Computational Acquisition of Robust Motor Skills

by

Mazen Al Borno

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

© Copyright 2017 by Mazen Al Borno
Humans have the ability to learn complex motor skills like ballet dancing, breakdancing, gymnastics and riding a bicycle. Ideally, these skills are robust to many conditions: they can be performed whether the day is windy or the ground is slippery. This thesis introduces computational tools for physics-based simulated characters to acquire robust motor skills. We show how to optimize control solutions to produce complex motor skills, such as breakdancing, handstands and hand walks. The cost function is specified in terms of intuitive motion properties (e.g., place the hands close to the ground, place the feet as high as possible, maximize angular momentum in some direction, etc.). Unlike previous work, our method does not require contacts to be pre-specified, which simplifies the synthesis process and allows tackling motions that were previously outside the scope of physics-based methods. Our approach is currently too slow for online applications that require control solutions for a range of tasks to be performed. We show that if we restrict the problem to specific movements, it is possible to design online controllers that achieve a range of tasks and that are robust to disturbances. We demonstrate this for a variety of rotational movements, including cartwheels, dives, flips and handsprings. The control solutions described so far are only valid for a small set of states of the character. We develop a method to systematically expand this set, thereby increasing the robustness of our control solutions and allowing for their sequencing. Once a skill is learned by a character with a particular body shape, we show how it can be transferred to a character with a different body shape. The method discovers surprisingly natural results, such as a heavier character avoiding losing balance by taking extra steps and stretching out its arms. For a given character, this opens up a large reservoir of skills that can be acquired.
Acknowledgements

I am grateful to:

• My truly remarkable advisor, Eugene Fiume. For his brilliance, patience, generosity, kindness and integrity. A pleasure to work with.

• My collaborators over the results in the thesis: Michael Black, Martin de Lasa, Eugene Fiume, Aaron Hertzmann, Ludovic Righetti, Javier Romero and Michiel van de Panne. The results in Chapter 4 were built with the feature-based framework developed in de Lasa et al. [29]. Javier Romero was instrumental in the implementation of Sec. 6.4.3.

• My committee members, David Fleet and Manfredi Maggiore. For very useful discussions that inspired some of the results in the thesis.

• My internal and external examiners, David Levin and Paul Kry. For their careful review of my thesis and their helpful comments.

• My friends at the University of Toronto (Daniyal, Mike, Pif, etc.) and outside. For bringing laughs and joy to my life. A special word for Joe Lazslo, one of the nicest persons I had the privilege to know and that I always miss.

• My parents, aunt, brother and sister-in-law. For their limitless generosity, sacrifice, and love. For reminding me of what truly is important in life. It is their name that should be on top of this thesis.
## Contents

1 Introduction  
   1.1 Thesis Overview ......................................................... 1  
   1.2 Contributions ........................................................... 3  

2 Background  
   2.1 Data-driven Approaches ............................................... 4  
   2.2 Physically-based Modeling ............................................ 5  
      2.2.1 System Dynamics ................................................... 6  
   2.3 Action-specific Controllers .......................................... 7  
      2.3.1 Joint-local Control ................................................. 7  
      2.3.2 High-level Control ............................................... 8  
      2.3.3 Jacobian Transpose ............................................... 9  
      2.3.4 Feature-based Control .......................................... 9  
   2.4 Trajectory Optimization .............................................. 10  
      2.4.1 Direct Methods .................................................... 11  
      2.4.2 Shooting Methods ................................................. 11  
   2.5 Feedback Control ..................................................... 12  

3 Trajectory Optimization for Movements with Complex Contacts  
   3.1 Introduction .............................................................. 14  
   3.2 Related Work ............................................................ 15  
   3.3 Methodology .............................................................. 16  
   3.4 Parameterization and Optimization .................................. 16  
   3.5 Balancing and Getting Up .............................................. 18  
      3.5.1 Balancing ............................................................. 18  
      3.5.2 Getting up: Athletic ............................................. 19  
      3.5.3 Getting up: Low-Energy .......................................... 20  
   3.6 Walking ................................................................. 20  
   3.7 Inverted Movements ..................................................... 21  
      3.7.1 Handstand ............................................................ 22  
      3.7.2 One-Handstand ..................................................... 23  
      3.7.3 Hand walk ........................................................... 23  
   3.8 Rotational Movements .................................................. 23  
      3.8.1 Headspin ............................................................ 24
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>Planar Pendulum</td>
<td>56</td>
</tr>
<tr>
<td>5.7</td>
<td>Simulated Character</td>
<td>56</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Full Branch Rollouts</td>
<td>57</td>
</tr>
<tr>
<td>5.7.2</td>
<td>Facing Direction Invariance</td>
<td>57</td>
</tr>
<tr>
<td>5.7.3</td>
<td>Implementation Details</td>
<td>58</td>
</tr>
<tr>
<td>5.7.4</td>
<td>Results</td>
<td>58</td>
</tr>
<tr>
<td>5.8</td>
<td>Discussion</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>Robust Physics-based Motion Retargeting with Realistic Body Shapes</td>
<td>63</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>63</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>65</td>
</tr>
<tr>
<td>6.3</td>
<td>Methodology</td>
<td>67</td>
</tr>
<tr>
<td>6.4</td>
<td>Optimization for Motion Re-Targeting</td>
<td>67</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Cost Function</td>
<td>68</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Optimization</td>
<td>69</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Mesh To Geometric Primitives</td>
<td>69</td>
</tr>
<tr>
<td>6.5</td>
<td>Robust Motion Tracking</td>
<td>71</td>
</tr>
<tr>
<td>6.5.1</td>
<td>LQR Feedback</td>
<td>72</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Trajectory-Tracking-LQR-Trees</td>
<td>73</td>
</tr>
<tr>
<td>6.6</td>
<td>Results</td>
<td>74</td>
</tr>
<tr>
<td>7</td>
<td>Discussion and Future Work</td>
<td>80</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td></td>
<td>81</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td><strong>Motion graph.</strong> From Kovar et al. [63].</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td><strong>Simulated character model.</strong> We model the human body as an articulated rigid body. The figure illustrates the fixed world frame $R_w$, the body reference frame $R_b$, and the translation vector $p_b$ between the reference frames.</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td><strong>SIMBICON controller.</strong> Finite state machine in SIMBICON for walking in Yin et al. [107].</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td><strong>Friction cone approximation.</strong> From Stewart et al. [112].</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td><strong>Trajectory optimization and contact.</strong> Trajectory optimization methods typically require contacts to be pre-specified because of the discontinuity in state-space that occurs at contact. From Posa et al. [112].</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td><strong>Spacetime window optimization schedule.</strong> We define the objective function for a motion by creating a sequence of spacetime windows. Each window defines the objective function for a fixed interval. Optimization is performed with an advancing schedule: the motion during windows 1 and 2 are optimized. Then the motion in windows 2 and 3 are jointly optimized, and so on, until every window has been visited.</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td><strong>Getting up motions. Top row:</strong> Starting from a supine position, optimizing over a sequence of four balancing windows yields a very athletic getting-up motion. The objective function is the same in each window, except for the rest pose term used in the final window. For each window except the last, the first and middle frames are shown. For the last window, the first and last frames are shown. <strong>Bottom row:</strong> A more normal getting-up motion from a prone position is produced by partially specifying contacts, for various windows, as described in the text. Each image shows the first frame for the corresponding window, except for the last image which shows the final frame.</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td><strong>Walking.</strong> First/middle frames of two successive walking windows. The objective function is the same in each window, except for swapping roles of stance/swing feet. Changing parameters of relevant objective function terms gives backward walking and turns, as shown in the accompanying video [8].</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td><strong>Handstand.</strong> The character begins in standing balance. After a transition window, handstand windows (with the same objective function) are used to achieve the task. Finally, the character returns to standing using two balance windows.</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td><strong>Hand walk.</strong> This motion combines objective terms from the handstand and from regular walking. The first and middle frames of each window are shown.</td>
<td>23</td>
</tr>
</tbody>
</table>
3.6 **Headspin.** The character keeps its head on the ground while producing large angular momentum. .................................................. 24

3.7 **Handspin.** The character keeps its hands on the ground while producing large angular momentum. .................................................. 25

3.8 **Backflip.** The backflip is produced by controlling COM height, contacts, and maximizing angular momentum. The first and middle frames of windows 2 to 4 are shown, together with the final frame. .................................................. 26

3.9 **Crawl.** First/middle frames of successive crawling windows. .................................................. 27

3.10 **Backflip-to-Handstand.** The first and middle frames of each window in a backflip followed by handstand and balance. The last figure is the final frame. .................................................. 28

3.11 **Multiple Backflips.** A sequence of backflips is generated by combining windows from the backflip and balance motions. The first and middle frames of each window of the sequence are shown. The last image is the final frame. .................................................. 29

4.1 **Controllers for rotational movements parameterized by phase.** We show that parameterizing controllers for rotational movements by a phase variable can make them general and robust. The variable is used to uniquely position the character in the trajectory instead of time. A natural choice of a phase variable for rotational motion is an angle in the plane of rotation. In this image, the angle of the line between the neck and the pelvis is the phase variable. .................................................. 36

4.2 **Cartwheel.** The first frame of the five stages of the cartwheel controller and the last frame are shown. The red line connects the pelvis and the neck. The angle of this line is used to switch between stages. .................................................. 37

4.3 **Backflip.** In the pre-airborne stage, the backflip controller generates an appropriate amount of angular momentum for the desired apex and style of the flip. In the airborne stage, the controller modifies the character’s inertia based on its current state and the desired landing position. .................................................. 40

4.4 **The airborne controller.** We model the character as a spinning disk with a radius that is proportional to the inertia. The line represents the orientation of the character. The leftmost disk models the current state of the character. The rightmost disk models the desired inertia and orientation at landing. The time between two consecutive disks is \( t_r/2 \), where \( t_r \) is the estimated time until landing. The airborne controller finds a target inertia (the middle disk) to connect the leftmost and rightmost disks. .................................................. 40

4.5 **Diving.** The diving and flip controllers are almost identical, which demonstrates the generality of the control laws. We show that the identical control strategy succeeds for a wide variety of heights, flips, and styles. .................................................. 41

4.6 **Back handspring.** The pre-airborne stage of the back handspring controller precisely regulates the angular momentum of the motion. In the airborne stage, the character raises its arms as the revolution progresses in order to ensure proper landing. In the post-airborne stage, the controller uses \( E_{AM} \) and \( E_{foot} \) to bring the character back on its feet. .................................................. 41

4.7 **Pirouette.** The pirouette controller uses angular momentum to make the body whirl about the supporting leg. .................................................. 42
4.8 Aerial. The aerial controller begins like the flip controller. When the character is near the end of its revolution, the target vertical position of the feet decrease as the revolution progresses to ensure a proper landing.

5.1 DOA expansion with an RRT. An RRT starts by sampling a random state in the domain $x^{\text{rand}}$ and finds its closest node $x^{\text{near}}$ in the tree. It then attempts to connect to $x^{\text{near}}$ (green arrow) starting from a state which is close and in the direction of $x^{\text{rand}}$. The intuition behind DFC is that attempting to connect to the dense parts of the tree is more likely to be successful (orange arrow) even if it is slightly further away according to the distance metric. The red and purple circles are the target and goal states, and the arrow in between is the initial controller $C_0$. The blue background is the domain covered by the tree.

5.2 Examples of Trajectory Optimization in a Sparse and Dense Tree. The blue tubes illustrate the domain covered by edges of the tree. The orange circle is a random state that we attempt to connect to its nearest state in the tree with trajectory optimization. The dotted curves are the trajectories followed when attempting the connection. The figures show that the attempt is more likely to be successful in a dense tree (right figure) than in a sparse tree (left figure).

5.3 (Left) DOA coverage with DFC. The DFC algorithm progressively covers the domain around the target state (the red circle). Increasingly larger subsets are illustrated in the figure. The orange circle is a random state sampled on the largest subset. TOPT is used to steer the state to its closest state in a smaller subset. (Right) Determining if a state is inside the DOA. We illustrate how we determine if a state $x'$ (the orange circle) is inside the DOA. In the figure, we perform simulation rollouts, starting from the controllers associated with the closest states in the tree (the orange arrows illustrate the sequence of controllers used for one closest state). We use PD controllers with targets provided by time-indexed splines. A rollout consists of moving up the nodes of a branch, until the root (the goal state) is reached. If one of the rollouts steers the character within an epsilon distance of the goal state, then $x'$ is inside the DOA.

5.4 Full Branch Rollouts. In this figure, we attempt to connect the orange node to its nearest node in the tree. The typical TOPT approach is to steer the node towards the nearest node (the dotted orange arrow). In some problems, we were not able to make any connection this way. We modify the TOPT by using the PD controllers on the closest branch in the rollouts (the solid orange arrows) after a short time interval elapses. The objective is to steer the character as close as possible the goal state. This optimization is more likely to succeed than attempting to find a motion directly to the goal state (the green arrow).

5.5 Getting Up Controllers. The figure is divided in three pairs of sequences. For each pair, the top sequence illustrates an initial getting up controller and the bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The yellow arrow points to the desired facing direction for the final pose of the character. The red character is the pose of the currently active node.
5.6 **Crouch-to-Stand Controller.** The top sequence consists of key-frames of the crouch-to-stand controller. The bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The blue line on the first frame is an external force applied on the head. The red character is the pose of the currently active node.

5.7 **Exploration regions.** The sequence shows typical states (orange character) in progressively larger subsets for the target states (the red character) associated with the “Getting Up 1” and the crouch-to-stand controllers. The subsets are $\alpha = 1$, $\alpha = 5$, and $\alpha = 10$ for the first three figures, and $\kappa = 100 \text{ Nm}$, $\kappa = 400 \text{ Nm}$, and $\kappa = 1000 \text{ Nm}$ for the last three figures.

5.8 **Visualizing the nodes in the Tree.** The first three figures shows the DOA Tree with 1, 10 and 40 nodes for the “Getting Up 2” controller. The last three figures shows the DOA Tree with 1, 10 and 40 nodes for the crouch-to-stand controller. These nodes are sampled from $\Omega$. The difference in velocities between the nodes cannot be visualized.

5.9 **The Size of the Tree.** The figure plots the size of the tree in log scale required to cover a subset for the “Getting Up 1” and the “Getting Up 2” controllers. The subsets are parametrized by $\alpha$, which is defined in Sec. 5.5.3. In the plot, the value of $\alpha$ is shown in the Subset-axis.

6.1 **Body-base retargeting.** On the left are two different 3D bodies and their approximation in terms of “capsules”. The red body is the “source” body for which we have motion capture. The green body is our “target” body that we wish to animate. We fit a physics-based controller for the red body that is able to accurately mimic the motion capture; the upper right shows two example motions. We then fit a physics-based controller for the target body but, due to its different physical shape, it is unable to reproduce the motions of the smaller, fitter, source. The motion deviates in ways that are natural and consistent with the body shape. The approach provides natural, automatic, variability in motion through retargeting to new body shapes. See the accompanying video for animations.

6.2 **From scans to capsules.** Each row depicts two different Dyna subjects, the first one is 1.69 meters tall and 62.2 kg, while the second one is 1.80 meters tall and 155.4 kg. From left to right, we show a) scans of the subjects, b) the unposed template as a triangulated mesh, c) the optimized capsule representation, and d) the adapted MuJoCo geometry primitives representation. In the first row, two last columns we represent joints color-coded according to their dimensionality: red, green and blue circles represent 3, 2 and 1 degrees of freedom respectively. The root joint is represented in magenta since it is represented with a 4D quaternion.

6.3 **Trajectory Tracking LQR-Tree.** The blue region illustrates $\chi$. The region is filled with LQR feedback. The long thick arrow is the sequence $x_{1:T}$ and the red circles are the random states sampled from $x_{1:T}$. The blue circles are sampled states that are successfully added to the tree. As can be seen in the figure, the nodes do not always connect back to the red circles due to the $\alpha$ term.
6.4 **Turning-twist motion.** In the top sequence, the target character has the same shape as the source character. In the bottom sequence, the target character is heavier. Note the position of the arms in the motion of the heavy character. In the accompanying video [6], it can also be seen that the heavy character has a different way of pivoting, and that a lighter character also displays a different way of performing the motion. 75

6.5 **Recovery steps.** This sequence illustrates how a heavy character can lose balance when getting up too quickly from a crouch position. In order not to lose balance, the character takes a few recovery steps forward. After avoiding the risk of falling, it takes a few steps backwards to return to its desired position, and continue dancing. 76

6.6 **Air-kick with high, moderate and low torque limits.** With high torque limits, the character jumps too high (leftmost). With moderate torque limits, the result is more natural (middle). With low torque limits, the character just slides (rightmost). 76

6.7 **Time Horizon.** The first sequence uses a 0.8 s window horizon in the optimization, the second 0.9 s, and the third 1.0 s. Note that in the second sequence, the heavy character does not kick as high as in the first sequence to avoid falling, but it still falls when performing the back kick. In the third sequence, the character stretches its leg backwards, but keeps its foot on the floor to stay in balance. 77

6.8 **Robustness.** The red line is an external force applied on the characters. The yellow character has an LQR controller. The orange character has an LQR-tree controller. 77
Chapter 1

Introduction

It is easy to be impressed by the remarkable motor abilities of athletes, gymnasts, ballet performers and breakdancers. We should also be impressed by our everyday movements, from sitting on a chair, to picking up a glass of water. Without being conscious of it, our motor system is constantly solving computationally challenging problems. It must rapidly determine the muscle activations to achieve a task (e.g., picking up a glass of water located anywhere on a table), under different conditions (e.g., while holding a bag or with slippery ground), with uncertainty (e.g., without eye glasses) and robustness (e.g., the glass must not tip over if someone starts pouring water in it).

Motivated by applications in robotics, biomechanics, computer graphics and motor neuroscience, one current research topic is to endow physically simulated characters with similar motor abilities as humans. Robots could increase manufacturing productivity, perform mundane tasks (e.g., build an IKEA chair [113], help the elderly [33], perform search and rescue operations (e.g., replace firefighters in dangerous situations), and improve the quality of life of amputees and the physically-challenged [23]. As robotics technology progresses, concerns are emerging on its ethical usage and societal impact [11, 96]. Some potential applications in biomechanics are to predict the effect of surgery on a person’s movements and gait, or to predict the difficulty an elderly person would have to perform a particular motion. In computer graphics, there is a need to animate imaginary creatures to produce narratives in film. Synthesizing motions by physical simulation could provide an alternative to the current labor-intensive process of key-framing and motion capture. Much of how the central nervous system produces movements remains unknown. Our control algorithms in simulated humans could provide insight on how humans produce movements, in the spirit of Marr’s level of analysis approach to neuroscience [88].

1.1 Thesis Overview

Humans have the ability to move with ease, grace, precision and speed in constantly changing environments and bodies. There are still no simulated characters (or robots) that come near the level of human performance. This thesis aims to close this gap by answering the following question: How can simulated characters acquire robust motor skills? By robust motor skills, we mean complex movements that can be performed from many states, for many tasks, for many body shapes, under disturbances and in real-time.

Our thesis introduces several techniques to make progress towards this goal. We present an opti-
mization method for motion synthesis that is less prone to local minima than previous methods, which were limited to low-dimensional models, approximate physics, low-energy movements and movements with simple contacts. We show how complex motions can emerge from simple and intuitive objectives. However, the method is currently too slow to synthesize control solutions for multiple tasks and under disturbances at interactive rates. We show that by restricting the problem to specific movements, it is possible to design controllers that satisfy these requirements. For our control solutions to be used from many states, we develop a method that systematically attempts to grow their domain of validity. Lastly, we show how a skill performed in a given body shape can be realistically transferred to a different body shape, thereby enabling for a character the ability to acquire a large repertoire of skills.

Chapter 2 is a review of the fundamentals of physics-based character animation and a survey of related work in control design and trajectory optimization.

Chapter 3 presents a method for motion synthesis. It can be very difficult and time-consuming to design controllers that produce complex and contact-rich movements. Instead of heuristics and trial-and-error, it is possible to synthesize motion through optimization, requiring high-level objectives only, such as: one hand should be on the ground or the center-of-mass should be above a particular height. The approach is known as trajectory optimization. Progress on full-body trajectory optimization has been limited because the problem is high-dimensional and the objectives are non-linear, which often leads optimization procedures into bad local minima resulting in unnatural motion. The best existing methods rely on motion capture data, which limits their generality, or use simplified character models that eliminate important degrees-of-freedom, such as planar models. We present a method for full-body trajectory optimization of human movement that does not rely on prior data. The main technical contribution is in the design of a set of modular and reusable objective functions for many types of motion. We show that just a few, easily-interpretable—but carefully-chosen—objectives are sufficient to describe and create many diverse types of motion, such as crawling, handstands, handspins and headpins. It would have been nearly impossible to synthesize these contact-rich motions with traditional trajectory optimization methods, which require the contact locations to be pre-specified to avoid local minima.

Trajectory optimization methods require solving a new offline optimization whenever there is a change in the task to be performed. Chapter 4 shows how action-specific controllers can be designed to capture a wide range of tasks at interactive rates. Previous research in character control has mainly focused on designing balancing and walking controllers. Controllers have been developed for rotational movements, but the solutions lack generality, robustness, and require a significant amount of manual tuning. We present an approach for the control design of rotational movements that mitigates these limitations. We show how to synthesize controllers for a variety of movements such as flips, dives and handsprings, using high-level features of the motion. These controllers can be used for many different tasks (e.g., flips with different apexes, dives at different heights, etc.) and are robust to disturbances in the environment.

Controllers are only suitable when the character’s state lies within a particular set of states, known as the domain of attraction. The domain of attraction is typically very small, which means that controllers are often unsuitable in the presence of disturbances, uncertainty or when attempting to sequence controllers together. Chapter 5 addresses the question: How can we expand the domain of attraction? In other words, how can we expand the set of states from which we can use our controllers? The question has previously been addressed for a low-dimensional system (1 degree-of-freedom system) \[120\]. The idea is to cover a region in state space with feedback controllers that are bound together as a randomized tree. Inspired by this idea, we introduce techniques and algorithms for the expansion process to be
performed on a simulated character. We will see that the process is significantly more efficient when the randomized tree has a dense topology. We also show how the method can be used to connect controllers together.

Chapter 6 of the thesis combines physics-based simulation and motion capture. The goal is to synthesize motion with physics simulation to be as close as possible to a motion capture clip, under different constraints. Given a motion capture clip performed by a person with a certain body shape, one question to ask is: how would the motion be different if the person had a different shape? This question arises in healthcare (e.g., to estimate how a person would move differently after surgery), computer graphics (e.g., to animate imaginary creatures) and robotics (e.g., to use human motion as reference data for a humanoid robot). The problem is known as motion retargeting. We show that the trajectory optimization method introduced in Chapter 3 can produce a motion that is close to the motion capture clip, when the simulated character has a similar shape to the actual person in the clip. By varying the shape of the character, we obtain realistic motion variations, such as a heavier character stretching its arms to maintain balance or taking extra steps to avoid falling. We then present an algorithm to track motion capture data through disturbances based on the domain of attraction work in Chapter 5. The algorithm constructs random trees along the motion capture sequence and uses the more effective linear quadratic regulator instead of proportional-derivative control for feedback.

1.2 Contributions

The thesis makes the following contributions:

- A trajectory optimization method that does not require motion capture, key-framing or pre-specifying contacts. We show that with simple objectives, it can synthesize movements that were previously outside the scope of physics-based methods.

- A controller design method for highly-dynamic rotational movements, which was an open problem mentioned in de Lasa et al. [29] and Brown et al. [17]. In addition, we show how these controllers can be made general and robust, properties that were lacking in previous results.

- A method to systematically expand the domain of validity of our control solutions. We demonstrate that the method can be used to increase controller robustness and to transition between different controllers.

- A method to realistically transfer a motion performed by a character with a given body shape to a character with a different body shape.

Our contributions address the following questions. How can a skill be synthesized without prior data? How can a skill be used for many tasks, under disturbances and at interactive rates? How can a skill be used in a large set of possible initial conditions? How can a skill be acquired by watching a character, possibly with a different body shape, perform it? Taken together, these questions fit the underlying question of how can simulated characters acquire robust motor skills, as humans do.

The following website contains the videos that illustrate the results in the thesis:

http://www.cs.toronto.edu/~mazen/thesis/video
Chapter 2

Background

This section provides the technical foundations for our work in physics-based character animation. Related work pertaining to our specific results will be discussed in the chapter to which it is most relevant. The section begins by highlighting the limitations of data-driven approaches in Sec. 2.1 and then provides an overview of physically-based modeling in Sec. 2.2. In Sec. 2.3 and Sec. 2.4, we present the two main control approaches for motion synthesis, namely action-specific control design and trajectory optimization. In Sec. 2.5 we introduce the concept of feedback with the linear quadratic regulator, a classical result in optimal control theory.

2.1 Data-driven Approaches

Motion capture, the process of digitally recording a person’s movements, is widely used to generate realistic character animation. However, motion capture by itself is only a way to playback motions, not produce new ones. One way to generate new motions is to re-sequence and interpolate motion clips from an existing database. This is the approach taken in motion graphs [63], a representative data-driven method for character animation. A motion graph is a directed graph where edges are motion clips and nodes are points of possible transitions between motion clips (see Fig. 2.1). Considerable manual effort is required to build a large motion capture database. To alleviate some of the effort, Arikan et al. [13] show how to automatically annotate the database (e.g., in run, walk or stand motions), so that it can then be searched to generate a desired sequence. Because the search time increases with the size of the database, there is a need to determine when a database is sufficiently large. This is addressed by Reitsma et al. [108], where it is shown how to evaluate the properties of a motion graph in a given environment.
(e.g., by the ability of a motion graph to reach all the points in an environment). This indicates when additional motion clips are necessary to generate a desired sequence.

The fundamental limitation of motion graphs and data-driven approaches is their inability to generate animations that are not in the database. Given that it is not possible to have motion clips that capture all the possible interactions between a character and its environment, another promising direction is to physically simulate the motion. Both approaches are not exclusionary: motion capture can help provide initial conditions and priors to improve the realism of physics simulation results, while physics simulation can help generalize motion capture to different environments and body shapes.

2.2 Physically-based Modeling

We model the human body as a tree structure of rigid bodies connected by joints. Consider a fixed world frame $R_w$ and a reference frame $R_b$ attached on a character. We denote by $p_b$ and $\phi_b$ the position in $\mathbb{R}^3$ and the orientation in $\text{SO}(3)$ of $R_b$ with respect to $R_w$. Let $q_b$ denote the $N_b$-dimensional vector of joint angles positions in the $R_b$ reference frame for the character’s $N_b$ joints. The set of generalized positions is denoted by $q = (p_b, \phi_b, q_b)$. In this thesis, we refer to $q$ and $\dot{q}$ as the joint positions and velocities. Fig. 2.2 displays a simulated character with 56 degrees-of-freedom.

![Figure 2.2: Simulated character model.](image)

The main components of a physics-based character animation system are as follows. We have a simulated character with given joint positions and velocities in an environment. A controller determines the input torques $\mathbf{u}$ to perform a desired motion. A physics simulator takes $\mathbf{u}$, performs collision detection and computes the contact, external and constraint forces applied on the character. The forces and torques
are used to compute the joint accelerations, a step known as forward dynamics. Numerical integration updates the joint positions and velocities, and the process iterates again to produce continuous motion.

2.2.1 System Dynamics

Consider simulating an articulated rigid body in a given environment. The simulated body’s Lagrangian is defined as the difference between the kinetic $K(q, \dot{q})$ and potential energies $V(q)$:

$$L(q, \dot{q}) = K(q, \dot{q}) - V(q). \quad (2.1)$$

From the Lagrangian, the equations of motion can be derived

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = B(q)u + J^T f, \quad (2.2)$$

where $B(q)$ is the torque distribution matrix, $u$ is the vector of actuator torques, $f$ is the vector of external forces $f = [f_1^T \ldots f_{Nc}^T]^T$ applied at $Nc$ contact points $c = [c_1^T \ldots c_{Nc}^T]^T$, and $J$ is the Jacobian matrix of the contact points $\frac{\partial c}{\partial q}$.

The equations of motion can be rewritten as

$$M(q)\ddot{q} + H(q, \dot{q}) = B(q)u + J^T f, \quad (2.3)$$

where $M(q)$ is called the joint-space inertia matrix (or the generalized mass matrix), and $H(q, \dot{q})$ is the vector of Coriolis, centrifugal and gravitational effects (see, e.g., [77]). For notational convenience, we drop the dependence on $q$ and $\dot{q}$ in the $M$, $H$ and $B$ variables. The external forces $f$ are usually ground contact forces. With a Coulomb friction model, we have the following constraint for static (or sticking) contact

$$||f||_2 \leq \mu f_n, \quad (2.4)$$

where $\mu$ is the coefficient of friction, and $f_t$ and $f_n$ are the tangential and normal component of the contact force. For sliding contact, $f_t$ has the magnitude of $\mu f_n$ and is in the direction opposite to the sliding motion [130]. Let $d_f$ denote the vector of vertical distances of possible contact points. Assuming a rigid contact model [130], we add the following constraints

$$f_n \geq 0, \quad (2.5)$$
$$d_f \geq 0, \quad (2.6)$$
$$f_n^T d_f = 0, \quad (2.7)$$

were inequalities (2.5) and (2.6) are element-wise constraints. The above constraints ensure that the contact forces are unilateral (i.e, the sign of $f_n$ implies that the character can push and not pull on the ground), and that there are no contact forces without actual contact (i.e, when $d_f = 0$). If a contact point $c_i$ reaches the contact surface with a negative velocity, a restitution law is applied to avoid surface penetration, which causes a discontinuity in the joint velocities [12] [130].

If the number of dimensions of $q$ is the same as the rank of $B$, then the model is fully actuated, meaning that we can cause an arbitrary acceleration in $q$. Otherwise, the model is underactuated [119].
Chapter 2. Background

A system can be fully actuated at certain states, and underactuated in other states. For instance, if the stance foot is constrained to be flat on the ground for a simulated character, which occurs during certain phases of bipedal walking, the control system is fully actuated. When the stance foot rotates about the toe, the character is in an underactuated phase because it has no control on its global position and orientation without recourse to contact forces [42].

Because $M$ is full-rank, we can rewrite Eq. 2.3 as

$$\ddot{q} = M^{-1}(Bu + J^Tf - H).$$

(2.8)

This is a second-order differential equation in the form

$$\ddot{q} = f(q, \dot{q}, u).$$

(2.9)

With a change of variables $x = [q, \dot{q}]$, we can express it in first-order

$$\dot{x} = F(x, u).$$

(2.10)

We refer to the joint positions and velocities as the character’s state.

Numerically integrating Eq. 2.10 we obtain the discrete form of the dynamics

$$x_{t+1} = \tilde{F}(x_t, u_t),$$

(2.11)

where the subscripts indicate the timestep. Assuming a forward Euler integrator [20], we have

$$x_{t+1} = x_t + h\dot{x}_t,$$

(2.12)

where $h$ is the integration timestep.

Physics-based methods introduce the difficult problem of how to control the character, that is how to determine the torques $u$ to apply to achieve a desired result. There are two general control approaches: designing action-specific controllers and trajectory optimization. We discuss representative work in these categories in the following sections.

### 2.3 Action-specific Controllers

Early research in character animation has mainly focused on designing controllers for specific movements such as bipedal locomotion, bicycling, diving, jumping, handsprings, etc. The approach taken treats individual joints independently with proportional-derivative (PD) control. We refer to the approach as joint-local control. While it is easy to control joints individually, it is very hard to synchronize them together to perform a motion. More recently, techniques have been introduced to control high-level features of the motion instead of individual joints.

#### 2.3.1 Joint-local Control

As an example of joint-local control, we discuss the well-known SIMBICON controller for bipedal locomotion [142]. SIMBICON uses a simple control strategy that is remarkably robust to external forces and
unexpected variations in the environment. It uses the finite state machine illustrated in Fig. 2.3. Each state in the finite state machine has a target pose. PD control drives the character to the target pose, which is usually not achieved. The character passes from one state to the other when a certain duration of time elapses or when a foot contact occurs. The torque \( u \) for a joint is determined by PD control

\[
    u = k_p(\dot{q} - q) - k_d \dot{q},
\]

where \( q \) and \( \dot{q} \) are the current and target values of the joint in body coordinates, and \( k_p \) and \( k_d \) are gains. Some contributing factors to the controller’s robustness are the use of world coordinates when controlling key joints such as the swing hip and the use of a balance feedback strategy that regulates the placement of the swing foot. This allows a character with a disturbed global orientation (i.e., the character is about to lose balance) the ability to position its swing foot based on the current positions of the COM and the stance foot in order to regain balance.

![Figure 2.3: SIMBICON controller. Finite state machine in SIMBICON for walking in Yin et al. [107].](image)

While SIMBICON is robust, it produces an unnatural (marching-like) walking gait. Wang et al. [127] optimize the parameters of SIMBICON to reproduce features of natural walking, such as active toe-off, near-passive knee swing, and leg extension during swing. In follow-up work, controllers are optimized for robustness to uncertainty from external forces, user inputs, and motor noise [128]. After specifying a probability distribution for the unknown variables (e.g., wind forces that can be applied on the character’s torso), control parameters are optimized with respect to the controller’s approximate expected return. The synthesized controller is shown to be robust to uncertainty from random wind forces, user inputs, etc., and at times to reproduce features of human behavior. For instance, when walking on a surface of variable roughness, the character lowers his center-of-mass and stretches its arms to maintain balance. In more recent work, Wang et al. [129] and Geijtenbeek et al. [37] show improved locomotion realism by using a more advanced biomechanical model of the human body, namely by replacing joint torques with muscle forces.

### 2.3.2 High-level Control

Instead of controlling individual joints together, there are techniques to control high-level features of the motion like the positions of the center-of-mass or an end-effector. We present two techniques for high-level control: Jacobian transpose and Feature-based control.
2.3.3 Jacobian Transpose

Introduced by Sunada et al. [115] and Pratt et al. [105], the Jacobian transpose method determines the joint torques that produce the same effect as an external force $f$ applied on a point $p$ on the character. The torques are given by

$$ u = J_p^T f, \tag{2.14} $$

where $J_p^T$ is the Jacobian of $p$. Coros et al. [24] use this technique to find the torques that have the same effect as an external force on the center-of-mass. This drives the current velocity of the center-of-mass towards a desired velocity. The Jacobian transpose method neglects Coriolis effects, gravitational effects and contact constraints. Feature-based controllers take into account the equations of motion to overcome these limitations. For example, a feature-based controller could drive the center-of-mass velocity towards a desired velocity, while ensuring that a foot stays in contact with the ground.

2.3.4 Feature-based Control

Feature-based controllers were introduced to the graphics community by Abe et al. [2] for balance motions. A feature $d$ is a map of the character’s state

$$ d = F(q, \dot{q}). \tag{2.15} $$

Some examples of features are the center-of-mass, end-effectors, angular momentum, etc.

Given the current character’s state $(q, \dot{q})$, feature-based controllers solve the following quadratic program at each timestep:

$$ \begin{align*}
\min_{q, \dot{q}, u} & \quad w_1 g_1 + \ldots + w_n g_n \tag{2.16} \\
M(q)\ddot{q} + H(q, \dot{q}) &= B(q)u + J_p^T f, \tag{2.17} \\
u &\in L, \tag{2.18} \\
f &\in K, \tag{2.19} \\
Jq + \dot{J}\dot{q} &= 0. \tag{2.20}
\end{align*} $$

The cost function is a weighted sum of quadratic objectives $g_1, \ldots, g_n$. Each objective $g_i$ encourages the current feature acceleration $\ddot{d}_i$ to be as close as possible to a desired feature acceleration $\ddot{\bar{d}}_i$

$$ g_i = ||\ddot{d}_i - \ddot{\bar{d}}_i||_2, \tag{2.21} $$

where $\ddot{d}_i = J_d q + J_d \dot{q}$ and $J_d$ is the Jacobian of feature $d_i$. Constraints (2.17) and (2.18) ensure that the solution satisfies the equations of motion and torque limits. Constraint (2.19) ensures that contact forces are inside the friction cone, which is the set of all contact forces that satisfies the coulomb friction constraint for static contact (see Eq. 2.4). Other constraints can be included for sliding contact [103].

A polyhedral approximation of the friction cone is usually used to obtain linear constraints [67] (see Fig. 2.4). The “no-slip” constraint (2.20) ensures that the acceleration of contact points be 0 to avoid slipping. There are a few variants of this formulation. For example, in Kuindersma et al. [67], the no-slip
condition ensures that the acceleration of contact points are negatively proportional to their velocity.

The feature-based approach has recently been widely adopted by the robotics community for three-dimensional balance and walking [53, 67]. We have not seen the same adoption in computer graphics, but the approach has been used to simplify the synthesis of balance, walking, jumping and rolling motions.

Figure 2.4: Friction cone approximation. From Stewart et al. [112].

2.4 Trajectory Optimization

In trajectory optimization, motion is synthesized by solving a constrained optimization problem with high-level objectives [132]. Unlike feature-based control, trajectory optimization methods are not restricted to a one timestep temporal horizon. The optimization constraints ensure that the motion is physically valid. For example, to have a character jump, an animator could specify the initial and final positions, the environment, how the motion should be performed (e.g., by minimizing energy or by twisting the body while in the air), etc. Trajectory optimization can then be used to produce a motion that satisfies the constraints and objectives. The motion will exhibit features not explicitly specified, such as the character must crouch before jumping. However, some of the exhibited features will be undesirable (i.e., unnatural postures or motion strategies are employed). This is due to local minima, inaccurate physical body models, a lack of psychological modeling, etc.

There are two main methods for trajectory optimization: direct and shooting methods. Shooting methods have the controls $u_{0:T-1}$ as free variables in the optimization. Starting from the initial state $x_0$, forward simulation with $u_{0:T-1}$ is used to obtain $x_{1:T}$. Direct methods optimize for $x_{0:T}$ and $u_{0:T-1}$, adding constraints to ensure the variables are compatible with the dynamics. Both methods have their benefits. Direct methods do not require any forward simulation, but require more free variables. Tedrake argues that direct methods are better conditioned and better suited to avoid local minima [119]. Shooting methods always lead to a physically valid solution, while direct methods must first converge to a physically valid solution.

Most trajectory optimization methods require pre-specifying contacts to avoid falling in local minima. This is due to the discontinuous jump in state-space that occurs at contact (see Fig. 2.5), which limits the performance of gradient-based optimizers. Overcoming this limitation is a topic of current research, e.g., [92, 103]. We also tackle this problem in Chapter 3 of the thesis.
2.4.1 Direct Methods

The direct transcription method is an example of a direct method for trajectory optimization [100]. It optimizes the values of the states and controls at \( N \) discrete points \((x_{1:N}, u_{1:N})\) with respect to a cost function \( g(\cdot) \), given constraints \( c(\cdot) \) that satisfy the system dynamics for a chosen integration scheme, etc. We assume a constant time interval \( dt \) between points. The optimization is in the form

\[
\min_{x_{1:N}, u_{1:N}} g(x_{1:N}, u_{1:N}) \quad \text{subject to} \quad c(x_{1:N}, u_{1:N}) = 0.
\]

With an euler integration scheme, we have constraints \( c_t \) in the form

\[
c_t = x_{t+1} - x_t - f(x_t, u_t)dt = 0.
\]

With the trapezoidal integration rule [20], we have

\[
c_t = x_{t+1} - x_t - \frac{dt}{2}(f(x_t, u_t) + f(x_{t+1}, u_{t+1})) = 0.
\]

Other constraints can be added, e.g., to enforce a desired final state or contact forces to be inside the friction cone. The direct collocation method for trajectory optimization as characterized in Eq. 2.25 can be seen as a direct transcription of Simpson’s integration rule [100]. Direct methods typically require the derivatives of the cost function \( g(\cdot) \) and the constraints \( c(\cdot) \) with respect to the free variables \( x_{1:N} \) and \( u_{1:N} \). The derivatives are used within an optimization algorithm such as Sequential Quadratic Programming (SQP) to move an initial guess towards a feasible solution [38].

2.4.2 Shooting Methods

We observe that direct methods over-parametrize the problem by solving for both the states and the controls. In shooting methods, given \( x_0 \), we optimize for \( u_{0:T-1} \), and obtain the next states \( x_{2:T} \) by forward simulation. Shooting methods also typically rely on gradients, possibly estimated with finite differences. Unlike direct methods, shooting methods do not handle hard constraints. For instance, to achieve a desired final state, one must penalize deviations from the final state in the cost function (e.g., with a quadratic error). Nunes et al. [97] use the method of multipliers to achieve tight constraints on final states with a shooting method.

We now discuss a well-known shooting method called Differential Dynamic Programming (DDP) [59].
It optimizes a trajectory \((x_{0:T}, u_{0:T-1})\) with respect to a cost function \(G = \sum_{t=0}^{T-1} l_t(x_t, u_t) + l_T(x_T)\), where \(l_t(x_t, u_t)\) is a state and control cost at timestep \(t\), and \(l_T(x_T)\) is a terminal cost.

The value function \(V_t(x_t)\) is defined as
\[
V_t(x_t) = \min_{u_t, \ldots, u_{T-1}} \sum_{j=t}^{T-1} l_j(x_j, u_j) + l_T(x_T).
\] (2.26)

At timestep \(T\), let \(V_T(x_T) = l_T(x_T)\). By the Dynamic Programming Principle, we can write the value function as follows
\[
V_t(x_t) = \min_{u_t} \{l_t(x_t, u_t) + V_{t+1}(f(x_t, u_t))\}.
\] (2.27)

The DDP algorithm approximates the value function with a second-order Taylor expansion around an initial trajectory \((x^{0}_{0:T}, u^{0}_{0:T-1})\) at each timestep. Given the approximate value function at timestep \(t + 1\), it finds the optimal controls at timestep \(t\), which can then be used to approximate the value function at \(t\). DDP starts from timestep \(T - 1\) and moves backwards until timestep 0 is reached. Given the new controls, a rollout is performed to obtain an improved trajectory \((x^1_{1:T}, u^1_{1:T-1})\), and the process is then repeated. The complete algorithm can be found in Levine [72, chapter 3]. The recent Iterative Linear Quadratic Gaussian (iLQG) method is closely related to DDP [117]. It is designed to be used in an online setting, therefore it uses a first order approximation of the dynamics instead of a second order approximation to reduce the cost of evaluating derivatives.

Once a control solution is found, one would like to use it in the presence of disturbances. In the next section, we discuss a common tool for this task.

### 2.5 Feedback Control

The most compelling results in systems control are restricted to linear systems only. Nonlinear systems are often linearized to leverage these results. One widely used result is the linear quadratic regulator (LQR), which provides an optimal control law for a linear approximation of the nonlinear system (see Eq. 2.11).

Given the following linear system
\[
x_{t+1} = A_t x_t + B_t u_t,
\] (2.28)
where \(x\) is the state, \(u\) is the control, \(A\) and \(B\) are the partial derivatives of the nonlinear system with respect to \(x\) and \(u\) respectively, and \(t\) is the timestep. The system is defined discretely from timesteps \(t = 0, \ldots, T - 1\).

Our goal is to find a control law that minimize the cost function
\[
J = x_T^T Q_T x_T + \sum_{t=0}^{T-1} (x_t Q_t x_t + u_t^T R_t u_t),
\] (2.29)
where \(Q\) is a symmetric positive semidefinite matrix specifying a state cost and \(R\) is a symmetric positive definite matrix specifying a control cost. We can solve for the controls with dynamic programming [14].
Chapter 2. Background

Let the value function be defined as

\[ V_T(x_T) = x_T^T Q_T x_T, \]
\[ V_t(x_t) = \min_{u_t} \{ x_t^T Q_t x_t + u_t^T R_t u_t + V_{t+1}(A_t x_t + B_t u_t) \}, \tag{2.30} \]

where \( V_{t+1}(x_{t+1}) = V_{t+1}(A_t x_t + B_t u_t) \). Expanding the value function at \( t = T - 1 \)

\[ V_{T-1}(x_{T-1}) = \min_{u_{T-1}} \{ x_{T-1}^T Q_{T-1} x_{T-1} + u_{T-1}^T R_{T-1} u_{T-1} + x_{T-1}^T A_{T-1}^T Q_T A_{T-1} x_{T-1} \]
\[ + 2x_{T-1}^T A_{T-1}^T Q_T B_{T-1} u_{T-1} + u_{T-1}^T B_{T-1}^T Q_T B_{T-1} u_{T-1} \}. \tag{2.31} \]

Differentiating the argument inside the min with respect to \( u_{T-1} \) and setting the result equal to 0, we have

\[ 2u_{T-1}^T R_{T-1} + 2x_{T-1}^T A_{T-1}^T Q_T B_{T-1} + 2u_{T-1}^T B_{T-1}^T Q_T B_{T-1} = 0. \tag{2.32} \]

Solving for the control,

\[ u_{T-1} = -(R_{T-1} + B_{T-1}^T Q_T B_{T-1})^{-1} B_{T-1}^T Q_T A_{T-1} x_{T-1}. \tag{2.33} \]

The inverse in Eq. 2.33 can be taken since the matrix is positive definite, given that both \( R_{T-1} \) and \( Q_T \) are positive definite. It can be shown that we can use the same procedure to find \( u_{T-2}, \ldots, u_0 \) [14].

The control terms are in the form \( u_t = L_t x_t \) (see Eq. 2.33). Because the control depends on the state \( x \), it is called feedback control or closed-loop control. When the control does not depend on the state, it is called open-loop control. Feedback is used to improve a controller’s robustness to disturbances. Da Silva et al. [26] introduced LQR control for a low-dimensional representation of a character model and for specific phases of bipedal walking. The use of LQR control for full-dimensional character models and for a variety of human motions has not yet been thoroughly investigated and represents fertile ground for new results in computer animation and biomechanics.
Chapter 3

Trajectory Optimization for Movements with Complex Contacts

3.1 Introduction

It has long been hypothesized that human motion can be described as optimizing simple objective functions based on task goals and energy consumption [50, 122]. Optimality is appealing as both a simple explanation for how we move and a simple parameterization for motion. The criteria for optimality, however, are not always clear. For character animation, optimality could allow the creation of many types of natural human movements from high-level specifications, without requiring labor-intensive keyframing or motion capture. Optimization of a movement is called trajectory optimization, or, equivalently, spacetime optimization. In Sec. 2.4, we reviewed different trajectory optimization techniques.

Unfortunately, progress on full-body trajectory optimization has been limited, because the problem is high-dimensional and objectives are non-linear, which often lead optimization procedures into undesirable local minima. Existing methods rely on motion capture data, which do not easily extend to generalization, or use simplified character models (such as planar models) that eliminate important degrees-of-freedom. We address the topic of generalizing motion capture data in Chapter 6. In this chapter, we address the topic of motion synthesis without motion capture data, which is an interesting and important scientific problem.

We present a method for full-body optimization of human movement. Optimization is performed using a small set of simple goals, e.g., specifying that a hand should be on the ground or that the center-of-mass should be above a particular height. These objectives are applied to short spacetime windows that can be composed to express goals over an entire animation, such as achieving standing balance or locomotion. Optimization is performed using Covariance Matrix Adaptation [49], which has shown to be convenient for our purposes to avoid falling in local minima.

The main technical contribution of this work is in the design of a set of modular and reusable objective functions for many types of full-body motion. We show that just a few, easily-interpretable—but carefully-chosen—objectives are sufficient to describe and create diverse types of motion, including walking, exercise, and breakdancing maneuvers. These skills include low and high-energy motions,

\[\text{1A publication related to this chapter appears as Al Borno et al. [7].}\]
as well as highly-rotational behaviors. Adjusting constraint parameters yields motions that are more superhuman or more typical of normal human movements. All motions reuse the same straightforward initialization procedure. To our knowledge, many of the presented motions have never been successfully synthesized by previous trajectory optimization methods.

Our work focuses on the problem of generating full-body motion, given a high-level plan, namely the sequence of spacetime windows. In this work, we specify this sequence by hand and leave higher-level plan generation as future work. In many cases, results do not perfectly match natural human motion. This may be due to the simplified nature of our musculoskeletal model and lack of interlimb contact; more accurate models [129] could help improve results. Though our optimizations are expensive, they demonstrate for the first time the feasibility of trajectory optimization for creation of a diverse set of whole-body motions. Future research could focus on accelerating the optimizations and improving realism.

3.2 Related Work

An open question in computer graphics is how to automatically synthesize physics-based character animation. To date, two main approaches have emerged to tackle this problem, trajectory optimization and controllers.

As seen in Chapter 2, trajectory optimization, or spacetime constraints [132], formulates motion synthesis as a constrained non-linear optimization. Early motion synthesis work focused on examples with a small number of joints and used key poses to direct animation [22, 83]. To broaden the class of movements, several approaches focused on highly-dynamic actions, governed by a small set of constraints [32, 78]. With the exception of challenging balancing scenarios [60], generalizing these methods to broader classes of full-body three-dimensional characters has proven difficult due to the high dimensional, non-linear, nature of these problems. Defining good objectives for low-energy motions such as walking has also proven to be difficult. Objectives that produce “natural” motions appear to be even more elusive to define.

Recent work has used motion capture data to ease authoring of new motions. Several editing methods have been proposed that adapt recordings to new situations, while maintaining physical properties [3, 40, 41, 102]. Safonova et al. [109] learn low-dimensional bases from motion data, thereby making optimization easier and capturing stylistic aspects of motion. Liu et al. [75] learn objective function parameters from motion data. Sok et al. [111] and Liu et al. [82] adapt reference input motions using randomized search. Such strategies have produced compelling results with many lifelike qualities. However, their output is limited to remain near provided input data. Our work requires neither good initialization nor motion capture to synthesize complex and contact-rich motions. The method can be used for both highly dynamic behaviors, such as flips and spins, as well as less energetic motions, including walking.

To keep problems tractable and limit the search space, a number of methods focus on cyclic behaviors. Wampler et al. [125] focus on periodic motions, for simplified characters with point feet, and optimize morphology along with gait parameters. Nunes et al. [97] use a periodic trajectory representation to generate a variety of running behaviors. Others focus on behaviors without contact, such as flying [137], and swimming [116], which further simplifies the optimization landscape. Closely related to our work, Mordatch et al. [92] optimize for motions with complex contacts, including climbing and co-operative
motions. Their method currently uses a simplified physics model and does not handle highly-dynamic motions such as flips, handspins and headpins.

As with several of these works, we also employ a shooting strategy (see Sec. 2.4.2), which means that our method can be applied to off-the-shelf dynamics simulators. Our approach works for a wide variety of cyclic and acyclic movements, allows composition of long motion sequences, and does not require any special treatment of complex contacts. Although we focus on human motion, our work can be easily extended to other types of creatures.

An alternative to trajectory optimization is to use action-specific controllers. This approach has been used to synthesize numerous locomotion and athletic behaviours [56, 107, 135, 142] with recent methods focused on low-dimensional parameterizations [2, 23, 25, 29, 86, 139] and robust navigation of uneven terrain [91, 138]. Our method is complementary to this work. Open-loop movements generated by our system could be used as the basis for controllers or libraries of control primitives. This would enable reuse of results from our system in interactive applications. The proposed look-ahead strategy is also closely related to model-predictive methods that plan over short windows into the future [27, 28, 91, 94].

The objectives we propose could also be used by methods that use optimization to tune settings for predefined controllers [23, 127, 129, 141]. Though we leave the addition of feedback for future work, our method requires no design of action-specific controller before new motions can be generated.

3.3 Methodology

The chapter uses the following methodology:

- A cost function is designed for each desired behavior.
- Optimization is performed over spline knots that represent a desired trajectory.
- The desired trajectory is tracked with a PD controller, which results in an actual trajectory.
- The cost function is evaluated on the actual trajectory.
- Optimization is performed with a stochastic, evolutionary algorithm on sequentially overlapping pairs of windows.

3.4 Parameterization and Optimization

We optimize full-body character motion in a physically-simulated environment, using only a small set of high-level objectives. Goals for the motion are expressed in terms of an objective function \( E \), with energy terms \( E_i \) for motion properties such as stride length and angular momentum. The specific terms of the energy depend on the type of desired motion.

We use a character with 41 degrees-of-freedom in the kinematic pose \( q \), and represent a motion as time-varying poses \( q_{1:T} \). Objectives \( E_i \) are combined by weighted combination with weights \( w_i \):

\[
E = \sum_i w_i E_i. \tag{3.1}
\]
Trajectory optimization applied directly to joint torques or to poses is very difficult, due to the highly-discontinuous relationship between these quantities and task completion. For this reason, we parameterize the motion in terms of a reference trajectory \( \hat{q}_{1:T} \). The reference trajectory \( \hat{q}_{1:T} \) is represented as a cubic B-spline. The free variables in optimization are the key-poses of the spline.

Given a reference trajectory:

\[
\hat{q}_{1:T} = Bs, \tag{3.2}
\]

where \( B \) is a spline basis matrix, the output motion \( q_{1:T} \) is computed by simulation. At each time index \( t \), the control torque \( u_t \) for a 1D joint degree-of-freedom (DOF) is determined by linear control:

\[
u_t = k_p(\hat{q}_t - q_t) - k_d\dot{q}_t, \tag{3.3}\]

where \( q_t \) and \( \hat{q}_t \) are the current and reference values of the joint angle, and \( k_p \) and \( k_d \) are gains.

The complete optimization for a given window is then:

\[
s^* = \arg\min_s E. \tag{3.4}\]

Key-poses are parameterized by Euler angles for spherical joints.

We divide the optimization problem into spacetime windows \([22, 76]\) because directly optimizing a long-duration motion would be prohibitively expensive. We find that windows are a convenient way to specify complex motions. Windows can be used for specific phases of gait or the relevant phases of other motions. Each window represents a fixed duration of the motion; we use 0.5 s windows, although further exploration could result in a different choice. We do not consider problems where a long-term look-ahead is required; hence, we employ a simple, advancing optimization schedule. To start, the first two windows of the motion are optimized together. Then, results for the second window are discarded and a new optimization is performed over the second and third window, holding the first window fixed. This process advances along pairs of windows until the entire motion is complete. This pairwise schedule allows each window to take the goals of the next window into account. As we discuss later, although we use fixed duration windows, this approach enforces only a loose constraint on motion timing. Multiple passes \([22, 76]\) could be used for problems requiring more look-ahead.

We use Covariance Matrix Adaption (CMA) \([49]\) to perform optimization, initializing all problems with a zero vector, which corresponds to an upright standing posture (i.e., the static pose \( \hat{q}_{1:T} = 0 \)). Hence, unlike previous work, no user effort or motion data is required for initialization. CMA is a derivative-free optimization algorithm that samples and evaluates candidate solutions from a normal distribution. The normal distribution is updated based on a number of elite samples, and the process is repeated until convergence or a maximum number of iterations is reached. In practice, we ran CMA with a maximum of 18,000 iterations and 16 samples. The optimization is performed in parallel on a dual quad-core Intel E5355 CPUs and takes roughly 20 minutes per window. We use OpenMP to parallelize CMA across all cores. Examples described in this chapter took between 1 and 15 hours, depending on the number of windows used. Once a window has been optimized, we reset the subsequent window and repeat the above described process. For the motions described in this chapter, all iterations of CMA are typically required for to converge on a solution.

We select servo gains \( k_p, k_d \) by hand and use the same values for all motions described in this chapter;
Figure 3.1: Spacetime window optimization schedule. We define the objective function for a motion by creating a sequence of spacetime windows. Each window defines the objective function for a fixed interval. Optimization is performed with an advancing schedule: the motion during windows 1 and 2 are optimized. Then the motion in windows 2 and 3 are jointly optimized, and so on, until every window has been visited.

consult Sec. 3.11 for PD gains values and simulator details. For a 0.5 s duration window, we place a spline knot every 0.1 s. The optimization limits the range of reference trajectories for all joints (see Table 3.1). Torque limits are enforced by clamping joint torques in the simulation; flips use torque limits of ±400 Nm, all other motions use ±200 Nm. The actual generated motion \( q \) has a joint limit only on the toe. Since we jointly optimize two windows at a time, and our character model has 35 actuated DOFs, the total number of variables for each pairwise optimization is 350. Motions where left-right body symmetry is enforced require 220 variables.

In the remainder of the chapter, we describe objectives required to produce a variety of motions. We first introduce objectives needed to generate variations of in-place balancing behaviors. More complex locomotion and acrobatic motions are described in subsequent sections. All motions can be seen in the accompanying video [6].

3.5 Balancing and Getting Up

We begin with a balance motion, which aims at placing the character in standing balance. We then show how it can be used to produce motions in which the character gets up from lying-down positions. While the objectives we use are quite simple, we find it surprising that these complex motions, which involve many contact changes, can be achieved from such simple specifications and optimizations.

3.5.1 Balancing

The standing balance goal entails optimizing the weighted sum of four objective terms over a given window. The first term

\[
E_{\text{COM}} = (c_T - \bar{c}_T)^2
\]

moves the center-of-mass (COM) at the end of the window \( c_T \) to a target position \( \bar{c}_T \) at a user-specified height over the approximate centroid of the base-of-support (we use the average positions of the centers of the feet projected on the plane). In our examples, the target height is 1.09 m. The second term keeps the feet on the ground:

\[
E_{\text{feet}} = y_{\text{leftFoot},T}^2 + y_{\text{rightFoot},T}^2.
\]
Chapter 3. Trajectory Optimization for Movements with Complex Contacts

19

Figure 3.2: Getting up motions. Top row: Starting from a supine position, optimizing over a sequence of four balancing windows yields a very athletic getting-up motion. The objective function is the same in each window, except for the rest pose term used in the final window. For each window except the last, the first and middle frames are shown. For the last window, the first and last frames are shown. Bottom row: A more normal getting-up motion from a prone position is produced by partially specifying contacts, for various windows, as described in the text. Each image shows the first frame for the corresponding window, except for the last image which shows the final frame.

The position of the feet on the plane is not specified. Because the first two terms are only active at the end of the window, they aim to produce a motion that ends in a balanced state, without constraining how the character gets there. Note that the exact instant that standing balance is attained is not specified; the optimal motion could reach the final pose before the end of the window.

The third term penalizes control torque over the entire window:

$$E_{\text{torque}} = \sum_{j,t} u_{j,t}^2,$$

where the summation is over all joint DOFs $j$ and time indices $t$ within the window. This term is used in all motions in this chapter.

Finally, a rest pose term

$$E_{\text{restPose}} = \sum_{j \in \mathcal{J}} \sum_t (q_{j,t} - \bar{q}_{j,t})^2$$

is used to ensure that the character is standing upright, with arms in desirable positions. The variable $q_{j,t}$ denotes the angle representation for joint $j$ at time $t$, and $\mathcal{J}$ denotes the set \{lumbar, thorax, shoulder, elbow\}. We find that constraining this subset of joints, when combined with the other terms, is sufficient to achieve a reasonable rest pose. This term is only used when a balance window is used as the final window of the optimization.

3.5.2 Getting up: Athletic

The balance objective in the previous section can be used directly for getting-up (Fig. 3.2 top row), since it aims to end the window in standing balance. Despite the lack of explicit contact planning, the optimization successfully finds viable contact sequences needed to produce upright motions. In the accompanying video [6], we show examples of getting up from prone and supine positions. The resulting
motion is very quick and exhibits highly-athletic abilities.

3.5.3 Getting up: Low-Energy

We can produce more typical getting-up motions by performing the optimization in a sequence of four windows, and by partially specifying the contacts for each window (Fig. 3.2, bottom row). We divide the optimization in four windows with different objectives to carefully guide it to a realistic solution. It is desirable to reduce the amount of guidance required, which could happen by designing better optimization methods or more realistic humanoid models.

All windows include the torque-squared efficiency term $E_{\text{torque}}$ (Eq. 3.7) and use variants of the balance objective. In the first window, we keep the character prone, while raising its body. To accomplish this, three objectives are used: torque minimization $E_{\text{torque}}$, contact constraints, and target COM height:

$$E_{\text{ground}} = \sum_t \left( y_{\text{hand},t}^2 + y_{\text{rhand},t}^2 + y_{\text{ltoe},t}^2 + y_{\text{rtoe},t}^2 + y_{\text{lknee},t}^2 + y_{\text{rknee},t}^2 \right)$$

$$E_{\text{COMh}} = (y_{\text{COM},T} - \bar{y}_{\text{COM},T})^2.$$  (3.9)

The $E_{\text{ground}}$ objective penalizes non-zero height for each corresponding point on the body during the entire window. The $E_{\text{COMh}}$ objective raises the COM to a target height $\bar{y}_{\text{COM},T} = 0.3$ m at the end of the window. In the second window, the character raises the right side of its body by pushing off the ground with its left limbs. This is done with the same objectives as above, but removing the left knee term, and increasing the COM target height to 0.5 m. In the third window, we deactivate all but the left hand, left toe, and left knee constraints, and raise the COM target to 0.55 m. The last window is a normal balance window.

We use a similar approach to generate a getting-up motion starting from a supine position. During the first window, objectives are used that encourage the character to get its back off the ground. We also place the hands and the feet on the ground during the entire window. During the second window, objectives keep the left hand on the ground and we include $E_{\text{COMh}}$ with a COM target height of 0.45 m. The last window uses the balance objective.

3.6 Walking

In this section, we show that full-body walking can be produced entirely through optimization with simple and human-interpretable objectives, and without relying on motion capture. We use one window for each swing phase (Fig. 3.3, i.e., one period from toe-off to heel-strike. The same objective function is used for each window, with feet handling swing and stance tasks alternating from window to window.

In total, six objective terms are used in each window. These terms control the desired distance and direction of travel, heading, balance, foot contact, angular momentum, and energetic efficiency. Different parameters to the heading and distance/direction terms are used to produce forward walking, backward walking, and turns.

We specify a desired step-length and direction, by a vector $\mathbf{v}$. For example, if we wish the character
to move 0.5 m along the $x$-direction in one swing phase, then $v = [0.5, 0, 0]^T$. We evaluate this as

$$E_{\text{stepdist}} = ||(c_T - c_1) - v||^2,$$  \hspace{1cm} (3.11)

where $c_1$ and $c_T$ denote the center-of-mass (COM) location at the start and end of the window.

The character’s heading is controlled by

$$E_{\text{heading}} = (\alpha_T - \bar{\alpha}_T)^2,$$  \hspace{1cm} (3.12)

where $\alpha_T$ and $\bar{\alpha}_T$ are the actual and the desired character heading of the character’s pelvis, at the end of the window. We define heading as the right-handed angle about the axis perpendicular to the ground plane, measured from the $x$-axis.

For heel-strike at the end of the window, we use

$$E_{\text{swingHeel}} = y_{\text{swingHeel}, T}^2,$$  \hspace{1cm} (3.13)

where $y_{\text{swingHeel}, T}$ is the position of the swing foot heel.

We want the stance foot to stay in contact with the ground, but we also want foot rolling; therefore we penalize vertical displacement of the stance toe $y_{\text{stanceToe}}$ from the ground over the whole window:

$$E_{\text{stanceToe}} = \sum_t y_{\text{stanceToe}, t}^2,$$  \hspace{1cm} (3.14)

At heel-strike, the position of the COM projected on the plane is roughly at the center of the base of support [32]. For this reason, we include $E_{\text{COM}}$ (Eq. 3.5), with a desired COM height that depends on the desired step length.

For walking motions, we modify the pose parameterization slightly, following Wang et al. [127]. Specifically the shoulder angle in the sagittal plane is coupled to the hip angle as

$$q_{\text{shoulder}} = \omega (q_{\text{rhip}} - q_{\text{lhip}})$$

$$q_{\text{rshoulder}} = -q_{\text{shoulder}},$$

where $\omega$ is determined by the optimization. This reduces the number of free variables in the walking motion optimizations to 348. Torques are also penalized using $E_{\text{torque}}$.

As demonstrated in the accompanying video [6], changing the heading direction and step distance can create forward walking, backward walking, and various sharp turns. Results are not entirely natural, in ways that reflect the biomechanical simplicity of the model and objectives.

### 3.7 Inverted Movements

In this section, we describe the objectives required to create a simple handstand and hand walk.
Chapter 3. Trajectory Optimization for Movements with Complex Contacts

Figure 3.3: Walking. First/middle frames of two successive walking windows. The objective function is the same in each window, except for swapping roles of stance/swing feet. Changing parameters of relevant objective function terms gives backward walking and turns, as shown in the accompanying video [6].

3.7.1 Handstand

A handstand window requires five energy terms that are applied over the entire window. Terms keep the hands on the ground and encourage the feet and the COM to be as high as possible:

\[ E_{\text{hands}} = \sum_t y_{\text{hand},t}^2 + \sum_t y_{\text{rhand},t}^2 \]  
\[ E_{\text{feetV}} = -\sum_t y_{\text{lfoot},t} - \sum_t y_{\text{rfoot},t} \]  
\[ E_{\text{COMV}} = -\sum_t y_{\text{COM},t} \]  

To penalize motions with head/ground contact, we use an objective:

\[ E_{\text{headHeight}} = \sum_t \delta(y_{\text{head},t}, \zeta_{\text{head}}) \]  
\[ \delta(y, \zeta) = \begin{cases} 
C & \text{if } y < \zeta \\
0 & \text{otherwise} 
\end{cases} \]

that keeps the head height \( y_{\text{head},t} \) above the threshold \( \zeta_{\text{head}} = 0.35 \text{ m} \) for the entire window by applying a large penalty \( C \) to frames that violate the constraint. We use \( C = 1000 \). The \( E_{\text{torque}} \) (Eq. 3.7) term is also included.

In Fig. 3.4 we illustrate a case where the characters starts from standing balance, goes to the handstand, and returns to standing balance. The standing-balance-to-handstand transition window contains two terms, one to ensure that the COM stays above a height threshold \( \zeta_{\text{COM}} = 0.6 \text{ m} \):

\[ E_{\text{COMHeight}} = \sum_t \delta(y_{\text{COM},t}, \zeta_{\text{COM}}) \]

and the torque penalty \( E_{\text{torque}} \). This transition window allows the character to enter a suitable position to begin the handstand motion; omitting it leads to a motion that achieves the handstand with superhuman speed and interpenetration. The handstand-to-standing-balance transition windows contain the same objectives as the balance motion with the addition of the \( E_{\text{COMHeight}} \) term. We use a similar approach to create transitions with other motions.

The handstand motion is achieved in an aggressive manner: the character pivots entirely on its
3.7 One-Handstand

We also demonstrate the ability to stand on one hand. This is achieved simply by removing the constraint on the right hand in $E_{\text{hands}}$ (Eq. 3.17). As shown in the accompanying video [6], the character achieves a one-handed handstand, raising the right hand in the air while remaining balanced.

3.7.3 Hand walk

We can combine elements of the handstand and walking objectives to generate a motion where the character walks on its hands. To do this, we reuse the $E_{\text{hands}}$ (Eq. 3.17) but only consider a single contact hand in each window, alternating hands between windows. Including $E_{\text{heading}}$ (Eq. 3.12) and $E_{\text{steppdist}}$ (Eq. 3.11) terms introduced in the walking objective generates the hand walk (Fig. 3.5). Stylistically, the hand walk and handstand are similar; transitions are somewhat athletic and the character is wobbly once inverted.

3.8 Rotational Movements

We now describe novel highly-rotational movements. In each case, large rotations are achieved by using an objective which encourages high angular momentum about a given axis.
3.8.1 Headspin

The headspin motion is a difficult behavior seen in breakdancing routines. During this motion the character spins quickly on its head (Fig. 3.6). The headspin employs the same objectives as the handstand, except that the $E_{\text{headHeight}}$ (Eq. 3.20) term is modified to ensure that the head remains close to the ground (i.e., below 0.35 m).

In addition, we seek to maximize the angular momentum about the vertical axis during the entire window:

$$E_{AM} = -\sum_t L^2_{y,t}. \quad (3.23)$$

We also penalize horizontal motion of the COM between the start and end of the window:

$$E_{\text{horiz}} = ||c_{\text{horiz,T}} - c_{\text{horiz,1}}||^2. \quad (3.24)$$

In the accompanying video [6], we show a sequence where the character starts from standing balance, spins on its head, and returns to standing balance. Two windows are necessary for the character to make a transition from standing balance to the headspin. The first window only contains the $E_{\text{torque}}$ term. For the second window we add the $E_{\text{hands}}$ term (Eq. 3.17). This has the effect of lowering the character’s COM and encouraging hand placement on the ground in preparation for the headspin. To end the headspin motion, we use one window containing $E_{\text{torque}}$ only. This results in the character lying on the ground. From that point on, balance windows (Sec. 3.5) are used to transition to standing balance.

We note that the character transitions from standing balance to the headspin very rapidly. A breakdancer would need more time to carefully position himself to start the headspin. This can be taken into account by designing additional transition windows.

3.8.2 Handspin

The handspin motion is a breakdancing maneuver where the character spins on its hands (Fig. 3.7). The handspin cost function consist of five terms, each described below.

As with the headspin, the handspin requires building up substantial whole-body angular momentum about the vertical axis. To accomplish this, we attempted to include objective terms that maximize angular momentum (e.g., Eq. 3.23), however this generated inhumanly fast spinning behaviors. Instead,
we found that using the objective:

\[ E_{AMRange} = - \sum_t \phi(L_{y,t}, \zeta_l, \zeta_u) \]  \hspace{1cm} (3.25)

\[ \phi(L_{y,t}, \zeta_l, \zeta_u) = \begin{cases} L_{y,t}^2 & \text{if } \zeta_l < L_{y,t} < \zeta_u \\ 0 & \text{otherwise} \end{cases} \]  \hspace{1cm} (3.26)

which seeks maximum thresholded cumulative angular momentum, over a window, produced more appealing results. In all our demos, we use \( \zeta_l = -10^9 \text{N.m.s} \) and \( \zeta_u = 0 \text{N.m.s} \), which were found empirically.

To ensure the head does not touch the ground during the handspin, we use \( E_{headHeight} \) (Eq. 3.20) with a 0.35 m threshold. We need the \( E_{hands} \) (Eq. 3.17) term to force the hands on the ground. With the objective terms covered so far, we found that the body tended to be too close to the ground for a typical handspin. We include \( E_{COMV} \) (Eq. 3.19) to address this. The \( E_{torque} \) (Eq. 3.7) term is included.

Although most aspects of the motion are not explicitly planned, many strategies for generating angular momentum, matching those of breakdancers, emerge automatically. First, the character drops to the ground and spreads its legs. Next, legs are moved in opposite directions; the top leg is quickly moved forward while the bottom leg simultaneously pushes backwards off the ground. Once spinning, the character uses its feet to push off the ground at intermittent intervals to continue rotating.

3.8.3 Flips

The flip motions consist of a rotation in the sagittal plane while the character is airborne (Fig. 3.8). This is the first time that flips have been generated without prior data; indeed, de Lasa et al. \cite{29} report being unable to identify suitable features for creating flip controllers.

We divide the motion into four windows and enforce left-right body symmetry in the reference trajectory. All the windows include \( E_{headHeight} \) (Eq. 3.20) and \( E_{COMHeight} \) (Eq. 3.22) to prevent the head and the COM from getting too close to the ground (i.e., less than 0.6 m). The \( E_{torque} \) (Eq. 3.7) term is also included. No additional terms are included in the first and third windows.

During the second window, we use \( E_{COMh} \) (Eq. 3.10) with a target COM height between 1.4 m and 1.6 m, depending on the desired style of the flip.

To generate a backwards flip, we include a term that maximizes angular momentum about the COM in the sagittal axis \( d \) during the entire window:

\[ E_{AMRange} = - \sum_t \phi(L_{d,t}, -\infty, 0). \]  \hspace{1cm} (3.27)

The only difference between the backwards and forwards flip is the sign of the angular momentum term
Chapter 3. Trajectory Optimization for Movements with Complex Contacts

Backflip. The backflip is produced by controlling COM height, contacts, and maximizing angular momentum. The first and middle frames of windows 2 to 4 are shown, together with the final frame.

This is a case where we found it difficult to resolve different objectives “fighting” each other. Specifically, if a large weight is used for $E_{AM\text{Range}}$, the character rotates but does not become airborne; however, if a large weight is used for $E_{COMh}$, the character does not rotate sufficiently. Following Wang et al. [127], we use a thresholded quadratic term:

$$E_{COMQ} = Q(y_{COM,T} - \bar{y}_{COM,T}, \epsilon)$$  \hspace{1cm} (3.28)

$$Q(d, \epsilon) = \begin{cases} d^2 & \text{if } |d| > \epsilon \\ 0 & \text{otherwise} \end{cases} \hspace{1cm} (3.29)$$

with $\epsilon = 0.05$m and set the weight on Eq. 3.28 to be very large (see Table 3.2). This objective penalizes COM displacement over the threshold and provides no penalty otherwise.

The fourth window uses the balancing objectives that were introduced earlier (Sec. 3.5.1). The accompanying video [6] shows several flip variations. When flipping backwards, the arms swing forward quickly during decompression to generate needed angular momentum prior to the ballistic motion. Near the apex of the aerial phase, the character tucks rapidly to increase angular velocity, before extending its legs for landing. Upon landing, its arms counter-rotate to aid balance. To generate needed rotations for the forward flip, the character takes an in-place hop and enters a tuck. The tuck (which resembles a sitting position) is held for nearly the entire movement. Hence, the character lands with bent knees and its hands can be seen touching the ground. A tighter tuck might help avoid high-loads on joints exhibited in the current motion. Alternatively, the forward flip could be initiated once the character has a greater forward velocity.

3.9 Ground Movements

This section describes two novel motions that keep hands and feet on the ground: push-ups and crawling.

3.9.1 Push-ups

Push-ups use four objectives. The first two encourage hand and toe links to remain on the ground. For the hands, we use Eq. 3.17 while

$$E_{toe} = \sum_{t} y_{\text{toe},t}^2 + \sum_{t} y_{r\text{toe},t}^2$$  \hspace{1cm} (3.30)
Chapter 3. Trajectory Optimization for Movements with Complex Contacts

Figure 3.9: Crawl. First/middle frames of successive crawling windows.

is used for the toes. To lower and raise the character’s body, we use

\[ E_{COMh} = (y_{COM,T} - \bar{y}_{COM,T})^2 \]  

(3.31)
and vary the COM target height between windows. In our examples, \( \bar{y}_{COM,T} \) is set to 0.25 m and 0.45 m. The \( E_{torque} \) (Eq. 3.7) term is also used.

3.9.2 Crawling

Crawling is a quadrupedal gait that uses both arms and legs for locomotion (Fig. 3.9). We divide crawling into two phases. In each phase, diagonally opposite limbs are considered to be in either stance or swing, with roles alternating from phase-to-phase. We optimize one window for each phase and use eight objectives for each window.

To encourage stance limbs to remain firmly planted we use two sets of objectives. Stance limb motion parallel to the ground plane is penalized by objectives

\[ E_{stanceHand} = ||p_{hand,T} - p_{hand,1}||^2 \]  

(3.32)
\[ E_{stanceKnee} = ||p_{knee,T} - p_{knee,1}||^2 \]  

(3.33)
\[ E_{stanceToe} = ||p_{toe,T} - p_{toe,1}||^2 \]  

(3.34)
where \( p \) denotes the position of each stance limb (i.e., hand, knee, toe), projected onto the ground plane, at the start and end of each window. Stance limb motion perpendicular to the ground plane is penalized using \( E_{ground} \) (see Eq. 3.9). We reuse previously described objectives for other aspects of the motion: \( E_{headHeight} \) (Eq. 3.20) and \( E_{COMHeight} \) (Eq. 3.22) ensure the head and the COM are at least 0.3 m above the ground, while \( E_{stepdist} \) (Eq. 3.11) encourages movement in the desired direction. The \( E_{torque} \) (Eq. 3.7) term is also included.

3.10 Longer Motions

It is straightforward to generate longer motions by concatenating different spacetime windows. Previous sections have presented several combinations of getting-up with acrobatic moves. Here we describe several longer sequences obtained by concatenating previously described windows. Concatenation is simplest when one window ends in a good initial pose for the next window; for example, the character can transition directly from a handstand window to a hand walk window. Other combinations require transition windows (Sec. 3.7.1).
3.10.1 Backflip-to-handstand

Fig. 3.10 demonstrates a backflip followed by a handstand, followed by standing balance. No transition windows were required to generate this sequence. It is interesting to note that the character can enter the handstand phase even if it is entirely airborne.

3.10.2 Multiple Backflips

Fig. 3.11 demonstrates a challenging sequence of two backflips. The transition from the first to the second backflip is achieved by the balance objectives, with a COM target height of 0.8 m for $E_{COM}$ to ensure that the character is slightly crouched. At this point, the character is in a good position to jump, hence we proceed with window 2 of the backflip motion. A minor modification is required to generate a flip motion where the character uses its hands to rotate. Specifically, the $E_{headHeight}$ term is modified to allow the head to get as close to 0.35 m from the ground.

3.10.3 Long Sequence of Moves

In the accompanying video [6], we demonstrate several long motions sequences that combine many previously described actions.

3.11 Simulator Details and Parameters

Our simulator uses Featherstone’s algorithm and the semi-implicit scheme of Guendelman et al. [44] ($1e^{-3}$ s). Ground contact uses an frictional ($\mu = 1$) inelastic impulse-based model. Height and weight for the correspond to a 50$^{th}$ percentile North American male. Skeletal dimensions/link masses are taken from Winter [131]. Link inertias are calculated using uniform density shapes scaled to match skeletal dimensions.
Chapter 3. Trajectory Optimization for Movements with Complex Contacts

Figure 3.11: Multiple Backflips. A sequence of backflips is generated by combining windows from the backflip and balance motions. The first and middle frames of each window of the sequence are shown. The last image is the final frame.

Table 3.1: Joint limits and PD gains. All limits are in radians. In all cases we use $k_v = 0.1k_p$. Special cases: neck limits are $\pm 0.4$ for the head spin; lumbar limits are $\pm 0.1, \pm 0.1$ and $\pm 0.05$ for walking and push-ups; hip limits are $\pm 0.2, \pm 0.2$ and $\pm 1.5$ for walking; wrist limits are $\pm 0.2$ for walking; knee limits $-0.5, 0$ for push-ups.

<table>
<thead>
<tr>
<th>DOF</th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$k_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>neck</td>
<td>$\pm 0.1$</td>
<td>$\pm 0.1$</td>
<td>$\pm 0.1$</td>
<td>1000</td>
</tr>
<tr>
<td>lumbar</td>
<td>$\pm 0.8$</td>
<td>$\pm 0.8$</td>
<td>$-0.8, 2.8$</td>
<td>1000</td>
</tr>
<tr>
<td>thorax</td>
<td>$\pm 0.1$</td>
<td>$\pm 0.1$</td>
<td>$\pm 0.1$</td>
<td>1000</td>
</tr>
<tr>
<td>shoulder</td>
<td>$\pm 3.14$</td>
<td>$\pm 0.8$</td>
<td>$\pm 3.14$</td>
<td>200</td>
</tr>
<tr>
<td>elbow</td>
<td>$-$</td>
<td>$-$</td>
<td>$0.2, 8$</td>
<td>200</td>
</tr>
<tr>
<td>wrist</td>
<td>$\pm 1$</td>
<td>$-$</td>
<td>$\pm 1$</td>
<td>100</td>
</tr>
<tr>
<td>hip</td>
<td>$\pm 0.6$</td>
<td>$\pm 0.8$</td>
<td>$\pm 1.5$</td>
<td>700</td>
</tr>
<tr>
<td>knee</td>
<td>$-$</td>
<td>$-$</td>
<td>$-2.8, 0$</td>
<td>700</td>
</tr>
<tr>
<td>ankle</td>
<td>$\pm 0.6$</td>
<td>$-$</td>
<td>$-0.2, 1.2$</td>
<td>100/150 ($x/z$)</td>
</tr>
<tr>
<td>toe</td>
<td>$-$</td>
<td>$-$</td>
<td>$0.3$</td>
<td>10</td>
</tr>
</tbody>
</table>

3.12 Discussion

We have presented the first method for full-body trajectory optimization without relying on motion capture, specified key-poses, or periodicity. We show that a small set of simple objectives is sufficient to synthesize a wide range of movements. Some of the most complex movements, such as breakdancing maneuvers and flips, have not previously been generated by any other physics-based method.

An important characteristic of our method is that it does not require contacts to be pre-defined; contacts that are specified are of the form “this end-effector should touch the ground somewhere at the end of this window.” Additionally, our method does not require the user to specify complete full-body pose at regular intervals. Combined, these factors makes it easier to generate a large variety of behaviors.

Overall, we did not find tuning the parameters to be particularly time-consuming. As can be seen in Sec. 3.11 most of the weights are powers of ten, and many of the objectives have the same magnitude across different motions. We believe it should be straightforward to apply our approach to other problems, such as for other types of animals [25, 125], and other types of terrain [25, 91, 138]. To create new motions, we first define the approximate sequence of required windows and objectives, starting from the end state of an existing behavior, and optimize each window in sequence. We author each window in sequence, caching results for previous windows, avoiding costly recomputation. As we gained experi-
Table 3.2: *Objective weights used for all examples.*

<table>
<thead>
<tr>
<th>Weight</th>
<th>Getting-Up</th>
<th>Walking</th>
<th>Handstand</th>
<th>Hand Walk</th>
<th>Head Spin</th>
<th>Hand Spin</th>
<th>Flips</th>
<th>Push-Ups</th>
<th>Crawling</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_{\text{torque}} )</td>
<td>( 10^{-7} )</td>
<td>( 10^{-8} )</td>
<td>( 10^{-6} )</td>
<td>( 10^{-6} )</td>
<td>( 10^{-8} )</td>
<td>( 10^{-6} )</td>
<td>(-)</td>
<td>(-)</td>
<td>( 10^{-1} )</td>
</tr>
<tr>
<td>( w_{\text{feet}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{ground}} )</td>
<td>( 10^{2} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>( 10^{3} )</td>
</tr>
<tr>
<td>( w_{\text{restPose}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{COM}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{COMV}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{COMh}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{COMHeight}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{hands}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{heading}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{headHeight}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{feetV}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{StepDist}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{AM}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{horiz}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{toe}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{stanceHand}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{stanceKnee}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{stanceToe}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>( w_{\text{swingHeel}} )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
</tbody>
</table>

enced designing motions using this approach, we found that we could reuse the small set of objectives we described in the chapter for a wide range of motions.

One limitation of our work is that the optimization is over a short time horizon. Some motions, like running to make a long jump over an obstacle, require longer-term planning. Future work could involve combining our approach with long-term planning so that the character can autonomously navigate complex environments. Another possibility is to design an interactive system where an animator can build long sequences of motions by combining individual motions as building blocks [22]. The animator could have control over the timing of the spacetime windows and the parameters of a particular motion (e.g., the height of a backflip).

In most cases, we have not achieved perfectly natural human movement. For example, the synthesized walking motion includes some awkward hip movement. This can be avoided by restricting the hip joint limits in the reference trajectories (Table 1) for walking, but not for the other motions that need a larger range of movement (e.g., flips). Such hand-tuning may no longer be necessary if we use biologically-inspired objectives and musculotendon models [129]. We suspect that doing so can greatly improve the style of the motions in our work, as we currently use a very rough approximation of physical energy as an objective (Eq. 3.7). Enabling inter-limb contacts in our simulator would also remove interpenetration artifacts from results. Our humanoid model uses the same torque limits for all the joints in the body, which implies, for example, that the toes can apply the same torques as the hips. A more realistic model would have torques limits as a function of the specific joint and the joint angle [15, 98]. This would help avoid some of the undesired superhuman behavior obtained with our current model.
At present, it remains unclear what is the best way to stylize physics-based animation. Previous methods, such as specifying specific poses (e.g., [24]), or learning styles from examples [75] should be applicable in our approach as well. Another possibility is to include additional objectives that seek to replicate reference recordings (e.g., [82]).

Our approach shows that full-body trajectory optimization is feasible, but currently quite slow. An open problem is how to speed-up the trajectory optimization so that it can be used in a real-time setting. In the next chapter, we will explore the use of controllers for motor skill acquisition because they are better suited for real-time settings.
Chapter 4

Feedback Control for Rotational Movements in Feature Space

4.1 Introduction

Trajectory optimization synthesizes motion by solving a constrained nonlinear optimization problem (see Sec. 2.4). Safonova et al. [109] and Fang et al. [32] use motion capture and pre-scripted animations to facilitate the optimization of flips and leaps. However, their approach can only generate motions that are near the input motions. In Chapter 3, we developed a sampling method to synthesize contact-rich and highly dynamics motions such as headpins and flips. One problem is that the trajectory optimization solution lacks robustness and generality. One can use feedback around the trajectory to obtain a more robust controller [28, 81, 94, 139], but it still falls short of being general. For instance, trajectory optimization can be used to generate a flip with a certain apex. Generating a flip with a different apex requires a new optimization, which takes several minutes. Because they are pre-computed, controllers are better suited than trajectory optimization for real-time applications.

The main focus of previous research in physics-based character animation has been on balancing [2, 86] and locomotion [91, 127, 125, 138, 142]. Rotation is a key component of many of the most impressive movements of ballet performers, breakdancers and gymnasts. There has been some previous research on rotational movements, but we will see that the solutions lack generality and robustness (see [56, 133, 134] and Sec. 3.8). The problem is particularly challenging because of the need to precisely regulate the character’s global orientation, angular momentum and inertia. Most existing methods rely on motion capture, which limits their generality, or on pre-scripted animations, which require substantial manual effort. Recently, some researchers have designed control algorithms for landing and rolling movements as a first step towards the study of rotational dynamics. Our work constitutes a further step towards this goal.

This chapter presents feature-based controllers for a wide variety of rotational movements, including cartwheels, dives and flips. Most of the rotations we demonstrate are planar, but the controllers are fully three-dimensional. Our control laws are intuitive to design and general, which we demonstrate by providing examples of the flip controller with different apexes, the diving controller with different

\[1\]

A publication related to this chapter appears as Al Borno et al. [8].
heights and styles, the cartwheel controller with different speeds and straddle widths, etc. The control strategies are often used across different types of movements. For instance, almost the same controller that makes a character land on its feet in a backflip also allows a diver to enter the water in a fully extended and straight posture. The identical controller can be applied to characters with very different body proportions. Our controllers do not rely on any input motion or offline optimization, and run at interactive rates. However, some effort is required to synthesize the controllers since they are hand-tuned.

We place a particular emphasis on the robustness of the controllers. As is well-known, time-invariant (or state-based) controllers are more robust to disturbances than time-based controllers since they do not attempt to adhere to the timing of a pre-defined motion. In this work, we show that controllers for rotational movements can be made robust by substituting time with a phase variable that positions the character in the revolution.

4.2 Related Work

The two general methods to design controllers are joint space control and feature-based control (also known as task space control). One can synthesize controllers for specific actions such as balancing, walking and jumping, or to track a reference motion [2, 28, 60, 71, 129].

Manually designing controllers in joint space can be very difficult and time-consuming due to the nonlinear interaction between individual joints. Offline optimization eases the process [5, 127], but manual specification of the controller’s structure is still needed. Once a controller is synthesized, it is usually specific for a character [55], and changing the properties of the movement is unintuitive or requires a new optimization. In early work, Hodgins et al. [56] and Wooten et al. [133, 134] synthesized dives, flips and handsprings in joint space. However, the controllers are highly specific because motion properties such as the style, the speed, the position of the apex of the flip, the height of the dive, etc., cannot be modified without re-tuning the parameters. Recently, Sehoon et al. [45] developed a control strategy for landing and rolling movements based on the optimization of an abstract model. In contrast, we found that control laws based on simple mechanics could generate movements involving an airborne stage such as flips and dives.

Feature-based controllers offer an abstraction layer to individual joints, where control is specified in terms of high level features of the motion, such as the trajectory of the center-of-mass [2]. Feature-based controllers are usually more intuitive to design and more robust to changes to the shape of the character than joint space controllers. Previous researchers have used feature-based controllers to generate balance [2, 80], walking [29, 138, 136], jumping [29], and recently rolling movements [17], but have not yet been able to generate highly dynamic movements such as cartwheels and flips [17, 29]. In this chapter, we present feature-based controllers for a wide variety of rotational movements. We show that the controllers succeed for a large range of inputs, which demonstrate their generality.

In the control systems literature, virtual constraints have become an important tool to design time-invariant controllers for various problems [87], including bipedal locomotion [1] [31, 12]. Let $\mathbf{q}$ be the vector of generalized joint positions of a mechanical system. The method of virtual constraints expresses the controlled variables $\mathbf{q}_i$, as a function of a phase variable $\theta(\mathbf{q})$:

$$\mathbf{q}_i = h_i(\theta),$$  \hspace{1cm} (4.1)
instead of expressing them as a function of time \( q_i(t) \). This is always possible when the phase variable is strictly monotonic with respect to time. Walking controllers often take \( \theta \) to be the angle of the line between the stance leg end and the hip \( [4, 42] \). We draw inspiration on this literature to design time-invariant controllers for rotational movements.

4.3 Methodology

The chapter uses the following methodology:

- A cost function is specified in terms of a sum of quadratic objectives and high-level features of the character’s state. Each objective measures the difference between a desired and actual feature velocity or acceleration at the current timestep.
- A quadratic program is solved at each timestep to obtain the torques that minimize the cost function, subject to constraints for physical accuracy, torque limits, etc.
- The desired features are parameterized by a phase variable instead of time to increase controller robustness.
- The desired features are designed to synthesize trajectories for a range of tasks and user inputs.

4.4 Preliminaries

We now provide an overview of feature-based control \([2, 29, 73]\). Let \( q, \dot{q}, \ddot{q} \) denote generalized joint positions, velocities and accelerations. A feature is a map of the character’s state:

\[
d = \mathcal{F}(q, \dot{q}).
\]  

(4.2)

Examples of features are the center-of-mass (COM), the angular momentum and an end-effector.

The objectives used in this chapter are \( E_{(l/r)\text{footContact}} \) and \( E_{(l/r)\text{handContact}} \) to keep the (left/right) foot and the (left/right) hand planted to the ground, \( E_{(l/r)\text{foot}} \) and \( E_{(l/r)\text{hand}} \) to control the trajectory of the (left/right) foot and (left/right) hand, \( E_{\text{COM}} \) to control the trajectory of the COM, \( E_{\text{AM}} \) to control the angular momentum, and \( E_{\text{pose}} \) to servo full-body joints to a rest pose.

Let \( \tilde{d}_i \) refer to the desired feature associated with objective \( E_i \). The objectives measure the difference between the desired and actual feature accelerations:

\[
E_i = \| \tilde{\ddot{d}}_i - \ddot{d}_i \|^2.
\]  

(4.3)

The \( E_{\text{AM}} \) objective, however, measures the difference between the desired and actual angular momentum velocities \([29]\). In this work, the desired feature accelerations are computed by linear control:

\[
\tilde{\ddot{d}}_i = k_p(\tilde{d}_i - d_i) - k_v \dot{d}_i,
\]  

(4.4)

where \( k_p \) and \( k_v \) are gains.

As in previous work \([2]\), we use a two-layer control architecture hierarchy. At the higher level, a control strategy determines the desired features. At the lower-level, a quadratic program is solved to map features to joint torques.
Let $v = [u^T, \ddot{q}^T, \zeta^T]^T$, where $u$ denotes joint torques and $\zeta$ denotes the basis weights of the linearization of the Coulomb friction cone $[2]$. The following quadratic program is solved at each simulation timestep:

$$
\begin{align*}
    v & = \arg\min_{\nu} \sum_i w_i E_i, \\
    C(v) & = 0, \quad D(v) \geq 0,
\end{align*}
$$

where the equality constraints $C$ are the equations of motion, and the inequality constraints $D$ account for contact forces, joint limits and torque limits $[29]$. We reviewed feature-based controllers in detail in Sec. 2.3.4. Given $u$, a simulator updates the character’s state by integration. In the next section, we discuss how we parameterize the feature-based controllers in this work.

### 4.5 Phase Parameterization

One can create feature-based controllers by specifying the desired features as functions of time $\tilde{d}(t)$. This is the approach advocated in most of the literature $[2, 28, 29, 86, 138]$. One exception is Brown et al. $[17]$, where it is shown how to design robust rolling controllers that track a reference motion independently of time. In this chapter, we extend the work of Brown et al. $[17]$ by designing time-invariant control schemes for a variety of rotational movements, without using reference motions. Previous time-invariant controllers were mostly restricted to locomotion and to joint space $[4, 42]$. To achieve our goal, we use the method of virtual constraints (4.1), with a small modification to make it applicable in feature space. Specifically, we specify the desired features as functions of a phase variable $\theta(q)$ and the character’s state:

$$
\tilde{d}_i = h_i(\theta, q, \dot{q}).
$$

It is redundant to include $\theta$ in $h_i(\cdot)$ since it is a function of $q$, but we do so to emphasize the importance of the phase parameterization in our control design. When presenting the details of the controllers, we will see that it is easy to define $h_i(\cdot)$ by hand, although optimization might be necessary for more complex motions.

In the remainder of this section, we discuss our choice of a phase variable and the benefits of the parameterization. The phase variable needs to be strictly monotonic in time in the undisturbed trajectory so that it can be used to uniquely position the character. An natural choice for $\theta$ in rotational movements is an angle in the plane perpendicular to the axis of rotation of the revolution. For rotations in the sagittal plane such as flips and cartwheels, we chose $\theta$ to be the angle of the line between the neck and the pelvis (see Fig. 4.1). There are many other possible choices. In this chapter, we use the convention that $\theta$ increases as the revolution progresses. A trajectory can be partitioned in domains when it does not have a variable that is strictly monotonic for its entire duration. For example, movements that involve multiple revolutions such as dives are partitioned by revolution, so that $\theta$ is strictly monotonic on each domain.

Parameterizing controllers by a phase variable offers two benefits. The first is to make the controllers more general. For example in a flip, the character roughly completes a full revolution, no matter how far, high or fast the character jumps. Generality is achieved by using this invariant property of the movement.
in the control structure. The second is to make the controllers more robust because of feedback, which we illustrate with the following example. In Fig. 4.2 we show the character performing a cartwheel under no disturbances. Now, suppose that strong wind forces push the character in the direction of travel, so that the character is performing the cartwheel faster than usual. If the desired features \( \vec{d} \) are parameterized by time, then the character is likely to go out of phase since it is ahead of schedule, but \( \vec{d} \) remains on schedule. This does not occur when the desired features \( \vec{d} \) are parameterized by \( \theta \). The reason is that when the character gets disturbed, \( \theta \) is correspondingly disturbed. The controller uses \( \theta \) to position the character in the revolution and use the corresponding features.

## 4.6 Rotational Movements

In this section, we present our approach to the synthesis of physics-based rotational movements. We emphasize that this work is about designing general controllers, not just trajectories. We will see that the design of complex rotational motions can be decomposed in stages that, in humans, correspond to relatively stable configurations that can be practised and rehearsed. For all the controllers in this chapter, the \( E_{\text{pose}} \) objective is active during the entire movement. The initial value of \( \vec{d}_{\text{pose}} \) is set to be the desired initial pose of the movement. This value is maintained during the entire movement, unless we mention changes in specific joints, similarly to Wu et al. [136]. Given a vector \( \vec{v} \), we denote its projection on the ground plane by \( \vec{v}^{zz} \) and on the vertical axis by \( \vec{v}^y \).

The simulator details are provided in Sec. 3.11. The system is single threaded and runs at 50-100% real-time on a 2.7 GHz core CPU. Simulation and control both run at 1 kHz.

### 4.6.1 Cartwheel

In this section, we describe the left side cartwheel controller. For the right side cartwheel, replace the left hand/foot with the right hand/foot, and vice versa. The cartwheel controller consists of five stages,
Chapter 4. Feedback Control for Rotational Movements in Feature Space

Figure 4.2: Cartwheel. The first frame of the five stages of the cartwheel controller and the last frame are shown. The red line connects the pelvis and the neck. The angle of this line is used to switch between stages.

with θ used to determine the current stage of the movement (see Fig. 4.2). Stage i is triggered when $\theta \in (\tilde{l}_i, \tilde{u}_i)$, where θ increases as the cartwheel progresses. The controller has the following structure:

$$\bar{d}_i = \lambda_i(q, \dot{q}), \quad \theta \in (\tilde{l}_i, \tilde{u}_i), \quad (4.8)$$

where $\lambda_i$ is the control law associated with stage $i$. The controller takes as input the desired straddle width, the desired distance between the hands when in contact ($s$), and a speed factor ($v$).

Let $\vartheta$ be the desired direction of the cartwheel. During the entire movement, an angular momentum $E_{AM}$ objective with $\bar{d}_{AM}$ in the direction perpendicular to $\vartheta$ is used. We control the speed of the cartwheel with the magnitude of $\bar{d}_{AM}$:

$$|\bar{d}_{AM}| = v|\bar{d}_{AM}|, \quad (4.9)$$

where $|\bar{d}_{AM}|$ is the magnitude of the angular momentum target at a nominal speed. Setting $v = 0.5$ and $v = 1.5$ generate cartwheels that are approximately 50% slower and 50% faster than the nominal speed, respectively.

On stage one, the rest pose of the lumbar joint is chosen so that the character tilts towards its left side. An $E_{COM}$ objective is used for the character to bend down and move in the $\vartheta$ direction:

$$\bar{d}_{COM}^{xz} = c^{xz} + \eta \theta^{xz}, \quad (4.10)$$
$$\bar{d}_{COM}^{y} = c^{y} + l, \quad (4.11)$$

where $c$ is the position of the COM, and $\eta$ and $l$ are scalars that satisfy $\eta > 0$ and $l < 0$. This objective remains active until both hands touch the ground. The feet are maintained on the ground with the $E_{rfootContact}$ and $E_{lfootContact}$ objectives. On stages two and three, the $E_{lhand}$ objective is used to place the character’s left hand:

$$\bar{d}_{lhand}^{xz} = c^{xz} + \eta \theta^{xz}, \quad (4.12)$$
$$\bar{d}_{lhand}^{y} = d_{lhand}^{y} \left(1 - \frac{\theta - \theta_i}{\theta_f - \theta_i}\right), \quad (4.13)$$

where $\theta_i$ is the value of $\theta$ when the character should begin to lower its hand, and $\theta_f$ is the value of $\theta$ when the hand should be on the ground. The same approach is used to place the right hand on stage three, except that the target vertical position goes to zero as the distance between the hands gets closer.
to its desired value:

$$\vec{d}_{rhand}^y (1 - \frac{\| \vec{d}^{xz}_{rhand} - \vec{d}^{xz}_{lhand} \|}{s})$$

(4.14)

The hand contact objectives $E_{handContact}$ are activated when the hands are sufficiently close to the ground. The rest poses of the hips are used to create the desired straddle width. As the character gets near the end of its revolution, on stages four and five, respectively, the $E_{lhandContact}$ and the $E_{rhandContact}$ objectives are deactivated. The $E_{lfoot}$ and $E_{rfoot}$ objectives are used so that the character can land its feet in a good position. We set $\vec{d}_{lfoot}$ and $\vec{d}_{rfoot}$ relative to the COM and $\phi$, which is an estimate of the character’s global orientation. Specifically, we have:

$$\vec{d}_{lfoot} = R_{\phi} (c^{xz} + p^{loff})$$

(4.15)

where $R_{\phi}$ is the rotation matrix associated with $\phi$, and $p^{loff}$ is a fixed horizontal offset term for the left foot. An analogous feedback law applies for $d_{rfoot}$. We found that the orientation of the lumbar joint is a good choice for $\phi$. When the left and right foot are close enough to the ground, we activate the $E_{lfootContact}$ and the $E_{rfootContact}$ terms. The movement ends with a balance controller, which is similar to what was presented in Abe et al. [2] and Kudoh et al. [65].

As illustrated in the accompanying video [6], the controller is robust to pushes that speed up or slow down the movement since it is time-invariant. The controller also exhibits robustness to pushes in random directions because the positions of the end-effectors are not specified in world coordinates, but as functions of the COM and the character’s orientation (see (4.12) and (4.15)). We also give different examples of cartwheels by varying the inputs, which clearly demonstrates that we are synthesizing control solutions that are more general than a single trajectory.

### 4.6.2 Flips

In this section, we provide the details of the backflip controller. We can generate frontflips using the same approach (see the accompanying video [6]). The main difference between the backflip and frontflip controller is the sign of the angular momentum target $\vec{y}_{AM}$ on the axis of rotation. To simplify the discussion, whenever we refer to the inertia, the angular momentum or the angular velocity, we refer to their components on the axis of rotation. The controller takes as input the desired apex of the flip ($\alpha$) and the desired inertia in the airborne stage ($I_d$), which is used to control the style of the flip. The controller is divided in two main stages: pre-airborne and airborne.

**Pre-airborne Stage**

At first, an $E_{COM}$ objective is used to place the character in a crouch position that is slowly tilting on its back. The feet are maintained to the ground with the $E_{rfootContact}$ and $E_{lfootContact}$ objectives. These contact objectives are deactivated when the character’s torso is oriented towards $\alpha$. The character is then directed to move towards the desired apex with $\vec{d}_{COM} = \alpha$.

At this point, an $E_{AM}$ objective is used to generate the necessary angular momentum for the character to flip. The character cannot land properly with insufficient or excessive angular momentum. Our approach to compute the angular momentum target is as follows. We begin by estimating the duration of the airborne stage of the flip: $t_{air} = 2 \dot{\epsilon}/g$, where $\dot{\epsilon}$ is the estimated velocity of the COM in the vertical
axis at the start of the airborne stage and $g$ is the acceleration due to gravity. We set $\dot{\epsilon}$ so that the height of the desired apex is achieved, namely, $\dot{\epsilon} = \sqrt{2g(\alpha - \epsilon)}$, where $\epsilon$ depends on the character’s stature. Assume that the flip is performed in the clockwise direction. It follows that the character’s average angular velocity in the airborne stage is $\bar{w} = \psi(\theta_{init}, \theta_{land})/t_{air}$, where $\theta_{init}$ is the estimated value of $\theta$ at the start of the airborne stage, $\theta_{land}$ is the desired value of $\theta$ at landing, and $\psi(\theta_{init}, \theta_{land})$ measures the clockwise angular distance between $\theta_{init}$ and $\theta_{land}$. We can now specify the target for $E_{AM}$: $\bar{y}_{AM} = I_d \bar{w}$. As can be seen in the accompanying video [6], changing $I_d$ creates more or less tucked flips.

**Airborne Stage**

The goal of the airborne controller is for the character to land with a desired orientation ($\theta_{land}$) and inertia ($I_{land}$). The controller relies on the fact that $\theta$ is strictly monotonic in the airborne stage to position the character in the revolution.

The desired average angular velocity is given by $\bar{w}_{avg} = \psi(\theta, \theta_{land})/t_r$, where $t_r$ is the estimated time remaining in the airborne stage. $t_r$ can be easily calculated assuming that the character is a projectile ($c$, $\dot{c}$). Note that the controller remains time-invariant because $t_r$ is calculated at each timestep from the character’s state. The current angular velocity and the desired angular velocity at landing are given, respectively, by $w = |M|/I$ and $w_{land} = |M|/I_{land}$, where $M$ denotes the angular momentum and $|\cdot|$ the absolute value. Let $w_m$ denote angular velocity at time $t_r/2$ in the future. We approximate the character’s average angular velocity with the composite trapezoid rule [20]:

$$w_{avg} = \frac{w + 2w_m + w_{land}}{4}. \ (4.16)$$

The value of $w_m$ is determined given that we want $w_{avg} = \bar{w}_{avg}$ to ensure a proper landing.

If $w < w_m$, the character is rotating too slowly, so we decrease the character’s inertia to increase the angular velocity. If $w > w_m$, the character is rotating too quickly, so we increase the character’s inertia to decrease the angular velocity. The inertia is modified with the rest pose of the joints in the set $L = \{\text{elbow, knee, lumbar}\}$. We use the following control law:

$$\bar{r}_i = r_i + k_i(I - I_m), \ (4.17)$$

where $r_i$ and $\bar{r}_i$ denote the current and desired pose in the sagittal plane of joint $i \in L$, $k_i$ is a gain, and $I_m = |M|/w_m$. We constrain $\bar{r}_i$ to be within joint limits. When the character has completed most of its revolution, its feet are placed to the ground with (4.15). The balance controller is then used.

**Discussion**

In the accompanying video [6], we show that the flip controller succeeds for a wide variety of desired apex positions. Our simple balance controller is not suitable for the more aggressive flips, where the character lands with so much momentum that it needs to take a step to maintain balance. We also show that the controller is robust to external pushes of 500 N for a duration of 0.1 s to 0.3 s. For example, if the character is pushed in the direction of the rotation while being airborne, it increases its inertia to slow down the rotation and successfully finish the movement. As the magnitude of the disturbances increases, the character can no longer sufficiently change its inertia to re-phase itself. We also show that
Chapter 4. Feedback Control for Rotational Movements in Feature Space

4.6.3 Diving

The goal of the diving controller is to have the character enter the water with a straight posture and with its arms raised upwards. The controller takes as input the desired number of complete revolutions that the diver must accomplish before entry.

The diving controller is almost identical to the flip controller (Sec. 4.6.2). Here, we highlight the few differences. One, the values of $\theta_{\text{land}}$ and $I_{\text{land}}$ are different because of the different desired landing positions. Second, we must take into account the number of complete revolutions to be performed when calculating the desired average angular velocity $\bar{\omega}_{\text{avg}}$. Lastly, the shoulders are now included in the set of joints $\mathcal{L}$, so that the character can have its arms raised upwards at entry.

In the accompanying video [6], we show that the diving controller succeeds for flips at different heights, angular momentum values, and desired number of complete revolutions. We show that the identical airborne controller can generate forward, backward, straight and armstand dives. Dives with straight knees are generated by removing the knee joint from $\mathcal{L}$. Twisting dives are generated with an $E_{\text{pose}}$ objective to throw one arm up and one arm down in the coronal plane [36] [134]. The character spreads its arms to stop the twisting, and the diving controller as described in the above paragraph is used to finish the motion. The twisting dive controller succeeds for different heights, but there is much room for improvement. For instance, we did not investigate how the character should begin and stop twisting based on its angular momentum and inertia, which would make the controller more general. The multiple axes of rotation make the problem harder to analyze.
Figure 4.5: Diving. The diving and flip controllers are almost identical, which demonstrates the generality of the control laws. We show that the identical control strategy succeeds for a wide variety of heights, flips, and styles.

![Diving Animation]

Figure 4.6: Back handspring. The pre-airborne stage of the back handspring controller precisely regulates the angular momentum of the motion. In the airborne stage, the character raises its arms as the revolution progresses in order to ensure proper landing. In the post-airborne stage, the controller uses $E_{AM}$ and $E_{foot}$ to bring the character back on its feet.

4.6.4 Back Handspring

The back handspring is an acrobatic movement where a character performs a backwards jump, lands on its hands, and then gets back on its feet. The controller takes as input the desired apex of the motion. The pre-airborne stages of the back handspring and backflip controllers (Sec. 4.6.2) differ only in the values of $\theta_{\text{land}}$ and $I_d$. Unlike backflips and dives, people do not significantly change their inertia in the airborne stage of back handsprings. Hence, only the rest pose of the shoulders vary when the character is airborne in our controller. Specifically, the character raises its arms as a function of $\theta$ in order to land on its hands. The character enters a post-airborne stage when its hands are close enough to the ground and the hand contact objectives are activated. An $E_{AM}$ objective is then used for the character to revolve around its hands. The character is brought back on its feet with the $E_{foot}$ objectives presented in (4.15), before ending with the balance controller.

4.6.5 Pirouette

The pirouette is a famous ballet movement in which the body whirls rapidly about one leg. A foot target objective is used to maintain the toe of the supporting leg on the ground throughout the rotation. We bring the COM above the supporting toe with an $E_{COM}$ objective. When the COM gets close enough to its desired position, an $E_{AM}$ objective is used to generate angular momentum in the vertical axis. The number of revolutions that the character can achieve depends on the momentum generated. The rest pose is modified in order to bring the arms in a curved position and to bent the raised leg. When the COM is too far away from its desired position, the character no longer has the necessary momentum.
Figure 4.7: Pirouette. The pirouette controller uses angular momentum to make the body whirl about the supporting leg.

Figure 4.8: Aerial. The aerial controller begins like the flip controller. When the character is near the end of its revolution, the target vertical position of the feet decrease as the revolution progresses to ensure a proper landing.

4.6.6 Front Aerial

The front aerial controller can be divided in the pre-airborne, airborne and post-airborne stages (see Fig. 4.8). The pre-airborne stage differs from the corresponding stage of the flip controller (Sec. 4.6.2) only in the values of the $\theta_{\text{land}}$ and $I_d$ parameters. In the airborne stage, the $E_{\text{pose}}$ objective is used to split the legs. As the character gets near the end of its revolution, its feet are placed to the ground with (4.15), except that the desired heights decrease as the revolution progresses, similarly to (4.13).

4.6.7 Motion Sequences

In the accompanying video [6], we provide examples of motion sequences that can be generated with our individual controllers. We sequence controllers together such that the state of the character at the end of each controller should be close enough to the desired initial state of the subsequent controller. In one example, the character successfully performs a series of backflips until its momentum becomes excessively high. In other examples, the character performs different types of movements in succession, for instance a backflip followed by a back handspring.
4.7 Discussion

We presented feature-based control algorithms for a wide variety of rotational movements, including aerals, cartwheels, dives, and flips, that do not require any input motion or offline optimization. Most of these movements have not previously been generated by physics-based methods without prior data. Our controllers are general and robust, important properties that are often lacking in previous results.

We use the following strategy when designing a controller. Our goal in the first iteration is to synthesize a single trajectory of the movement, without concern on making the controller time-invariant. On the second iteration, we substitute time with a phase variable and verify that it increases the robustness of the controller. On the last iteration, we attempt to generalize the controller so that it can synthesize a manifold of trajectories, e.g., cartwheels with different speeds and styles, aerals with different apexes, etc. In our experience, the first iteration can require a significant amount of trial-and-error (e.g., less than 15 minutes for the flip, but more than 5 hours for the back handspring), the second iteration is straightforward, and the last iteration requires insight in order to translate invariant properties of the movement into control strategies.

Our results lack some aspects of natural human motion. All models of motion control, including both synthetic controller models and those based on motion capture, will yield visual artifacts in some situations. As computational models of the human body become more comprehensive, however, the scope of realistic motions we will be able to simulate will grow. This will come at the cost of increased model dimensionality, so that reduced dimension models will become all the more important.

One important limitation of this work is that most of our rotations are planar. We believe that our phase parameterization can be extended to three-dimensional rotations by simply choosing \( \theta \) to be an angle in one of the planes of rotation. However, designing controllers by hand becomes more difficult as the motion complexity increases. For instance, we were not able to synthesize some movements that are at the periphery of performance, such as certain ballet moves that require an enormous amount of precision. For these cases, developing alternative methods is essential. One approach could be to optimize the virtual constraints (4.7), but how to parameterize them is an open problem [42]. This could remove some of the artifacts in our synthesized motions, such as the character slightly tilting on the side in a flip or the cartwheel not being perfectly straight. Another approach could be to design a user interface to control the features, e.g., the trajectory of the hand or the COM. This could make control synthesis even more intuitive, closely resembling the way a spotter would work with an athlete.

The key element behind the robustness of the controllers is their time-invariance. Most of our movements were parameterized by a phase variable \( \theta \), which we chose to be the angle of the line between the neck and the pelvis. An open research question is how to automatically select \( \theta \) in order to increase the controller’s robustness.

We now compare feature-based control for motor skill acquisition with the trajectory optimization method introduced in Chapter 3. Feature-based control generally requires more work for the designer than trajectory optimization, which only requires specifying the cost function. However, there are cases where determining the cost function to produce a behavior is difficult. For instance, we were not able to find a cost function for cartwheels after several days of experiments, while we designed a cartwheel in feature space in just a few hours. Even for simple behaviors such as standing balance, crouch and reach, trajectory optimization sometimes finds completely unnatural solutions due to local minima, modeling inaccuracies, etc. These behaviors can be easily specified in feature space.

We were not able to synthesize contact-rich movements like headspins and handspins with feature-
based controllers because the contacts have to be pre-specified. These movements were synthesized with trajectory optimization (Sec. 3.8.1 and Sec. 3.8.2). We have synthesized backflips with both methods. It can be observed in the accompanying videos [6] that trajectory optimization discovers natural features of human motion that are absent in our feature-based controller, such as the arms swinging to produce extra angular momentum before take-off. Overall, we find that feature-based controllers can achieve multiple tasks, be robust to disturbances and body shape. However, they rely on heuristics and are restricted to specific movements. Trajectory optimization makes the opposite trade-offs.

This work is a step towards closing the gap between the skill set of animated characters and the remarkable prowess of dancers and gymnasts. Our controllers could be used as building blocks for the synthesis of more complex rotational movements. Given a set of action-specific controllers, how can we combine arbitrary controllers together, i.e., not carefully chosen combinations as in Sec. 4.6.7? In other words, given controllers for flips, front aerials, and cartwheels, how can a character perform a flip, followed by a cartwheel, before finishing with a front aerial? One would need a method to augment and estimate the domain of attractions of our controllers. This is the problem we address in the next chapter of this thesis.
Chapter 5

Domain of Attraction Expansion for Character Control

5.1 Introduction

It is very difficult to find control solutions for the locomotion of high degree-of-freedom humanoid characters. Such solutions form small, elusive manifolds within a highly nonlinear motion domain. What’s worse is that they often lack robustness (e.g., fall over with the smallest push), and are difficult to sequence into composite motions. As such, they may demonstrate the existence of control solutions but not feasibility or practicality. Given a character with skills, e.g., balancing, jumping, rising, etc., a certain set of initial states can achieve a set of desirable states. We refer to the set of initial states as the domain of attraction (DOA) and the set of desirable states as the goal set. Skills that are performed from states outside the DOA do not achieve a desirable outcome. The chapter deals with the following question: how can skills be performed over a much larger set of situations? In other words, how can we expand the DOA?

Inspired by Tedrake \[120\], we present an algorithm to do this using random trees. One contribution of this chapter is to show how DOA expansion can be performed for a 25 degrees-of-freedom (underactuated) character as opposed to a 1 degree-of-freedom (fully actuated) pendulum. Secondly, we identify a major source of inefficiency in performing DOA expansion with typical sampling-based motion planning algorithms such as Rapidly Exploring Random Trees (RRTs) \[69\]. In this work, we advocate for a new motion planning algorithm that constructs dense random trees. This is achieved by biasing our trees to grow in breadth before depth, while RRTs are biased conversely. The key intuition is that dense trees make it easier to steer a state inside the DOA.

The goal of DOA expansion is to cover as large a portion as possible of a given domain of interest, i.e., to enlarge as much as is required the set of states that can be brought to the goal set. As in Tedrake’s work, we trade the aim of finding optimal policies, which would be intractable, for the aim of finding policies that are good enough, i.e., locally optimal. We perform DOA expansion on getting up, crouch-to-stand, jumping and standing-twist controllers. Our controllers are obtained by optimizing a time-indexed spline that provides proportional-derivative (PD) target angles to achieve a final target

\[1\] A publication related to this chapter appears as Al Borno et al. \[9\].
state. We could, however, have chosen to perform DOA expansion on any initial controller, e.g., \cite{2, 24}. The trees are constructed offline, but can be queried quickly enough for near real-time control. The queries use the character’s state to locate the appropriate time-indexed target PD-reference trajectory for control.

5.2 Related Work

As discussed in Chapter 2, a physics-based motion can be synthesized by optimizing a trajectory, e.g., \cite{92, 132}, or by designing control laws for specific movements, e.g., \cite{56, 132}. The methods we developed in Chapters 3 and 4 are examples of the former and the latter, respectively. In both these categories, data-driven methods have been explored to improve the realism of the synthesized motions, e.g., \cite{71, 82}. Open-loop controllers have small DOAs; closed-loop controllers such as SIMBICON \cite{142} can have large DOAs. Both are only suitable for initial states inside the DOA. We are interested in the question of how to control a simulated character outside the DOA.

There are two general approaches for character control. One is model predictive control (MPC), which re-plans a new trajectory at each timestep from the current state. The other is to pre-compute a control policy. The latter requires examining a domain in state space and preparing actions for every state that could be encountered. The former does not, making it attractive because the domain can be very large. This comes at the cost of more easily falling into local minima.

Recent work by Tassa et al. \cite{117} and Hämäläinen et al. \cite{46} are examples of MPC methods. The iLQG method of Tassa et al. (see Sec. 2.4.2) can have the character get up from arbitrary lying positions on the ground. The character, however, gets up unrealistically with one bounce, implying that very large torques are at play. In our own experiments, the character fails to get up with the iLQG method when more conservative torques limits are used, e.g., $\pm 300$ Nm. The multimodal sampling method of Hämäläinen et al. can have the character balance and get up from a wide variety of scenarios, but it fails on others. It is for the failure cases that some offline pre-computation is necessary. We discuss our results in relation to these methods in Sec. 5.7.4.

Our work in this chapter aligns with the pre-computation of a control policy category. Wang et al. \cite{128} optimize the parameters of single controller given a distribution of initial states, but the method does not generalize well when the initial states are too far away. Sok et al. \cite{111} construct a control policy from optimized trajectories that track motion capture data, but does not tackle the case of the optimization falling in local minima. Our work shows how trajectories can be connected together to avoid local minima. Atkeson et al. \cite{120} demonstrate for low-dimensional problems how to approximate the globally optimal policy with local estimates of the value function. To make the problem more tractable, Tedrake \cite{120} forgoes the goal of finding globally optimal policies for the goal of finding good-enough policies instead. The main idea is to use RRTs and feedback controllers to cover a desired domain in state space. Trajectory optimization (TOPT) is used to connect two states together. Mordatch et al. \cite{92} and Posa et al. \cite{104} are recent TOPT methods for offline motion synthesis. Our work is inspired by Tedrake’s approach. The LQR-Trees algorithm uses direct collocation to synthesize open-loop trajectories, the linear quadratic regulator (LQR) for feedback control (see Sec. 2.5), and Lyapunov functions for estimating the DOA. Our work uses the model-free, shooting method in Chapter 3 to synthesize trajectories stabilized with PD controllers indexed by time, and forward dynamics to evaluate the DOA. The method provides a fundamentally different approach to the single-optimized-trajectories
that result from MPC; it develops a tree of reference trajectories that is computed offline and then exploits it online. While previous character animation chapters have produced workable DOAs in some scenarios, none of them answer the question of how to significantly grow the DOA. Our work does not use motion capture data, but we speculate it would be possible in order to guide our trajectory optimization towards realistic solutions, e.g., [61].

We perform DOA expansion on crouch-to-stand, jumping, standing-twist, and various getting up controllers. There is previous work on getting up motions by Lin et al. [74], where motion capture data of the character getting up from one initial state is extended to new lying postures by using physics simulation. Our synthesized motions are entirely physically based, requiring no motion capture stitching. An important and under-studied problem in character animation and robotics is how to connect controllers together, i.e., how can a character perform skills sequentially, like getting up, followed by a standing-twist and a jump? In related work, Liu et al. [81] uses policy search to learn linear feedback matrices for very specific motion transitions. Firmin et al. [34] provides a control language for skill authoring, where the resulting DOA depends on the skill of the designer. As will be shown, none of these approaches are as generic as the one we develop here. We show how DOA expansion can be used to connect controllers together.

5.3 Methodology

The chapter uses the following methodology:

- A random tree is constructed offline to bring states inside the DOA of a controller.
- States are randomly sampled in progressively larger subsets for the tree to grow densely.
- The method in Chapter 3 is used to optimize a time-indexed spline that provides the target angles for a PD controller. The cost function evaluates how closely a state outside the DOA can be steered to a neighboring node in the tree.
- To determine if an initial state is inside the DOA, we perform forward simulations with the PD controllers of the closest branches in the tree. If at least one end state is close enough to a desired end state (the root), we consider that the initial state is inside the DOA.

5.4 Domain of Attraction Expansion

Our goal is to perform DOA expansion on initial controllers to make them suitable for a larger set of initial states. The main idea in the expansion process is to sample a state outside the DOA, connect it with the current DOA using trajectory optimization (TOPT), apply feedback control on the trajectory so that a nearby region is now also inside the DOA, and repeat this process by sampling again. In this section, we present algorithms to construct random trees offline that cover as much as possible of a domain of interest in state space. In Sec. 5.5, we describe the techniques developed to implement the various operations in the algorithms. We also show how the trees can be used in near real-time for character control. In Sec. 5.6 and Sec. 5.7, we analyze the performance of the algorithms on a pendulum and on a simulated character.
Let $\Gamma$ be a tree of tuples $(x, C, p, T)$, where $x$ is a state, $C$ is a feedback controller, $p$ is a pointer to the parent node, and $T$ is the duration of the edge. Let $\chi$ and $\Omega$ denote the current and desired DOAs. We use the term $x_{\text{target}}$ to refer to a state inside the DOA of an initial controller $C_0$. A state is inside the DOA of a controller if, using the controller from that state, the character’s end state is inside a goal set, i.e., within an epsilon distance of a goal state $x_{\text{goal}}$. We specify a single target and goal state for our controllers, but with a slight change in the formulation it is possible to handle multiple states (e.g., by checking if the end state is within an epsilon distance of one out of many goal states). Let $T_0$ denote the duration required to bring $x_{\text{target}}$ to the goal set using $C_0$.

**Algorithm 1** RRTFC algorithm

1: $\Gamma$.add_node($x_{\text{goal}}$, NULL, NULL, 0)
2: $p \leftarrow$ pointer to the root
3: $\Gamma$.add_node($x_{\text{target}}$, $C_0$, $p$, $T_0$)
4: for $k = 0, \ldots, K$ do
5: Randomly sample a state $x_{\text{rand}}$ inside $\Omega$
6: if $x_{\text{rand}} \notin \chi$ then
7: Find the nearest neighbor $x_{\text{near}}$ in $\Gamma$ to $x_{\text{rand}}$
8: Obtain state $x_{\text{new}}$ by extending $x_{\text{rand}}$ towards $x_{\text{near}}$
9: Solve a TOPT to steer $x_{\text{new}}$ to $x_{\text{near}}$
10: $x_1, x_2, \ldots, x_T \leftarrow$ full trajectory from the TOPT
11: if $x_T \in \chi$ then
12: Obtain a feedback controller $C$
13: $p \leftarrow$ pointer to the node that has $x_T$ in its DOA
14: $\Gamma$.add_node($x_1$, $C$, $p$, $T$)
15: $\Gamma$.add_node($x_2$, $C$, $p$, $T - 1$)
16: \ldots
17: $\Gamma$.add_node($x_{T-1}$, $C$, $p$, 1)
18: end if
19: end if
20: end for
21: return $\Gamma$

### 5.4.1 RRTFC Algorithm

We begin by presenting the main algorithm in Tedrake’s work, which we call the RRT Feedback Coverage (RRTFC) algorithm (see Algorithm 1). The main property of RRTs is to bias the trees towards the unexplored regions in the domain. This is achieved by sampling a random state inside $\Omega$ before choosing a state $x_{\text{new}}$ inside the same domain, but closer to the tree (see Fig. 5.1). A naive random tree algorithm would have directly sampled a random state near the tree. As shown in Lavalle et al. [70], the RRT is remarkably more efficient at finding a path to a target than the naive random tree.

A TOPT problem is then solved to steer $x_{\text{new}}$ towards its closest state $x_{\text{near}}$ in the tree. For high-dimensional, highly-nonlinear problems, the TOPT often fails to steer $x_{\text{new}}$ close enough to $x_{\text{near}}$ to be inside the DOA. In our experience, this is a major source of inefficiency in performing DOA expansion because solving TOPT problems is time-consuming. Note that the connection is successful if the state at the end of the trajectory $x_T$ is inside the DOA of any controller in the tree, not necessarily the DOA of the controller associated with $x_{\text{near}}$ (see Fig. 5.2). We do not test if the states at the previous timesteps are inside the DOA because it would be too expensive. If the connection is successful, the states at all
timesteps are added to the tree, even if they are not inside \( \Omega \). This increases the likelihood of \( x_f \) landing inside the DOA.

Figure 5.1: DOA expansion with an RRT. An RRT starts by sampling a random state in the domain \( x^{\text{rand}} \) and finds its closest node \( x^{\text{near}} \) in the tree. It then attempts to connect to \( x^{\text{near}} \) (green arrow) starting from a state which is close and in the direction of \( x^{\text{rand}} \). The intuition behind DFC is that attempting to connect to the dense parts of the tree is more likely to be successful (orange arrow) even if it is slightly further away according to the distance metric. The red and purple circles are the target and goal states, and the arrow in between is the initial controller \( C_0 \). The blue background is the domain covered by the tree.

5.4.2 DFC Algorithm

RRTs have the property of growing long branches quickly to efficiently explore the domain and find a path to a given state. This property does not seem important in the context of DOA expansion, where the objective is to find a path to every state in the domain as opposed to a single state. This suggests that we can design a motion planning algorithm specifically to improve the efficiency of the DOA expansion process. The algorithm should aim to construct a dense tree since this increases the likelihood of \( x_f \) being inside the DOA, thereby reducing the number of failed connections (see Fig. 5.1).

We now present our approach to do this, which we call the Dense Feedback Coverage (DFC) algorithm (see Algorithm 2).

The idea is to cover the domain around \( x^{\text{target}} \) with progressively larger subsets \( \Omega_0 \subset \Omega_1 \subset \ldots \subset \Omega \), where \( \Omega_0 \) denotes the DOA of the initial controller (see left Fig. 5.3 and Sec. 5.5.3 for an example of how to define the subsets). For each domain \( \Omega_i \), the nearest neighbor is constrained to be in a subset \( \Omega_h, h < i \). A connection is successful as long as the state at the end of the TOPT lands somewhere in \( \Omega_h \) that is covered by the tree. This is why it is much easier to steer the character inside the DOA of a region occupied with many nodes as opposed to a region occupied with very few nodes. In DFC, each subsequent subset effectively gets a larger target area to perform new connections. This is akin to how a problem is divided and solved in dynamic programming. To solve the problem of bringing a state in a given domain inside the DOA, we use the solutions of subproblems found at decreasing subsets of the domain.

We now describe the heuristics we use to set the parameters in the algorithm. The optimal collection of subsets \( \Omega_k \) will depend on the problem. Generally, more subsets yield more samples that are already
inside the DOA, while fewer subsets yields more samples that fail to connect. We choose the maximum
number of iterations $K_i$ to be roughly proportional to the fraction of the domain that is covered by the
subset. The choice of the nearest neighbor subset (line 9) will also depend on the problem. To some
extent, the smaller the subset, the more likely it is for the final state of the TOPT to land inside the
DOA. However, a subset that is too small, i.e., too far away from $x^{rand}$, will increase the likelihood of
a failed connection. One strategy is to choose the smallest subset such that the distance between $x^{rand}$
and $x_{near}$ is within a specified value. We typically simply choose the nearest neighbor to be inside subset
$\Omega_{i-1}$ when expanding subset $\Omega_i$.

If the DOA of the initial controller is particularly small, it might be inefficient to attempt to steer
all the nearby states inside it. In other words, when the current DOA is very small, it could be even
harder to steer the nearby states inside a subset of the DOA, which is the strategy employed by DFC.
In this case, we first perform a DOA expansion on a subset of the desired domain using other methods,
e.g., by using RRTFC on $\Omega_1$. We then continue the DOA expansion process on the entire domain with
DFC. This strategy is used for the pendulum. For the simulated character, we employ DFC on the entire
domain directly.

A sample that fails to connect to the tree at a given subset can successfully connect when resampled
at the same or larger subset. The difference would result from having a new nearest neighbor or from
the end state after a TOPT to be inside a region now covered by the tree. This is why we sample from
$\Omega_i$ and not from $\Omega_i \setminus \Omega_{i-1}$, i.e, we do not exclude $\Omega_{i-1}$. However, it is possible to sample as a function
of the domain covered by the tree. For example, if the tree covers 95% of $\Omega_{i-1}$, we can have 5% of the
samples in $\Omega_{i-1}$ and 95% in $\Omega_i$. Sec. 5.5.8 describes how we can estimate the domain coverage.

\section*{5.5 Techniques}

We now describe our implementations of the operations in the RRTFC and DFC algorithms.

\subsection*{5.5.1 Trajectory Optimization and Feedback Control}

The method introduced in Chapter 3 optimizes a reference trajectory $q_{1:T}$ represented by a parametrized
cubic B-spline. The output motion $q_{1:T}$ is computed by forward dynamics. The torque $u_i$ for a joint is

---

\begin{figure}
\centering
\includegraphics[width=\textwidth]{sparse_dense_tree.png}
\caption{Examples of Trajectory Optimization in a Sparse and Dense Tree. The blue
tubes illustrate the domain covered by edges of the tree. The orange circle is a random state that we
attempt to connect to its nearest state in the tree with trajectory optimization. The dotted curves are the
trajectories followed when attempting the connection. The figures show that the attempt is more likely to
be successful in a dense tree (right figure) than in a sparse tree (left figure).}
\end{figure}
Algorithm 2 DFC algorithm

1: $\Gamma$.add_node($x^{goal}$, NULL, NULL, 0)
2: $p \leftarrow$ pointer to the root
3: $\Gamma$.add_node($x^{target}$, $C_0$, $p$, $T_0$)
4: Divide the domain in sets $\Omega_0 \subset \Omega_1 \subset \Omega_2 \subset \ldots \subset \Omega_N$, where $\Omega_N = \Omega$
5: for $i = 1, \ldots, N$ do
6: for $k = 0, \ldots, K_i$ do
7: Randomly sample a state $x^{rand}$ inside $\Omega_i$
8: if $x^{rand} \notin \chi$ then
9: Find the nearest neighbor $x_{near}$ in $\Gamma \cap \Omega_h$ to $x^{rand}$, where $h < i$
10: Solve a TOPT to steer $x^{rand}$ to $x_{near}$
11: $x_1, x_2, \ldots, x_T \leftarrow$ full trajectory from the TOPT
12: if $x_T \in \chi$ then
13: Obtain a feedback controller $C$
14: $p \leftarrow$ pointer to the node that has $x_T$ in its DOA
15: $\Gamma$.add_node($x_1$, $C$, $p$, $T$)
16: $\Gamma$.add_node($x_2$, $C$, $p$, $T - 1$)
17: \ldots
18: $\Gamma$.add_node($x_{T - 1}$, $C$, $p$, 1)
19: end if
20: end if
21: end for
22: end for
23: return $\Gamma$

determined by PD control:

$$u_t = k_p(\hat{q}_t - q_t) - k_dq_t,$$

(5.1)

where $q_t$ and $\hat{q}_t$ are the current and reference values of the joint angle. All joints have PD gain values of $k_p = 700 \text{ Nm/rad}$ and $k_d = 1 \text{ Nms/rad}$.

We use this TOPT method to steer the character to a desired state $x^d = (q^d, \dot{q}^d)$ in $T$ timesteps, while minimizing a measure of effort. The optimization problem is given by:

$$s^* = \arg \min_s \{w_1 \text{dist}(x_T, x^d) + w_2 \sum_{i=1}^{T} ||u_i||^2\},$$

(5.2)

where $s$ are the values of spline knot positions that are equally-spaced every 0.1 s, the distance metric dist is defined in Sec. 5.5.4, $u$ is the vector of joint torques, and $w_1$ and $w_2$ are weights. The optimization is performed with CMA and is initialized with a kinematic interpolation between the start and desired poses.

Note that this method returns both the trajectory and the closed-loop controller in the RRTFC (lines 9 and 12) and DFC (lines 10 and 13) algorithms. Although our joint-local PD controllers are weak, i.e., have small DOAs, we can build robust controllers by combining many weak ones together. Alternatively, one could optimize these controllers to make them more robust, e.g., as in Liu et al. [81]. There is a trade-off on whether it is more efficient to spend time building a few robust controllers or combining many weak ones.

A simple extension to our trajectory optimization method can be done to optimize for the duration
Chapter 5. Domain of Attraction Expansion for Character Control

Figure 5.3: (Left) DOA coverage with DFC. The DFC algorithm progressively covers the domain around the target state (the red circle). Increasingly larger subsets are illustrated in the figure. The orange circle is a random state sampled on the largest subset. TOPT is used to steer the state to its closest state in a smaller subset. (Right) Determining if a state is inside the DOA. We illustrate how we determine if a state $x^r$ (the orange circle) is inside the DOA. In the figure, we perform simulation rollouts, starting from the controllers associated with the closest states in the tree (the orange arrows illustrate the sequence of controllers used for one closest state). We use PD controllers with targets provided by time-indexed splines. A rollout consists of moving up the nodes of a branch, until the root (the goal state) is reached. If one of the rollouts steers the character within an epsilon distance of the goal state, then $x^r$ is inside the DOA.

of the movement. We add a time variable to the optimization and a cost of time objective to penalize excessively slow movements. Specifically, we add a linear cost $w_3T$ to Eq. (5.2). Optimizing the movement duration every time the algorithm attempts to connect two states would be prohibitively slow. For this reason, whenever the distance between the initial and desired state is less than a chosen threshold, we use a constant movement duration, e.g., 0.2 s.

5.5.2 DOA Modeling

We now describe how we determine if a state $x^r$ is inside the DOA after a TOPT, which corresponds to line 11 in RRTFC and line 12 in DFC. We model the DOA in an implicit fashion by using multiple forward dynamics simulations, rather than having an explicit model as in Tedrake [120] that does not currently scale to high dimensions. We start by finding the $V$ closest states to $x^r$. We then perform $V$ simulation rollouts starting from $x^r$, using the controllers that were optimized in Sec. 5.5.1. The rollouts proceed until the root of the tree is reached (see right Fig. 6.3). We then compare the $V$ final states with the goal state. If at least one of the states is within an epsilon ball of the goal state, then we consider $x^r$ to be inside the DOA. Pseudo-code is provided in Algorithm 3. The reason we consider $V$ closest states instead of the closest state only is that any distance metric will be imperfect. We often find $x^r$ to be inside the DOA of a different node than the “closest” node. We usually choose $V$ to be between 50 and 250. With a larger value, it is more likely to find a path that leads to the goal set, but the DOA expansion process is also more time-consuming.

Minimal Time

Note that Algorithm 3 halts when the first of $V$ rollouts leads to the goal set. One could continue performing the remaining rollouts to not only find a path to the goal set, but a path that minimizes
some objective (time, energy, etc). Algorithm 4 minimizes the time objective. While this slightly slows down the DOA expansion process, it quite noticeably improves the quality of the motions, as shown in the accompanying video [6].

Using the Tree Online

Once the tree is constructed, Algorithm 3 can be used online to control the character. When searching for the nearest nodes in the tree, we only take those at timestep 1 (line 15 in Algorithm 2) into consideration to speed-up the process because we know that they are inside the desired domain Ω. This is also how we determine if a randomly sampled state is inside the DOA in the tree construction process, which corresponds to line 6 in RRTFC and line 8 in DFC. On a single core, Algorithm 3 runs on average in real-time because only a few rollouts are usually required before finding a solution. The rollouts should not be long for this method to achieve interactive rates, e.g., 4 s. We have tried learning a nearest neighbor classifier that maps a state to a controller, i.e., a branch in the tree, to bypass the need to perform the rollouts. This required an excessively large number of samples for a small domain, making it poorly scalable.

Algorithm 3 [IsInside, Parent] = findStartState(x′)

1: IsInside ← false
2: Parent ← NULL
3: Find the V closest nodes to x′
   for i = 1, . . . , V do
5: n ← pointer to the i-th closest node
6: x′f ← x′
7: while n ̸= ROOT do
8: Do a rollout from x′f with the nc controller
9: x′f ← state at the end of the rollout
10: n ← np (pointer to the parent node)
11: end while
12: if dist(x′f, xgoal) < ϵ then
13: IsInside ← true
14: Parent ← pointer to the i-th closest node
15: break
16: end if
17: end for

5.5.3 Sampling States

The DFC algorithm samples states in the progressively larger domains Ω1, . . . , ΩN. For the getting up motions of the simulated character, this is achieved by sampling uniformly in progressively larger intervals around the target state, subject to joint limits. We parameterize each actuated joint in Euler angles and treat each axis independently for multi-dimensional joints. For joint j with pose rj in the target state, the pose of the generated state is sampled uniformly in the interval [max(rj − αi/mj, lj), max(rj + αi/mj, uj)], where αi is a scalar that parametrizes Ωi, mj is the mass of the associated body link, lj and uj are lower and upper joint limits. For instance, α = 5 creates angle ranges of 27° for the hips, 46° for the knees and 90° for the shoulders, subject to joint limits. The Euler angle parameterization can lead to a non-uniform sampling of rotations due to the singular configurations [66]. Our joint limits, however, avoid
Algorithm 4 [IsInside, Parent] = findBestStartState(x')

1: IsInside ← false
2: Parent ← NULL
3: s ← ∞
4: Find the V closest nodes to x'
5: for i = 1, ..., V do
6: t ← 0
7: n ← pointer to the i-th closest node to x'
8: x' ← x'
9: while n ≠ ROOT do
10: Do a rollout from x' with the nC controller
11: x' ← state at the end of the rollout
12: n ← np
13: t ← t + nT
14: end while
15: if dist(x', xgoal) < ε then
16: IsInside ← true
17: if t < s then
18: s ← t
19: Parent ← pointer to the i-th closest node
20: end if
21: end if
22: end for

this issue. The root orientation of the character is parametrized with a quaternion. The orientation of the sampled state is determined by spherical linear interpolation of a random orientation and the target state orientation, where the interpolation parameter is a function of $\alpha_i$. We then drop the character from the air and wait a short amount of time. The state of the character lying on the ground becomes our random sample. For the other controllers (crouch-to-stand, jumping, etc.), the state is sampled as follows. Starting from the target state, we apply an external force in a random direction on the character for a duration of 0.1 s. The random sample is the state immediately after the external force is applied. The magnitude of the force is chosen randomly on the interval $[0, \kappa_i]$, where the scalar $\kappa_i$ parametrizes $\Omega_i$. The values of $\alpha_i$ and $\kappa_i$ increase with $i$.

5.5.4 State Distance Metric

Given a random state $x^{rand}$, the DOA expansion algorithms need to find its nearest state in the tree. The metric used to measure the distance between states $x$ and $x'$ can have an important impact on the efficiency of the algorithms, but it plays an even more critical role in RRTFC because DFC is designed to reduce the importance of the nearest neighbor. For simplicity, we use the weighted joint angles metric:

$$
\text{dist}(x, x') = \sum_j w_j (r_j - r'_j) + \tilde{w}_j (\dot{r}_j - \dot{r}'_j),
$$

(5.3)

where the sum is over the joint angles and the root angle, $w_j$ is set to the mass of the associated body link, and $\tilde{w}_j = 0.1 w_j$. 
5.5.5 Optimality Heuristic

After sampling a random state, the choice of its nearest neighbor has an important effect on the quality of the motion. One optimality heuristic is to choose \( N \) nearest neighbors instead of one, and solve \( N \) TOPTs. Of the successful connections, we choose the one with the minimal duration to the target. This approach slows down the DOA expansion process proportionally to \( N \). In the supplemental video [6], we see that the motion is performed more directly with \( N = 10 \) as opposed to \( N = 1 \).

5.5.6 Connecting Controllers

An important application of DOA expansion is to connect different controllers together. If the state at the end of one controller is close enough to DOA of the second controller, then DOA expansion can be directly applied on the second controller for the connection. The only difference in Algorithm 2 is that the samples are now chosen randomly from the distribution of the states at the end of the first controller. If the first state is too far from the DOA, then we start by synthesizing a transition controller (see Sec. 5.5.1) that steers it as close as possible to the DOA. We then perform DOA expansion on the transition controller, again with the random samples taken from the distribution of states at the end of the first controller.

5.5.7 Hybrid of Offline and Online Optimization

The method described so far requires pre-specifying a controller for every possible state that could be encountered in the desired domain, which is generally very large due to the dimensionality of the character. MPC methods, on the other hand, do not require this pre-computation, but more easily fall into local minima. We now present a hybrid approach that attempts to get the best of both worlds, i.e., combine MPC methods with some offline pre-computation to avoid both the curse of dimensionality and the local minima. The idea is to construct a small tree offline that sketches the general path to arrive at the goal state. Online optimization is then used to track a branch in the tree. In more detail, to determine if a state \( x^r \) is inside the DOA (lines 8 and 12 in Algorithm 2), we first find its nearest state in the tree \( x^{near} \). We then perform an online trajectory optimization to steer \( x^r \) towards \( x^{goal} \), instead of performing rollouts with the controllers of the closest nodes (line 10 in Algorithm 4). For the simulated character, we perform 100 CMA iterations of 30 short rollouts (e.g., 0.7 s) in the method described in Sec. 5.5.1 which runs at about 5% real-time on a machine with eight logical cores. Fewer iterations would be required for longer rollouts or different methods could be used, e.g., [46]. The cost function used is Eq. 5.2, with \( x^d = x^{goal} \). At each timestep \( t \), the control torque in the rollouts is determined by \( u_t = u_{t}^{ng} + u_t^r \), where \( u_{t}^{ng} \) is the control torque determined from the PD controllers associated with the branch from \( x^{near} \) to \( x^{goal} \), and \( u_t^r \) is the newly optimized portion of the control torque. We found that this technique returns a better solution (measured in terms of the cost function) than to simply initialize the new trajectory optimization with its nearest neighbor’s solution. In one experiment, this approach returns a better solution nine times out of ten when \( x^r \) is sampled near \( x^{near} \) using the approach described in Sec. 5.5.3 with \( \kappa = 0.5 \). When \( x^r \) is sampled further away from \( x^{near} \) (e.g., by using \( \kappa = 2.5 \)), both approaches had comparable performances.
5.5.8 DOA Volume

To estimate the proportion of the domain that is covered by the tree, we sample a large number of states inside the domain and determine if they are inside the DOA using Sec. 5.5.2. The proportion of samples inside the domain provides the estimate of the proportion of the domain that is covered.

5.6 Planar Pendulum

We first perform DOA expansion on a simple system to compare the efficiency of RRTFC and DFC. The system is a planar pendulum in the vertical plane with state \( x = [\theta, \dot{\theta}] \), where \( \theta \) is the joint position. The mass of the pendulum is 3.10 kg and the torque limits are \( \pm 200 \text{ Nm} \). The goal is to stabilize the pendulum to its unstable equilibrium \( [\pi, 0] \), i.e., the upright position, from all the initial states in the domain \( \Omega = [0, 0] \times [\pi, 0] \). We consider that the pendulum has reached the equilibrium if \( x \in \Omega_0 \), where \( \Omega_0 = \{ x : ||x - [\pi, 0]||_2 < 0.01 \} \). We solve the TOPT with the method of Sec. 5.5.1, where we limit the number of CMA iterations to 20 and the number of samples to 12. For the DFC algorithm, we divide the domain in four subsets \( \Omega_k \), defined as follows: \( \Omega_k = [\pi (1 - \alpha_k), 0] \times [\pi (1 + \alpha_k), 0] \), where \( \alpha_k = k/4 \), and \( k = 1, \ldots, 4 \). When using the DFC algorithm, we perform a DOA expansion on \( \Omega_1 \) using RRTFC because \( \Omega_0 \) is too small to be used as a target for all the nearby states.

We compare RRTFC (on the entire domain) and DFC for an average of ten runs, where the entire domain is probabilistically covered by both algorithms, i.e., 200 consecutive random samples are inside the DOA. Given that determining if a state is inside the DOA is much less expensive than a TOPT, the main factors that will affect the efficiency of the algorithms are the number of samples that fail to be added to the tree \( n_f \) and the number of samples that are successfully added \( n_s \). RRTFC has \( n_f = 38 \) and \( n_s = 13 \), while DFC has \( n_f = 13 \) and \( n_s = 16 \). In this case, DFC reduces the number of failed connections by about 65%, but requires 23% more nodes to cover the domain. Overall, DFC is more efficient since it reduces the total number of TOPTs by 44%. We obtain similar results when dividing the domain in 8 or 16 subsets. Similarly, we compared the algorithms on a double pendulum with state \( x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2] \), where each link has mass 3.10 kg. For the TOPT, we limit the number of CMA iterations to 50 and the number of samples to 25. Covering the domain \( \Omega = [\pi - 0.2, \pi - 0.2, 0, 0] \times [\pi + 0.2, \pi + 0.2, 0, 0] \) required RRTFC \( n_f = 950 \) and \( n_s = 25 \), and DFC \( n_f = 132 \) and \( n_s = 25 \), which corresponds to an 84% difference in efficiency.

5.7 Simulated Character

We perform DOA expansion on getting up, crouch-to-stand, jumping, and standing-twist controllers for a simulated character. The getting up controllers express different strategies (see Fig. 5.5). Our initial controllers are synthesized by manually specifying between one and three key-frames as targets in a TOPT problem (Sec. 5.5.1) with the last key-frame corresponding to the desired end state, and by specifying the durations between poses. The goal of DOA expansion is to increase the set of initial states from which the controllers can be used. We also regulate the final global orientation of the character, i.e., it must get up, twist, etc., at a given facing direction, which makes the problem harder.
Chapter 5. Domain of Attraction Expansion for Character Control

Figure 5.4: Full Branch Rollouts. In this figure, we attempt to connect the orange node to its nearest node in the tree. The typical TOPT approach is to steer the node towards the nearest node (the dotted orange arrow). In some problems, we were not able to make any connection this way. We modify the TOPT by using the PD controllers on the closest branch in the rollouts (the solid orange arrows) after a short time interval elapses. The objective is to steer the character as close as possible the goal state. This optimization is more likely to succeed than attempting to find a motion directly to the goal state (the green arrow).

5.7.1 Full Branch Rollouts

Performing DOA expansion on the crouch-to-stand, jumping and standing-twist controllers proved to be particularly more challenging than the getting up controllers, probably because momentum effects are more present. The initial controllers have so small DOAs that it is very difficult to steer any state inside them. For this reason, we modify the TOPT method as follows. Let $C$ denote an arbitrary controller that steers the character from state $x_1$ to $x^{goal}$. When performing a TOPT to connect a randomly sampled state to the DOA of $C$, we previously attempted to minimize the distance of the state at the end of the rollout to $x_1$. Now, our TOPTs will minimize the distance to $x^{goal}$, using $C$ after a short time interval of 0.2 s in the rollouts. In other words, the TOPT optimizes the torques used in the preliminary time interval; the following torques are specified by $C$. This significantly slows down the DOA expansion process since we are now performing rollouts for the entire duration of the branch of the tree instead of the edge only (see Fig. 5.4). Once the DOA of the tree gets sufficiently large, it is no longer necessary to perform the full branch rollouts.

5.7.2 Facing Direction Invariance

The task of our getting up controllers is to have the character reach standing balance with a desired facing direction. Removing the facing direction constraint simplifies the problem. One way to build a rising controller with a large DOA is to supply a single prone or supine target pose for the character to achieve, which should not be difficult because of the facing direction invariance, and wait until the motion comes largely to rest. The target pose provides a nicely repeatable starting state from which to then apply some known rising strategy, thus avoiding the need for DOA expansion. In the accompanying video [9], we show that this technique does not work with our controllers. The reason is that the character never perfectly reaches the target pose. Very slight errors can make the controllers fail. Our experiments show that a small tree, e.g., 80 nodes, is sufficient to obtain a robust rising controller that is invariant with
Figure 5.5: Getting Up Controllers. The figure is divided in three pairs of sequences. For each pair, the top sequence illustrates an initial getting up controller and the bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The yellow arrow points to the desired facing direction for the final pose of the character. The red character is the pose of the currently active node.

5.7.3 Implementation Details

In the TOPT step of the DFC algorithm (line 10), we use a maximum of 600 CMA iterations and 150 samples. For our simulations, we use the MuJoCo physics engine [121] with a 0.01 s timestep, a coefficient of friction of 1, and torque limits of ±150 Nm. Our simulated character has a mass of 40.5 Kg, a height of 1.61 m, and 25 degrees-of-freedom.

5.7.4 Results

The getting up, crouch-to-stand and standing-twist controllers were constructed offline, and the jumping controller was constructed with the offline-online approach (Sec. 5.5.7). A standing controller with a similar approach to Hämäläinen et al. [46] is used when the character is near the last standing pose. The
Chapter 5. Domain of Attraction Expansion for Character Control

Figure 5.6: Crouch-to-Stand Controller. The top sequence consists of key-frames of the crouch-to-stand controller. The bottom sequence illustrates how DOA expansion steers the character to the target (the state in the first frame of the top sequence). The blue line on the first frame is an external force applied on the head. The red character is the pose of the currently active node.

Figure 5.7: Exploration regions. The sequence shows typical states (orange character) in progressively larger subsets for the target states (the red character) associated with the “Getting Up 1” and the crouch-to-stand controllers. The subsets are $\alpha = 1$, $\alpha = 5$, and $\alpha = 10$ for the first three figures, and $\kappa = 100 \text{ Nm}$, $\kappa = 400 \text{ Nm}$, and $\kappa = 1000 \text{ Nm}$ for the last three figures.

The number of nodes required to cover a given subset depends on the controller. This can be seen in Fig. 5.9, where we plot the total number of nodes required to cover progressively larger subsets for some of our controllers. Intuitively, we would expect the rate of growth of the tree to increase with larger subsets. The plot suggests that the trees grow exponentially with the subset parameter (the vertical axis is in log scale). In Fig. 5.8, we plot some of the nodes in the trees for the “Getting Up 3” and the crouch-to-stand controllers to help visualize the expansion process. Table 5.1 provides some statistics on the trees constructed for our controllers. Our trees were constructed by increasing $\alpha$ and $\kappa$ by increments of 0.25-1 and 25 Nm, respectively. The values were chosen so that the distance between two states in subsequent subsets would not be too difficult to overcome by the trajectory optimizer. The trees required between one and three days of computation on a desktop computer.

Performing DOA expansion on an initial motion effectively extends it to new initial states. If the goal is to get up from an arbitrary lying position, the method does not attempt to discover how to get up from scratch. Instead, it attempts to find a path to the place where a getting up strategy is given. In the accompanying video, we show that the character can get up with the desired facing direction from a large set of initial states (see Fig. 5.5). The strategies employed show the character moving from prone to supine (and vice versa), and rotating itself on the ground to regulate its global orientation. Similarly, it is shown how the character can successfully perform crouch-to-stand, jumping and standing-twist motions when force impulses on the order of 7.5 Ns, 10 Ns and 12.5 Ns are applied in random directions on the character and the results are contrasted when no DOA expansion is performed.
Figure 5.8: Visualizing the nodes in the Tree. The first three figures shows the DOA Tree with 1, 10 and 40 nodes for the “Getting Up 2” controller. The last three figures shows the DOA Tree with 1, 10 and 40 nodes for the crouch-to-stand controller. These nodes are sampled from $\Omega$. The difference in velocities between the nodes cannot be visualized.

Figure 5.9: The Size of the Tree. The figure plots the size of the tree in log scale required to cover a subset for the “Getting Up 1” and the “Getting Up 2” controllers. The subsets are parametrized by $\alpha$, which is defined in Sec. 5.5.3. In the plot, the value of $\alpha$ is shown in the Subset-axis.

(see Fig. 5.6). The discovered solutions include the character taking a step or leaning on its knee to stay in balance and perform the motion. In contrast, the Sequential Monte Carlo (SMC) method of Hämäläinen et al. [46] fails to have the character get up under certain prone and supine postures. The same observation applies to the method of Tassa et al. [117] when more realistic torque limits are used.

In the supplemental video [6], we show how the iLQG provided in MuJoCo and our implementation of the SMC method fall into local minima when attempting to steer the character inside the DOA of the standing-twist controller. More extensive searches allow offline methods to avoid some of these local minima. In one experiment, we use the SMC method to steer randomly sampled initial states to the DOA of a getting up controller. The method succeeds nine, four and two times out of ten trials when the sampled states are inside the subsets $\alpha = 2, 6, 8$, respectively. After performing DOA expansion, we succeed ten, ten and nine times for the same subsets. The failure case in the DOA method is due to not having performed enough iterations offline or failing the find the node that leads the state inside the DOA online (Sec. 5.5.2). For the SMC method, we implemented Algorithm 1 in Hämäläinen et al. [46] since our problem does not have a dynamically changing environment, and we compute the torques by PD control instead of their approach, which is specific to the ODE physics simulator. DOA expansion can be used to steer the “failure” states inside the DOA of these MPC methods. The accompanying videos [6] contain examples of how DOA expansion can be used to connect controllers together. We
Controller & Size & Height & Depth & Subset \\ Getting Up 1 & 1377 & 14 & 7.3 & $\alpha = 8$ \\ Getting Up 2 & 751 & 12 & 5.4 & $\alpha = 8.5$ \\ Getting Up 3 & 277 & 41 & 19.3 & $\alpha = 10$ \\ Crouch-to-stand & 193 & 4 & 2.1 & $\kappa = 75\text{ N}$ \\ Standing-Twist & 177 & 4 & 2.1 & $\kappa = 125\text{ N}$ \\ Jumping & 489 & 4 & 1.9 & $\kappa = 100\text{ N}$ \\

Table 5.1: Tree Statistics. The table provides information on the trees for our controllers. The “Size” field gives the number of nodes in the tree, the “Height” field gives the height of the tree, the “Depth” field gives the average depth of the nodes, the “Subset” field gives the largest subset parameter reached during the DOA expansion process.

<table>
<thead>
<tr>
<th>Nbr Samples</th>
<th>500</th>
<th>1000</th>
<th>2500</th>
<th>5000</th>
<th>30000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbr TOPTs RRTFC</td>
<td>57</td>
<td>32</td>
<td>29</td>
<td>24</td>
<td>19</td>
</tr>
<tr>
<td>Nbr TOPTs DFC</td>
<td>22</td>
<td>18</td>
<td>19</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>Reduction</td>
<td>61%</td>
<td>44%</td>
<td>35%</td>
<td>21%</td>
<td>32%</td>
</tr>
</tbody>
</table>

Table 5.2: RRTFC vs DFC. The table compares the efficiency of RRTFC and DFC when performing DOA expansion on the domain $\alpha = 0.5$ for one of our getting up controllers. We give the number of TOPTs necessary to cover the domain, averaged over three runs. We use increasingly better trajectory optimizers that are obtained by increasing the total number of samples in the CMA optimization. The “Reduction” field gives the percentage of fewer TOPTs when using DFC over RRTFC.

show how a crouch-to-stand controller can be connected to a jumping controller, using a standing-twist transition controller. We also show how jumping and stand-crouch controllers can be used repeatedly in a sequence.

The quality of the the trajectory optimizer, i.e., how closely can it steer the character towards the target, has an important impact on the efficiency of DOA expansion. Intuitively, we expect that the less accurate the optimizer is, the more beneficial it will be to have a dense tree. We perform the following experiment to test this hypothesis. We construct five CMA optimizers of increasing quality that are obtained by increasing the total number of samples in the rollouts. The optimizers are then used to perform DOA expansion on the domain $\alpha = 0.5$ for one of our getting up controllers. Table 5.2 provides the number of TOPTs ($n_s + n_f$ in Sec. 5.6) required to cover the domain for average of three runs. The duration of the rollouts is set to 0.2 s so that the TOPTs are equally costly. The data indicates that DFC is more computationally efficient than RRTFC, particularly when it is difficult to connect two states together with TOPT. We also compared RRTFC and DFC when covering a larger domain, namely $\alpha = 10$, using the CMA optimizer with 500 samples. The DFC method required 562 TOPTs, as opposed to 4763 for RRTFC, which constitutes an 88% reduction in the computational cost of DOA expansion. For this problem, we found that only 5 out of 40 successful TOPTs actually connect the random sample $\mathbf{x}^{\text{new}}$ to the target node $\mathbf{x}^{\text{near}}$ (see Sec. 5.4.1); the rest connect to other nodes in the tree.
5.8 Discussion

RRTs have the property to first grow towards the unexplored regions of the domain and then to fill the regions in between branches. It is in the first stage that DOA expansion with an RRT is particularly inefficient. Once the exploration stage is over, the RRT progressively becomes more dense and, consequently, more effective. For instance, using the tree constructed for one of the getting up controllers, we note that only one out of ten TOPTs succeed when the tree has only two nodes, while seven out of ten TOs succeed when the tree has 80 nodes. The difference between the DFC and the RRTFC becomes more pronounced with larger domains because they imply a longer exploration stage.

In this work, we use a shooting method for the TOPT and PD control for feedback, but other methods could have been employed. As was pointed out by Tedrake [120], the core DOA expansion algorithms are compatible with a number of methods, some of which will be better suited for certain applications than others. It would be valuable to compare the efficiency of the expansion process when using different methods, for example, LQR control instead of PD control. We investigate this in the next chapter.

Our synthesized motions do not always look natural. Nonetheless, DOA expansion is a computationally tractable way to bring states inside the DOA when they were previously outside. We now discuss current avenues of research to achieve more natural results. The first is to perform DOA expansion on a set of controllers instead of a single controller. Humans do not have a single way to perform a motion, yet it is the assumption made when performing DOA expansion on a single controller. The second is to use a motion capture database to guide the TOPTs to more human-like solutions. The last is on how to determine movement duration in TOPT, i.e., how long should it take for the character to move from one state to another? In this work, we use a constant movement duration whenever the distance between the states is within a threshold. It would be valuable to develop a method that either efficiently optimizes for a cost of time objective term or estimates the movement duration from prior data.

It is possible to modify the DFC algorithm to perform DOA expansion on states that are more likely to occur in practice by changing the sampling distribution in Sec. 5.5.3. It may, however, be useful to sample states that are unlikely to occur to then bring other states inside the DOA. One possible approach is to sample states that are unlikely to occur if they are in the direction of further, but more likely states. As was pointed out by Glassman et al. [39], metrics such as dist (Sec. 5.5.4) can be very inefficient because that they do not take the dynamics and constraints of the system into account. Investigating alternative metrics is a possible topic for future work. It would also be valuable to thoroughly experiment more advanced density estimation techniques and different sampling strategies in Sec. 5.5.3, e.g., quasi-random, MCMC, etc., as was pointed out in Branicky et al. [16]. The controllers in this chapter are all time-based, which are known to lack robustness. It would be valuable to develop a method that efficiently transforms our time-based controllers into robust state-based controllers (e.g., [42], [62]).

DOA expansion could play an important role in having simulated characters perform skills from a large number of situations and transition between different skills. While previous work were limited to low-dimensional problems, we have shown how it can be performed efficiently for a high-dimensional simulated character. We hope that our work will stimulate future research on DOA expansion so that it becomes a common tool in control design.
Chapter 6

Robust Physics-based Motion Retargeting with Realistic Body Shapes

6.1 Introduction

Figure 6.1: Body-base retargeting. On the left are two different 3D bodies and their approximation in terms of “capsules”. The red body is the “source” body for which we have motion capture. The green body is our “target” body that we wish to animate. We fit a physics-based controller for the red body that is able to accurately mimic the motion capture; the upper right shows two example motions. We then fit a physics-based controller for the target body but, due to its different physical shape, it is unable to reproduce the motions of the smaller, fitter, source. The motion deviates in ways that are natural and consistent with the body shape. The approach provides natural, automatic, variability in motion through retargeting to new body shapes. See the accompanying video for animations.

We all move differently. How we perform an action depends on our body shape. The motions of a tall heavy man and a small slim woman balancing on one foot, hopping and bending, are distinct. Much of this is due to the specific distribution of mass over the body. Physical differences result in movement
differences. In this chapter, we explore how movements can be learned by watching other people, with
different body shapes, move. Given motion capture data from one subject, our goal is to realistically
animate humanoids with different body shapes doing the same movement. The result will be movements
that differ in ways that are tailored to the body of the individual preforming the action (Fig. 6.1). We
refer to the body shape of the character in the motion capture as the source body and the body shape
of the different character as the target body.

Motion capture (mocap) data is one of the main sources of animated behaviour in graphics due to
its inherent, ground truth realism that is unavailable to any other motion generation technique. While
commercial animation productions can afford recording the reference motion with actors that resemble
well the physique of their animated counterparts, this is typically not the case for animation in games
or crowd simulations. In these scenarios, it is not uncommon to apply mocap data to characters whose
bodies do not match the physical characteristics of the original actors. This mismatch results in well
documented artifacts like footskating [64], shape interpenetration or simply a lack of realism. We would
like to adapt a given mocap sequence to any possible human shape in a fully automated, but realistic
manner.

To a large extent, the lack of realism in the retargeting of mocap sequences is due to the physical
implausibility of the motion when applied to physically different target bodies. A natural solution to
this problem is to enforce physical constraints on the retargeted mocap data. The classic spacetime opti-
mization framework [132] (see Sec. 2.4) has been used for this purpose by imposing physical constraints
on an optimization problem that minimize the deviation of the simulated motion from the reference
motion [75]. However, such systems have traditionally been constrained to restrictive, expert-designed
constraints like foot placements [75], or are dedicated to very specific cyclic motions like walking, running
or cycling [55].

Moreover, most of the previous work considers unrealistic, mannequin-like body proportions (with
the notable exception of Hodgins et al. [54]), or retargets mocap data to non-human embodiments, like
animals, imaginary creatures or robots [80].

Our approach is different: Rather than retargeting motion despite large shape differences, we would
like to exploit common, natural shape differences between people to generate natural variations of human
motion given a single source mocap sequence. Unlike traditional retargeting of mocap, for which the
goal is precise kinematic mapping, our goal is to use mocap together with simulation to create physically
plausible maps. Due to differences in body shape (limb lengths and proportions, muscle and fat distri-
bution, etc.), a target subject can require different foot placements and can execute different motions
while attempting to replicate the source motion. Some of these subjects will struggle to imitate the
action, having to adapt it to their own body. In this chapter, we explore how natural motions, including
natural tracking “failures”, can emerge from a simple optimization objective. By using physical models
derived from real 3D scans, we show how our optimized physical motions have certain characteristics
present in real sequences, like heavier subjects raising their arms to improve balance or tripping when the
motion is too difficult to follow. This is different from previous work, which showcases motion mapped
to imaginary creatures, for which we have less intuition about how a natural motion looks.

Given a mocap sequence, we estimate how the sequence would be performed by actors of different
body shapes (Fig. 6.1). To that end, we use a more realistic, parametric, 3D model of the human body
[85]. Given any body shape, we approximate it with a set of “capsules” that capture the coarse shape of
the body, yet are simple enough to enable practical physics simulation (Fig. 6.1 left). We then generate
plausible motions for these bodies that try to replicate the kinematics of mocap, but take into account that body shape influences what we can do and how we do it (Fig. 6.1 right).

Another factor that makes motion unique for a particular shape and varied in different situations is their way of reacting to unexpected events. We would like to generate motions that are robust to external perturbations. To achieve this, we extend our work on domain of attraction expansion (Chapter 5) to adapt existing controllers to new observed states in an efficient manner.

Summarizing, our overall goal is to make mocap more useful by adapting it to new characters and external forces in a physically plausible way. That is, our goal is to make mocap data generalize better (more naturally and more robustly) to new shapes and new situations. Our approach is based on physical simulation of the motion and has several contributions. First, we generate more varied and realistic motions than previous methods by removing hard constraints on the retargeting objective and using real but diverse target body shapes. Second, we approximate real body shapes with geometric primitives, thereby obtaining realistic characters for physical simulation. Third, we develop a new algorithm for creating a tree of LQR controllers to sustain disturbances. This provides a step towards animated characters that behave realistically in a wide range of scenarios.

6.2 Related Work

There is significant prior work on modeling human body shape accurately and on retargeting mocap data to new characters. There is little work combining these threads. We argue here that this is important: as human body models become more important, realistic avatars will need to move realistically to avoid the “uncanny valley” [57].

**Mocap and animation.** Mocap is often used to directly animate a character given a mapping from the mocap skeletal motion to the character rig. Although the problem has received significant attention from the graphics community, today there is an enormous amount of manual labor in the process of mapping and refining these motions. A typical solution for this involves solving the pose for the target character through inverse kinematics. Since this problem is under-constrained, some approaches limit the frequency of the motion and minimize the amount of induced pose change [40]. Similar constraints are applied to simplified skeletons [90] in order to deal with both geometric and topological differences between skeletons.

An alternative approach is to learn a statistical motion model from several example mocap sequences. This allows for the animation of characters with natural variability. Research in biomechanics [123], machine learning [118], computer vision [35, 99, 110, 124] and graphics [43, 68, 106, 126] address this topic. These models typically do not take into account the dynamics of the motion, so directly applying this procedural motion to new characters produces results that may lack realism. Moreover, some of these models require motion recordings that are varied and aligned in time (e.g., cyclic motion), which limits their applicability.

**Physics and mocap.** Exploiting physics for processing and editing mocap data has a long history in graphics. Zordan et al. [144] propose a physics prior to estimate joint trajectories from 3D marker position data. The approach does not reconstruct the control torques to replicate the mocap motion, which is a challenging problem. Other researchers impose strict physical plausibility on either mocap data [73]
or human-edited animations [102, 109]. Direct methods for trajectory optimization (see Sec. 2.4.1) have been used for motion tracking [102, 114], but do not allow for contact changes and have yet to be demonstrated on a wide variety of movements.

In recent work, Han et al. [48] show that these limitations can be avoided by using the iterative Linear Quadratic Gaussian (iLQG) algorithm (see Sec. 2.4.2) and smoothed contact dynamics [117], thereby enabling mocap data tracking in near real-time. The main limitations are on the physical accuracy of the contact dynamics, which causes visible ground penetrations and contact forces to be applied at a distance, and on the realism of the control torques used (the torque limits are not mentioned, but other iLQG methods require unrealistically large torques to avoid local minima [117]). Other researchers have investigated stochastic global optimization and derivative-free methods. For instance, Liu et al. [82] show how a sampling-based approach can be used for the reconstruction of contact-rich movements. In subsequent work, the method is improved to synthesize highly-dynamic movements and to reduce motion wobbliness [80]. This is accomplished by computing the average of multiple solutions. Our work uses the sampling-based optimization method introduced in Chapter 3 to synthesize movements from high-level objectives, without prior data. Here, we show that the method can be used for motion tracking. Unlike Liu et al. [80], it produces good quality motions without requiring optimization over multiple solutions.

**Robust controllers.** Physical controllers inferred from mocap can be used to generate new motions with a wide range of variability in response to external disturbances. We can categorize methods that increase controller robustness depending on whether their optimization is online or offline. Recent online optimization methods have proved to be quite robust by planning over a future time horizon [46, 47, 48]. However, they can easily fall in local minima because their online nature do not allow exhaustive searches. In the offline category, the method of Ye et al. [140] synthesizes motion from data of real subjects reacting to external forces, but it is challenging to record reactions from a sufficiently varied set of people, motions and disturbances. More recent methods optimize linear feedback policies for motion tracking under disturbances [30, 79]. To move beyond linear policies, Tedrake [120] introduces a method to develop a nonlinear feedback policy that covers a domain of interest in state space by combining local linear quadratic regulators (LQR) controllers [26, 89] in a random tree. The method is only demonstrated on a low-dimensional problem (one degree-of-freedom). Inspired by this work, we show how to scale the approach to a 25 degrees-of-freedom character in Chapter 5. In this chapter, we improve our previous work by replacing PD controllers with more robust LQR controllers (see Sec. 2.5) and by showing how to robustly track a motion sequence.

**Human body shape.** There has been significant work on learning realistic human body shapes from 3D scans; we focus on these rather than hand-constructed models. These human shape models can typically be posed using a skeleton [10, 19, 51, 85, 143] or per-part rotations [12, 54], allowing animation from mocap data (e.g., [84]). The retargeting of mocap to a new body shape in Loper et al. [84] is purely kinematic and ignores the body shape in the process.

There is previous work on modeling how realistic body shape influences the dynamics of soft tissue [85, 101]. These methods predict soft-tissue deformations as a function of the motion of the body over time. They do not, however, do the opposite; that is, the body shape never influences the motion.
Combining body shape and physics. In early work, Hodgins and Pollard \cite{55} adapt a controller for a given body shape to a new body shape. This is described for a limited number of body shapes and focus only on a few cyclic motions.

Here, we combine the above lines of research and explore how body shape influences motion.

6.3 Methodology

The chapter uses the following methodology:

- The method in Chapter 3 is used to optimize a time-indexed spline that provides the target angles for a PD controller. The cost function evaluates how closely a physically simulated character tracks mocap data.

- By varying the shape of the simulated character, we investigate how shape affects motion. Realistic body models are obtained from scans of actual human shapes.

- We build random trees based on the method in Chapter 5 to robustly track mocap data. Instead of PD controllers for feedback, LQR controllers are obtained by first linearizing the dynamics at each timestep around the states and controls synthesized with trajectory optimization. Then, by solving a dynamic programming problem for the optimal controls given a quadratic cost function that penalizes deviations around the linearizing trajectory.

6.4 Optimization for Motion Re-Targeting

The motion retargeting problem involves transforming a motion performed by one character to a similar motion performed by a character with a different shape. Typically, the original motion cannot be performed exactly because of the different physical characteristics.

Let $\bar{q}_{j,t}$ be the observed angular value of joint $j$ at time $t$, $\bar{q}_t = [\bar{q}_{1,t}, \ldots, \bar{q}_{N,t}]^T$ be a vector of all joints at time $t$, and $\bar{q}_{1:T} = \{\bar{q}_1, \ldots, \bar{q}_T\}$ be the sequence of all joint angles over time.

Our goal is to estimate the torques necessary to reproduce the observed motion, subject to physical constraints. This is a hard optimization problem which we have not been able to solve directly. Our approach uses a trajectory optimization method for physically-based retargeting. In particular, we build on the method presented in Chapter 3 and optimize a physical model to approximate an observed trajectory of kinematics.

To simplify the optimization, we introduce a fictitious “control” trajectory $\hat{q}_{1:T-1}$ represented by a cubic B-spline. Optimizing the kinematics is achieved by indirect optimization of this control trajectory. The optimization below is over the spline knots, which correspond to the angles of the actuated joints. For clarity, however, we omit the knots below and simply write the trajectories.

Given a control trajectory, $\hat{q}_{1:T-1}$, the torques, $u_t$, at each timestep $t$ are determined by PD control:

$$u_t = k_p(\hat{q}_t - q_t) - k_d\dot{q}_t,$$

\label{eq:6.1}
\( x_{1:T} = [q_{1:T}, \dot{q}_{1:T}] \). Here, all joints have PD gain values of \( k_p = 700 \text{ Nm/rad} \) and \( k_d = 1 \text{ Nms/rad} \). The joint torques are limited to \( \pm 100 \text{ Nm} \) for all joints.

A good \( \hat{q}_{1:T-1} \) is one that produces motions \( x_{1:T} \) by forward simulation that match the observed mocap states \( \bar{x}_{1:T} = [\bar{q}_{1:T}, \dot{\bar{q}}_{1:T}] \), where \( \dot{\bar{q}}_{1:T} \) is evaluated with finite differences. We did not evaluate the stability of the computed derivatives, but they have not caused our optimization to fall in an undesirable local minima. We search for these using a sampling-based approach (described below). Given a sample \( \hat{q}_{1:T-1} \), we run the simulation to get \( x_{1:T} \) and evaluate it with respect to a cost function.

### 6.4.1 Cost Function

The cost function \( E(x_{1:T}) \) encourages the physics simulation motion to be as close as possible to the mocap motion. We use a weighted sum of objectives \( E_i \), with weights \( w_i \):

\[
E(x_{1:T}) = \sum_i w_i E_i(x_{1:T}).
\]

(6.2)

We now define and discuss the objectives.

The \( E_{\text{COM}} \) objective encourages agreement between the character’s simulated center-of-mass and the corresponding character’s center-of-mass in the mocap data:

\[
E_{\text{COM}} = \sum_t \| c_t - \bar{c}_t \|^2.
\]

(6.3)

Likewise, we use the \( E_{\text{COM}v} \) objective for the velocity of the center-of-mass:

\[
E_{\text{COM}v} = \sum_t \| \dot{c}_t - \bar{\dot{c}}_t \|^2.
\]

(6.4)

We always use the 2-norm in this chapter. Note that the center-of-mass can be computed from the pose of the body and its shape.

The \( E_{\text{trackingPoses}} \) and \( E_{\text{trackingVel}} \) objectives encourage the simulation to have similar poses and velocities as the mocap:

\[
E_{\text{trackingPoses}} = \sum_{j,t} \beta_j (q_{j,t} - \bar{q}_{j,t})^2,
\]

(6.5)

\[
E_{\text{trackingVel}} = \sum_{j,t} \beta_j (\dot{q}_{j,t} - \dot{\bar{q}}_{j,t})^2,
\]

(6.6)

where \( j \) indexes the character’s joints, excluding the root, and \( \beta_j \) is a weight. We currently use \( \beta = 1 \) for all joints, except for the hips and the shoulders, where we use \( \beta = 10 \) and \( \beta = 3 \) because of their importance on end-effector position. We do not have an objective to minimize the differences in end-effector positions directly, but it can be included if it is required for some application.

The \( E_{\text{ROOT}} \) term penalizes the difference between the unit quaternions parametrizing the root orientations [58]:

\[
E_{\text{ROOT}} = \sum_t \arccos(\langle |q_{\text{root}}, \bar{q}_{\text{root}}| \rangle),
\]

(6.7)
where $\langle \cdot \rangle$ denotes the inner product and $| \cdot |$ the absolute value.

Lastly, the $E_{\text{jerk}}$ term is used to reduce motion jitteriness:

$$E_{\text{jerk}} = \sum_t \| \dddot{q}_t \|^2. \quad (6.8)$$

We found the jerk term to produce better looking motions than the commonly used torque term:

$$E_{\text{torque}} = \sum_{j,t} u_{j,t}^2. \quad (6.9)$$

The weights used are $w_{\text{COM}} = 15$, $w_{\text{COM}v} = 0.5$, $w_{\text{trackingPoses}} = 0.5$, $w_{\text{trackingVel}} = 0.0005$, $w_{\text{ROOT}} = 4$, and $w_{\text{jerk}} = 1e^{-10}$.

### 6.4.2 Optimization

Optimization involves sampling control trajectories, $q_{1:T-1}$, running the forward simulation to get a motion, $x_{1:T}$, and then evaluating the cost using $E(x_{1:T})$. The optimization is performed with CMA [49], using 200 samples and 2000 iterations; that is, 400,000 simulations. The method is initialized with the $0$ vector, which corresponds to a standing pose. We use the MuJoCo [95] physics simulator with a 0.01 s timestep and a coefficient of friction of 1.

We divide the optimization for the entire motion in sequentially overlapping pairs of windows. We optimize using two adjacent 0.4s windows with 6 knots each, spaced at equal time intervals. Using two windows provides temporal continuity during optimization and we discard the results of the second window. The optimization time is about 25 minutes per pair of windows on a 4Gz Intel Core i7 machine with 4 cores. Our mocap clips are recorded at 60 fps and have a typical length of 5 seconds.

### 6.4.3 Mesh To Geometric Primitives

Given a realistic human shape, our goal is to retarget motion to this shape realistically. Actual differences across human bodies are more subtle than the ones in classic retargeting experiments, like asymmetric limb proportions [40, 80]. To capture these realistic shapes, we use data from the Dyna dataset [101], which registers real body scans (Fig. 6.2(a)) and normalizes their pose (Fig. 6.2(b), data provided by the authors). These pose-normalized meshes are pre-segmented into parts as defined by the SMPL body model [85].

Even though MuJoCo supports simulation of characters whose body parts are defined as triangulated meshes, we simplify our character mesh templates to a capsule-based representation to improve efficiency and compatibility with other engines.

To that end, given a mesh representing a human body in a canonical T-pose, we approximate it in terms of simple geometric primitives. Here, we use a “capsule” (i.e., extruded sphere) representation as illustrated in Fig. 6.2 (c). Capsules are one of the most common representations in game engines due to their efficiency for collision detection. Our representation consists of one capsule per body part in the SMPL template, excluding finger segments. The position of each capsule in the zero pose is determined by the joint location of the corresponding part in SMPL; see Eq. (10) in [85]. By having a one-to-one relationship between capsules and body parts, the kinematic structure in the SMPL body model is directly applicable to our capsules. The orientation of the capsules is set according to the
Figure 6.2: From scans to capsules. Each row depicts two different Dyna subjects, the first one is 1.69 meters tall and 62.2 kg, while the second one is 1.80 meters tall and 155.4 kg. From left to right, we show a) scans of the subjects, b) the unposed template as a triangulated mesh, c) the optimized capsule representation, and d) the adapted MuJoCo geometry primitives representation. In the first row, two last columns we represent joints color-coded according to their dimensionality: red, green and blue circles represent 3, 2 and 1 degrees of freedom respectively. The root joint is represented in magenta since it is represented with a 4D quaternion.
Chapter 6. Robust Physics-based Motion Retargeting with Realistic Body Shapes

principal direction of variation in the template body part. Furthermore, we optimize the radius and length of each capsule so that it adapts to each individual body shape.

Given a triangulated mesh template defined in terms of vertices \( T \), which generate a continuous surface mesh \( T \), our task is to compute the radius \( r \) and length \( l \) of all capsules to minimize the cost function:

\[
E(r, l) = \sum_{v_t \in T} \min_{p_c \in C(r, l)} \|v_t - p_c\| + \lambda \sum_{v_c \in C(r, l)} \min_{p_t \in T} \|v_c - p_t\|.
\] (6.10)

The vectors of radii and lengths, \( r \) and \( l \), generate a set of 3D vertices, \( C \), that form a union of capsules, corresponding to a continuous surface \( C \). This objective penalizes the distance from every template vertex \( v_t \) to the capsules surface \( C \), and the distance from every capsule vertex \( v_c \) to the template surface \( T \). The contribution of each distance is weighted by a parameter \( \lambda \) that is empirically set to 0.5. The optimal capsule radii and lengths are obtained through dogleg gradient-based optimization. Hands and feet are poorly approximated by capsules. Therefore, we model the feet with parallelepipeds and the hands with ellipsoids.

After this optimization, our capsule model resembles the original body template (see Fig. 6.2(c)) and can be kinematically animated using the same pose representation as in SMPL since they share the joint locations and kinematic structure. However, the parameterization of the SMPL pose is not suited for physical simulation because all the joints are three-dimensional. We reduce the articulation of the knees and toes to one-dimensional joints, and the elbows to two-dimensional joints, resulting in a total of 57 parameters coming from the 21 joints, the root translation and orientation, which is represented with a quaternion. The poses in SMPL’s axis-angle representation are converted for our simulation’s reduced Euler angles representation. Since both representations can be easily transformed into global transformation matrices (see Eq. 4 in [85]), we compute the reduced Euler angle representation whose global transformations are closest in terms of the Frobenius norm to the global transformations from the original axis-angle representation.

One of the benefits of using a body representation based on geometric primitives is the efficiency of collision detection. Our capsule shape optimization does not prevent intersections between capsules, which is beneficial to obtaining a better approximation of the shape (see, e.g., the capsules in the torso or the thighs). Therefore, collision detection in the physics simulation is disabled for the capsules that are intersecting in the rest pose. For each capsule, we assign a mass that is proportional to the volume minus the volumetric intersection with the parent capsules in the rest pose, so that no portion of the volume is counted twice.

6.5 Robust Motion Tracking

We have seen how to optimize a controller that tracks input mocap data. The controller is not robust to disturbances such as external forces, which is necessary for interactive settings. Intuitively, we would like to learn ways of steering disturbed motions back to the stable control model. During training time, we learn controllers that can do just that under random disturbances.

More specifically, we build a random tree of LQR controllers across our control solution. The method extends our work in Chapter 5. Instead of PD controllers, we use LQR controllers, which take into
account the coupling among all the degrees-of-freedom, therefore enlarging the region of stability of the controllers. We have introduced LQR control in Sec. 2.5. Here, we show how trees of LQR controllers can be used to robustly track a motion sequence, which is a problem that was not previously tackled.

6.5.1 LQR Feedback

Let \( x = [q, \dot{q}] \) and \( u \) denote the vector of joint torques. Using the discrete time-system dynamics form, we have

\[
x_{t+1} = f(x_t, u_t).
\]  

(6.11)

Given an initial trajectory \( x^0_{1:T} \) and \( u^0_{1:T-1} \), let \( \tilde{x}_t = x_t - x^0_t \) and \( \tilde{u}_t = u_t - u^0_t \). We linearize Eq. (6.11) around \( \tilde{x}_{1:T} \) and \( \tilde{u}_{1:T-1} \) to obtain a time-varying linear system

\[
\tilde{x}_{t+1} = A_t \tilde{x}_t + B_t \tilde{u}_t,
\]  

(6.12)

where \( A_t = \frac{\partial f}{\partial x}(x^0_t, u^0_t) \) and \( B_t = \frac{\partial f}{\partial u}(x^0_t, u^0_t) \). These terms are evaluated with numerical differentiation; see Mason et al. [89] for a discussion on differentiating the orientation of the root, which is an element of SO(3). Let \( J_t \) refer to the cost-to-go function, which accumulates current and future quadratic costs in our state and control variables

\[
J_T(\tilde{x}_T) = \tilde{x}_T^T Q_T \tilde{x}_T,
\]  

(6.13)

\[
J_t(\tilde{x}_t) = \tilde{x}_t^T Q_t \tilde{x}_t + \tilde{u}_t^T R_t \tilde{u}_t + J_{t+1}(A_t \tilde{x}_t + B_t \tilde{u}_t).
\]  

(6.14)

where \( Q_T, Q_t \), and \( R_t \) are given. The \( Q_T \) matrix determines the cost for being at the terminal state \( \tilde{x}_T \), and the \( Q_t \) and \( R_t \) matrices determine the costs for being at state \( \tilde{x}_t \) and using control \( \tilde{u}_t \) at timestep \( t \). It can be shown, e.g., [14, Chapter 4], that the optimal feedback law is given by

\[
\tilde{u}_t^* = L_t \tilde{x}_t,
\]  

(6.15)

or in terms of Eq. (6.11),

\[
\tilde{u}_t^* = u_t^0 + L_t(x_t - x^0_t).
\]  

(6.16)

The feedback term \( L_t \) is given by

\[
L_t = -(B_t^T K_{t+1} B_t + R_t)^{-1} B_t^T K_{t+1} A_t,
\]  

(6.17)

where \( K_t \) is defined as

\[
K_T = Q_T
\]  

(6.18)

and

\[
K_t = A_t^T [K_{t+1} - K_{t+1} B_t^T (B_t^T K_{t+1} B_t + R_t)^{-1} B_t^T K_{t+1}] A_t + Q_t.
\]  

(6.19)
We linearize the dynamics at each timestep since computing the feedback terms takes less than a minute for all our motions. If computation or memory requirements become too cumbersome, it is possible to linearize less often, e.g., at contact changes. We use the same cost-to-go function for all our motions. For simplicity, we choose (for all timesteps) $Q$ to be the identity matrix, and $R$ to be a diagonal matrix with entries 1, except for the torso joints that have entries 15. This means that our state cost penalizes errors across joints equally. The same comment applies for our control cost, except that we penalize errors in the torso more because unrealistically large torques were being used there. It would be valuable to investigate better cost functions, so that state errors in the more important joints for the task incur a larger cost. For example, in a walking motion, errors in the hips are more critical than in the wrists.

### 6.5.2 Trajectory-Tracking-LQR-Trees

In Sec. 6.4 we have shown how to synthesize a trajectory $x^0_{1:T}$ and $u^0_{1:T-1}$ that tracks mocap data. Our goal is to stay close to this trajectory when unexpected disturbances occur. Offline, we provide examples of disturbances. Each time a perturbed state fails to stay close to $x^0_{1:T}$, we attempt to steer it using trajectory optimization towards a region where it will succeed (a region covered by an LQR controller). If the connection is successful, LQR control is used on the newly optimized trajectory, hence making it more likely that we will be prepared for other disturbances. The LQR-Tree is a random tree of LQR controllers that attempt to cover as much as possible of a domain of interest in state space. The tree is constructed offline, but it can then be used for control in near real-time (see Sec. 5.5.2). In this work, we assume that the maximal length of the mocap sequence is 4 s or 5 s. To use the tree online for longer sequences would require a change in Algorithm 3, which we leave for future work.

Let $\Gamma$ be a tree of tuples $(x, u, L, p)$, where $x$ is a state, $u$ is a feedforward controller, $L$ is a feedback controller, and $p$ is a pointer to the parent node. We refer to the node with state $x$ as node $x$ for brevity. We start by stabilizing the trajectory $x^0_{1:T}$ and $u^0_{1:T-1}$ with LQR, and then add the corresponding nodes to $\Gamma$, where the parent of node $x_k^0$ is $x_{k+1}^0$.

Let $\chi$ denote the set of states from which we can track $x^0_{1:T}$ accurately enough, i.e., according to a given threshold and for a chosen distance metric. We refer to $\chi_{j:j'}$ as the subset of $\chi$ that is associated with nodes $x^0_{j:j'}$. The Trajectory-Tracking-LQR-Tree method systematically steers states originally outside $\chi$ inside it. The method samples random states as follows. We randomly choose a state $x^0$ in the sequence, and we apply an external force in a random direction, location and magnitude on the character for a given duration. Let $F$ refer to a list that specifies the maximum magnitude of the external force, e.g., $F = [100 \text{ N}, 200 \text{ N}, 300 \text{ N}, \ldots]$. We make the controller progressively more robust by first sampling $\Lambda$ states with $F(1)$, then with $F(2)$, etc.

Let $x^0$ denote a state outside $\chi$. We attempt to connect it back to $x_{k-\alpha:k+\alpha}^0$, where $\alpha \in \mathbb{N}$ modulates how important it is to re-enter $\chi$ near timestep $k$ (the start of the disturbance). This is done by choosing a nearby node $x^0$ in $\Gamma$ that is connected to a node in $x^0_{k-\alpha:k+\alpha}$. We then solve a trajectory optimization problem to steer $x^0$ to $x^0$, resulting in a new trajectory $x^0_{1:N}, u^0_{1:N-1}$, where $N$ is either a constant or a free variable in the optimization. If $x^0 \in \chi_{k-\alpha:k+\alpha}$, we have $x^0 \in \chi$ by performing LQR stabilization on $x^0_{1:N}, u^0_{1:N}$ and adding the new nodes to $\Gamma$. We then repeat the process by sampling a new state. In this work, we construct a random tree, not a rapidly exploring random tree as in [120] nor a dense random tree as in Chapter 5. However, the algorithm is compatible with these approaches. The method is described in Algorithm 4.
Algorithm 5 Tracking-LQR-Tree($x^0_{1:T}$, $u^0_{1:T-1}$, $Q$, $R$, $F$, $\alpha$)

1: $[A_{1:T-1}, B_{1:T-1}] \leftarrow \text{Linearize } f(x, u) \text{ around } (x^0_{1:T}, u^0_{1:T-1})$
2: $L_{1:T-1} \leftarrow \text{LQR}(A_{1:T-1}, B_{1:T-1}, Q, R)$
3: $\Gamma.\text{add}_\text{node}(x^0_T, 0, 0, \text{NULL})$
4: $\ldots$
5: $p \leftarrow \text{pointer to the node with } x^0_2$
6: $\Gamma.\text{add}_\text{node}(x^0_1, u^0_1, L_1, p)$
7: $\text{for } f_{\text{max}} \text{ in } F \text{ do}$
8: 8: $\text{for } \lambda = 1 : \Lambda \text{ do}$
9: 9: Randomly choose a state $x^0_k$ in $x^0_{1:T-1}$
10: 10: Apply random external force of max magnitude $f_{\text{max}}$
11: 11: $x^p \leftarrow$ the perturbed state
12: 12: if $x^p \notin \chi_{[k-\alpha, k+\alpha]}$ then
13: 13: $x^n \leftarrow$ nearby node connected to a node in $x^0_{k-\alpha, k+\alpha}$
14: 14: Solve a trajectory optimization to connect $x^p$ to $x^n$
15: 15: $(x^p_{1:N}, u^p_{1:N-1}) \leftarrow$ the synthesized trajectory
16: 16: if $x^p_N \in \chi_{[k-\alpha, k+\alpha]}$ then
17: 17: $[A_{1:N-1}, B_{1:N-1}] \leftarrow \text{Linearize } f(x, u) \text{ around } (x^p_{1:N}, u^p_{1:N-1})$
18: 18: $L_{1:N-1} \leftarrow \text{LQR}(A_{1:N-1}, B_{1:N-1}, Q, R)$
19: 19: $p \leftarrow \text{pointer to the node that causes } x^p_N \in \chi_{[k-\alpha, k+\alpha]}$
20: 20: $\Gamma.\text{add}_\text{node}(x^p_{N-1}, u^p_{N-1}, L_{N-1}, p)$
21: 21: $\ldots$
22: 22: $p \leftarrow \text{pointer to the node with } x^p_2$
23: 23: $\Gamma.\text{add}_\text{node}(x^p_1, u^p_1, L_1, p)$
24: 24: end if
25: end if
26: end for
27: end for

6.6 Results

To evaluate our method, we need movement data from people of different sizes and shapes. To that end, we use two sources of data. The first comes from the public Dyna dataset [31], which contains body shapes and motions of ten subjects with very different body shapes. While most available mocap data in the world is captured from relatively fit people, Dyna also contains motion of heavy people. We also use traditional mocap data [21], for which we have no ground truth 3D shape information. For this data, we use MoSh to extract the 3D body shapes and SMPL-compatible poses [84].

We estimate the controller for the ground truth 3D shape for each sequence. In the accompanying video [6] and in the figures, this source body is always shown in red. Even when the target character has the same (estimated) shape as the source character, it does not always perfectly track the motion. This is due to differences in the joint and torque limits, errors in the mocap data itself, local minima in the optimization, and differences between our articulated rigid body and the human body. Nevertheless,
Chapter 6. Robust Physics-based Motion Retargeting with Realistic Body Shapes

Figure 6.3: Trajectory Tracking LQR-Tree. The blue region illustrates $\chi$. The region is filled with LQR feedback. The long thick arrow is the sequence $x_{0:T}$ and the red circles are the random states sampled from $x_{0:T}$. The blue circles are sampled states that are successfully added to the tree. As can be seen in the figure, the nodes do not always connect back to the red circles due to the $\alpha$ term.

Figure 6.4: Turning-twist motion. In the top sequence, the target character has the same shape as the source character. In the bottom sequence, the target character is heavier. Note the position of the arms in the motion of the heavy character. In the accompanying video [6], it can also be seen that the heavy character has a different way of pivoting, and that a lighter character also displays a different way of performing the motion.

the resulting motions approximate the mocap, look believable, and are driven by a physical controller that has many advantages over simple retargeting as we show below.

We vary the body shapes among 10 different shapes from the Dyna dataset and optimize their motion controller. These target bodies are shown in green. Given this data, we mix and match source and target motions, for example taking the motion of a light person and animating a heavy person and vice versa. Our evaluation is visual and, for this purpose, we provide many examples in the accompanying video [6].

The accompanying video [6] and Fig. 6.4 show different characters performing a turning-twist motion. Note that the heavier character stretches its arms more and, interestingly, places its legs in a different configurations than the other characters; to turn-twist, it pivots around one foot instead of moving both feet. We also note that the lighter character has trouble keeping its arms at the same angle as the mocap character and is more bouncy during the motion, while the character with the original shape performs better overall.
Figure 6.5: **Recovery steps.** This sequence illustrates how a heavy character can lose balance when getting up too quickly from a crouch position. In order not to lose balance, the character takes a few recovery steps forward. After avoiding the risk of falling, it takes a few steps backwards to return to its desired position, and continue dancing.

When the characters have similar shape, the difference in their motion is subtle (e.g., a slightly lighter character being more bouncy or a slightly heavier character bending more to perform a movement). When the characters have vastly different shapes, the difference can be drastic, even leading to tracking failures. When tracking fails, it does so in a realistic way, similar to how you would expect a person fail to perform the motion.

Consider, for example, the dancing motion in Fig. 6.5. Here the source character is substantially slimmer than the target. It is therefore able to crouch lower and move quicker than it is possible for the heavy character. To avoid losing balance, the heavy character takes a few steps forward to stabilize itself, and then takes a few steps backwards to get back near the mocap data. What is interesting is that the steps taken have a natural look and they are entirely emergent from the optimization. Imposing foot contact constraints as is custom in classic spacetime optimization would have prevented the emergence of this natural behavior ([40], see Sec. 2.4).

Figure 6.6: **Air-kick with high, moderate and low torque limits.** With high torque limits, the character jumps too high (leftmost). With moderate torque limits, the result is more natural (middle), With low torque limits, the character just slides (rightmost).

We also present the results for a challenging air-kick motion by a slim character in Fig. 6.6. We show that with high torque limits (+300 Nm), the heavy character performs the motion reasonably well. The result seems unnatural since we would not expect a heavy character to jump this high. With more severe torque limits, ±100 Nm and ±50 Nm respectively, we get a more satisfying result: the character jumps with a lower height than in the former, and just slides on the floor in the latter.

An example illustrating the importance of the size of the time window is shown in Fig. 6.7, where a heavy character tracks two consecutive kicking motions. The usual time horizon for our optimization is 0.8s (that is two time windows of 0.4s each). In the accompanying video [10] and in Fig. 6.7 we see the character losing balance on a kick with the usual time horizon. When the time horizon is progressively
Figure 6.7: *Time Horizon.* The first sequence uses a 0.8 s window horizon in the optimization, the second 0.9 s, and the third 1.0 s. Note that in the second sequence, the heavy character does not kick as high as in the first sequence to avoid falling, but it still falls when performing the back kick. In the third sequence, the character stretches its leg backwards, but keeps its foot on the floor to stay in balance.

Figure 6.8: *Robustness.* The red line is an external force applied on the characters. The yellow character has an LQR controller. The orange character has an LQR-tree controller.

increased, the characters improves its motion, for instance by discovering that it cannot stretch its leg back as much as the source character.

We also present results where the target character is holding an object (a stick and a block), while the source character does not. The object’s weight has a visible effect on the motion; the heavier the object, the more it affects the movement and makes it hard to perform. However, the object can also be helpful to the character; e.g., the stick is used to maintain balance in difficult postures.

We tested the robustness of our solutions by applying on the character random external disturbances that are up to 500 N for 0.1 s. We show how the LQR-tree succeeds in performing the motion, whereas the LQR controller fails. In Fig. 6.8, we compare the reactions to a disturbance on the foot in the air when the character is kicking. As the character is about to fall in one direction, it rotates its arms in the opposite direction, thereby allowing itself to rotate back up and continue performing the motion.

We have chosen $\alpha = 25$ timesteps (see Sec. 6.5.2). A small value of $\alpha$ makes it more difficult to find a solution, while a large value means that we can skip tracking a portion of the sequence. The ankles and the torso movements are at times too jittery, indicating that more severe torque limits are necessary on these joints.

The experiments demonstrate that our method produces different motions for different body shapes,
that these motions are plausible and appropriate for the shape, and that our new LQR-tree formulation is robust to external forces.

One of our experiments highlights some limitations of our approach. In the accompanying video [6], we show how a thin character tracks the running-in-place motion of a heavy character, and how a heavy character tracks the running-in-place motion of a thin character. We note that the thin character stretches its arms too widely when performing the motion. But while the heavy character has a good reason to stretch its arms (i.e., it helps it maintain balance), the light character is more stable and does not need to do this. Consequently, the motion lacks perceptual believability. With multiple examples of the same motion by people of different shapes, one could develop a method to determine which portions of a motion should be tracked closely. We also note that the heavy character has difficulty running at the same pace as the thin character. A method could be developed to adjust the timing of the motion, so that a heavy character could run at a more comfortable pace.

Using LQR controllers instead of the PD controllers in Chapter [5] makes the process of finding robust solutions more efficient, as a single LQR controller often replaces 20-30 PD controllers. This comes at a very modest computational cost (computing the gains), but at a memory cost (storing the gains), which becomes an issue as the trees grow larger.

Our data could enable a perceptual study exploring how sensitive human viewers are to motions matching the body shape. We hypothesize that viewers are more sensitive to motion matching when the motion is dominated by the physics (ballistic motions, balancing, etc.). For other motions like walking or motions with little dynamics, we suspect that they are less sensitive. This could be important for knowing when mocap can be transferred without physics simulation.

One application of our work with objects is in the film industry. For example, when actors simulate a fighting scene with swords, sticks, batons, etc., they are not usually given actual weapons, but replicas that have very little weight. In the final motion, the actor is holding a massive, heavy object, but the motion may not reflect the weight of the object. Note that our current method does not model grasping; this is an interesting direction for future work.

The cases where the physical character departs from the mocap motion (e.g., by losing balance) are very interesting. Our method provides indicators on which portions of a motion will be difficult for people of different shape and strength. When combined with a more detailed musculoskeletal model, it could be useful in healthcare, e.g., to predict how an elderly person’s gait would change with a prosthesis.

Our work opens up new avenues of research, such as modeling how people learn motor skills. When attempting to learn a motor skill (e.g., karate, yoga, aerobic exercise, etc.), a novice watches an expert perform the movement, and then attempts to replicate it. Since the novice is not fully aware of his own limitations, he misses here and there, stumbles and falls over. Next time, he has a better idea of what he can and cannot perform. Instead of following the expert exactly, he adapts the expert’s movements to his own body. In other words, it is as if the novice has a longer time horizon. By varying the time horizon, we have a way to model how the motion changes as the performer moves from being a novice to an expert. It would be very interesting to compare our results with experimental data. Another interesting problem would be to model how people fall. In this work, when the character falls, it lets its head bang on the ground. An actual person would want to avoid a concussion and would place its end-effectors in a way to avoid injury.

In summary, we have presented a method to take mocap data from a source person and transfer it realistically to different target bodies in a way that is perceptually realistic and that can adapt to
external forces. The results suggest that this physics-based approach is viable for generating varied and realistic animations from mocap data. The source motion in our work does not need to be mocap data. It could come from classical rigged character animation to be “physicalised”. As computer vision research continues to make progress in tracking bodies in video [18], the source motion could also come from anywhere on the internet. One could imagine using the videos of athletes and dancers on YouTube[^1] to train our simulated characters with a large variety of skills.

[^1]: http://www.youtube.com
Chapter 7

Discussion and Future Work

Athletes, breakdancers, ballet dancers, and gymnasts are able to use their bodies in such a coordinated and precise manner that people enjoy watching them. Learning complex motor skills require long periods of trial-and-error. After some time, a movement can be successfully performed multiple times, but only in precise conditions that specify, for example, the position taken at the start of the movement, the roughness of the ground, the presence of wind, etc. With more practice, sensitivity to these conditions decreases. We refer to this learning process as acquiring robust motor skills. This thesis developed computational tools for the acquisition of robust motor skills in simulated characters. In Chapter 3, we presented a trajectory optimization method to synthesize complex movements from high-level objectives (e.g., place the hands as close as possible to the ground, place the feet as high as possible, etc.), without requiring key-framing, motion capture, or the contacts to be pre-specified. This allowed us to synthesize movements that were not previously synthesized with physics-based methods, such as crawling, headspins and handspins.

Our trajectory optimization method returns a control solution for precise initial and environmental conditions for the character. To overcome this limitation, we introduced a method to expand the domain of conditions where a skill can be performed in Chapter 5. The idea is to use trajectory optimization to steer states inside a domain where the control solutions are valid. Feedback is applied on the new trajectories to bring in even more states inside the domain. The first iteration of the algorithm used PD control for feedback, while the second iteration was made more efficient by using LQR control instead. Other than setting some constants (weights for the cost function, PD gains, etc), the method is entirely automatic. If the particular motion being performed is known, then it is possible to use human insight on the nature of the motion to design controllers that are suitable for many tasks and that are robust to disturbances. This is what was accomplished in Chapter 4 when investigating the design of general and robust feature-based controllers for rotational movements. Both methods are not mutually exclusive: human insight and more formal control methods such as domain of attraction expansion can be combined together.

When learning a new skill, we often start by watching others perform it. Think of a person learning martial arts. A novice starts by watching a teacher perform a movement. The novice then tries out the movement for himself, and after some practice, learns how to adapt it to his own body. This is the problem that was tackled in Chapter 6. We have shown how a motion performed with a given body shape can be realistically adapted to a different body shape. Interesting results include a heavier
Chapter 7. Discussion and Future Work

character taking a few recovery steps to avoid falling or stretching out its arms more widely to maintain balance, or a lighter character being more bouncy when turn-twisting. This allows a character to acquire a large repertoire of motor skills since it can learn from any character, without regards to its shape. In summary, our thesis introduced algorithms for characters to acquire skills that can be used in many situations and that can be learned from different characters.

We now discuss problems for future work. The method presented in Chapter 3 synthesizes motion with respect to a cost function that is a weighted sum of terms for task performance and a measure of effort expenditure. The synthesized motions do not always look natural due to the optimization falling in local minima or to inaccuracies in the biomechanical model of the human body. It would be valuable to see how the synthesized motions differ when more detailed models are used (e.g., by using muscle forces instead of joint torques). This was investigated for walking, but not for other movements [93, 129]. This would give an indication of when a model is detailed enough for a task. Even with a very detailed model, the optimization would still fall in local minima in some situations. Hence, there is a need to combine physics-based optimization approaches for motion synthesis with prior data such as a motion capture database. The prior data could help avoid local minima (e.g., unlikely postures), while the physics-based optimization could help generalize to domains with little prior data.

Our work on motion retargeting can help indicate when a person with a given body shape would have difficulty performing a movement. A promising application of the method would be to predict the movements of the elderly, the injured and the disabled. Comparing the results with experimental data could allow us to uncover some features of the cost function that we, as humans, optimize to produce our movements. We also want to explore if varying the temporal window can be used to model motor learning. We have seen examples where shortening the window makes the synthesized movement look like it is being performed by a novice. For instance, when the character fails, it fails in the portions of the movement that seem difficult for novices to perform. We would like to study this in more detail and in more examples, to conclusively determine if varying the temporal window is an effective method to synthesize a continuum of motor skills, from novice to expert.
Bibliography


