LOCOMOTION SYNTHESIS IN COMPLEX PHYSICALLY SIMULATED ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

by

Jie Tan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
December 2015

LOCOMOTION SYNTHESIS IN COMPLEX PHYSICALLY SIMULATED ENVIRONMENTS

Approved by:

Karen Liu, Advisor
College of Computing
Georgia Institute of Technology

Frank Dellaert
College of Computing
Georgia Institute of Technology

Greg Turk, Advisor
College of Computing
Georgia Institute of Technology

James O'Brien
Department of Electrical Engineering
and Computer Science
University of California at Berkeley

Jarek Rossignac
College of Computing
Georgia Institute of Technology

Date Approved: October 23, 2015

ACKNOWLEDGEMENTS

First, I want to thank my advisors, Greg Turk and Karen Liu. It is a great pleasure to work with you two in the last six years. You serve as great role models not only in research but also in life. I feel that I am deeply indebted to both of you. You have built up a huge portion of knowledge that I possessed today. You encouraged me to pursue my interests and to follow my passion. You taught me how to think creatively and helped me develop rigorous procedures to approach research problems. More importantly, you helped me build my confidence, especially in my first years in US when I was facing language and culture barriers. I feel extremely fortunate to have you as my advisors. Without your guidance, I would be nowhere as successful as I am today. I believe that what I have learned from you in the last six years will carry on to influence me and benefit me throughout my whole life.

I want to thank the members of my thesis committee: Jarek Rossignac, Frank Dellaert and James O'Brien as well as my qualifier committee members: Irfan Essa and Blair Macintyre. Each of you has helped me become a stronger researcher in your own way. You have broaden my knowledge, shaped my rigorous thinking, taught me effective ways of communication and helped me develop social skills.

I also want to thank my friends and labmates: Huamin Wang, Chris Wojtan, Sumit Jain, Yuting Ye, Karthik Raveendran, Mark Luffel, Tina Zhuo, Yunfei Bai, Sehoon Ha, Kristin Siu, Albert Li, Yangfeng Ji, Alexander Zook, Alex Clegg, Edward Liu, and many more. Thank you for your helpful discussions with me about my research, my presentation and my life.

Finally, I want to thank my family. I want to thank my wife, Yuting, for your unlimited support every day throughout my PhD journey. I also want to thank you for your understanding, especially for your help on modeling, rendering and video editing before each SIGGRAPH deadline. I want to thank my parents, Yangsun and Xiuchun. It is you that shaped me into who I am today. I am extremely grateful to your guidance and supports.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Understanding and synthesizing locomotion of humans and animals will have far-reaching impacts in computer animation, robotic and biomechanics. However, due to the complexity of the neuromuscular control and physical interactions with the environment, computationally modeling these seemingly effortless locomotion imposes a grand challenge for scientists, engineers and artists. The focus of this thesis is to present a set of computational tools, which can simulate the physical environment and optimize the control strategy, to automatically synthesize locomotion for humans and animals.

We first present computational tools to study swimming motions for a wide variety of aquatic animals. This method first builds a simulation of two-way interaction between fluid and an articulated rigid body system. It then searches for the most energy efficient way to swim for a given body shape in the simulated hydrodynamic environment.

Next, we present an algorithm that can synthesize locomotion of soft body animals that do not have skeleton support. We combine a finite element simulation with a muscle model that is inspired by muscular hydrostat in nature. We then formulate a quadratic program with complementarity condition (QPCC) to optimize the muscle contraction and contact forces that can lead to meaningful locomotion. We develop an efficient QPCC solver that solves a challenging optimization problem at the presence of discontinuous contact events.

We also present algorithms to model human locomotion with a passive mechanical device: riding a bicycle in this case. We apply a powerful reinforcement learning algorithm, which can search for both the parametrization and the parameters of a control policy, to enable a virtual human character to perform bicycle stunts in a physically simulated environment.

Finally, we explore the possibility to use the computational tools that are developed for computer animation to control a real robot. We develop a simulation calibration technique which reduces the discrepancy between the simulated results and the performance of the

robot in the real environments. For certain motion planning tasks, this method can transfer the controllers optimized for a virtual character in a simulation to a robot that operates in a real environment.

CHAPTER I

INTRODUCTION

Mother Nature has created a diverse set of awe-inspiring motions in the animal kingdom: Birds can fly in the sky, fishes can swim in the water, geccos can crawl on vertical surfaces, and cats can reorient themselves in mid-air. These motions are elegant, agile, efficient, and more importantly, inspiring. Some of the best human innovations can be attributed to the inspiration from locomotion of animals. For example, airplane, submarine and more recently, MIT cheetah and StickyBot, all can find deep roots from nature's design. Studying the motions designed by nature is not only a scientific quest that quenches our curiosity, but also an important step towards synthesizing them in a way that can fundamentally change our life.

The research of motion synthesis will have broad and impactful applications. For example, in movies and games, we need to faithfully reproduce the motions of animals and humans on the screen. Moreover, we often need to create animations for fantasy creatures that may not exist on our planet. The synthesized motions of these characters need to appear realistic to give the audience an immersive experience. The applications of motion synthesis go far beyond just the entertainment industry. In health-care, it can enable us to build powerful robotic exoskeletons that help gait habilitation, assist paralyzed to walk and boost human power. In robotics, motion synthesis can help us develop robotic systems with extensive flexibility, agility and maneuverability. These next-generation robots are expected to free human from dangerous missions in military, exploration and rescuing.

Although we often take our locomotion ability for granted since we can perform them so effortlessly, motion synthesis is a notoriously difficult problem because locomotion involves sophisticated neuromuscular control, sensory information processing, motion planning, coordinated muscle activation, and complicated interactions between the body and its physical

environment. It poses a grand challenge for scientists, engineers and artists. Despite extensive studies for centuries, we are still nowhere near fully understanding the underlying physics and control mechanisms that govern these motions. The focus of this dissertation is to present a set of powerful computational tools that facilitate the motion synthesis for locomotion of humans and animals, with the applications in character animation and robotics.

In the last few decades, we have seen tremendous advance in motion synthesis. Some of the most breathtaking movies, such as Harry Potter, Avatar and Life of Pi, rely heavily on computer generated character animations. Nowadays, it is almost impossible for the audience to tell apart the computer-synthesized motions from the real footage. In robotics, motion synthesis is also widely used to design agile and robust locomotion controllers. The MIT Cheetah can run up to 10 miles per hour and jump over obstacles. The Big Dog from Boston Dynamics can walk robustly in adversary environments, including icy or rocky terrains. The soft body robots can take advantage of their flexible bodies to navigate narrow and unstructured spaces. The humanoid robots, such as Petman and Asimo, was able to demonstrate a repertoire of locomotion skills, including walking, running, dancing and climbing stairs.

Behind these realistic animations and sophisticated robots lies countless hours of tedious manual work of highly-specialized experts. For example, to produce a 100-minute feature film at Pixar can take dozens of artists and engineers more than five years of development. In today's animation pipeline, the most popular techniques to synthesize motions are key frames or motion capture, both of which require artistic expertise and laborious manual work. Even worse, the knowledge and efforts that are put into one animation sequence are not necessarily generalizable to other motions. In my point of view, these are not efficient or principled ways of motion synthesis.

A principled way to synthesize locomotion is to develop computational tools, including physical simulations and optimization methods, to study the fundamental factors that have shaped our motions. Our motions are shaped through

millions of years of optimization (evolution) in a world that obey physical laws. This insight has motivated a new paradigm of *simulation-driven controller optimization* in both character animation and robotics. The two key components of this paradigm are physical simulation and controller optimization. We first build a physical simulation to model the physical world and then perform optimization to control the motions of characters so that they can move more efficiently and robustly in the simulated environment. With these computational tools, natural movements, including walking, running, flying and swimming emerge automatically. In addition to these basic locomotion tasks, the research in this field has developed algorithms that allow virtual characters to recover balance from unexpected perturbations, to move in different styles, to navigate through rough terrains and to demonstrate highly skillful stunts.

Despite the impressive achievements of this paradigm, the gracious, agile and diverse motions of the real creatures still remain unmatched, especially when the creatures are moving in complex environments. Performing controller optimization in a complex physically simulated environment presents unique challenges. First of all, complex physical environments are time-consuming to simulate. Second, the characters that we wish to control may have a large number of muscles (actuators), which results in a high dimensional nonconvex optimization problem. Finding the global optima is usually computationally infeasible. In addition, the forceful interactions between the character and its environment introduce further complications in simulation and optimization. Generally speaking, controlling high-dimensional dynamic systems, governed by highly nonlinear differential equations and coupled through complex mechanisms, is considered a nearly unsolvable problem. As a result, most of the prior research make simplifications on simulation models and optimization algorithms to make the computation tractable. However, many of these simplifications were made without considering the optimality of the control problems, which severely limits the power of this simulation-driven optimization approach. In this dissertation, we will investigate some of these simplifications and develop novel algorithms for those components that should not be simplified.

1.1 Thesis Overview

This dissertation presents a set of computational tools to synthesize locomotion in complex physical environments. In contrast to prior works that use simplified models, we develop new algorithms to improve and to combine the state-of-the-art simulation and optimization techniques to tackle the challenges of motion synthesis. We start with a survey of related work in the fields of character animation and robotics (Chapter 2). We then investigate motor control for various locomotion tasks in a hydrodynamic environment (Chapter 3), for soft body characters (Chapter 4) and with a passive mechanical device (Chapter 5). In Chapter 6, we explore the techniques to transfer the controllers developed in the simulation to robots operating in the real world. We conclude the thesis with conclusions and suggestions for future work (Chapter 7).

1.1.1 Locomotion in Hydrodynamic Environment

Figure 1: Aquatic creatures swim in a physically simulated hydrodynamic environment.

The oceans cover over seventy percent of the area on our planet. They contain a wide variety of creatures that use swimming as their primary form of locomotion. Scientific studies show that the swimming gaits of the aquatic creatures are highly efficient compared to the man-made underwater vehicles. Studying their swimming motions could help us discover better propulsion mechanisms and design more efficient undersea vehicles to explore the largest uncharted territory on our planet. In Chapter 3, we apply numerical optimization to automatically discover the most energy efficient swimming gaits for given aquatic creatures in a physically-simulated hydrodynamic environment.

A main challenge in physical simulation is to model the complex interaction between two different types of dynamic systems, such as the two-way coupling between the fluid

and a swimmer represented as an articulated rigid body system. We present an accurate physical simulator [149] that simultaneously solves the Navier-Stokes equations for fluids, the Lagrangian dynamics for an articulated rigid body and matches their accelerations at fluid-solid boundaries. The simulation results of swimming fish and eels show vortex trails that are in agreement with laboratory measurements.

Simulating fluid itself is hard; optimizing locomotion in a hydrodynamic environment is even more challenging. Previous methods have resorted to simplified fluid models. However, studies have shown that fish takes advantage of surrounding vortices, which are omitted in the simplified models, to provide energy boosts. Incorporating an accurate Navier-Stokes fluid model in the simulation presents new challenges in controller optimization: The optimization space is full of local minima due to the chaotic fluid behavior. Evaluating the gradient of the objective function is time-consuming. We demonstrate that sampling-based optimization algorithms are effective tools to overcome these challenges. This approach found efficient swimming motions that are comparable to those of real-world animals (Figure 1).

1.1.2 Locomotion for Soft Body Characters

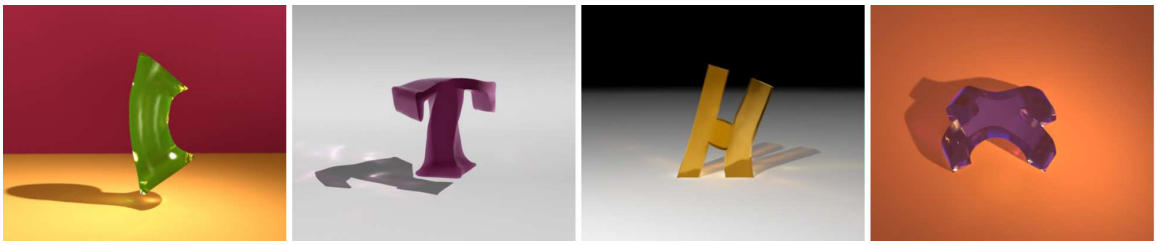

Figure 2: Soft body characters perform different forms of locomotion.

While most research in character animation and robotics focus on characters that are made exclusively from rigid parts, we have seen an increasingly amount of efforts in the last few years to develop soft body robots. While these research demonstrates a huge potential of soft body robots and their broad applications, it demands a new set of computational tools to study and to synthesize motions for soft body characters.

To model soft body characters, we not only need to simulate the passive dynamics of deformation, but also the actuation of muscles. In Chapter 4, we present a new mathematical model for artificial muscles [151] that are motivated by the muscle structures of soft body animals in nature. Similar to real muscles, these artificial ones are arranged in groups and only allowed to contract. Complex movements need to be accomplished by the coordinated contraction of multiple muscle groups. We develop a finite element method with this muscle model to simulate soft body animals.

Optimizing controllers in the presence of discontinuous contact forces is a long-standing problem. Controlling locomotion for soft body characters (Figure 2) exacerbates the difficulty. The deformation of the body constantly changes the contact configuration between the character and the ground. A common practice is to separate contact planning and controller optimization. We identify that this simplification eliminates effective control strategies, and this causes the soft body character to lose balance. We derive an elegant solution to this problem that combines contact planning with controller optimization. We formulate a quadratic program with complementarity conditions (QPCC) and develop an efficient solver for QPCC problems derived from locomotion control with contacts. As a result, effective control strategies emerge automatically from the QPCC solution.

1.1.3 Locomotion with Mechanical Devices

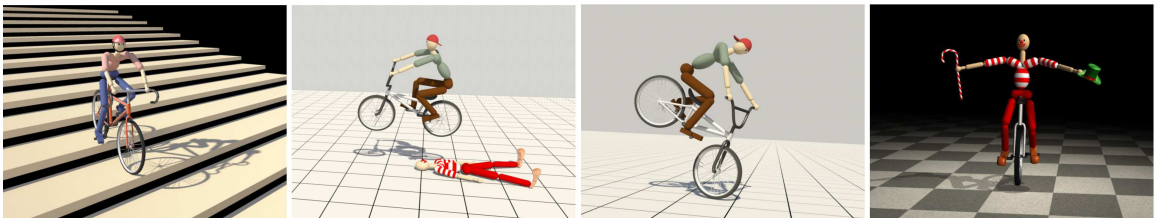

Figure 3: A human character performs stunts on a road bike, a BMX bike and a unicycle.

Human has invented numerous mechanical tools to ease our life. Robots in the future can work much more efficiently if they can take advantage of these existing tools. Instead of manually programming the robots to master each tool, we hope that they can learn how to use them autonomously. Learning to ride a bicycle is an excellent case study. The bicycle, which has greatly boosted the efficiency of locomotion, was voted as the best invention since

the 19th century. Even though the dynamics of bicycles is relatively well-understood, riding a bicycle is challenging due to the inherently unstable dynamics. In Chapter 5, we present a machine learning algorithm [148] that allows a virtual character to learn to ride a bicycle in a physically simulated environment.

In addition to the basic maneuvers, we hope that the character can learn more challenging but visually spectacular stunts (Figure 3). Performing stunts requires fast reaction, precise control and years of practice. This challenges the best human riders, let alone a machine learning algorithm. When we design the policy search algorithm, we find that the widely-used assumption of a predetermined controller parametrization severely limits the search space. It leaves the hard work to the user to design a good parametrization. We decide that optimizing the parametrization automatically is equally important as optimizing the parameters. Our algorithm evolves both the policy parametrization and the parameters simultaneously. This significantly improves the quality of the resulting controllers. Eventually, our simulated characters learn to perform a wide variety of bicycle stunts within hours, which is even faster than the best human stunt bikers.

1.1.4 Locomotion Controller Transfer from Virtual to Real World

Above three works demonstrate that with the powerful computational tools for character animation, natural, agile and robust motions can be synthesized efficiently and autonomously. However, creating lifelike robots is still an extremely challenging, trial-and-error process that is restricted to experts. The fast evolution of 3D printing technology will soon trigger a shift in the robotics industry from mass production to personalized design and fabrication, which will result in an immediate need for a faster, cheaper and more intuitive way to design robotic controllers. The computational tools we developed can potentially automate and streamline the process if we can transfer the controllers from the virtual simulation to the real world.

Transferring controllers optimized in a simulation onto a real robot is a non-trivial task. An optimal controller that works in a state-of-the-art simulation often fails in a real environment. This is known as the *Reality Gap*. This gap is caused by various simplifications

in the simulation, including inaccurate physical model, unmodeled actuator dynamics, assumptions of perfect sensing and zero latency. In Chapter 6, we investigate some of these simplifications and present a general framework of simulation calibration. Simulation calibration optimizes simulation parameters to minimize the discrepancy between the data collected from real experiments on the robot and that generated in the simulation. After calibration, the simulation becomes more faithful to the real-world dynamics. Controllers that are designed with the improved simulator can work in both the virtual and the real world.

1.2 Contributions

The computational tools presented in this dissertation provide several contributions to the communities of character animation and robotics. These contributions are as follows.

A stable simulation of two-way coupling between fluids and articulated rigid bodies. We present a novel swimming simulator that can simultaneously solve the dynamics of fluids, articulated rigid bodies and their two-way interactions. Compared to the traditional two-way coupling solver that alternates the fluid update and the rigid body update, our method is more numerically stable. We are able to use time steps of 33ms for all our experiments without any stability problem. Using larger time steps makes our simulation orders of magnitude faster than the alternating solver. As a result, we can discover a swimming gait within days of computation while using the traditional two-way coupling technique may take weeks.

A finite element simulation with a muscle model for soft body animals. Based on the muscle structure of muscular hydrostat in real soft body animals, we develop a muscle model for the simulated characters. Combined with the finite element method for the passive deformation of the body, it provides intuitive ways to control the character in a coordinated manner. The use of this muscle model reduces the dimensionality of the control problem and results in more natural-looking motions.

An QPCC solver for controller optimization with changing contacts. Controlling locomotion with contacts is a long-standing problem in continuous optimization because changes of contact situation (static, sliding or breaking) can introduce discontinuities to the dynamics. A commonly-used technique is to separate contact planning and controller optimization. However, this could eliminate effective locomotion strategies. We solve this problem by formulating a quadratic program with complementarity conditions (QPCC). We also develop an efficient solver for QPCC problems with contacts. This method can optimize both the contact situation and forces simultaneously. As a result, interesting and effective locomotion strategies emerge automatically from the QPCC solution.

A reinforcement learning algorithm that searches both the parametrization and the parameters of a policy. We present the first reinforcement learning algorithm which demonstrate that extremely challenging locomotion tasks, such as bicycle stunts, can be learned efficiently in simulation. Most of the stunt actions are learned within one hour, which is even faster than the performance of best human stunt bikers. These results present a new benchmark for future research in reinforcement learning. The key to such an efficient learning algorithm is an evolutionary optimization that can search for the parametrization and the parameters of a control policy simultaneously. We believe that this reinforcement learning algorithm can be generalized to master other challenging locomotion tasks.

CHAPTER II

RELATED WORK

This chapter presents a summary of previous work that is most relevant to our methods. The two key components of our computational tools are physical simulation and controller optimization. We will begin with a brief review of related work in these two fields. Since the ultimate goal of our research is an end-to-end computational framework that can autonomously design robotic controllers from high-level task specifications, we need to transfer the controllers found in the simulation to the real robots. We will conclude this chapter with an overview of related works in controller transfer from virtual to real environments in robotics.

2.1 Physical Simulation

In order to simulate the physical environment, researchers often borrow numerical tools from the applied mathematics literature. The most widely used methods include Eulerian methods [43, 140], Lagrangian methods [125, 113], Hybrid methods [40] and position-based dynamics [115, 100]. These methods have produced realistic simulations for a wide variety of dynamical systems, such as rigid bodies, deformable bodies, fluids and clothes. In the next few sections, we will focus on simulation of fluids, soft bodies, contacts and bicycle dynamics, which are used in our work.

2.1.1 Fluids

Simulating fluids involves numerically solving the governing equations of motions, Navier-Stokes equations. Two popular methods in computer animation are Lagrangian method and Eulerian method. The Lagrangian approach treats the fluid as a particle system [106, 125, 113, 128]. In contrast, the Eulerian approach treats the fluid as a continuous velocity field discretized on a computational grid. Foster and Metaxas's work [43] was the first that solved the full 3D Navier-Stokes equations to animate fluids. Stam [140] improved it,

achieving unconditionally numerical stability by introducing the semi-Lagrangian method for the convection term and implicit solver for the viscosity and pressure terms. Although the Eulerian method is more computationally expensive, it has produced stunning visual results when simulating a wide range of physical phenomena, including fire and smoke[119, 42], explosion [182], surface tension [62, 169], non-Newtonian fluid [47, 14] and multi-phase flow [98, 5]. We choose to use the Eulerian method in Chapter 3 because it is easier to enforce the incompressibility condition of fluids, which turns out to be important in swimming motions.

When studying swimming motions, simulating fluids alone is not enough. Swimming involves two-way interactions between the character and the fluid. Accurately modeling this two-way interaction is essential to simulate swimming motions. Many researchers have proposed various ways to simulate the two-way coupling between fluids and solids. Takahashi et al. [147] presented a simple alternating two-way coupling method. The velocities of the solid objects served as the boundary conditions for the fluid motion while the pressure field solved from the Navier-Stokes equations was integrated at the solid surface to provide a net force and a net torque exerted on the solid objects. Arash et al. [8] represented the solids by mass-spring models and fluids by marker particles. The interactions were calculated through the mutual forces between the marker particles and mass nodes at the interface. Carlson et al. [26] proposed the rigid fluid method that treated solids as fluids at first and then projected the velocity field in the solid region onto a subspace satisfying the rigid constraints. Guendelman et al. [51] made use of an alternating approach that was generalized to include octree and thin shells. They solved the pressure field for a second time by adding solid masses to the fluid grid density, which improves the pressure field. Klingner et al. [80] used a tetrahedral mesh for accurate boundary discretization and extended the mass conservation (projection) step to include the dynamics of rigid body. This was extended to model the interaction between fluids and deformable bodies [29]. Batty et al. [17] derived a fast variationl approach that allowed sub-grid accuray using regular grids. Robinson et al. [129] developed a generic and momentum conserving technique to couple fluids to rigid/deformable solids and thin shells. The coupled system is symmetric indefinite

and solved using MINRES. Our two-way coupling technique resembles that of Klingner et al. [80]. The main difference is that we simulate the interaction between fluids and articulated rigid bodies instead of rigid bodies. Furthermore, our represent of the articulated rigid body in the generalized coordinates further complicates the swimming simulation.

2.1.2 Soft Bodies

In Chapter 4, we study the locomotion on land for soft body characters. To simulate the squishy characters, we need a physical model for deformable objects. Since the seminal work introduced by Terzpoulos [155], researchers in computer graphics have simulated a wide variety of deformable phenomena including cloth [10, 24], elasticity [114], and plasticity [121, 14]. Although mass-spring systems [64, 96] are widely-used due to its simplicity, it is difficult to specify the spring stiffness to capture the desired material property. A more accurate technique is the Finite Element Method (FEM) [16], which typically uses a tetrahedral mesh to find weak solutions of the dynamic equations. The robustness of FEM simulation can be improved by handling inverted tetrahedra [65], remeshing ill-conditioned elements [14], or preserving volume without locking artifacts [66]. To improve the performance of FEM simulation, linear strain model and precomputed stiff matrix are often used. However, these models are only valid for small deformations. To simulate large deformations, Müller *et al.* [114] proposed a corotational method to fix the volume inflation artifacts. Nesme *et al.* [116] suggested that linearization around the current deformed configuration reduces ghost torques. Precomputed deformation modes have also been used to interactively deform large structures [69, 12, 78]. In our work, we choose to use the corotational linear FEM [114, 116] to simulate the soft body dynamics. We also use implicit integration to further increase the simulation stability and speed.

2.1.3 Contact Modeling

Locomotion on land is achieved by a character actively and purposefully pushing the ground. As a result, an equal and opposite ground reaction force is exerted on the character through the contacts and changes its center of mass. Modelling contacts between two dynamic systems is an active research area in physical simulation. Early simulations use penalty forces

[155], whose magnitude is proportional to the depth of penetration. Despite its simplicity, this method requires stringent simulation time steps and careful parameter tunning. More recently, constraint-based methods, such as linear complementarity problem (LCP), are used to handle contacts. Stewart and Trinkle proposed an LCP formulation using an implicit time-stepping method to guarantee non-penetration, directional friction, and approximated Coulomb's friction cone conditions [144]. Based on the LCP framework, many improved contact models were introduced recently in computer graphics, including using an efficient iterative method [39], a simple staggered sequence of projections [71], or a progressive constrained manifold refinement [122]. Our implementation of contact handling in Chapter 4 is based on the LCP condition [151], which is solved using Lemke's algorithm.

2.1.4 Bicycle Dynamics

In Chapter 5, we study human riding a bicycle. Accurately modeling the bicycle dynamics is important in this research. Early studies of bicycle dynamics date back to more than a century ago. As described in Meijaard et al. [102], Whipple [173] and Carvallo [27] independently derived the first governing equations of bicycle dynamics. These equations were revised to account for the drag forces [30], tire slip [138] and the presence of a rider [165]. Rankine [127] discussed the balance strategy of "steering towards the direction of falling", which forms the foundation of many studies on bicycle balance control, including ours. Despite this long history of research and the seemingly simple mechanics of bicycles, some physical phenomena exhibited by the bicycle movement still remain mysterious. One example is the self-stable characteristic of bicycles: A moving bicycle within a narrow range of forward speed can automatically correct its falling motion without any human intervention. In addition to the early belief that this phenomenon was attributed to gyroscopic effects of the rotating wheels [79] or the *trail*[1] [70], Kooijman et al. [83] showed that the mass distribution over the whole bicycle also contributes to the self-stability. Even though the dynamic equations provide us with some intuition, we do not solve them directly in our work because they are tailored specifically to normal riding situations where both tires touch

[1]The trail is the distance between the front wheel ground contact point and the steering axis.

the ground. This will be a major restriction in bicycle stunts. Instead, we modify a generic physical simulator, Open Dynamic Engine (ODE) [139], to model the bicycle dynamics.

2.2 Controller Optimization

Controlling character locomotion has been extensively studied in both computer animation and robotics. Starting from the seminal work of Hodgins et al. [61] which demonstrated sophisticated biped controllers, such as gymnastic vaulting or tumbling, researchers have investigated different forms of locomotion, including walking [181, 172], running [61, 86], flying [175] and swimming [49]. We refer the readers to an update-to-date review [44] on physically-based character animation.

Two main categories of methods for controller optimization are trajectory optimization and reinforcement learning. Trajectory optimization was introduced more than two decades ago to generate physically plausible character animations [174]. The user provides the start and the end configurations of the character and an objective function, this method generates trajectories of joint angles by minimize the objective function subject to the physical constraints. Trajectory optimization has been applied to control the iconic jumping Luxo Jr lamp [174], humanoid characters [94, 67, 179], and characters with arbitrary morphologies [167]. The resulting motions are physically plausible and follow the animation principles such as anticipation and follow-through [156]. However, trajectory optimization often leads to large optimization problems, which is time-consuming to solve and the solutions could often stuck at local minima.

Reinforcement learning algorithms optimize controllers by formulating and solving a Markov Decision Process (MDP). It finds optimal actions at different states. If the transition model is known, value iteration is a popular method. Researchers have successfully applied (fitted) value iteration to generalize motion capture data [158, 92], to carry out locomotion tasks [31], and to manipulate objects with hands [6]. Applying value iteration to continuous state and action spaces is nontrivial because discretizing the space does not scale well to high dimensions [146] and using function approximation often converges to a poor local minimum or might not converge at all [157, 22]. Policy search [117] is another reinforcement

learning algorithm, which can be easily generalized to high-dimensional continuous space. It directly searches for a mapping between the state space and the action space, without the need to construct a value function. Many studies on locomotion control [180, 170, 33, 172, 45] performed policy search on parameterized controllers. However, the policy parametrization need to be carefully designed because policy search can only search the control space defined by the parametrization. A poorly designed parametrization could eliminate effective controllers or introduce too many local minima to the control space. In Chapter 5, we will discuss how to automatically design controller parametrizations. Given a parametrization, one popular optimizer for policy search is Covariance Matrix Adaptation (CMA) [53]. It is a stochastic optimization algorithm, which has been proven effective even when the problem domain is highly discontinuous [176, 171, 107]. While most of the related work mentioned above is in the field of character animation, interested readers can find surveys of reinforcement learning in robotics [81]. Next, we will review the related works that are specific to the locomotion tasks in the dissertation.

2.2.1 Locomotion in Hydrodynamic Environments

Tu and Terzopoulos pioneered the animation of swimming fish using a a mass-spring system for the fish body and a simplified fluid model [162, 154, 50]. They used simulated annealing and the simplex method to discover swimming gaits. Their simulation also incorporated vision sensors, motor controllers, and behavioral modeling of eating, escape, schooling and mating. The major difference between their paper and ours lies in the fluid model and the optimization technique. This early paper used a simplified fluid model while ours adopts a full Navier-Stokes solver and introduces a two-way coupling method between fluids and articulated figures in generalized coordinate.

Sims [137] investigated the simulated evolution of creature locomotion. Sims' creatures were composed of blocks that are connected by articulated joints. He used genetic programming to evolve both the creature bodies and their controllers. In addition to walking and jumping behaviors, some of his creatures also learned to swim in a simplified fluid environment.

Wu and Popović [175] used an articulated skeleton and deformable elements for feathers in order to animate the flight of birds. They used an optimization process to find the best wing beats in order to accurately follow a given path. Yang et al. [178] used an articulated body representation, a simplified fluid model, and several layers of control to model human swimmers. Si et al. [135] applied a biologically motivated Central Pattern Generator (CPG) to control human swimming motions. Although they simulated the fluid by solving Navier-Stokes equations, their two-way coupling method was based on the simplified model. Kwatra et al. [85] used an articulated body representation and two-way coupling between the body and a fluid simulation to model human swimming. They used motion capture data of swimming motions as input to the swimmer control. Lentine et al. [90] used an articulated skeleton with a deformable skin layer and two-way coupling to a fluid simulator to model figures that are moving in fluids. They optimized for certain styles of motion using objective functions designed for effort minimization and drag minimization/maximization. Their results also clearly demonstrated that using a full fluid simulator gives more realistic results than using a simplified fluid model.

In the field of computational fluid dynamics (CFD), there is a small but growing literature on the simulation of swimming creatures. These studies are typically focused on a single swimming style of one particular creature, and they usually make use of sophisticated fluid dynamics code, at a large cost in computational complexity, to generate more accurate and detailed fluid simulation. Often these studies are informed by laboratory studies of the creature in question, including flow data that has been gathered using methods such as particle image velocimetry [48]. A good representative of such work is the investigation of Shiragaonkar et al. of the knifefish, which is a fish that propels itself using waves that travel along its elongated lower fin (gymnotiform swimming) [133]. The simulator for this work used an immersed boundary method, and the simulations were performed on a 262 compute node Linux cluster. Another example of such a study is the work of Kern and Koumoutakos [73] on the simulation of eels (anguilliform swimming). In this work, the fluid grid is matched to the eel body by using a cylindrical grid in most of the domain and a hemisphere-based grid for the head of the eel. They used the CMA technique [53] to

optimize a five parameter motion model.

2.2.2 Locomotion for Soft Body Characters

Controlling physically simulated soft bodies is a practical problem in computer animation. Previous work offers a rich repertoire of techniques that enable the artists to control the shape of soft bodies. Many methods proposed to track a given input animation or keyframes using interpolated resting shapes [82], a constrained Lagrangian solver [19], a linear quadratic regulator [13], or reduced spacetime optimization [11]. Martin *et al.* [101] introduced an example-based approach for simulating soft bodies with desired behaviors. The user supplies the system with a few poses to guide the simulation results toward preferred shapes. Shape control for soft bodies has also been applied to physics-based facial animation. Sifakis [136] formulated an optimization to automatically determine muscle activation that tracks a sparse set of motion capture markers.

In contrast to shape control, locomotion control for soft bodies is relatively less explored in computer animation. Previous work has shown that mass-spring systems can be used to simulate motion of worms, snakes, and fish [105, 161, 49]. Miller [105] utilized anisotropic frictional forces such that a worm can slide forward by contracting elastic body segments. Tu and Terzopoulos [161] applied a simple fluid dynamic model to provide forward thrust when a fish deforms its body. Kim and Pollard [76, 77] demonstrated that much more complex locomotion can be achieved by effective soft body control. They combined an efficient skeleton-driven FEM simulator and an optimization-based controller to create many interesting behaviors, such as a star fish crawling out of a box and a fish flipping back and forth. More recently, Schulz et al. [131] animated the soft body locomotion using spacetime optimization.

Although there are a large amount of related work in locomotion control for characters that are represented as articulated rigid bodies, there is one important difference in controlling soft body locomotion: we cannot apply the commonly-used joint torques to soft body characters. We need different actuation signals. For example, Coros et al. [34] used

dynamically changing rest shapes as actuation signals to control locomotion. Another actuation signal is muscle contraction. One important contribution of our work in Chapter 4 is a muscle model. Previous work has modeled dynamics of muscles and demonstrated that complex interplay among bones, muscles, ligaments and other soft tissues can be modeled for individual body parts, including the neck [89], the upper body [185, 37, 88], and hands [160, 145]. Using the volumetric data from the visible human data set, Teran *et al.* integrated a B-spline representation for muscles, a tetrahedra mesh for soft tissues, and a triangulated surface for each bone to simulate musculoskeletal behaviors [152, 153]. A striking difference of our work is that we focus on controlling deformation behaviors without skeletal support. This type of control mechanism resembles biomechanical movements using muscular-hydrostats, such as the tentacles of cephalopod mollusks or the trunks of elephants [74]. By using muscle contraction alone, we can generate functional motor skills, including elongating, shortening, bending, and twisting. We show that visually appealing behaviors that cannot be produced by skeleton-based systems emerge with appropriate control.

The main difficulty in locomotion is to control an under-actuated system by exploiting external contact forces. Velocity-based LCPs for contact modeling can have infinitely many solutions, but general LCP solvers, such as Lemke's algorithm, are incapable of ascertaining the quality of the solutions for a given criterion. This drawback is particularly undesirable when solving an optimal control problem that exploits the contact and dynamic state of the system. One approach to this problem is contact-invariant optimization [109, 110]. Due to the lack of robust schemes to formulate optimization with arbitrary objective function and linear complementarity constraints, many previous methods explicitly assumed that the contacts remain static [2, 67, 76] while optimizing control forces subject to equations of motion. This assumption significantly restricts the effectiveness of the controller for locomotion and balance because the controller is not allowed to actively exploit contact breakage, slipping contacts, or rolling contacts to achieve control goals. A few previous studies in mathematics addressed the problems of linear and convex quadratic programs with complementarity constraints (LPCCs and QPCCs) [63, 9]. They showed that global resolution of nonconvex problems in these two subclasses, including those infeasible and

unbounded, can be accomplish in finite time. In Chapter 4, we introduce an efficient QPCC solver for controller optimization problems with contacts.

2.2.3 Locomotion with a Passive Mechanical Device

Compared to walking and running, fewer studies focus on locomotion that involves a character controlling another device with complex dynamics. Van de Panne and Lee [164] built a 2D ski simulator and that relies on user inputs to control the character. This work was later extended to 3D by Zhao and Van de Panne [184]. Planar motions, including pumping a swing, riding a seesaw and even pedaling a unicycle, were studied [60]. Hodgins et al. [61] demonstrated that normal cycling activities, including balance and steering, can be achieved using simple proportional-derivative (PD) controllers for the handlebar angle. These linear feedback controllers are sufficient for normal cycling, but they cannot generate the bicycle stunts demonstrated in Chapter 5.

One central problem of riding a bicycle is to keep balance. The balance problem has been studied extensively in previous work on locomotion synthesis. Balance can be maintained by exerting virtual forces [124, 32], applying linear feedback [87, 181, 35, 32], using nonlinear control policies [112], planning the contact forces [112, 151], employing reduced models [159, 86, 107, 32, 179] and training in stochastic environments [171].

The bicycle control problem has been investigated in the reinforcement learning literature. Randløv and Alstrøm [126] used SARSA(λ), a model free reinforcement learning algorithm, to learn to balance a bicycle and ride to a goal. This algorithm requires a large number of simulation trials and the converged result is still not ideal. Ng and Jordan [117] applied policy search to the same bicycle learning problem. They parameterized the policy with neural networks and used the policy gradient to find the optimal network weights. The resulting controller significantly outperformed the results in the previous study. Our method is inspired by the policy search algorithm. However, to adopt this algorithm to learn more challenging tasks, we need to overcome two difficulties: First, we do not have reliable policy gradient information because of the frequent contact events. Second, we do not know a good policy parametrization, which is difficult to design manually by trial and

error for each bicycle stunt. We use NEAT [142] to address both difficulties. NEAT was first introduced to graphics by Allen and Faloutsos [4]. They used it to evolve locomotion controllers but were not able to achieve stable and sustained walking.

2.3 Controller Transfer from Virtual Characters to Real Robots

Research in computer animation has demonstrated robust locomotion control for challenging tasks in physically simulated environments. However, we have not seen any robots that can demonstrate similar capabilities. This gap is known as *Reality Gap*: A controller that can work effectively in physical simulation may not work in the real environment. This gap is caused by sensor noise, latency, hardware limitations, unmodeled dynamics, inaccurate physical model and other unknown factors. Nolfi and Floreano [120] outlined the problems that are related to crossing the Reality Gap and identified the key difficulties. A large amount of approaches were proposed in robotics to cross this Reality Gap. We refer the readers to Eaton [38] for a comprehensive review of this topic.

One way to cross the Reality Gap is to increase the robustness of the controller so that it is more likely to work in a different environment. A more robust controller can be found by injecting noise to the simulation [104, 68, 103], leveraging multiple simulators [20], and optimizing the controller through ensembles of perturbed models [108]. Although these methods do not explicitly involve experiments on the robot during controller optimization, they have been shown effective to increase the probability of a successful controller transfer.

Another direction to close the Reality Gap is to improve the simulation model so that it better reflects the real world dynamics. The simulation is improved by measuring and minimizing the discrepancy between the simulation results and the data collected in robot experiments. Ha and Yamane [52] modeled this discrepancy using Gaussian process. Abbeel et al. [1] used an inaccurate physical model but successively grounded the policy evaluations using real-life trials. Mouret et al. [111, 84] derived a measure of transferability by comparing fitness scores between the simulation and the real experiments. Grounded simulated learning approach [41] iteratively optimized the controller, measured the discrepancy and modified the simulator using supervised learning algorithms. Bongard and Lipson [21]

coevolved the controller and the simulator using an iterative estimation-exploration process. Similarly, Zagal et al. [183] introduced the "back-to-reality" approach, which also involved the coevolution but used a different measure of discrepancy. In Chapter 6, we present a simulation calibration process to transfer controllers from virtual to real environments.

CHAPTER III

LOCOMOTION IN HYDRODYNAMIC ENVIRONMENTS

3.1 Motivation

We live on a planet that is covered mostly by water, in which a wide variety of creatures use swimming as their primary form of locomotion. There are an astonishing variety of body shapes and patterns of motion that are used by swimmers across the animal kingdom. Some of the many creature swimming patterns from nature include using thrust from a tail, moving an elongated body sinusoidally, using paddle-like motions of flippers, kicking with legs, and gentle bird-like flapping of fins. Our research goal is to develop a general platform for finding efficient swimming motion for a given creature body shape. There are a number of application areas that can benefit from realistic swimming simulation, including feature film animation [143], biological investigation of swimming mechanics [73, 133], locomotion of user-created creatures in video games [56], and the invention of new modes of propulsion for underwater vehicles [15].

Today, most scientific models for swimming motion are customized to specific species with predefined locomotion patterns [133]. These models are highly accurate but are difficult to generalize to a variety of creatures. The existing 3D swimming animations, on the other hand, demonstrate a lifelike underwater ecosystem with rich variety of creatures. However, their motions are typically animated manually or based on simplified physical models. Having a generic set of tools that can produce physically realistic aquatic motion for a wide array of creatures is challenging and has not been shown in previous work.

At the heart of synthesizing realistic aquatic locomotion lies the problems of simulation and control. Solving these two problems simultaneously under hydrodynamics presents some unique challenges. First, the relation between the movement of the aquatic animal and the forces exerted by surrounding fluid is extremely complex. Thus it is difficult to solve using an optimization approach. Any small changes in undulation or flapping gait can

result in drastically different control strategies. In addition, the morphology of aquatic animals is astonishingly diverse and results in fundamentally different locomotion mechanisms. Designing control strategies based on ad-hoc observation or careful tuning of parameters would be extraordinarily difficult to generalize to the vast biodiversity found in nature.

This chapter describes a complete system for controlling a wide variety of aquatic animals in a simulated fluid environment. Our goal is a system that balances between physical realism and generality. Given an aquatic animal that is represented by an articulated rigid body system, our system can automatically find the optimal locomotion in a hydrodynamically-coupled environment. Our system does not require any prior knowledge of the animal's behavior and minimizes the effort of manually tuning the physical and control parameters.

The system consists of two main components: simulating motion and optimizing control strategies. We simulate articulated rigid bodies submerged in invisid, incompressible fluid governed by the Navier-Stokes equations. The animal can exert torques to exercise each actuated joint. Through accurate two-way coupling of the rigid bodies and the fluid, the joint motion will lead to *some* locomotion in the fluid, but purposeful and balanced locomotion requires careful coordination and synchronization among those actuated joints. The second component provides an automatic way to discover joint motion that achieves a desired goal in locomotion (i.e. a joint motion that yields the fastest or the most energy efficient locomotion). We employ an optimization technique called Covariance Matrix Adaptation (CMA) to explore the domain of possible joint trajectories.

We evaluate our system by demonstrating optimized swimming gaits for a wider variety of aquatic animals and swimming strategies, including clownfish, eels, sea turtles, frogs, manta rays and some imaginary creatures. In addition, we compare the swimming motion in a Navier-Stokes fluid with motion in a simplified fluid. Our results show that these motions can differ dramatically depending on which fluid model is used.

3.2 Swimming Simulation

We simulate fluids by solving the Navier-Stokes equations on a MAC grid and we simulate the articulated rigid body using generalized coordinates. We modify the projection step of

the fluid solver to take into consideration the dynamics of the articulated figure.

3.2.1 Fluid Simulation

We simulate fluid using the inviscid, incompressible fluid equations (sometimes called the Euler equations):

$$\nabla \cdot \mathbf{u} = 0$$

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla \mathbf{u}) - \frac{1}{\rho}\nabla p + \mathbf{f}$$

where $\mathbf{u} = (u, v, w)$ is the velocity of fluids, p is the pressure, ρ is the density and $\mathbf{f}$ accounts for the external body forces. We do not include a viscous term because such effects are negligable for the motion of the large animals in our examples. If we were studying swimming of millimeter sized creatures, however, incorporating viscous effects would be mandatory.

The standard way to solve the above equations on a MAC grid can be described in following two steps. First, we calculate an intermediate velocity field $\mathbf{u}^*$ by only considering the convection $\mathbf{u} \cdot \nabla \mathbf{u}$ and the body force $\mathbf{f}$:

$$\mathbf{u}^* = \mathrm{SL}(\mathbf{u}^n, \Delta t) + \Delta t \mathbf{f} \tag{1}$$

where $\mathbf{u}^n$ is the velocity at n^{th} time step. We use the Semi-Lagrangian method [140] to integrate the convection term and apply BFECC [75] to reduce the numerical dissipation.

Next, we solve the following Poisson equation with Neumann boundary conditions $\mathbf{u} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}$ at the solid boundary and Dirichlet boundary conditions $p = 0$ at the free surface. Then we project the intermediate velocity field to ensure the incompressibility condition.

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{2}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p \tag{3}$$

In this work, we modify the second step ((2) and (3)) to take into account the interaction between the fluid and the articulated rigid bodies.

3.2.2 Articulated Rigid Body Simulation

In this section, we will describe the numerical techniques that we use to move the body parts of an articulated figure. Later, in Chapter 3.3, we will describe the optimization technique that we use to discover efficient swimming gaits.

The dynamic equations of an articulated rigid body in generalized coordinates can be expressed as follows.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \tau_{int} + \tau_{ext} \tag{4}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are vectors of positions, velocities and accelerations of joint degrees of freedom respectively. $\mathbf{M}(\mathbf{q})$ is the mass matrix and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ accounts for the Coriolis and Centrifugal force. τ_{int} and τ_{ext} are internal and external generalized forces.

Given the current state $\mathbf{q}^n$ and $\dot{\mathbf{q}}^n$, we can evaluate $\mathbf{M}$ and $\mathbf{C}$ of (4). For the external forces τ_{ext}, we consider the fluid pressure force. We make use of the modified PD controller of Tan et al. [150] in order to calculate the internal force τ_{int} that closely tracks a reference trajectory. Although the details of this method can be found in [150], we include an overview of this method below. The reference swimming trajectory is computed by an optimization process described in Chapter 3.3. Once we know both the external and internal forces, we can solve the acceleration $\ddot{\mathbf{q}}^n$ and advance to the next time step via explicit Euler integration.

Modified Proportional-Derivative Controller In computer animation, a PD servo (5) provides a simple framework to compute control forces for tracking a kinematic state of a joint trajectory:

$$\tau^n = -k_p(q^n - \bar{q}^n) - k_d\dot{q}^n \tag{5}$$

where k_p and k_d are the gain and damping coefficient. In general, high gain PD servos result in small simulation time steps in order to maintain stability.

The aquatic creatures in this work require high gain PD servos to track the desired swimming gait closely against strong fluid pressure. However, we cannot reduce the time step to accommodate stability due to the time-consuming fluid simulation. To achieve

these two conflicting goals, large time steps and high gains, we modify the PD controller as follows. Instead of using the current state q^n and $\dot{q}^n$ to compute the control force, we compute the control forces using the state at next time step q^{n+1} and $\dot{q}^{n+1}$:

$$\tau^n = -k_p(q^{n+1} - \bar{q}^{n+1}) - k_d\dot{q}^{n+1} \tag{6}$$

eq. (6) can be linearized at q^n and $\dot{q}^n$ as:

$$\tau^n = -k_p(q^n + \Delta t\dot{q}^n - \bar{q}^{n+1}) - k_d(\dot{q}^n + \Delta t\ddot{q}^n)$$

Applying the modified PD controller to the articulated rigid body simulation with multiple degrees of freedom, we solve the acceleration as

$$\ddot{\mathbf{q}}^n = (\mathbf{M} + \mathbf{K}_d\Delta t)^{-1}(-\mathbf{C} - \mathbf{K}_p(\mathbf{q}^n + \dot{\mathbf{q}}^n\Delta t - \bar{\mathbf{q}}^{n+1}) - \mathbf{K}_d\dot{\mathbf{q}}^n + \tau_{ext})$$

where both $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal matrices that indicate the gains and damping coefficients.

3.2.3 Two-way Coupling Between Fluids and Articulated Rigid Bodies

The two-way coupling between the incompressible fluid and the articulated figures should satisfy following three conditions.

1. The normal velocity at the interface between the fluids and the articulated rigid bodies should agree with each other.

2. The motion of the articulated rigid body resulting from the fluid pressure force must be consistent with the Lagrangian equations of motion.

3. The fluid should be incompressible.

Two-way coupling is ensured by having the fluid exerting pressure forces on the rigid bodies, while at the same time the motion of the rigid bodies affects the pressure distribution of the fluid.

Our simultaneous two-way coupling technique is inspired by Klingner et al. [80] since we both start from the acceleration at cell faces. Their method uses a tetrahedral mesh

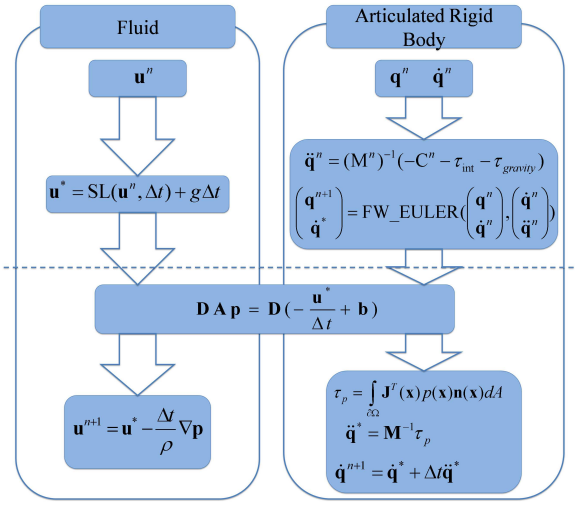

Figure 4: The computational steps for simultaneous coupling between fluids and articulated rigid bodies.

to represent the fluid, and their rigid bodies are in Cartesian space. Our simulator uses a regular MAC grid and we couple this fluid with articulated figures that are described in generalized coordinates. Similar to Klingner et al. [80], we split the coupling into two steps (Figure 4). In the first step, the two systems are solved independently ignoring the pressure. The fluid solver calculates the intermediate velocity field $\mathbf{u}^*$ using (1). The articulated rigid body solver determines the acceleration $\ddot{\mathbf{q}}$ without external pressure forces and calculates the intermediate velocity $\dot{\mathbf{q}}^*$.

In the second step, we consider the motion of the two systems together so that they will satisfy all above three conditions. We first voxelize the body segments of the articulated figure (represented by water-tight polygon meshes) onto the MAC grid and we mark those cells inside the body segments as SOLID. For two-way coupling, we are particularly interested in the faces between a SOLID cell and a FLUID cell (defined as *coupled faces*). The velocity at a coupled face can be expressed in generalized coordinates by the Jacobian of the articulated rigid body and the joint velocity:

$$\mathbf{u}^*_{solid} = \mathbf{J}\dot{\mathbf{q}}^*$$

where $\mathbf{J}$ is the $3 \times m$ (m is the number of degrees of freedom) Jacobian matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_m} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_m} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \cdots & \frac{\partial z}{\partial q_m} \end{pmatrix}$$

Now consider the effect of the pressure field, which exerts forces and applies accelerations along the face normals $\mathbf{n}$. If a face is shared by two FLUID cells, the acceleration is $\frac{1}{\rho}\nabla\mathbf{p}\cdot\mathbf{n}$. If a face is shared by a FLUID cell and a SOLID cell (a coupled face), we need to take into account all the pressure values surrounding the articulated rigid body. We first construct a $k \times n$ selection matrix $\mathbf{S}$ to pick out of $\mathbf{p}$ the pressures at the coupled faces, where k is the number of the coupled faces and n is the number of FLUID cells. Thus the vector $\mathbf{Sp}$ constains all the pressure values surrounding the articulated rigid body. Each element p_i of $\mathbf{Sp}$ contributes a pressure force $(\Delta x)^2 p_i \mathbf{n}_i$ to the articulated rigid body, which we transform to the generalized coordinate:

$$\tau_{p_i} = \mathbf{J}_i^T (\Delta x)^2 p_i \mathbf{n}_i$$

The total generalized force exerted by the fluid pressure on the articulated rigid body is

$$\tau_p = (\Delta x)^2 \hat{\mathbf{J}}\mathbf{Sp}$$

where $\hat{\mathbf{J}} = [\mathbf{J}_0^T\mathbf{n}_0 \quad \ldots \quad \mathbf{J}_k^T\mathbf{n}_k]$. The pressure force results in the acceleration in generalized coordinates

$$\ddot{\mathbf{q}}_\mathbf{p} = \mathbf{M}^{-1}\tau_p$$

We transform the acceleration back to Cartesian space, and the magnitude of the acceleration at the coupled face is

$$a = \mathbf{n}^T(\mathbf{J}\ddot{\mathbf{q}}_\mathbf{p} + \dot{\mathbf{J}}\dot{\mathbf{q}}^*)$$

The second term $\dot{\mathbf{J}}\dot{\mathbf{q}}^*$ comes from the fact that the Jacobian matrix changes over time. Stacking the accelerations at the coupled faces into a vector, we have

$$\mathbf{a} = (\Delta x)^2 \hat{\mathbf{J}}^T \mathbf{M}^{-1} \hat{\mathbf{J}}\mathbf{Sp} + \dot{\hat{\mathbf{J}}}^T \dot{\mathbf{q}}^* \tag{7}$$

where $\dot{\hat{\mathbf{J}}} = [\dot{\mathbf{J}}_0^T\mathbf{n}_0 \quad \ldots \quad \dot{\mathbf{J}}_k^T\mathbf{n}_k]$.

Since the velocity field should be divergence free at the beginning of the next time step,

$$\nabla \cdot \mathbf{u}^{n+1} = \nabla \cdot (\mathbf{u}^* + \Delta t \mathbf{a}) = 0 \tag{8}$$

the accelerations due to the pressure must satisfy the following equation.

$$\nabla \cdot \mathbf{a} = -\frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*$$

Putting everything together, we reach the final linear system:

$$\mathbf{DAp} = \mathbf{D}(-\frac{\mathbf{u}^*}{\Delta t} + \mathbf{b}) \tag{9}$$

$$\mathbf{A} = \begin{cases} \frac{1}{\rho}\mathbf{G} & \text{faces shared by two FLUID cells} \\ (\Delta x)^2 \hat{\mathbf{J}}^T \mathbf{M}^{-1} \hat{\mathbf{J}} \mathbf{S} & \text{coupled faces} \end{cases}$$

$$\mathbf{b} = \begin{cases} \mathbf{0} & \text{faces shared by two FLUID cells} \\ -\dot{\hat{\mathbf{J}}}^T \dot{\mathbf{q}}^* & \text{coupled faces} \end{cases}$$

where $\mathbf{D}$ and $\mathbf{G}$ are the discretization of the divergence and gradient operators on a MAC grid.

We construct a system of linear equations (9) for the pressure field, which considers all of the three conditions to be satisfied by the coupled system. The fluid and solid velocity agrees at the interface (condition 1) because the velocity defined at the coupled faces are shared by the fluid and the articulated body. The movement of the articulated rigid body under the fluid pressure satisfies the equation of motion (condition 2) because (7) is derived from the dynamics (4). The fluid is incompressible (condition 3) because we enforce the divergence free condition by (8). The linear system is of the same size as the discretized Poisson equation (2) in a typical fluid simulation. The main difference is that the rows correpsonding to the cells adjacent to the SOLID cells have more non-zero entries. Furthermore, it is also symmetric positive definite, which allows the use of fast solvers such as the Preconditioned Conjugate Gradient method. After solving the pressure field, we project the velocity field to make it divergence free using (3) and update the articulated rigid body by considering the pressure forces.

3.3 Swimming Gait Optimization

We have described the two-way interaction between fluids and an articulated rigid body system. In particular, Chapter 3.2.2 describes how we move the body parts using torques and how we compute the torques for a given reference gait. In this section, we describe an algorithm to automatically design optimal controllers for an active articulated rigid body systems that is moving in a hydrodynamic environment. Our method generates physically realistic strokes based on the swimming efficiency of the stroke.

3.3.1 Swimming Gait Representation

Given the geometric and physical properties of an articulated rigid body system, we formulate an optimization to solve for the reference trajectory of PD controller at each actuated joint, q_i. We want to use a compact representation for the reference trajectory because incorporating a fluid simulation into the optimization is computational intensive. Because aquatic locomotion is typically cyclic, we parameterize the reference trajectory as periodic cycles in generalized coordinates.

$$q_i(t) = A_i \sin(\frac{2\pi t}{T_i} + \phi_i) + C_i$$

where A_i, T_i, ϕ_i and C_i are the amplitude, period, phase and offset of a sine function. Using this parameterization, each reference trajectory $q_i(t)$ is parameterized by four values. In most cases we just optimize over two parameters, amplitude and phase, and leave the period and offset fixed.

3.3.2 Objective Function

The objective function in our optimization tries to balance between efficiency and energy expenditure of the swimming gait; the creature should move as fast as possible in the desired direction without using too much energy. Furthermore, the creature should try to avoid self-collisions and remain within the joint limits. In practice, the choice of objective function can vary by creatures, fluid conditions, or the user's application. Here we choose a simple objective function to find natural swimming motion:

$$E = -E_{distance} + w_1 E_{deviation} + w_2 E_{energy} + w_3 E_{collision} \tag{10}$$

where $E_{distance}$ measures the change of the creature's root position $\Delta\mathbf{p}$ along a specified direction $\mathbf{d}$ from time 0 to time t_f:

$$E_{distance} = \mathbf{d}^T(\Delta\mathbf{p})$$

$E_{deviation}$ measures the deviation from the specified direction and the initial orientation.

$$E_{deviation} = ||\Delta\mathbf{p} - \mathbf{d}^T(\Delta\mathbf{p})\mathbf{d}|| + ||\Delta\alpha||$$

where $\Delta\alpha$ stands for the change of root orientation in t_f, expressed using the exponential map. Since we're optimizing the gait of straight swimming, we penalize any orientation changes. We choose the weight $w_1 = 0.2$ for all the examples.

E_{energy} penalizes the energy expenditure of the swimming gait. We calculate the work done by the actuated joints over the duration of the swimming gait:

$$E_{energy} = \int_0^{t_f} \sum_i \tau_i \dot{q}_i dt$$

Instead of penalizing energy expenditure linearly, we modulate E_{energy} with a discontinuous function represented as the objective weight w_2. Instead of constantly trying to avoid using any energy, this modulation allows the creature to freely consume a certain amount of energy, while avoiding excessive use of torques.

$$w_2 = \begin{cases} 0 & \text{if } E_{energy} < E_{energyBound} \\ 1 & \text{otherwise} \end{cases}$$

where $E_{energyBound}$ is a user specified parameter.

$E_{collision}$ penalizes self-intersection. We detect self-intersection and calculate the overlapping volumes using a fast approximate method. We first voxelize the articulated rigid body using a fine grid (the typical grid resolution is about 100^3). If a cell is inside a body link, we increment the counter for that cell. At each time step, we sum up all the cells with counter number larger than one and multiply by the cell volume to approximate the overlapping volume.

$$E_{collision} = \int_0^{t_f} V_{overlap} dt$$

where w_3 is chosen to be 500 for all the examples.

3.3.3 Optimization

Solving the above optimization presents unique challenges. First, we do not have good initial guesses for different swimming patterns of a wide variety of animals with distinctive morphologies. Second, evaluating gradient is expensive because each evaluation needs to simulate the fluid. Last but not least, our objective function is discontinuous and prone to local minima due to sub-optimal swimming gaits, collision penalties, and the modulation of the energy penalty term. For the above reasons, it is difficult to apply traditional continuous optimization algorithms that perform gradient-guided "local" search near the initial guesses. To solve these problems, we perform gait optimization using Covariance Matrix Adaptation (CMA). CMA is a stochastic sample-based optimization algorithm. It does not rely on good initializations and it does not need to compute gradient. More importantly, CMA is a "global" search algorithm that can explore multiple local minima. Although there is no guarantee that CMA will converge at the global minimum, in practice, we observe that it often finds good local minima in moderately high dimensional control spaces (e.g. 20-30 dimensions). I believe that other sample-based optimization algorithms, such as Particle Swarm Optimization [72], may also work well for this problem, although they are not tested in our implementation.

CMA is based on evaluating the objective function for a given population of samples over the parameter space (in our case the joint trajectories). Some fraction of the best samples are then used to update the mean and a covariance matrix that determines the distribution of samples that are evaluated in the next generation. More details of the CMA method can be found in [53].

For each CMA sample, if it violates the user specified joint limits we simply discard it and select another sample. Because the joint limit test takes very little computation time, discarding infeasible samples at this stage is more "economical" than investing major computation effort on them but assigning them a near-zero weight at the end. Once a sample is accepted, we simulate the motion by applying the sampled swimming gait and evaluate the resulting motion using the objective function. To speed up CMA for solving such high-dimensional problems, we include two heuristics in our implementation for some examples.

First, we utilize symmetry for some of our articulated figures: When a creature's body shape is symmetric, often its gait is also symmetric. In such cases, half of the optimization variables are enough to characterize the gait of the whole body because we mirror them to the other half of the body. This assumption is applied to reduce the required computational time, but it is not necessary. Second, for creatures that have more independent appendages, we separate the degrees of freedom in groups and progressively improve the solution by optimizing each group. For example, we assign the forelimbs and hindlimbs of a frog into two separate groups. During the optimization, we first search for the swimming gait for the hindlimbs while freezing the motions of the forelimbs. We then search for the swimming gait of the forelimbs with the optimal hindlimb motions that we already found.

3.4 Path Following

In addition to forward thrust, aquatic creatures also employ very efficient turning maneuvers, such as pitching up and down or turning left and right. The optimization technique described in Chapter 3.3 can be modified to learn various maneuvers. Once the aquatic creature builds a repertoire of swimming maneuvers, we can combine different maneuvers to achieve a high-level task such as path following.

First, we add another term to $E_{distance}$ in (10) to maximize the turning angle towards the desired direction:

$$E_{distance} = \mathbf{d}^T(\Delta\mathbf{p}) + \mathbf{r}^T(\Delta\alpha)$$

where $\mathbf{r}$ is the desired axis of rotation. We set the desired swimming direction $\mathbf{d}$ half way from the current facing direction towards the turning direction. We also change $E_{deviation}$

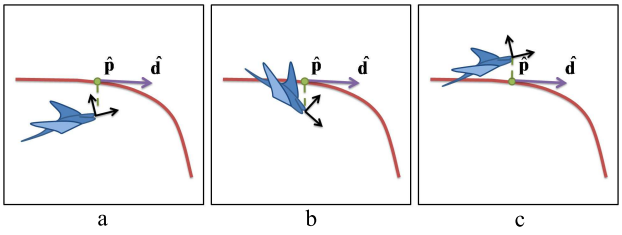

Figure 5: Three different situations that determine if the creature chooses a "swim straight", "pitch up" or "pitch down" maneuver.

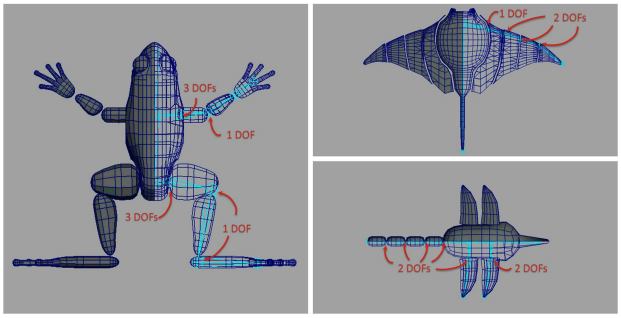

Figure 6: The joint configurations of the frog, the manta ray and the alien.

to penalize the undesired orientation changes.

$$E_{deviation} = ||\Delta\mathbf{p} - \mathbf{d}^T(\Delta\mathbf{p})\mathbf{d}|| + ||\Delta\alpha - \mathbf{r}^T(\Delta\alpha)\mathbf{r}||$$

We solve the above optimization using the CMA method in the same way as described in Chapter 3.3.3.

Once different maneuvers have been learned, we apply a simple heuristic to decide which maneuver to choose to follow the path. At the beginning of each cycle of the motion, we find the nearest point $\mathbf{p}$ on the path to the root of the articulated figure and transform $\mathbf{p}$ and its tangential direction $\mathbf{d}$ to the root coordinate system. We denote the transformed position and direction as $\hat{\mathbf{p}}$ and $\hat{\mathbf{d}}$. Without loss of generality, let's consider a one dimensional example. $\hat{p}_z$ is the z-component of $\hat{\mathbf{p}}$, which means the point is above or beneath the root of the articulated figure. Similarly, $\hat{d}_z$ indicates the path is going upwards or downwards relative to the root orientation. We choose the different maneuvers based on the following rules.

$$\text{Maneuver} = \begin{cases} \text{Go Straight} & \text{if } (\hat{p}_z \geq \epsilon \text{ and } \hat{d}_z \leq -\epsilon) \\ & \text{or } (\hat{p}_z \leq -\epsilon \text{ and } \hat{d}_z \geq \epsilon) \\ \text{Pitch Up} & \text{if } \hat{p}_z > \epsilon \text{ and } \hat{d}_z > \epsilon \\ \text{Pitch Down} & \text{if } \hat{p}_z < -\epsilon \text{ and } \hat{d}_z < -\epsilon \end{cases}$$

where ϵ is a small positive value to prevent the articulated figure from repeatedly choosing alternating turning maneuvers due to small deviations from the path. The first case indicates that the nearest point on the path is above/below the articulated figure while the

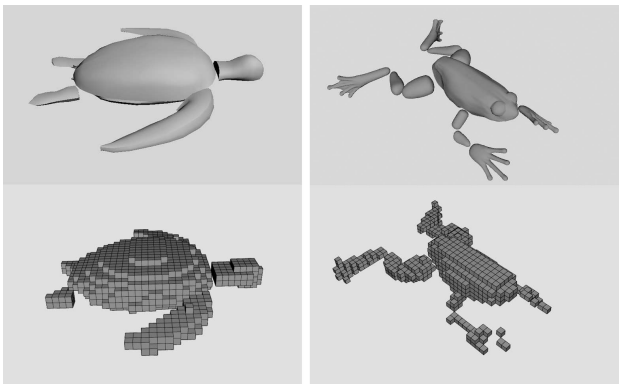

Figure 7: The voxelized representations of the turtle and the frog. The input shapes of the articulated creatures are represented by water-tight polygon meshes. We voxelize these body shapes onto the simulation grid each time step to simulate the two-way coupling between the fluid and the creature.

direction of the path is going downwards/upwards. In other words, the articulated figure is swimming towards the path (Figure 5a). We choose the action "swim straight" in this situation. On the other hand, the second and third cases indicate the articulated figure is swimming away from the path (Figure 5b and 5c) and we choose "pitch up" or "pitch down" accordingly.

3.5 Results

We implemented our method using C++ and ran CMA on a cluster with a maximum of 100 iterations and with a population size of 16 for 2D and 31 for 3D examples. Each CMA sample evaluates the objective function by simulating two cycles of swimming motions. The optimization took from several hours to two days, depending on the model and the grid resolution. After we found the swimming gait, we ran the swimming simulation on a 2.26GHz CPU with a single core. All of the data for our swimming examples are summarized in Table 1. In most of the cases, we use two optimization variables, amplitude and phase, for each degree of freedom. We set the period to one second and the offset to zero. When training the turning gaits, we included the offset in the optimization variables. For the accordian example, the degrees of freedom are interdependent and there is no phase shift among the different degrees of freedom. Thus one optimization variable is enough to characterize its motion. We also exploited the strong symmetry in geometry for some

Table 1: Parameters and performance of examples. Num DOFs is the number of degrees of freedom for the articulated rigid body. Opt Dims is the number of optimization variables. Sim Res is the grid resolution for the simulation and Sim Time is the average simulation time per frame.

Examples	Num DOFs	Opt Dims	Sim Res	Sim Time
accordian	10	1	120×80	1.37s
eel(2D)	4	8	128×64	0.64s
turtle(2D)	4	4	64×64	0.34s
fish	3	6	$64 \times 32 \times 32$	1.45s
eel(3D)	6	12	$64 \times 32 \times 32$	1.31s
manta-ray	14	21	$64 \times 32 \times 32$	10.92s
turtle(3D)	10	10	$64 \times 32 \times 32$	11.29s
frog	18	18	$96 \times 64 \times 48$	12.79s
alien	16	24	$96 \times 36 \times 24$	10.75s

creatures, such as turtles and frogs, to halve the optimization dimensions. We illustrate the joint configurations for some creatures in Figure 6 and the voxelized representations of creatures in Figure 7.

In our implementation, we made three simplifications to reduce the simulation cost. 1) Instead of using a large computational domain to cover the whole space that the creature might swim to, we use a smaller domain that is about two to four times larger in each dimension than the creature's bounding box. This domain moves with the creature when the creature approaches a boundary. 2) At the boundary of the computional domain, we impose the Dirichlet boundary condition $p = 0$ so that the fluid outside the domain is free to flow in and vice versa. 3) Since the density of most aquatic creatures is similar to that of the fluid, we ignore the force of gravity in our simulator.

We describe the results of our swimming optimization method. Please see the video[1] to observe the swimming animations. To visualize the fluid flow, we draw *particles traces*, which show the trajectory of massless particles inside the fluid in a short period of time (15 frames). We modulate the transparency of the particle traces in 3D examples according to the magnitude of the vorticity in order to focus attention on the visually interesting regions of the flow. In 2D examples, we colored the traces to indicate the directions of the vortices.

[1] https://dl.dropboxusercontent.com/u/36899427/articulatedswimmingcreatures.mp4

Figure 8: A four-link clown fish swims. Carangiform swimmers like this flex the front of their body a little, with the majority of the motion near the tail. Note that this fish sheds two separate trails of vortices.

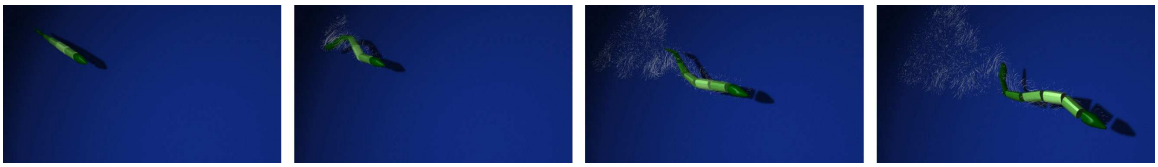

Figure 9: A swimming seven-link eel. Anguiliform swimmers undulate their whole body as if a wave is travelling from head to tail, and shed two separate trails of vortices from the tail.

There are many body shapes and styles of locomotion for fish, and our first set of results investigates several of these. Figure 8 shows a four-segment model of a fish, modelled after the body shape of the clownfish. We used CMA to optimize for efficient forward motion, and snapshots of the resulting motion are given in the figure. Note that the forward body flexes just a little, with the majority of the motion near the tail, which is in good agreement for the style of motion known as the carangiform mode [93]. Using the same objective function, we optimized a seven-segment figure that was designed to mimic an eel body. Figure 9 shows that the resulting motion is that of a travelling wave along the body of the creature, as is typical of real eel swimming (anguiliform mode). Note that the wake of our eel has two separate trails of vortices that are shed from the tip of the tail, as has been observed in lab studies of eels [163]. We show in Figure 10 that a different wake structure appears when an eel swims in a 2D fluid environment, that of a single vortex street. The difference of the wake structure between the 2D and 3D simulations agrees with Kern and Koumoutsakos's study of eels [73].

Our final example of fish motion is that of a manta ray. The manta has a body that is thin in the vertical direction and that has large pectoral fins that extend in the horizontal direction. It swims by slow flapping strokes of these wing-like pectoral fins (the rajiform

swimming mode), somewhat like a slow-motion version of a bird flapping its wings. Although the manta ray does not seem to be a good candidate to be modelled as an articulated figure, we wanted to see how far the articulated models could be pushed. We modelled the ray's pectoral fins as four rows of thin plates that are connected to one another near the leading edge of the fin. The resulting swimming motion from the optimization procedure exceeded our expectations, producing the same graceful flapping that these creatures use to swim (see Figure 11).

We tested our path following approach using the manta ray model. We used our optimization method to find efficient swimming for forward motion, an upward turn and a downward turn. We then gave the manta ray a vertical S-shaped path to follow using our path following controller. The simulated ray was able to follow the path quite closely, as the composite image in Figure 13 shows. Note that this path following motion was created with a single simulation, based on gait switching between the three learned basic motions. We also tested the path following algorithm using a simple 2D turtle model. We show that the turtle cannot swim straight without using the path following technique due to the accumulation of numerical errors. When the path following technique is applied, the turtle actively adjusts its swimming motions according to its position and orientation and successfully swims straight.

Figure 12 shows the motion of a sea turtle that was created using optimization. Adult sea turtles are underwater fliers, moving themselves forward with a flapping motion of their

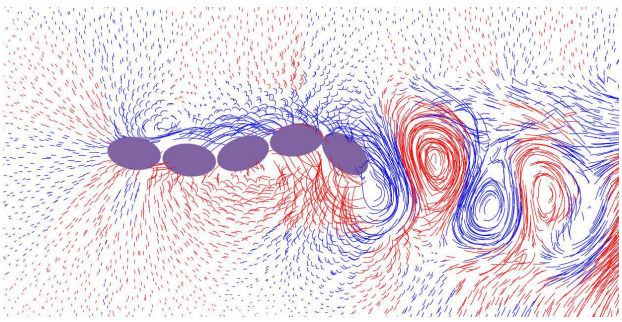

Figure 10: A five-link eel swims in a 2D fluid environment. In contrast to the simulation in 3D, an eel swimming in 2D fluid sheds only one single vortex street. Red traces show the counter-clockwise vortices while blue traces show the clockwise vortices.

Figure 11: A manta ray swimming forward. Rajiform swimmers swim by slow flapping strokes like a slow-motion version of a bird flapping its wings.

Figure 12: A turtle swims in water with a flapping motion of its two front flippers.

two front flippers that is called a *powerstroke* [177]. Note that our turtle results show the characteristic rotating of the front flippers during the upward stroke. Figure 14 shows the results of our swimmer optimization for a model frog. As with real frogs, the large rear legs provide the forward thrust using a classic frog kick. Note that the frog uses its forelimbs with a small range of motion. We think this is because the contribution from the arms is small relative to the contribution from the legs. Based on our observation, some real frogs do not use their forelimbs much when swimming.

In the accompanying video, we also demonstrates that articulated figures can differ dramatically in their swimming motion depending on whether the simulated fluid is a simple

Figure 13: A manta ray follows an S-shaped path by choosing maneuvers from "swimming straight", "pitch up" and "pitch down". The red curve is the path specified by the user.

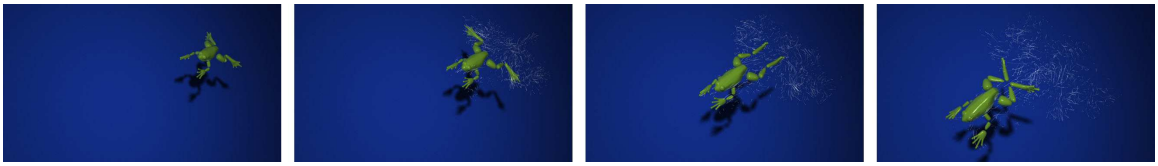

Figure 14: A frog mainly relies on its large rear legs to provide forward thrust in the water.

Figure 15: An alien aquatic creature that swims in water by undulating its tails and flapping its wings. Note the two pairs of wings are slightly out of phase to mimic flapping motion of larger wings.

model or a full Navier-Stokes (NS) solver. Our simple fluid simulator calculates the force as the square of the normal component of the velocity of a moving surface element. This simplified fluid model is identical to that in [175, 90]. We show that swimming in different fluid models leads to different locomotions. Figure 16 shows a 2D swimmer that compresses and relaxes its body in an accordian-like manner moves through the water in the NS fluid but stay in one place in the simple fluid. We demonstrate that the gaits trained in different fluid models differ considerably. The swimming gait for a fish trained in NS fluid smoothly flaps its tail to propel itself forward. When this same fish model is optimized using the simple fluid, the resulting motion is considerably different, gaining thrust mainly from bending at a sharp angle at the middle joint of the body. These differences in motion between a simple fluid and the NS fluid are in agreement with the findings of Lentine et al. [90].

In order to test the generality of our method, we applied our swimming optimization to an articulated figure that has no counterpart in the real world (see Figure 15). This is the swimming version of the task of finding plausible walking motions for a user-created land creature [56, 168]. Our alien creature has two pairs of limbs on the trunk of its body, and in addition has a long and powerful tail. The motion that was found by our optimization combines a whip-like motion of the tail together with coordinated rowing from the pairs of limbs. Although there is no point of comparison in the real world for this creature's motion, the resulting swimming pattern looks entirely plausible.

Although our method requires little prior knowledge about the swimming gait of the creature, there are some parameters that users can change, including the energy bound, the period of the motion for each degree of freedom, and joint limits. This provides users the freedom to achieve different motion, agile or slow, by changing these parameters. In particular, we tuned the period of the swimming gait and the energy bound in our examples. We used a period of one second for all of the examples. We made this choice deliberately because choosing a longer period means longer optimization time (each CMA sample need to simulate two cycles of swimming motions). However, we believe that including the period into the optimization will probably give more interesting results because different periods could make a big difference in the final swimming gait. We leave this as future work. To set the energy bounds, we began by trying out several energy bounds for an initial animal, the fish shown in the leftmost of Figure 1. Once we were satisfied with the results, we then used this as our standard energy bound. For a new creature, we scaled this standard energy bound according to the mass of the new creature relative to the mass of the fish. Users can also change the weights in the objective function. In our examples, we set all these parameters by intuition without much tuning. The weights are reported at the end of each paragraph that introduces the different objectives in Chapter 3.3.2.

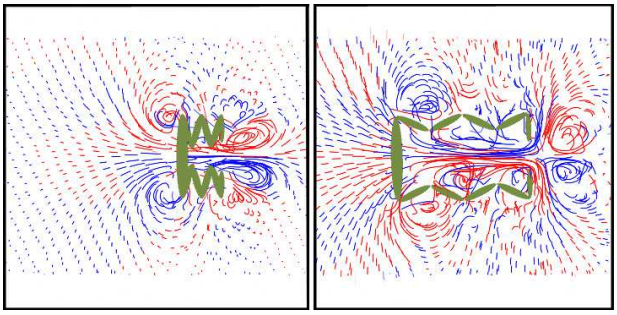

Figure 16: An imaginary creature swims forward by compressing and relaxing its body in an accordion-like manner in a Navier-Stokes fluid model. The images are two snapshots in the animation sequence. This demonstrates that including the Navier-Stokes fluid model is necessary to capture certain swimming patterns, such as jet propulsion, because a simplified fluid model does not allow forward motion for such modes of locomotion.

3.6 Discussion

We have demonstrated that our approach creates natural swimming behavior for a wide variety of animal bodies. For short-bodied fish and eels, our results show vortex trails that are in agreement with laboratory measurements and other published simulation results. For the other creatures, our optimized motions have the same overall appearance of the real-world animals, although lab data is not available. Our articulated body representation of creature anatomy is quite general, even allowing us to animate forms such as the manta ray that are not usually thought of as articulated figures.

Although we have successfully applied our method to various aquatic creatures with disparate body shapes and joint configurations, our approach does have limitations. Our two-way coupling method needs to voxelize the articulated rigid body, and the accuracy for representing the articulated figure depends on the grid resolution. Thin features cannot be captured by the fluid simulator (Figure 7). We believe that incorporating adaptive grids [97] or unstructured meshes [25, 80] can dramatically increase the accuracy of the two-way coupling. Furthermore, the two-way coupling method is tailored for the interaction between fluids and articulated figures. Even though many aquatic creatures have a skeleton and can be represented well by articulated figures, there are exceptions such as jellyfish. Our framework for discovering the optimal swimming gaits and path following is still valid for soft-body creatures, but we would need an efficient two-way coupling mechanism to simulate these swimming motions. We leave this as future work.

We use the sine function to parameterize the joint space. There are quite a few motions that cannot be depicted by a single sine function, such as gliding. One possible way to improve this is to to use a weighted sum of multiple sine functions with different amplitudes, phases and periods [50]. However, this would require more optimization variables and more computational resources to discover a swimming gait. In addition, other simplifications to ease optimization, including enforcing symmetry and hardcoding the swimming frequency to be one hertz, further restrict the motion space to a smaller subset of possible swimming motions employed in nature.

Our simulated swimmers seem to use more energy than the real creatures do because

the simulated water is more viscous than real water. Even though we use the inviscous Navier-Stokes equation (Euler equation) to simulate the fluid, there is numerical viscosity. We chose to use relatively coarse grid, and thus incur large numerical viscosity, to keep the computational time tractable because CMA optimization need to simulate the two-way coupling thousands of times. In addition, while the real aquatic creatures take advantage of their streamline shaped body to reduce the fluid drag, the simulated creatures are voxelized and the resultant stair-step shaped body is not particularly efficient inside the fluid.

There are a number of interesting avenues for future work. There are many ways this approach could be expanded to give more control to animators, including different path following strategies and higher-level behavior control. Our work has concentrated on continuous motion, but many animals have distinctly different movements for situations such as escaping a predator. It would be interesting to investigate these faster, intermittent motions. Swimming at the *surface* of the water could be studied, including motions such as a human swimmer doing the crawl or a whale jumping out of the water (breaching). Finally, taking a cue from the work of Karl Sims, it would be fascinating to simultaneously optimize for both swimming motion and body shape.

CHAPTER IV

LOCOMOTION FOR SOFT BODY CHARACTERS

4.1 Motivation

A large variety of soft body animals live on our planet. Some examples of such creatures are slugs, starfish, earthworms, octopus, and jellyfish. These animals move in interesting ways due to their flexible body shapes and lack of skeleton support. In this chapter we present a method of animating soft body characters, that is, characters that do not have a skeleton. In particular, our emphasis is on creating animations of soft body creature locomotion, including crawling, walking, rolling and jumping. Many hand-drawn animated characters move in such a flexible manner that they seem to be boneless. The animation principle of *squash-and-stretch* can be seen in its purest form with soft body characters. In addition, as exemplified by our own tongues, even animals with skeletons can have body parts that move without the help of bones. Our research is driven by the intellectual challenge of simulating the locomotion of such soft body creatures, without resorting to any form of rigid elements in our models.

There are two key aspects of anatomy that allow real soft-bodied creatures to move: volume preservation and muscle contractions. Our animation system makes use of these same principles. Soft body tissue is volume preserving, due primarily to the incompressible nature of water. This volume preservation puts constraints on the degree of deformation that a soft body may undergo. We use volume-preserving finite elements to match this aspect of soft body tissue. The second important aspect of soft body creatures is that they control the shape of their body by the contraction of muscles. If such a creature shortens only the muscles that run down the right side of its body, this will cause the creature to bend towards the right. Note that volume preservation and muscle contractions often work in concert to produce motion. If a cylindrical creature uses radial muscle contraction to make itself thinner, then the constraint of volume preservation means that at the same time

the creature will stretch lengthwise.

Each of our soft body characters is represented as a tetrahedral mesh and simulated using the finite element method. Our models typically contain hundreds of tetrahedral elements, and controlling such a high degree of freedom model poses a challenge. The aforementioned muscles from real animals provide a way of reducing the degrees of freedom in our characters. In addition to the tet mesh, each of our characters is augmented with a collection of polyline paths, each of which represents a muscle fiber. A character changes its body shape by contracting these muscle fibers, and this induces a shape change in the collection of tetrahedra near the fibers. In theory, such a character could be controlled by specifying the timing of various muscle contractions. However, unlike controlling articulated figures using joint torques, the complex interplay between muscles and soft body shapes makes the control problem exceedingly challenging; even bending a limb of a soft body creature is much more difficult than bending a joint of an articulated figure. For these reasons, we decided that controlling a soft body creature by specifying the changes to each muscle fiber would be a tedious method of control.

Our system provides a collection of intuitive controls for soft body creature motion, such as moving a point of the character to a given position, or regulating the character's linear or angular momentum. With this collection of intuitive controls, we are able to animate a variety of soft body characters, and in particular, we can demonstrate a wide array of locomotion methods. To move a character, we specify a set of high-level goals (possibly time-varying), and these goals are turned into an objective function that is passed to our solver. For each time step, we formulate and solve a constrained optimization problem, and this gives us new muscle lengths. These muscle lengths induce changes in stress that are applied to the tetrahedral elements, and we then use our physics simulator to advance the system forward in time.

An important part of our constraint solver is contact planning, and this proves to be a challenge for soft bodies. At each time step, our solver must be able to predict how a change to the muscle contractions will influence the points of contact between the character and the ground. For articulated figures, most optimization-based controllers assume that each

point of contact is static, which makes contact resolution relatively straightforward to solve. In our system, we cannot assume static contact because sliding and breaking contact turn out to be quite important strategies to control soft characters. For instance, a soft creature may need to widen its base in order to balance, and this means that the points of contact must slide. Depending on the motion goals that are given for a character, the best way to minimize the resulting objective function might be to maintain static contact, to break contact, or to allow sliding contact along the ground. The behavior needs to be decided for each point of contact, and this results in a high dimensional and discontinuous optimization problem. We formulate this as a linear complementarity problem with a quadratic objective function. Although similar problems have been recently proposed in other fields [23, 9], we believe that our solution method is new to graphics.

A natural alternative to our approach would be to represent a character as a rigid, articulated skeleton, and to surround the skeleton with soft tissue that deforms. Such a character representation could even demonstrate elongation and contraction with the use of translational joints. This approach would have several advantages, including the availability of numerous tools that can be used to control an articulated figure. We made a deliberate choice to avoid using rigid elements entirely. We think that using only soft elements will be more likely to result in motions that are more faithful to actual soft body creatures. Our approach avoids the possibility that the character motion shows hints of a hidden skeleton. In addition, using only muscle contractions keeps our character motions "honest" in terms of the magnitude of forces that such characters can apply without a skeleton. Perhaps the most important reason that we avoid the use of rigid elements is our desire to expand the creature body forms that we can simulate using computers. Many animals in nature move without the use of skeletons, so is it possible to create a computer simulation that mirrors this fascinating phenomenon?

4.2 System Overview

We design a wide variety of locomotion controllers for soft bodies, including balance, walking, crawling, jumping, sliding and rolling. Given the geometry of a soft-body creature and

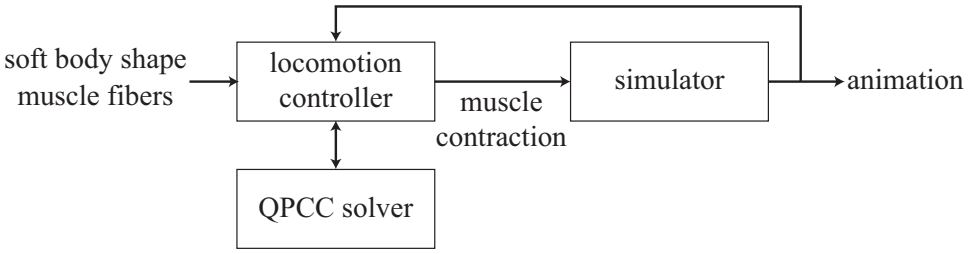

Figure 17: Overview of our system.

the arrangement of its muscle fibers, our controller computes required muscle contraction to propel the creature to achieve desired locomotion while maintaining balance. At each time step, the controller formulates a quadratic program with complementarity constraints (QPCC) to solve for the optimal muscle contraction under discretized dynamic equations of motion and frictional contact constraints. The objective function, assuming a convex quadratic form, can be designed arbitrarily to address control goals of the desired locomotion. The optimal muscle contraction is passed to a FEM simulator to calculate the next state. Figure 26 illustrates the main components of our system.

4.3 Soft Body Simulation and Modeling

Before we introduce the control algorithms for locomotion, we first describe the methods for simulating soft bodies and computing muscle forces.

4.3.1 Finite Element Simulation

A soft body creature is represented as a tetrahedral mesh and is simulated using a modified corotational linear FEM [114]. We chose FEM instead of a mass-spring model because it is difficult to enforce volume preservation for the material with mass-spring systems. At each time step, the state of the creature, $\mathbf{p}$, is computed through numerical integration of the dynamic equations of motion:

$$\mathbf{M}\ddot{\mathbf{p}} = \mathbf{f}_x + \mathbf{f}_e + \mathbf{f}_d + \mathbf{f}_m \tag{11}$$

where $\mathbf{M}$ is the mass matrix of the discretized soft body and $\mathbf{p}$ is the nodal position of the deformed shape. The forces on the right hand side, $\mathbf{f}_x$, $\mathbf{f}_e$, $\mathbf{f}_d$, and $\mathbf{f}_m$, indicate external,

elastic, damping, and muscle forces respectively. The external force $\mathbf{f}_x$ includes gravity, contact force, and user perturbation force.

As notation, when we are specifying a quantity $\mathbf{q}$ for a single element, we will write this as $\hat{\mathbf{q}}$. To compute the elastic force for each element, we adapted the method suggested by Nesme *et al.* [116]:

$$\hat{\mathbf{f}}_e = -\hat{\mathbf{B}}^T \hat{\mathbf{D}} \hat{\mathbf{B}} (\hat{\mathbf{p}} - \hat{\mathbf{R}} \hat{\mathbf{x}}) \tag{12}$$

where $\hat{\mathbf{x}}$ indicates the nodal position in the rest shape and $\hat{\mathbf{R}}$ transforms the element from the reference coordinates to the deformed coordinates. $\hat{\mathbf{B}}$ is the strain-displacement matrix in the deformed coordinates and $\hat{\mathbf{D}}$ is the stress-strain matrix. We use the Poisson ratio 0.45 in $\hat{\mathbf{D}}$ to make the soft body nearly incompressible while avoiding locking artifacts [66]. Although volume preservation is not enforced strictly, our experiments show that the volume change is below 15% and is not visually noticeable. This formulation linearizes the elastic force around the current deformed shape, rather than around the rest shape, as used in most FEM implementations. We chose this formulation because it eliminates "ghost torques" caused by the error of linearization around the rest shape [116]. When the material is soft or when the deformation is small, ghost torques do not cause visible artifacts. However, this formulation is necessary for our case, because soft body locomotion requires large deformation with relatively stiff materials to support the weight of the creature.

We assemble the individual stiffness matrices $\hat{\mathbf{B}}^T \hat{\mathbf{D}} \hat{\mathbf{B}}$ for each element into a large stiffness matrix $\mathbf{K}$ for the whole system. The elastic force for all the FEM nodes can be expressed by $\mathbf{f}_e = -\mathbf{K}(\mathbf{p} - \mathbf{R}\mathbf{x})$. For damping force, we use simple Rayleigh damping model to compute its effect: $\mathbf{f}_d = -\mathbf{C}\dot{\mathbf{p}} = -(\mu\mathbf{M} + \lambda\mathbf{K})\dot{\mathbf{p}}$. We set $\mu = 0$ and $\lambda = 0.2$.

4.3.2 Muscle Modeling

We model muscle fibers as polygonal curves with a small number of segments. Each muscle segment can contract along its current direction, but it cannot extend or bend. Based on the arrangement of muscle fibers, we can bundle them into muscle groups. There are three types of muscles which lead to different control tasks. The *longitudinal muscles* are linear muscles that extend from one end of the body to the other end. Their main function is

to shorten or bend the body. The *radial muscles* are a set of short muscles that span a cross-section of the body. When radial muscles contract, the volume-preserving nature of the tissue causes the body to elongate. *Helical muscles* wrap around the body in a helical shape. When a helical muscle contracts, the body twists.

Muscle contraction induces muscle force on nearby FEM elements. Each muscle segment is modeled as a spring with a changeable desired length. The spring force caused by each segment is computed as: $f = k(l_d - l)$, where k is the stiffness of the muscle fiber, l_d is the current desired length, and l is the current length of the segment. We treat f as a virtual force, which is realized by muscle stress imposed on nearby elements. When a muscle segment contracts with the virtual force f in the direction of $\mathbf{d}$ in the reference coordinates, the effect of contraction is as muscle stress $\mathbf{F}$:

$$\mathbf{F} = \mathbf{U} \begin{bmatrix} f & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{U}^T \tag{13}$$

where $\mathbf{U}$ is the matrix that rotates vector $(1, 0, 0)^T$ in the reference coordinates to be aligned with $\mathbf{d}$.

Each FEM element may be affected by multiple muscles. The accumulated muscle stress experienced by an element i is a weighted sum of all the muscle stresses that have an influence on the element i, denoted in the deformed coordinates of element i as:

$$\hat{\sigma}_m^i = \sum_j w_{ij} \hat{\mathbf{R}} \mathbf{F}_j \hat{\mathbf{R}}^T \tag{14}$$

where w_{ij} weighs the influence of muscle fiber j on the element i. The value of w_{ij} is based on the shortest distance d_{ij}, from the muscle fiber to the center of the element in the reference coordinates. We use a Gaussian kernel as the attenuation function, $h(d) = \exp(\frac{d^2}{\sigma^2})$, where σ is the variance of the Gaussian function. The influence weight w_{ij} is defined as

$$w_{ij} = \frac{h(d_{ij})}{\sum_{k \in group(j)} h(d_{ik})} \tag{15}$$

The denominator in eq. (15) normalizes the influence of muscle fibers within the same group. This normalization allows different muscle groups, typically with different functionalities, to exert their influence on the soft body simultaneously.

Once we compute the muscle stress for each element $\hat{\sigma}_m$, we calculate the force at each face of the element by multiplying $\hat{\sigma}_m$ with the area-weighted face normal in the deformed coordinates. Finally, we evenly distribute the force at each face to the vertices to obtain $\mathbf{f}_m$ at each node. As a shorthand, we define a muscle force matrix $\mathbf{A}$ to express the relation between the muscle force on each node and the effect of muscle contraction.

$$\mathbf{f}_m = \mathbf{A}(\mathbf{l}_d - \mathbf{l}) \tag{16}$$

Note that $\mathbf{A} \in \mathbb{R}^{3n \times m}$ where n is the number of nodes in the FEM mesh and m is the number of muscle segments, which is also the dimension of our control variables.

4.3.3 Numerical Integration

To ensure the stability of our system with large time steps, we use an implicit integrator to solve the dynamic equations. After substituting each force terms into eq. (11), we arrive at the following dynamic equation:

$$\mathbf{M}\ddot{\mathbf{p}} = \mathbf{f}_x - \mathbf{K}(\mathbf{p} - \mathbf{R}\mathbf{x}) - \mathbf{C}\dot{\mathbf{p}} + \mathbf{A}(\mathbf{l}_d - \mathbf{l}) \tag{17}$$

Applying the implicit integrator, we can rewrite the equation as,

$$\begin{aligned} \widetilde{\mathbf{M}}\dot{\mathbf{p}}^{n+1} &= \mathbf{M}\dot{\mathbf{p}}^n + \Delta t(\mathbf{f}_x^n - \mathbf{K}(\mathbf{p}^n - \mathbf{R}\mathbf{x}^n) + \mathbf{A}(\mathbf{l}_d - \mathbf{l}^n)) \\ &= \widetilde{\mathbf{f}}^n + \mathbf{f}_c + \widetilde{\mathbf{A}}\mathbf{l}_d \qquad (18) \\ \mathbf{p}^{n+1} &= \mathbf{p}^n + \Delta t\dot{\mathbf{p}}^{n+1} \qquad (19) \end{aligned}$$

where superscript n indicates the discretized time index and Δt is the time step. We define $\widetilde{\mathbf{M}} = \mathbf{M} + \Delta t\mathbf{C} + \Delta t^2\mathbf{K}$ and $\widetilde{\mathbf{A}} = \Delta t\mathbf{A}$, and single out the contact force as $\mathbf{f}_c$, which is part of $\mathbf{f}_x$ and will be discussed in the next section. $\widetilde{\mathbf{f}}^n$ accounts for the remaining terms on the right hand side of eq. (18).

4.4 Locomotion Control

To create functional locomotion using the simulation framework described in previous section, we need a control algorithm to compute the appropriate muscle contractions. Our control algorithm formulates an optimization at each time step to solve for the desired muscle contraction $\mathbf{l}_d$ that achieves the control goals subject to physical constraints.

4.4.1 Optimization

We express the objective function in the optimization as a convex quadratic function of the next state of the soft body: $G(\mathbf{p}^{n+1})$. This general function form is sufficient to encode a wide variety of control goals while retaining convexity of the optimization. Using eq. (18) and (19), the optimization minimizes a reparameterized objective function which implicitly enforces the equations of motion:

$$\min_{\mathbf{l}_d} \quad G(\mathbf{p}^n + \Delta t \widetilde{\mathbf{M}}^{-1}(\widetilde{\mathbf{f}}^n + \mathbf{f}_c + \widetilde{\mathbf{A}}\mathbf{l}_d)) \tag{20}$$

In addition to the objective function, the optimization must satisfy two constraints. The first constraint enforces the range of muscle contractions: $0.5\mathbf{l}_0 \leq \mathbf{l}_d \leq \mathbf{l}_0$, where $\mathbf{l}_0$ denotes the muscle length at the rest pose. The second constraint enforces valid contact under Coulomb's friction model. We adapt an implicit time-stepping LCP method to regulate contact velocity and contact force, $\mathbf{f}_c = \mathbf{N}f_\perp + \mathbf{D}\mathbf{f}_\parallel$, where $\mathbf{N}$ is the unit normal vector, $\mathbf{D}$ is a set of tangential directions at the contact point, and $f_\perp$ and $\mathbf{f}_\parallel$ are the magnitudes of normal and tangent forces. The optimization with constraints can be written as

$$\min_{\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel, \lambda} G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) \tag{21}$$

subject to

$$0.5\mathbf{l}_0 \leq \mathbf{l}_d \leq \mathbf{l}_0$$

$$\mathbf{0} \leq \begin{bmatrix} \mathbf{f}_\perp \\ \mathbf{f}_\parallel \\ \lambda \end{bmatrix} \perp \begin{bmatrix} \mathbf{N}^T\dot{\mathbf{p}}^{n+1} \\ \mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda \\ \mu\mathbf{f}_\perp - \mathbf{E}^T\mathbf{f}_\parallel \end{bmatrix} \geq \mathbf{0}$$

where μ is the friction coefficient and $\mathbf{E}$ is a block-diagonal matrix of $\mathbf{e}$, which is a vector of ones. The complementarity constraints also introduce auxiliary variables λ. The physical meaning of λ is related to the tangent velocity of a sliding contact. Please see Anitescu and Portra [7] for a complete review of LCP formulation.

A quadratic program with linear complementarity constraints (QPCC) is well known for its nonconvexity and disjunctive features, which cannot be solved efficiently by standard nonconvex solvers. Previous work simplified this problem by assuming that the current

contacts will remain static (the velocities of contact points remain zero) at the next time step. If this assumption is not consistent with the simulated result, the controller will try to correct it at the next time step. In soft body control, assuming static contacts is too restrictive and significantly reduces the effectiveness of the controller. As a result, we cannot drop the complementarity constraints in the optimization. We will introduce a new iterative solver to QPCC for contact modeling in Chapter 4.5.

4.4.2 Low-level Controllers

We develop three types of low-level control mechanisms by formulating different objective functions $G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel)$ in eq. (21). In Chapter 4.6, we demonstrate that these three basic mechanisms can be combined to design fundamentally different locomotion controllers.

Momentum control. Regulating momentum is of paramount importance for biped balance and locomotion. Previous work [99] has demonstrated that controlling the linear momentum relative to the contact support is a simple but very effective balance strategy. We use the following objective function to regulate the linear momentum, $\mathbf{L}$.

$$G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) = \|\dot{\mathbf{L}}(\dot{\mathbf{p}}^{n+1}, \dot{\mathbf{p}}^n) - \bar{\dot{\mathbf{L}}}\|^2 \tag{22}$$

The desired change of linear momentum $\bar{\dot{\mathbf{L}}}$ is defined as

$$\bar{\dot{\mathbf{L}}} = mK_p(\bar{\mathbf{c}} - \mathbf{c}^n) - K_d\mathbf{L}^n \tag{23}$$

where m is the mass of the creature and $\mathbf{c}$ is the center of mass (COM) position. K_p and K_d are the stiffness and damping coefficients for the feedback control. For balance control, the desired COM position $\bar{\mathbf{c}}$ is computed based on the center of the contact support area.

Angular momentum also plays an important role in balance. For soft body creatures, controlling angular momentum is also essential to rolling motion.

$$G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) = \|\dot{\mathbf{H}}(\dot{\mathbf{p}}^{n+1}, \dot{\mathbf{p}}^n, \mathbf{p}^n) - \bar{\dot{\mathbf{H}}}\|^2 \tag{24}$$

where $\bar{\dot{\mathbf{H}}}$ denotes the target value for the change of angular momentum and $\dot{\mathbf{H}}$ computes the change of angular momentum at the next state.

Base control. In addition to controlling the momentum, we can increase the contact area to provide a wider range of support to the COM. This balance strategy is particularly interesting for soft bodies. By squashing and stretching its entire body, a soft body creature can adjust its base area at will to maintain balance. We define an objective function that controls the projected base area A by matching its change rate to a desired rate $\bar{\dot{A}}$:

$$G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) = \|\dot{A}(\dot{\mathbf{p}}^{n+1}, \mathbf{p}^n) - \bar{\dot{A}}\|^2 \tag{25}$$

We compute A by projecting a defined base area to the ground surface with normal vector $\mathbf{n}$:

$$A = \frac{1}{2} \sum_i \left((b_i - a_i) \times (c_i - a_i)\right)^T \mathbf{n} \tag{26}$$

where index i loops over all triangles in the base area, and a_i, b_i, and c_i are the vertices of the ith triangle. When computing $\dot{A}$, we evaluate the velocity terms at $\dot{\mathbf{p}}^{n+1}$ and the position terms at $\mathbf{p}^n$. This approximation has negligible effect on accuracy, but keeps our objective function convex.

Position and velocity tracking. Direct control of a Cartesian position or velocity is also an effective way to regulate locomotion. For example, tracking the trajectory of a foot is essential for producing a walking gait. The following objective function minimizes the distance between a particular body point at the next time step and a target Cartesian point $\bar{\mathbf{p}}$

$$G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) = \|f(\mathbf{p}^{n+1}) - \bar{\mathbf{p}}\|^2 \tag{27}$$

where f is a function that selects a node from $\mathbf{p}^{n+1}$. If we redefine f as a function that computes the COM, eq. (27) can be used to track the COM. Likewise, we can track the relative position of two body points by replacing $f(\mathbf{p}^{n+1})$ with $f_1(\mathbf{p}^{n+1}) - f_2(\mathbf{p}^{n+1})$, where f_1 selects the first node and f_2 selects the second node from $\mathbf{p}^{n+1}$. We can also use a similar objective function to track velocity.

$$G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) = \|f(\dot{\mathbf{p}}^{n+1}) - \bar{\dot{\mathbf{p}}}\|^2 \tag{28}$$

4.5 QPCC for contact modeling

The control framework described in Chapter 4.4 requires an efficient QPCC solver that can handle 50 to 100 complementarity variables. Solving QPCC in general is difficult due to the presence of the linear complementarity constraints. A naïve way to solve QPCC is to evaluate all the valid combinations of the complementarity constraints and output the minimizer. This exhaustive method is guaranteed to find the global minimum. However, the computational time grows exponentially with the number of variables involved in the complementarity constraint. In our control problem, we have 10 variables for each contact point (one for $f_\perp$, eight for $\mathbf{f}_\parallel$ and one for λ). Thus, a few contact points alone will render the exhaustive method computational impractical.

We propose a more efficient way to solve a QPCC for contact problems, such as eq. (21). Our iterative QPCC solver starts with an initial guess, which is a set of linear constraints that are compatible with the complementarity conditions. For the initial guess, we manually set some elements of $\mathbf{f}_\perp$, $\mathbf{f}_\parallel$, and λ to be zero and their complementary pairs to be nonnegative, in a way that the state of those variables has a physical meaning. For example, if we assume all contact points are static, we arrive at the following convex QP:

$$\min_{\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel, \lambda} G(\mathbf{l}_d, \mathbf{f}_\perp, \mathbf{f}_\parallel) \tag{29}$$

subject to

$$0.5\mathbf{l}_0 \leq \mathbf{l}_d \leq \mathbf{l}_0$$

$$\mathbf{0} \leq \mathbf{f}_\perp, \qquad \mathbf{N}^T \dot{\mathbf{p}}^{n+1} = \mathbf{0} \tag{30}$$

$$\mathbf{0} \leq \mathbf{f}_\parallel, \qquad \mathbf{D}^T \dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda = \mathbf{0} \tag{31}$$

$$\mathbf{0} = \lambda, \qquad \mu \mathbf{f}_\perp - \mathbf{E}^T \mathbf{f}_\parallel \geq \mathbf{0} \tag{32}$$

After solving the above QP, we examine the complementarity conditions at the minimizer. We identify those inequality constraints that reach their boundary at the minimizer as candidates for pivoting. Our algorithm pivots one of those candidates at a time. That is, we set the candidate to equality constraint and flip its complementary counterpart from

equality to inequality. By pivoting the complementarity constraints, we formulate a new QP with a set of different linear constraints and we solve for the minimizer for this new QP. We repeat this process until all the candidates reach the local minimum of the QPCC, i.e. until we encounter a minimizer that lies in the interior of the feasible region. The candidate that yields the best local minimum is returned as the solution of the QPCC. Our QPCC solver explores the nonconvex feasible region based on the following heuristics: Each pair of the complementarity constraints defines a feasible region formed by two intersecting half-hyperplanes. If a minimizer hits the boundary of the half-hyperplane, exploring the other half-hyperplane might give a better minimizer. Figure 18 shows a two dimensional example.

Our algorithm further exploits the structure of a contact problem to improve the performance. Instead of arbitrarily selecting a candidate to pivot, we can group the complementarity constraints according to their physical meaning and pivot a whole group together. There are three different situations for each contact point: static, sliding and contact breakage. Pivoting constraints in eq. (30) indicates a switch between a static (or a sliding) contact and contact breakage. Pivoting constraints in eq. (31) and (32) indicates a switch between static and sliding contact. For example, if $\mathbf{f}_\perp(i) = 0$ is the result of solving the QP

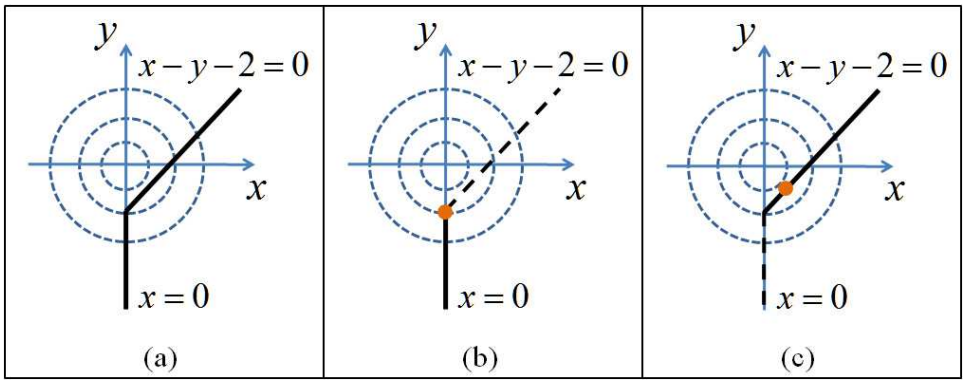

Figure 18: A simple 2D QPCC example. The complementarity constraints are $0 \leq x \perp x - y - 2 \geq 0$. (a) The feasible region lies in two intersecting half-hyperplanes, shown as two black line segments. (b) With the initial guess of $x = 0$ and $x - y - 2 \geq 0$, the minimizer, shown as an orange dot, is located at the boundary of the inequality constraint. (c) After pivoting the constraint, setting $x \geq 0$ and $x - y - 2 = 0$, we find a better minimizer (global minimizer in this simple case).

(eq. (29)), it implies that breaking ith contact point might lead to a better minimizer for the QPCC. The solver will pivot the corresponding constraints: $\mathbf{f}_\perp(i) \geq 0 \rightarrow \mathbf{f}_\perp(i) = 0$ and $(\mathbf{N}^T\dot{\mathbf{p}}^{n+1})(i) = 0 \rightarrow (\mathbf{N}^T\dot{\mathbf{p}}^{n+1})(i) \geq 0$. The new QP will be solved subsequently. Conversely, when a free point restores a static contact, we apply the opposite pivoting.

If a friction cone condition (eq. (32)) for the ith contact point needs to be pivoted, this implies that the ith contact point is about to slide and switching it from static to sliding might lead to a better minimizer. The solver then changes the inequality constraint $(\mu\mathbf{f}_\perp - \mathbf{E}^T\mathbf{f}_\parallel)(i) \geq 0$ to equality and changes the corresponding equality constraint $\lambda(i) = 0$ to inequality. In addition, we need to pivot some constraints in eq. (31) to specify the direction of the sliding contact, which can be estimated using the static friction from the current minimizer. We project this static friction force to each of the tangential direction of the ith contact point $\mathbf{D}(i)$ and find the two directions (the mth and nth direction in $\mathbf{D}(i)$) that have the largest magnitude. The sliding force direction is estimated to be along the convex combination of mth and nth directions. We pivot the constraints in the following way,

$$\begin{aligned}
&\mathbf{f}_\parallel(i,m) \geq 0, \quad (\mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda)(i,m) = 0 \\
&\mathbf{f}_\parallel(i,n) \geq 0, \quad (\mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda)(i,n) = 0 \\
&\mathbf{f}_\parallel(i,j) = 0, \quad (\mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda)(i,j) \geq 0, \quad \forall j \neq m,n
\end{aligned}$$

where $\mathbf{f}_\parallel(i,j)$ is the magnitude of the friction force along the jth direction for the ith contact point[1]. For the special case, where the friction force is exactly along the mth (or nth) direction, we only pivot the complementarity constraints involving the mth (or nth) direction. For a switch between sliding to static contact, we use the same pivoting mechanism but pivot the constraints the opposite way.

Instead of searching exhaustively in the feasible region of the QPCC, our solver systematically explores the feasible region based on the above mentioned heuristics. Although the objective value is not guaranteed to decrease monotonically, our experiments show that the

[1] $\mathbf{f}_\parallel(i,j)$ is actually the $(i \cdot N + j)$th element of $\mathbf{f}_\parallel$ assuming that N tangential directions are used for the linearized friction cone for each contact point.

objective value decreases drastically within a small number of iterations. The minimizer found by our solver is, in all the experiments, significantly closer to optimal than the one solved under the static contact assumption. We report the results of the experiments in Chapter 4.7.

Implementation. Our solver requires a feasible initial guess. We can use the trivial solution under static contact assumption (eq. (29)) as an initial guess, or the solution from the previous QPCC when it is available (warm start). Occasionally, they might result in an infeasible QP. For those cases, we assume the same muscle activation as in last time step, remove the objective function and muscle length constraints from eq. (21) and solve a pure LCP. The contact situation from the LCP solution is then used as the initial guess for the QPCC.

We implement the QPCC solver using a graph expansion algorithm (See Appendix A for details). Each QP with linear constraints is a node in the graph. We visit each node twice, starting from the initial guess as the root. In the first visit, we solve the QP, assign the objective value to the node, and store the set of candidates to be pivoted. After first visit, we push the node into a priority queue based on its objective function value. A node is visited the second time when it is at the top of the queue. In the second visit, we pivot the constraints from its candidate set. Each pivot generates a child node. We discard the child node if it already exists in the current graph. If the child node is new, we visit the node for the first time and push it into the queue. The second visit is completed when all the constraints in the candidate set are pivoted. We then pop the next node in the queue and repeat this process. The algorithm terminates when the priority queue is empty or the number of visited nodes exceeds a threshold. The final solution is the best minimizer found so far by the QPCC solver.

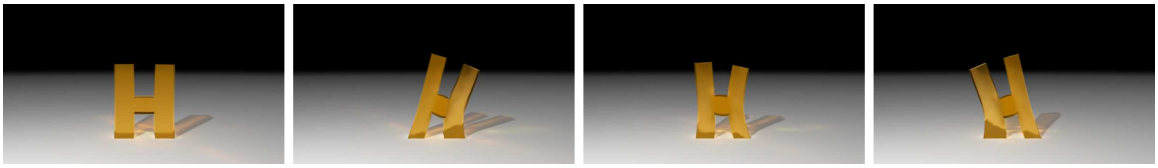

Figure 19: An H-shaped soft body character does its morning exercises by swinging its body from one side to the other.

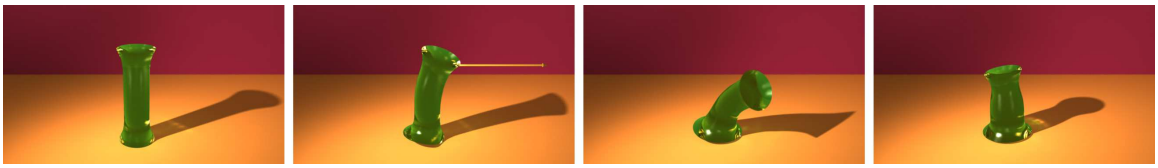

Figure 20: An I-shaped soft body character tries to maintain balance under perturbation by regulating its momenta, widening its base and lowering its center of mass.

4.6 Results

In this section we describe the results of our soft body locomotion controllers. Please see the video[2] to watch the locomotion animations. Our system is implemented in C++, and we generated the tetrahedral mesh for FEM simulation using TETGEN [134]. We used the GPU to create layered depth figures for collision detection, and we used contact patches (multi-resolution volume contact) [3] instead of points as the contact primitives. For each contact patch, we use eight tangential directions to linearize the friction cone, which provides sufficient accuracy while keeping the QPCC tractable. The examples were run on a workstation with a 2.26GHz CPU and 4GB of memory. All the data of our locomotion examples are summarized in Table 2.

We design many different shapes of the soft body characters, all of which are chosen from the English alphabet. Figure 19 shows an H-shaped character doing morning exercises. The character is designed with four longitudinal muscles and one radial muscle for each leg (Figure 25a). It is animated by specifying the trajectory of the desired center of mass (COM), which is moved left/right by a sine function. We use the position and velocity tracking controller from Chapter 4.4.2 to track the desired COM. Note that when the "H" swings left, its right side elongates and gets thinner while the left side shortens and becomes

[2] http://dl.dropbox.com/u/36899427/softbodylocomotion.mp4

fatter due to the volume preservation. This animation clearly exemplifies the principle of *squash and stretch*.

Balance. We design an I-shaped character (Figure 20) to demonstrate static balance. We gave the character four longitudinal muscles that allow it to bend in any direction. This character is perturbed by a large force exerting at its head, and it attempts to recover its balance. Static balance of the "I" turns out to be one of the most difficult task among all our examples. The geometry of the letter "I" does not have limbs or other appendages to help it regulate the linear and angular momentum. The squishy body and lack of skeletal support make the task even more challenging. In addition to momentum control for balance, which is not enough to prevent the "I" from falling, we exploit the advantage of its flexible body shape. We include a term in the objective function that encourages it to widen its support base. With this wider base, the contact area is increased and the COM is lowered, which helps with the balance task. Without our QPCC solver, base widening would be difficult to achieve, because it requires frequent switching from static to sliding contacts. In addition, we observe that right after the perturbation, half of the base is lifted from the ground and the contact area concentrates on the rim of the base to provide the maximum amount of angular momentum to combat the perturbation. It is similar to a human lifting his or her heels and using only toes to balance when pushed from behind. This natural contact

Table 2: Parameters and performance of examples. # tets: the number of elements in the FEM simulation. # dofs: the number of muscle degrees of freedom for the soft body. Sim time, opt time and total time are the average simulation, optimization and total time (in second) per frame.

examples	# tets	# dofs	# contact patches	sim time	opt time	total time
exercise(H)	1901	52	4	0.54	0.05	0.59
balance(I)	1066	48	4	0.31	0.28	0.59
slide(F)	705	48	4	0.25	0.23	0.48
jump(I)	1066	104	4	0.33	0.48	0.81
jump(T)	1219	51	4	0.63	0.30	0.93
roll(O)	911	40	6	0.26	0.18	0.44
crawl(I)	620	26	8	0.18	0.24	0.42
walk(X)	1128	112	4	0.54	0.70	1.24

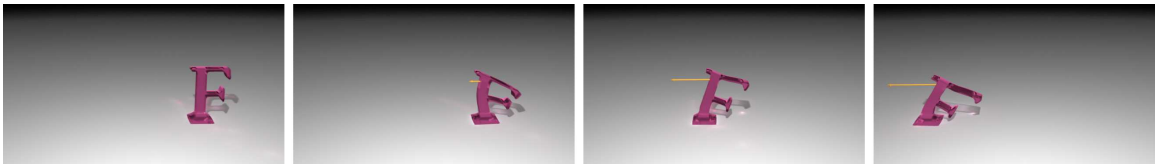

Figure 21: An F-shaped soft body character maintains balance under a persistent and continuously increasing pulling force on a slippery surface. It actively leans backward to avoid tipping over.

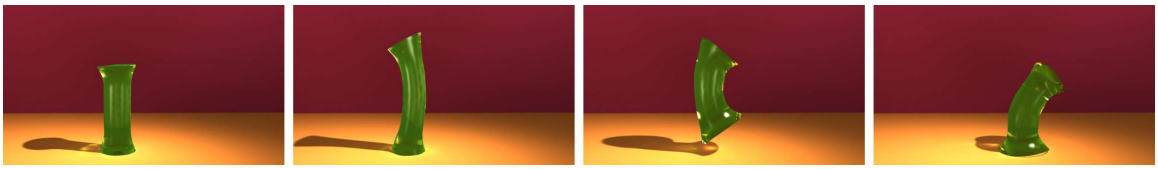

Figure 22: An I-shaped soft body character squashes and stretches its whole body to jump forward.

strategy emerges automatically from our QPCC solution. To compare our QPCC solver and a more commonly used QP solver with linear constraints, we produced two animation sequences of the "I" balancing, one with each solver. QPCC produced natural and effective balance motions, including changing contact situation, lowering the COM, widening the base and regulating momentum. In contrast, the QP solver (which only allows for static contact constraints) resulted in a falling motion.

Sliding. In the example of Figure 21, instead of applying a perturbation force that lasts for a short time, we exert an continuously increasing pulling force on the "F" standing on a slippery surface. We design a sliding balance controller for this special balance task. The controller estimates the optimal relative position between the center of base (COB) and the COM such that the total angular momentum is zero. We use the position and tracking controller to track the optimal COB. The sliding balance controller also benefits from our QPCC solution since planning the movement of the COB involves planing the change of contact situation (from static to sliding). We instrument the vertical stroke of the "F" with four longitudinal muscles. Even though no muscle resides in the horizontal parts of this character, the two horizontal strokes are still

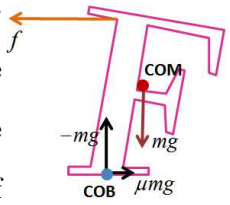

Figure 23: A T-shaped soft body character twists using helical muscles when jumping.

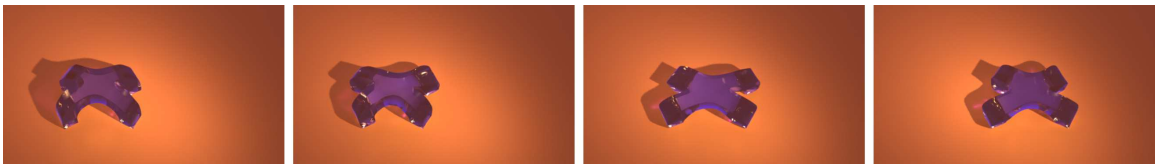

Figure 24: An X-shaped soft body quadraped walks by slowly lifting and moving one foot at a time.

influenced by the muscles in the main body. The first sequence of sliding balance in the video shows that the "F" leans left while it is dragged towards the right. As the drag force increases, the "F" leans more and more to prevent from tipping over. The second sequence shows the sliding motion when it is dragged to the left. The sliding motion is different from the first one due to the asymmetry of the body shape. The elongation and oscillation of the top stroke of the "F" demonstrates the animation principle of *follow through*. In the third sequence, we applied the sliding balance controller 0.3 second after the start of dragging to delay the character's response time. The slow response of the "F" makes it difficult to maintain the optimal COB-COM relative position. It struggles to keep balance by constantly switching between sliding and breaking contact (small jumps), and eventually it manages to balance. These changes of contact, due to the QPCC solver, makes the controller more robust and the soft body character more lifelike.

Jumping. Jumping is an visually interesting form of locomotion for soft body characters as it is often seen in cartoons and animations. Our jumping controller consists of three separate controllers for takeoff phase, airborne phase, and landing phase. During the takeoff phase, we use the position and velocity tracking controller to follow a desired trajectory of the COM. We also set $\bar{\dot{H}} = \mathbf{0}$ to the angular momentum controller, which prevents large rotation at takeoff. During the airborne phase, we control the relative position between the COB and COM. Extending the COB towards the direction of jumping helps the character

balance after landing. Upon landing, we switch to the static balance controller. Figure 22 shows a forward jumping motion of the same character "I" with a slightly different fiber arrangement. We add four more longitudinal muscles and one radial muscle to help it with this highly dynamic motion. Another sequence in the video shows successive jumps in place. Figure 23 demonstrates a twist jump and the use of helical muscles (Figure 25b). Before the "T" takes off, we set $\bar{\dot{H}} = (0, 600, 0)^T$ to make it twist its body.

Rolling. In the video, we also demonstrate locomotion by rolling. We designed an O-shaped character with two loops of muscle fibers arranged as two concentric circles (Figure 25c). Each fiber consists of 20 independent segments, which allow the "O" to control its shape locally. The rolling motion is initiated by moving the COM in front of the contact patches. We use the angular momentum control to make it roll. In the first animation, we set the desired change of angular momentum $\bar{\dot{H}}$ to be $(0, 0, -200)^T$ in the first 90 frames. We observe that the character actively changes its shape by shifting its weight to the right in order to roll. After the character starts rolling, we disable the controller and simulate the passive rolling. The character recovers to its original symmetric rounded shape and the rolling stops after a while due to friction. In the second animation, we compare our result

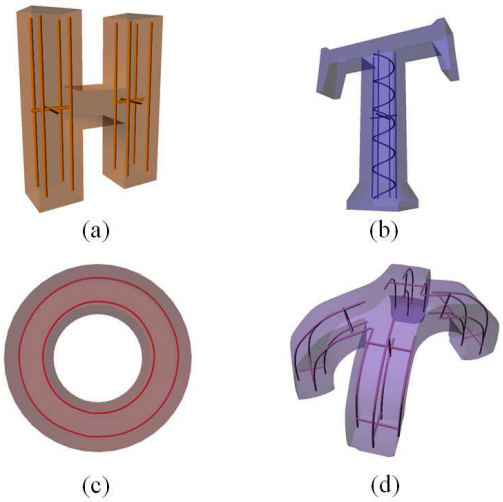

Figure 25: Examples of the muscle fiber designs for various soft body characters. Each curve inside the character represents a muscle fiber, which consists of a number of independently contracting degrees of freedom.

with the motion solved by a QP using the static contact assumption. When we only allow static contact, the "O" never begins rolling because this motion requires the character to break contact, which is prohibited by the static contact assumption (eq. (30)). The third sequence shows that the "O" starts to roll right, deaccelerates, stops and rolls to the left by applying a time varying $\bar{H}$ to the controller.

Crawling. Crawling is often used by soft body creatures in nature, such as earthworms. To demonstrate crawling motion, we flatten the "I" and lay it down on the ground. In addition to the four longitudinal muscles run along four sides of the body, we add another radial muscle in the middle of its body to facilitate the elongation of the body. We specify the trajectory of its four corners for the crawling motion; the back of the character moves while it is contracting, and the front moves when it elongates. We use the position and velocity tracking controllers to match the trajectory. As Miller noted, such creatures have oriented scales that result in anisotropic friction [105]. We incorporate just such an anisotropic friction into our contact model by modifying the contact force to $\mathbf{f}_c = \mathbf{N}f_\perp + \mathbf{DSf}_\parallel$, where $\mathbf{S}$ is a diagonal scaling matrix that modulates the frictional force according to the direction of motion. We set the friction coefficient in the backward direction to be 10 times larger than all other directions. In the video, we demonstrate the earthworm style of crawling. The whole body of the character lies flat on the ground at all times and it moves forward by repeatedly shortening and elongating its body. The contact strategy of this form of crawling is complex. During shortening, the front end of the body is in static contact while the rear end is sliding forward. During elongating, the front end switches to sliding contact while the rear end switches to static contact. It is challenge to capture this complex contact strategy using the traditional control mechanism, but it emerges automatically by solving QPCC.

We also demonstrates an inchworm style of crawling, using the same body geometry and muscles as the earthworm. For this style of motion, the body bends upward periodically at the middle and the contacts mostly concentrate at the two ends of the body. We achieve this effect using the same controller as in the earthworm style crawling with an additional

constraint that the upper longitudinal muscle cannot contract. While other muscles contract to tracking the trajectory, the asymmetric muscle contractions bend the body upwards naturally.

Walking. Figure 24 demonstrates the walking motion of an X-shaped quadruped. We instrument four longitudinal muscles and two radial muscles for each limb of the "X" (Figure 25d) and specify the trajectories for all of its four feet. We apply the position and velocity tracking controller, to track this the walking motion. The first sequence in the video shows a careful and slow gait that moves only one foot at a time. The second walking sequence shows a faster walking gait by simultaneously lifting and moving two feet at a time. The breaking of contact when a foot lifts from the ground is handled automatically by the QPCC solver.

4.7 Discussion

We have presented a system for animating soft body characters, with a particular emphasis on locomotion. Key aspects of our approach include the coordinated deformation of groups of finite elements using virtual muscle fibers, the specification of high-level goals by the animator, and the use of a new solver that handles static, sliding and breaking contact cases. Our system allows us to create soft body characters that demonstrate a variety of locomotion behaviors, including crawling, hopping, walking, sliding and rolling. Our characters move in an organic manner, and they follow the animation principles of anticipation, squash-and-stretch, and follow through.

One important contribution of this chapter is the formulation of QPCC and its solver. QPCC is an NP-hard problem [23] and our method provides an effective heuristic. To evaluate our QPCC solver, we tested it on 10 QPCC problems with 98 variables and 40 pairs of linear complementarity constraints. We compared our solutions (QPCC in Table 3) with the ground truth, as well as with the solutions based on the static contact assumption (QP in Table 3). The ground truth is computed by an exhaustive search, i.e. , solving a QP for every combination of complementarity variables and selecting the one with the lowest objective value. We evaluated our results using a "gap ratio", defined as the ratio of

Table 3: The results of the numerical experiments of the QPCC solver.

	QPCC			QP			Ground Truth		
	Avg	Best	Worst	Avg	Best	Worst	Avg	Best	Worst
Obj Value	1483.2	21.8	697.9	5222.8	2072.0	1246.9	772.9	0.0	0.0
#Iterations	17	3	31	1	1	1	10000	10000	10000

difference in the optimal value between our solver and the ground truth, to the difference in the optimal value between the static contact assumption and the ground truth. On average of 10 problems, the gap ratio is 6.29, indicating that our solver yields solutions 6.29 times closer to the ground truth than the solutions based static contact assumption. We also selected the best case and the worst case according to the gap ratio and reported them in Table 3. Although the empirical results showed that our QPCC solver can effectively solve contact resolution problem and balance between the quality of solution and computational time, the QPCC solver does not guarantee finding the minimizer in polynomial time. In the worst case, it takes the same amount of time as the exhaustive search to find the global minimizer.

Our system has a few limitations. The optimization scheme described in Chapter 4.4 only optimizes the control variables for the next time step. This type of greedy algorithm sometimes leads to unnaturally large muscle contraction or discontinuities in motion. For example, the rolling "O" demo in the video exhibits some unnatural vibration. Furthermore, the greedy algorithm prevents us from simulating anticipatory behaviors in motion. For example, we were not able to develop a "cartwheel" controller for the I-shaped character because a natural and stable cartwheel motion requires optimizing a long-window of trajectory. This issue can potentially be solved by implementing long-horizon optimization or model predictive control methods.

One of the issues that we hope to explore in the future is to expand our solution techniques to handle longer-term goals that cannot be reached using our current optimization method. In this work, all muscles are manually designed. We would like to develop an automatic muscle design algorithm to incorporate more sophisticated muscle structures. We would also like to investigate muscle design for chunkier creatures. For example, it is not

immediately obvious how muscle fibers should be arranged for the Stanford Bunny. Another possibility is to note that in our current system, we only use contracting muscle fibers in order to change the shape of our characters. It would be interesting to explore other forms of shape control, such as elongating muscles or sheets of virtual muscles. Another possible direction would be to explore the animation of soft body characters in water, since many real soft-body creatures live in an aquatic environment. Finally, our current animator controls are provided as program modules, and an easier way to use them would be to plug them together using a graphical user interface.

CHAPTER V

LOCOMOTION WITH A PASSIVE MECHANICAL DEVICE

5.1 Motivation

The invention of tools has greatly increased the efficiency of our work and the quality of our lives. The bicycle is such an invention that has drastically improved the efficiency of human locomotion. In fact, it was voted as the best invention since the 19th century [18] because it is an inexpensive, fast, healthy and environmentally friendly mode of transportation. However, riding a bicycle is nontrivial due to its inherently unstable dynamics: The bike will fall without forward momentum and the appropriate human control. Getting onto the bike, balancing, steering and riding on bumpy roads all impose different challenges to the rider. As one important child development milestone, learning to ride a bicycle often requires weeks of practice. We are interested to find whether it is possible to use a computer to mirror this learning process. In addition to the basic maneuvers, the most skillful riders can jump over obstacles, lift one wheel and balance on the other, and perform a large variety of risky but spectacular bicycle stunts. Performing stunts requires fast reaction, precise control, years of experience and most importantly, courage, which challenges most people. Can we design an algorithm that allows computers to automatically learn these challenging but visually exciting bicycle stunts?

Designing an algorithm to learn to ride a bicycle presents unique challenges. Riding a bicycle involves complex interactions between a human rider and a bicycle. While the rider can actively control each joint, the bicycle is a passive system that can only be controlled by the human rider. To control a single degree of freedom (DOF) of a bicycle, coordinated motions of multiple DOFs of the rider are required. Moreover, the main difficulty in locomotion is to control an under-actuated system by exploiting external contact forces. Manipulating contact forces on a bicycle is indirect. All of the rider's control forces need to first go through the bicycle dynamics to affect the ground reaction forces and vice versa.

This extra layer of bicycle dynamics between the human control and the contact forces adds another layer of complexity to the locomotion control. Balance on a bicycle is challenging and it is different from balance when standing or walking. While a human can stand stably due to the large contact area of the feet, a static bicycle cannot stay upright due to its narrow tires. Humans can balance during walking by planning and changing their foot placement. In contrast, the bike rider cannot abruptly change the contact position, which can only be changed gradually through steering. The limited range of human motion on a bicycle makes balance even harder. When the character's hands are constrained to hold the handlebar and their feet are on the pedals, the character loses much of the freedom to move various body parts. He or she cannot employ ankle or hip postural strategies or wave their arms to effectively regulate the linear and the angular momenta.

Balance during bicycle stunts is far more challenging than normal riding. As a stunt example that illustrates the balance issues we plan to tackle, consider a bicycle endo (the third image of Figure 3), in which the rider lifts the rear wheel of the bicycle and keeps balance on the front wheel. In this pose, the rider encounters both the longitudinal and lateral instabilities. The small contact region of one wheel and the lifted center of mass (COM) due to the forward leaning configuration exacerbate the balance problem. Furthermore, the off-the-ground driving wheel makes any balance strategies that involve acceleration impossible. The unstable configurations and the restricted actions significantly increase the difficulty of balance during a stunt.

This chapter describes a complete system for controlling a human character that is riding a bicycle in a physically simulated environment. The system consists of two main components: simulating the motion and optimizing the control policy. We simulate the bicycle and the rider as an articulated rigid body system, which is augmented with specialized constraints for bicycle dynamics. The second component provides an automatic way to learn control policies for a wide range of bicycle maneuvers. In contrast to many optimal control algorithms that leverage the dynamics equations to compute the control signals, we made a deliberate choice not to exploit the dynamics equations in our design of the control algorithm. We believe that learning to ride a bicycle involves little reasoning about physics

for most people. A four-year-old can ride a bicycle without understanding any physical equations. Physiological studies show that learning to ride a bicycle is a typical example of implicit motor learning [28], in which procedural memory guides our performance without the need for conscious attention. Procedural memory is formed and consolidated through repetitive practice and continuous evolution of neural processes. Inspired by the human learning process, we formulate a partially observable Markov decision process (POMDP) and use policy search to learn a direct mapping from perception to reaction (procedural memory).

Both prior knowledge of bicycle stunts and an effective searching algorithm are essential to the success of policy search. After studying a variety of stunts, we classify them into two types. We apply feed-forward controllers for the *momentum-driven* motions and feedback controllers for the *balance-driven* ones. We study videos of stunt experts to understand their reactions under different situations and used this information to design the *states* and the *actions* of our controllers. We employ a neural network evolution method to simultaneously optimize both the parametrization and the parameters of the feedback controllers. The way we incorporate the prior knowledge into our system and the effective evolutionary method are both essential to the success of policy search.

We evaluate our system by demonstrating a human character riding different types of bicycles and performing a wide variety of stunts (Figure 3). We also evaluate the importance of optimizing the parametrization of a policy. We share our experiences with different reinforcement learning algorithms that we have tried throughout this research project.

5.2 Overview

We have designed a system that allows a virtual human character to learn to ride a bicycle and perform a wide variety of stunts. The goal of our system is to learn a control policy that initiates a particular action at any state. Given the state space, the action space and the reward function for each bicycle task, the offline learning subsystem starts with an initial set of candidate policies, iteratively evaluates (using simulation) and evolves them (using CMA or NEAT) until the optimal policy is found. This optimal policy allows the user to interact

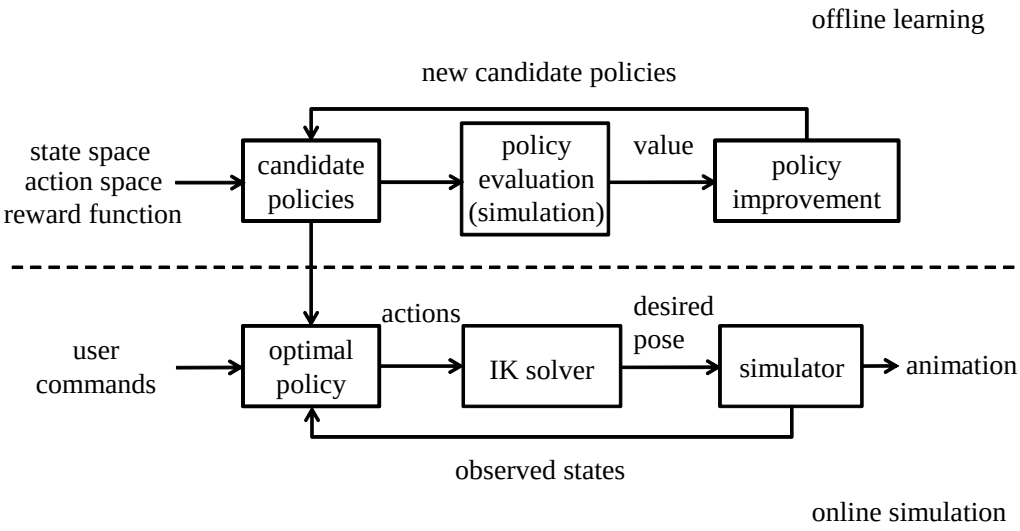

Figure 26: Overview of our algorithm.

with the bicycle simulation in real time by giving commands such as steering to the left. The online simulation subsystem first extracts the observable states, such as the tilt angle, the falling speed, and the actual and the user-specified handlebar angle. It then queries the policy to determine appropriate actions, such as turning the handlebar at a certain speed to fulfill the user's command while still maintain the balance of the bicycle. Executing actions, such as turning the handlebar, requires a coordinated full-body motion of the rider. An Inverse Kinematics (IK) solver maps the compact set of actions to the rider's full-body pose. The simulator tracks this desired pose and at the same time simulates the dynamics of both the human rider and the bicycle. Figure 26 illustrates the main components of our system.

5.3 Bicycle and Rider Simulation

Our simulator is based on Open Dynamic Engine (ODE) [139], but we augmented it with additional constraints to simulate both the bicycle and the human rider. We treat the bicycle and the rider as a system of articulated rigid bodies. Since the dynamics is represented in the maximal coordinates, each rigid body has six DOFs and its dynamic equation is

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} m\mathbf{g} \\ -\dot{\mathbf{I}}\omega \end{bmatrix} + \mathbf{J}^T \begin{bmatrix} \mathbf{f} \\ \tau \end{bmatrix} \tag{33}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{I}$ is the inertia tensor, $\mathbf{v}$ and ω are the linear and angular velocities, $\mathbf{f}$ and τ are the constraint forces and torques, which come from the bicycle chains, the joints, the actuators and the contacts. $\mathbf{J}^T$ is the transposed Jacobian matrix that maps the constraint forces and torques to the body.

Chains transfer power from the pedals to the rear wheel on a bicycle. We use a linear equality constraint to realize the effect of a bicycle chain.

$$\mathbf{n}^T(\alpha\omega_A - \omega_B) = 0 \tag{34}$$

where bodies A and B are the pedals and the rear wheel. $\mathbf{n}$ is the common direction of their rotational axes and α represents the gear ratio, which is the ratio between the number of teeth on the chain-ring and the number on the rear sprocket. Note that eq. (34) models a fixed-gear bicycle. In some tasks, we disabled this constraint to mimic the effect of the free wheel, allowing the rider to glide without pedaling.

The other constraints are standard from the implementation of ODE. For completeness of presentation, we include a brief description of such constraints in Appendix B. Note that we use the actuator constraints instead of the traditional PD servos to track a desired pose. We have found that using the actuator constraints enables us to simulate at large time steps (0.01s), which significantly speeds up the computation.

We made several simplifications in the simulation. We used ball joints to attach the rider's feet to the pedals. This treatment is similar to wearing toe clips in the real world. We also used ball joints to connect the rider's hands with the handlebar. For some tasks in which the rider is seated, we further constrained the relative position and orientation between the rider's pelvis and the bicycle seat.

5.4 Learning to Ride a Bicycle

5.4.1 Markov Decision Process

We formulate the bicycle control problem as a POMDP. A Markov decision process (MDP) is a tuple $(S, A, R, D, P_{sas'}, \gamma)$, where S is the *state* space; A is the *action* space; R is the *reward function*; D is the distribution of the initial state s_0; $P_{sas'}$ is the transition probability; and $\gamma \in [0, 1]$ is the discount factor. For example, in the bicycle balance task,

we choose the state space S to include the tilt angle of the bike, the tilting speed and the handlebar angle. We choose the action A to be turning the handlebar at a certain speed. We choose the reward function R at the state s to be

$$R(s) = \begin{cases} 1 & \text{if the bicycle remains upright,} \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

The initial state s_0 is drawn from a random perturbation near the upright orientation of the bicycle. The state transition is calculated using simulation, and we do not discount the rewards ($\gamma = 1$).

A *policy* is a mapping from states to actions: $\pi : S \mapsto A$. The *return* of a policy is the accumulated rewards along the state trajectory starting at s_0 by following the policy π for N steps.

$$V^{\pi}(s_0) = \sum_{i=0}^{N} R(s_i)$$

The *value* of a policy is the expected return with respect to the random initial state s_0 drawn from D.

$$V(\pi) = E_{s_0 \sim D}[V^{\pi}(s_0)] \tag{36}$$

The optimal solution of an MDP is the policy that has the maximum value $\pi^* = \arg\max_{\pi} V(\pi)$. The optimal policy in the bicycle balance example decides how to turn the handlebar under different situations so that the bicycle can stay upright for the longest time.

Our MDP is partially observable because we choose to observe only a selected subset of all the simulation states. We have found that focusing on a small number of relevant states for each task results in a more efficient learning process. The actions are also selected based on our prior knowledge of the task. Table 4 and 5 summarize all the states and actions used across our different examples. The 25 states in Table 4 may seem exhaustive, but we only use a subset of them (typically not more than eight states) for each task.

5.4.2 Policy Search

We apply policy search to optimize the control policies. Unlike value iteration, policy search can be easily applied to MDPs in high dimension and with continuous state and action spaces. This algorithm searches for the optimal policy within a parameterized functional

Table 4: States and their descriptions. The rider's pelvis position, torso orientation and angular velocity are calculated in the bicycle frame's coordinates.

state	description
t	time (used in the feed-forward controllers)
θ	handlebar angle
α	roll angle of the bike (tilt left/right)
$\dot{\alpha}$	roll speed
β	pitch angle of the bike
$\dot{\beta}$	pitch speed
γ	yaw angle of the bike
$\dot{\gamma}$	yaw speed
v_r	rear wheel linear speed
v_f	front wheel linear speed
h_r	rear tire height above the ground
h_f	front tire height above the ground
x	pelvis position along x-axis
y	pelvis position along y-axis
z	pelvis position along z-axis
ϕ	torso orientation in the Sagittal plane
ψ	torso orientation in the Coronal plane
χ	torso orientation in the Transverse plane
$\dot{\phi}$	torso angular speed in the Sagittal plane
$\dot{\psi}$	torso angular speed in the Coronal plane
$\dot{\chi}$	torso angular speed in the Transverse plane
$\Delta\theta$	difference between actual and desired handlebar angle
$\Delta\beta$	difference between actual and desired pitch angle
Δv_r	difference between actual and desired rear wheel speed
Δv_f	difference between actual and desired front wheel speed

space $\pi^* \in \Pi$. During policy search, one or more random policies are generated as an initial guess. These candidates are evaluated and improved iteratively. Policy improvement can be guided using the policy gradient [117], trajectory optimization [91] or other optimization techniques [57]. Policy search ends when the iteration converges or the maximum number of iterations is reached.

5.4.2.1 Policy Parametrization

We use two types of parametrizations for the bicycle control problem: splines for feed-forward control and neural networks for feedback control. We found that most of the stunts can be categorized into momentum-driven, balance-driven or a combination of the two. The momentum-driven stunts involve vigorous full body motions to manipulate the bicycle to a

Table 5: Actions and their descriptions. The rider's pelvis and torso movements are relative to the bicycle frame's coordinates.

action	description
$\dot{\theta}$	steering
$\dot{v}_r$	accelerating or braking
$\dot{v}_f$	accelerating or braking on a front-wheel-driven bicycle
τ_f	front braking
$\dot{x}$	pelvis motion along x-axis
$\dot{y}$	pelvis motion along y-axis
$\dot{z}$	pelvis motion along z-axis
$\dot{\phi}$	torso motion in the Sagittal plane
$\dot{\psi}$	torso motion in the Coronal plane
$\dot{\chi}$	torso motion in the Transverse plane
$\tilde{x}$	desired pelvis position along x-axis
$\tilde{y}$	desired pelvis position along y-axis
$\tilde{z}$	desired pelvis position along z-axis
$\tilde{\phi}$	desired torso orientation in the Sagittal plane

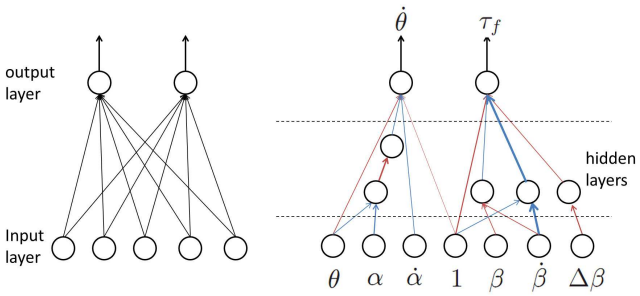

Figure 27: Left: A simple neural network with input and output layers that are directly connected. Right: A neural network learned using our algorithm for balancing on the front wheel. Blue arrows mean negative weights while red mean positive weights. The width of the arrows encodes the magnitude of the weights.

desired orientation. Coordinated full body motions with large magnitude are essential, but the short duration of this type of stunts makes balance easy to maintain. For this reason, we use feed-forward controllers and represent the action trajectories as cubic Hermite splines. Assuming that the number of control points is given, the parameters to optimize are the time and the value of the control points[1].

Balance-driven stunts require that the rider carefully adjusts his or her COM and maintains a stunt pose for a longer period of time. Feedback balance control is vital to the duration of the performance, which determines the success or failure of the stunt. We use

[1]We do not optimize the tangents at the control points and we set them to be zero.

neural networks for their ability to approximate a wide range of functions. The inputs to a network are the observed states, and the outputs are the actions. Figure 27 Left illustrates a simple neural network that directly connects the input and the output layers. The output of neuron i is

$$v_i = \sigma(\sum_j w_{ij} v_j)$$

where w_{ij} is the connection weight between neuron i and j, and σ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$.

Parametrization determines the potential quality of the optimal policy. The network shown in Figure 27 Left is too simple for representing a complex policy required by bicycle stunts. However, it is not clear how to manually design the network structure, given the control policies of unsuccessful stunts. For this reason, we use NEAT to search for both the structure of the neural network and its weights simultaneously, which finds far better policies than searching over a fixed network. Figure 27 Right demonstrates the learned network for the balance task of a bicycle endo using NEAT. See Chapter 5.4.2.3 for more details.

5.4.2.2 Policy Evaluation

To evaluate a policy, we formulate a reward function in the following form:

$$R(s) = R_t(s) + wR_r(s) \tag{37}$$

where R_t and R_r are task-specific and regularization terms respectively. w is the weight.

We use eq.(35) as the task-specific reward for balance-driven tasks. As the reward is accumulated over time, the return counts the number of frames that the bicycle stays upright. The task-specific reward varies for each momentum-driven stunt. For example, the reward for initiating an endo (lifting the rear wheel) is to maximize the negative pitch angle of the bike $R_t = -\beta$. We refer the readers to Chapter 5.5 and for more detailed descriptions of task-specific rewards.

Given the task-specific reward term alone, multiple optimal policies could exist. Taking the balance task as an example, a policy that rides in a straight line and another that

oscillates in a sinusoidal path by periodically swinging the handlebar can both balance well and thus yield the same value. The regularization term is mainly used to eliminate this ambiguity. We use the regularization term $R_r = \frac{1}{|\theta|+\epsilon}$ to express our preference of riding straight. In our examples, all the regularizers are in the form of

$$R_r = \frac{1}{|X| + \epsilon}$$

where X can be substituted by α for the upright bicycle position, $\Delta\theta$ for the desired steering angle, $\Delta\beta$ for the desired pitch angle, Δv for the desired speed, (x, y, z) and (ϕ, ψ, χ) for small changes of rider's pelvis position and torso orientation. A small number ϵ in the denominator is used to bound the reward.

We do not explicitly minimize the rider's effort in the reward function because it is difficult to balance the effort minimization objective and the task-specific objective for difficult stunt actions. However, we limit the maximum actuated joint torques of the rider in the simulation to ensure that the rider does not possess super-human strength.

We run multiple simulations with different initial configurations s_0, which are sampled from random perturbations of the default bicycle velocity and orientation, to evaluate the value of a policy. At each simulation step, our algorithm calculates the reward for the current state, and accumulates this until the bicycle falls or after 1000 time steps. The average return of all the simulations is the value of the policy.

5.4.2.3 Policy Improvement

Many policy improvement methods utilize the policy gradient [123] to perform iterative ascending operations. However, our simulation of bicycle stunts involves frequent discrete events such as establishing and breaking contact, which invalidates the gradient information. For this reason, we use sample-based stochastic optimization techniques. We apply CMA to search for the feed-forward controllers since the parametrization of splines is fixed. We use NEAT to search for feedback controllers, including the structure and the weights of the neural network. NEAT has many similarities to genetic algorithms, but it is tailored to the creation of neural networks. We will describe NEAT briefly below. For further details we refer readers to the original paper [142].

NEAT iteratively performs evaluation, selection, crossover and mutation. To maximize the value of a policy, NEAT starts with a simple network structure, in which the input and the output layers are directly connected. A population of such networks with random weights is drawn as an initial guess. These candidate policies are evaluated and the top 20% are selected to survive. Pairs of randomly-selected surviving policies are crossed over to produce a new generation (more on this below). Mutations (with low probability) can perturb connection weights, add a neuron or add a connection. Note that the addition of a neuron or a connection complexifies the network structure and enriches the class of functions that it can represent.

Crossover is nontrivial in NEAT because the parent neural networks can have different topologies. To overcome this difficulty, the newly-added neuron or connection is assigned a unique innovation number, which tells the history of the mutation and how to match up neurons or connections between parents during crossover. The neurons or connections that share the same innovation number across parents are from the same ancestor, which will be inherited randomly by the child sample. The neurons or connections that have no counterparts in the other parent are from different mutations. They will be inherited from the parent with the higher value.

The evolution ends when the policy values do not increase over a certain number of iterations or the maximum number of iterations is reached.

5.5 Results

In this section we describe the results of our system. Please watch the video[2] for the bicycle riding and stunt animations. Our system was implemented in C++, and we used ODE with additional chain constraints to simulate both the bicycle and the human rider. The simulator runs in real time on a desktop workstation with a 2.26GHz CPU and 4GB of memory. We generated 90 samples per iteration and 50 iterations for offline policy search. The computations were distributed across 16 CPU cores on a cluster. The learning time ranges from a few minutes to half an hour depending on the number of simulations used to

[2]https://dl.dropboxusercontent.com/u/36899427/siggraph2014.mp4

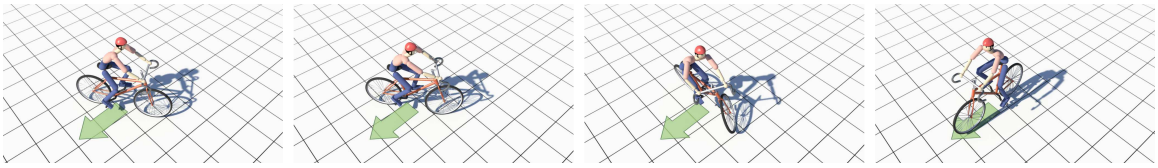

Figure 28: A character steers the road bike towards the green arrow.

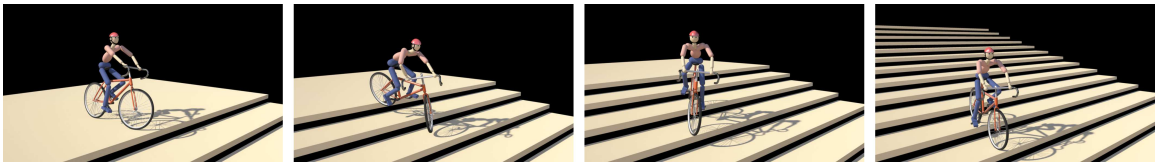

Figure 29: A character rides down a set of stairs without falling over.

estimate the expected return (eq. 36). Table 6 summarizes the choices of states and actions for each bicycle task.

We designed three different bicycles and a unicycle to test our controllers on a variety of tasks. *Road bikes* (Figure 28) are designed to travel at speed on paved roads. They have very narrow tires to reduce the rolling resistance. The seats are mounted high so that the riders can bend their upper bodies down for less air resistance. We use a *BMX bike* (Figure 31) for stunts. BMX bikes are usually considerably smaller for nimble and agile handling. They have fat tires to facilitate balance and to increase traction. BMX bicycle parts can often be customized to meet the needs of different stunts. *High wheelers* (Figure 34) are an old style of bicycle appearing in the late 19th century. They have a large front wheel and a much smaller rear wheel. This peculiar design makes high wheelers difficult to ride due to the center of mass being high and not far behind the front wheel. Any sudden stop could send the rider over the handlebars. A *unicycle* (Figure 35) has only one wheel and no handlebar. The rider needs to be concerned about the balance in both the longitudinal and the lateral directions. Different cycle designs greatly affect the handling characteristics and change the behavior of the riders. This variety puts the generality of our algorithm to the test. We modeled all the cycles and calculated their mass properties in SolidWorks.

Balance and steering. Riding a bicycle requires balance and steering. Balance can be maintained by steering toward the falling direction, which generates centrifugal force to push the bike upright. Figure 28 shows that our learned controller enables the rider to balance and steer the bike towards a user-specified direction. The bike follows the green arrow closely even when the user changes the desired direction abruptly. This agile steering behavior is achieved through "counter-steering": a momentarily steering in the opposition direction to initiate a sharp turn [127], which emerged automatically from the policy search. We also tested the robustness of our balance and steering controller on a bumpy terrain, which is represented as a noisy height field sampled from a uniform distribution $h \sim U(0, 0.05)$ (unit: meter). Even though the bicycle jumps and the handlebar is perturbed constantly, the rider still manages to balance and closely follows the desired direction. In addition, the accompanying video shows an initial starting motion, in which the rider's left foot is scripted to push the ground and move towards the pedal. Based on this single trajectory of foot and the learned balance policy, we used IK to generate the full-body motion of the starting phase.

Going down stairs. Figure 29 shows the character riding down a series of stairs. Each step is 0.15m high and 0.8m wide. We used the same balance controller as in the previous

Table 6: Choices of states and actions for each bicycle task. Note that in the momentum-driven tasks, the actions only depend on time t while the remaining states are used to compute the reward.

task	states	actions
momentum-driven		
going over curbs	t, β	$\dot{v}_r, \tilde{\phi}$
endo (lifting)	t, β	$\tau_f, \tilde{y}, \tilde{z}$
front wheel pivot	$t, \dot{\gamma}$	$\dot{\theta}, \tau_f, \tilde{y}, \tilde{z}$
bunny hop	t, h_f, h_r	$\tilde{y}, \tilde{z}$
balance-driven		
balance and steering	$\theta, \Delta\theta, \alpha, \dot{\alpha}$	$\dot{\theta}$
wheelie	$\alpha, \dot{\alpha}, \beta, \dot{\beta}, \Delta\beta, v_r, \psi, \dot{\psi}$	$\dot{v}_r, \dot{\psi}$
endo (balance)	$\theta, \alpha, \dot{\alpha}, \beta, \dot{\beta}, \Delta\beta$	$\dot{\theta}, \tau_f$
back hop	$\alpha, \dot{\alpha}, \beta, \dot{\beta}, \Delta\beta, x, y, z$	$\dot{x}, \dot{y}, \dot{z}$
high wheeler (stunt)	$\beta, \dot{\beta}, \Delta\beta, \Delta v_f$	$\dot{v}_f$
unicycle	$\alpha, \dot{\alpha}, \beta, \dot{\beta}, v_r, \Delta v_r, \chi$	$\dot{v}_r, \dot{\chi}$

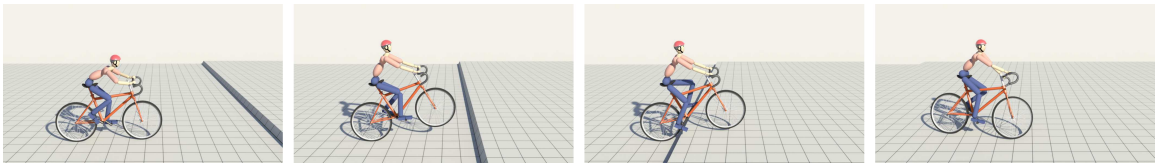

Figure 30: A character lifts the front wheel to ride over a curb.

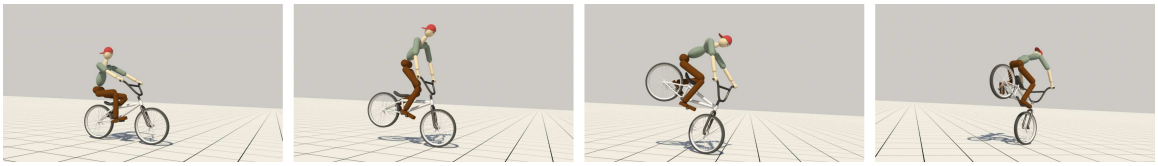

Figure 31: A character performs an endo and balance on the front wheel.

example. This balance task is more challenging because the frequent loss of contact and the sudden collisions between the front tire and the ground narrow the window of effective control and introduce large perturbations. Initially, the rider needs to make large corrections with the handlebar to keep balance when the forward speed is low. As the bicycle travels faster, the corrections become smaller and steadier.

Going over curbs. Riding over curbs (Figure 30) can be performed by lifting the front wheel using feed-forward control only. We therefore parameterized the actions with two splines (Table 6) and trained the controller using CMA. We used a task-specific reward function to maximize the pitch of the bicycle $R_t = \beta$ during lifting. In the animation, as the bicycle approaches a curb (0.12m high), the rider first leans forward and then pushes her upper body backwards. When the arms are stretched out to the maximum length, the sudden deceleration of the upper body pitches the whole bike upwards. The pitch angle is further increased by pedaling faster. This sequence of highly coordinated movements is discovered automatically by the policy search algorithm. Once the front wheel goes over the curb, the balance controller takes over and keeps the bike upright when the rear wheel hits the curb.

Real time user interaction. A user can interact with our bike simulation in real time. We video captured a sequence that shows a person using a joystick that is equipped with motion sensors to control the rider and the bicycle. The rider goes over a curb, survives

a crosswind, goes down a set of stairs, and follows a curvy path to the goal. Note that the user only gives high level commands such as the desired steering angle and the timing of lifting the front wheel. The balance, the actual steering and the rider's motions are all controlled by the policy learned from the offline training process.

Wheelie. A wheelie is a stunt maneuver in which the rider first lifts the front wheel and maintains balance on only the rear wheel. Lifting the front wheel on a BMX bike is considerably easier than on a road bike due to the shorter distance between the wheels. It can be achieved by increasing the speed of pedaling without any noticeable upper body movements. For this reason, we used only a feedback controller (Table 6) to perform a wheelie, including both the initial lift and the later balance. Once the front wheel leaves the ground, the rider adjusts the forward speed to keep the desired pitch angle. He leans his upper body to the left or to the right to correct any lateral instability.

Endo. Figure 31 shows an endo. In contrast to a wheelie, an endo lifts the rear wheel and balances on the front wheel. In spite of its symmetry to a wheelie, an endo requires an entirely different set of skills and environments. Endos are usually performed on a gentle downward slope, which provides the needed forward momentum when the driving wheel is off the ground. We used a slope of 2.5 degrees in our example. We first search for a feed-forward controller that maximizes the negative pitch angle $R_t = -\beta$ to initiate the stunt. The resulting controller slowly moves the rider's pelvis to the back, and then quickly throws it to the front to lift the rear wheel.

The feed-forward controller is succeeded by a feedback balance controller. To maintain an endo, the rider continuously applies or releases the front brake for longitudinal balance and steers the handlebar towards the direction of leaning for lateral balance. This stunt is especially challenging. If the pitch angle is too large, when the COM is above or in front of the front tire contact, such a gentle slope cannot provide enough acceleration to prevent overturning. If the pitch angle is too shallow, to prevent the rear wheel from dropping to the ground, braking hard will quickly halt the bicycle and make the balance strategy of "steering toward falling" ineffective. This complicated coupling between the pitch angle,

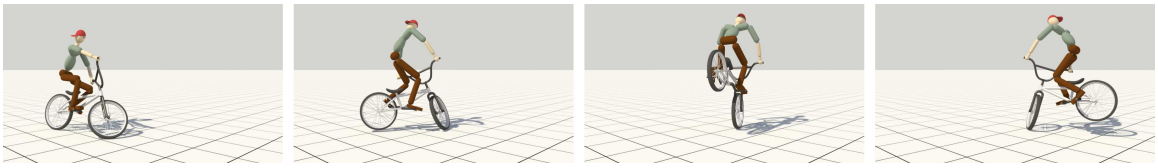

Figure 32: A character completes a quick 180-degree turn by pivoting the bicycle on the front wheel.

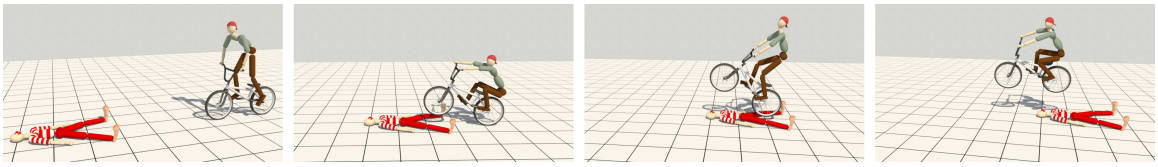

Figure 33: A character performs the American bunny hop over a clown lying on the ground.

the speed and the lateral balance makes heavy demands on the policy search algorithm. NEAT successfully finds a policy that can maintain balance for an extensively long period of time. Figure 27 Right illustrates the complex neural network required for this task.

In the accompanying video, we also demonstrate the learning process of an endo. The animation shows the resulting balance controller after one, five and ten iterations. As more iterations are finished, the rider gradually masters an endo and maintains balance for a longer period of time.

Front wheel pivot. A front wheel pivot (Figure 32) is a fast way to turn the bicycle 180 degrees by pivoting on the front wheel. We used a feed-forward controller and applied two separate task-specific rewards during two phases of this motion. The first reward function maximizes the angle of turning during the pivoting phase.

$$R_{t_1} = \begin{cases} \dot{\gamma}\Delta t & \text{if } h_r > 0.01, \\ 0 & \text{otherwise.} \end{cases}$$

After the rear wheel touches the ground, we switch to a learned balance controller and measure how long the bicycle can stay balanced: $R_{t_2} = 1$ if the bicycle remains upright. Without the second reward function, the "optimal" policy can produce a large roll during the pivoting phase, after which the rider cannot recover balance. In the animation, the rider performs an endo after turning the handlebar sharply to the left. As a result, the rider and the bike pivot around the front wheel and the 180-degree turn finishes within three seconds.

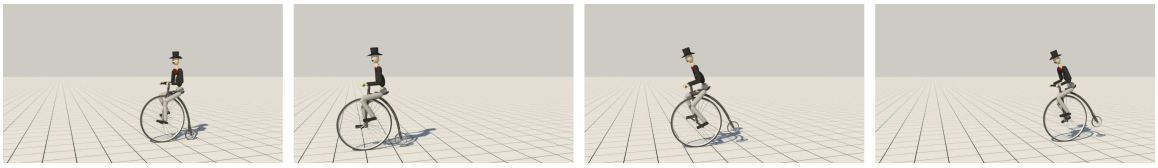

Figure 34: A character rides a high wheeler and performs a stunt in which he rides backward on a single wheel.

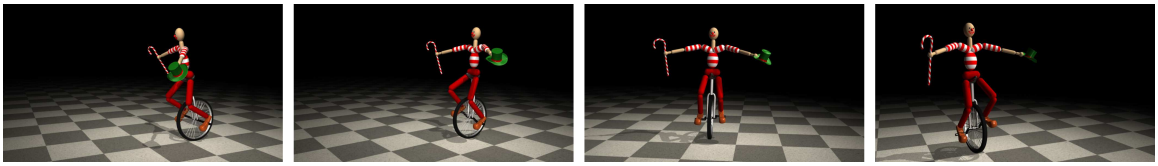

Figure 35: A clown rides a unicycle.

Back hop. The back hop is another way to balance on the rear wheel. This feedback balance strategy uses small hops to change the relative position between the contact and the COM. In the animation, the rider and the bike start at an initial pose in which the COM is behind the contact point between the rear wheel and the ground. The bike will fall backwards if the rider does not correct for this. He bends his knees and then extends them to bring the rear wheel off the ground. He quickly pulls the handlebar towards himself in mid-air to adjust the pitch of the bicycle. When the rear wheel lands, the COM comes closer to the ground contact position. As a result, the rider can continue to hop and balance for a long time.

Bunny hop. Figure 33 shows the rider performing a bunny hop on a BMX bike. A bunny hop is a stunt where the rider jumps with the bike over an obstacle or a trench. The task-specific reward function for the feed-forward controller is evaluated based on the height of both tires above the ground $R_t = h_f h_r$. Right before the hop, the rider first leans forward and then moves his pelvis rapidly towards the back of the bicycle. This vigorous motion tilts the bicycle upward. The rider then jumps with the bicycle over a clown lying on the ground. We were pleased to see that the optimal policy for the bunny hop motion includes a phase that tilts the bicycle up, which is essential to jumping for a greater height and distance. This style is known as the "American bunny hop".

Riding a high wheeler. The peculiar design of a high wheeler makes "headers" a significant hazard, in which the rider gets pitch forward off the bicycle. We took on the challenge and designed a stunt that we had never seen performed on a high wheeler (Figure 34). During the stunt, the rider intentionally stops the bike to initiate a forward rotation. He then carefully changes pedaling speed to avoid taking a header and successfully rides backward on a single wheel. The rider can balance for a few seconds even without a lateral balance controller, probably due to the prominent gyroscopic effect from the large rotating front wheel. When the rider starts to lose balance, he accelerates to return to the normal riding mode, in which we used the same balance and steering controller as in the road bike examples.

Riding a unicycle. The unicycle and the bicycle have many similarities. We have found that our algorithm is general enough to handle balance on a unicycle. Similar to some bicycle stunts, the longitudinal balance on a unicycle can be maintained via speeding-up or slowing-down, while the lateral balance can be maintained through steering toward falling. Unfortunately, unicycles do not have handlebars to steer. To steer the unicycle to the left, the rider needs to twist his upper body to the right. The unicycle will counter-rotate to the left due to the conservation of angular momentum. Figure 35 shows a clown riding a unicycle. To start riding, the clown first pedals backward, which leans the unicycle to the front. He then accelerates until the actual speed matches the user-specified desired speed. During riding, the clown repeatedly twists his waist to keep the lateral balance, which causes the unicycle to travel in a slightly sinusoidal path.

5.6 Discussion

We have presented an algorithm for animating bicycle stunts. Key aspects of our approach include a fast and stable physical simulation, policy search for POMDP, and the use of a sample-based optimization solver that searches for both the structure and the weights of a neural network. Our system allows us to control a human character that performs a variety of bicycle stunts, including wheelie, endo, bunny hop, front wheel pivot and back hop. Our characters can learn to master the most demanding balance tasks in minutes,

which challenges most people and takes months of practice to learn.

Although the learning algorithms presented in this chapter are general, the qualities of simulated motions vary with different tasks. We notice that some of the stunt motions do not look as compliant as those performed by real human riders due to three possible reasons. First, we chose large torque limits to allow robust control for various challenging maneuvers (Table 7). We used the same torque limits both in offline learning and online simulation to generate the results shown in the accompanying videos. To investigate whether we can achieve more compliant motions, we reduce the joint torque limits for simulation until the rider can no longer maintain balance (shown as the values in the parenthesis in Table 7). Although our controllers are robust when executed with smaller amount of torques, the resulting human motion appear very similar. The second possible reason is that we did not minimize rider's effort during offline learning. We found that it is difficult to weigh the effort minimization term because it competes with the task-specific objective. One possible solution is to implement prioritized optimization [36] to incorporate the effort minimization objectives. The last reason is probably due to the use of actuator constraints in ODE. The actuator constraints usually result in more stable simulation. However, since the joint torques are solved together with all other constraint forces, this allows the character to react instantaneously. We believe that the lack of reaction latency contributes to the stiffness of the motion. Using traditional PD servos could mitigate this problem but could also significantly increase the time of learning and simulation.

We further examine the torque trajectories (Figure 36), and observe that the controllers learned different strategies to achieve different tasks. In the endo example, the torques switch abruptly between the two extreme values. It is similar to the "bang-bang control" that frequently arises in time-critical tasks, indicating that the task of endo might also be time-critical. In contrast, the torque trajectory of riding a bicycle on a flat terrain shows more smooth transitions, which implies that normal biking does not require instant reactions and is thus an easier task.

Policy search is a powerful method for learning optimal controllers. However, manually designing parametrization of the policies could be challenging for difficult tasks. Combining

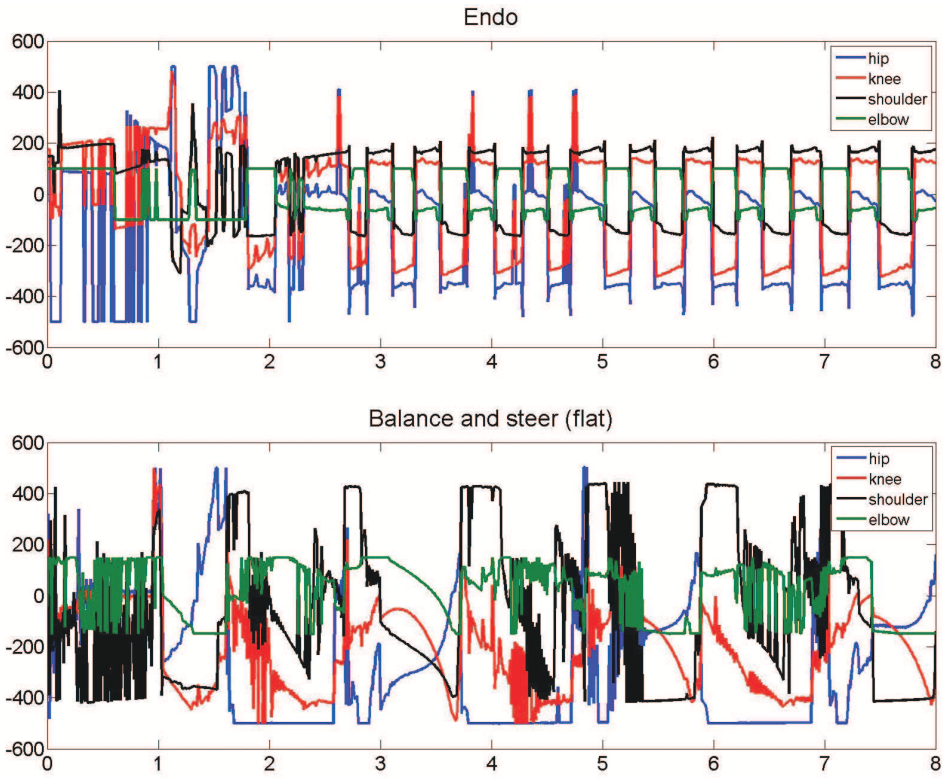

Figure 36: Torque trajectories over time of performing an endo and riding a bicycle on a flat ground.

policy search with NEAT makes it easy to use and unleashes its full power. We have demonstrated that searching for both the parametrization and the parameters of a policy creates robust controllers for a wide variety of bicycle balance tasks. Since many previous studies focused on optimizing the parameters alone, we evaluated the necessity of optimizing the parametrization. Figure 37 Left and Right compares two policy search results with a fixed and with an evolving parametrization for the balance task of an endo. Both searches started with the same parametrization: a neural network with direct connections between the input and the output layers. We used CMA to search for only the weights while we used NEAT to evolve both the weights and the network structure. Both algorithms were run for 50 iterations with 90 samples per iteration. We ran each search ten times with different random seeds to reduce the stochastic bias. Figure 37 Left plots the curves of the policy value versus the number of iterations in the CMA search. Note that none of the searches reached a value of 3000. In comparison, eight out of ten NEAT searches found policies scored

higher than 3000 (Figure 37 Right). Three of them reached almost 7000. The average final policy value of NEAT was almost twice its CMA counterpart. A similar comparison was conducted between neuroevolution with and without augmenting topology, and this told the same story (Figure 37 Middle vs. Right). Even though we only reported the detailed comparison for this particular example, we observed the same trend for other bicycle stunts: policy search with an evolving parametrization significantly outperforms search with a fixed parametrization. The neural network structure for all the balance-driven tasks are shown in Appendix C.

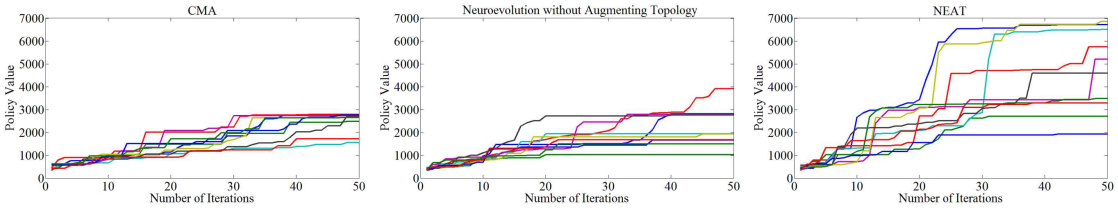

Figure 37: A comparison between policy searches with a fixed and an evolving parametrization. Left: The policy value vs. the number of iterations for ten policy searches on a fixed parametrization using CMA. Middle: Results of ten policy searches on a fixed parametrization using neuroevolution without augmenting topology. Right: Results of ten policy searches on an evolving parametrization using NEAT.

As illustrated in Chapter 5.5, many bicycle stunts need balance in both the longitudinal and lateral directions. In our implementation, we decoupled the balance task in these two directions and learned the task in two steps. In the first step, we focused on longitudinal balance and chose only the relevant states and actions. We artificially increased the width of the tire to 20cm so that the controller did not need to be concerned about the loss of balance in the lateral direction. This is analogous to using training wheels in real life. Once the longitudinal controller had been learned, we fixed that part of the neural network, reduced the width of the tire back to normal[3] and then performed the second step of training for lateral balance. Although in theory, optimizing a controller for the longitudinal and the lateral balance simultaneously would have a better global optimum, in practice, searching for a policy in a higher dimensional space is subject to local minima and thus could produce

[3]The tire width of the road bike and the high wheeler is 1.5cm while the tire width of the BMX bike and the unicycle is 3cm.

inferior results. In all our examples, this two-step training found better policies than training both longitudinal and lateral balance simultaneously.

Although we chose to use neural network and NEAT to tackle this problem, there could be other alternatives. One possibility is to use optimal control algorithms, such as Linear Quadratic Regulator (LQR) or Dynamic Differential Programming (DDP). These methods start with a nominal trajectory and linearize the dynamics around it. However, it is not clear to us how to design good nominal trajectories for a wide variety of challenging bicycle stunts. Moreover, bicycle stunts involve frequent contact changes. This makes the dynamics nonlinear and discontinuous. Thus, using linearized dynamics is unlikely to succeed. We have also tried a couple of other reinforcement learning techniques. They ended up unsuccessful, but these valuable experience has led us to our current solution. Our first attempt was to use standard value iteration method with a discretized state space. The high dimensional state space made the tabular representation infeasible. We encountered the convergence issue when the state space was parameterized by polynomial and Gaussian kernel bases. We also experimented with a few model-free learning algorithms. For example, SARSA(λ) demonstrated some potential (i.e. the normal cycling was successful), but the computation time was too long even for the simplest task. Although our final decision was to use policy search, the initial experiments were unsuccessful due to an overly simplified policy parametrization: a neural network without hidden layers. Using quadratic features to enrich the inputs of the network, we had some limited success with the optimal solutions solved by CMA. However, without a systematic way to perform feature selection, the likelihood of finding a successful local minimum is low due to the large number of quadratic features. Finally, we chose NEAT, a bottom-up approach that complexifies the parametrization from the simple network. This method consistently found policies that worked for all our examples.

NEAT provides an effective way to search for both the parametrization and the parameters, which frees us from laborious and unintuitive manual tuning of the network structure. However, to formulate an MDP, we still need to design the state space, the action space

and the reward function. Selecting the right features to represent the states and the actions is vital to the success of the entire algorithm. The rule of thumb is to choose only a small number of features that are most relevant to the task. If too many states and actions are used (e.g. all the simulation states as the state space and all the joint torques as the action space), it is not likely that NEAT will find a successful controller due to the high dimensional search space. Selecting the states and actions for a specific task often requires domain knowledge. For example, knowing how to steer a unicycle (twisting the upper body) is essential for the unicycle balance task. Likewise, designing a good reward function requires some trial-and-error experiments. Our reward functions initially contained only the task-specific terms. We found that the algorithm often discovered a policy that had a high value but resulted in undesired motions: The rider performed redundant movements as long as this did not affect the balance. We eliminated these undesired motions by adding regularization terms to the reward function. However, selecting states, actions and reward functions is unavoidable when formulating and solving MDPs. Inverse reinforcement learning [118] might be a promising approach, but this requires data generated from the optimal policy, such as motion capture data from real stunt bikers. Another promising future research is to automatically extract features for different bicycle stunts using deep learning. Please refer to Chapter 7.2 for more discussion on this topic.

Our system has a few limitations. We used ball joints to attach the feet to the pedals. These bilateral joint constraints simplify the simulation but render some motions less realistic: The brief moment in our bunny hop animation when the rear wheel pops off the ground just before the hop typically is not seen in the real stunt. Faithfully modeling the contact between the feet and the pedals could solve this problem. However, this will make the simulation more expensive and the learning more difficult. In the learning, we separated the feed-forward and feedback controllers. This treatment works well if the feed-forward controller is applied for a short period and then we switch to the feedback controller. However, in a real-life performance, stunt bikers exhibit both feed-forward and feedback controls at the same time, which allows them to perform longer and more difficult stunts. This might be one of the reasons that our algorithm cannot generate a 360-degree front wheel pivot. Our

interactive simulation does not allow the user to arbitrarily concatenate the stunts. Successful stunts demand a stable initial pose, appropriate speed and sometimes a favorable terrain. All these requirements together prevents us from concatenating stunt controllers arbitrarily. Learning additional intermediate controllers between stunts might be a possible solution.

There are a number of interesting avenues for future work. We believe that our algorithm will not be limited to the bicycle control tasks. It has the potential to solve other challenging control problems, such as gymnastics, skateboarding, surfing and dancing. Deep learning [58] and deeper neural networks could be explored for even more difficult stunts. Our work has concentrated on controllers for individual stunts. It would be interesting to investigate a sequence of stunts that traverse a bicycle course with obstacles. Finally, it would be fascinating to manufacture a mini bicycle and a robotic rider and apply our controllers to make them perform stunts in the real world.

Table 7: Torque limits (Unit: Nm) for different tasks. The values without the parenthesis are used for learning. The values within the parenthesis are the lowest torque limits that can be used in the simulation before the motion starts to fail. The amount of reduction is a measure of the robustness of the learned controllers. More reduction means a more robust controller. The last column reports the reduction ratios of average torque limits, which are sorted from high to low. This ranking is a possible indication of the difficulties (from low to high) that the learning algorithm think for each task.

task	wrist	elbow	scapula	shoulder	pelvis	hip	knee	ankle	ratio
balance and steering (flat)	50 (5)	150 (20)	200 (30)	550 (40)	2000 (100)	500 (150)	500 (50)	500 (50)	0.9
unicycle	10 (1)	100 (5)	100 (20)	150 (10)	2000 (30)	100 (95)	100 (90)	60 (20)	0.89
wheelie	10 (10)	100 (30)	100 (50)	150 (80)	2000 (100)	500 (150)	100 (100)	60 (60)	0.81
going over curbs	50 (30)	150 (50)	200 (50)	550 (80)	2000 (100)	500 (150)	500 (100)	500 (60)	0.76
high wheeler	50 (10)	150 (50)	200 (20)	550 (200)	2000 (300)	500 (440)	500 (360)	500 (230)	0.64
balance and steering (bumpy)	50 (50)	150 (100)	200 (100)	550 (350)	2000 (400)	500 (300)	500 (350)	500 (200)	0.58
front wheel pivot	250 (50)	500 (150)	500 (150)	550 (400)	2000 (600)	500 (500)	500 (280)	500 (80)	0.58
staircase	50 (40)	150 (100)	200 (90)	550 (350)	2000 (400)	500 (450)	500 (450)	500 (80)	0.56
bunny hop	20 (15)	50 (50)	500 (150)	550 (150)	2000 (750)	500 (400)	500 (380)	500 (220)	0.44
endo	50 (50)	100 (100)	100 (100)	550 (400)	2000 (600)	500 (470)	500 (485)	500 (150)	0.45
back hop	250 (90)	500 (400)	500 (500)	550 (545)	2000 (900)	500 (500)	500 (450)	500 (140)	0.33

CHAPTER VI

CONTROLLER TRANSFER FROM THE VIRTUAL TO THE REAL WORLD

6.1 Motivation

In character animation, we are able to reproduce many of the diverse and agile locomotion in nature. However, we have not yet seen the same level of motor capabilities in robotics. There is a large gap between what a character can do in the virtual environment and what a robot can do in the real world. This gap is due to the tremendous challenges when designing controllers with the real hardware since the majority of the robot controllers today are designed directly on the hardware. Robot hardware is expensive, has severe limitations (accuracy, repeatability, noise, torque limits, etc.), and requires frequent maintenance. For these reasons, designing robotic controllers is a time-consuming, labor intensive and trial-and-error process that is limited only to highly-specialized engineers. We have demonstrated in the previous chapters that with the powerful computational tools, controllers can be designed autonomously in a virtual environment. Can we extend the computational tools developed for character animations to design motion controllers for robots?

The major challenge to directly apply the methods in character animation to robotics is the Reality Gap: Controllers that work effectively for a virtual character in the simulation may perform poorly on a robot in the real environment. The cause of the Reality Gap is the various simplifications in simulation algorithms, such as simplifying dynamics models, using inaccurate physical parameters, ignoring hardware limitations, noise and latency. For example, we do not model the internal mechanism of a servo in character animation. We often use rough estimations of the physical parameters, such as mass, COM and moment of inertia of a character in the simulation. Even though we can acquire some of these parameters from the Computer-Aided Design (CAD) specification files of a robot, they are also inaccurate given the manufacturing and assembling errors. Furthermore, we usually do

not model the noise in the environment and the latency in the hardware communication in character animation.

Although crossing the Reality Gap entirely is an extremely challenging problem, we can still tap into the power of the computational tools if we can sufficiently narrow down the differences between the simulation results and the real robot performance. In this work, we develop a system with three components. In addition to physical simulation and controller optimization that we use extensively in character animation, we introduce *simulation calibration* to narrow down the Reality Gap. During simulation calibration, we collect real performance data on the robot, and use it to improve our physical simulator. We optimize a set of simulation parameters to minimize the discrepancy between the simulation results and the collected real data. Through calibration, the simulator can capture the real world dynamics more faithfully. This calibrated simulator is used again in controller optimization to improve the quality of the controller. Depending on the task, a controller optimized in the simulation could be successfully transferred to a robot within a small number of iterations of simulation calibration and controller optimization. As a result, our system drastically reduces the number of time-consuming robot experiments and replaces the tedious manual tuning with an automatic optimization process.

We evaluate our system using four motion planning tasks: rising from a leaning, sitting, or kneeling position to an erect stance, and flipping from a standing to a handstanding pose. These tasks play an important role in our daily life. They are so common that we perform them many times everyday. Although most of us can achieve these tasks without difficulties, they present big challenges for some elderly persons and patients with hamstring injuries. We choose to study these motions and synthesize them on a robot, given its important health-care applications. One simple solution to achieve these tasks is to utilize static balance. The robot can increase the area of the support polygon by establishing new contact points, and then raise its body slowly while maintaining the COM within the support polygon. We choose not to use this strategy because in real life, we humans can perform these motions in a more agile fashion, and we hope that our controller can demonstrate comparable agility. For this reason, our controllers will utilize impulsive actions and take

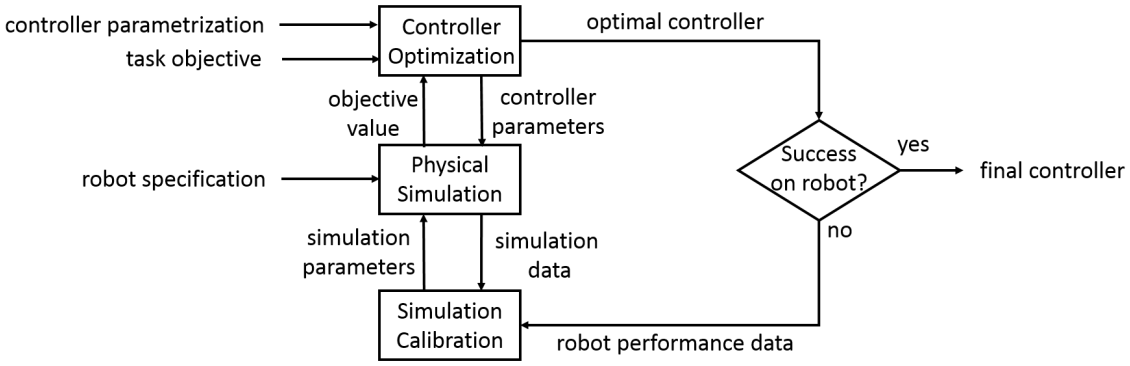

Figure 38: Overview of our algorithm.

advantage of the accumulated momentum to rise. In addition, since the main contribution of our method is simulation calibration, to best test its effectiveness, we only use feedforward controllers in our evaluations. Otherwise, if feedback controllers are used, the stability region of the tasks could be drastically increased, which makes the simulation accuracy less critical. Our results show that simulation calibration is effective to transfer the controllers from the virtual to the real environments for certain motion planning tasks. In all the test cases, at most two iterations of calibration is needed before the controllers work on the real robot.

6.2 Overview

We have developed a system that can automatically design motion controllers for robots (Figure 38). Given the specification of the robot, including its body shape, the physical properties of each body, and the types of joints, we build a physical simulation using Dynamic Animation and Robotics Toolkit (DART) [95]. In addition, we also incorporate into the simulation the torque limits, servo models and communication latency, which are often omitted in character animation. The controller optimization subsystem runs thousands of simulations to search for the optimal controller that maximizes the task-related fitness function. We then test this optimal controller on the robot. If the robot successfully completes the task, a working robotic controller is found and our algorithm terminates. Otherwise, we record the robot performance data and feed it into the simulation calibration subsystem. Simulation calibration runs another optimization, which searches for the optimal simulation

parameters to minimize the discrepancy between the performance of the robot in the simulation and in the real world. The loop of controller optimization and simulation calibration is performed iteratively until the controller works successfully on the real robot. In the next three sections, we will present the algorithmic details of these components.

6.3 Physical Simulation

6.3.1 Dynamics Equations

We model the robot as an articulated rigid body system in our simulator. We represent the states of the system $(\mathbf{x}, \dot{\mathbf{x}})$ in generalized coordinates, where $\mathbf{x}$ includes the global position $\mathbf{p}$, the orientation $\mathbf{r}$ of the root link, and the joint angles $\mathbf{q}$. We solve the governing equations of motion in generalized coordinates:

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) = \tau + \mathbf{J}^T\mathbf{f} \tag{38}$$

where $\mathbf{M}(\mathbf{x})$ is the mass matrix and $\mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})$ is the Coriolis and Centrifugal force. τ are joint torques exerted by the actuators. $\mathbf{J}$ is the Jacobian matrix and $\mathbf{f}$ is the external contact force, which is computed based on linear complementarity conditions. In our implementation, we use DART to compute the contact force and numerically integrate the system state $(\mathbf{x}, \dot{\mathbf{x}})$ over time.

6.3.2 Actuator Model

In character animation, the joint torque τ is often chosen as the control signal since the torques can be directly integrated in eq.(38). However, the control signal for the robot that we use in the experiments is the desired joint angles $\bar{\mathbf{q}}$. Given the difference between the desired and the current angle $q - \bar{q}$ of each joint, the servo first maps it to a corresponding power level U that is equivalent to changing the voltage across the motor and the voltage is eventually converted to the joint torque τ according to the internal actuator dynamics.

We choose BIOLOID GP as our robot platform. BIOLOID GP is a humanoid robot that consists of 18 degrees of freedom powered by Dynamixel AX-12/AX-18 servos. All the actuators on the lower body of the robot are Dynamixel AX-18, which use the following mapping (Figure 39) between the joint angle difference $q - \bar{q}$ and the power level U. The

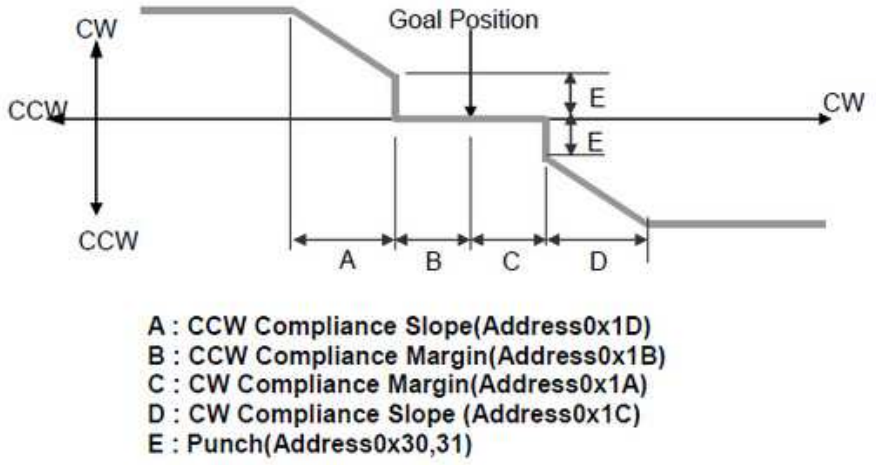

Figure 39: The mapping between $q - \bar{q}$ and U for an AX-18 actuator. This figure is from the user manual of Dynamixel AX-18 Actuator [130]. The x-axis is $q - \bar{q}$ while the y-axis is U.

intervals A and D determine the slope of the actuator response for counter-clockwise and clockwise motions respectively. Smaller values mean steeper response slopes, in which case the actuator follows the desired angle more closely. However, too small a value can lead to overshooting problems. B and C are the compliance margins. If the error of angle is within a small margin specified by B and C, the servo does not output any torque. E, the punch, is the minimum power level before the servo shuts down. In practice, we set A and D to be the same so that the simulated servo will behave the same no matter it rotates clockwise or counter-clockwise. In addition, since B, C and E are very small compared to A and D, we ignore their effects and approximate the mapping as linear within the intervals $q - \bar{q} \in A \bigcup B \bigcup C \bigcup D$ with the slope k_e:

$$U = k_e(q - \bar{q}) \tag{39}$$

To derive the relation between the power level U and the output torque τ, we adopt a model for the ideal DC motor [132]. It is valid to assume an ideal model because the AX-18 servos use high-quality DC motors. The derivation follows by considering the power balance in the motor at a constant voltage U:

$$P_{electric} = P_{mechanic} + P_{heat} \tag{40}$$

where $P_{electric}$ is the electrical power, $P_{mechanic}$ is the mechanical power, and P_{heat} is the

power dissipated as heat. From eq.(40), we can get the following relation:

$$UI = \dot{q}\tau_{motor} + RI^2 \tag{41}$$

where I is the current and R is the motor winding resistance. In an ideal DC motor, the torque is linearly proportional to the current $\tau_{motor} = k_\tau I$. Plugging it into the above equation, we arrive at the relation between the voltage U and the total torque generated by the motor τ_{motor}:

$$U = k_\tau \dot{q} + \frac{R}{k_\tau}\tau_{motor} \tag{42}$$

where k_τ is the torque constant, which is determined by the hardware design of the motor. The total torque generated by the motor is not yet the output torque that drives the motor shaft due to the friction inside the motor. The total torque can be decomposed into the output torque τ and the friction torque τ_f.

$$\tau_{motor} = \tau + \tau_f \tag{43}$$

The friction torque can be further divided into viscous friction and Coulomb friction [132]:

$$\tau_f = k_v \dot{q} + k_c \operatorname{sgn}(\dot{q}) \tag{44}$$

where k_v and k_c are friction coefficients for the viscous and Coulomb friction respectively. $\operatorname{sgn}(x)$ is the sign function that equals 1 if x is positive, -1 if x is negative and 0 otherwise.

Combining eq.(42), (43) and (44), we get the relation between the error of the joint angle $q - \bar{q}$ and the output torque τ.

$$\begin{aligned} \tau &= \frac{k_\tau k_e}{R}(q - \bar{q}) + (-k_v - \frac{k_\tau^2}{R})\dot{q} - k_c \operatorname{sgn}(\dot{q}) \\ &= -k_p(q - \bar{q}) - k_d \dot{q} - k_c \operatorname{sgn}(\dot{q}) \end{aligned} \tag{45}$$

where $k_p = -\frac{k_\tau k_e}{R}$ and $k_d = k_v + \frac{k_\tau^2}{R}$. We call these values k_p, k_d and k_c the *actuator gains*. It is possible to compute these actuator gains if the related parameters are given in the specification sheet of the motor. Plugging eq.(45) into (38), and taking torque limits

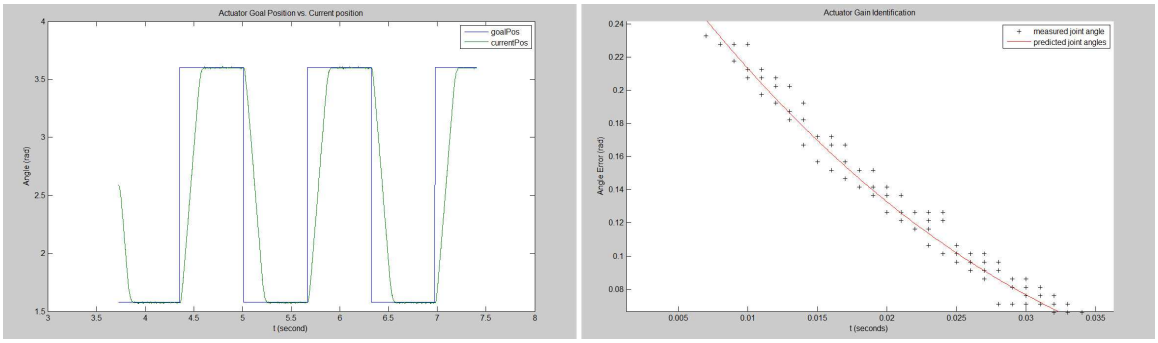

Figure 40: Actuator Identification. Left: the time series of input desired joint angle and the measured joint angle for an AX-18 servo. Right: the time series of actual error of joint angle and the predicted error using the identified actuator gains.

$[\tau_{min}, \tau_{max}]$ into consideration, we get the dynamics equation that use the desired joint angles as the control signal.

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) = \tau + \mathbf{J}^T\mathbf{f}$$

where

$$\tau = \begin{cases} \tau_{min} & \text{if } \tau < \tau_{min}, \\ \tau_{max} & \text{if } \tau > \tau_{max}, \\ -k_p(\mathbf{q} - \bar{\mathbf{q}}) - k_d\dot{\mathbf{q}} - k_c\,\mathrm{sgn}(\dot{\mathbf{q}}) & \text{otherwise.} \end{cases}$$

Actuator Gain Identification. We design robot experiments to identify the actuator gains k_p, k_d and k_c, since the specification of the servos does not provide the necessary information to compute them. In the experiment, we clamp the entire robot on a table except for the left foot. We then send a periodic control signal $\bar{q}(t)$ to the servo at the left ankle (blue curve in Figure 40 Left). The desired joint angle stays at the maximum value for 0.67 second, then changes to the minimum value and stays for another 0.67 second and repeats. We record the trajectory of the actual joint angle $q(t)$ through the experiment (green curve in Figure 40 Left). We manually segment out portions of these two curves where the power level is approximately linear to the error $\Delta q = q - \bar{q}$ (the union of intervals A, B, C and D in Figure 39). The black "+" in Figure 40 Right shows this error over time $\Delta q(t)$ in a typical segment.

Given $q(t)$ and $\bar{q}(t)$, we can apply regression to estimate the actuator gains. From eq.

(45), we have

$$\ddot{q} = I^{-1}(-k_p \Delta q - k_d \dot{q} - k_c \operatorname{sgn}(\dot{q})) \tag{46}$$

where I is the moment of inertia of the foot with respect to the rotating axis. The above equation is derived by plugging into $\tau = I\ddot{q} + \dot{I}\dot{q}$ and the fact that $\dot{I}\dot{q} = 0$ because the foot is a rigid body that rotates along a fixed axis. Ideally, $\ddot{q}$ and $\dot{q}$ can be computed using finite difference. However, the measurement of $q(t)$ is too noisy and finite difference would greatly magnify the noise. To solve this problem, we first smooth $q(t)$ by performing a 4th-order polynomial regression:

$$\min_{a,b,c,d,e} \int ||q(t) - (at^4 + bt^3 + ct^2 + dt + e)||^2 \mathrm{d}t \tag{47}$$

where a, b, c, d, e are the polynomial coefficients. This regression gives us a smooth analytical expression of $q(t)$. We then compute $\ddot{q}$ and $\dot{q}$ by differentiate this polynomial analytically:

$$\dot{q}(t) = 4at^3 + 3bt^2 + 2ct + d \tag{48}$$

$$\ddot{q}(t) = 12at^2 + 6bt + 2c \tag{49}$$

Combining eq. (47), (48) and (49), we can perform another regression to compute the actuator gains.

$$\min_{k_p,k_d,k_c} \int ||\ddot{q}(t) - I^{-1}(-k_p(q(t) - \bar{q}(t)) - k_d\dot{q}(t) - k_c \operatorname{sgn}(\dot{q}(t)))||^2 \mathrm{d}t \tag{50}$$

Our experiments and computation show that the actuator gains are $k_p = 9.272(N \cdot m/rad)$, $k_d = 0.3069(N \cdot m \cdot s/rad)$, and $k_c = 0.03(N \cdot m)$. To verify the correctness of these values, we plug them into the simulator and repeat the same experiment in the simulation. The red curve in Figure 40 Right is the error over time predicted in our simulation, which agrees well with the data that was collected from the robot experiment.

Latency. To guarantee the stability of the simulation, we use 1ms as the simulation time step. Many animation systems use the same simulation and control frequency, which means that a control signal $\bar{\mathbf{q}}$ is updated every simulation time step. However, the average latency of the whole control loop on our robot is 16ms. It is measured by a timer in our program

between the time that the program starts sending the actuator commands to the robot and the time that it finishes reading the sensor measurements from the robot. To better match our simulation with the actual latency, we choose to only update the control signal every 16 time steps.

6.4 Controller Optimization

Given the physical simulation, we can design controllers to enable the robot to achieve various motion planning tasks in the simulated environment. The four tasks that we use to test our system are rising from a leaning, sitting or kneeling position to an erect stance, and flipping from a standing to a handstanding pose. For each task, the joint configuration of the initial pose and the final pose are provided by the user. The goal of controller optimization is to find a sequence of control signals $\bar{q}(t)$ so that the robot can move from the initial to the final pose without losing balance. We purposefully choose to use only feedforward controllers[1] in this work, which means that the control signal $\bar{\mathbf{q}}(t)$ is a only function of time t and does not depend on the states of the robot. With the feedforward control alone, the controller transfer can only succeed if the simulation is close enough to the real-world environment. This will put the simulation calibration subsystem into more thorough tests.

[1]There is still an internal feedback loop in the actuators to track the desired joint angle (See Chapter 6.3.2). However, this feedback loop is not fully programmable.

We first formulate a trajectory optimization problem for each task.

$$\max_{\bar{\mathbf{q}}(t),T} V_{ctrl}(\mathbf{x}(t)) \tag{51}$$

subject to

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) = \tau + \mathbf{J}^T\mathbf{f} \tag{52}$$

$$\tau = \begin{cases} \tau_{min} & \text{if } \tau < \tau_{min}, \\ \tau_{max} & \text{if } \tau > \tau_{max}, \\ -k_p(\mathbf{q} - \bar{\mathbf{q}}) - k_d\dot{\mathbf{q}} - k_c\,\mathrm{sgn}(\dot{\mathbf{q}}) & \text{otherwise.} \end{cases} \tag{53}$$

$$\bar{\mathbf{x}}(0) = \mathbf{x}_0 \tag{54}$$

$$\bar{\mathbf{q}}(t) = \mathbf{q}_T, \text{if } t \geq T \tag{55}$$

This optimization searches for the duration T of the rising motion and the trajectory of the desired joint configuration $\bar{\mathbf{q}}(t)$ to maximize a task-related fitness function V_{ctrl}, and subject to physical constraints (eq.(52) and (53)) and boundary conditions (eq.(54) and (55)). $\mathbf{x}_0$ is the initial condition, and $\mathbf{q}_T$ is the final pose, both of which are provided by the user. Note that although we can specify the global translation $\mathbf{p}_0$, rotation $\mathbf{r}_0$ and joint angles $\mathbf{q}_0$ in the initial condition, we can only specify the desired joint angles for the final pose because the global translation and rotation are determined by the physical simulation.

Assuming no interbody collision happens when the robot executes $\bar{\mathbf{q}}(t)$, in our tasks, the robot can always reach the final pose $\mathbf{q}_T$ within a small error due to the weight of the bodies that each actuator supports. The criterion of success for all the tasks is whether the robot remains upright at the end of its motion. We use the following fitness function to reward controllers that keep balance throughout the entire motion.

$$V_{ctrl}(\mathbf{x}(t)) = \int_0^{T+1} \frac{1}{\alpha(t) + \epsilon} \mathrm{d}t \tag{56}$$

where $\alpha(t)$ is the angle between the up direction in the local frame of the robot's torso and the up direction in the global frame $(0, 0, 1)$. It measures how far the robot is from losing its balance. ϵ is a small positive number to prevent the denominator from being zero. We choose $\epsilon = 0.1$ in all our tasks. Note that the upper limit of the integration is $T + 1$. The

extra one second is to wait for the robot to settle down. We use the time horizon $T + 1$ because it is still possible that the robot can fall during the settling down phase and our fitness function will penalize this situation.

Two difficulties remain to solve the above optimization. First, the size of the optimization is large, which makes it computationally expensive to solve. Since our robot has 18 degrees of freedom and a rising motion can take a few seconds, the above space-time optimization problem can easily have hundreds to thousands of variables. Instead of directly searching this high dimensional space, we parameterize the controllers to make the computation tractable. Although there are many ways that we can parameterize the control space, designing the most effective control parametrization is not the focus of this work. In fact, we intentionally choose to use a simple parametrization to highlight the effect of our simulation calibration. We use a sparse set of keyframes $\bar{\mathbf{q}}_1, \bar{\mathbf{q}}_2, ..., \bar{\mathbf{q}}_n$ to parameterize the trajectory of the desired poses $\bar{\mathbf{q}}(t)$. In between the keyframes, we linearly interpolate the poses from two adjacent keyframes. With this simplification, the *control parameters* that we need to optimize reduce to only a few keyframes and the time interval between adjacent keyframes. We further halve the size of the problem by exploiting the symmetry of the motion. We find that all four tasks can be achieved with symmetric motions. Thus we constrain that the joint motions on the left bodies mirror those on the right bodies. In addition, since we do not want the robot to use its hands to help standing up, we freeze the joints at the shoulders and the elbows. This focus the controller to the motions of the lower body, which further reduces the size of the optimization problem.

The second difficulty is that during the motion, discrete contact events can happen frequently. They invalidate the gradient information, which imposes additional challenges for continuous optimization algorithms. We choose to use Covariance Matrix Adaptation (CMA) [55] to optimize the control parameters. Starting from an initial Gaussian distribution, CMA samples this distribution for a set of control parameters, evaluates them using physical simulations, discards the inferior samples and updates the distribution according to the remaining good samples. With a number of iterations, the distribution moves and shrinks, and eventually converges to a good controller parameter that can successfully fulfill

the task in the simulation.

6.5 Simulation Calibration

Although the optimal controllers $\bar{\mathbf{q}}(t)$ can work effectively in the simulation, they may fail to achieve the tasks when used on the robot due to the Reality Gap. We develop a simulation calibration subsystem, whose goal is to reduce the discrepancy between the simulated results and the real robot performance. This significantly increases the chance that the controller optimized in the simulation can be transferred to the real robot. In this subsystem, we formulate an optimization (eq. (57)) that searches for the *simulation parameters* θ so that the *discrepancy* E_{cali} is minimized between the simulated results and the robot performance in the real environment.

$$\min_{\theta} E_{cali} \tag{57}$$

Many parameters need to be set before a physical simulation starts, for example, the mass, the moment of inertia, the COM of each body segment, the coefficient of restitution and friction, and the gains of the actuators. The accuracy of a physical simulation heavily relies on the correctness of these parameter settings because changing simulation parameters can drastically alter the simulation results. Usually, simulation parameters can be set according to the specification sheet of the robot. However, we find that many of these parameters are incorrect. For example, from the CAD file, we calculate the total mass of the robot to be less than 1.1kg, but our own measurement using a scale reads 1.5kg. Furthermore, the height of the COM differs more than 1cm between CAD file and our measurement. This difference in parameters could be due to the manufacturing errors and the weight of cables, glues, nuts and bolts that were used in assembling the robot. Instead of trusting these simulation parameters, we decide to adjust them during simulation calibration.

We improve the simulation accuracy by minimizing the discrepancy E_{cali}, which is defined as the difference between the state trajectories in the simulation and those collected in the real robot experiment when the same controller is used in both scenarios. Recall that

our entire algorithm is an iterative process. At the nth iteration, the optimal controller $\bar{\mathbf{q}}_n(t)$ produces the state trajectory $\mathbf{x}_n(t)$ in the simulation and $\tilde{\mathbf{x}}_n(t)$ on the real robot[2]. Together with the $n-1$ pairs of optimal controllers $\{\bar{\mathbf{q}}_i(t)\}_{i=1,\ldots,n-1}$ and their associated state trajectories $\{\bar{\mathbf{x}}_i(t)\}_{i=1,\ldots,n-1}$ from previous iterations, we compute the discrepancy using the following expression.

$$E_{cali} = \frac{1}{n}\sum_{i=1}^{n}\int_{0}^{T+1} ||\tilde{\mathbf{x}}_i(t) - \mathbf{x}_i(t)||^2_{\mathbf{W}}\mathrm{d}t \tag{58}$$

where $\mathbf{W}$ is a diagonal weight matrix, which encapsulates the relative importance of each joint. Due to the complex interplay between the simulation results and the simulation parameters, the optimization (57) is nonlinear and nonconvex. Similar to controller optimization, we choose to use CMA as the optimization solver. In this case, each CMA sample is a candidate set of simulation parameters θ. To evaluate each CMA sample, we set the parameters θ in the physical simulator, execute the controllers $\{\bar{\mathbf{q}}_i(t)\}_{i=1,\ldots,n}$ to simulate the robot motions $\{\mathbf{x}_i(t)\}_{i=1,\ldots,n}$, and then compute the objective function eq. (58).

In our work, we initialize the simulation parameters as follows. We set the physical properties of each body segment, including the mass, the moment of inertia and the COM according to the CAD files. We set the actuator gains based on the measurement from experiments (Chapter 6.3.2). We leave all other parameters as default values in DART. Although these parameters are not accurate, they serve as a good initial guess. During simulation calibration, we search the parameter space within a bounded range centered at the initial guess. In addition, we also employ two simplifications to speed up the optimization. First, we manually select the most relevant simulation parameters that need to be optimized. All our tasks are to achieve a specific final pose while keeping balance. Since accurate actuator gains determine whether the robot can reach and hold the final pose, and correct COM's play an important role in balance control, we decide that the actuator gains of servos and the COM of each body are the most important simulation parameters in our

[2]we use the average of the multiple trajectories as $\tilde{\mathbf{x}}_n(t)$ in the robot experiment because even with the same controller, we can get slightly different trajectories due to the varied initial conditions, the noise from the sensor, from the actuator and from the environment.

case. This manual selection drastically reduce the search space of the optimization. Second, we zero out most of the diagonal entries of $\mathbf{W}$ in eq. (58) except for the rows corresponding to the global orientation. More specifically, we measure the discrepancy based solely on α, the angle between the up direction in the local frame of the robot's torso and the up direction in the global frame $(0, 0, 1)$.

$$E_{cali} = \frac{1}{n} \sum_{i=1}^{n} \int_{0}^{T+1} (\tilde{\alpha}_i(t) - \alpha_i(t))^2 \mathrm{d}t \tag{59}$$

The objective function eq. (59) captures the most important features that characterize the success or failure of our tasks (see eq.(56)), and eliminates the tedious manual tuning of the weight matrix $\mathbf{W}$.

6.6 Results

In this section we present the results of our system. We use four motion planning tasks to test our system: The robot rises from a leaning, sitting or kneeling position to an erect stance and flipping from a standing to a handstanding pose. Please watch the accompanying video[3] for the robot performance in the simulation and in the real world. Our system was implemented in C++, and we used DART with our actuator model to simulate the physics of the robot and its surrounding environments. The entire system runs on a laptop with 2.6GHz quad-core CPU and 16GB of memory. In controller optimization and simulation calibration, the CMA uses 32 samples per iteration and at most 50 iterations. We implemented a parallel version of CMA that can distribute the computation across all four cores on the CPU. It takes less than 15 minutes to find an optimal solution in controller optimization or simulation calibration.

We used the BIOLOID GP, a humanoid robot that consists of 18 degrees of freedom, in our experiments. The communication between the PC and the robot is through a serial port. To control the robot, a host program on the PC writes the desired pose $\bar{\mathbf{q}}$ to the serial port that is connected to the robot. A separate program that runs on the robot's onboard microprocessor listens to this port and sends the desired joint angle to each actuator. At the

[3] https://dl.dropboxusercontent.com/u/36899427/controllerTransfer.mp4

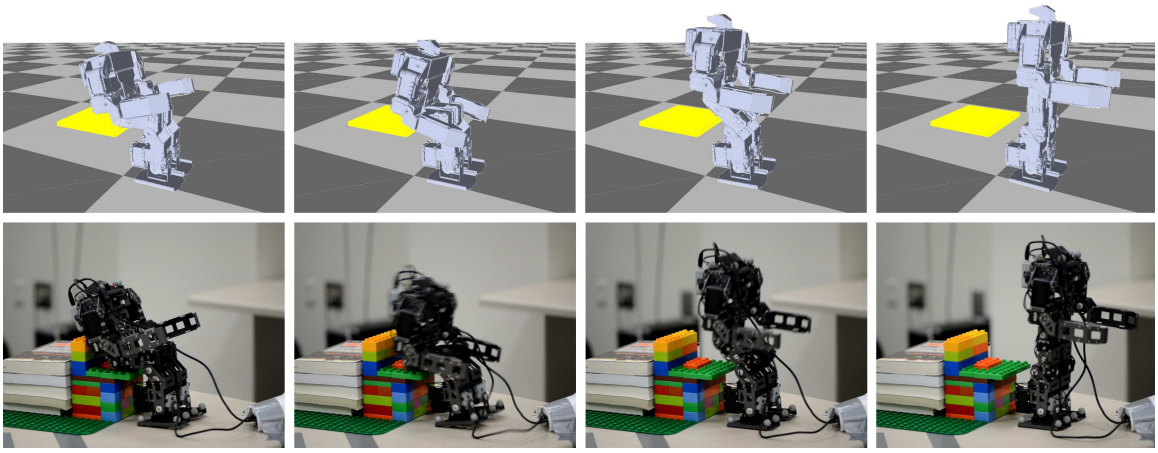

Figure 41: The results of the sit-to-stand task in the simulation and on the real robot.

same time, the robot performance data $\tilde{\mathbf{x}}$ is measured and sent back to the computer. We use onboard rotary encoders to measure joint angles and a VICON motion capture system to measure the global position and orientation of the robot's torso.

6.6.1 Rising from a Sitting Position

The first task that we have tested is to rise from a sitting pose to a standing pose (Figure 41). The initial and final poses q_0 and q_T are shown in the leftmost and rightmost images in Figure 41. We parameterize the controller with three keyframes q_0, q_1 and q_T. In addition to q_0 and q_T specified by the user, the controller optimization subsystem needs to search for the keyframe q_1, and the two time intervals t_1 and t_2 between the three keyframes. Note that we only change the joint angles of the hips and the knees and keep all other joints motionless throughout the entire motion.

We purposefully choose the initial pose that the legs of the robot extend forward and the projection of the robot's COM in the vertical direction falls far behind the contact points of the feet. If the robot simply extends the hips and the knees to stand up, it will fall backwards. Despite this challenging setup, our system successfully finds a controller that enables the robot to stand up in the simulation. Figure 41 shows that the robot first builds up a forward momentum by quickly leaning its upper body to the front. It then starts to extend the hips and the knees at the moment when the COM is approaching the boundary of the support polygon spanned by the feet. This effective standing-up strategy is found

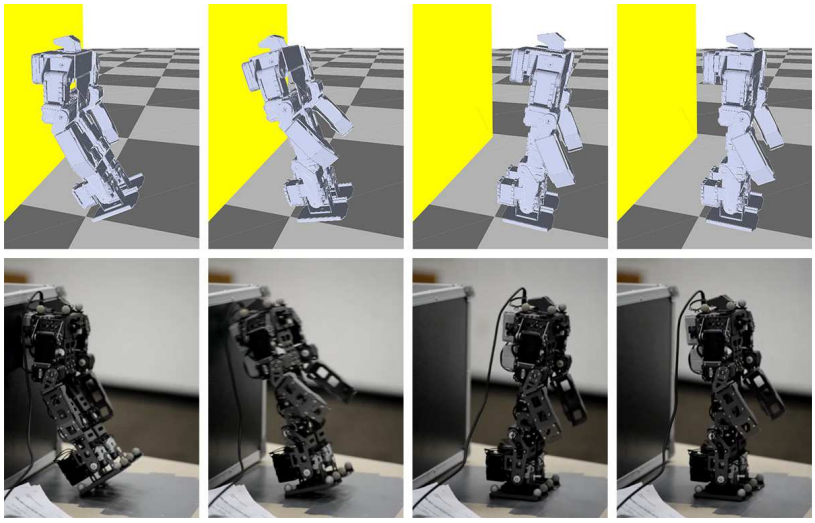

Figure 42: The results of the lean-to-stand task in the simulation and on the real robot.

automatically by the controller optimization subsystem.

When applying this controller to the real robot, we are surprised to find that it works directly, without the need of simulation calibration. The robot stands up from a chair in the same way as its simulated counterpart does in the virtual world. This shows that the Reality Gap is not always a problem. In some tasks, the stability region of a controller is so large that it can make the discrepancy between the virtual and the real world less critical.

6.6.2 Rising from a Leaning Position

In this task, the robot needs to rise from leaning on the wall (the leftmost image in Figure 42) to a standing position (the rightmost image in Figure 42). In the initial configuration, the hip joints are bent and they are straightened out in the final configuration while all other joints do not move. The initial and the final poses are the only two keyframes for this task.

The goal of controller optimization is to find an appropriate time interval T between these two keyframes. If the time inteval is too long, the robot moves slowly, and cannot accumulate enough momentum to rise up. If this time inteval is too short, the robot move abruptly, which will cause the upper body to bounce off the wall too quickly and fall forward. Without simulation calibration, the optimization cannot find a working controller for this task. The robot cannot rise up when $T \leq 0.10s$ and overshoots when $T > 0.10s$.

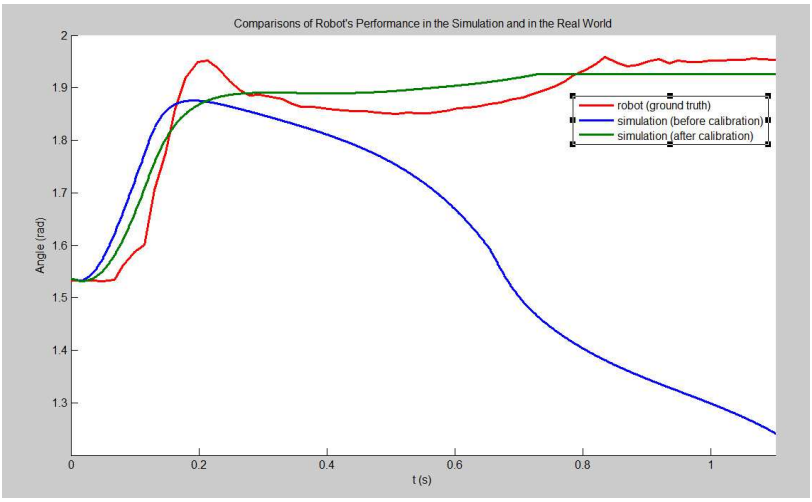

Figure 43: Comparisons of the robot's global orientation over time in the simulation (before/after calibration) and in the real environment.

According to the objective function value, the optimal controller, which still fails the task, uses $T = 0.11s$ to move from the initial to the final pose. Using this controller, the robot rises up too quickly and falls forward in the simulation. When we apply this controller to the real robot, we find that its performance in the real world differs drastically from that in the simulation. Rather than falling forward, the robot in the real world cannot rise up. The red and blue curves in Figure 43 show the the trajectories of the robot's global orientation in the simulation and in the real world. After one iteration of simulation calibration, the discrepancy is greatly reduced (Figure 43 green curve). We optimize the controller again in the calibrated simulator. This time, the optimal controller works both in the simulated and in the real environment.

We are able to transfer the controllers from the simulation to the real environment with only one iteration of simulation calibration. To better understand how simulation calibration works over multiple iterations, we perform an additional evaluation. Figure 44 shows the fitness functions after different number of iterations of simulation calibration. The blue curve is the fitness function on the real robot by varying the control parameter T in the range of $[0, 0.11]$. It serves as the ground truth. The fitness function stays at a high value when $T \in [0, 0.1]$, which means that the real robot can successfully rise if the controller uses less than 0.1s to change the pose from the initial to the final configuration.

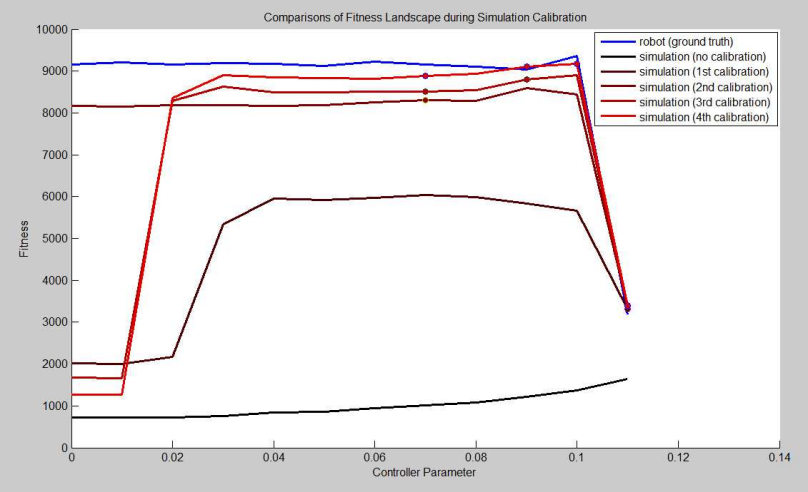

Figure 44: Comparisons of the fitness functions as more and more iterations of simulation calibration are performed.

In contrast, without simulation calibration, the fitness function (lowest black curve) stays at a low value for the entire control space. In other words, no controller exists that can make the robot stand up in the simulation. The gap between the blue and the black curves shows the difference of controller performance between in the simulation and in the real world. One iteration of system calibration brings the fitness function in the simulation towards the ground truth. As more iterations are performed, the fitness function in the simulation (brown and red curves) gradually approaches the ground truth, and the difference of the controller performance shrinks in this process. Note that a large discrepancy still exists in the region of the parameter space where $T < 0.02s$. This is probably caused by two reasons. First, in the region of $T < 0.02s$, the torque output of the servo is at its limit but the torque limit is not considered in simulation calibration. Second, the controllers and the data (the red circles in Figure 44) that we use in simulation calibration concentrate on the right half of the parameter space, which makes it difficult to generalize to a region where the data is scarce ($T < 0.02$). However, this could be beneficial in many applications because the computational resource is focused at the important regions near the successful controllers.

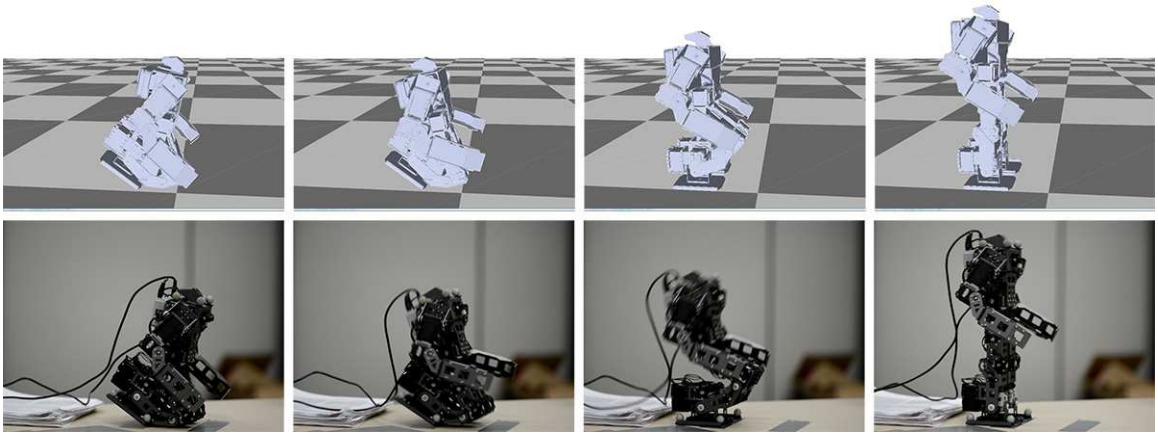

Figure 45: The results of the kneel-to-stand task in the simulation and on the real robot.

6.6.3 Rising from a Kneeling Position

Figure 45 shows that the robot stands up from a kneeling pose. Between the user-specified initial and final poses, the controller consists of two additional keyframes. The optimization needs to search for these keyframes and the time intervals between adjacent keyframes. Similar to other examples, we only allow the joints on the lower body of the robot to move. The controller optimized in the simulation demonstrates an agile getting-up motion: The robot first leans its upper-body backwards. As its COM is moving to the back, it quickly bends the hip, flexes its ankles and stands up. This entire motion resembles one of the most agile ways that we human get up from a kneeling position when we do not use our hands for additional support. Although this controller works perfectly in the simulation, the robot falls backward in the real world. After simulation calibration, the performance of the simulated robot comes closer to the real world scenario: The robot also falls backward in the simulation. Using the calibrated simulator, we optimize a new controller, with which the robot can successfully stand up from the kneeling position in the real world (Figure 45).

6.6.4 Flipping to a Handstand Position

We test our system with a challenging gymnastic action: flipping to a handstand position from a standing pose (Figure 46). There are two unique challenges in this task. First, the speed and the curvature of the initial arching motion is crucial and only a narrow range of such speed and curvature can lead to a balanced handstand. Second, the USB cable that

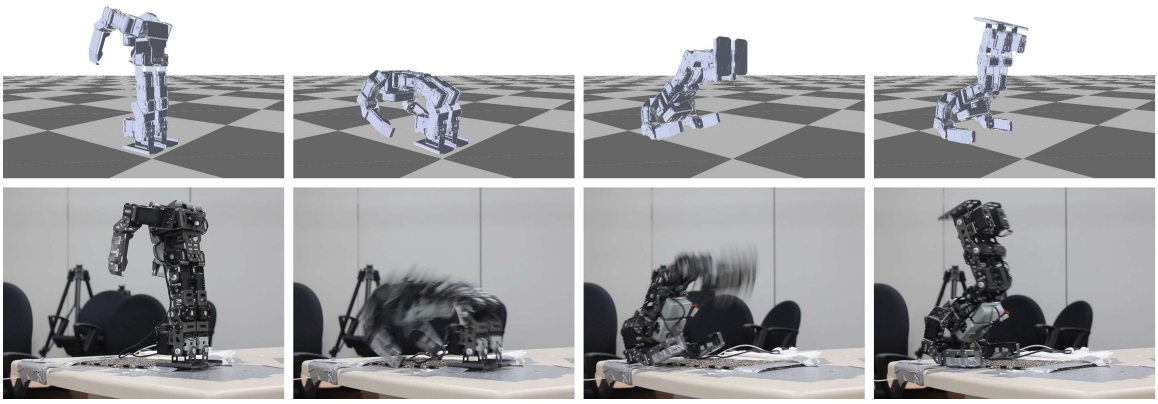

Figure 46: The results of the stand-to-handstand task in the simulation and on the real robot.

connects the robot to the computer will inevitably hit the ground during the backflip, which injects a strong perturbation that is not modeled in our simulation.

With two iterations of simulation calibration and controller optimization, our system finds a successful controller that works both in the simulation and in the real world: The robot arches back rapidly and lifts its feet after the arms touch the ground. It shows that our system can automatically design controllers for challenging motion planning tasks, even with strong unmodeled perturbations.

6.7 Discussion

This chapter has presented an end-to-end solution to automatically design motion controllers for robots. This solution consists of a set of computational tools: a simulation tool that simulates the dynamics of the robot and its environment, an optimization tool that automatically searches for a controller in the virtual environment and a calibration tool that improves the simulation accuracy to ease controller transfer from the virtual to the real world. This powerful system allows us to efficiently design controllers of a humanoid robot to achieve four different tasks, rising from leaning, sitting and kneeling poses to an erect stance, and flipping from a standing to a handstanding pose.

Since the main goal of this work is to demonstrate that the computational tools developed for character animation, including physical simulation and controller optimization, can be applied to robotics, the biggest challenge is to transfer the controllers developed

in the simulation to the real environments. In all the examples, at most two iterations of calibration is needed before we can successfully transfer the controller to the real robot. However, we want to emphasize that the goal of simulation calibration is not to find the true simulation parameters. Instead, it finds a set of parameters that reduce the discrepancy between the simulation and the real experiment for a specific motion planning task. We observe that the simulation parameters optimized for the task of lean-to-stand can be different from those of kneel-to-stand. In other words, the calibrated simulator is only valid for the current task and should not be used in a different task. It would be more interesting if simulation calibration can discover the true parameter value so that the calibrated simulator can be used for different tasks. We believe that this is possible if we use controllers and their corresponding robot performance data from multiple different tasks as the input of simulation calibration.

Crossing the Reality Gap is important for robotics because it can truly unleash the power of the computational tools and fundamentally change how robot controllers will be designed. Due to the variety and complexity of the causes of the Reality Gap, crossing it is an extremely challenging research problem. Our work only scratches the surface of this problem. A lot of future research need to be conducted in this area. For example, our motion planning tasks are relatively short, simple and do not require feedback controllers. We plan to extend our algorithm to more difficult tasks, such as walking and running. In our examples, we have shown that adjusting the COM and the actuator gains are enough, but other simulation parameters might also be important for a wider range of tasks. Including more simulation parameters and performing feature selection would be a promising direction for future work. In addition, some discrepancies between the simulation and the real world may not be explained by inaccurate simulation parameters alone. Unmodeled dynamics could also contribute to the discrepancy. Combining parametric and non-parametric models in simulation calibration for unmodeled dynamics is also an interesting avenue for future work.

CHAPTER VII

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this dissertation, we have presented a principled way to synthesize locomotion of humans and animals. Our algorithms can control characters of different morphologies to move efficiently and robustly in complex physically-simulated environments and to achieve challenging tasks. The key components of our algorithms are a set of powerful computational tools, including physical simulation and controller optimization. Although combining simulation and optimization is not a novel idea in motion synthesis, in contrast to prior work, we examine and identify those commonly-used simplifications that can affect the quality of the motions. We eliminate these simplifications by designing new simulation and optimization techniques. Our simulators are faster, more stable and more accurate. Our optimizator can search a higher-dimensional space with both continuous and discrete variables, which may lead to better optimal solutions. These computational tools make it possible to study a more diverse set of motions in nature than were previous possible in the character animation literature.

Chapter 3 described computational tools to study the diversity of swimming motions for aquatic creatures with different body shapes. This is made possible by an accurate swimming simulation and a powerful evolutionary optimization. Compared to the simplified fluid model, our swimming simulation solves the Navier-Stokes equations. It can capture important features of water, including incompressibility and vortices, that affect swimming strategies. In contrast to the traditional alternating two-way coupling technique, our simulation solves the dynamic equations of fluids and articulated rigid bodies simultaneously. This increases the numerical stability and drastically speed up the computation. Simulating the hydrodynamic environment with Navier-Stokes equations introduce new challenges to the classical optimization algorithms. We demonstrated that CMA works well in this

scenario. As a result, our algorithms can discover the most efficient swimming gait for a given creature automatically, without any human intervention. Our results showed that the synthesized swimming motions agree well with those employed by real aquatic animals.

Chapter 4 presented computational tools to study locomotion of soft body characters without skeleton support. We developed a muscle model that worked seamlessly with the state-of-the-art FEM simulation for soft materials. This muscle model is inspired by muscle structures found in real soft body animals. It lowers the dimensionality of the control space and ensures that the overall motion is coordinated. We demonstrated how to use finite-horizon trajectory optimization to control the locomotion. The key to our success is to identify that the widely-used simplification that separates contact planning with controller optimization is not good enough to achieve a stable locomotion. To solve this problem, we formulated a QPCC and developed an efficient solver. Consequently, effective control strategies and natural locomotion emerge automatically from the optimization solution.

Chapter 5 demonstrated computational tools to study agile human motions on a bicycle. We developed the first reinforcement learning algorithm that allows a virtual human character to learn bicycle stunts in a physically simulated environment. The algorithm is so efficient that most of stunt actions are learned in hours, which is even faster than the best human stunt bikers. An important lesson we have learned in this work is that it is difficult to design a good controller parametrization manually, especially for challenging locomotion tasks. The common practice of using a fixed policy parametrization tuned by users can severely limit the power of policy search algorithms. We eliminated this restriction by using NEAT, an algorithm that can simultaneously optimize both the parametrization and the parameters of a neural network. Eventually, the virtual character learned to perform a wide variety of stunts automatically, without the tedious manual tuning of controller parametrizations.

Chapter 6 explored an efficient method to develop humanoid robot controllers for the tasks of rising from a leaning/sitting/kneeling position to an erect stance. We built an accurate physical simulation, optimized controllers in the simulation and transferred the controller to a real robot. We investigated several factors that lead to the Reality Gap

and demonstrated in several cases that this gap may be crossed with an improved physical simulation. We perform iterative simulation calibration using data collected from robot experiments. After a small number of iterations, the controller designed in a simulation can be successfully transferred to the robot. This work shows that it is possible to apply the computational tools that were developed for character animations to design robotic controllers. This is an important milestone towards a fully automatic computational framework that can design the next-generation robots with extensive agility and manoeuvrability.

7.2 Future Work

The work presented in this dissertation opens the door to many promising directions for future work. Some of these future directions might be accomplished in the short term by combining existing algorithms to study locomotion that is not investigated in this dissertation. For example, one interesting direction is to study swimming motions of soft body animals. Studying soft body locomotion in water may have more fundamental impacts than articulated swimming creatures (Chapter 3) or soft body locomotion on land (Chapter 4). Most of the soft body animals on our planet, such as squid, octopus, sea slug and jellyfish, reside in oceans and they present more diversity in terms of swimming gaits. Their flexible body shapes could enable more efficient propulsion mechanisms than those of animals with rigid skeletons. For example, a jellyfish was found as ocean's most efficient swimmer [46]. In addition, the surface of aquatic animals with skeletons is still highly deformable due to the presence of skins, ligament and muscles. It is more appropriate to model them as soft bodies rather than as articulated rigid bodies, as we did in Chapter 3. I believe that it would be promising to combine the approaches in Chapter 3 and 4 to synthesize swimming motions for soft body characters.

A medium term future project is to use deep neural networks in reinforcement learning. Although NEAT frees us from laborious manual tuning of neural network structure in Chapter 5, we still need to specify the state and the action space for each task. For example, to keep balance on a bicycle, states should include the center of mass or the bicycle leaning angle. However, they may not carry over to a different task. Manually designing states and

actions would not scale to more sophisticated characters, more complicated environments, or more challenging tasks. The way that we use these hand-engineered states and actions in reinforcement learning today is analogue to using HoG or SIFT features in computer vision a few years ago. Recent advance in computer vision has shown promising results, which use deep neural networks, such as autoencoder [166] or Restricted Boltzmann machine [59], to learn features automatically. I believe that an important next step for reinforcement learning in character animation is to employ similar techniques to automatically discover important features for different locomotion tasks.

In the long run, the fast development of numerical algorithms and computational power will enable radical increase in efficiency and accuracy of physical simulations. The improved simulations will shrink the the Reality Gap rapidly, which will make it much easier to transfer controllers from the simulation to the real world. As a result, we envision that the two separate research fields of character animation and robotics will eventually merge and the computational tools will be shared in both fields. This will inevitably trigger a fundamental revolution in robotics in the near future, which will rely heavily on the computational tools in controller design. Chapter 6 is an important step towards realizing this vision. Although it has shown promising results, we have not yet put it into a comprehensive test with more challenging tasks. One possible stress test is to use this method to transfer bicycle stunt controllers developed in Chapter 5 to a real robot. This test will provide us insights about the causes of the Reality Gap and help us to eventually cross it.

APPENDIX A

QPCC SOLVER

A.1 QPCC for Contact

The QPCC problem for contact modeling is:

$$\min_{\dot{\mathbf{p}}^{n+1},\mathbf{u},\mathbf{f}_\perp,\mathbf{f}_\parallel,\lambda} G(\dot{\mathbf{p}}^{n+1},\mathbf{u}) \tag{60}$$

subject to

$$\mathbf{u}_{lb} \leq \mathbf{u} \leq \mathbf{u}_{ub}$$

$$\widetilde{\mathbf{M}}\dot{\mathbf{p}}^{n+1} = \mathbf{f}_c + \mathbf{A}\mathbf{u} + \widetilde{\mathbf{f}}^n \tag{61}$$

$$\mathbf{0} \leq \begin{bmatrix} \mathbf{f}_\perp \\ \mathbf{f}_\parallel \\ \lambda \end{bmatrix} \perp \begin{bmatrix} \mathbf{N}^T\dot{\mathbf{p}}^{n+1} \\ \mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda \\ \mu\mathbf{f}_\perp - \mathbf{E}^T\mathbf{f}_\parallel \end{bmatrix} \geq \mathbf{0} \tag{62}$$

where $G(\dot{\mathbf{p}}^{n+1},\mathbf{u})$ is a convex quadratic objective function of next velocities $\dot{\mathbf{p}}^{n+1}$ and control variables $\mathbf{u}$, which are bounded by $\mathbf{u}_{lb}$ and $\mathbf{u}_{ub}$.

Equation 61 is the discretized dynamic equation (Equation 8 in the paper), where $\widetilde{\mathbf{M}}$ is the mass matrix with terms from implicit integrator, $\mathbf{f}_c$ and $\mathbf{A}\mathbf{u}$ are the contact force and control force (scaled by the time step) respectively, and $\widetilde{\mathbf{f}}^n$ accounts for all other terms in the dynamic equation.

Equation 62 is the LCP formulation to regulate contact velocity and contact force, $\mathbf{f}_c = \mathbf{N}f_\perp + \mathbf{D}\mathbf{f}_\parallel$, where $\mathbf{N}$ is the unit normal vector, $\mathbf{D}$ is a set of tangential directions at the contact point, and $f_\perp$ and $\mathbf{f}_\parallel$ are the magnitudes of normal and tangent forces. μ is the friction coefficient and λ is an auxiliary variable whose physical meaning is related to the tangent velocity of a sliding contact. This form is slightly more general than the QPCC formulation (Equation 11) in the paper.

A.2 Implementation of QPCC Solver for Contact

Algorithm 1: Pseudo-code of the QPCC solver.

```
1  (x*, f*) = Solve(QPCC)
2  begin
3      ithIter = 0;
4      f* ← ∞;
5      x* ← null;
6      priority_queue ← [];
7      visitedQP_set ← {};
8      CCSpec ← GenerateInitialGuess();
9      QP ← GenerateQP(QPCC, CCSpec);
10     (QP.minimizer, QP.fval) ← QP.Solve();
11     priority_queue.Enqueue(QP);
12     while visitedQP_set.Size() < maxNumVisitedQP and priority_queue.Empty() =
       False do
13         QP ← priority_queue.Dequeue();
14         visitedQP_set.Add(QP);
15         if QP.fval < f* then
16             f* ← QP.fval;
17             x* ← QP.minimizer;
18         childQP_list ← GenerateChildQP(QPCC, QP, CCSpec);
19         foreach childQP in childQP_list do
20             if visitedQP_set.Find(childQP) then
21                 continue;
22             (childQP.minimizer, childQP.fval) ← childQP.Solve();
23             priority_queue.Enqueue(childQP);
24         ithIter ← ithIter + 1;
25     return (x*, f*);
```

Algorithm 1 summarizes the implementation of our QPCC solver. The solver takes the QPCC (Equation 60-62) as input and outputs the minimizer $\mathbf{x}^*$ and minimum objective function value f^*. Our iterative QPCC solver starts with an initial guess of the contact situation, which is a set of linear constraints that are compatible with the complementarity conditions. Function *GenerateInitialGuess* in Line 8 generates such an initial guess. We

usually choose static contact situation as the initial guess:

$$\begin{aligned} \mathbf{0} \leq \mathbf{f}_{\perp}, \qquad & \mathbf{N}^T \dot{\mathbf{p}}^{n+1} = \mathbf{0} \\ \mathbf{0} \leq \mathbf{f}_{\parallel}, \qquad & \mathbf{D}^T \dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda = \mathbf{0} \\ \mathbf{0} = \lambda, \qquad & \mu \mathbf{f}_{\perp} - \mathbf{E}^T \mathbf{f}_{\parallel} \geq \mathbf{0} \end{aligned}$$

Occasionally, this initial guess is inconsistent with the dynamic constraints (Equation 61), resulting in an infeasible problem. In such a case, we solve the Mixed LCP problem (Equation 61 and 62), assuming $\mathbf{u} = \mathbf{0}$, to obtain a feasible initial guess. The output of *GenerateInitialGuess* is a boolean array *CCSpec* that specifies a set of linear constraints from the complementarity conditions. If the ith entry of *CCSpec* is True, we choose the linear constraints $Cond_1 = 0$ and $Cond_2 \geq 0$ for the ith pair of complementarity conditions $0 \leq Cond_1 \perp Cond_2 \geq 0$. If it is False, we choose $Cond_1 \geq 0$ and $Cond_2 = 0$.

Function *GenerateQP* in Line 9 generates a QP by replacing the complementarity conditions of QPCC with the linear constraints specified in *CCSpec*. In Line 10, we solve the initial QP and record its minimizer and minimal function value. We add this QP to an priority queue, which sorts the QP's by their function values in an increasing order. Line 12 starts the iteration, which terminates until the number of explored QP's exceeds a threshold or the priority queue is empty.

In each iteration, we retrieve the QP at the head of the queue (Line 13), add it to the set of explored QP's and update f^* and $\mathbf{x}^*$ if necessary (Line 14-17). Function *GenerateChildQP* generates a list of new QP's by pivoting complementarity conditions (Line 18), which we will discuss in the next paragraph. For each new QP in the list, if it has been already explored, we discard it (Line 20-21). Otherwise, we solve the new QP and add it to the queue (Line 22-23). When the algorithm terminates, we output the best minimizer and function value that we found during the iterations (Line 25).

We summarize the detail of *GenerateChildQP* in Algorithm 2. The input of *GenerateChildQP* includes QPCC, current QP and its constraint specification for complementarity conditions: *CCSpec*. In Line 4, we extract the solution for each contact from the minimizer

of the QP. Function *ExtractFromMinimizer* selects the correct components associated with a specific contact point from the minimizer. In Line 6, we identify the index of the normal force-velocity condition pair for the ith contact, where *IdInCC* returns the index of a specific pair among all complementarity conditions. Line 7 identifies the case of contact establishment and performs the corresponding pivoting (Line 8). Line 11 checks whether the contact

Algorithm 2: Generate a list of new QP's based on the minimizer of the current QP

```
1  QP_list = GenerateChildQP(QPCC, QP, CCSpec)
2  begin
3      foreach  i in Contacts.Size() do
4          (ṗ, f_⊥^i, f_‖^i, λ^i) ← ExtractFromMinimizer(QP.minimizer);
5          newCCSpec ← CCSpec;
6          id ← IdInCC(0 ≤ f_⊥^i ⊥ (N^i)^T ṗ ≥ 0);
7          if CCSpec[id] = True and (N^i)^T ṗ =0 then
8              newCCSpec[id] ← False;
9              newQP ← GenerateQP(QPCC, newCCSpec);
10             QP_list.Add(newQP);
11         else if CCSpec[id] = False and f_⊥^i =0 then
12             newCCSpec[id] ← True;
13             newQP ← GenerateQP(QPCC, newCCSpec);
14             QP_list.Add(newQP);
15         id ← IdInCC(0 ≤ λ_⊥^i ⊥ μf_⊥^i − (E^i)^T f_‖^i ≥ 0);
16         id_list ← IdInCC(0 ≤ f_‖^i ⊥ (D^i)^T ṗ + E^i λ^i ≥ 0);
17         if CCSpec[id] = True and μf_⊥^i − (E^i)^T f_‖^i = 0 then
18             newCCSpec[id] = False;
19             foreach id1 in id_list do
20                 newCCSpec[id1] ← True;
21             f ← D^i f_‖^i;
22             j ← argmax_k f^T(D^i.Col(k));
23             newCCSpec[j] ← False;
24             newQP ← GenerateQP(QPCC, newCCSpec);
25             QP_list.Add(newQP);
26         else if CCSpec[id] = False and λ^i = 0 then
27             newCCSpec[id] ← True;
28             foreach id1 in id_list do
29                 newCCSpec[id1] ← False;
30             newQP ← GenerateQP(QPCC, newCCSpec);
31             QP_list.Add(newQP);
32     return QP_list;
```

is about to break and pivots the constraint accordingly, which enables the contact breakage (Line 12). Line 15-16 identify the indices of friction cone and friction direction conditions associated with the ith contact. If the static friction force has reached the boundary of friction cone, we switch the contact situation from static to sliding (Line 17-18) and estimate the sliding friction direction using the static friction direction (Line 19-23). We apply the opposite pivoting (from sliding to static) when the sliding velocity reaches zero (Line 26-29). A new QP is generated under the new contact situation and added into the list of new QP's (Line 9-10, 13-14, 24-25 and 30-31) whenever pivoting happens. After processing through all the contact points, the function returns the new QP list.

A.3 Test Case

It is nontrivial to test and debug an implementation of a QPCC solver. We propose the following test case problem: A particle with mass m lies on the ground and it wants to jump up. The particle can only control a jumping force $f\mathbf{N}$, where $\mathbf{N} = (0, 1, 0)$ is the up direction. The magnitude of the jumping force is bounded by $0 \leq f \leq f_{ub}$. What is the optimal jumping force if the goal is to jump as high as possible? The answer is trivially using maximal force possible. However, the solution is not necessarily f_{ub} under some contact configuration. Our hope is that the QPCC solver can find the optimal contact configuration such that the optimal solution reaches f_{ub}.

One formulation of the above problem is:

$$\min_{\dot{\mathbf{p}}, f, f_\perp, \mathbf{f}_\parallel, \lambda} \; -f^2$$

subject to

$$0 \leq f \leq f_{ub}$$

$$m\dot{\mathbf{p}} = \Delta t(m\mathbf{g} + \mathbf{N}f_\perp + \mathbf{D}\mathbf{f}_\parallel + \mathbf{N}f)$$

$$\mathbf{0} \leq \begin{bmatrix} f_\perp \\ \mathbf{f}_\parallel \\ \lambda \end{bmatrix} \perp \begin{bmatrix} \mathbf{N}^T\dot{\mathbf{p}} \\ \mathbf{D}^T\dot{\mathbf{p}} + \mathbf{E}\lambda \\ \mu f_\perp - \mathbf{E}^T\mathbf{f}_\parallel \end{bmatrix} \geq \mathbf{0}$$

where Δt is the time step used in the simulation and $\mathbf{g}$ is gravity.

It is easy to verify that the minimizer is

$$(\dot{\mathbf{p}}, f, f_{\perp}, \mathbf{f}_{\parallel}, \lambda) = (\frac{\Delta t}{m}(m\mathbf{g} + \mathbf{N}f_{ub}), f_{ub}, 0, \mathbf{0}, 0)$$

and the optimal function value is $-f_{ub}^2$. A correctly implemented QPCC solver based on Algorithm 1 and 2 should converge to this optimal solution in two iterations. In the first iteration, an initial QP using static contact situation is generated and solved. The solution should be

$$(\dot{\mathbf{p}}, f, f_{\perp}, \mathbf{f}_{\parallel}, \lambda) = (0, |m\mathbf{g}|, 0, \mathbf{0}, 0)$$

The function *GenerateChildQP* will pivot the constraint to switch from static contact to contact breakage and generate a new QP. In the second iteration, the solution of this new QP is exactly the optimal solution of the QPCC problem.

Once the implementation passes this simple test case, more challenging cases should be tested. For example, control a single rigid body, an articulated rigid-body system, or a soft body with the presence of contact.

APPENDIX B

CONSTRAINTS IN ODE

We are going to describe how various constraints are formulated in ODE for completeness of presentation.

Joint constraints Joints that connect two rigid bodies constrain their relative motions. The hinge, universal and ball joints impose five, four and three constraints respectively, each of which is a linear equality constraint.

$$\mathbf{J}_A \begin{bmatrix} \mathbf{v}_A \\ \omega_A \end{bmatrix} - \mathbf{J}_B \begin{bmatrix} \mathbf{v}_B \\ \omega_B \end{bmatrix} = 0 \tag{63}$$

where $\mathbf{J}_A$ is a row in the Jacobian matrix that maps the velocities of body A to one component of the linear velocity at the joint position or the angular velocity perpendicular to the joint axes.

Actuator constraints An actuator is attached to each joint of the human character to enable it to actively control its joint motion. The actuators generate internal torques to track the desired pose given by the IK solver (See Figure 2 in the paper). The following linear equality constraint should be satisfied for each actuated degree of freedom (DOF).

$$\mathbf{n}^T(\omega_A - \omega_B) - \tilde{\dot{q}} = 0 \tag{64}$$

where $\mathbf{n}$ is the axis of the actuated DOF. $\tilde{\dot{q}}$ is the desired actuator angular speed, which is the difference between the desired and current DOF value, divided by the time step.

Contact constraints ODE uses the standard friction pyramid to model the contact forces between the bicycle and the ground. Let $f_\perp$, $f_\parallel^1$ and $f_\parallel^2$ be the normal and two tangential components of the contact force $\mathbf{f}_c$.

$$\mathbf{f}_c = f_\perp \mathbf{n} + f_\parallel^1 \mathbf{t}_1 + f_\parallel^2 \mathbf{t}_2$$

where $\mathbf{n}$ is the ground normal, $\mathbf{t}_1$ and $\mathbf{t}_2$ are the two orthogonal tangential bases of the ground.

Along the normal direction, the contact velocity and contact force satisfy the following linear complementarity constraints.

$$v_\perp \geq 0,\ f_\perp \geq 0,\ v_\perp f_\perp = 0 \tag{65}$$

The normal contact velocity $v_\perp$ can be calculated by multiplying the Jacobian at the contact location with the body velocities and then projected along the normal direction.

$$v_\perp = \mathbf{n}^T \mathbf{J} \begin{bmatrix} \mathbf{v} \\ \omega \end{bmatrix} \tag{66}$$

Along the tangential direction, one of the following friction cone conditions must be satisfied.

$$\begin{aligned} v_\parallel^i > 0, &\quad f_\parallel^i = -\mu f_\perp \\ v_\parallel^i < 0, &\quad f_\parallel^i = \mu f_\perp \\ v_\parallel^i = 0, &\quad -\mu f_\perp \leq f_\parallel^i \leq \mu f_\perp \end{aligned} \tag{67}$$

where $i \in \{1, 2\}$ and μ is the friction coefficient. The tangential velocities $v_\parallel^i$ can be calculated similar to eq. (66).

The dynamics equation, together with all the constraints (63), (64), (65) and (67), form a mixed linear complementarity program, which can be efficiently solved using a variant of Dantzig's algorithm [139].

APPENDIX C

IMPLEMENTATION DETAILS OF LEARNING BICYCLE STUNTS

C.1 Neural Network Structures

We used NEAT to search for both the topologies and the weights of neural networks for balance-driven tasks. In addition to the endo balance controller, which is shown in the paper, we demonstrate other learned neural networks in Figure 47. Note that NEAT performs feature selection (removing connections) for the controllers of balance and steering and back hop. It also performs structural complexification (adding nodes and connections) in the examples of the wheelie and riding a unicycle. We found that it is unintuitive to interpret these network structures, and this is a common problem of using neural networks.

C.2 Reward Functions

We summarize the reward functions that are used to learn different bicycle tasks in Table 8. We accumulate the rewards for 1000 time steps or until the bicycle loses its balance $|\alpha| > 0.5$ or when the stunt fails $|\Delta\beta| > 0.5$.

Table 8: Reward functions for different tasks.

task	reward function
momentum-driven	
going over curbs	β
endo (lifting)	$-\beta$
front wheel pivot	$\begin{cases} \dot{\gamma}\Delta t & \text{if the pivoting phase has started,} \\ 1 & \text{if the pivoting phase has ended,} \\ 0 & \text{otherwise.} \end{cases}$
bunny hop	$h_f h_r$
balance-driven	
balance and steering	$1 + \frac{1}{\Delta\theta+1}$
wheelie	$1 + \frac{1}{\Delta\beta+1} + \frac{1}{\psi+0.1} + \frac{2}{\alpha+0.1}$
endo (balance)	$1 + \frac{1}{\Delta\beta+1} + \frac{1}{\Delta v_f+0.1} + \frac{1}{\alpha+1} + \frac{1}{\theta+1}$
back hop	$1 + \frac{1}{y+0.1} + \frac{1}{z+0.1} + \frac{1}{x+0.1}$
high wheeler (stunt)	$1 + \frac{1}{\Delta\beta+1} + \frac{1}{\Delta v_f+1}$
unicycle	$1 + \frac{1}{\Delta v_r+1} + \frac{2}{\alpha+0.1} + \frac{1}{\chi+1}$

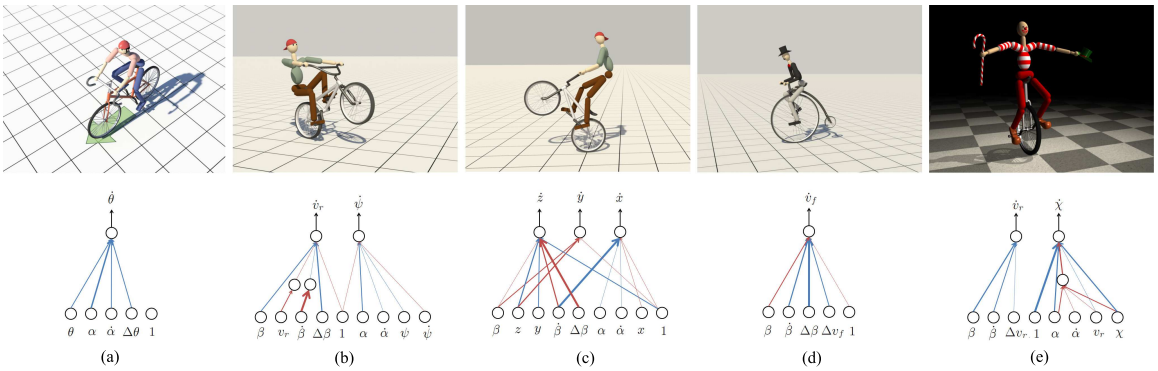

Figure 47: Balance-driven tasks and the neural network structures of their corresponding controllers: (a) balance and steering, (b) wheelie, (c) back hop, (d) riding a high wheeler (stunt) and (e) riding a unicycle. Red edges have positive weights and blue edges have negative weights. The thickness of the edges shows the relative magnitudes of the weights.

C.3 Bicycle Specifications

We describe the physical specifications of the three bicycles and the unicycle in Table 9.

C.4 Simulation and Optimization Parameters

We used Open Dynamic Engine as our physical simulator. Table 10 summarizes the simulation parameters used in our examples.

We applied CMA to search for the splines for the momentum-driven tasks. We used 90 samples per iteration, maximum of 50 iterations and the default values for other parameters [54].

We applied NEAT to search for the neural networks for the balance-driven tasks. The implementation can be found at Stanley [141]. Table 11 summarizes the parameters used

Table 9: Specifications of different bicycles and the unicycle. The mass unit is kg and the length unit is m. The distance between wheels only accounts for the horizontal distance.

specification	road bike	BMX bike	high wheeler	uni-cycle
mass of the frame	7.0	5.6	8.0	2.0
mass of the handlebar	3.5	3.5	4.0	NA
mass of the front wheel	1.7	0.71	5.0	NA
mass of the rear wheel	1.7	0.71	0.5	2.3
radius of the front wheel	0.32	0.25	0.45	NA
radius of the rear wheel	0.32	0.25	0.11	0.32
width of the tires	0.015	0.03	0.015	0.03
distance between wheels	1.03	0.85	0.58	NA

Table 10: Simulation parameters.

parameter	value
time step	0.01s
constraint force mixing (CFM)	10^{-10}
error reduction parameter (ERP)	0.99
friction coefficient (BMX bike examples)	2.0
friction coefficient (other examples)	1.0

in our NEAT optimization.

Table 11: NEAT parameters.

parameter	value
max number of iterations	50
population size	90
survival rate	0.2
crossover rate	0.7
mutation rate	0.2
chance of adding a link	0.05
chance of adding a node	0.05
chance of replacing a weight	0.1
max weight perturbation	0.5

REFERENCES

[1] Abbeel, P., Quigley, M., and Ng, A. Y., "Using inaccurate models in reinforcement learning," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), pp. 1–8, ACM, 2006.

[2] Abe, Y., da Silva, M., and Popović, J., "Multiobjective control with frictional contacts," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pp. 249–258, 2007.

[3] Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., and Kry, P. G., "Volume contact constraints at arbitrary resolution," *ACM Trans. Graph.*, vol. 29, pp. 82:1–82:10, July 2010.

[4] Allen, B. and Faloutsos, P., "Evolved controllers for simulated locomotion," in *Motion in Games, Lecture Notes in Computer Science*, pp. 219–230, 2009.

[5] Ando, R., Thuerey, N., , and Wojtan, C., "A Stream Function Solver for Liquid Simulations," *Transactions on Graphics (SIGGRAPH)*, p. 8, August 2015.

[6] Andrews, S. and Kry, P., "Goal directed multi-finger manipulation: Control policies and analysis," *Computers & Graphics*, vol. 37, no. 7, pp. 830 – 839, 2013.

[7] Anitescu, M. and Potra, F. A., "Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems," *Nonlinear Dynamics*, vol. 14, pp. 231–247, 1997.

[8] Arash, O. E., Genevaux, O., Habibi, A., and michel Dischler, J., "Simulating fluid-solid interaction," in *in Graphics Interface*, pp. 31–38, 2003.

[9] Bai, L., Mitchell, J. E., and Pang, J.-S., "On convex quadratic programs with linear complementarity constraints," *In submission*, 2011.

[10] Baraff, D. and Witkin, A., "Large steps in cloth simulation," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pp. 43–54, 1998.

[11] Barbič, J., da Silva, M., and Popović, J., "Deformable object animation using reduced optimal control," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, 2009.

[12] Barbič, J. and James, D. L., "Real-time subspace integration for St. Venant-Kirchhoff deformable models," *ACM Transactions on Graphics (SIGGRAPH 2005)*, vol. 24, pp. 982–990, Aug. 2005.

[13] Barbič, J. and Popović, J., "Real-time control of physically based simulations using gentle forces," *ACM Trans. on Graphics (SIGGRAPH Asia 2008)*, vol. 27, no. 5, pp. 163:1–163:10, 2008.

[14] Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G., "A finite element method for animating large viscoplastic flow," *ACM Trans. Graph.*, vol. 26, July 2007.

[15] Barrett, D., Grosenbaugh, M., and Triantafyllou, M., "The optimal control of a flexible hull robotic undersea vehicle propelled by an oscillating foil," in *Autonomous Underwater Vehicle Technology, 1996. AUV'96., Proceedings of the 1996 Symposium on*, pp. 1–9, IEEE, 2002.

[16] Bathe, K.-J., *Finite Element Procedures*. London: Prentice-Hall, 2007.

[17] Batty, C., Bertails, F., and Bridson, R., "A fast variational framework for accurate solid-fluid coupling," in *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[18] BBC, "Bicycle chosen as best invention," *BBC News*, 2005.

[19] Bergou, M., Mathur, S., Wardetzky, M., and Grinspun, E., "TRACKS: Toward Directable Thin Shells," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 26, pp. 50:1–50:10, jul 2007.

[20] Boeing, A. and Braunl, T., "Leveraging multiple simulators for crossing the reality gap," in *International Conference on Control Automation Robotics & Vision*, pp. 1113–1119, 2012.

[21] Bongard, J. C. and Lipson, H., "Nonlinear system identification using coevolution of models and tests.," *IEEE Trans. Evolutionary Computation*, vol. 9, no. 4, pp. 361–384, 2005.

[22] Boyan, J. A. and Moore, A. W., "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7*, pp. 369–376, MIT Press, 1995.

[23] Braun, S. and Mitchell, J. E., "A semidefinite programming heuristic for quadratic programming problems with complementarity constraints," *Computational Optimization and Application*, vol. 31, pp. 5–29, 2005.

[24] Bridson, R., Fedkiw, R., and Anderson, J., "Robust treatment of collisions, contact and friction for cloth animation," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pp. 594–603, 2002.

[25] Brochu, T., Batty, C., and Bridson, R., "Matching fluid simulation elements to surface geometry and topology," in *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, (New York, NY, USA), pp. 47:1–47:9, ACM, 2010.

[26] Carlson, M., Mucha, P. J., and Turk, G., "Rigid fluid: animating the interplay between rigid bodies and fluid," in *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, (New York, NY, USA), pp. 377–384, ACM, 2004.

[27] Carvallo, M. E., "Théorie du mouvement du monocycle et de la bicyclette," *Journal de L'Ecole Polytechnique*, vol. 5, 1900.

[28] Chambaron, S., Berberian, B., Delbecque, L., Ginhac, D., and Cleeremans, A., "Implicit motor learning in discrete and continuous tasks: Toward a possible account of discrepant results," *Handbook of Motor Skills: Development, Impairment, and Therapy*, pp. 139–155, 2009.

[29] Chentanez, N., Goktekin, T. G., Feldman, B. E., and O'Brien, J. F., "Simultaneous coupling of fluids and deformable bodies," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, (Aire-la-Ville, Switzerland, Switzerland), pp. 83–89, Eurographics Association, 2006.

[30] Collins, R. N., *A mathematical analysis of the stability of two-wheeled vehicles*. PhD thesis, University of Wisconsin, 1963.

[31] Coros, S., Beaudoin, P., and van de Panne, M., "Robust task-based control policies for physics-based characters," *ACM Trans. Graph.*, vol. 28, pp. 170:1–170:9, Dec. 2009.

[32] Coros, S., Beaudoin, P., and van de Panne, M., "Generalized biped walking control," *ACM Transctions on Graphics*, vol. 29, no. 4, p. Article 130, 2010.

[33] Coros, S., Karpathy, A., Jones, B., Reveret, L., and van de Panne, M., "Locomotion skills for simulated quadrupeds," *ACM Transactions on Graphics*, vol. 30, no. 4, 2011.

[34] Coros, S., Martin, S., Thomaszewski, B., Schumacher, C., Sumner, R., and Gross, M., "Deformable objects alive!," *ACM Trans. Graph.*, vol. 31, pp. 69:1–69:9, July 2012.

[35] da Silva, M., Abe, Y., and Popović, J., "Interactive simulation of stylized human locomotion," in *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, (New York, NY, USA), pp. 82:1–82:10, ACM, 2008.

[36] de Lasa, M. and Hertzmann, A., "Prioritized optimization for task-space control," in *International Conference on Intelligent Robots and Systems (IROS)*, October 2009.

[37] DiLorenzo, P. C., Zordan, V. B., and Sanders, B. L., "Laughing out loud: control for modeling anatomically inspired laughter using audio," in *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pp. 125:1–125:8, 2008.

[38] Eaton, M., *Evolutionary Humanoid Robotics*. Springer Publishing Company, Incorporated, 2015.

[39] Erleben, K., "Velocity-based shock propagation for multibody dynamics animation," *ACM Trans. Graph.*, vol. 26, June 2007.

[40] Fan, Y., Litven, J., Levin, D. I. W., and Pai, D. K., "Eulerian-on-lagrangian simulation," *ACM Trans. Graph.*, vol. 32, pp. 22:1–22:9, July 2013.

[41] Farchy, A., Barrett, S., MacAlpine, P., and Stone, P., "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pp. 39–46, International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[42] Fedkiw, R., Stam, J., and Jensen, H. W., "Visual simulation of smoke," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), pp. 15–22, ACM, 2001.

[43] Foster, N. and Metaxas, D., "Realistic animation of liquids," *Graph. Models Image Process.*, vol. 58, pp. 471–483, Sept. 1996.

[44] Geijtenbeek, T. and Pronost, N., "Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review," *Computer Graphics Forum*, vol. 31, no. 8, pp. 2492–2515, 2012.

[45] Geijtenbeek, T., van de Panne, M., and van der Stappen, A. F., "Flexible muscle-based locomotion for bipedal creatures," *ACM Transactions on Graphics*, vol. 32, no. 6, 2013.

[46] Gemmell, B., Costello, J., Colin, S., Stewart, C., Dabiri, J., Tafti, D., and Priya, S., "Passive energy recapture in jellyfish contributes to propulsive advantage over other metazoans," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, no. 44, pp. 17904–17909, 2013.

[47] Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F., "A method for animating viscoelastic fluids," in *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, (New York, NY, USA), pp. 463–468, ACM, 2004.

[48] Grant, I., "Particle image velocimetry: a review," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 211, no. 1, pp. 55–76, 1997.

[49] Grzeszczuk, R. and Terzopoulos, D., "Automated learning of muscle-actuated locomotion through control abstraction," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, (New York, NY, USA), pp. 63–70, ACM, 1995.

[50] Grzeszczuk, R. and Terzopoulos, D., "Automated learning of muscle-actuated locomotion through control abstraction," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, (New York, NY, USA), pp. 63–70, ACM, 1995.

[51] Guendelman, E., Selle, A., Losasso, F., and Fedkiw, R., "Coupling water and smoke to thin deformable and rigid shells," in *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, (New York, NY, USA), pp. 973–981, ACM, 2005.

[52] Ha, S. and Yamane, K., "Reducing Hardware Experiments for Model Learning and Policy Optimization," *IEEE International Conference on Robotics and Automation*, 2015.

[53] Hansen, N. and Kern, S., "Evaluating the CMA evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature-PPSN VIII*, pp. 282–291, Springer, 2004.

[54] Hansen, N., "Covariance matrix adaptation: source code," 2006. https://www.lri.fr/ hansen/cmaes_inmatlab.html.

[55] Hansen, N., *The CMA Evolution Strategy: A Tutorial*, 2009.

[56] Hecker, C., Raabe, B., Enslow, R. W., DeWeese, J., Maynard, J., and van Prooijen, K., "Real-time motion retargeting to highly varied user-created morphologies," in *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, (New York, NY, USA), pp. 27:1–27:11, ACM, 2008.

[57] Heidrich-Meisner, V. and Igel, C., "Evolution strategies for direct policy search," in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X*, (Berlin, Heidelberg), pp. 428–437, Springer-Verlag, 2008.

[58] Hinton, G. E., "Learning multiple layers of representation," *Trends in Cognitive Sciences*, vol. 11, pp. 428–434, 2007.

[59] Hinton, G. E., "A practical guide to training restricted boltzmann machines.," in *Neural Networks: Tricks of the Trade (2nd ed.)* (Montavon, G., Orr, G. B., and Mller, K.-R., eds.), Lecture Notes in Computer Science, pp. 599–619, Springer, 2012.

[60] Hodgins, J. K., Sweeney, P. K., and Lawrence, D. G., "Generating natural-looking motion for computer animation," in *Proceedings of the Conference on Graphics Interface '92*, (San Francisco, CA, USA), pp. 265–272, Morgan Kaufmann Publishers Inc., 1992.

[61] Hodgins, J. K., Wooten, W. L., Brogan, D. C., and O'Brien, J. F., "Animating human athletics," in *SIGGRAPH*, pp. 71–78, Aug. 1995.

[62] Hong, J.-M. and Kim, C.-H., "Discontinuous fluids," in *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, (New York, NY, USA), pp. 915–920, ACM, 2005.

[63] Hu, J., Mitchell, J. E., Pang, J.-S., Bennett, K. P., and Kunapuli, G., "On the global solution of linear programs with linear complementarity constraints," *SIAM Journal on Optimization*, vol. 19, pp. 445–471, 2008.

[64] Institut, X. P. and Provot, X., "Deformation constraints in a mass-spring model to describe rigid cloth behavior," in *In Graphics Interface*, pp. 147–154, 1996.

[65] Irving, G., Teran, J., and Fedkiw, R., "Invertible finite elements for robust simulation of large deformation," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pp. 131–140, 2004.

[66] Irving, G., Schroeder, C., and Fedkiw, R., "Volume conserving finite element simulations of deformable models," *ACM Trans. Graph.*, vol. 26, July 2007.

[67] Jain, S., Ye, Y., and Liu, C. K., "Optimization-based interactive motion synthesis," *ACM Transaction on Graphics*, vol. 28, no. 1, pp. 1–10, 2009.

[68] Jakobi, N., Husbands, P., and Harvey, I., "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*, pp. 704–720, Springer-Verlag, 1995.

[69] James, D. L. and Pai, D. K., "Multiresolution Green's function methods for interactive simulation of large-scale elastostatic objects," *ACM Trans. Graph.*, vol. 22, pp. 47–82, January 2003.

[70] Jones, D. E. H., "The Stability of the bicycle," *Physics Today*, vol. 23, pp. 34–40, 1970.

[71] Kaufman, D. M., Sueda, S., James, D. L., and Pai, D. K., "Staggered projections for frictional contact in multibody systems," *ACM Trans. Graph.*, vol. 27, pp. 164:1–164:11, December 2008.

[72] Kennedy, J. and Eberhart, R. C., "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[73] Kern, S. and Koumoutsakos, P., "Simulations of optimized anguilliform swimming," *Journal of Experimental Biology*, vol. 209, no. 24, p. 4841, 2006.

[74] Kier, W. M., "Tongues, tentacles and trunks: The biomechanics of movement in muscular-hydrostats," *Zoological Journal of the Linnean Society*, vol. 83, pp. 307–324, 1985.

[75] Kim, B., Liu, Y., Llamas, I., and Rossignac, J., "Advections with significantly reduced dissipation and diffusion," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 135–144, 2007.

[76] Kim, J. and Pollard, N. S., "Direct control of simulated non-human characters," *IEEE Computer Graphics and Applications*, vol. 31, pp. 56–65, July 2011.

[77] Kim, J. and Pollard, N. S., "Fast simulation of skeleton-driven deformable body characters," *ACM Transactions on Graphics*, vol. 30, October 2011.

[78] Kim, T. and James, D. L., "Skipping steps in deformable simulation with online model reduction," *ACM Trans. Graph.*, vol. 28, pp. 123:1–123:9, December 2009.

[79] Klein, F. and Sommerfeld, A., "Stabilität des fahrrads," *Über die Theorie des Kreisels, Ch. IX, Section 8*, pp. 863–884, 1910.

[80] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F., "Fluid animation with dynamic meshes," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 820–825, ACM, 2006.

[81] Kober, J., Bagnell, J. A. D., and Peters, J., "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, July 2013.

[82] Kondo, R., Kanai, T., and Anjyo, K.-i., "Directable animation of elastic objects," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pp. 127–134, 2005.

[83] Kooijman, J. D. G., Meijaard, J. P., Papadopoulos, J. M., Ruina, A., and Schwab, A. L., "A Bicycle Can Be Self-Stable Without Gyroscopic or Caster Effects," *Science*, vol. 332, pp. 339–342, Apr. 2011.

[84] Koos, S., Mouret, J.-B., and Doncieux, S., "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pp. 119–126, ACM, 2010.

[85] Kwatra, N., Wojtan, C., Carlson, M., Essa, I., Mucha, P., and Turk, G., "Fluid simulation with articulated bodies," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 1, pp. 70–80, 2009.

[86] Kwon, T. and Hodgins, J., "Control systems for human running using an inverted pendulum model and a reference motion capture sequence," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, (Aire-la-Ville, Switzerland), pp. 129–138, Eurographics Association, 2010.

[87] Laszlo, J., van de Panne, M., and Fiume, E., "Limit cycle control and its application to the animation of balancing and walking," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, (New York, NY, USA), pp. 155–162, ACM, 1996.

[88] Lee, S.-H., Sifakis, E., and Terzopoulos, D., "Comprehensive biomechanical modeling and simulation of the upper body," *ACM Trans. Graph.*, vol. 28, pp. 99:1–99:17, September 2009.

[89] Lee, S.-H. and Terzopoulos, D., "Heads Up! Biomechanical Modeling and Neuromuscular Control of the Neck," *ACM Transactions on Graphics*, vol. 25, pp. 1188–1198, July 2006.

[90] Lentine, M., Grétarsson, J., Schroeder, C., Robinson-Mosher, A., and Fedkiw, R., "Creature Control in a Fluid Environment," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 5, pp. 682–693, 2010.

[91] Levine, S. and Koltun, V., "Guided policy search," in *ICML '13: Proceedings of the 30th International Conference on Machine Learning*, 2013.

[92] Levine, S., Wang, J. M., Haraux, A., Popović, Z., and Koltun, V., "Continuous character control with low-dimensional embeddings," *ACM Trans. Graph.*, vol. 31, pp. 28:1–28:10, July 2012.

[93] Lindsey, C., "Form, function, and locomotory habits in fish," *Fish physiology*, vol. 7, pp. 1–100, 1978.

[94] Liu, C. K. and Popović, Z., "Synthesis of complex dynamic character motion from simple animations," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, (New York, NY, USA), pp. 408–416, ACM, 2002.

[95] Liu, K., "Dynamic animation and robotics toolkit," 2012. http://dartsim.github.io/.

[96] Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L., "Fast simulation of mass-spring systems," *ACM Trans. Graph.*, vol. 32, pp. 214:1–214:7, Nov. 2013.

[97] Losasso, F., Gibou, F., and Fedkiw, R., "Simulating water and smoke with an octree data structure," in *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, (New York, NY, USA), pp. 457–462, ACM, 2004.

[98] Losasso, F., Shinar, T., Selle, A., and Fedkiw, R., "Multiple interacting liquids," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 812–819, ACM, 2006.

[99] Macchietto, A., Zordan, V., and Shelton, C. R., "Momentum control for balance," *ACM Trans. Graph.*, vol. 28, pp. 80:1–80:8, July 2009.

[100] Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y., "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, pp. 153:1–153:12, July 2014.

[101] Martin, S., Thomaszewski, B., Grinspun, E., and Gross, M., "Example-based elastic materials," *ACM Trans. Graph.*, vol. 30, pp. 72:1–72:8, Aug. 2011.

[102] Meijaard, J. P., Papadopoulos, J. M., Ruina, A., and Schwab, A. L., "Linearized dynamics euqations for the balance and steer of a bicycle: a benchmark and review," *Proceedings of the Royal Society A*, 2007.

[103] Miglino, O., Lund, H. H., and Nolfi, S., "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, pp. 417–434, 1996.

[104] Miglino, O., Nafasi, K., and Taylor, C. E., "Selection for wandering behavior in a small robot," *Artificial Life*, vol. 2, pp. 101–116, 1994.

[105] Miller, G. S. P., "The motion dynamics of snakes and worms," *SIGGRAPH Comput. Graph.*, vol. 22, pp. 169–173, June 1988.

[106] Monaghan, J. J., "Smoothed particle hydrodynamics," *Annual review of astronomy and astrophysics*, vol. 30, pp. 543–574, 1992.

[107] Mordatch, I., de Lasa, M., and Hertzmann, A., "Robust physics-based locomotion using low-dimensional planning," in *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, (New York, NY, USA), pp. 71:1–71:8, ACM, 2010.

[108] Mordatch, I., Lowrey, K., and Todorov, E., "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," *IEEE International Conference on Robotics and Automation*, 2015.

[109] Mordatch, I., Popović, Z., and Todorov, E., "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, (Aire-la-Ville, Switzerland, Switzerland), pp. 137–144, Eurographics Association, 2012.

[110] Mordatch, I., Wang, J. M., Todorov, E., and Koltun, V., "Animating human lower limbs using contact-invariant optimization," *ACM Trans. Graph.*, vol. 32, pp. 203:1–203:8, Nov. 2013.

[111] Mouret, J., Koos, S., and Doncieux, S., "Crossing the reality gap: a short introduction to the transferability approach," *CoRR*, vol. abs/1307.1870, 2013.

[112] Muico, U., Lee, Y., Popović, J., and Popović, Z., "Contact-aware nonlinear control of dynamic characters," in *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, (New York, NY, USA), pp. 81:1–81:9, ACM, 2009.

[113] Müller, M., Charypar, D., and Gross, M., "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, (Aire-la-Ville, Switzerland, Switzerland), pp. 154–159, Eurographics Association, 2003.

[114] Müller, M., Dorsey, J., McMillan, L., Jagnow, R., and Cutler, B., "Stable real-time deformations," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pp. 49–54, 2002.

[115] Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J., "Position based dynamics," *J. Vis. Comun. Image Represent.*, vol. 18, pp. 109–118, Apr. 2007.

[116] Nesme, M., Payan, Y., and Faure, F., "Efficient, physically plausible finite elements," in *Eurographics 2005, Short papers, August, 2005* (Dingliana, J. and Ganovelli, F., eds.), (Trinity College, Dublin, Irlande), 2005.

[117] Ng, A. Y. and Jordan, M., "Pegasus: A policy search method for large MDPs and POMDPs," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, (San Francisco, CA, USA), pp. 406–415, Morgan Kaufmann Publishers Inc., 2000.

[118] Ng, A. Y. and Russell, S. J., "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, (San Francisco, CA, USA), pp. 663–670, Morgan Kaufmann Publishers Inc., 2000.

[119] Nguyen, D. Q., Fedkiw, R., and Jensen, H. W., "Physically based modeling and animation of fire," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, (New York, NY, USA), pp. 721–728, ACM, 2002.

[120] Nolfi, S. and Floreano, D., *Evolutionary Robotics: The Biology,Intelligence,and Technology.* Cambridge, MA, USA: MIT Press, 2000.

[121] O'Brien, J. F. and Hodgins, J. K., "Graphical modeling and animation of brittle fracture," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pp. 137–146, 1999.

[122] Otaduy, M. A., Tamstorf, R., Steinemann, D., and Gross, M., "Implicit contact handling for deformable objects," *Computer Graphics Forum (Proc. of Eurographics)*, vol. 28, apr 2009.

[123] Peters, J. and Schaal, S., "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, May 2008.

[124] Pratt, J. E., Chew, C.-M., Torres, A., Dilworth, P., and Pratt, G. A., "Virtual model control: An intuitive approach for bipedal locomotion.," *Int'l J. Robotic Research.*, vol. 20, no. 2, pp. 129–143, 2001.

[125] Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A. E., and Whitaker, R. T., "Particle-based simulation of fluids.," *Comput. Graph. Forum*, vol. 22, no. 3, pp. 401–410, 2003.

[126] Randløv, J. and Alstrøm, P., "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)* (Shavlik, J. W., ed.), (San Francisco, CA, USA), pp. 463–471, Morgan Kauffman, 1998.

[127] Rankine, W. J. M., "On the dynamical principles of the motion of velocipedes," *The Engineer*, 1870.

[128] Raveendran, K., Wojtan, C., and Turk, G., "Hybrid smoothed particle hydrodynamics," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, (New York, NY, USA), pp. 33–42, ACM, 2011.

[129] Robinson-Mosher, A., Shinar, T., Gretarsson, J., Su, J., and Fedkiw, R., "Two-way coupling of fluids to rigid and deformable solids and shells," in *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, (New York, NY, USA), pp. 46:1–46:9, ACM, 2008.

[130] ROBOTIS, *AX-18F/Ax-18A e-Manual*.

[131] Schulz, C., von Tycowicz, C., Seidel, H.-P., and Hildebrandt, K., "Animating deformable objects using sparse spacetime constraints," *ACM Trans. Graph.*, vol. 33, pp. 109:1–109:10, July 2014.

[132] Schwarz, M. and Behnke, S., "Compliant robot behavior using servo actuator models identified by iterative learning control," in *RoboCup 2013: Robot World Cup XVII [papers from the 17th Annual RoboCup International Symposium, Eindhoven, The Netherlands, July 1, 2013].*, pp. 207–218, 2013.

[133] Shirgaonkar, A., Curet, O., Patankar, N., and MacIver, M., "The hydrodynamics of ribbon-fin propulsion during impulsive motion," *J. Exp. Biol*, vol. 211, pp. 3490–3503, 2008.

[134] Si, H., "Tetgen: A quality tetrahedral mesh generator and a 3D Delaunay triangulator," January 2006.

[135] Si, W., Lee, S.-H., Sifakis, E., and Terzopoulos, D., "Realistic biomechanical simulation and control of human swimming," *ACM Trans. Graph.*, vol. 34, pp. 10:1–10:15, Dec. 2014.

[136] Sifakis, E., Neverov, I., and Fedkiw, R., "Automatic determination of facial muscle activations from sparse motion capture marker data," *ACM Trans. Graph.*, vol. 24, pp. 417–425, July 2005.

[137] Sims, K., "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, (New York, NY, USA), pp. 15–22, ACM, 1994.

[138] Singh, D. V., *Advanced concepts of the stability of two-wheeled vehicle-application of mathematical analysis to actual vehicles*. PhD thesis, University of Wisconsin, 1964.

[139] Smith, R., "Open dynamics engine," 2008. http://www.ode.org/.

[140] Stam, J., "Stable fluids," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, (New York, NY, USA), pp. 121–128, ACM Press/Addison-Wesley Publishing Co., 1999.

[141] Stanley, K., "Neuroevolution of augmenting topology: source code," 2002. http://www.cs.ucf.edu/ kstanley/neat.html.

[142] Stanley, K. O. and Miikkulainen, R., "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, June 2002.

[143] Stanton, A. and Unkrich, L., "Finding Nemo [motion picture]," *United States: Walt Disney Pictures*, 2003.

[144] Stewart, D. and Trinkle, J. C., "An implicit time-stepping scheme for rigid body dynamics with Coulomb friction," *International Journal of Nnumerical Methods in Engineering*, vol. 39, pp. 2673–2691, 1996.

[145] Sueda, S., Kaufman, A., and Pai, D. K., "Musculotendon simulation for hand animation," *ACM Trans. Graph.*, vol. 27, pp. 83:1–83:8, August 2008.

[146] Sutton, R. S. and Barto, A. G., *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[147] Takahashi, T., Ueki, H., Kunimatsu, A., and Fujii, H., "The simulation of fluid-rigid body interaction," in *ACM SIGGRAPH 2002 conference abstracts and applications*, SIGGRAPH '02, (New York, NY, USA), pp. 266–266, ACM, 2002.

[148] Tan, J., Gu, Y., Liu, C. K., and Turk, G., "Learning bicycle stunts," *ACM Trans. Graph.*, vol. 33, pp. 50:1–50:12, July 2014.

[149] Tan, J., Gu, Y., Turk, G., and Liu, C. K., "Articulated swimming creatures," in *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pp. 58:1–58:12, ACM, 2011.

[150] Tan, J., Liu, K., and Turk, G., "Stable proportional-derivative controllers," *Computer Graphics and Applications*, 2011.

[151] Tan, J., Turk, G., and Liu, C. K., "Soft body locomotion," *ACM Trans. Graph.*, vol. 31, pp. 26:1–26:11, July 2012.

[152] Teran, J., Blemker, S., Hing, V. N. T., and Fedkiw, R., "Finite volume methods for the simulation of skeletal muscle," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pp. 68–74, 2003.

[153] Teran, J., Sifakis, E., Blemker, S. S., Ng-Thow-Hing, V., Lau, C., and Fedkiw, R., "Creating and simulating skeletal muscle from the visible human data set," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, pp. 317–328, May 2005.

[154] Terzopoulos, D., Tu, X., and Grzeszczuk, R., "Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world," *Artificial Life*, vol. 1, no. 4, pp. 327–351, 1994.

[155] Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K., "Elastically deformable models," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pp. 205–214, 1987.

[156] Thomas, F. and Johnston, O., *The illusion of life: Disney animation.* Hyperion, 1995.

[157] Thrun, S. and Schwartz, A., "Issues in using function approximation for reinforcement learning," in *In Proceedings of the Fourth Connectionist Models Summer School*, Erlbaum, 1993.

[158] Treuille, A., Lee, Y., and Popović, Z., "Near-optimal character animation with continuous control," *ACM Trans. Graph.*, vol. 26, July 2007.

[159] Tsai, Y.-Y., Lin, W.-C., Cheng, K. B., Lee, J., and Lee, T.-Y., "Real-time physics-based 3D biped character animation using an inverted pendulum model," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 325–337, Mar. 2010.

[160] Tsang, W., Singh, K., and Eugene, F., "Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pp. 319–328, 2005.

[161] Tu, X. and Terzopoulos, D., "Artificial fishes: Physics, locomotion, perception, behavior," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 43–50, ACM, 1994.

[162] Tu, X. and Terzopoulos, D., "Artificial fishes: physics, locomotion, perception, behavior," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, (New York, NY, USA), pp. 43–50, ACM, 1994.

[163] Tytell, E. and Lauder, G., "The hydrodynamics of eel swimming. I. Wake structure," *Journal of Experimental Biology*, vol. 207, no. 11, pp. 1825–1841, 2004.

[164] van de Panne, M. and Lee, C., "Ski stunt simulator: Experiments with interactive dynamics," in *Proceedings of the 14th Western Computer Graphics Symposium*, I3D '05, 2003.

[165] Van Zytveld, P., *A method for the automatic stabilization of an unmanned bicycle.* Department of Aeronautics and Astronautics, Stanford University., 1975.

[166] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A., "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), pp. 1096–1103, ACM, 2008.

[167] Wampler, K. and Popović, Z., "Optimal gait and form for animal locomotion," *ACM Trans. Graph.*, vol. 28, pp. 60:1–60:8, July 2009.

[168] Wampler, K. and Popović, Z., "Optimal gait and form for animal locomotion," in *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, (New York, NY, USA), pp. 60:1–60:8, ACM, 2009.

[169] Wang, H., Mucha, P. J., and Turk, G., "Water drops on surfaces," in *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, (New York, NY, USA), pp. 921–929, ACM, 2005.

[170] Wang, J. M., Fleet, D. J., and Hertzmann, A., "Optimizing walking controllers," *ACM Trans. Graph.*, vol. 28, pp. 168:1–168:8, Dec. 2009.

[171] Wang, J. M., Fleet, D. J., and Hertzmann, A., "Optimizing walking controllers for uncertain inputs and environments," *ACM Trans. Graph.*, vol. 29, pp. 73:1–73:8, July 2010.

[172] Wang, J. M., Hamner, S. R., Delp, S. L., and Koltun, V., "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Trans. Graph.*, vol. 31, pp. 25:1–25:11, July 2012.

[173] Whipple, F. J. W., "The stability of the motion of a bicycle," *Quarterly Journal of Pure and Applied Mathematics*, vol. 30, pp. 312–348, 1899.

[174] Witkin, A. and Kass, M., "Spacetime constraints," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, (New York, NY, USA), pp. 159–168, ACM, 1988.

[175] Wu, J.-c. and Popović, Z., "Realistic modeling of bird flight animations," in *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, (New York, NY, USA), pp. 888–895, ACM, 2003.

[176] Wu, J.-c. and Popović, Z., "Terrain-adaptive bipedal locomotion control," in *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, (New York, NY, USA), pp. 72:1–72:10, ACM, 2010.

[177] Wyneken, J., "Sea Turtle Locomotion: Mechanisms, Behavior, and Energetics," *The biology of sea turtles*, p. 165, 1997.

[178] Yang, P.-F., Laszlo, J., and Singh, K., "Layered dynamic control for interactive character swimming," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, (Aire-la-Ville, Switzerland, Switzerland), pp. 39–47, Eurographics Association, 2004.

[179] Ye, Y. and Liu, C. K., "Optimal feedback control for character animation using an abstract model," in *SIGGRAPH '10: ACM SIGGRAPH 2010 papers*, (New York, NY, USA), pp. 1–9, ACM, 2010.

[180] Yin, K., Coros, S., Beaudoin, P., and van de Panne, M., "Continuation methods for adapting simulated skills," *ACM Trans. Graph.*, vol. 27, no. 3, 2008.

[181] Yin, K., Loken, K., and van de Panne, M., "SIMBICON: simple biped locomotion control," in *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, 2007.

[182] Yngve, G. D., O'Brien, J. F., and Hodgins, J. K., "Animating explosions," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, (New York, NY, USA), pp. 29–36, ACM Press/Addison-Wesley Publishing Co., 2000.

[183] Zagal, J. C., Ruiz-del-Solar, J., and Vallejos, P., "Back-to-Reality: Crossing the reality gap in evolutionary robotics," in *IAV 2004: Proceedings 5th IFAC Symposium on Intelligent Autonomous Vehicles*, Elsevier Science Publishers B.V., 2004.

[184] Zhao, P. and van de Panne, M., "User interfaces for interactive control of physics-based 3D characters," in *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, (New York, NY, USA), pp. 87–94, ACM, 2005.

[185] Zordan, V. B., Celly, B., Chiu, B., and DiLorenzo, P. C., "Breathe easy: model and control of human respiration for computer animation," *Graph. Models*, vol. 68, pp. 113–132, March 2006.