate with the push to attempt to mitigate the push. This is a reflexive reaction to the impulse before the character has had time to choose how to decide whether to attempt some kind of recover or brace for a fall. We implement this in the low level controller with the objective of trying keep the linear momentum near zero. To achieve this objective the character must induce some angular momentum in reaction to the push. This phase only lasts the duration of human reaction time but results in a very lifelike initial reaction to the impulse.

5.2 Fall Error Measures

To optimize the parameters of a fall controller we need an objective function to measure the quality of a fall. We use a weighted sum of two terms for our objective function. The first term is designed to measure the quality of a fall and the second is regularization term designed to ensure the system does not cheat and produce an unrealistic or infeasible motion.

The robotics literature presents a variety of different fall quality measures. Many of them involve minimizing force at the contact points or in the joints [12, 11, 41] while others minimized the velocity of the center of mass at impact [34]. Force at the point of impact was also used in all the biomechanics literature we reviewed. We experimented with using maximum impact force, maximum velocity at impact, and maximum jerk at impact and empirically found maximum impact force to produce better looking results. Since impact force is also preferred in the literature this is what we used for our results. Specifically we added up the maximum contact force felt by each body part and used that as our main measure of fall quality.

We experimented with several different regularization terms to keep the character from coming up with an extreme and unrealistic falling strategy. We tested penalizing high joint torques to prevent extreme movements. There were also issues with the character jumping so high that it wouldn't hit the ground before the simulation ended so we tried penalizing final center of mass velocity and final center of mass height. In the end we simply penalized the sum of all the ground reaction forces. Since all external forces come from pushing off the ground this prevents the character from being able to perform anything too extreme.

5.3 Trajectory Optimization

Now we have the fall controller and objective function defined we simply apply a gradient free global optimization technique to find the best parameter values to generate the best fall for a given situation. We use Covariance Matrix Adaptation Evolution strategy (CMA-ES) [18] to perform this optimization. CMA-ES is a stochastic global optimization algorithm that iteratively updates the mean and covariance matrix of a normal distribution. It first samples a set of points from the current normal distribution. It then updates the mean to be the objective-function value weighted mean of the sample points. Next the covariance matrix is updated to estimate the inverse of the Hessian matrix at the mean. These iterations proceed until some specified termination condition such as the objective function value improving too little or the covariance becoming too small. You can then choose the result to be either the current mean or the lowest valued sample point.

We use CMA-ES because it is simple to implement, does not require derivatives and has good convergence properties. It also works well on even ill-conditioned problems with many local minima. This is important because our objective function involves solving complex contacts between the character and the ground so small changes in the parameter values could cause large changes in the resulting motion. CMA-ES has also been used in many previous works in animation and has been shown to work well on these types of optimization problems [30, 46, 49, 51, 52, 50, 54].

Chapter 6

Results and Discussion

All our experiments were run using a semi-implicit multibody simulator with both the simulation and control optimization running at $60hz$. In Section 6.1 we present the experiments and results obtained from our contact rich rolling controller. In Section 6.2 we present the result of our falling trajectory optimization.

6.1 Rolling Results

We tested our rolling controller with forwards, backwards, and sideways rolls on various different kinds of terrain. We also experimented using key framed poses and poses derived from motion capture.

6.1.1 Pose Based Rolling

To test our rolling controller without any motion capture data we created two poses for each type of roll and let the system choose the most appropriate blend between these poses using our blending pose tracker. We found it was possible for the controller to roll with only one pose, however the resulting animation looked unrealistic since it

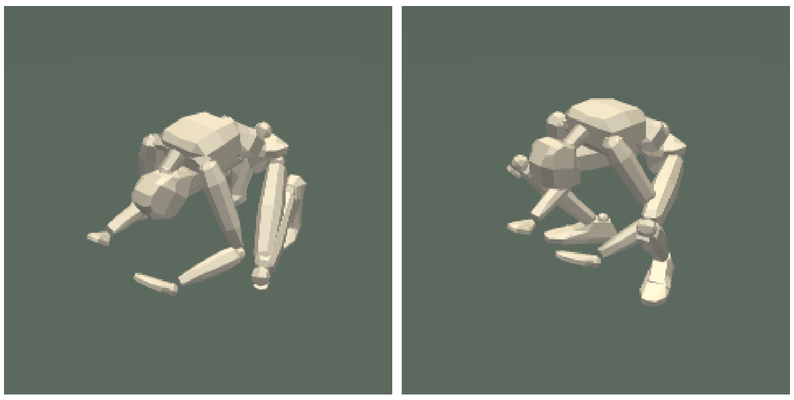


Figure 6.1: Poses used for the forwards posed based roll

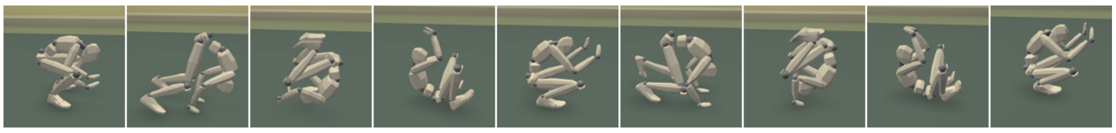


Figure 6.2: A sequence of images from a simulation of a forward roll with keyframed base poses

deviated very little from that single pose. All the poses used were created by hand using the Blender 3D graphics software.

Forwards Rolling: The poses used for the forward roll are shown in Figure 6.1. The first pose has the hands out in front of the head in order to protect the head when rolling forwards over the head. The second pose is very similar but has the hands down near the feet and the legs slightly extended. This is so that blending between these poses allows the character to adjust the height of the hands and the extension of the legs while rolling. Figure 6.2 shows some images from the resulting simulated roll. Once the character gets going with the roll it doesn't adjust the pose much except to push off the ground a bit with its legs.

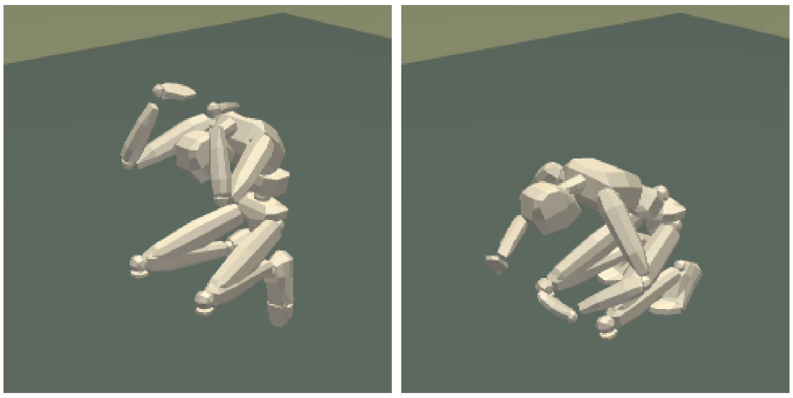


Figure 6.3: Poses used for the backwards posed based roll

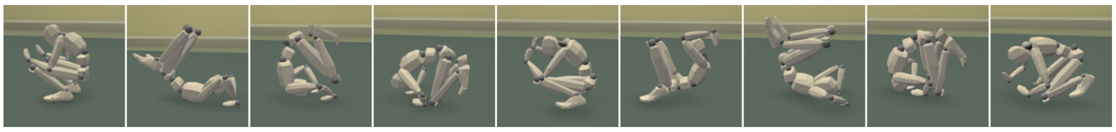


Figure 6.4: A sequence of images from a simulation of a backward roll with keyframed base poses

Backwards Rolling: The backwards roll poses are shown in Figure 6.3. The two poses vary quite a bit more than the forwards roll. The first pose has the hands behind the head with the knees bent and pointing perpendicular to the abdomen. This is for when the character is rolling off its feet and onto its back. The configuration of the legs lets the feet push off the ground as long as possible and the hands are behind the head to make sure they hit the ground before the head. The second pose has the knees tucked closer to the chest and the hands out in front. This is intended to keep the hands on the ground after the character rolls over its head so it can push off the ground to continue the roll. A sequence of frames from a simulation of the backwards roll is shown in Figure 6.4. The optimization consistently chooses the first pose for rolling over the back and then transitions into the second pose as it rolls over the head and back onto the feet.

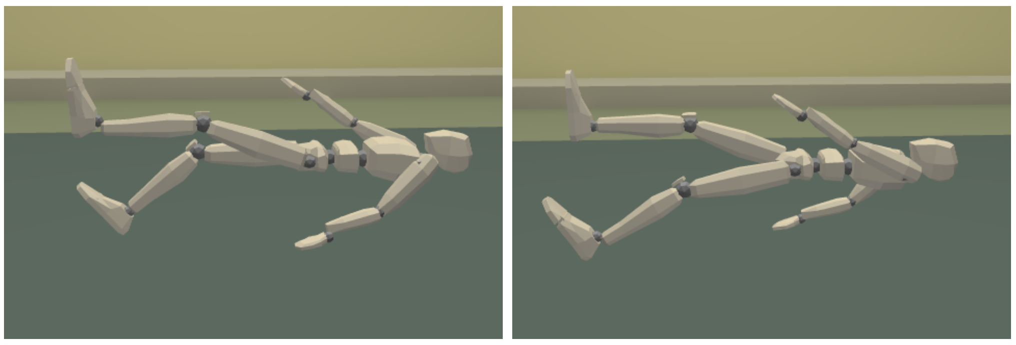


Figure 6.5: Poses used for the sideways posed based roll

Sideways Rolling: The poses used for the sideways pose based rolling are shown in Figure 6.5. The two poses are essentially mirror images of each other, each having one arm and the opposite leg angled upwards, and the other arm and leg angled down. This was intended to let the character push with its arms and legs when on its back and its stomach. We were unable to get robust or consistent rolling behavior with the pose based sideways roll. The character wold often get stuck on its back or stomach while trying to roll over. This is likely because of the he shape of the character is more oblong when rolling sideways so a more careful coordination of poses is needed for successful sideways rolling.

6.1.2 Motion Capture Rolling

To test the motion capture based rolling we used three different motion capture clips to produce two different forwards roll styles and one sideways roll. As mentioned in Section 4.2 we still use the blend pose tracker to give the character some freedom in pose, however the two poses it can blend between are based on the current phase of the roll, one slightly ahead of the current phase of the roll and one slightly behind. We found that using poses that were roughly 10% ahead and 10% behind the current phase worked well and that the actual amount was not particularly important. Overall we found the

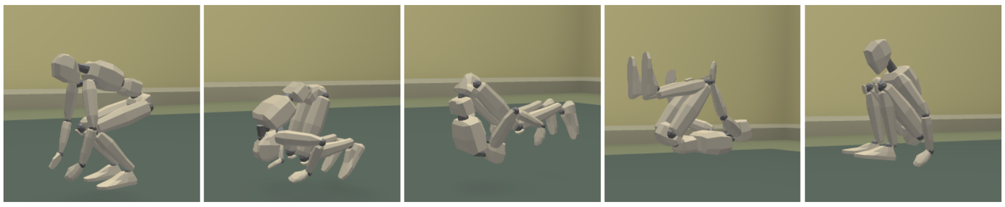


Figure 6.6: A sequence of poses form the forward roll reference motion

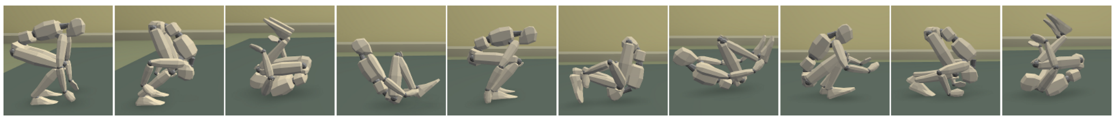


Figure 6.7: A sequence of images from a simulation of a forward roll based on motion capture

motion capture based rolls to be both more robust and more realistic looking than the pose based rolls.

Symmetric Forward Roll: We first tested the motion capture based rolling on a symmetric over the head forwards roll. The motion capture data was slightly modified in order to make it symmetric and cyclic. Figure 6.6 shows several frames from the reference motion. Compared to the poses used in the pose based forwards roll it has the knees closer together and a more pronounced push off with the legs. Figure 6.7 shows a sequence of images of the simulated roll using this reference motion. The simulated motion maintains the look of the reference motion while still being reactive to environment. Compared with the pose based forwards roll this goes through a much larger range of poses over the course of the roll. The resulting roll is much more natural looking than the pose based one.

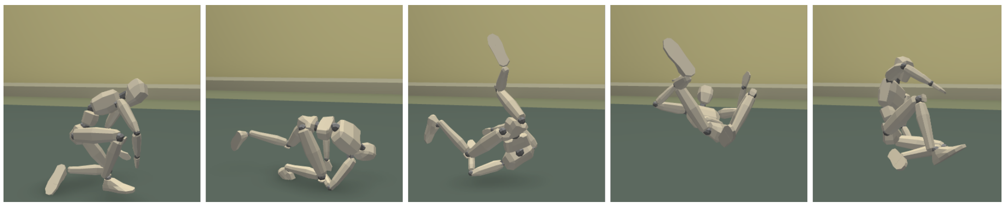


Figure 6.8: A sequence of poses form the parkour roll reference motion

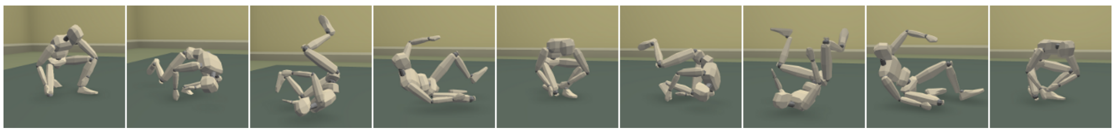


Figure 6.9: Frames of a parkour-style roll simulated from motion capture data

Non-symmetric Forward Roll: The simulations shown thus far have all been using symmetric poses or reference motions. However this is not a requirement. We also tested our controller with a non-symmetric forward roll over the shoulder which we call the "parkour" roll. Figure 6.8 shows some frames from the parkour reference motion and Figure 6.9 shows the resulting simulated roll. This is the type of roll performed by martial artists to minimize damage from a fall. This roll is overall smoother than the others partly because it avoids the difficulty of rolling over the head. The skill and smoothness of the reference motion also contributes to the quality of the resulting simulation.

Sideways Roll: The third reference motion we tested was a sideways log roll. Figure 6.10 shows the reference motion used for this experiment. Unlike the pose based sideways roll this one has the arms above the head which gives the arms more freedom to push on the ground. The model for this motion also has shoulder joints which are very helpful in preventing the character from getting stuck. The simulated sideways roll is shown in Figure 6.11.

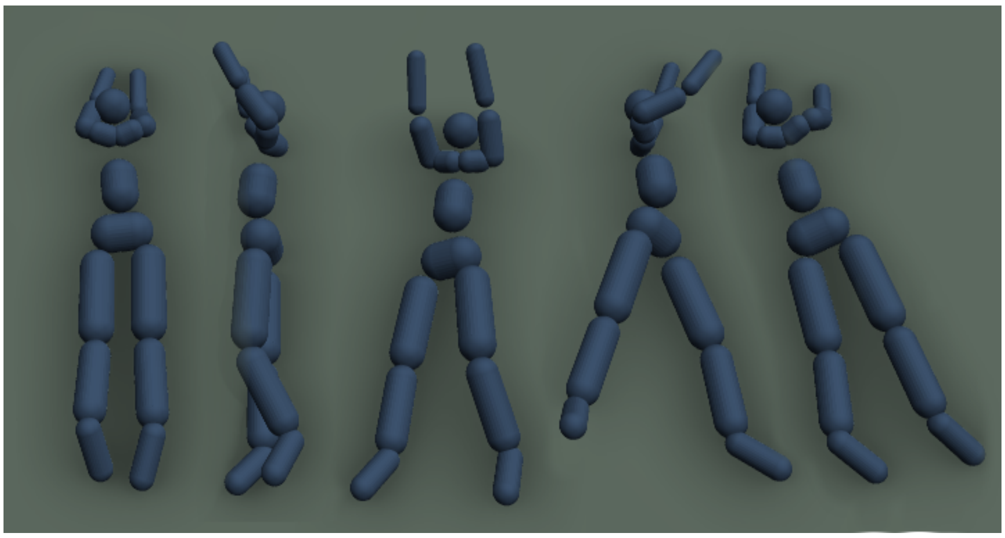


Figure 6.10: A sample of the poses from the sideways roll reference motion

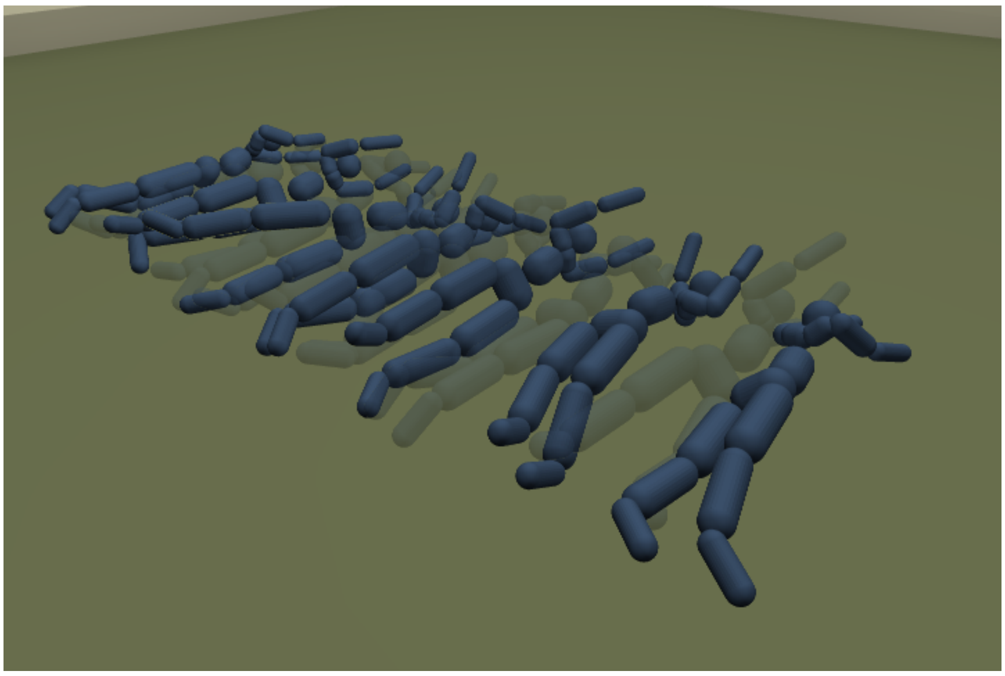


Figure 6.11: Progression of a sideways roll simulated from motion capture data

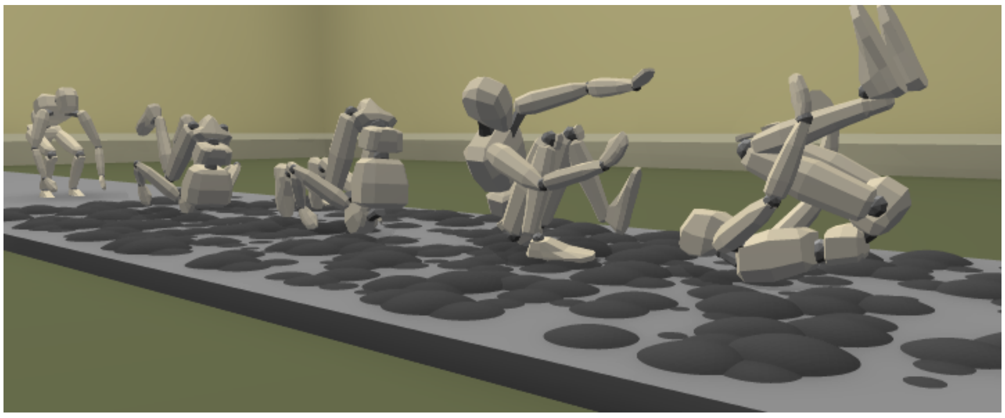


Figure 6.12: The forwards reference motion roll on rough terrain

6.1.3 Terrain Changes

Our rolling controller makes no assumptions about the environment and simply uses the contacts it has available to try and make forward progress. As such, it is able to work not just on level ground but in a wide variety of environments. To test the robustness of this control method we experimented on using the rolling controller over several different kinds of terrain.

For our first example we tested the forwards motion capture based roll on very uneven, but overall level ground. The rough ground was created with 30 centimeter radius spheres placed in a grid at 15 centimeter intervals. The spheres were placed mostly below the ground, each protruding form 0 to 10 centimeters above the ground to make frequent small bumps in the flat ground. Figure 6.12 shows the rough terrain and simulated roll over the terrain. The character is thrown off balance by the uneven terrain as can be seen in the 4th frame of Figure 6.12, however it is able to recover and continue rolling without much trouble.

We also tested the forwards roll on slanted surfaces. Figure 6.13 shows the roll

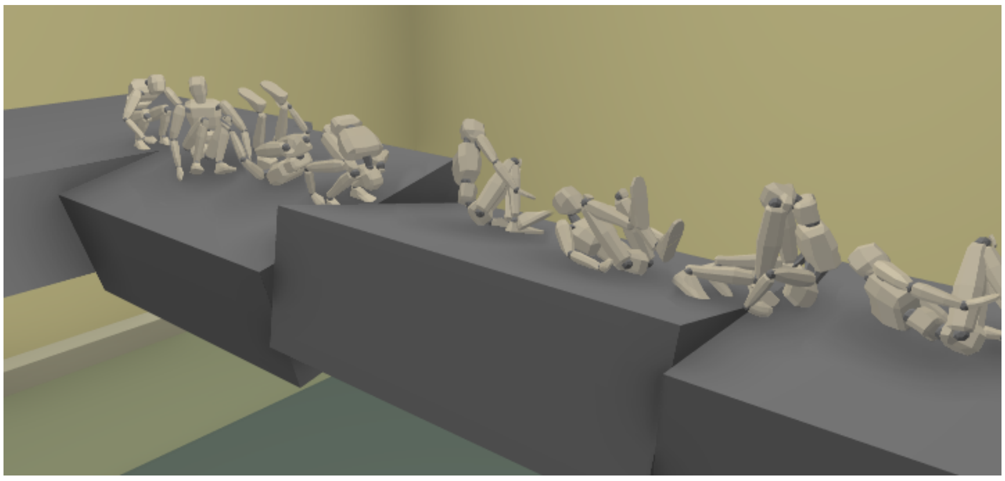


Figure 6.13: The forwards reference roll on terrain with varying slants

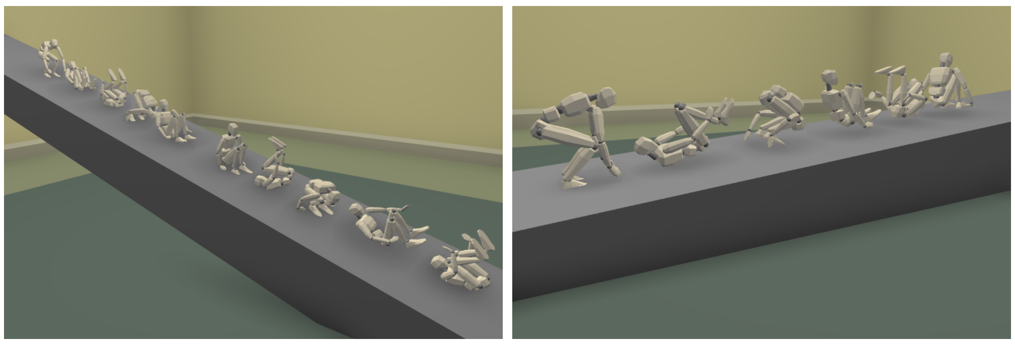


Figure 6.14: Rolling down a 30 degree decline and up a 10 degree incline

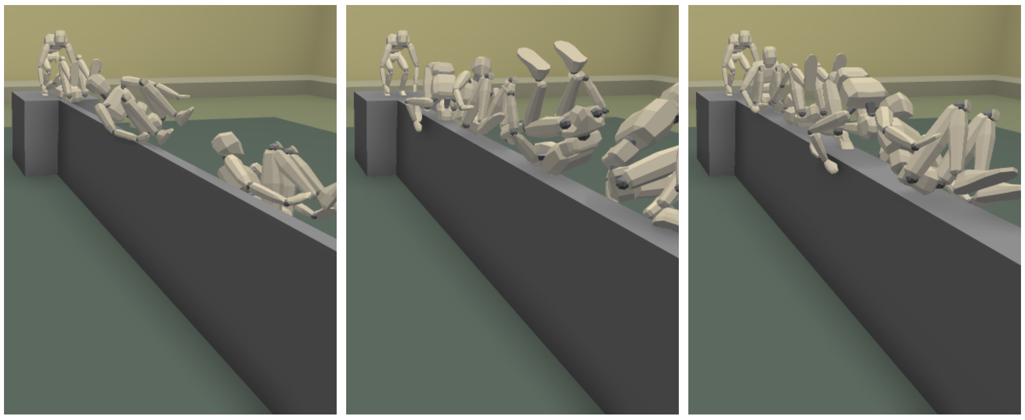


Figure 6.15: The forwards reference roll on 10, 20, and 30 centimeter balance beams

on different lateral slants. The lateral position objective and angular excursion objectives keep the character rolling relatively straight even as it transitions from one slant to another. The character was easily able to roll on 30 degree lateral slants. Figure 6.14 shows the roll on a 12 degree decline and a 5 degree incline. The controller could roll indefinitely on the steep downhill however the uphill roll would often get stuck after three rolls. On steeper declines the controller is unable to control the speed of the roll and eventually gets out of control.

As a third test for the forwards roll we had the character roll on balance beams of various widths. Figure 6.15 shows the forwards roll on three different beam thicknesses. On a 10 centimeter wide balance beam the controller was only able to complete two very unstable rolls before falling off. On the 20 centimeter beam the controller is able to complete three rolls before falling and on a 30 centimeter was able to roll almost indefinitely. Part of the reason the controller can roll on such thin beams is when the character's hands make contact with the side of the beam the controller can then use that contact to keep the character balanced on the beam.

We also test the sideways roll down a steep staircase. Since the sideways roll

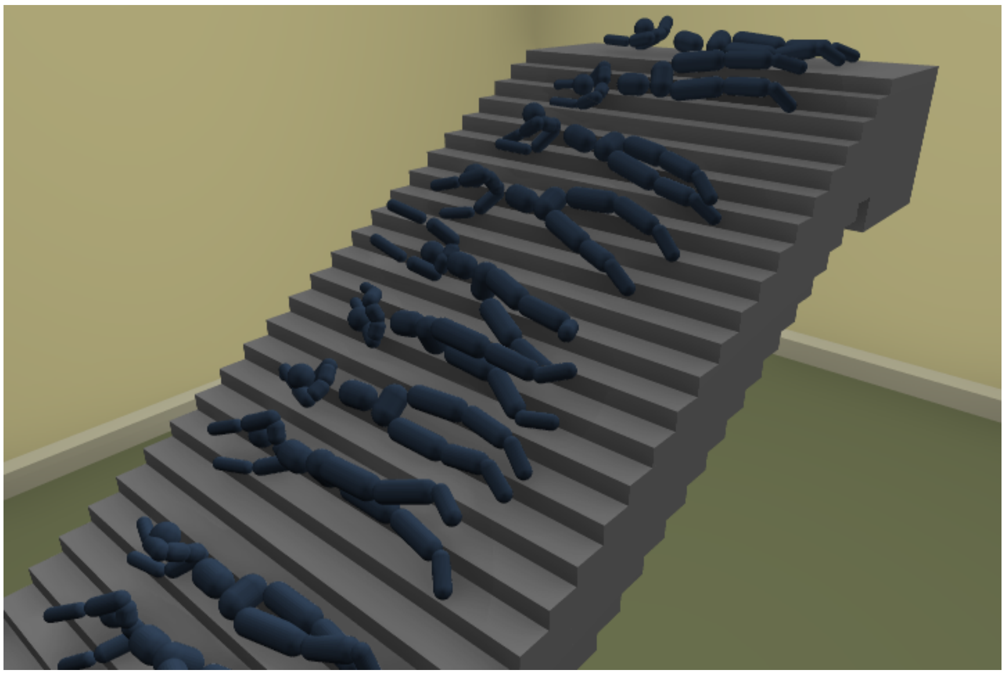


Figure 6.16: The sideways reference roll going down a staircase

is slower and has a smaller inertia, its speed can be more easily controlled and it can roll down much steeper declines. Figure 6.16 shows the sideways roll on a staircase with stairs 10 centimeters high and 20 centimeters wide.

6.2 Falling Results

To improve the speed of our simulations and trajectory optimization we used to simplified models for our falling results. The first model is a simple 5 link chain with links representing the foot, calf, thigh, abdomen, and head. The second model is similar except it also included two 2-link arms. A limitation of the simulator used is that all joints must be ball joints however this is really only an issue at the knee joint as the ankle, hip and neck joints can be reasonably approximated as ball joints. As mentioned in Section 5.1 we experimented with several different fall error measures and maximum impact force empirically produced the most visually pleasing results. We also penalized forces on the

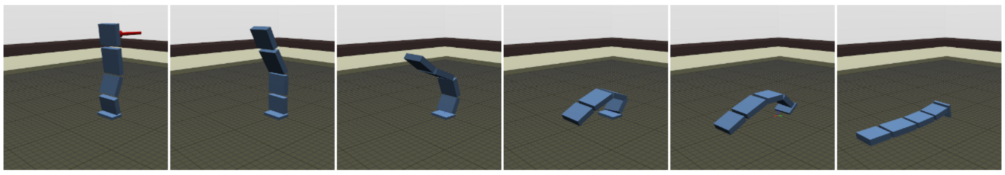


Figure 6.17: An fall controlling the center of mass that emulates the Ukemi strategy

head 10 times more than other body parts. Without this the character would fall on the head since that is the easiest body part to get in contact with the ground first.

To induce all our falls we applied a 400 N lateral force to the center of mass of the character's head for 100 ms. For the CMA-ES optimization we run it until the step size is less than 10^-3 or the change in function value is less than 1. The optimization would generally converge within 1000 iterations. To avoid local minima we ran the optimization with three different random initial states. The random restarts usually produced very similar results. Each optimization takes about 16 hours since the low level controller runs at about half real time and each fall is run for 2 seconds of simulation time.

For a basic test we optimized a fall from a forward push while only controlling the center of mass trajectory with a three control point cubic splines giving us a total of 9 parameters. Figure 6.17 shows the resulting optimized fall. The resulting fall is very similar to the Ukemi strategy mentioned in the robotics literature [34]. It starts with a crouching motion and just before the head hits the ground it pushes off propelling the character forwards to slightly reduce its downwards velocity and distribute the impact over more body parts.

We next tried adding in control of the angular excursion of the character. The angular excursion is also controlled with a three point cubic spline, increasing the number

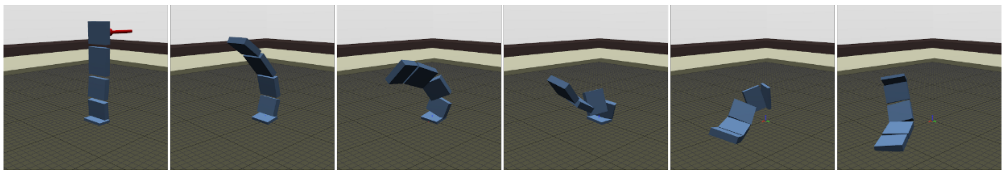


Figure 6.18: A forwards fall controlling the center of mass and angular excursion

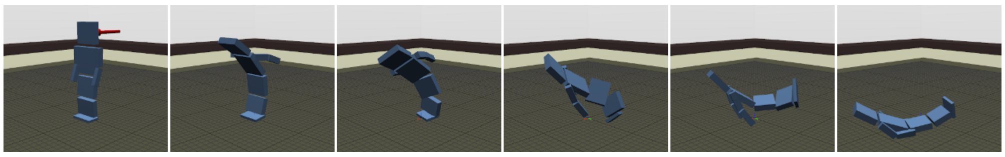


Figure 6.19: A forwards fall controlling the center of mass and angular excursion with arms

of parameters of the optimization to 18. Figure 6.18 shows the resulting fall from this optimization. Instead of the Ukemi strategy the character uses the angular excursion control to twist around and land on its more curved back. This makes for a much more softer landing with a slight roll along the back.

The previous two experiments demonstrate the complex motion that can result from just very basic control parameters. However a real person is likely to use their arms to aid in breaking a fall. We can control the arms again using cubic splines as described in Section 5.1. In addition we use a single spline to control both arms symmetrically since when breaking a fall humans generally place both arms in the same direction. We also only control the shoulder joint of the character and have the tracker try and maintain a

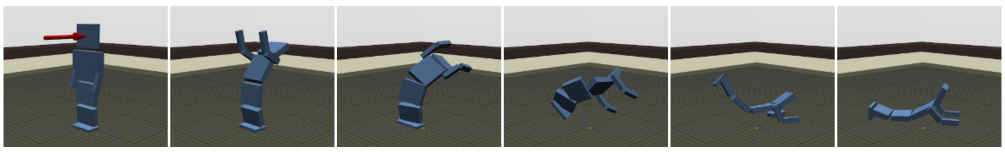


Figure 6.20: A backwards fall controlling the center of mass and angular excursion

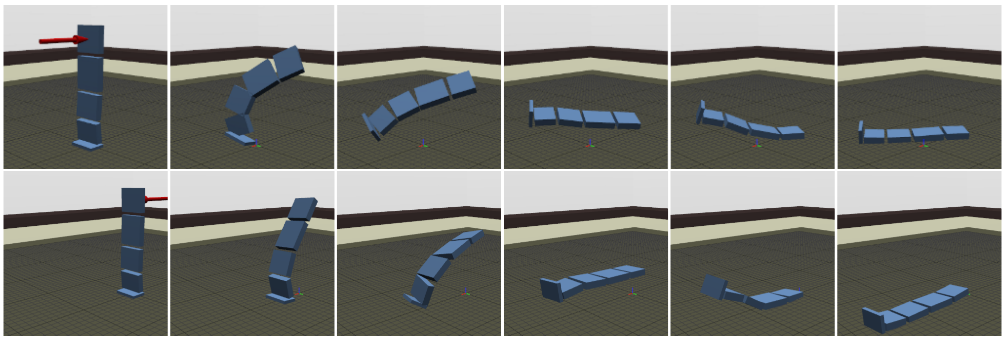


Figure 6.21: Forwards and backwards falls without the reflex phase.

slight bend in the elbows. This allows the character to orient the arms in any direction and break it's fall with fewer parameters. Once again we found that a three point spline was able to give fine control over the arms to produce a convincing fall. The arm control spline combined with the center of mass and angular excursion splines gives us a total of 27 parameters. Figure 6.20 shows the forwards fall using the arms and Figure 6.19 shows the forwards fall. Both falls share a similar structure in that The reflex phase causes each to swing the arms back to rotate with the impact and then twist around to land on the arms first. In the case of the forwards push this swings the arms back behind the character. The character then turns around so its hands hit first to absorb the impact. For the backwards push the arms swing up and the character bends backwards and then turns around to land on the forearms. In both cases the arms make contact with the ground first and are used to reduce the impulse from the impact.

Finally, we performed an optimization without the reflex phase to demonstrate its importance in producing a good fall. Figure 6.21 shows the results of this experiment. Without the reflex phase the character is unable to adequately rotate with the impact and ends up losing contact with the ground. The result is the character almost free falling from standing to landing on its back.

Chapter 7

Conclusion

Creating realistic looking animations can be a very difficult task. We have presented a method for creating falling and rolling animations by controlling physically simulated characters. Since falling and rolling are such dynamic and contact rich motions they can be very difficult to create by hand, making the ability to simulate them a usefully tool. Since these techniques are based on physical simulation they could be applied not only to animation but robotics as well.

To create these controllers we have used a method for measuring and controlling the orientation of a multibody called angular excursion. The angular excursion is intended to be analog of the center of mass for the characters orientation. By controlling it we can get a character to maintain its overall orientation about an axis which is key to performing a roll. It is also very useful in falling as it allows the character to rotate and face a particular direction to break its fall.

We have also introduced a heuristic for determining which contacts are important for performing a task. The multi-objective quadratic programming controller we use needs to know which points should remain in contact at each time step. Since the motions we are simulating have constantly changing contacts, being able to determine which contact points are active at a given point in time is vital to the performance of these controllers.

To demonstrate the effectiveness and robustness our rolling controller we tested it with a variety of different poses and in a variety of different environments. It is able to roll successfully on slopes in various directions and over rough terrain. For the falling controller we trained it using various control methods and for pushes in different directions and it was able to generate interesting falls in most cases.

7.1 Limitations and Future Work

Although these controllers can produce nice animations there are several limitations to them. As is often the case with physics based animation, our system produces nice looking results but cannot be easily controlled to produce the specific results an animator may want. For example we can give the rolling controller a desired speed, however it may not achieve that speed because it must obey the physical constraints of the system. The falling controller has the limitation that it is not very general. Each learned fall technique is tailored to one specific type of fall and applying it to a slightly different fall may not produce good results. Also since it is uses a sampling based optimization it is slow to find the best fall and does not scale well as the dimensionality increases.

These limitations are opportunities for a variety of improvements to this system. It is unlikely, for example, that one would want a character to roll endlessly like our controller does. Developing a way for the roll to transition into some other task like walking or running would be very useful. The falling controller could be improved by not just learning how to react to a specific fall, but a set of feedback rules on how it should react to a variety of falls. If the leaned policy could be used on any type of fall then the long optimization time would not be as much of a drawback. As with the rolling controller it would also be useful if the fall could transition to some other task like rolling or getting up.

Bibliography

[1] Muhammad Abdallah and A. Goswami. A Biomechanically Motivated Two-Phase Strategy for Biped Upright Balance Control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1996–2001. IEEE, 2005.

[2] Yeuhi Abe, Marco da Silva, and Jovan Popović. Multiobjective control with frictional contacts. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 249–258, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[3] Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, pages 231–247, 1997.

[4] David Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10(2):292–352, 1993.

[5] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. *ACM SIGGRAPH Computer Graphics*, 23(3):233–242, July 1989.

[6] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. In *ACM SIGGRAPH 2011 papers on - SIGGRAPH '11*, page 1, New York, New York, USA, 2011. ACM Press.

[7] Marco da Silva, Yeuhi Abe, and Jovan Popović. Interactive simulation of stylized human locomotion. *ACM Trans. Graph.*, 27(3):82:1—-82:10, August 2008.

[8] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Trans. Graph.*, 29(4):131:1—-131:10, July 2010.

[9] Kenny Erleben. *Stable , Robust , and Versatile Multibody Dynamics Animation.* PhD thesis, University of Copenhagen, Denmark, 2005.

[10] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 251–260, New York, NY, USA, 2001. ACM.

[11] Kiyoshi Fujiwara, Shuuji Kajita, Kensuke Harada, Kenji Kaneko, Mitsuharu Morisawa, Fumio Kanehiro, Shinichiro Nakaoka, and Hirohisa Hirukawa. An optimal

planning of falling motions of a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 524–529, December 2006.

[12] Kiyoshi Fujiwara, Shuuji Kajita, Kensuke Harada, Kenji Kaneko, Mitsuharu Morisawa, Fumio Kanehiro, Shinichiro Nakaoka, and Hirohisa Hirukawa. Towards an Optinal Falling Motion for a Humanoid Robot. In *IEEE-RAS International Conference on Humanoid Robots*, pages 456–462, 2007.

[13] Kiyoshi Fujiwara, Fumio Kanehiro, Shuji Kajita, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. UKEMI: falling motion control to minimize damage to biped humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2521–2526, 2002.

[14] Kiyoshi Fujiwara, Fumio Kanehiro, Shuuji Kajita, Kazuhito Yokoi, Hajime Saito, Kensuke Harada, Kkenji Kaneko, and Hirohisa Hirukawa. The first human-size humanoid that can fall over safely and stand-up again. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1920–1926, October 2003.

[15] Kiyoshi Fujiwara, Fumio Kanehiro, W I T A Shuuji, Hmhisa Hirukawa, and S Kajita. Safe knee landing of a human-size humanoid robot while falling forward. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 503–508, September 2004.

[16] Larry Gritz and James K Hahn. Genetic Programming Evolution of Controllers for 3-D Character Animation. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 139–146. Morgan Kaufmann, 1997.

[17] Brenda E. Groen, Vivian Weerdesteyn, and Jacques Duysens. Martial arts fall techniques decrease the impact forces at the hip during sideways falling. *Journal of biomechanics*, 40(2):458–62, January 2007.

[18] Nikolaus Hansen. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, 102(2006):75–102, 2006.

[19] E. T. Hsiao and S. N. Robinovitch. Common protective movements govern unexpected falls from standing height. *Journal of biomechanics*, 31(1):1–9, January 1998.

[20] Sumit Jain and C. Karen Liu. Controlling physics-based characters using soft contacts. *ACM Trans. Graph.*, 30(6):1, December 2011.

[21] Shivaram Kalyanakrishnan and Ambarish Goswami. Predicting Falls of a Humanoid Robot through Machine Learning. In *Innovative Applications of Artificial Intelligence*, 2010.

[22] Shivaram Kalyanakrishnan and Ambarish Goswami. Learning to Predict Humanoid Fall. *International Journal of Humanoid Robotics*, 8(2):245–273, 2011.

[23] J. G. Daniël Karseen and Martijn Wisse. Fall detection in walking robots by multi-way principle component analysis. *Robotica*, 27(8):796–80, 2009.

[24] Kyu-Jung Kim and James a Ashton-Miller. Biomechanics of fall arrest using the upper extremity: age differences. *Clinical Biomechanics*, 18(4):311–318, May 2003.

[25] Sung-Hee. Lee and Ambarish Goswami. Fall on Backpack: Damage Minimizing Humanoid Fall on Targeted Body Segmet Using Momentum Control. *ASME 2011 8th International Conference on Multibody Systems, Nonlinear Dynamics, and Control (MSNDC) inside International Design Engineering Technical Conference (IDETC)*, August 2011.

[26] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. *ACM Trans. Graph.*, 29(4):128:1—-128:10, July 2010.

[27] Adriano Macchietto, Victor Zordan, and Christian R Shelton. Momentum control for balance. *ACM Trans. Graph.*, 28(3):80:1—-80:8, July 2009.

[28] Thomas B. Moeslund and Erik Granum. A Survey of Computer Vision-Based Human Motion Capture. *Computer Vision and Image Understanding*, 81(3):231–268, March 2001.

[29] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, November 2006.

[30] Igor Mordatch, Martin De Lasa, and A Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.*, 2010.

[31] Umashankar Nagarajan and Ambarish Goswami. Generalized Direction Changing Fall Control of Humanoid Robots Among Multiple Objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3316–3322, May 2010.

[32] JT Thomas Ngo and Joe Marks. Spacetime constraints revisited. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 343–350, New York, NY, USA, 1993. ACM.

[33] Nam Nguyen, Nkenge Wheatland, David Brown, Brian Parise, C Karen Liu, and Victor Zordan. Performance capture with physical interaction. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 189–195, 2010.

[34] Kunihiro Ogata, Koji Terada, and Yasuo Kuniyoshi. Falling motion control for humanoid robots while walking. In *7th IEEE-RAS International Conference on Humanoid Robotics*, pages 306–311, November 2007.

[35] Kunihiro Ogata, Koji Terada, and Yasuo Kuniyoshi. Real-time selection and generation of fall damage reduction actions for humanoid robots. In *8th IEEE-RAS International Conference on Humanoid Robots*, pages 233–238. Ieee, December 2008.

[36] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. Mit Press Series in Artificial Intelligence. MIT Press, 1981.

[37] Wieber Pierre-Brice. On the stability of walking systems. In *International Workshop on Humanoid and Human Friendly Robotics*, 2002.

[38] Jerry Pratt, Chee-Meng Chew, Ann Torres, Peter Dilworth, and Gill Pratt. Virtual Model Control: An Intuitive Approach for Bipedal Locomotion. *The International Journal of Robotics Research*, 20(2):129–143, February 2001.

[39] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *ACM SIGGRAPH Computer Graphics*, 25(4):349–358, July 1991.

[40] Reimund Renner and Sven Behnke. Instability Detection and Fall Avoidance for a Humanoid using Attitude Sensors and Reflexes. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2967–2973, October 2006.

[41] J. Ruiz-del Solar, R. Palma-Amestoy, R. Marchant, I. Parra-Tsunekawa, and P. Zegers. Learning to fall: Designing low damage fall sequences for humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8):796–807, July 2009.

[42] Ari Shapiro, Fred Pighin, and Petros Faloutsos. Hybrid control for interactive character animation. *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 455–461, 2003.

[43] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, SIGGRAPH '94, pages 15–22, New York, New York, USA, 1994. ACM Press.

[44] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. *ACM Trans. Graph.*, 26(3), July 2007.

[45] Craig Sunada, Dalila Argaez, Steven Dubowsky, and Constantinos Mavroidis. A coordinated Jacobian transpose control for mobile multi-limbed robotic systems. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1910–1915, 1994.

[46] Jie Tan, Yuting Gu, Greg Turk, and C Karen Liu. Articulated swimming creatures. *ACM Trans. Graph.*, 30(4):58:1—-58:12, July 2011.

[47] Jie Tan, Greg Turk, and C. Karen Liu. Soft body locomotion. *ACM Trans. Graph.*, 31(4):1–11, July 2012.

[48] Bing Tang, Zhigeng Pan, Le Zheng, and Mingmin Zhang. Interactive generation of falling motions. *Computer Animation and Virtual Worlds*, 17(3-4):271–279, July 2006.

[49] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Trans. Graph.*, 28(3):1, July 2009.

[50] Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers. *ACM Trans. Graph.*, 28(5):168:1—-168:8, December 2009.

[51] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29(4):1, July 2010.

[52] Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph.*, 31(4):1–11, July 2012.

[53] Wayne L Wooten. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans.* Phd, Georgia Institute of Technology, 1998.

[54] Jia-chi Wu and Zoran Popović. Terrain-adaptive bipedal locomotion control. *ACM Trans. Graph.*, 29(4):72:1—-72:10, July 2010.

[55] Yuting Ye and C. Karen Liu. Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph.*, page 1, 2010.

[56] KangKang Yin, Stelian Coros, and Philippe Beaudoin. Continuation methods for adapting simulated skills. *ACM Transactions on*, 27(3):1, August 2008.

[57] Kangkang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.*, 26(3):105, July 2007.

[58] Seung-kook Yun, Ambarish Goswami, and Yoshiaki Sakagami. Safe fall: Humanoid robot fall direction change through intelligent stepping and inertia shaping. In *2009 IEEE International Conference on Robotics and Automation*, pages 781–787. IEEE, May 2009.

[59] David Zeltzer. Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, November 1982.

[60] Victor B. Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '02*, page 89, New York, New York, USA, 2002. ACM Press.