DEVELOPING AGILE MOTOR SKILLS ON VIRTUAL AND REAL HUMANOIDS

A Thesis
Presented to
The Academic Faculty

by

Sehoon Ha

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

Georgia Institute of Technology
December 2015

DEVELOPING AGILE MOTOR SKILLS ON VIRTUAL AND REAL HUMANOIDS

Approved by:

Dr. C. Karen Liu, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Greg Turk
School of Interactive Computing
Georgia Institute of Technology

Dr. Jarek Rossignac
School of Interactive Computing
Georgia Institute of Technology

Dr. Jun Ueda
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Katsu Yamane
Disney Research Pittsburgh

Date Approved: 31 July 2015

To my parents,

Sangsoo Ha and Jiwon Lee,

who have given me the best love and support

throughout my life.

ACKNOWLEDGEMENTS

I cannot begin to express my thanks to my wife, Jennifer Gahee Kim, who sincerely supports me with love, wisdom, insight, and delicious food. It is very special to share my life with someone who is not just my wonderful wife, but also the greatest friend a person could ever have. You make my life happier than ever.

I also want to thank my mother Jiwon Lee and my father Sangsoo Ha for guiding me throughout my life. I would not have been able to find, start, and finish my academic life without their unconditional love and support.

I am also grateful to my brother Jihoon, his family Eunchae and Dongkwon, and parents-in-law Eungyoung Lee and Boogyun Kim for encouraging me to pursue my degree. Especially, my brother has been a great tutor who teaches me how to add numbers, how to program, how to solve puzzles, and many more in the most interesting ways.

I want to thank the members of my committee: Greg Turk, Jarek Rossignac, Jun Ueda, Katsu Yamane, and my advisor C. Karen Liu. Each of you encouraged and help me to become a much stronger and solid researcher with insightful suggestions on my research direction and communication skills.

I would like to extend my sincere thanks to my lab-mates who have shared invaluable discussions during the entire program: a co-advisor Yuting Ye, Jie Tan, Yunfei Bai, Karthik Raveendran, Sumit Jain, Yuting Gu, Chen Tang, John Turner, Alex Clegg, Kihwan Kim, Chirs Wojtan, Jason Williams, Jihun Yu, Tina Zhou, Mark Luffel, Topraj Gurung, Kristin Siu, Yunseong Song, Jeongseok Lee, Wenhao Yu, and many more. In addition, I would like to thank my friends/mentors: Yeongjin, Seongmin, Jaeshik, and Donggu. I am really sorry if I forgot anyone.

I wish to particularly thank Evan Kanso, Jovan Popović, Jim McCann, and Katsu Yamane, who provided me with encouragement and patience throughout the duration of my internships. These experiences widen my research horizons.

Special thanks to my mentors, Wangjae Lee and Jaehong Kim, who guides my life to a better direction.

I would like to express my deepest appreciation to my advisor, C. Karen Liu. I feel extremely fortunate to have her as my advisor. Her intellectual insights and creativity for research problems inspired to become a better researcher. Her enthusiasm for research encourages me to keep working on more challenging problems, which is one of the most valuable lessons learned in my life. It was a great honor to be under her supervision for the last six years. I will miss all interactions with you, Karen, including both insightful discussions and fun jokes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Demonstrating strength and agility on virtual and real humanoids has been an important goal in computer graphics and robotics. However, developing physics-based controllers for various agile motor skills requires a tremendous amount of prior knowledge and manual labor due to complex mechanisms of the motor skills. The focus of the dissertation is to develop a set of computational tools to expedite the design process of physics-based controllers that can execute a variety of agile motor skills on virtual and real humanoids. Instead of designing directly controllers real humanoids, this dissertation takes an approach that develops appropriate theories and models in virtual simulation and systematically transfers the solutions to hardware systems.

The algorithms and frameworks in this dissertation span various topics from specific physics-based controllers to general learning frameworks. We first present an online algorithm for controlling falling and landing motions of virtual characters. The proposed algorithm is effective and efficient enough to generate falling motions for a wide range of arbitrary initial conditions in real-time. Next, we present a robust falling strategy for real humanoids that can manage a wide range of perturbations by planning the optimal contact sequences. We then introduce an iterative learning framework to easily design various agile motions, which is inspired by human learning techniques. The proposed framework is followed by novel algorithms to efficiently optimize control parameters for the target tasks, especially when they have many constraints or parameterized goals. Finally, we introduce an iterative approach for exporting simulation-optimized control policies to hardware of robots to reduce the

number of hardware experiments, that accompany expensive costs and labors.

CHAPTER I

INTRODUCTION

The human desire of being physically impressive has a long history dated back to ancient Greece. The ancient Olympic Games celebrated the glory of physical dominance of body strength, as well as athletic agility, to achieve the goals faster and more effortlessly. Today, physical agility continues to captivate our imagination in various areas, such as sports, entertainment, or robot industry. In film or game industries, agile motions are important content to make their games or movies more fun and entertaining. Besides entertainment values, being agile and strong also holds great practical values for robotic technologies, which can help search and rescue. When the Fukushima Daiichi nuclear meltdown occurred in 2011, it was too dangerous for human operations because of high radiation levels. A few robots were deployed to measure temperature and radioactivity but failed due to limited speed and range of operations [4]. The robots would have achieved the assigned tasks if they had had enough agility to locomote on uneven terrains and climb over disaster debris.

Although "agile" motions include several examples, jumping, vaulting, or rolling, they are not precisely defined to be discussed. The word "agile" is defined as "able to move quickly, easily, effortlessly, and gracefully". While moving "quickly" can be directly associated with high-momentum motions, moving "easily", "effortlessly", and "gracefully" are more difficult to physically quantify. For instance, moving "easily" can be due to different reasons, such as the innate strength of the subject, the accumulated experience, or the ability to negotiate with the environment. In this disseration, we define "agile motions" as "able to move quickly and effortlessly in a complex environment" , which highlights high momentum and well-coordinated

strategies that exploit the environment. One great example that showcases a suite of agile motions is Parkour, a sport that features moving from the current location to the destination in the most efficient way. The skills of Parkour are well aligned with our definition of agile motions because they require high momentum and dynamic planning to negotiate with the obstacles and features in the environment.

The aim of this dissertation is to design a set of computational tools to develop agile motor skills on virtual and real humanoids. Although the grand goal would be to achieve agile motor skills on humanoid robots, there are many unsolved practical issues to directly design agile motions on the hardware, such as control delays, sensor noises, or safety concerns. Unlike biped walking where adequate models and control theories are already available, agile motions have not been studied extensively in the literature of computer animation and robotics. Therefore, the appropriate approach toward agile real humanoids is to first develop theoretical models and control algorithms on the physics-based simulation, where researchers in computer graphics can generate complex and impressive motions; then the next step is to transfer the solution to real hardware with additional processes, such as tuning parameters or revising control mechanisms. Simulation-developed solutions will greatly reduce the expensive cost of hardware experiments and prevent unexpected dangerous situations.

Agile motor skills are difficult tasks for humanoids. General characteristics of agile motions, high momentum or frequent changes of contacts, increase difficulties of most existing issues in motor control problems tremendously by requiring accurate control routines and efficient planning algorithms. In addition to the existing issues, agile motions generate few more unique challenges that need to be addressed. The first issue is to ensure the safety of humanoids. Because learning and executing agile motions often accompany both intentional and inadvertent falls, it is necessary to learn how to fall safely without introducing large damage. Secondly, agile motions are not well-understood unlike other motions, such as running or reaching. They

cover a wide range of diverse skills, such as jumping, vaulting, or rolling which are governed by different control mechanisms and principles. We want to develop a set of general computational tools for designing a variety of agile motion controllers, instead of manually designing all controllers individually. Finally, simple agile motion trajectories that are developed in virtual simulation are less likely to work on physical systems because small errors can be quickly accumulated and result in drastic deviations from the original agile motions. The discrepancy between virtual and real systems must be handled systematically for utilizing simulation-optimized solutions. These additional, unique challenges motivated us to address the following problems in this dissertation, as steps to achieve agile motions on virtual and real humanoids.

1.1 Falling Strategies for Humanoids

Falling and landing motions are a set of fundamental motor skills in various agile motions to protect humanoids themselves. Because athletic movements frequently involves transitions between airborne and contact phases, humanoids must know how to absorb the shock at the landing moment and avoid damage to the body parts. In addition, well-executed falling motions will lead a smooth transition to the next motion.

We discuss two different falling controllers for virtual and real humanoids. In Chapter 3, we discuss falls with the airborne phases for a virtual humanoid, which are inspired by agile jumping and falling motions of Parkour practitioners. On the other hand in Chapter 4, we focus on falls without the airborne phases for real humanoids, which can be initiated by unexpected perturbations. In both scenarios, our falling controllers are robust enough to handle a wide range of initial conditions, such as heights and speeds. The effectiveness of both presented strategies are validated by measuring amounts of joint stresses or contact forces to body parts in physics simulation, and experimentally tested on a small-size humanoid.

1.1.1 Falling and Landing Motion Control for Virtual Characters

In Chapter 3, we show how to create a robust controller for generating agile and natural falling motions of the virtual character that can land from various heights and velocities. The goals of the controller are to reduce the joint stress at the impact and get back on its feet to prepare the next action (Figure 1).

Inspired by falling skills of Parkour, we formulate the falling problem with three phases, *airborne*, *impact*, and *rolling* based on the contact states of a virtual character. First, two sub-controllers are designed for the *airborne* and *rolling* phases and a regression analysis is conducted to find an optimal landing angle that can connect two sub controllers at the *impact* phase. we will demonstrate that the motion generated by the proposed controller looks natural and induces smaller joint stress, which is still four times lower than a uncontrolled rag-doll motion.

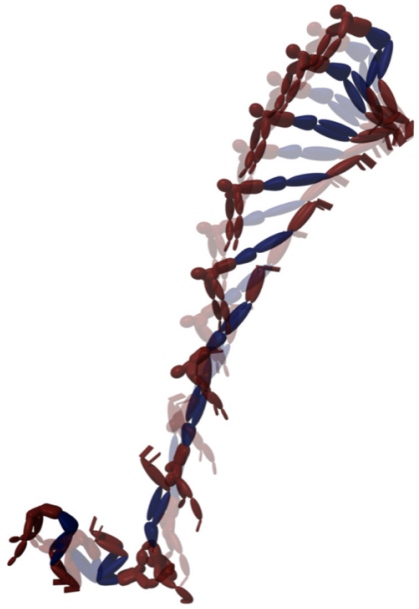


Figure 1: A fall of a virtual character.

1.1.2 Multiple Contact Planning for Humanoid falls

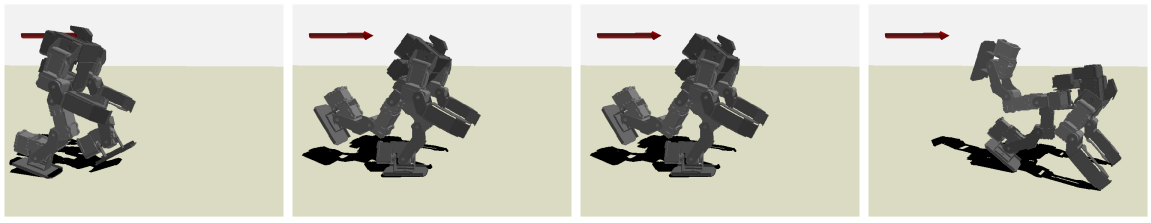


Figure 2: A two-step falling strategy of a humanoid robot

Chapter 4 describes a robust falling strategy which plans for appropriate responses to a wide variety of falls, from a single step to recover from a gentle nudge, to a rolling motion to break a high-speed fall. Our key observation is that many existing falling techniques [103, 109, 5] assume specific sequences of contacts and aim to break specific ranges of falls and requires us to an additional decision layer to choose the best falling strategy to the given state. Instead, our multiple contact planning algorithm provides a unified framework for various existing falling strategies, which can adjust the number and order of contacts with respect to different magnitudes of perturbations. In the proposed continuous space of strategies, our algorithm can efficiently find the best contact sequences for the given initial state using a simplified model and dynamic programming. To verify our framework, a variety of scenarios are tested on simulated humanoids and the actual hardware (Figure 2) to show that our algorithm plans versatile and effective falling strategies that successfully reduce damage to robots.

1.2 Learning Framework for General Agile Motions

Unlikely well-defined tasks such as reaching or walking, agile motions cover a wide range of diverse skills such as vaulting, flipping, or rolling. Because these agile motions are governed by different mechanisms and principles, manually designing and fine-tuning specialized controllers for all tasks is not a practical approach. Design of an individual physics-based controller for a new motor skill is a time-consuming task which requires a lot of manual efforts from the controller designer, from the design of the control mechanism to the tweaking of low-level control parameters.

Our goal is to design an intuitive and efficient learning framework for general motor skills, including jumping, flipping, vaulting, and rolling. For an easy-to-use learning framework, we develop a novel interactive interface that designs controllers using only human-readable instructions inspired by how human coaches teach dynamic motor skills. Further, the parameters of the designed controllers are efficiently optimized

using our novel algorithms to reduce a turn-around time. Using the proposed interface and optimization techniques, users can intuitively and easily develop physics-based controllers for general agile motor skills.

1.2.1 Iterative Design of Dynamic Controllers

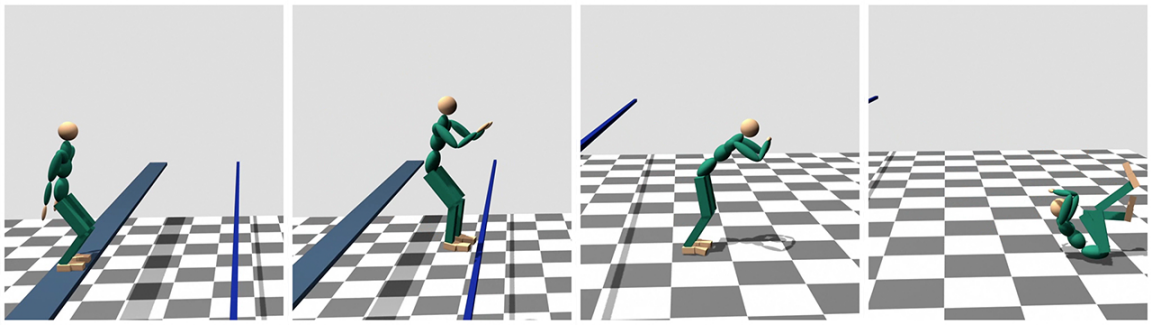


Figure 3: A challenging stunt that a character jumps twice and rolls on the ground.

In Chapter 5, we describe an iterative framework to design physics-based controllers that executes very agile stunts (Figure 3). The aim of this chapter is to design an intuitive and interactive framework that a user can easily design complex controllers only using high-level, human-readable instructions, which is inspired by a human learning process of coaching and practicing stages. During the coaching stage, the user provides instructions for revising task objectives, adding constraints, or updating control mechanisms based on the current skill level. To enable interactive coaching, we introduce "control rigs" as an intermediate layer of control module which allows more coordinated control of the low level control variables and and provides more intuitive mapping to high-level human instructions. During the practicing stage, control parameters are efficiently determined using CMA-ES, which will be further improved in the following chapters. The details of controllers development process using our iterative learning framework are shown with example iterative training procedures of Parkour motions.

1.2.2 Optimization with Failure Learning

In Chapter 6, we describe a new optimization algorithm for highly constrained problems, which are formulated when the user imposes a lot of instructions to the virtual character. A controller with many constraints is difficult to be optimized due to the relatively small feasible regions with many local minima. Our key idea comes from humans ability to learn from failure. Because failure in the real world is usually associated with pain or injury, humans tend to be very effective in characterizing the cause of failure and trying to avoid the same mistakes in the future. Under the concept of "learning from failure", the proposed algorithm CMA-C (Covariance Matrix Adaptation with Classification) utilizes the failed simulation trials to approximate an infeasible region in the space of control rig parameters so that it can predict validity of newly generated samples, resulting in a faster convergence than the standard CMA-ES.

1.2.3 Optimization for Parametrized Motor Skills

In Chapter 7, we explain an evolutionary optimization algorithm for learning parameterized skills to achieve whole-body dynamic tasks. The parametrization of the learned motor skills is an essential ability because a robot can adapt the skill to a new situation, without learning the entire motor skill from scratch. Instead of maintaining a single Gaussian distribution, the algorithm reduces the number of samples by evolving a parametrized probability distribution which describes a mapping from a task parameter to the optimal control parameters. we test the proposed optimization algorithm for learning three parametrized dynamic motor skills on a simulated humanoid robot, including jumping, kicking, and walking.

1.3 Transferring Controllers from Simulation to Hardware

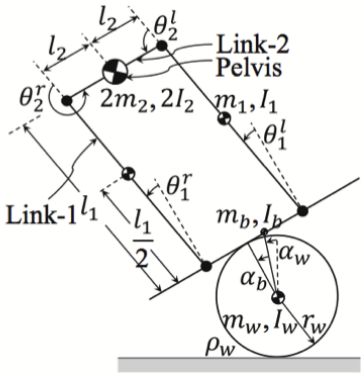


Figure 4: A design of a simple legged robot on a bongoboard.

In Chapter 8, we describe an iterative approach for learning hardware models and optimizing control policies using simulation. The proposed learning framework is designed for reducing the number of expensive hardware experiments. Computer simulation is often used to replace hardware experiments, but it is difficult to obtain accurate simulation models and simulation-optimized control policies are not likely to work on hardware. To fill the gap between two system, we propose an algorithm for learning hardware models and optimizing policies. Instead of learning hardware models from scratch, the proposed approach only learns the difference from simulation models using Gaussian process. As a proof of concept, we validate the algorithm on two different simulation models, one with perfect contacts and one with realistic contacts, by finding a balancing controller for a simple bipedal robot on a bongo board (Figure 4).

1.4 Contributions

The control and optimization methods discussed in this dissertation provide several contributions to the computer animation and robotics community. These contributions are as follows:

- **A falling and landing strategy for virtual characters.** The falling strategy presented in the dissertation generates a natural falling and landing motion that falls from a wide range of heights and initial speeds, continuously rolls on the ground, and gets back on its feet without inducing large stress on joints at any moment.

- **A multiple contact falling strategy for robots.** We introduce a falling

strategy for humanoid robots to break a fall with minimal damage to the body parts by utilizing multiple contact points.

- **An iterative learning framework for dynamic motor skills.** We propose an iterative and interactive learning framework using human readable instructions that can teach a variety of agile motions to virtual characters. Starting from a basic controller, the proposed framework allows a user to easily train complex physics-based controllers with only intuitive high-level instructions from the user.

- **An optimization technique for highly constrained problems.** We introduce a novel efficient optimization algorithm, CMA-C, that is designed for the problem with many constraints and smaller feasible regions. The algorithm converges faster than the standard CMA-ES, by approximating the infeasible region using learned classifiers.

- **An optimization technique for parametrized tasks.** We introduce an efficient evolutionary optimization algorithm for learning parametrized whole-body dynamic tasks. By evolving parameterized sample distributions, our algorithm converges faster than the baseline algorithm, CMA-ES.

- **A model-based policy search for reducing hardware experiments.** We propose an iterative approach for learning hardware model and optimizing policies with as few hardware experiments as possible by learning *dynamics bias*, which is difference between simulation and hardware models.

CHAPTER II

BACKGROUND

The aim of this chapter is to review relevant prior work done in computer graphics and robotics, and biomechanics. We will start with a brief discussion on various physics-based animation techniques for generating realistic and agile motions of virtual humanoids. We then briefly outline control strategies for reducing damage of humanoid falls in both robotics and graphics, which inspired the proposed falling strategies in this dissertation. Afterward, we will review the *human-in-the-loop* principle, which is adopted for designing our learning framework for general agile motions. Because this dissertation proposes several policy search algorithms for optimizing the controllers in simulation and deploying them on hardware, we will conclude this chapter with a review of relevant prior optimization techniques in computer graphics and robotics.

2.1 Physics-based simulation of agile motor skills

Throughout the entire sessions, this dissertation utilizes physics simulation to generate agile motions of virtual characters and test control policies before deploying on real robots. Various physics-based animation techniques have been proposed in computer graphics to plan and control various motor skils, ranging from locomotion to agile stunts of Parkour.

2.1.1 Physics-based simulation for character animation

Physics-based character animation is a promising approach to creating realistic and interactive animations, but designing controllers remains difficult largely due to the complex relationship between the control and the state variables. Early work [37, 105]

demonstrated that a variety of motions can be achieved by controlling the individual joints with manually designed state machines. Since this seminal work was published, researchers in computer animation have been searching for new control algorithms that are more robust, more generalizable, and more automatic. Using motion capture data for reference trajectories was a step toward a more automatic process for controller design [112, 89], however, the simulated motions cannot deviate much from the input data. An improved approach applied linear or nonlinear quadratic regulators to track reference trajectories, leading to more robust controllers against perturbations [19, 70]. Combination of PD servos and a specialized balance controller driven by a simple state machine was a very successful strategy [108], which enabled much follow-on work in biped control [100, 13, 54, 42]. Global planning of momentum has also been applied to a wide range of motion from standing balance [60] to locomotion [66, 107] to highly dynamic motion [33, 58, 8, 111] Coros *et al.* adopted Jacobian transpose control from robotics literature [95] to generate stable biped and quadruped locomotion [13, 14]. Ha *et al.* further demonstrated the effectiveness of the Jacobian transpose control on dynamic stunts [59, 33, 32].

2.1.2 Physics-based controllers for agile motions

Physics-based controllers for agile motions are more extensively developed in virtual simulation due to the limitation of hardware. The work presented in Chapter 3, 5, and 6 is inspired by prior animation techniques designed for various motor tasks. Previous work has demonstrated that highly dynamic motions with a long ballistic phase can be synthesized using physics simulation or kinematic approaches. Hodgins *et al.* [37, 105] showed that carefully crafted control algorithms can simulate highly athletic motions, including diving, tumbling, vaulting, and leaping. Faloutsos *et al.* [23] composed primitive controllers to simulate more complex motor skills, such as a kip-up move or a dive down stairs. Liu *et al.* [59] successfully tracked contact-rich

mocap sequences using a sampling-based approach. They showed that vigorous motions with complex contacts, such as a dive-roll or a kip-up move, can be dynamically simulated, provided full body mocap sequences as desired trajectories. Zhao and van de Panne [110] provided a palette of parametrized actions to build a user interface for controlling highly dynamic animation. Other techniques directly edit ballistic motion sequences under the constraints imposed by conservation of momentum [61, 90], or apply a hybrid method for synthesizing dynamic response to perturbation in the environment [87]. If the contact positions and timing are known, spacetime optimization techniques can also generate compelling dynamic motions [57, 24, 85, 94]. This thesis the approach of physical simulation, but we seek for a more general and robust control algorithm such that the controller can operate under a wide range of initial conditions and allow for runtime perturbations.

2.2 Control of humanoid falls

A goal of falling strategies is to minimize damage or joint stress to humanoid when it falls. This is an important problem to protect virtual and real humanoids during learning and executing agile motor skills, which has been well studied within a variety of research areas, such as computer graphics, robotics, biomechanics, and martial arts.

2.2.1 Falling detection techniques

Although the falling strategies proposed in Chapter 3 and 4 majorly focus on reducing damage to body parts, they require a fall detection module for predicting and estimating falls, which is assumed in this disseration. It will predict a fall and try to recover the balance if it is possible, and activate a falling controller if falling is inevitable. Various machine learning techniques has been proposed to detect falls, such as Principal Component Analysis [45] or Supported Vector Machine [47]. Horn and Gerth [38] detects unstable situations with Gaussian Mixture Model or Hidden Markov Model and activates appropriate reflex controls, such as crouching. Renner

and Benke [79] proposed to detect instability using an aggregated sensor deviation and stabilize the gait with manually designed reflex controllers. The falling strategies presented in this dissertation focus on control of falling motions to reduce damage when the robot detects falling, presumably with one of the above techniques.

2.2.2 Falling damage reduction strategies

In this section, we will review the related work on the falling strategies proposed in Chapter 3 and 4. Various techniques in different disciplines have been proposed to minimize damage on a humanoid when it falls. Fujiwara *et al.* [28, 30, 27, 26] proposed falling techniques inspired by Japanese martial arts (*Ukemi*). Ogata *et al.* [75, 74] evaluates the risk of falling with predicted ZMP and optimizes COM trajectories to reduce damage. Ruiz-del-Solar *et al.* [84, 83] designed low damage falling sequences for soccer robots and verified them in the simulation. Wang *et al.* [103] formulated an optimization of whole body trajectories as a nonlinear programming problem and solved it with heuristics. Lee and Goswami [53] proposed a control strategy that reorients the robot to fall with a backpack for absorbing shock. Yun and Goswami [109] addressed a "tripod" strategy that stops with a swing foot and two hands to maintain the final COM location higher from the floor. To protect the surrounding environment, [31] proposed a fall direction-changing strategy that utilizes foot placement and inertia shaping.

Besides the related work for falls caused by external perturbations, there are additional works that focus on falls from higher places. In those cases, control strategies during long airborne phase become critical for safe landing. The falling algorithm in Chapter 3 draws inspiration from kinesiology literature and sport practitioners. In particular, the techniques developed in freerunning and parkour community are of paramount importance for designing landing control algorithms capable of handling arbitrary scenarios [22, 39]. In robotics, Bingham *et al.* [10] proposed an algorithm

that leverages nonholonomic trajectory planning inspired by the falling cat to orient an articulated robot through configuration changes to achieve a pose that reduces the impact at landing.

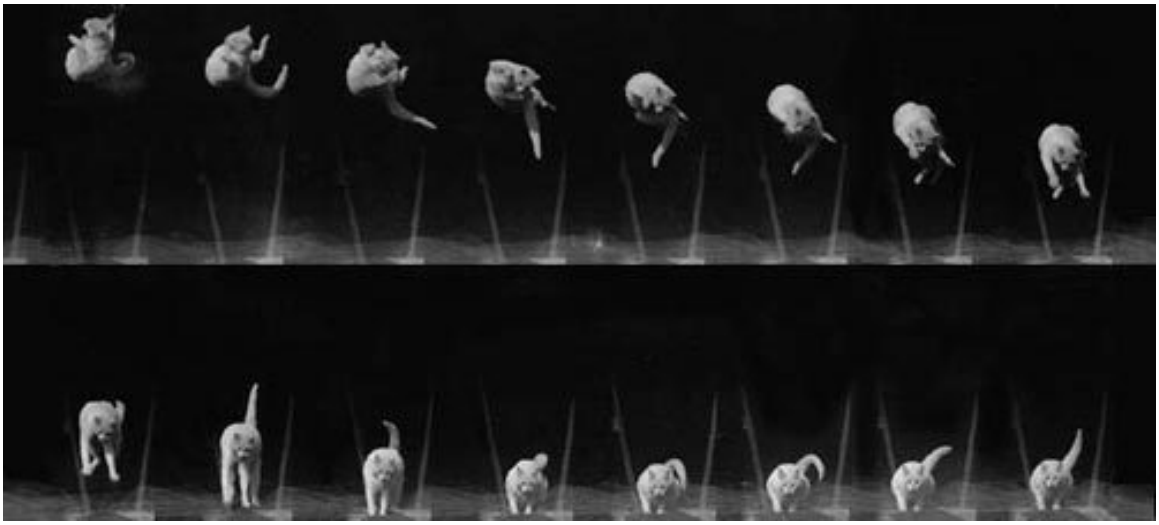


Figure 5: A cat is able to right itself as it falls to land on its feet, irrespective of its initial orientation.

Many animals have astonishing capabilities to achieve different maneuvers in the air by manipulating their body articulations. Cats are known for landing with feet from any initial falling condition [44, 64, 86] (Figure 5). Lizards swing their tails to stabilize their bodies during a leap [55]. Pigeons reorient their bodies to achieve a sharp turn when flying at low speed [80]. These behaviors inspire scientists and engineers to develop intelligent devices and control algorithms. This dissertation has a similar goal that we study how human body can change shape in the air to reduce damage at landing.

2.3 Human-in-the-loop interfaces

Without human guidance, fully automated optimization algorithms sometimes produce undesired solutions due to unexpected factors or situations. For instance, finding the optimal jumping motion with the desired height can be achieved in many different joint trajectories, and excessive usages of hips or heels make some optimal motions

look unnatural. In Chapter 5, we propose the semi-automatic learning framework which involves a human in the optimization process to effiently find optimal and natural motions within a short amount of time. This paradigm is so called *human-in-the-loop* (HITL) optimization, which has proven effective for various problems, such as vehicle planning [104] or interface optimization [77]. The level of user interaction varies from simply selecting of the generated solutions [88] to directly editing the search parameters and constraints [91]. Unlike most previous work which primarily focused on developing user interaction and visualization techniques for HTIL optimization systems, we develop a new controller design framework that exploit the nature of HITL computation paradigm.

2.4 Policy search algorithms

From Chapter 6 to 8, we describe policy search algorithms for finding optimal controllers. In this section, we will review existing algorithms that are classified into two categories based on the existance of simulation models. We will also cover related work on special cases, optimization of parameterized motor skills.

2.4.1 Model-free policy search algorithms

Chapter 6 and 7 are inspired by existing model-free policy optimization techniques where the policy is improved through a number of hardware trials [67, 51]. Unfortunately, these methods generally require hundreds of trials, which is unrealistic for tasks such as humanoid balancing and locomotion. One way to overcome this issue is to limit the parameter space by using task-specific primitives [72] or to provide a good initial trajectory by human demonstration [9]. However, it is not clear how to extend these approaches to dynamically unstable robots or tasks that cannot described by joint trajectories.

Various optimization techniques have been applied to improve the motion quality or the robustness of the controller. In character animation, a sampling-based method,

Covariance Matrix Adaption Evolution Strategy (CMA-ES) [34], has been frequently applied to discontinuous control problems, such as biped locomotion [100, 101, 102], parkour-style stunts[58, 32], or swimming [97]. To compensate the expensive cost of sampling-based algorithm, different approaches have been proposed, including exploiting the domain knowledge [100, 101, 102], shortening the problem horizons [89], or using a classifier to exclude infeasible samples [32]. Based on the previous success of CMA-ES, we proposed new sampling-based algorithms that resemble the evolution process of distribution.

2.4.2 Model-based policy search algorithms

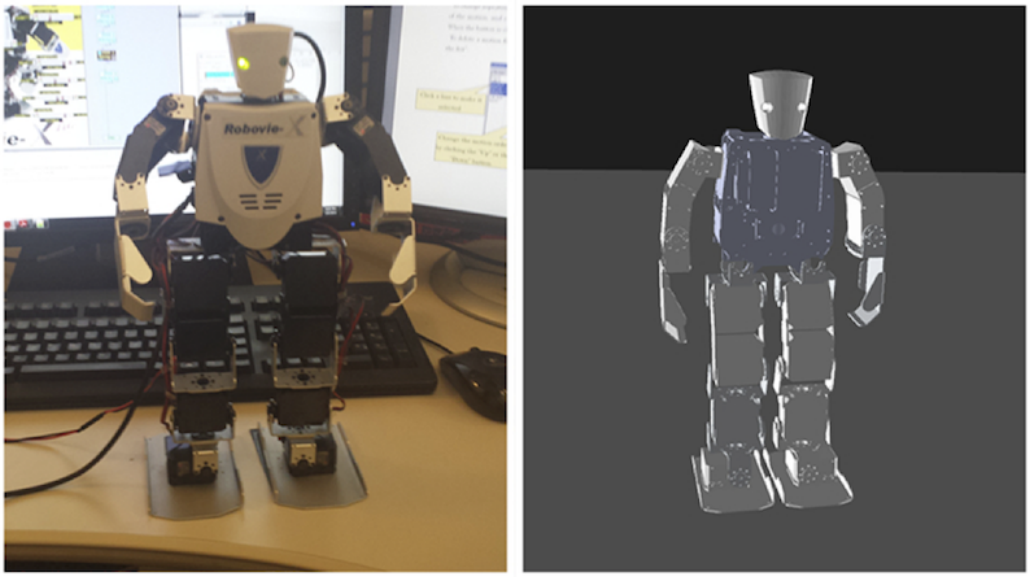


Figure 6: Difference between a real robot and its simulation model results different motions from same controllers.

Chapter 8 proposes a model-based policy search algorithms that utilizes simulation to reduce the number of hardware trials. In this category of algorithms, difference between a robot and its simulation model becomes a serious problem (Figure 6). Classical parameter identification techniques [46] partially solve this problem by fitting model parameters to experimental data, but they are still limited to factors that can actually be modeled. Furthermore, these approaches assume that the data set is large

enough to accurately estimate the parameters. In large and unstable systems such as humanoid robots, it is often difficult to collect enough data [106].

A number of researchers have attempted to overcome the drawbacks of these approaches by combining simulation and real-world data [96, 65, 76]. Abbeel et al. [6] used an inaccurate model to estimate the derivative of the cost with respect to the policy parameters. Ko et al. [48] used Gaussian Process to model the difference between a nonlinear dynamics model and the actual dynamics and applied the model to reinforcement learning for yaw control of a blimp. However, they do not iterate the process to refine the model. Deisenroth et al. [21] also used Gaussian Process for learning the dynamics model from scratch. Similarly, Morimoto et al. [68] used Gaussian Process for learning simplified dynamics of human locomotion. Sugimoto et al. [93] used *sparse pseudo-input Gaussian Process* (SPGP) that accounts both variances of inputs and outputs to handle sensor noises. Instead, Tangkaratt et al. [98] used *least-squares conditional density estimation* (LSCDE) to learn the dynamics model without Gaussian assumption on the transitions. Cutler et al. [15] trained a policy in multiple fidelity simulators with discretized actions. Ross and Bagnell [81] theoretically proved that their iterative system identification method converges even the system is not in the assumed class. Please refer to Section 6 of [50] for more complete survey on this topic.

2.4.3 Policy search algorithms for parametrized tasks

There is a large body of research work on generalization of learned motor skills to achieve new tasks, which is discussed in Chapter 7. da Silva *et al.* [18, 16, 17] introduced a framework to represent the policies of related tasks as a lower-dimensional piecewise-smooth manifold. Their method also classifies example tasks into disjoint lower-dimensional charts and model different sub-skills separately. Much research aimed to generalize example trajectories to new situations using dynamic movement

primitives (DMPs) to represent control policies [41]. A DMP defines a form of control policies which consists of a feedback term and a feedforward forcing term. Ude *et al.* [99] used supervised learning to train a set of DMPs for various tasks and built a regression model to map task parameters to the policy parameters in DMPs. Muelling *et al.* [69] proposed a mixture of DMPs and used a gate network to activate the appropriate primitive for the given target parameters. Kober *et al.* [52] trained a mapping between task parameters and meta-parameters in DMPs using a cost-regularized kernel regression. Through reinforcement learning framework, they computed a policy which is a probability distribution over meta-parameters. Matsubara *et al.* [62] trained a parametric DMP by shaping a parametric-attractor landscape from multiple demonstrations. Stulp *et al.* [92] proposed to integrate the task parameters as part of the function approximator of the DMP, resulting in more compact model representation which allows for more flexible regression. Neumann *et al.* [73] modified the existing learning algorithm (REPS) to learn a hierarchical controller that has parameterized options.

All these methods described above depend on collecting a set of examples. This presents a bottleneck to learning because an individual control policy needs to be learned for each task example drawn from the distribution of interest. da Silva *et al.* further proposed using unsuccessful policies as additional training samples to accelerate the learning process [16]. For dynamic motor skills which involve intricate balance tasks, unsuccessful policies generated during training a particular task are of no use to other tasks because they often lead to falling motion. Hausknecht *et al.* [36] demonstrated a quadruped robot kicking a ball to various distances, but whole-body balance was not considered in their work. Another challenge regarding dynamic tasks is that each task can be achieved by a variety of policies, some of which might be overfitting the task. Interpolating these overfitted policies can lead to unexpected results. In this dissertation, we proposed a new algorithm that tends to generate more coherent

mapping between task parameters and policy parameters because we simultaneously learn the policies for the entire range of the tasks.

The next chapter will describe falling strategies for virtual characters and real robots, which are essential for protecting humanoids from severe damage.

CHAPTER III

FALLING AND LANDING MOTION CONTROL FOR VIRTUAL CHARACTERS

Figure 7: A simulated character lands on the roof of a car, leaps forward, dive-rolls on the sidewalk, and gets back on its feet, all in one continuous motion.

This chapter introduces a new method to generate agile and natural human landing motions in real-time via physical simulation without using any mocap or prescripted sequences. We develop a general controller that allows the character to fall from a wide range of heights and initial speeds, continuously roll on the ground, and get back on its feet, without inducing large stress on joints at any moment. The character's motion is generated through a forward simulator and a control algorithm that consists of an airborne phase and a landing phase. During the airborne phase, the character optimizes its moment of inertia to meet the ideal relation between the landing velocity and the angle of attack, under the laws of conservation of momentum. The landing phase can be divided into three stages: impact, rolling, and getting-up. To reduce joint stress at landing, the character leverages contact forces to control linear momentum and angular momentum, resulting in a rolling motion which distributes impact over multiple body parts. We demonstrate that our control algorithm can be applied to a variety of initial conditions with different falling heights, orientations, and linear and angular velocities. Simulated results show that our algorithm

can effectively create realistic action sequences comparable to real world footage of experienced freerunners.

3.1 Motivation

One of the great challenges in computer animation is to physically simulate a virtual character performing highly dynamic motion with agility and grace. A wide variety of athletic movements, such as acrobatics or freerunning (parkour), involve frequent transitions between airborne and ground-contact phases. How to land properly to break a fall is therefore a fundamental skill athletes must acquire. A successful landing should minimize the risk of injury and disruption of momentum because the quality of performance largely depends on the athlete's ability to safely absorb the shock at landing, while maintaining readiness for the next action. To achieve a successful landing, the athlete must plan coordinated movements in the air, control contacting body parts at landing, and execute fluid follow-through motion. The basic building blocks of these motor skills can be widely used in other sports that involve controlled falling and rolling, such as diving, gymnastics, judo, or wrestling.

We introduce a new method to generate agile and natural human falling and landing motions in real-time via physical simulation without using motion capture data or pre-scripted animation (Figure 7). We develop a general controller that allows the character to fall from a wide range of heights and initial speeds, continuously roll on the ground, and get back on its feet, without inducing large stress on joints at any moment. Previous controllers for acrobat-like motions either precisely define the sequence of actions and contact states in a state-machine structure, or directly track a specific motion capture sequence. Both cases fall short of creating a generic controller capable of handling a wide variety of initial conditions, overcoming drastic perturbations in runtime, and exploiting unpredictable contacts.

Our method is inspired by three landing principles informally developed in freerunning community. First, reaching the ground with flexible arms or legs provides cushion time to dissipate energy over a longer time window rather than absorbing it instantly at impact. It also protects the important and fragile body parts, such as the head, the pelvis, and the tailbone. Second, it is advisable to distribute the landing impact over multiple body parts to reduce stress on any particular joint. Third, it is crucial to utilize the friction force generated by landing impact to steer the forward direction and control the angular momentum for rolling, a technique referred to as "blocking" in the freerunning community. These three principles outline the most commonly employed landing strategy in practice: landing with feet or hands as the first point of contact, gradually lowering the center of mass (COM) to absorb vertical impact, and turning a fall into a roll on the ground, with the head tightly tucked at impact moment.

However, translating these principles to control algorithms in a physical simulation is very challenging. During airborne, the controller needs to plan and achieve the desired first point of contact and the angle of attack, in the absence of control over the characters global motion in the air. Instead of solving a large, nonconvex two-point boundary value problem, we develop a compact abstract model which can be simulated efficiently for real-time applications. To strike the balance between accuracy and efficiency, our algorithm replans the motion frequently to compensate the approximation due to the simplicity of the model. When the character reaches the ground, the controller needs to take a series of coordinated actions involving active changes of contact points over a large area of human body. Our algorithm executes three consecutive stages, impact, rolling, and getting-up by controlling poses, momentum, and contacts at key moments. Furthermore, the airborne and landing phases are interrelated and cannot be considered in isolation: the condition for a successful landing defines the control goals for the airborne phase while the actions taken during

airborne directly impact the landing motion. We approach this problem in a reverse order of the action sequence: designing a robust landing controller, deriving a successful landing condition from this controller, and developing an airborne controller to achieve the landing condition.

We demonstrate that our control algorithm is general, efficient, and robust. We apply our algorithm to a variety of initial conditions with different falling heights, orientations, and linear and angular velocities. Because the motion is simulated in real-time, users can apply perturbation forces to alter the course of the character in the air. Our algorithm is able to efficiently update the plan for landing given the new situations. We also demonstrate different strategies to absorb impact, such as a dive roll, a forward roll, or tumbling. The same control algorithm can be applied to characters with very different body structures and mass distributions. We show that a character with unusual body shape can land and roll successfully. Finally, our experiments empirically showed that the algorithm induces smaller joint stress, except for the contacting end-effectors. In the worst case of our experiments, the average joint stress is still four times lower than landing as a passive ragdoll.

3.2 Overview

We introduce a physics-based technique to simulate strategic falling and landing motions from a wide range of initial conditions. Our control algorithm reduces joint stress due to landing impact and allows the character to efficiently recover from the fall. The character's motion is generated through a forward simulator and a control algorithm that consists of an *airborne phase* and a *landing phase*. These two phases are related by an appropriate *landing strategy*, which describes the body parts used for the first contact with the ground, a desired landing pose, and an ideal landing condition that describes the relation between landing velocities and the angle of attack in successful landing motions. We develop two most common types of landing

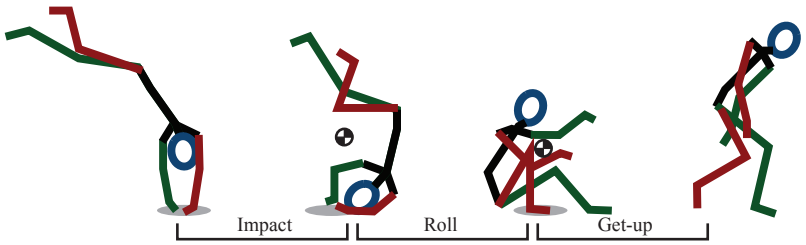


Figure 8: Three stages in the landing phase.

strategies: hands-first and feet-first, and introduce a sampling method to derive the ideal landing condition for each strategy.

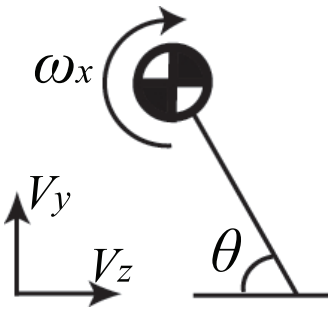
At the beginning of a fall, the character first decides on a landing strategy. During the airborne phase, the character optimizes its moment of inertia to achieve the ideal landing condition. The landing phase is divided into three stages: impact, rolling, and getting-up (Figure 8). The impact stage begins when the character reaches the ground. During the impact stage, the character leverages the friction forces from the ground to control linear and angular momentum. After the COM moves beyond the hand contact area, the character switches to the rolling stage in which continuous change of contact carries out. In preparation for standing up, the character needs to maintain the rolling direction and plant its feet on the ground. When the COM passes through the first foot, the character starts to elevate the COM in order to compete the landing process in an upright position.

3.3 Landing Strategy

Given an initial condition at the beginning of a fall, the character can choose to land with the hands-first strategy or the feet-first strategy. In general, the hands-first strategy is chosen only for aesthetics purpose because it is less robust and suitable only for falls with planar angular momentum (about the pitch axis). In contrast, the feet-first strategy can handle a wide range of arbitrary initial conditions because it includes an extensive foot-ground contact duration to modulate the momentum before

rolling. A landing strategy also includes a desired landing pose. Our algorithm only requires a partial pose to stretch the arms or legs at landing, depending on whether the hands-first or the feet-first strategy is chosen. We manually specify this partial pose for each strategy (Figure 9).

Figure 9: The left and middle are the desired landing poses for the hands-first strategy and the feet-first strategy, respectively. The right is the ready-to-roll pose for the feet-first strategy, which we track only the upper body.



An integral part of our landing strategy is the landing condition, a simple equation that compactly characterizes successful landing motions. If the character manages to turn a fall into a roll and gets back on its feet at the end of the roll, we consider it successful. Because a successful landing highly depends on whether the character is able to control the momentum at the moment of the first contact (T), our algorithm defines the landing condition as a relation between the global linear velocity $\mathbf{v}^{(T)}$, global angular velocity $\omega^{(T)}$, and the angle of attack $\theta^{(T)}$, which approximates the global orientation of the character. The actual coefficients of the landing condition depend on the design of the landing controller, which cannot be derived analytically, but can be learned from examples generated by the landing controller. We apply a sampling method, similar in spirit to the approach Coros *et al.* [12] presented for biped locomotion, to determine the landing condition for a particular landing strategy.

For the hands-first strategy with planar motion, we consider a four-dimensional space spanned by $\theta^{(T)}$, $v_y^{(T)}$, $v_z^{(T)}$ and $\omega_x^{(T)}$. Given a sample in the parameter space, we

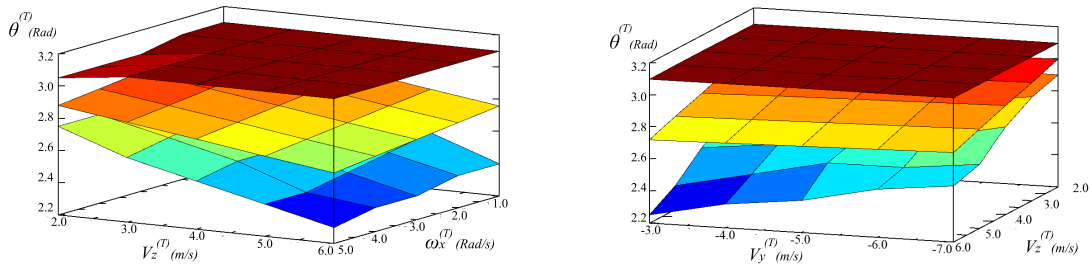


Figure 10: Samples for hands-first landing strategy. Successful samples are bounded between top and bottom planes along $\theta^{(T)}$ axis. The middle plane, average of the two, indicates the linear relation of the ideal landing condition.

run our landing controller to test whether the character can successfully get up at the end. Empirical results from thousands of random samples show that the successful region is mostly continuous and linear (Figure 10). We can bound the successful samples in the $\theta^{(T)}$ axis using two hyperplanes. Taking the average of the maximum and the minimum planes, we derive a linear relation between the angle of attack and the landing velocities as

$$\theta^{(T)} = a\, v_y^{(T)} + b\, v_z^{(T)} + c\, \omega_x^{(T)} + d \tag{1}$$

where a, b, c, and d are the coefficients of the fitted hyperplane. Note that Equation (1) is a sufficient but not necessary condition for successful landing. Most points between the maximal and minimal hyperplanes also lead to successful landing motions. This means that even when the character cannot meet the landing condition exactly, it still has a good chance to land successfully. For the feet-first strategy, in theory, we need to consider all six dimensions of linear velocity and angular velocity. However, our empirical results show that non-planar velocities do not affect $\theta^{(T)}$ as long as they stay within a reasonable bound (Figure 11). As a result, the feet-first strategy is able to handle non-planner falling motion using the same parameters (but different coefficients) in Equation (1).

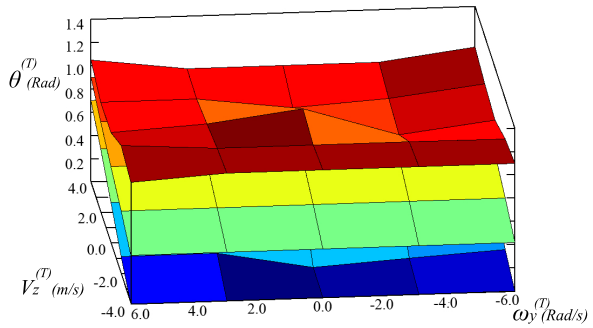


Figure 11: Samples in the space of $v_z^{(T)}$, $\omega_y^{(T)}$, and $\theta^{(T)}$. The spinning velocity $\omega_y^{(T)}$ has minimal effect on the success of a sample.

3.4 Airborne Phase

Once the character decides on a landing strategy, the goal of the airborne phase is to achieve the corresponding landing pose and landing condition. Because momentum is conserved in air, the linear velocity, the total airborne time T, as well as the angular momentum are already determined by the initial condition of the fall. However, the character can still control the angular velocity $\omega_x^{(T)}$ and the angle of attack $\theta^{(T)}$ by varying its pose (i.e. actuated degrees of freedom (DOFs) excluding the global position and orientation) to change the moment of inertia. To most effectively achieve the desired landing condition, we design our airborne algorithm based on the strategy employed in platform diving competition, where a highly trained athlete performs a sequence of predefined poses to manipulate the final orientation and angular velocity.

To this end, our airborne controller uses a PD servo to track a sequence of poses that lead to the ideal landing condition. The sequence of poses is replanned frequently to correct the errors caused by perturbation and numerical approximation. Each time the algorithm makes a new plan, an optimal sequence of poses from the current moment to the landing moment is computed. This sequence starts with the current pose $\mathbf{q}_0$ and ends at the desired landing pose $\mathbf{q}_T$ (determined by the landing strategy), with a duration of T seconds. Our control algorithm searches for an intermediate pose $\mathbf{q}^*$ and a duration Δt^*, such that the character can reach the ideal landing condition by

changing to $\mathbf{q}^*$ immediately and holding the pose $\mathbf{q}^*$ for Δt^* seconds before changing to the final pose $\mathbf{q}_T$.

We formulate an optimization to solve for an intermediate pose $\mathbf{q}$ and its holding duration Δt that can best achieve the ideal landing condition. The cost function $g(\mathbf{q}, \Delta t)$ is defined in Equation 2.

$$g(\mathbf{q}, \Delta t) = \theta^{(T)}(\mathbf{q}, \Delta t) - a\ v_y^{(T)} - b\ v_z^{(T)} - c\ \omega_x^{(T)}(\theta^{(T)}) - d \tag{2}$$

Note that $\omega_x^{(T)}$ is a function of $\theta^{(T)}$ because we need global orientation of the character at time T to compute the global angular velocity. If we can compute $\theta^{(T)}$, Equation (2) can be readily evaluated. Unfortunately, for a complex 3D multibody system, an analytical solution for $\theta^{(T)}$ is not available. We could resort to numerical simulation of the entire airborne phase, in which the character goes through $\mathbf{q}_0$, $\mathbf{q}^*$, and $\mathbf{q}_T$ subsequently. However, involving forward simulation of a full skeleton in the cost function is too costly for our real-time application. Instead, we simulate a simple proxy model with only six DOFs. When the character is holding a pose, the proxy model behaves like a rigid body with a fixed inertia. When the character transitions from one pose to another, we assume the inertia of the proxy model changes linearly within a fixed duration Δt_C ($\Delta t_C = 0.1s$ in our implementation). By simulating the proxy model for the duration of T, we obtain the angle of attack $\theta^{(T)}$ and angular velocity $\omega^{(T)}$ as follows.

$$\begin{aligned}\mathbf{R}(\theta^{(T)}) = \mathbf{R}(\theta^{(0)}) &+ \int_{t=0}^{\Delta t_c} [\mathbf{I}_A^{-1}(t)\mathbf{L}]\mathbf{R}(\theta^{(t)})dt \\ &+ \int_{t=\Delta t_c}^{\Delta t_c+\Delta t} [\mathbf{I}^{-1}(\mathbf{q}, \theta^{(t)})\mathbf{L}]\mathbf{R}(\theta^{(t)})dt \\ &+ \int_{t=\Delta t_c+\Delta t}^{2\Delta t_c+\Delta t} [\mathbf{I}_B^{-1}(t)\mathbf{L}]\mathbf{R}(\theta^{(t)})dt \\ &+ \int_{t=2\Delta t_c+\Delta t}^{T} [\mathbf{I}^{-1}(\mathbf{q}_T, \theta^{(t)})\mathbf{L}]\mathbf{R}(\theta^{(t)})dt; \end{aligned} \tag{3}$$

$$\omega^{(T)} = \mathbf{I}^{-1}(\mathbf{q}_T, \theta^{(T)})\mathbf{L} \tag{4}$$

where $\mathbf{R}$ is the rotation matrix, $\mathbf{I}(\mathbf{q})$ is an inertia matrix evaluated at pose $\mathbf{q}$, and $\mathbf{L}$ is the angular momentum. $\mathbf{I}_A(t)$ is an interpolated inertia matrix between $\mathbf{I}(\mathbf{q}_0)$ and $\mathbf{I}(\mathbf{q})$, and similarly, $\mathbf{I}_B(t)$ is an interpolated matrix between $\mathbf{I}(\mathbf{q})$ and $\mathbf{I}(\mathbf{q}_T)$. The operator [ ] represents the skew symmetric matrix form of a vector.

To formulate an efficient optimization for real-time application, we represent the domain of intermediate pose as a finite set of candidate poses, instead of a continuous high-dimensional Euclidean space. This simplification is justified because a handful of poses is sufficient to effectively change the moment of inertia of the character. As a preprocess step, our algorithm automatically selects the candidate set $\mathbf{Q}$ from a motion capture sequence in which the subject performs range-of-motion exercise. The selection procedure begins with a seed pose $\bar{\mathbf{q}}_0$ and increments the set by adding a new pose $\bar{\mathbf{q}}_{new}$ which maximizes the diversity of inertia (Equation 5). In our experiment, 16 poses are sufficient to present a variety of moment of inertia (Figure 12).

$$\bar{\mathbf{q}}_{new} = \underset{\mathbf{q}\in M}{\operatorname{argmax}}(\min_{\bar{\mathbf{q}}_j \in Q} \|I(\mathbf{q}) - I(\bar{\mathbf{q}}_j)\|)\} \tag{5}$$

where M contains the poses in the range-of-motion sequence, Q contains the currently selected candidate poses, and $I(\mathbf{q})$ computes the inertia of pose $\mathbf{q}$.

To find optimal $\mathbf{q}^*$ and Δt^* for each plan, we start from the current pose as $\mathbf{q}_0$ and loop over each candidate pose in Q. For each candidate pose $\bar{\mathbf{q}}_i$, we search for the best Δt such that $g(\bar{\mathbf{q}}_i, \Delta t)$ is minimized. The search can be done efficiently using one-dimensional Fibonacci algorithm and the proxy-model simulation. The optimal intermediate pose $\mathbf{q}^*$ and its optimal duration Δt^* are used for airborne control.

By design, our algorithm trades off accuracy for efficiency; we use a fast but less accurate proxy-model simulation and a small set of predefined poses. Our algorithm is very efficient so that the character can frequently reassess the situation and replan new poses to correct any errors or adapt to unexpected perturbations.

The frequency of replanning can be determined differently for $\mathbf{q}^*$ and Δt^*. In our

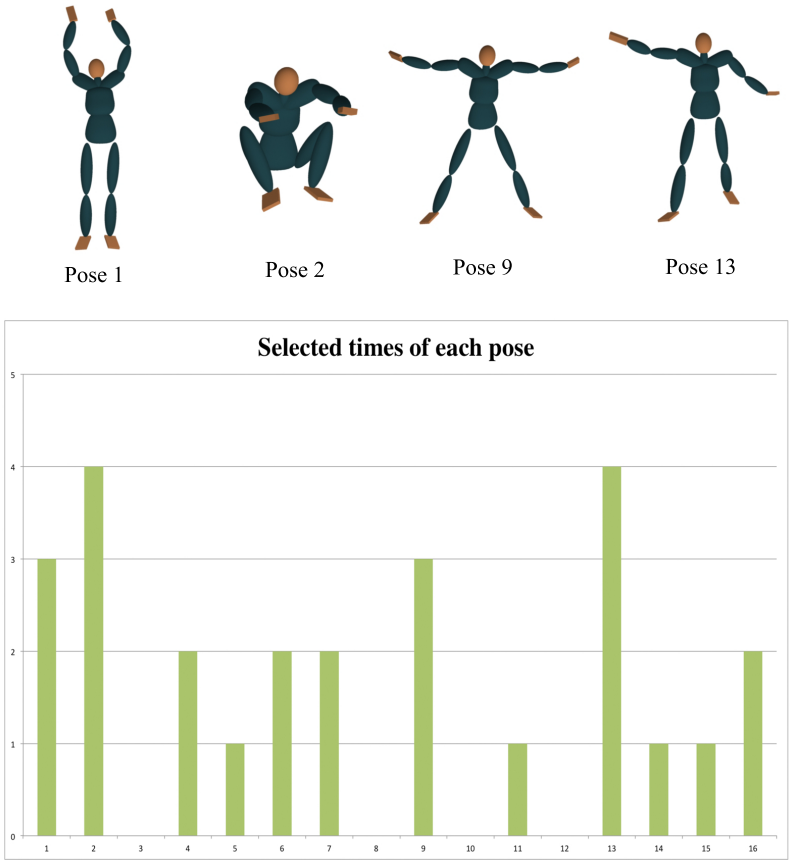


Figure 12: Among 16 poses in **Q**, pose 1, 2, 9, and 13 are frequently selected by the airborne controller

implementation, we replan $\mathbf{q}^*$ at a much lower frequency than Δt^* to avoid unnatural frequent change of poses. In addition, we stop replanning when the character is within 0.3 seconds away from the ground.

3.5 Landing Phase

During landing, the character braces for impact, executes rolling action, and gets up on its feet. Although these three stages take very different actions, they share common control goals: modulating the COM and posing important joints. We apply the same control mechanism via *virtual forces* and *PID joint-tracking* to produce the final control forces for the forward simulator (Figure 13).

Virtual forces are effective in controlling the motion of the COM. To achieve a

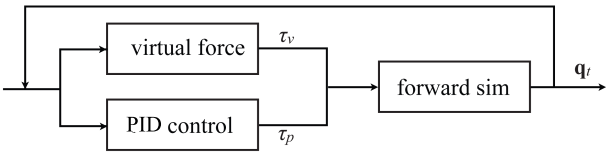


Figure 13: Landing phase controller.

desired acceleration of the COM, $\ddot{\mathbf{c}}$, we compute the virtual force as $\mathbf{f}_v = m\ddot{\mathbf{c}}$ where m is the mass of the character. The equivalent joint torque as if applying $\mathbf{f}_v$ to a point $\mathbf{p}$ on the body is $\tau_v = \mathbf{J}^T(\mathbf{p})\mathbf{f}_v$, where $\mathbf{J}(\mathbf{p})$ is the Jacobian computed at the body point $\mathbf{p}$. If $\mathbf{p}$ is on a body node in contact with the ground, we apply the opposite force ($\mathbf{f}_v = -m\ddot{\mathbf{c}}$) in order to generate a ground reaction force that pushes the COM in the desired direction. To prevent the character from using excessively large joint torques, we limit the magnitude of the sum of virtual forces. A successful landing motion also requires posing a few important joints at each of the three stages. We track these partial poses with PID servos: $\tau_p = k_p(\bar{q} - q) + k_i \int (\bar{q}_t - q_t)dt - k_v\dot{q}$, where k_p, k_i and k_v are the proportional, integral, and derivative gains respectively, and $\bar{q}$ is the desired joint angle. The final control torque is $\tau_v + \tau_p$. We limit the magnitude of the virtual force to $3000N$ to prevent excessive usage of joint torques.

3.5.1 Impact Stage

Impact stage is the most critical stage during landing, which requires careful control and execution. Human athletes tend to act like a spring to absorb the effect of impact by flexing their joints between the points of first contact and the COM. Meanwhile, they also utilize friction force from the ground contact to adjust forward linear momentum and angular momentum. Applying these principles, our algorithm utilizes virtual force technique to achieve contact forces for desired momentum. In addition, we use joint tracking to provide sufficient stiffness at contacting limbs and smooth transition to the next stage. If the character chooses the hands-first strategy, the final pose at the end of compression can seamlessly connect to the rolling stage.

With the feet-first strategy, an additional "thrusting" step is required to transition to the rolling stage. We define a "ready-to-roll" pose that guides the character toward a rolling motion (Figure 9, Right). During this additional step, the character tracks the ready-to-roll pose while using its feet to thrust forward after its COM compressed to the lowest point (Figure 14).

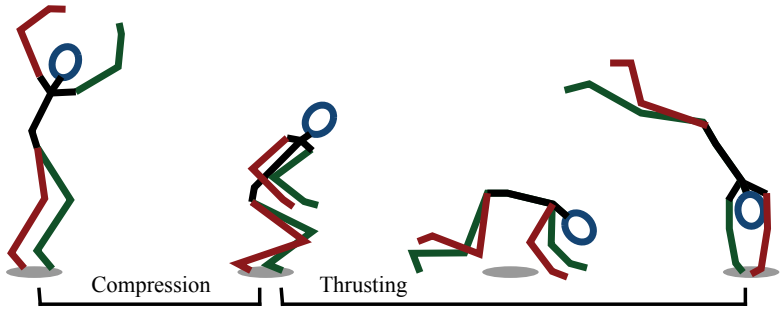


Figure 14: Two-step impact stage for the feet-first strategy.

Virtual force. The most important goal during the impact stage is to stop the downward momentum before the character tragically crashes into the ground. We do so by applying virtual forces to control the vertical position and velocity of the COM. In addition, our algorithm favors virtual forces that result in temporally smooth ground reaction forces to distribute the impact evenly over time. With these control goals, our algorithm aims to use constant acceleration of the COM to achieve the desired COM position $\bar{c}_y$ and velocity $\bar{\dot{c}}_y$ from the current state (c_y and $\dot{c}_y$).

$$\ddot{c}_y = \frac{1}{2}(\bar{\dot{c}}_y^2 - \dot{c}_y^2)/(\bar{c}_y - c_y) \tag{6}$$

A virtual force of $-m\ddot{c}_y$ in the vertical direction is then evenly distributed to the end-effectors that are in contact with the ground.

Virtual forces in the horizontal direction are important to achieve the desired forward linear momentum and angular momentum at the end of compression, or to achieve the desired forward thrust for the feet-first strategy. We use a simple feedback mechanism to compute the desired horizontal acceleration of the COM.

$$\ddot{c}_{x/z} = k_v(\bar{\dot{c}}_{x/z} - \dot{c}_{x/z}) \tag{7}$$

Table 1: Control parameters.

	Hip	Lower spine	Upper spine	Neck	Knee
k_p	90.0	300.0	180.0	10.0	60.0
k_d	20.0	60.0	40.0	2.0	13.0
	Ankle	Clavicle	Shoulder	Elbow	Wrist
k_p	15.0	180.0	120.0	60.0	9.0
k_d	6.0	40.0	27.0	13.5	4.0
$\bar{c}_y$	$\bar{\dot{c}}_y$	$\bar{\dot{c}}_{x/z}$	k_v	k_p (Eq 8)	ω_{MAX}
0.4m	0.0m/s	4.0m/s	500	800	3.3 Rad/s

where $\bar{\dot{c}}_{x/z}$ is the desired COM velocity in forward and lateral directions and k_v is the damping coefficient. The corresponding virtual force is distributed to the contacting end-effectors inversely proportional to their distances to the COM.

Joint tracking. In addition to virtual forces, we use PID servos to maintain joint angles of the torso and limbs that are not in contact, while limbs in contact with the ground act like viscous dampers (PID control with a zero spring coefficient). We also use PID control to keep the chin tucked to reduce the chance of the head impacting the ground. Please see Table 1 for all the parameters in our implementation. We set the constant integral gain k_i of contacting limbs as 50, and 0 for all other joints.

3.5.2 Rolling Stage

Once the character's COM passes the hand-ground contact area with sufficient forward linear and angular momentum, rolling becomes a relatively easy task. As long as the character is holding a pose with a flexed torso, a reasonable rolling motion will readily carry out. If the character wishes to land back on its feet and get up after rolling, it must also maintain forward momentum and lateral balance during the roll.

Virtual force. To this end, we apply a virtual force to guide the horizontal position of the COM toward the feet area, while restricting it above the support polygon formed by contact points. The virtual force is applied on the character's hands so that it can use the entire upper body to maintain momentum and balance. The virtual force produces the desired acceleration of the COM computed using a

feedback mechanism:

$$\ddot{c}_{x/z} = k_p(\bar{c}_{x/z} - c_{x/z}) \tag{8}$$

where the desired position $\bar{c}$ is set to be the location of the left foot.

Joint tracking. During rolling, the character tracks a simple pose to tuck the head, flex the torso, and position the legs appropriately. We treat legs asymmetrically to both facilitate momentum control and improve the aesthetics of the motion. When the character rolls on its back, it brings the left knee closer to the chest and casually stretches the right leg. This arrangement helps the character to regulate the angular velocity using the right leg while getting ready to stand up on its left foot. Based on the forward angular velocity at the beginning of the rolling stage, we adjust the desired tracking angles for the right knee as:

$$\theta_R = max((1 - \omega_x/\omega_{MAX})\pi, 0) \tag{9}$$

3.5.3 Getting-Up Stage

The last stage of landing phase is to stand up using the remaining forward momentum. When the COM passes the foot contact, the character will start to elevate its COM to a desired height.

Virtual force. Similar to previous stages, we again apply virtual forces on the feet and the hands to control the vertical and the horizontal positions of the COM respectively. We compute $\ddot{c}_y$ using the same formula from Section 3.5.1 with different desired height of the COM. For $\ddot{c}_{x/z}$, we use the same formula as in Section 3.5.2.

Joint tracking. During the getting-up stage, our algorithm simply tracks the torso and the head to straighten the spine and untuck the chin.

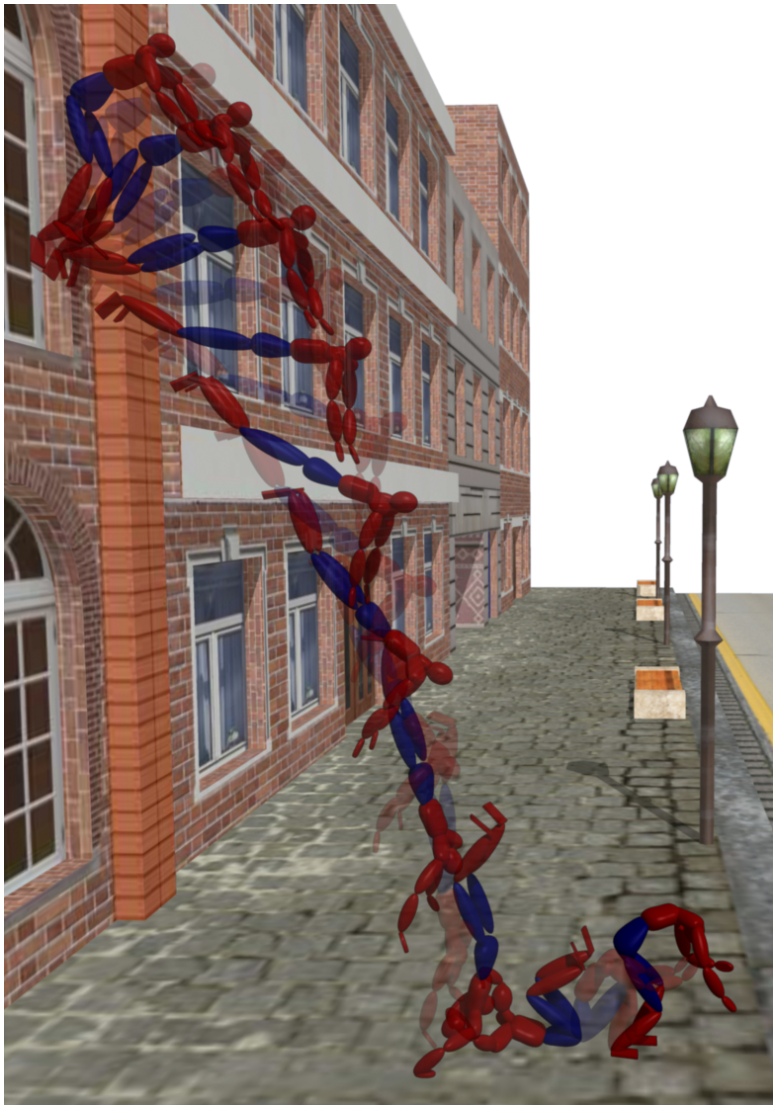


Figure 15: Hands-first landing motion.

3.6 Results

To evaluate the generality of our algorithm, we simulated landing motions with a wide range of initial conditions (Table 2), various landing styles (hands-first, feet-first, consecutive rolls), and different skeleton models. We also demonstrated that our algorithm is robust to unpredicted runtime perturbations and different physical properties of the landing surface. Please see the accompanying video to evaluate the quality of our results.

Table 2: Initial conditions of the examples shown in the video (in order of appearance)

Hands-first landing strategy						
$\vec{C}_{y(\mathrm{m})}$	$v_{x(\mathrm{m/s})}$	$v_{y(\mathrm{m/s})}$	$v_{z(\mathrm{m/s})}$	$\omega_{x(\mathrm{Rad/s})}$	$\omega_{y(\mathrm{Rad/s})}$	$\omega_{z(\mathrm{Rad/s})}$
10.6	0.0	0.0	4.0	8.7	0.0	0.0
5.8	0.0	0.0	2.3	5.0	0.0	0.0
10.6	0.0	0.0	6.0	2.5	0.0	0.0
2.5	0.0	0.4	8.0	5.0	0.0	0.0
Feet-first landing strategy						
$\vec{C}_{y(\mathrm{m})}$	$v_{x(\mathrm{m/s})}$	$v_{y(\mathrm{m/s})}$	$v_{z(\mathrm{m/s})}$	$\omega_{x(\mathrm{Rad/s})}$	$\omega_{y(\mathrm{Rad/s})}$	$\omega_{z(\mathrm{Rad/s})}$
6.0	0.0	0.0	5.0	4.0	-1.0	-5.8
2.7	0.0	-1.0	0.0	0.0	0.0	0.0
5.5	1.0	0.0	0.0	0.0	5.0	0.0
9.6	-2.0	0.0	-3.5	0.9	2.1	-3.9

Feet-first landing strategy. The most recommended landing strategy from freerunning community is the feet-first landing. Our results verify that the feet-first landing strategy is indeed very robust for falls with arbitrary linear and angular momentum. There are two key advantages of using feet as the first point of contact. First, average human has longer and stronger legs than arms. Using legs to land provides more time and strength to compress and absorb vertical impact. Second, the feet-first strategy has an additional thrusting step after compression and before rolling stage. During the thrusting step, the character can utilize the contact forces to drastically change the linear and angular velocity in preparation for rolling. Our results show that a successful forward roll can be carried out even when the character is falling with backward and lateral linear velocity or nonplanar angular velocity.

For the feet-first strategy, the coefficients of the landing condition in Equation (1) are: $a = -0.01, b = -0.06, c = -0.03$, and $d = 0.45$. When the character transitions to the rolling stage, we specified an asymmetric ready-to-roll pose to increase the visual appeal of the motion.

Hands-first landing strategy. Using hands as the first point of contact can generate visually pleasing stunts (Figure 15). For falls with dominant planar velocity (v_z

and ω_x), the hands-first strategy performs as well as the feet-first strategy. However, when the initial condition has large lateral linear momentum or angular momentum in yaw and roll axes, the hands-first strategy becomes less robust. Unlike the feet-first strategy, which has an additional thrusting step, the hands-first strategy is unable to change forward direction drastically after landing. This imposes stringent conditions on the contact forces because, in order to roll successfully, the contact forces must counteract non-planner momentum, while stopping downward momentum and maintaining forward momentum. Such forces usually violate the unilateral constraint of ground reaction force.

For the hands-first strategy, the coefficients of the landing condition are: $a = -0.01, b = -0.06, c = -0.03$, and $d = 3.08$. Note that the coefficients are identical to those of the feet-first strategy except for the constant term, indicating that the gradient of the angle of attack with respect to the landing velocity is the same between feet-first and hands-first landing strategies.

Consecutive rolls. Once the character starts rolling, it is rather effortless to continue on. By looping the end of the rolling stage back to the beginning, we showed that the character was able to make two consecutive rolls to break a fall with large forward speed. Falling on multiple surfaces is also easy to simulate using our controller. One example demonstrated a continuous sequence of the character landing on the roof of a car, leaping forward, landing again on the sidewalk, and finishing with a dive roll (Figure 7). With our controller, a variety of impressive action sequences can be generated easily without any recorded or pre-scripted motions.

Different skeleton models. The character model we used to generate most examples has a height of 164cm, a weight of 59 kg, and 49 DOFs. The controllers designed for this character can be applied to a drastically different character whose torso is twice as long and twice as wide, comparing to the default character. It also has very

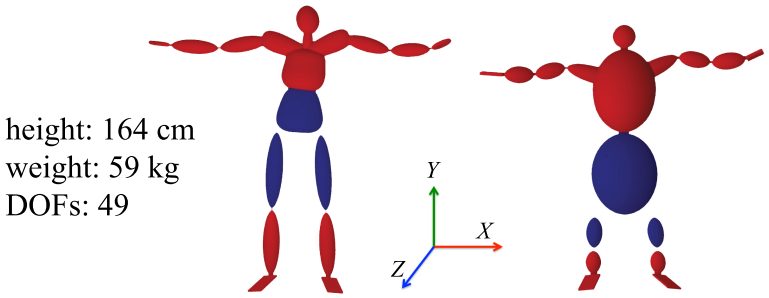


Figure 16: Left: The character model used for most examples. Right: A character with a disproportionately large torso and short legs.

short legs and a small head (Figure 16). We tested both hands-first and feet-first landing strategies on this new character. The results are similar in quality to the default character, although the new character hits its head on the ground because it is difficult to tuck the head with such a short neck. All the control parameters remain the same for the second character, except for $\bar{c}_y$ increasing by $5cm$ and the desired landing angle increasing by $0.25rad$.

Runtime perturbations. One great advantage of physical simulation is that the outcome can be altered on the fly based on user interactions. We demonstrated the interactivity of our simulation in two different ways. First, the user can directly "drag" the character to a different location or orientation when the character is in the air. This example shows off robustness and efficiency of our airborne controller. As the character being relocated, it starts to recalculate and finds a new plan to execute in real-time. Second, we let the user shoot cannons at the character as a source of external forces. When a cannon hits the character, it exerts force and torque on the character, causing a passive response followed by active replanning and execution.

Different landing surfaces. We tested our controller on surfaces with different elasticities and friction coefficients. When the character lands on an elastic surface, such as a gymnastic floor or a trampoline, the character tumbles in the air instead

of rolling on the ground. We generated a continuous sequence where the character stopped the fall on an elastic surface by tumbling three times and finishing with a forward roll. This example shows that various interesting acrobatic sequences can be generated by simply concatenating our falling and rolling controllers repeatedly. In another example, we reduced the friction coefficient to simulate an icy surface. The character was able to use the same control algorithm to roll, but failed to stand up at the end.

3.6.1 Evaluation

Performance. All the results shown in the video were produced on a single core of 3.20GHz CPU. Our program runs at 550 frames per second. The bottleneck of the computation is the optimization routine in the airborne controller. We use Open Dynamic Engine to simulate the character. The time step is set at 0.2 millisecond, and runs the airborne optimization in 50 Hz.

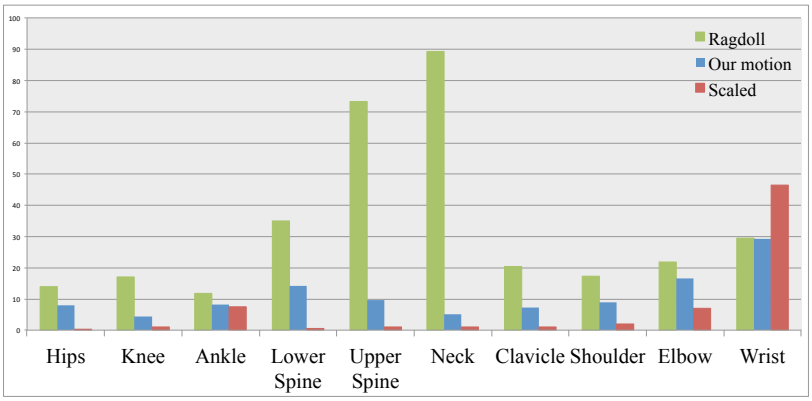


Figure 17: Maximal stress for each joint from a hands-first landing motion. Results are quantitatively similar across all of our simulations. Green: Ragdoll motion. Blue: Our motion. Orange: Joint stress scaled by mass.

Joint stress. We approximated joint stress as the constraint force that holds two rigid bodies together at a joint. For each joint, we computed the maximal joint stress during the landing phase (Figure 17). We observed that, in most trials, the joints

which endure the most impact are those connected to contacting end-effectors (i.e. hands or feet). The spine joints (lumbar and thoracic vertebrae) and hip joints are also subject to large impact. However, when we scaled each joint by the total mass it supports (e.g. the hip joint supports the mass of the entire leg), we found that the joint stress has low variance across the entire character's body, with the exception of the joints near the end-effectors.

When we compared the joint stress between our motion and a passive ragdoll motion with the same initial condition, the ragdoll motion caused much more damage on the neck and the spine (Figure 17). In fact, the only joints that endured similar amount of stress were those used for the first point of contact (e.g. wrists or ankles). These results validate that our controller indeed produces safer landing motion and protects important body parts. We repeated the experiments for different initial conditions. In the worst case of our experiments, the average joint stress is still four times lower than landing as a passive ragdoll. The data also show that our controller generates less damaging landing motion even when the character cannot roll successfully, such as dropping from 20 meters.

Comparison with video footages. We compared our simulated motion side-by-side with a collection of video footages ([7]). The simulations are based on the same landing strategy and our best guess of the initial conditions from the videos. Although it is not possible to achieve identical motions, results show that our motion is qualitatively similar to the video footages.

3.6.2 Limitations

The main limitation of our work is the lack of balance control after the character stands up. There are many existing balance control algorithms we could implement. However, we chose to defer the implementation until we decide on what the character's next action should be. In the freerunning scenario, the character transitions

to running motion seamlessly right after a roll. If freerunning is our goal, we would modify the current get-up control algorithm to provide more forward thrust. Other possibilities of the next action include walking, stepping, jumping, or standing still. Different next actions will result in different balance strategies. Ideally, a character should be equipped with motor skills to execute all different balance strategies and autonomously determines which strategy to execute, but this is considered out of the scope of this work.

Another limitation is the predefined landing pose for each landing strategy. This inflexibility can negatively affect the character's ability to adapt to different environments. For example, if the character lands on a narrow wall, the landing pose needs to be adjusted on the fly. One possible solution is to use a simple inverse kinematics method to compute desired joint angles before landing.

Figure 18: Feet-first landing motion.

3.7 Discussion

We introduced a real-time physics-based technique to simulate strategic falling and landing motions. Our control algorithm reduces joint stress due to landing impact and allows the character to efficiently recover from the fall. Given an arbitrary initial position and velocity in the air, our control algorithm determines an appropriate landing strategy and an optimal sequence of actions to achieve the desired landing velocity and angle of attack. The character utilizes virtual forces and joint-tracking

control mechanisms during the landing phase to successfully turn a fall into a roll. We demonstrated that our control algorithm is general, efficient, and robust by simulating motions from different initial conditions, characters with different body shapes, different physical environments, and scenarios with real-time user perturbations. The algorithm guides the character to land safely without introducing the large stress at every joint except for the contacting end-effectors.

Freerunning is a great exemplar to demonstrate human athletic skills. Those wonderfully simple yet creative movements provide a rich domain for future research directions. Based on the contribution of this work, we would like to explore other highly dynamic skills in freerunning, such as cat crawl, underbar, or turn vault. These motions are extremely interesting and challenging to simulate because they involve sophisticated planning and control in both cognitive and motor control levels, as well as complex interplay between the performer and the environment.

The landing strategies described in this work are suitable for highly dynamic activities, but not optimal for low-clearance falls from standing height. There is a vast body of research work in biomechanics and kinesiology studying fall mechanics of human from standing height. One future direction of interest is to integrate this domain knowledge with physical simulation tools to explore new methods for fall prevention and protection.

CHAPTER IV

MULTIPLE CONTACT PLANNING FOR HUMANOID FALLS

This chapter introduces a new planning algorithm to minimize the damage of humanoid falls by utilizing multiple contact points. Given an unstable initial state of the robot, our approach plans for the optimal sequence of contact points such that the initial momentum is dissipated with minimal impacts on the robot. Instead of switching among a collection of individual control strategies, we propose a general algorithm which plans for appropriate responses to a wide variety of falls, from a single step to recover a gentle nudge, to a rolling motion to break a high-speed fall. Our algorithm transforms the falling problem into a sequence of inverted pendulum problems and use dynamic programming to solve the optimization efficiently. The planning algorithm is validated in physics simulation and experimentally tested on a BioloidGP humanoid.

4.1 Motivation

A humanoid in an interactive environment is often exposed to the risk of falling due to unexpected contacts or perturbations. A fall can potentially cause detrimental damage to the robot and enormous cost to repair. To reduce the likelihood of damaging robots during online operations, researchers and engineers often cover the robot exterior with soft guards to absorb the impact of falls [63, 49]. Though practical, these extra parts can potentially limit the range of motion or change the contact behaviors.

Alternatively, the robot can learn how to fall safely like humans do. Fujiware *et al.*

[28, 30, 29, 27, 26] proposed fall strategies inspired by *Ukemi*, a set of techniques used in *Judo*. Ogata *et al.* [75, 74] evaluated the risk of falls using predicted the zero moment point (ZMP) and optimized the center of mass trajectory to reduce damage. Ruiz-del-Solar *et al.* [84, 83] designed falling sequences for simulated robots playing soccer. Wang *et al.* [103] directly solved joint trajectories of a three-link robot subject to the full-body dynamics. Lee and Goswami [53] proposed a control strategy that reorients the robot to fall on its backpack. Yun and Goswami [109] described a "Tripod" strategy that utilizes a swing foot and two hands to stop the fall with an elevated center of mass. Goswami *et al.* [31] proposed a direction-changing fall control strategy that utilizes foot placement and inertia shaping to protect both the robot and objects/humans in the surroundings.

These existing methods are effective for specific scenarios in which the perturbations are assumed within certain range. To select the best method for the given scenario, an additional decision making process that classifies initial conditions is required. In this chapter, we hypothesize that there exists a continuous space of falling strategies which can be characterized by the sequence of contact positions on the robot. In this hypothesized space, the existing falling techniques can be viewed as special cases. For example, the strategy proposed by Ogata *et al.* [75] employed a single contact at hands. Fujiwara *et al.* [26] proposed to use two contacts, knees and hands, to stop a fall with higher initial momentum. In *Judo*, multiple contact points from shoulders to hips are used during a forward roll to break a high-speed fall [5].

We introduce a general algorithm that unifies the existing techniques for falling strategies. Our algorithm reacts to a wide range of initial perturbations by automatically determining the total number of contacts, the order of contacts, and the position and timing of contacts. The sequence of contact is optimized such that the initial momentum is dissipated with minimal damage on the robot. We introduce an abstract model, which consists of an inverted pendulum and a telescopic "stopper",

to approximate the reactive motion when humans fall. The efficient optimization is achieved by recursive planning in the space of abstract model using dynamic programming. We demonstrate that our algorithm plans various falling strategies with different contact sequences on simulated humanoids and on the actual hardware.

4.2 The Problem

Consider a biped humanoid on the ground with an unstable initial state due to a large initial velocity. If the robot cannot recover its balance, what is the least damaging way to fall on the ground? It is well known that the damage incurred at the instance of impact is mainly due to the sudden change of momentum, which requires a large impulse applied in a very short period of time. To completely stop a fall, the robot has no choice but to absorb the initial momentum in its entirety. However, the change of momentum needs not to happen so suddenly. With an ideal control policy, the robot should be able to reduce the magnitude of the impulse at peak by distributing one large impulse to multiple smaller impulses over multiple contacts with the ground.

In the discretized time domain, we define the *instantaneous impulse* at each time step n as

$$j_n = \int_{hn}^{h(n+1)} f_y(t)dt = hf_y(hn) \tag{10}$$

where h is the discretized time interval and $f_y(t)$ is the sum of the vertical contact forces at time t. Because the robots considered in this work are made of hard materials, the contacts between the robot and the ground can be approximated as collisions between two ideal rigid bodies. With this assumption, the largest instantaneous impulse during each contact period typically occurs at the *impact moment*, the instance when the contact first establishes. The maximum impulse for the entire falling process can then be defined as $\max j_n,\ n \in \mathcal{T}$, where $\mathcal{T} = \{\hat{t}^{(i)} | i = 1 \cdots k\}$. We denote an impact moment for the contact i as $\hat{t}^{(i)}$ and the total number of contacts as k. Using this expression, the goal of our problem is to find a sequence of contact locations on

the robot and their timing, such that $\max j_n$ is minimized.

4.3 The Algorithm

Our approach to this problem is inspired by the observation on how humans extend a leg or an arm at the appropriate moment to stop a fall. In this section, we will describe an abstract model to approximate this behavior, use this model to plan the optimal sequence of contacts, and finally execute the plan on the humanoid robots.

4.3.1 Abstract Model

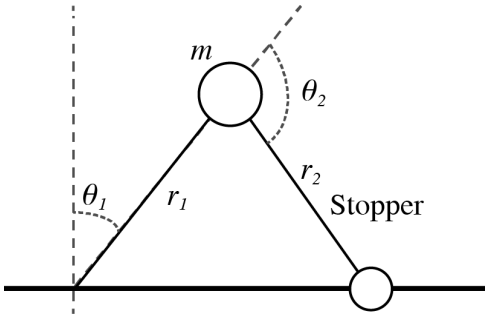


Figure 19: The abstract model consists of a telescopic inverted pendulum and a massless stopper.

The falling motion of biped humanoids has been modelled by a simple 2D inverted pendulum with a massless telescopic rod [30, 27]. The pivot and the center of mass (COM) of the pendulum represent the center of pressure (COP) and the COM of the robot respectively in the sagittal plane. To model the behavior of breaking a fall using contact, we add an additional massless telescopic rod, called a stopper, to the standard telescopic inverted pendulum (Figure 19). The configuration of our abstract model can be defined by the pendulum length (r_1), the angle between the pendulum rod and the vertical line (θ_1), the length of the stopper (r_2), and the angle between the pendulum rod and the stopper rod (θ_2). We assume that the abstract model has control over r_1, θ_2, r_2, but θ_1 is left unactuated. By controlling these three variables, our goal is to minimize the vertical impulse at the contact.

Because the stopper is massless, the equations of motion of the abstract model only depend on θ_1 and r_1 and can be written as:

$$r_1\ddot{\theta}_1 + 2\dot{r}_1\dot{\theta}_1 - g\sin\theta_1 = 0 \tag{11}$$

$$m\ddot{r}_1 - mr_1\dot{\theta}_1^2 + mg\cos\theta_1 = \tau_{r_1}, \tag{12}$$

where m is the mass of the robot and g is the gravitational constant. The control force τ_{r_1} is computed based on the desired velocity of the pendulum rod, $\dot{r}_1^d$:

$$\tau_{r_1} = \frac{m}{h}(\dot{r}_1^d - \dot{r}_1) - mr_1\dot{\theta}_1^2 + mg\cos\theta_1. \tag{13}$$

Though not involved in the equations of motion, the stopper will change the momentum of the abstract model whenever it establishes a contact with the ground. Let the COM of the pendulum and the tip of the stopper be (x_1, y_1) and (x_2, y_2) respectively, with respect to the pivot $(0, 0)$. The momentum due to the vertical impulse j generated by the stopper at the contact can be expressed as:

$$\begin{aligned} P_y^+ &= m\dot{y_1}^+ = m\dot{y_1}^- + j \\ L^+ &= I\dot{\theta_1}^+ = I\dot{\theta_1}^- - (x_2 - x_1)j, \end{aligned} \tag{14}$$

where P and L are linear and angular momentum of the abstract model and I is the estimated inertia. Because we do not know the fullbody pose when planning in the space of the abstract model, we approximate the inertia using the initial configuration of the robot at the beginning of the fall. The superscripts $^-$ and $^+$ denote the quantities before and after the impact respectively. For inelastic collision, the vertical velocity at the tip of the stopper after the impact should be equal to zero, leading to the following equation:

$$\begin{aligned} 0 = \dot{y}_2^+ &= \dot{y}_1^+ - (x_2 - x_1)\dot{\theta}_1^+ \\ &= (\dot{y}_1^- + \frac{j}{m}) - (x_2 - x_1)(\dot{\theta}_1^- - \frac{(x_2 - x_1)j}{I}) \\ &= (\frac{1}{m} + \frac{1}{I}(x_2 - x_1)^2)j + \dot{y}_2^-. \end{aligned} \tag{15}$$

Equation (15) gives a formula to compute the vertical impulse j:

$$j = -\frac{\dot{y}_2^-}{\frac{1}{m} + \frac{1}{I}(x_2 - x_1)^2}. \tag{16}$$

4.3.2 Multiple Contacts

Multiple abstract models can be strung together to model falling with a sequence of contacts. Each abstract model describes the motion from the impact moment $\hat{t}_i$ to the next impact moment $\hat{t}_{i+1}$. The first abstract model is initialized by the initial states of COM and COP of the robot at the beginning of the fall. As the current stopper hits the ground, a new abstract model is initialized: the COM of the current pendulum at $\hat{t}_{i+1}$ becomes the initial COM of the new abstract model and the tip of the current stopper becomes the pivot of the new abstract model. Using multiple abstract models to represent the falling motion, our goal is to search for a sequence of contacts whose maximum vertical impulse is minimized.

Before we formulate the search problem formally, it is important to note that the number of contacts used to break a fall, in theory, can be arbitrarily large. In the limit, if a robot could morph into a rolling ball without slipping, the number of contacts would be infinite and the vertical impulse of the fall would be zero (the initial momentum is never dissipated). In practice, however, the robot has only a limited number of preferred contact points, such as feet, knees, or hands. Furthermore, these contact points can only be applied in certain sequences due to the hardware design and kinematic constraints.

Utilizing these constraints, we introduce a data structure, called a *contact graph*, to narrow down the search space to only those contact sequences achievable by a given robot. A contact graph is a directed graph $G(V_c, E_c)$ which vertices are the preferred contact points on the robot. If there exists an edge from node c_1 to node c_2, it indicates that c_2 is a valid subsequent contact point to c_1. Given a robot, we can design a contact graph to represent all possible falling strategies the robot can

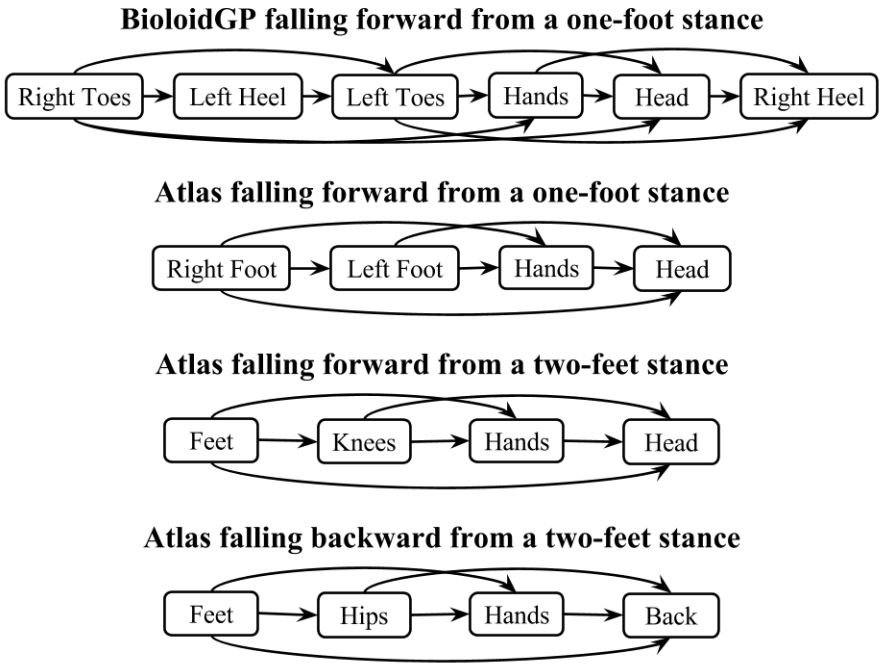


Figure 20: Contact graphs

employ. For example, a path from feet to knees to hands in Figure 20 represents the falling strategy described in [28].

Plan for contact sequence With the contact graph and the initial state of the robot as input, we now describe our algorithm that searches for an optimal sequence of contacts using multiple abstract models.

We formulate the problem as a Markov Decision Process, a framework for modeling decision making with a long-term cost. We define a state at each impact moment as $\mathbf{x} = \{c_1, \hat{t}, \theta_1, r_1, \dot{\theta}_1, \dot{r}_1\} \in \mathcal{X}$, where c_1 denotes the contact point on the robot, $\hat{t}$ denotes the time when the impact occurs, and other parameters describe the position and the velocity of the pendulum at the impact moment. An action $\mathbf{a} = \{c_2, \theta_2, \Delta t, \dot{r}_1^d\} \in \mathcal{A}$ describes the contact point on the robot used as the stopper (c_2), the position of the stopper at the next impact moment (θ_2), the elapse time from the previous impact moment to the next impact moment (Δt), and the desired speed of the pendulum length during the current contact ($\dot{r}_1^d$). Note that the length

of the stopper r_2 at the next impact moment can be derived from r_1, θ_1, and θ_2 by calculating the intersection of the stopper and the ground.

Our goal is to search for the best action sequence in $\mathcal{A}$ to minimize the maximum impulse. The long-term cost of an action $\mathbf{a}$ taken at a state $\mathbf{x}$ can be expressed as

$$\max\big(g(\mathbf{x},\mathbf{a}), v(f(\mathbf{x},\mathbf{a}))\big), \tag{17}$$

where $g(\mathbf{x},\mathbf{a})$ is the local cost function which computes the vertical impulse due to the action $\mathbf{a}$ taken at the state $\mathbf{x}$, $f(\mathbf{x},\mathbf{a})$ is the transition function which outputs the state after taking $\mathbf{a}$ at $\mathbf{x}$, and $v(\mathbf{x})$ is the *cost-to-go* function that yields the minimal impulse starting from $\mathbf{x}$ following the best actions. The cost-to-go function can be expressed recursively as

$$v(\mathbf{x}) = \min_{\mathbf{a}} \max(g(\mathbf{x},\mathbf{a}), v(f(\mathbf{x},\mathbf{a}))). \tag{18}$$

Determining the best action from a given state is a 4D search problem. Every evaluation of an action (Equation (17)) invokes a cost-to-go function (Equation (18)), which recursively generates another 4D search problem. Although the recursive search exponentially expands with the number of contacts, the state space we need to consider is quite limited due to the monotonicity nature of falling motion. That is, as the robot falls, θ_1 changes monotonically from the initial value to 0 (or π). Likewise, $\dot{\theta}_1$ will never exceed the range between the initial velocity and 0. As a result, the algorithm visits a large number of repeated states during the search. We exploit this property using dynamic programming with k-nearest neighbor algorithm to significantly expedite the search (details later).

Algorithm 1 shows the evaluation of the cost-to-go function. For a given state $\mathbf{x}$, we search in the 4D action space with respect to the bounds in each dimension. The range of the desired rod velocity ($\dot{r}_1^d$) is based on the specifications of the robot (Line 6). The elapse time between two consecutive impact moments (Δt) is bounded by the time that takes the pendulum to fall from θ_1 to the ground (Line 9). The actual

Algorithm 1: *cost-to-go*$(\mathbf{x})$

```
if θ̇_1 < 0 then
    return 0;
if kNN has x then
    return kNN[x];
j̄ = ∞;
for ṙ_1^d ∈ V do
    x^now = x;
    Δt = 0;
    while θ_1^now < π/2 do
        S = generate_stoppers(x^now, Δt);
        for c_2, θ_2 ∈ S do
            a = {(c_2, θ_2, Δt, ṙ_1^d)};
            x^+ = f(x^now, a);
            j^+ = g(x^now, a);
            j^* = cost-to-go(x^+);
            j̄ = min(j̄, max(j^+, j^*));
        x^now = simulate_pendulum(x^now, ṙ_1^d);
        Δt = Δt + h;
kNN[x] = j̄;
return j̄;
```

Algorithm 2: generate_stoppers$(\mathbf{x}, \Delta t)$

```
S = ∅;
for c_2 ∈ {c_2 | (c_1 → c_2) ∈ E_c} do
    for θ_2 ∈ [−π, π] do
        r_2 = r_1 cos(θ_1)/ − cos(θ_1 + θ_2);
        if K_{c_1→c_2}[r_1][r_2][θ_2] = 0 then
            continue;
        if |θ_2 − θ̂_2(c_1, c_2)| > (t̂ + Δt)θ̇_max then
            continue;
        S = S ∪ {(c_2, θ_2)};
return S;
```

range of Δt depends on a forward simulation process (Line 17). The candidates of c_2 are defined by the contact graph and the corresponding range of θ_2 for each candidate is determined by the kinematic limits of the robot. c_2 and θ_2 together define a set of feasible stoppers, $\mathcal{S}$, for the next contact (Line 10). Algorithm 2 describes the details on generating the feasible stopper set.

The feasibility of the stopper depends on whether the robot can achieve the kinematic constraints imposed by r_1, r_2, θ_2 for a particular contact transition from c_1 to c_2 (Line 5). Instead of solving an inverse kinematic problem, we expedite the feasibility test by building lookup tables as a preprocess. We first create 10000 distinctive random configurations of the robot within its joint limits. For each connected pair of nodes (c_1, c_2) in the contact graph, we create a 3D lookup table $\mathcal{K}_{c_1 \to c_2}[r_1][r_2][\theta_2]$. If the attributes of an entry (i.e. r_1, r_2, and θ_2) match r_1, r_2, and θ_2 extracted from one of the 10000 robot configurations within tolerance intervals, we mark that entry one, and zero otherwise. With this lookup table, we can efficiently accept or reject a proposed stopper based on the kinematic constraints of the robot.

In addition, we need to make sure that the stopper can reach the proposed θ_2 within $\hat{t} + \Delta t$ second from its initial position, $\hat{\theta}_2(c_1, c_2)$, at the beginning of the fall. For each pair of connected contact points (c_1, c_2) on the contact graph, we precompute the angle $\hat{\theta}_2(c_1, c_2)$, defined as the angle between the vector from c_1 to COM and the vector from COM to c_2, on the initial configuration of the robot. If the proposed θ_2 cannot be achieved by moving at the maximum speed in $\hat{t} + \Delta t$ seconds, we reject the proposed stopper (Line 7).

Whenever we need to evaluate the cost-to-go of a new state, we first check whether there exists a previously visited state sufficiently close to the new state. If so, the cost-to-go of the previously visited state is returned (Line 3). If not, we recursively expand the search to the next contact. The similarity function measures the Euclidean distance between two states weighted by $\mathbf{w} = [100.0, 1.0, 1.0, 0.1, 2.0, 4.0]$ to account

for the different units in different dimensions of the state space.

Algorithm 1 terminates when the velocity of the pendulum is zero or negative (Line 2), indicating that the initial momentum is completely dissipated. After the termination, we do a forward pass to recover the sequence of contacts by following the best actions. For each contact i, we record the state at the impact moment $\mathbf{x}^{(i)}$ and the optimal action $\mathbf{a}^{(i)}$ that leads to the state of the next impact moment $\mathbf{x}^{(i+1)}$. We define the contact plan as $\mathcal{P} = \{(\mathbf{x}^{(1)}, \mathbf{a}^{(1)}) \cdots (\mathbf{x}^{(k)}, \mathbf{a}^{(k)})\}$, where k is the total number of contacts.

Plan for whole-body motion The final step is to execute the contact plan solved by Algorithm 1 on the humanoid robot. Our approach is to solve for a sequence of whole-body configurations, $\mathbf{q}^{(1)}, \cdots, \mathbf{q}^{(k)}$, to match the contact plan. $\mathbf{q}^{(i)}$ is defined as a set of actuated joint angles which will be tracked by the robot during contact i using PID control. For each $(\mathbf{x}^{(i)}, \mathbf{a}^{(i)})$ in $\mathcal{P}$, we formulate an optimization problem as follows:

$$\mathbf{q}^{(i)} = \underset{\mathbf{q}}{\operatorname{argmin}} \Big(\|\mathbf{z}(\mathbf{q}, c_1^{(i)}) - \mathbf{p}_0\|^2 + \|\mathbf{c}(\mathbf{q}) - \mathbf{p}_1(\mathbf{x}^{(i)})\|^2 + \|\mathbf{z}(\mathbf{q}, c_2^{(i)}) - \mathbf{p}_2(\mathbf{x}^{(i)}, \mathbf{a}^{(i)})\|^2 \Big). \quad (19)$$

The first term in the objective function tries to match the current contact position of the robot, $\mathbf{z}(\mathbf{q}, c_1^{(i)})$, to the pivot of the abstract model, $\mathbf{p}_0$. The second term tries to match the COM of the robot, $\mathbf{c}(\mathbf{q})$, to the position of the pendulum, $\mathbf{p}_1(\mathbf{x}^{(i)})$. Finally, the third term tries to match the next contact position of the robot, $\mathbf{z}(\mathbf{q}, c_2^{(i)})$, to the tip of the stopper, $\mathbf{p}_2(\mathbf{x}^{(i)}, \mathbf{a}^{(i)})$. After solving the optimal sequence of poses, the robot is commanded to track the pose $\mathbf{q}^{(i)}$ from the beginning of contact $c_1^{(i)}$ to the beginning of contact $c_2^{(i)}$.

4.4 Experiments

We evaluated our multiple contact planning algorithm on two simulated humanoids, BioloidGP [1] and Atlas [2], as well as the actual hardware of BioloidGP. Our algorithm was compared against a naive approach without planning–the robot simply tracks the initial pose throughout the fall. For the two simulation settings, our evaluation metric is the maximum impulse as previously defined. For the hardware experiments, we measured the maximum acceleration of the head.

4.4.1 Simulation Results

We used an open source physics engine, DART [20, 56] with 0.0005s time step (h) to simulate the motion of the humanoids. Contacts and collisions were handled by an implicit time stepping, velocity-based LCP (linear-complementarity problem) to guarantee non-penetration, directional friction, and approximated Coulomb friction cone conditions.

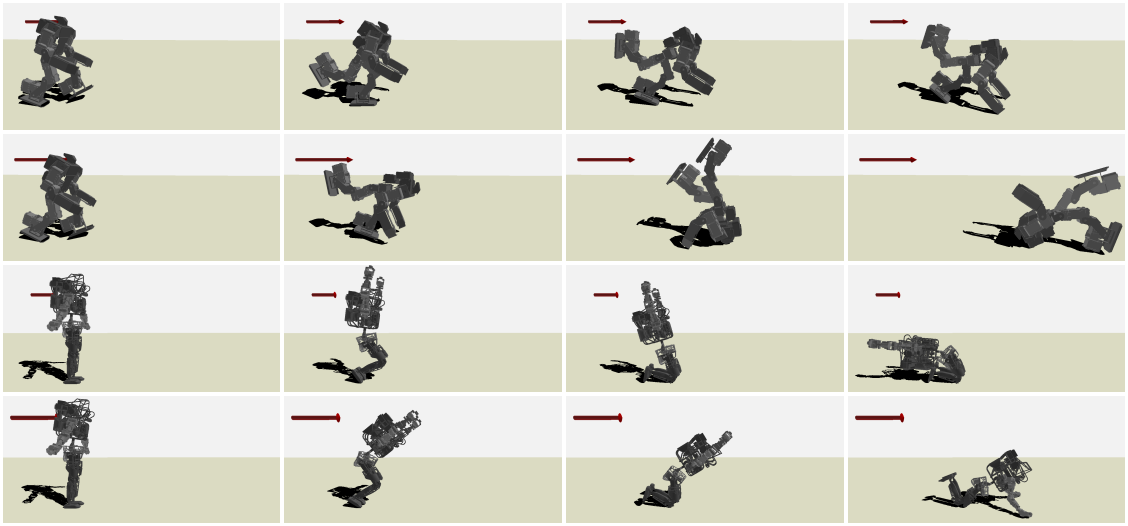


Figure 21: First row: BioloidGP forward falling from a one-foot stance due to a 5.0N push. Second row: BioloidGP forward falling from a one-foot stance due to a 8.0N push. Third row: Atlas forward falling from a two-feet stance due to a 1000N push. Fourth row: Atlas forward falling from a two-feet stance due to a 2000N push.

Table 3: The initial conditions and the results of BioloidGP simulations.

Mag.(N)	Unplanned	Planned	Ratio	Contacts	Remarks
0.5	0.8889	0.2063	23.2%	right toes, left heel, left toes	Stepping
1.5	0.6789	0.2776	40.9%	right toes, left heel, left toes, hands	Tripod
5.0	0.9312	0.3885	41.7%	right toes, left heel, left toes, hands	Tripod
8.0	1.2170	0.5884	48.4%	right toes, left heel, left toes, hands, head, right heel	Rolling

4.4.1.1 BioloidGP

BioloidGP is a small humanoid robot with 16 degrees of freedom (DOFs) (34.6cm, 1.6kg). Our first set of tests applied pushes with different magnitudes to the robot. Starting with the same one-foot stance, we ran four tests with pushes ranging from 0.5N to 8.0N, applied for 0.1 second at beginning of the fall. We set the joint angle limits at $\pm 150^\circ$ and the torque limits at $0.6Nm$. We approximated the speed limit of the COM in the vertical direction and set the limits of the rod length velocity $\dot{r}_1^d$ at $\pm 0.03m/s$. The input contact graph is shown in Figure 20. Due to the relatively large feet of BioloidGP, heels and toes were treated as two separate contacts.

Table 3 describes the details of the initial conditions and the results of each test. The columns of the table denote the magnitude of perturbation, the maximum impulses of the unplanned and the planned motions, the impact ratio of planned to unplanned motion, the optimal contact sequence, and a short description of the emergent falling strategy. As we expected, our planning algorithm used more contacts when the initial momentum was large. For a push with 0.5N, the robot took a single step to recover the fall. For the cases of 1.5N and 5.0N, our algorithm planned a contact sequence with the left heel, the left toe, and both hands, reminiscent to the Tripod strategy proposed by [109]. When we increased the magnitude of the push to 8.0N, the rolling strategy, effective for breaking high speed falls [5], automatically emerged. Please refer to the supplementary video and Figure 21 for all the results.

Comparing to unplanned motions, our algorithm only caused 23.2% to 48.4% of the maximum impulse. To verify how well the contact plan $\mathcal{P}$ was executed, we compared the COM trajectories between the abstract model and the robot (Figure 22).

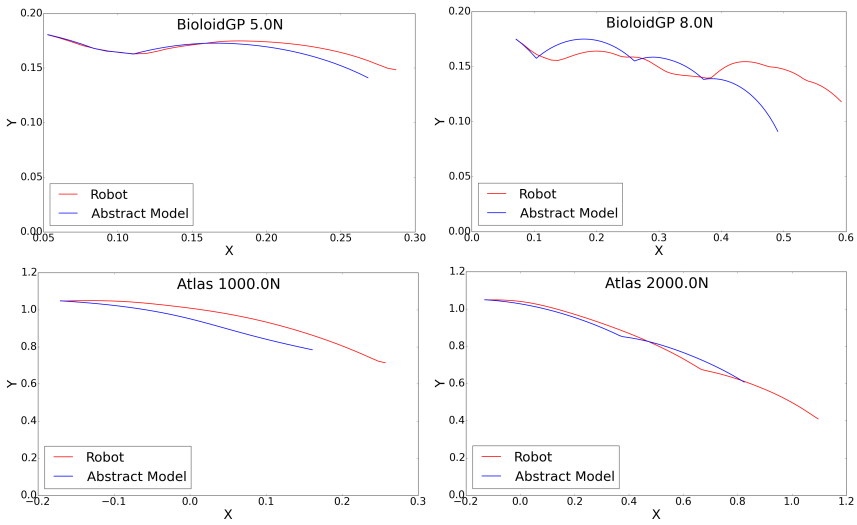


Figure 22: COM trajectories between the abstract model (Blue) and the robot (Red). Top left: BioloidGP forward falling from a one-foot stance due to a 5.0N push. Top right: BioloidGP forward falling from a one-foot stance due to a 8.0N push. Bottom left: Atlas forward falling from a two-feet stance due to a 1000N push. Bottom right: Atlas forward falling from a two-feet stance due to a 2000N push.

Most plans were executed well with an exception of the 8.0N case due to the accumulated errors over a longer motion sequence. Still, in this case the maximum impulse was significantly reduced due to the distribution of impulse over multiple contacts.

The contact graph is an important input that defines all possible contact sequences for the given humanoid. We ran an additional test to modify the contact graph of the BioloidGP robot. By removing the "hands" node, the 8.0N push resulted in a hands-free rolling sequence.

4.4.1.2 Atlas

We also evaluated our algorithm on a large humanoid, Atlas (188cm, 150kg, 28DOFs). We followed the joint limits and the torque limits described in the URDF file provided by Boston Dynamics [2]. The limits of the rod length velocity $\dot{r}_1^d$ were set at $\pm 0.3 m/s$. We ran six test cases with three initial settings: a forward push from a one-foot stance pose, a forward push from a two-feet stance pose, and a backward push from a two-feet stance pose. For each setting, we pushed the robot with two different magnitudes.

Table 4: The initial conditions and the results of Atlas simulations.

Initial Stance	Direction	Mag.(N)	Unplanned(Ns)	Planned(Ns)	Ratio	Contacts
One foot	Forward	1000	363.9	37.8	10.4%	right foot, left foot
One foot	Forward	2500	401.5	281.1	70.0%	right foot, left foot, hands
Two feet	Forward	1000	392.8	214.0	54.5%	feet, knees
Two feet	Forward	2000	322.7	199.7	61.8%	feet, knees, hands
Two feet	Backward	300	338.8	176.5	52.1%	feet, hands
Two feet	Backward	500	344.6	243.9	70.8%	feet, hips, hands, back

The input contact graphs are shown in Figure 20.

Table 4 shows the initial settings and the results for all the tests. Again, our algorithm suggested to use more contacts for pushes with higher magnitudes. For the same setting (falling forward from a one-foot stance pose), we observed a change of strategy from taking a small step ($1000N$) to using Tripod strategy ($2500N$). In the case of falling forward from a two-feet stance pose, the robot landed on its knees when the push was weak ($1000N$), similar to the strategy proposed by [29]. When the external force became stronger ($2000N$), the robot utilized an additional contact with hands, similar to the strategy reported in [26, 74]. For backward falls, the robot was able to stop a gentle nudge ($300N$) using only hands but needed to use three contacts, hips, hands, and back, to stop a stronger push ($500N$), similar to [28]. Please refer to the supplementary video and Figure 21 for all the results.

Comparing to unplanned motions, our algorithm caused 10.4% to 70.8% of the maximum impulse. Because Atlas has relatively short arms, the backward falls presented more challenges than the forward falls. The planned and executed COM trajectories for falling forward from a two-feet stance pose are compared in Figure 22.

4.4.2 Hardware Results

Finally, we ran two experiments on the hardware of BioloidGP (Figure 23). In the first experiment, BioloidGP started with a statically unbalanced position and zero velocity. The optimal plan simply used hands to stop the COM from descending. In the second experiment, the robot was pushed forward by a linear actuator with the magnitude of 0.5N. In this case, BioloidGP used two contacts, knees and hands, to

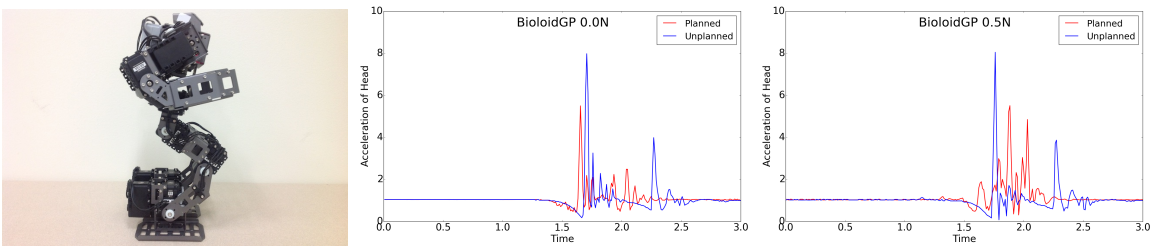


Figure 23: We measured the acceleration at the head of BioloidGP (Left). For both 0.0N (Middle) and 0.5N (Right) cases, the planned motions (Red) yielded about 68% of the maximum acceleration of the unplanned motions (Blue).

stop the fall. We attached an accelerometer to the head of BioloidGP and measured the maximum acceleration during the fall. For both cases, the maximum acceleration resulted from our algorithm was about 68% of that resulted from the unplanned motion (Figure 23).

We ran an additional experiment to show that BioloidGP is capable of deploying the rolling strategy to stop a high-speed fall. We gave BioloidGP a strong shove by hand at the beginning. The large initial momentum resulted in a somersault motion with five contacts. Due to the safety concern, we did not perform the same experiment to produce an unplanned motion for comparison. All the hardware experiments can be viewed in the supplementary video.

4.4.3 Limitations

Our algorithm has a few limitations. First, the planning takes 1.0 to 10.0 seconds to compute in all our experiments. As a result, the algorithm is not ready to deploy in the real-world situations where robots need to react autonomously in real-time. However, our preliminary results show that an optimized contact plan typically can reduce damage for a range of initial conditions, not just for the initial conditions it was optimized for. For example, the optimized contact plan of BioloidGP for 5.0N push yields 35% to 50% of the maximum impulse for pushes ranging from $2.5N$ to 6.5N. The preliminary results imply that it is possible to precompute a set of contact plans which sparsely covers the space of all possible initial conditions. The robot

can choose one plan with the most similar initial condition to the online situation to execute.

The two criteria we use to exclude the infeasible stoppers in Algorithm 2 are tend to be too conservative. In particular, using $\hat{\theta}_2(c_1, c_2)$ from the initial robot configuration to approximate the position of the stopper at each impact moment can be erroneous as the angles between limbs are continuously changing during the fall. Adding other criteria, such as torque limits, to exclude infeasible stoppers might lead to more efficient search.

Our algorithm is limited to planar motion. Falls that require non-planar plans, such as those described in [109] and [31] cannot be effectively stopped by our algorithm. One possible future work is to use a more complex model, such as a reaction mass pendulum with a rigid body inertial mass, proposed by [31].

Finally, we observed that many motions did not end with an balanced, erect stance, because a balanced final pose is not the goal of our planning algorithm. If a balanced final pose is a desired feature, we can simply activate an additional static balance controller after the last contact is executed. Because the momentum at the final contact is near zero, maintaining a static balance is not a difficult task.

4.5 Conclusion

We presented a general algorithm to minimize the damage of humanoid falls by utilizing multiple contact points. For an initial state with arbitrary planar momentum, our algorithm optimizes the contact sequence using abstract models and dynamic programming. Unlike previous methods, our algorithm automatically determines the total number of contacts, the order of contacts, and the position and timing of contacts, such that the initial momentum is dissipated with minimal damage to the robot.

CHAPTER V

ITERATIVE DESIGN OF DYNAMIC CONTROLLERS

Inspired by how humans learn dynamic motor skills through progressive process of coaching and practices, we introduce an intuitive and interactive framework for developing dynamic controllers. The user only needs to provide a primitive initial controller and high-level, human-readable instructions as if s/he is coaching a human trainee, while the character has the ability to interpret the abstract instructions, accumulate the knowledge from the coach, and improve its skill iteratively. We introduce "control rigs" as an intermediate layer of control module to facilitate the mapping between high-level instructions and low-level control variables. Control rigs also utilize the human coach's knowledge to reduce the search space for control optimization. In addition, we develop a new sampling-based optimization method, Covariance Matrix Adaptation with Classification (CMA-C), to efficiently compute control rig parameters. Based on the observation of human ability to "learn from failure", CMA-C utilizes the failed simulation trials to approximate an infeasible region in the space of control rig parameters, resulting a faster convergence for the CMA optimization. We demonstrate the design process of complex dynamic controllers using our framework, including precision jumps, turnaround jumps, monkey vaults, drop-and-rolls, and wall-backflips.

5.1 Motivation

Mastering a dynamic motor skill, such as a handstand in gymnastics, or a precision jump in Parkour, usually requires an iterative process with interactive coaching and repetitive practices. Based on the current skill level of the trainee, the coach gives

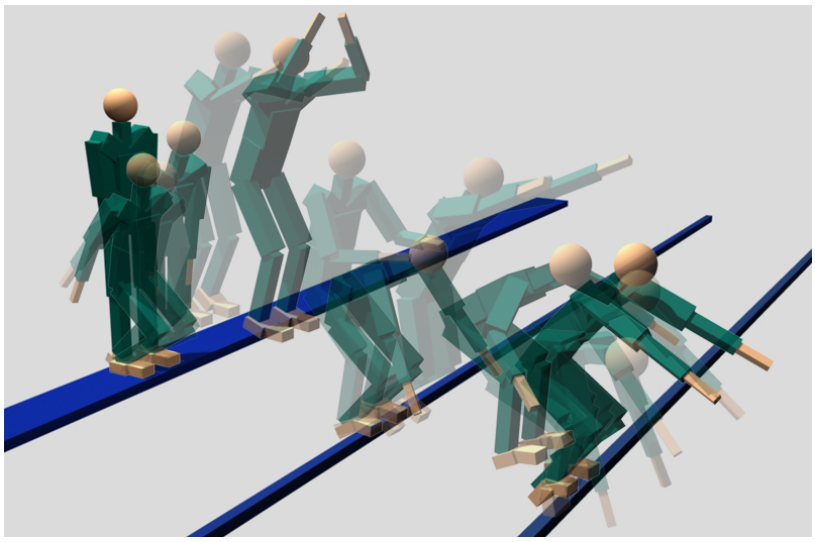


Figure 24: Two precision jumps on narrow rails.

instructions that emphasize the key areas for improvement. The trainee then internalizes the new information and improves the skill through practices. The learning process alternates between coaching and practicing stages until the skill is acquired. In contrast, teaching a physically simulated character a new motor skill entirely depends on the effort of the controller developer, from the design of the control architecture to the tweaking of low-level control parameters. We hypothesize that the controller development can be greatly simplified by exploiting the same learning principles humans use to acquire new motor skills. In addition, if designing new controller can be done in the similar fashion as coaching a human trainee, the existing controllers can be easily adapted, extended, or concatenated for new situations.

This paper attempts to formalize the methodologies humans use to learn dynamic motor skills. We present an algorithmic framework to facilitate the iterative learning process of coaching and practicing. During the coaching stage, the user only needs to provide a primitive initial controller and high-level, human-readable instructions as if s/he is coaching a human trainee. For example, a human coach typically uses high-level instructions, such as "extend the legs" or "push the ground", rather than specific descriptions of joint angles. During the practicing stage, the character is capable of

following the instructions and improving its motor skill effectively on its own. That is, the character has the autonomous ability to interpret the abstract instructions, accumulate the knowledge from the coach, and optimizes its motion based on the guidance.

The main challenge of this work is to formalize these elusive principles of human learning into mathematical models for controller design. Our underlying assumption is that any motor skill can be achieved by using simple proportion-derivative (PD) style control and Jacobian transpose control at every actuated joint and every body part, *if* the control parameters are properly determined. This formulation, however, introduces a prohibitively large space of control variables for the existing optimization methods. Our controller design framework solves this issue by utilizing human coaching knowledge to select an appropriate subset of control variables (coaching stage) and relying on a new sampling-based optimization method to determine the value of control variables (practicing stage).

Using high-level, human-readable instructions can potentially simplify the controller design, but directly mapping high-level instructions to low-level control variables is a challenging task. We introduce an intermediate layer of control module, called control rigs, to interpret the human instructions during the coaching stage. A control rig simultaneously controls a set of low-level control variables in a coordinated fashion. For example, a control rigs flexes legs uses a single parameter, the distance between the waist and the feet, to control the target joint angles of the PD controllers on the hips, knees, and ankles. With this intermediate layer, mapping human instructions to the low-level control variables can be done automatically by selecting appropriate control rigs. In addition, using control rigs instead of low-level variables reduces the search space for the optimization. We design a set of control rigs from frequently used instructions for Parkour training. These control rigs are general and can be reused for coaching different sports.

To determine the control variables efficiently during the practicing stage, we introduce the concept of "learning from failure" using a sampling-based optimization method. Our key insight is that the failed samples contain as much useful information as the successful ones. For example, falling on the ground or hitting obstacles are valuable experiences to learn vaulting. Instead of throwing away those failed simulation trials, our algorithm uses them to approximate the boundary of the feasible region in the control variable space. Having an approximated feasible region accelerates the optimizations by preventing the character to repeat failures committed before. Based on this idea, we build Support Vector Machines in concert with Covariance Matrix Adaptation (CMA), called Covariance Matrix Adaptation with Classification (CMA-C). The main advantage of CMA-C is that it exploits every simulated trial; the successful ones are used to contract the covariance matrix while the failed ones are used to refine the boundary of feasible region.

We demonstrate the design process of complex dynamic controllers using our framework, including precision jumps, turnaround jumps, monkey vaults, drop-and-rolls, and wall backflips. We show that the character started out with basic controllers; using PD control to track a few roughly specified poses; and were able to learn these complex dynamic skills within minutes with only a few high-level instructions from the user. Once a controller is developed, parameterizing it to a family of similar controllers for concatenation can be done without additional effort from the user.

5.2 Overview

We introduce a general framework to design dynamic controllers using high-level, human-readable instructions (Figure 25). The iterative process begins with an input controller. We view the initial controller as a blackbox because our algorithm does not interact with its internal implementation. For all our experiments, we used a simple pose-tracking controller with 4 to 6 keyframes. During training, the initial

controller is improved iteratively through alternating stages of *coaching* and *practicing*. The output is a new controller that meets the requirements of the user. Once a controller is developed, we can generalize it by parameterization or concatenation for new situations.

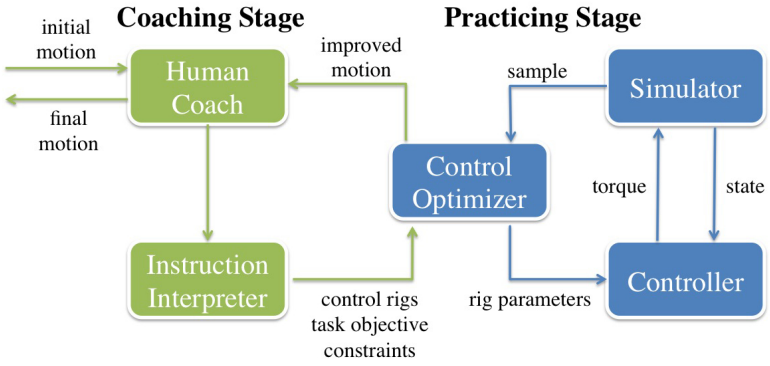


Figure 25: Overview diagram.

During each coaching stage, the user provides high-level instructions to correct undesired behaviors, change task objectives, or add different styles to the motion. According to the type of the instructions, the *instruction interpreter* automatically selects the appropriate control rigs, and modifies constraints or the objective function for the optimization.

During each practicing stage, the *control optimizer* searches for control rig parameters using our new algorithm, Covariance Matrix Adaptation with Classification (CMA-C). The control optimizer uses the current estimate of rig parameters to generate motion samples. A controller can be represented as a function g, which takes the current state $\mathbf{q}_t$ as input and generates torque τ_t. In addition, we denote s as a function that simulates the motion from a given state under a given controller, and outputs the final state $\mathbf{q}_f$ of the simulation.

$$\mathbf{q}_f = s(\mathbf{q}_t, g) \tag{20}$$

5.3 Coaching Stage

The coaching stage takes in high-level user instructions and interprets them by revising the task objective or adding constraints. We introduce "control rigs", as an intermediate layer between high-level instructions and low-level control variables. A control rig, predefined by our framework, is a function of a set of low-level control variables. It allows for more coordinated control of the low-level variables and provides more intuitive mapping to high-level instructions. The main advantage of using control rigs is that it reduces the optimization time significantly by suggesting the most effective directions to optimize. Although a control rig needs to be manually defined, it can be shared by different instructions or repurposed for new motor tasks.

5.3.1 Instructions

Although coaching strategies and styles vary widely, the basic instructions commonly used to break down a complex movement are surprisingly consistent. Most instructions provide a numerical or categorical correction to improve a particular part of the body. For example, "Lower the center of mass more" or "raise your arms to 45 degrees". From observing Parkour training sessions and tutorials, we define a set of coaching instructions in Table 6.

5.3.2 Control Rigs

A control rig r is a pre-defined function of a set of low-level PD or Jacobian Transpose controllers. A PD controller tracks the target joint angle based on the feedback rule: $\tau = k_s(\hat{\theta} - \theta) - k_d(\dot{\theta})$, where k_s and k_d are the gain and the damping coefficient of the joint and $\hat{\theta}$ is the desired value for the joint angle. A Jacobian Transpose controller computes the required joint torques to produce the desired "virtual force", $\mathbf{f}_v$, at a point $\mathbf{x}$, using the Jacobian Transpose mapping: $\tau = \mathbf{J}^T(\mathbf{x})\mathbf{f}_v$, where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix at $\mathbf{x}$. These two types of low level controllers, when combined properly, can effectively control the pose and global motion of the character. A control

rig, $\tau = r(\mathbf{q}_t, \mathbf{p})$, takes in the current state $\mathbf{q}_t$ and the rig parameters $\mathbf{p}$ to produce torques which are added to the total torques applied to the character.

We keep a set of active control rigs, $\mathcal{R} = \{r_1, ..., r_m\}$, during training. As the user provides more instructions, more control rigs are included to the active set. With optimized parameters $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_m\}$, we obtain an improved controller g_R.

$$g_{\mathcal{R}}(\mathbf{q}, \mathcal{P}) = g(\mathbf{q}) + \sum_{i=1}^{m} r_i(\mathbf{q}, \mathbf{p}_i) \tag{21}$$

In this paper, we designed five control rigs from common instructions used for learning gymnastics and Parkour. Each control rig, r_i, has a set of arguments determined by the instruction from which r_i is created, and a set of rig parameters which are optimized during the practicing stage (Table 5).

1. TargetJoints rig consists of the PD controllers on a set of joints specified by the instruction. The rig parameters are defined as the target joint angles of the PD controllers.

2. IKPose rig solves for the joint angles to meet a desired relative position of two bodies specified by the instruction. The solved joint angles are then used as the target angles for a set of PD controllers.

3. Stiffness rig adjusts the gains of the PD controllers on a set of joints specified by the instruction. The rig parameters are defined as the gains.

4. VirtualForce rig consists of a Jacobian Transpose controller which computes required torques to produce a virtual force at the center of mass of a body link specified by the instruction. The rig parameter is defined as the desired virtual force

5. FeedbackVirtualForce rig consists of a Jacobian Transpose controller on a body link specified by the instruction. Instead of treating the desired virtual force as

Table 5: Control rigs.

Rig Type <Arguments>	**Description**	**Rig Parameters**
TargetJoints <joint a, $\cdots$ >	Command a set of joints to achieve desired angles simultaneously.	Desired joint angles
IKPose <body a, body b>	Compute a target pose to meet the desired relative position between body a and body b. Command joints to achieve the target pose.	Desired distance or desired angles
Stiffness < joint a, $\cdots$ >	Command a set of joints with desired stiffness.	Gains
VirtualForce <body a>	Apply torques which produce the desired virtual force at the center of mass of body a.	[a] Desired virtual force in end-effector frame
FeedbackVirtualForce <body a, $\hat{\mathbf{C}}$ >	[b] Apply torques which produce the virtual force at the cetner of mass of body a. The virtual force is computed by a feedback rule.	-

[a] f_u is in the direction of C - S. f_v is orthogonal to f_u and parallel to the contacting surface.

[b] We use the following feedback rule on the center of mass of the whole body: $f_v = k_s(\hat{\mathbf{C}}-\mathbf{C})-k_d\dot{\mathbf{C}}$, where $k_s = 300$ and $k_d = 6$.

a rig parameter, it is computed using the feedback rule: $f_v = k_s(\hat{\mathbf{C}} - \mathbf{C}) - k_d\dot{\mathbf{C}}$, where $\mathbf{C}$ and $\hat{\mathbf{C}}$ are the center of mass of the whole body and its desired position. Intuitively, this rig computes the torques that generate a virtual force to bring the center of mass closer to the desired position.

We show that these five control rigs are sufficient to generate the motor skills demonstrated in the result section.

5.3.3 Instruction Interpreter

The instruction interpreter translates a human-readable instruction into two possible actions: modifying the task objective or adding constraints for the optimization problem. In addition, if these actions involve new control rigs, the instruction interpreter will add them to the optimization domain. We define a set of simple rules that map the instructions to the control rigs. Please see details in Table 6.

The task objective. The task objective is evaluated at the final state of the motion by an objective function .

$$f(\mathcal{P}) = \sum_{i=1}^{n} \|h_i(s(\mathbf{q}_0, g_{\mathcal{R}}) - \hat{h}_i\| \tag{22}$$

When the user specifies an instruction from Table 6 (except for the last instruction,

Table 6: Interpretation of instructions. Each instruction is associated with a control rig, an objective term and/or a contraint. $\mathbf{q}_f$, $\mathbf{q}_f^{prev}$: the final state of the current motion and the previous motion. $\mathbf{C}, \mathbf{S}, \mathbf{P}, \mathbf{L}$: the center of mass, center of pressure, linear momentum, and angular momentum. $\mathbf{pos}_{limb}$, $\mathbf{rot}_{limb}$: the limb position and orientation. q_{joint}, ks_{joint} : the joint angle and stiffness.

Instruction	**Introduced control rig**	**Introduced objective / constraint**
FLEX|EXTEND joint BY θ	TargetJoints<joint>	$\|q_{joint}(\mathbf{q}_f) - \theta\|$
MOVE [a] direction BY δ	IKPose< [b] contact, root>	[c] $\|\mathbf{C}(\mathbf{q}_f) \cdot \mathbf{d} - (\mathbf{C}(\mathbf{q}_f^{prev}) \cdot \mathbf{d} + \delta)\|$
TRANSLATE limb direction BY δ	IKPose<root, limb>	$\|\mathbf{pos}_{limb}(\mathbf{q}_f) \cdot \mathbf{d} - (\mathbf{pos}_{limb}(\mathbf{q}_f^{prev}) \cdot \mathbf{d} + \delta)\|$
ROTATE limb direction BY δ	IKPose<root, limb>	$\|\mathbf{rot}_{limb}(\mathbf{q}_f) \cdot \mathbf{d} - (\mathbf{rot}_{limb}(\mathbf{q}_f^{prev}) \cdot \mathbf{d} + \delta)\|$
SPEED NEAR $\hat{\dot{\mathbf{C}}}$	VirtualForce<contact>	$\|\dot{\mathbf{C}}(\mathbf{q}_f) - \hat{\dot{\mathbf{C}}}\|$
SPEEDUP/ SLOWDOWN γ %	VirtualForce<contact>	$\|\mathbf{P}(\mathbf{q}_f) - (\mathbf{P}(\mathbf{q}_f^{prev}) * \gamma)\|$
TURNFASTER/ TURNSLOWER γ %	VirtualForce<contact>	$\|\mathbf{L}(\mathbf{q}_f) - (\mathbf{L}(\mathbf{q}_f^{prev}) * \gamma)\|$
BALANCE	FeedbackVirtualForce< contact, $\hat{\dot{\mathbf{C}}}_{bal}$ >	$\|\dot{\mathbf{C}}(\mathbf{q}_f)\|$
RELAX|STIFFEN joint BY γ %	Stiffness<joint>	$\|ks_{joint}(\mathbf{q}_f) - (ks_{joint}(\mathbf{q}_f^{prev}) \cdot \gamma)\|$
PUSH/PULL limb AGAINST surface	VirtualForce<limb>	-
PLACE body|C|S NEAR body|C|S|constant	-	$\|\mathbf{pos}_{body\|C\|S}(\mathbf{q}_f) - \mathbf{pos}_{body\|C\|S\|constant}(\mathbf{q}_f)\|$
[d] body|C|S IN RELATION TO body|C|S|constant	-	relation(body|C|S, body|C|S|constant)

[a] We used a dictionary to translate the dir keyward into direction vector d. For instance, if the dir keyword is "upward", $d = (0, 1, 0)^T$.

[b] "contact" indicates the body parts in contact

[c] We project $\mathbf{C}(\mathbf{q})$ in the desired direction, $\mathbf{d}$, specified in the instruction.

[d] The last instruction creates a constraint, instead of an objective function.

which will be explained in the next paragraph), a new term, $\|h_i(\mathbf{q}_f) - \hat{h}_i\|$, will be added to the objective function. h_i is a function that evaluates a quantity derived from the final state. $\hat{h}_i$ is the desired value for that quantity. For example, to interpret the instruction "TRANSLATE hands forward BY $0.5m$" (Figure 26), the keyword "forward" is first mapped to the predefined direction $\mathbf{d} = [1, 0, 0]^T$. Then $h_i(\mathbf{q}_f)$ is defined as the average position of the hands in the forward direction ($\mathbf{pos}_{hands}(\mathbf{q}_f) \cdot \mathbf{d}$). Finally, the desired value $\hat{h}_i$ is computed by adding $0.5m$ to the previous average position of the hands in the forward direction ($\mathbf{pos}_{hands}(\mathbf{q}_f^{prev}) \cdot \mathbf{d} + 0.5$).

Although we use the final state in Equation 22, in our implementation the entire motion sequence is available for evaluation. Thus the cost function can depend on any state in the motion sequence. In addition, we can formulate a cost function that

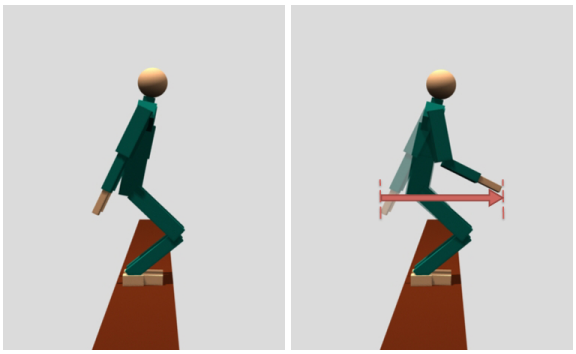


Figure 26: The user can adjust the positions of hands by giving an instruction "TRANSLATE hands forward BY $0.5m$". The instruction will add an IKPose rig for arms and modify the desired position of hands by $0.5m$ in the forward direction.

affects the timing of the motion by evaluating the elapsed time for each phase.

Constraints. Most dynamic motor skills are subject to constraints, such as maintaining balance or contact conditions. When the character's motion fails to meet any of the constraints, the simulation will terminate immediately. The failed motion adds a penalty term, $K(T_{max} - t)$, to the objective function (Equation 22) to penalize early failure. t indicates the time when failure occurs, and K and T_{max} are set to 1000 and 2 respectively. T_{max} can be adjusted based on the expected duration of the successful motion.

In our instruction set, the last instruction (Table 6) introduces a constraint to enforce spatial relationship between two body parts. For example, "head IN FRONT OF root" instruction places a lower bound on the x position of head $(\mathbf{d} \cdot (pos_{head}(\mathbf{q}_f) - pos_{root}(\mathbf{q}_f)) > 0$ where $\mathbf{d} = (1, 0, 0)^T)$.

CHAPTER VI

OPTIMIZATION WITH FAILURE LEARNING

6.1 Practicing Stage

The practicing stage optimizes the parameters of each control rig selected by the coaching stage. Although the search space is largely reduced by using control rigs rather than low-level control variables, we still need to solve a non-convex and discontinuous optimization problem. Much previous work in computer animation applied Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to problem in this nature. The standard procedure at each iteration involves generating samples in the control variable space, using the samples to simulate motions, and evaluating each motion according to the cost function.

However, the standard CMA-ES does not exploit two distintive features of our problem. First, our problem has a clear definition of infeasible samples, such as the control parameters that result in an imbalanced motion. Second, because our framework is an iterative process, we have a series of optimization problems sharing very similar feasible regions. Without leveraging these features, the standard CMA-ES spends much computation time on repeatedly evaluating failed samples.

We designed a new algorithm, called CMA-C, based on the observation of human's ability to *learn from failure*. Because failure in the real world is usually associated with pain or injury, humans tend to be very effective in characterizing the cause of failure and trying to avoid the same mistakes in the future. To this end, CMA-C simultaneously builds a set of Support Vector Machines (SVMs) along with the evolution of CMA. Each SVM approximates the infeasible region of a particular type of failure. CMA-C accelerates the optimization by preventing redundant evaluations

of failed samples. Moreover, the constructed SVMs can be reused by subsequent optimizations because they share similar feasible regions, further speeding up the computation significantly.

Table 7: CMA-C evaluation on five problems. CMA-C improves the computation by four to five times. μ, λ, and σ represents CMA parent size, population size, and step size. $\hat{\mathbf{C}}$, $\hat{\mathbf{P}}$, $\hat{\mathbf{L}}$, and $\hat{\mathbf{S}}$ indicate the desired COM, linear momentum, angular momentum, and the COP.

Problem	Objective function	Constraints ($c_1 \vert c_2 \vert$ ...)	μ	λ	σ	Domain	Feasible region	Noise	CMA-ES (evals/total time)	CMA-C (evals/total time)
Toy1	$f(x,y) = 0.1((x-3.9)^2+(y-3.9)^2)$	$(x>4 \text{ or } y>4) \vert x+y<7.5$	4	8	5	$[-5,5]^2$	Convex	Asymmetric Random	663.2 / 8ms	144.8 / 70ms
Toy2	$f(x,y) = 0.1((x-3.9)^2+(y-3.9)^2)$	$(x>4 \text{ or } y>4) \vert x+y<7.5$	4	8	5	$[-5,5]^2$	Convex	Symmetric Undulation	1026.4 / 9ms	229.6 / 116ms
Toy3	$f(x,y) = 0.1((x-3.1)^2+(y-3.6)^2)$	$(x>4 \text{ or } y>4) \vert (x<3 \text{ or } y<3 \text{ or } (x<3.5 \text{ and } y<3.5))$	4	8	5	$[-5,5]^2$	Non convex	Symmetric Undulation	1027.2 / 10ms	205.2 / 107ms
Leaning	$\Vert\mathbf{C}(\mathbf{q}_f) - \hat{\mathbf{C}}\Vert^2$	Fall forward | Fall backward | Time-out	5	10	1	$[-1,1]^4$	-	-	126.25 / 148s	23.75 / 32s
Thrusting	$\Vert\mathbf{P}(\mathbf{q}_f) - \hat{\mathbf{P}}\Vert^2 + \Vert\mathbf{L}(\mathbf{q}_f) - \hat{\mathbf{L}}\Vert^2$	Fall forward | Fall backward | Negative angular momentum	5	10	1	$[-1,1]^3$	-	-	101.25 / 15s	77.5 / 12s
Landing	$\Vert\mathbf{C}(\mathbf{q}_f) - \hat{\mathbf{C}}\Vert^2 + \Vert\mathbf{S}(\mathbf{q}_f) - \hat{\mathbf{S}}\Vert^2$	Fall forward | Fall backward | Invalid contact	5	10	1	$[-1,1]^4$	-	-	105.0 / 210s	22.5 / 44s

6.1.1 CMA-C

CMA-C is designed for solving a general optimization problem with multiple constraints.

$$\begin{aligned} \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x}) \\ &\text{subject to } c_i(\mathbf{x}) = 0, \text{ where } i = 1 \cdots n \end{aligned} \tag{23}$$

In our formulation, the cost function $f(\mathbf{x})$ evaluates the performance of the simulated motion generated by a given set of control rig parameters (i.e. $\mathbf{x}$ refers to $\mathcal{P}$). Each constraint $c_i(\mathbf{x})$ represents a type of failure when the motion cannot satisfy it. $c_i(\mathbf{x})$ can be in the form of inequality constraint, but we only show equality constraints for clarity. CMA-C can be applied to any problem with the form of Equation (23), but it is particularly effective when evaluating $f(\mathbf{x})$ is costly or when the problem is highly constrained.

Our main idea is to construct classifiers using SVMs while running CMA. For each constraint, we build a SVM to represent its feasible region. Every time a sample is randomly drawn, we first use SVMs to predict whether this sample will satisfy all constraints. If so, we evaluate it using $f(\mathbf{x})$. Otherwise, the sample is discarded without evaluation. In the context of our problem, we use SVMs to predict whether a set of control variables yields successful motion before we simulate it. If a sample is predicted to satisfy all the constraints, we evaluate its cost and assign a label for each SVM: $+1$ if the sample indeed satisfies the corresponding constraint and -1 otherwise. At the end of each CMA iteration, we refine the boundary of each SVM using all the samples. Algorithm 3 and 4 outline the procedure of CMA-C.

When the SVM makes a correct prediction, it speeds up the convergence of CMA. When the SVM makes an incorrect prediction (i.e. generates a sample with -1 label), our algorithm still utilizes the negative sample to improve the boundary of the feasible region. However, there are two issues with this algorithm when applied in practice.

Algorithm 3: CMA-C

```
Initialize m, C, σ;
Initialize SVM_{1..n};
while not terminate do
    for i = 1 → λ do
        x_i = selectSample(m, C, σ, SVM_{1..n});
        (f_i, e_{i,1}, ..., e_{i,n}) = fitness(x_i);
    sort(x_{1..λ});
    (m, C, σ) = updateCMA(x_{1..λ}, f_{1..λ}, m, C, σ);
    for i = 1 → n do
        if enough samples for SVM_i and γ_i is null then
            γ_i = 1/2(kσ)^2
        updateSVM(SVM_i, γ_i, x_{1..λ}, e_{1..λ,i});
```

Algorithm 4: selectSample()

```
Data: m, C, σ, SVM_{1..n}
Result: selected sample x
while not reach maximum trials do
    x = gaussianSelection(m, σ^2 C);
    reject = False;
    for i = 1 → n do
        if SVM_i.activated() and SVM_i.predict(x) < 0 then
            reject = True;
            break;
    if not reject then
        return x
return gaussianSelection(m, σ^2 C);
```

First, a SVM requires a sufficient number of positive and negative samples before it can be activated. This requirement imposes a long "warm-up" time if the initial CMA distribution has low likelihood to generate feasible samples. Second, although SVMs can use kernels to represent non-linear boundaries, tweaking kernel parameters in SVMs can greatly affect the results of classification.

The first issue is exasperated by problems with a large number of constraints and a relatively small feasible region, such as the problem of developing parkour controllers. To reduce the warm-up time, we represent each constraint with a SVM individually, instead of using one aggregate SVM to represent the intersection of all constraints. During the optimization, we superimpose all the activated SVMs and approximate the feasible region by taking the intersection of all positive regions. Building multiple SVMs significantly reduces the warm-up time because a positive sample for an aggregate SVM must satisfy all constraints, where as a positive sample for each separate SVM only needs to satisfy one constraint. Collecting enough positive samples to activate an aggregate SVM clearly takes much longer time than activating each individual SVM (Figure 27). In our toy problems, the first SVM is constructed after 17.2 evaluation on average, an aggregate SVM requires 101.7 evaluations.

In addition, we propose a stochastic sampling scheme to address the general issue due to insufficient samples at the beginning of the optimization. This scheme accepts a sample with the probability of a sigmoid function $\frac{1}{1+e^{-\alpha x}}$, where x is the signed distance from the sample to the boundary. With a smaller α, this stochastic scheme is more conservative about accepting and rejecting samples. As more samples are available and the SVM boundary becomes more accurate, we can increase α to have a harder rule. In our experiments, we use $\alpha = 4$ throughout the entire optimization.

The second issue regards the tuning of the kernel parameters. We chose a frequently used kernel, Gaussian radial basis function (RBF), in our implementation.

$$k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{24}$$

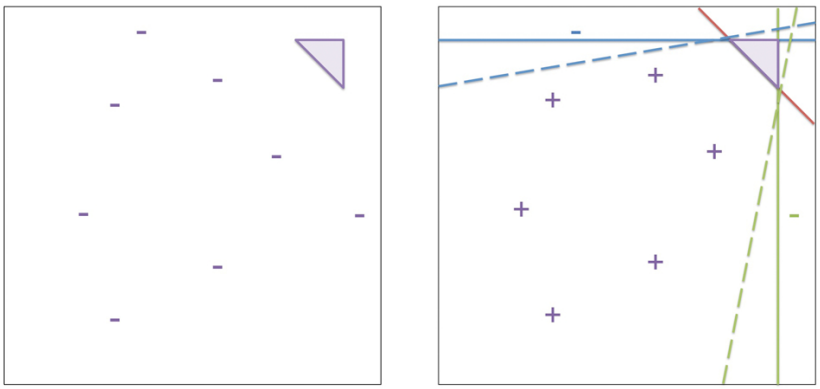


Figure 27: A comparison of a single SVM and multiple SVMs. Left: Using a single SVM to represent the feasible region (purple triangle), the SVM cannot be activated after eight samples, due to the lack of positive samples. Right: If the feasible regions is represented by the intersection of three constraints, each of which is approximated by an individual SVM, eight samples are sufficient to active two SVMs (shown in green and blue). Dashed lines indicate the current SVM approximation of constraints.

Intuitively, a RBF adds a "bump" around each sample. The bump with the ideal size, determined by γ, should overlap with a few neighboring samples. If γ is too large, the trained SVMs will over-fit the data. If it is too small, the classification will have very low accuracy. Consequently, tuning γ requires the knowledge of the current sample distribution.

Fortunately, we know the exact sample distribution because all the samples are drawn from the current estimate of Gaussian distribution, $\mathbf{x} \sim N(\mathbf{m}, \sigma^2\mathbf{C})$, provided by CMA. With this information, we set γ to cover only portion of the current CMA gaussian distribution, using a simple rule:

$$\gamma = 1/2(k\sigma)^2 \tag{25}$$

where k determines the proportion of the radial basis function with respect to the sample distribution. We use $k = 0.1$ for all the experiments.

6.1.2 Analysis on Toy Problems

In addition to the real problems on motion control, we designed three 2D problems to evaluate and visualize the results of CMA-C. We found that CMA-C outperforms CMA-ES significantly when the problem has a large portion of infeasible region or when the infeasible region is highly nonlinear or discontinuous. In other words, if the optimizer is more likely to be "trapped" in the infeasible region, CMA-C has a better chance to "escape" and converge at a feasible and optimal solution.

The first two toy problems have convex feasible regions, while the third one has a non-convex feasible region (Figure 28, Table 7). To mimic the nonlinearity and discontinuity in the objective function of the real problems, we added random noise to the objective function of the toy problems. For the first toy problem, we add different levels of random noise in feasible and infeasible regions as follows:

$$f_{toy}(x,y) = \begin{cases} f(x,y) + k_{toy}^{f} \cdot (rand(0,1)) & \text{if } \forall i, c_i = 0 \\ f(x,y) + k_{toy}^{inf} \cdot (rand(0,1)) & \text{otherwise} \end{cases} \tag{26}$$

where $k_{toy}^{f} = 0.05$ and $k_{toy}^{inf} = 2.0$. In the second and third problems, we add same sinusoidal noise in both feasible and infeasible regions:

$$\begin{aligned} f_{toy}(x,y) = f(x,y) \\ + k_{toy} \cdot |sin(\omega_{toy} \cdot (x+y)) + sin(\omega_{toy} \cdot (x-y))| \end{aligned} \tag{27}$$

where the magnitude and frequency of the sinusoidal function $k_{toy} = 2.0$ and $\omega_{toy} = 10.0$.

We compared the performance of standard CMA-ES and CMA-C on each problem by evaluating 20 times with different initial seeds. For a problem with costly sample evaluation routines, such as the controller design problem focused in this paper, CMA-C significantly reduces the total computation time. However, if the sample evaluation time is negligible, such as our toy examples, reducing the number of evaluations does

Table 8: CMA-C on the toy problem I (Table 7) with various ratios of the infeasible area to the feasible area. All other conditions are the same.

Area Ratio	CMA-ES (evals)	CMA-C (evals)	Performance gain
50	237.2	153.2	155%
200	663.2	144.8	458%
800	1343.6	200.0	672%

Table 9: CMA-C on the toy problem II (Table 7) with various magnitude and frequency of noise. All other conditions are the same.

k_{toy}	ω_{toy}	CMA-ES (evals)	CMA-C (evals)	Performance gain
1.0	10.0	638.8	195.2	327%
2.0	10.0	1026.4	229.6	447%
4.0	10.0	1820.4	309.2	589%
1.0	20.0	735.2	220.0	334%
2.0	20.0	1079.6	272.0	397%
4.0	20.0	1574.8	348.0	452%

not speed up the total computation time. In fact, CMA-C can be slower than CMA-ES due to the additional computation on constructing SVMs.

To further evaluate the performance of CMA-C, we conducted two sets of analyses by varying the ratio of the infeasible area to the feasible area (problem I) and the level of noise (problem II). Both analyses were designed to add difficulty in finding feasible solutions to the optimization problem. Our results, shown in Table 8 and Table 9, indicate that the performance of CMA-C increases when either the magnitude of the noise in the infeasible region or the infeasible area increases, while the frequency of noise does not have a significant correlation to the performance gain.

In addition, we validated the approximated feasible regions by visualizing the difference between constructed SVM and the ground truth (Figure 28). The results show that the non-convex SVM boundary in the second problems match the ground truth well. The only difference resides in the upper left corner of the L-shape feasible region in the second problem.

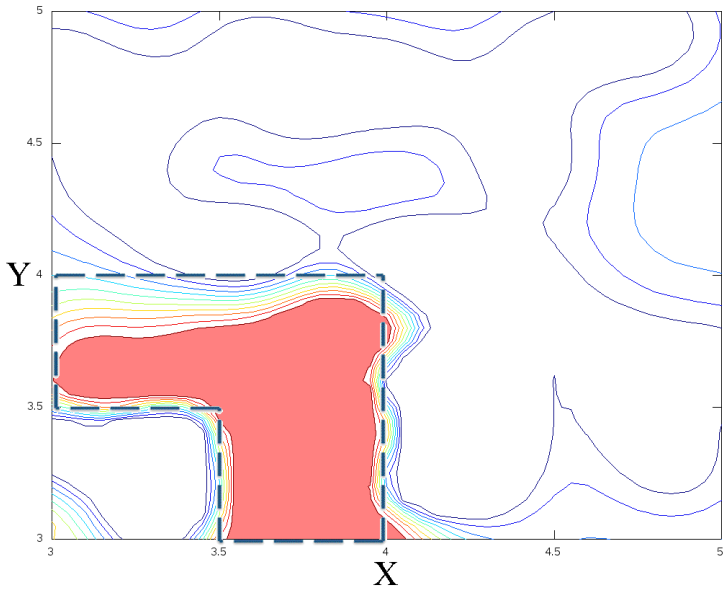


Figure 28: Contour of the trained SVMs for the second toy problem (Table 7). The feasible region classified by SVMs is filled with red. The ground truth feasible region is outlined by the dashed lines. For clarity, the figure is zoomed into the region from $[-5, 5]^2$ to $[3, 5]^2$.

6.1.3 Analysis on Real Problems

We also evaluated the performance of CMA-C on the problems of controller design (Table 7). On average, CMA-C performed five times faster than CMA-ES to achieve the same optimal value. In some cases, CMA-C reached desired objective value, while CMA-ES got stuck in the local minima. The advantage of CMA-C is even more prominent when solving a series of optimizations with gradually introduced constraints, because CMA-C can simply overlap the new SVM with previously constructed ones. However, when the infeasible region is relatively small, the difference in performance between CMA-C and CMA-ES is not obvious. For example, thrusting controller does not fully exploit the advantages of CMA-C due to its relatively small infeasible region.

6.2 Parameterization and Concatenation

Once a controller is developed, generalizing it to a family of similar controllers can be done easily by parameterizing the task objective function (Equation (22)):

$$f(\mathcal{P}, \mathbf{u}) = \sum_{i=1}^{n} \|h_i(\mathbf{q}_f) - \hat{h}_i(\mathbf{q}_f, \mathbf{u})\| \tag{28}$$

where $\mathbf{u}$ is the varying task parameter among the parameterized controllers. For example, if $\mathbf{u}$ represents the forward leaning angle and h_i evaluates the center of mass position at the final state, we can produce a family of new targets for the center of mass by varying $\mathbf{u}$ in $\hat{h}_i(\mathbf{q}_f, \mathbf{u})$. Because all the controllers share the same constraints, the SVMs built for the original controller can be reused. As a result, optimizing a family of parameterized controllers can be done efficiently without additional user effort.

Our framework also supports concatenation of controllers by utilizing the parameterized controllers. Given two controller A and B, the naive way to optimize both controllers is putting them together in one big problem and optimizing control parameters $\mathcal{P}_A$ and $\mathcal{P}_B$ simultaneously. To resolve the issue of increasing dimensionality, we first parameterize A with the task parameter $\mathbf{u}$ to generate a family of controllers $\mathcal{A}$. When optimizing B, we include $\mathbf{u}$ as a free variable along with other rig parameters for B. The simulated motions depend on $\mathbf{u}$ because different $\mathbf{u}$ results in different initial states of the simulation. Once the optimizer obtains an optimal value $\mathbf{u}^*$, we choose one controller whose task parameter is the closest to $\mathbf{u}^*$ from $\mathcal{A}$. Finally we concatenate the chosen controller and B to generate a longer motion sequence.

6.3 Results

We developed a few different Parkour movements to demonstrate the generality of our framework. All the results shown in the video were produced on a single core of 3.20GHz CPU. Our program runs at 1000 frames per second with the time step

of 0.5 millisecond. We used RTQL8 [82], an open-source simulator for multibody dynamics based on generalized coordinates. The character has 33 degrees of freedom. Except for the first six degrees of freedom that describe the global translation and orientation, all other degrees of freedom can be actuated.

Table 10: Instructions used to train a precision jump.

Phase	Instruction
Leaning	MOVE downward BY 0.1m
Leaning	MOVE downward BY 0.2m
Leaning	head IN FRONT OF root
Thrusting	SPEED near $[1.2, 2.4, 0.0]^T$
Thrusting	SLOWDOWN 20%
Thrusting	spin about z FORWARD
Airborne	PLACE feet NEAR $[0.8, 0.0, 0.0]^T$
Airborne	PLACE COM NEAR $[0.7, 0.4, 0.0]^T$
Airborne	MOVE upward BY 0.1m
Landing	PLACE COM NEAR $[0.7, 0.5, 0.0]^T$
Landing	BALANCE
All	RELAX arms BY 30%
Leaning	FLEX elbows BY 20°
Airborne	FLEX elbows BY 80°

6.3.1 User Input

Our system requires the user to break down a complex motion into phases and provide an initial controller for each phase. Determining the phases of a motion is at the user's discretion. In our experiments, we simply used the timing of contact change to determine phases. Similarly, the initial controllers do not have significant impact on the final controllers. All the initial controllers we used in our experiments were simple PD controllers with the same gains and damping coefficients. The only goal of each controller was to track a single target pose roughly defined by the user (Figure 29).

During the training process, the user needs to provide instructions iteratively to improve the motor skill of the character. However, we observed that the character could successfully learn a motor skill, even when the instructions were clearly not

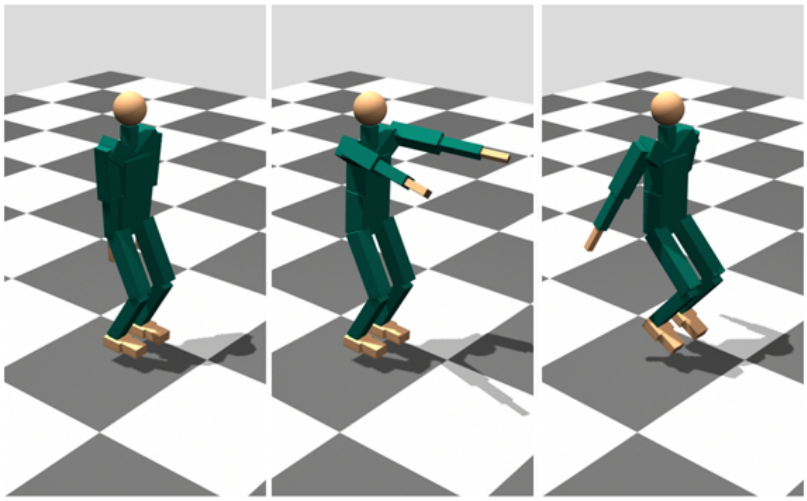


Figure 29: We use these three target poses for all the initial controllers, except for the rolling phase of drop-and-roll.

optimal. For example, in the scenario of training a precision jump (Table 10), the user gave repetitive commands using "MOVE downward BY" instruction to adjust the leaning angle of the character. For more complex cases, we believe that the user's prior knowledge about the motion will become an important factor. This is also true for learning motions in real world; a better coach can diagnose problems of the movement more precisely and effectively.

When the user gives conflicting instructions, such as "SPEEDUP 200%" and "TRANSLATE torso backward", the optimizer will yield a solution which tries to achieve both conflicting goals, but the resulting motion could be undesirable or unpredictable.

6.3.2 Training Dynamic Skills

Precision Jump. A precision jump is a jumping motion that lands on a narrow target, such as the top of a wall or a rail. The precise distance and the smaller target require more accurate and coordinated control. We broke the entire sequence into four phases based on the contact state: leaning, thrusting, airborne, and landing. We trained each phase separately and applied parameterization and concatenation

technique to sequence them into one controller for precision jump. The final state of the motion satisfies the balance condition, which limits the ground projection of center of mass within the suppor polygon and the velocity of center of mass to near zero.

The initial controller for each phase was a simple PD controller tracking a roughly designed pose (Figure 29). Because of the lack of control on the global state, the initial controller resulted in falling motion immediately. In all of our examples, we designed initial controllers in the same fashion.

Training all four phases took 14 instructions in total. We listed all the instructions in Table 10. For each phase, the average number of control rigs used was 3.25, which resulted in 4.25 rig parameters to optimize. The optimization time for each phase on average took two minutes. The task parameters we used for parameterizing four phases are the leaning angle, thrusting direction, and airborne traveling distance. The average time spent on parameterization of one phase is 30 minutes.

During the coaching stages, we first gave instructions to guide global motion so the character can successfully perform the jump without falling. Later, we added instructions to modify styles on the upper body. The instructions we used might not be the most effective ones because we are not experts in Parkour. For example, we used a few consecutive instructions to lower the center of mass, which could have been done in one instruction. Similarly, we instructed the character to flex the elbows in the leaning phase and later added the same style in the airborne phase. Although the total number of instructions can be reduced, our goal here is to demonstrate how this framework is used by a non-expert who tends to give imprecise and incremental instructions based on the feedback from the trainee.

Turnaround Jump. A turnaround jump requires the performer to jump in place while turning in a precise angle. Although it consists of the same four phases as a precision jump, the difference is that it involves asymmetric motion and the control

of angular momentum.

In our experiment, training all phases of a turnaround jump required nine instructions. For each phase, the average number of control rigs was 3.75 and the average number of rig parameters was 4.75. The optimization time for each phase was on average one minute. We parameterized the thrusting angular momentum, which took 12 minutes to complete.

We found that the coaching skill of the human user can also improve by using this framework. Because of our previous experience in coaching a precision jump, we used fewer instructions to train a turnaround jump. We also gained better insight on developing more natural landing controller. That is, we instructed the character to raise its center of mass at landing so that the character had more room to absorb the impact.

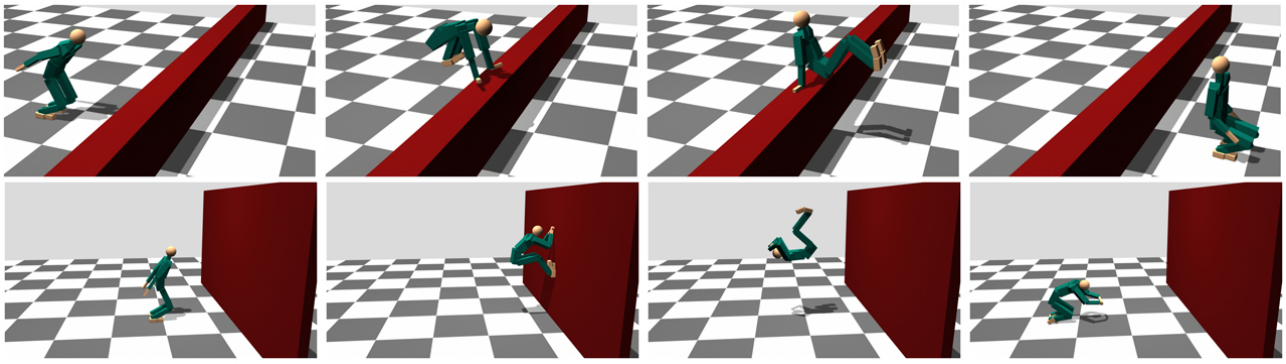


Figure 30: Monkey vault and wall-backflip.

Monkey Vault. A monkey vault is a basic Parkour movement for getting over obstacle without slowing down the motion (Figure 30). The performer approches the obstacle with squat position and uses the hands to reach for the vault. As the performer jumps in the air, s/he leans forward and tucks the legs against the upper body. Since the hands are placed wider than the shoulder width, the legs can pass through in between the arms. We broke the vaulting motion into six phases: leaning, thrusting, airborne, pushing, extending, and landing.

Training all six phases of a monkey vault took 26 instructions. For each phase, the average number of control rigs was 2.33 and the average number of rig parameters was 3.66. The optimization time for each phase was on average three minutes. We parameterized the monkey vault controller by its leaning angle, thrusting direction, pushing direction, and extending velocity. The computation time for parameterization was 40 minutes for each phase.

Training a monkey vault was the most challenging task among all our experiments because our intuition of monkey vault was limited. For example, it was not clear to us when the character should accelerate or which direction the character should push. The instructions we used were more back-and-forth and repetitive due to our unfamiliarity to this motor skill.

Drop-and-roll. The purpose of rolling in Parkour is to protect joints from the landing impact. We designed a rolling controller from a standing position and later showed that it can be concatenated to a precision jump or a monkey vault to complete a drop-and-roll motion. We broke the rolling motion to three phases: leaning, thrusting, and follow-through.

Training the entire rolling motion required eight instructions. For each phase, the average number of control rigs was 2.66 and the average number of rig parameters was 4.66. The optimization time for each phase was on average 1 minute. We parameterized the rolling controller by the leaning angle and the angular momentum at the thrusting phase. The computation time for parameterization was about ten minutes for each phase.

Developing a drop-and-roll controller is relatively easy because it has only three phases and the follow-through phase is very passive. As long as the angular momentum is sufficient, the character naturally goes into rolling motion.

Wall-backflip. A wall-backflip is a combination of a wall-run and a backflip (Figure

30). The performer jumps toward the wall, kicks the wall at contact, flips backward in the air, and lands on both feet. We broke the task into six phases: leaning, thrusting, airborne, kicking, flipping, and landing.

Training all six phases require 18 instructions. For each phase, the average number of control rigs was 2.66 and the average number of rig parameters was 4.00. The optimization time for each phase was on average two minutes. We parameterized the controller by the angular momentum at the kicking phase. The computation time for parameterization was about 30 minutes for each phase.

The experience of coaching monkey vault greatly helped us to train a wall-backflip. They share similar phases, but the main difference is that the character needs to maximize the backward angular momentum at the kicking phase. Without optimization, the direction of the kicking force would have been a difficult parameter to tune because it must generate sufficient angular momentum without causing slipping contact.

6.3.3 Limitations

Our controllers are able to withstand some perturbations. For example, the same precision jump controller trained for landing on the floor can be applied to landing on a rail. However, most controllers become unstable when additional push forces are applied to the character. We suspect that the instability is due to the feedforward nature of the virtual force control rig.

Using contact states to break a task into smaller phases is a good strategy for the examples we demonstrated, but it is not sufficient for more complex and timing-sensitive motor skills, such as tic-tac or wall-run. These movements switch controllers not only based on contact states, but also on character's pose, speed, or spatial relation to the environment. One possible solution is to design more sophisticated control rigs which include time-varying rig parameters.

In the absence of a running controller, we were not able to generate some motor skills which require a high initial forward momentum to carry out the motion smoothly. We believe that the monkey vault motion can be largely improved if we concatenate it with an adequate running controller.

Although the performance of CMA-C on our Parkour problems is five times faster than the standard CMA on average, the large variance in performance gain (150% to 650%) indicates that further evaluation on a broader set of benchmark problems is needed. Our toy problems do not provide comprehensive analyses on CMA-C because they made specific assumptions about the shape of the cost functions, such as asymmetric random noise or symmetric undulation. These assumptions may not reflect the true landscape of the cost functions in the Parkour problems. We conjecture that the success of CMA-C on the Parkour problems is due to the large infeasible regions and multiple local minima in the cost functions, which may cause standard CMA to converge slower.

6.4 Conclusion

We present a novel framework for controller design using human coaching and learning techniques. The framework takes in a blackbox controller and improves it through an iterative learning process of coaching and practicing. The user can directly give high-level instructions to the virtual character using control rigs while the character can efficiently optimize its motor skill using a new sampling-based optimization method, CMA-C. Once a controller is developed, parameterizing it to a family of similar controllers for concatenation can be done without additional effort from the user.

In this work, we demonstrated that developing complex motor controllers does not need any motion trajectories. However, if the user wishes to utilize motion examples to train the virtual character, instead of verbal instructions, our framework can be adapted by including a control rig that modulates the reference trajectories, similar

to the sampling approach proposed by Liu *et al.* [59, 58].

Our current implementation requires the user to construct instructions as a script according to the template grammars. One possible future direction is to augment our framework with a natural language processor so the user can use more colloquial commands to train the character, such as "Lower your body a bit more", instead of "MOVE COM down BY 0.2m". In addition, we would like to explore Kinet-like sensors to enable the possibility of "teaching by demonstration". Therefore, one possible future direction is to augment our framework with two different types of interfaces: a natural language processor and a Kinect-like sensor. These two interfaces will allow the user to train the character by describing the motion in human language while demonstrating the movement using his/her own body.

CHAPTER VII

OPTIMIZATION FOR PARAMETRIZED MOTOR SKILLS

Learning a parameterized skill is essential for autonomous robots operating in an unpredictable environment. Previous techniques learned a policy for each example task individually and constructed a regression model to map between task and policy parameter spaces. However, these techniques have less success when applied to whole-body dynamic skills, such as jumping or walking, which involve the challenges of handling discrete contacts and balancing an under-actuated system under gravity. This paper introduces an evolutionary optimization algorithm for learning parameterized skills to achieve whole-body dynamic tasks. Our algorithm simultaneously learns policies for a range of tasks instead of learning each policy individually. The problem can be formulated as a nonconvex optimization whose solution is a closed segment of curve instead of a point in the policy parameter space. We develop a new optimization algorithm which maintains a parameterized probability distribution for the entire range of tasks and iteratively updates the distribution using selected elite samples. Our algorithm is able to better exploit each sample, greatly reducing the number of samples required to optimize a parameterized skill for all the tasks in the range of interest.

7.1 Motivation

Being able to reinterpret learned motor skills to create and perform appropriate motions in new situations is an essential ability for autonomous robots. One promising approach to achieving skill generalization is to teach robots not just the skill for a specific task, but a parameterized skill, the ability to tackle a family of related tasks varying by some parameters. Instead of switching from policy to policy as the task

description varies, a parameterized skill automatically maps the task parameters to appropriate policy parameters and executes the control policy optimally according to the task.

The challenge of learning a parameterized skill lies in learning the mapping between task parameters and policy parameters. Many existing techniques ([92, 16]) learn a policy for each example task individually and construct a regression model to map between task and policy parameter spaces. Successful learning of parameterized skills for manipulation tasks, such as throwing darts [52, 18], reaching objects [99, 62, 25], or hitting a table tennis ball [52], have been demonstrated. However, it is unclear whether these techniques can be applied to learning whole-body dynamic skills, such as jumping or walking, which involve the challenges of handling discrete contacts and balancing an under-actuated system under gravity. Because a whole-body dynamic task can often be achieved by multiple distinctive policies, learning each task individually might lead to a set of incoherent policies which invalidates the regression model. This is equivalent to solving a set of highly nonconvex optimization problems and attempting to interpolate all the arbitrarily chosen local minima.

This paper aims to develop new algorithms for learning parameterized skills for whole-body dynamic tasks. Instead of learning each task individually and later building a regression model, we propose to simultaneously learn the policies for a range of tasks. A naive approach to achieve this goal is to formulate the search of the mapping function between task parameters and policy parameters as an nonconvex (due to dynamic differential equations) and nondifferentiable (due to contacts) optimization and apply a sampling-based, gradient-free solver, such as CMA-ES [35], to find the solution. This approach can be highly ineffective when the mapping function is complex and sample generation is computationally expensive.

In this paper, we develop a new evolutionary optimization algorithm to tackle

above issues arise from learning parameterized skills. Similar to CMA-ES, our optimization algorithm draws samples from the current probability distribution and selects elite samples to update the probability distribution for the next generation. Unlike CMA-ES, however, we maintain a parameterized probability distribution for the entire range of tasks, instead of a single distribution for only one task. Solving multiple tasks simultaneously, our algorithm is able to better exploit each sample, greatly reducing the number of samples required to optimize a parameterized skill for all the tasks in the range of interest.

We demonstrate the algorithm on a simulated humanoid robot, BioloidGP [1], learning three parameterized dynamic motor skills, including vertical jump, kick a ball, and walk. We also deploy the walking policy to the hardware of BioloidGP. Furthermore, we demonstrate the sample efficiency of our algorithm by comparing with CMA-ES on solving four CEC'15 benchmark problems [11].

7.1.1 Related work

There is a large body of research work on generalization of learned motor skills to achieve new tasks. da Silva *et al.* [18, 16, 17] introduced a framework to represent the policies of related tasks as a lower-dimensional piecewise-smooth manifold. Their method also classifies example tasks into disjoint lower-dimensional charts and model different sub-skills separately. Much research aimed to generalize example trajectories to new situations using dynamic movement primitives (DMPs) to represent control policies [41]. A DMP defines a form of control policies which consists of a feedback term and a feedforward forcing term. Ude *et al.* [99] used supervised learning to train a set of DMPs for various tasks and built a regression model to map task parameters to the policy parameters in DMPs. Muelling *et al.* [69] proposed a mixture of DMPs and used a gate network to activate the appropriate primitive for the given target parameters. Kober*et al.* [52] trained a mapping between task parameters

and meta-parameters in DMPs using a cost-regularized kernel regression. Through reinforcement learning framework, they computed a policy which is a probability distribution over meta-parameters. Matsubara *et al.* [62] trained a parametric DMP by shaping a parametric-attractor landscape from multiple demonstrations. Stulp *et al.* [92] proposed to integrate the task parameters as part of the function approximator of the DMP, resulting in more compact model representation which allows for more flexible regression. Neumann *et al.* [73] modified the existing learning algorithm (REPS) to learn a hierarchical controller that has parameterized options.

All these methods described above depend on collecting a set of examples. This presents a bottleneck to learning because an individual control policy needs to be learned for each task example drawn from the distribution of interest. da Silva *et al.* further proposed using unsuccessful policies as additional training samples to accelerate the learning process [16]. For dynamic motor skills which involve intricate balance tasks, unsuccessful policies generated during training a particular task are of no use to other tasks because they often lead to falling motion. Hausknecht *et al.* [36] demonstrated a quadruped robot kicking a ball to various distances, but whole-body balance was not considered in their work. Another challenge regarding dynamic tasks is that each task can be achieved by a variety of policies, some of which might be overfitting the task. Interpolating these overfitted policies can lead to unexpected results. Our method tends to generate more coherent mapping between task parameters and policy parameters because we simultaneously learn the policies for the entire range of the tasks.

Various optimization techniques have been applied to improve the motion quality or the robustness of the controller. In computer animation, a sampling-based method, Covariance Matrix Adaption Evolution Strategy (CMA-ES) [35], has been frequently applied to discontinuous control problems, such as biped locomotion [101, 102], parkour-style stunts[58, 32], or swimming [97]. To compensate the expensive

cost of sampling-based algorithm, different approaches have been proposed, including exploiting the domain knowledge [101, 102], shortening the problem horizons [89], or using a classifier to exclude infeasible samples [32]. Based on the previous success of CMA-ES, we developed a new sampling-based algorithm tailored to optimizing parameterized motor skills.

7.2 Parameterized Optimization Problems

Let us begin with a general optimization problem, whose goal is to find a point in $\mathcal{R}^n$ that minimizes a given objective function $f(\mathbf{x})$. By varying some parameters in the objective function, we can create a family of similar optimization problems, whose objective functions are denoted as $f(\mathbf{x}; w)$. We define the varying parameter $w \in [0, 1]$ as *task interpolation parameter*, which controls the values of some parameters in $f(\mathbf{x})$. The solution to such a parameterized problem is no longer a point in $\mathcal{R}^n$, but rather a closed segment of curve, which can be represented by a function of the task interpolation parameter, $\mathbf{x}(w)$, with a closed domain $[0, 1]$. The function form of the solution segment, $\mathbf{x}(w)$ is unknown in many cases but can be approximated by a simpler function. We assume this simple function is continuous and can be represented by some function parameters $\boldsymbol{\phi} \in \mathcal{F}$. For example, we can assume that $\mathbf{x}(w)$ is a linear line segment,

$$\mathbf{x}_{\boldsymbol{\phi}}(w) = (1 - w)\boldsymbol{\phi}_0 + w\boldsymbol{\phi}_1 \tag{29}$$

where $\boldsymbol{\phi} = [\boldsymbol{\phi}_0^T, \boldsymbol{\phi}_1^T]^T$ are the parameters that define the solution segment. Likewise, we can assume that $\mathbf{x}(w)$ is a cubic curve segment:

$$\mathbf{x}_{\boldsymbol{\phi}}(w) = (1 - w)^3\boldsymbol{\phi}_0 + 3w(1 - w)^2\boldsymbol{\phi}_1 + 3w^2(1 - w)\boldsymbol{\phi}_2 + w^3\boldsymbol{\phi}_3 \tag{30}$$

where $\boldsymbol{\phi}$ is $[\boldsymbol{\phi}_0^T, .., \boldsymbol{\phi}_3^T]^T$.

Our goal is to develop an efficient optimization method to find the solution segment (parameterized by $\boldsymbol{\phi}$) for the entire family of problems simultaneously. To this end,

we define a new objective function which evaluates the integral of the parameterized objective function over the closed domain:

$$\hat{f}(\boldsymbol{\phi}) = \int_0^1 f(\mathbf{x}_{\boldsymbol{\phi}}(w); w)dw. \tag{31}$$

In practice, however, $\hat{f}(\boldsymbol{\phi})$ can only be numerically approximated in most dynamic control problems, whose objective function f is usually not closed-form. To this end, we discretize $\hat{f}(\boldsymbol{\phi})$ as follows:

$$\hat{f}(\boldsymbol{\phi}) = \frac{1}{M} \sum_{w_i \in \mathbf{w}} f(\mathbf{x}_{\boldsymbol{\phi}}(w_i); w_i) \tag{32}$$

where $\mathbf{w}$ is a set of real values evenly discretizing the range $[0, 1]$ and M is the size of the set $\mathbf{w}$. For example, if $M = 6$, $\mathbf{w}$ is $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

7.2.1 Parameterized Motor Skills

The parameterized optimization problem provides a general framework for learning parameterized motor skills for robots. We will begin with the notations of a standard control problem and extend them to a parameterized problem. A controller $\mathbf{C} = \{\pi, f\}$ consists of a policy function π and a cost function f. A policy function outputs the control force $\boldsymbol{\tau}_t$ for a given state $\mathbf{s}_t$,

$$\boldsymbol{\tau}_t = \pi(\mathbf{s}_t; \mathbf{x}) \tag{33}$$

where $\mathbf{x}$ is the vector of policy parameters. Starting from the initial state of the robot $\mathbf{s}_0$, a physics simulator generates a sequence of motions $\mathcal{S}(\mathbf{x}) = \{\mathbf{s}_0, \cdots, \mathbf{s}_T\}$ according to the equations of motion and the policy parameters $\mathbf{x}$. The motion quality produced by the policy parameters can be evaluated by the cost function:

$$f(\mathbf{x}) = \sum_j \|\mathbf{h}_j(\mathcal{S}(\mathbf{x})) - \hat{\mathbf{h}}_j\|^2 \tag{34}$$

where f evaluates a set of features $\mathbf{h}_j(\mathcal{S}(\mathbf{x}))$, at a given state of the motion ($\mathbf{s} \in \mathcal{S}(\mathbf{x})$). The desired value of each feature is denoted as $\hat{\mathbf{h}}_j$. For example, a feature $\mathbf{h}_j(\mathbf{s}_t)$ can

be the center of mass of the robot at the state $\mathbf{s}_t$ and $\hat{\mathbf{h}}_j$ can be the desired position of the center of mass.

To create controllers that can achieve a range of tasks, rather than specializing in only one specific task, we redefine the policy parameters as $\mathbf{x}_{\boldsymbol{\phi}}(w)$, a mapping between a given task interpolation parameter w and the policy parameter $\mathbf{x}$. We define $\mathbf{x}_{\boldsymbol{\phi}}(w)$ as *parameterized skill function*. Likewise, we represent the desired target value $\hat{\mathbf{h}}_j$ as a function of the task interpolation parameter, w, between two desired values $\hat{\mathbf{h}}_j^0$ and $\hat{\mathbf{h}}_j^1$:

$$\begin{gathered} \hat{\mathbf{h}}_j(w) = (1-w)\hat{\mathbf{h}}_j^0 + w\hat{\mathbf{h}}_j^1 \\ \text{where } \ 0 \leq w \leq 1. \end{gathered} \tag{35}$$

The parameterization redefines the policy function and the cost function as follows:

$$\boldsymbol{\tau}_t = \pi(\mathbf{s}_t; \mathbf{x}_{\boldsymbol{\phi}}(w)), \tag{36}$$

$$f(\mathbf{x}_{\boldsymbol{\phi}}(w); w) = \sum_j \|\mathbf{h}_j(\mathcal{S}(\mathbf{x}_{\boldsymbol{\phi}}(w))) - \hat{\mathbf{h}}_j(w)\|^2 \tag{37}$$

Analogous to Equation (32), the goal of the parameterized optimization is to find the best mapping parameters $\boldsymbol{\phi}$ that minimizes the objective function,

$$\hat{f}(\boldsymbol{\phi}) = \frac{1}{M} \sum_{w_i \in \mathbf{w}} f(\mathbf{x}_{\boldsymbol{\phi}}(w_i); w_i). \tag{38}$$

7.3 Optimization Algorithms

For most control problems involving dynamic equations and contacts, the objective function $\hat{f}(\boldsymbol{\phi})$ (Equation (38)) is highly non-convex and often non-differentiable. A common practice is to use a sampling-based optimization algorithm to find the optimal solution. In particular, CMA-ES [35] has been successfully applied to many high-dimensional control problems. We will first summarize the core ideas of CMA-ES and use it as a baseline to compare with our new optimization algorithm.

7.3.1 Baseline algorithm: CMA-ES

CMA-ES is a sampling-based, derivative-free evolutionary algorithm for solving optimization problems. The evolution process updates a multivariate normal distribution

$$\mathbf{x}_k \sim \mathbf{m} + \sigma \mathcal{N}(\mathbf{0}, \mathbf{C}) \text{ for } k = 1..\lambda \tag{39}$$

where $\mathbf{m}$ is the mean, σ is the step size, and $\mathcal{N}(\mathbf{0}, \mathbf{C})$ is the multivariate normal distribution with zero mean and covariance matrix $\mathbf{C}$. Initially, $\mathbf{m}$ is set to zero and $\mathbf{C}$ is set to identity. In each generation, λ new candidates are sampled from the current multivariate normal distribution. All the samples are evaluated by the objective function and the best μ samples are selected to update the mean and the covariance matrix for the next generation. When the termination criteria are met, the final mean value is output as the optimal point of the optimization. A number of follow-on research introduced different schemes for sample selection and different rules for updating the probability distribution. A comprehensive introduction can be found in the tutorial by Hansen *et al.* [35].

The rule for adapting covariance matrix is a key characteristic of CMA-ES. Among different versions, (1+1)-CMA-ES [40] has the simplest mechanism to evolve covariance matrix. In every generation, if a sample is not better than the mean from the previous generation, it will be discarded and a new one will be drawn. The process repeats until a better sample is generated (called elite sample). The elite sample will replace the mean and update the covariance matrix via a procedure called **updateCov**, which details are summarized in Appendix. **updateCov** takes as input the previous mean and the elite sample, and outputs the adapted covariance matrix $\mathbf{C}$.

7.3.2 Our Algorithm for Parameterized Optimization Problem

Directly applying CMA-ES to solve a parameterized optimization problem is difficult due to the expanded dimensionality. Consider a policy parameter space $\mathcal{X} = \mathcal{R}^n$ and a cubic representation for the parameterized skill function, $\mathbf{x}_{\phi}(w)$, where $\mathcal{F} = \mathcal{R}^{4n}$.

The dimension of $\mathcal{F}$ can be higher if we use more complex function forms to represent $\mathbf{x}_{\boldsymbol{\phi}}(w)$. Sampling in the high-dimensional $\mathcal{F}$ can result in very slow convergence for CMA-ES.

We propose a new sampling-based algorithm to combat the issue of expanded dimensionality. Our key idea is to sample in the space of policy parameters, $\mathcal{X}$, instead of the space of $\mathcal{F}$. This alternative sampling space will lead to the same solution because there is a one-to-one mapping between a point in $\mathcal{F}$ and a curve segment in $\mathcal{X}$. Using CMA-ES to evolve the mean point in $\mathcal{F}$ is equivalent to evolving the *mean segment* in $\mathcal{X}$. The obvious advantage of drawing samples from the space of policy parameters is that the dimensionality is invariant to the complexity of the parameterized skill function. Moreover, evaluating a sample $\mathbf{x}$ only requires one trial of a simulation $\mathcal{S}(\mathbf{x})$ (Equation (37)), while evaluating a sample $\boldsymbol{\phi}$ requires integration of f over $w \in [0, 1]$. Using the discretized approximation shown in Equation (38), every evaluation of $\boldsymbol{\phi}$ costs M simulation trials.

Since we are solving for a solution segment rather than a single solution point, the definitions of the mean and covariance matrix also need to be modified. We define a continuous function: $\mathbf{m}_{\boldsymbol{\phi}}(w) : [0, 1] \mapsto \mathcal{R}^n$ to represent the *mean segment*. Similarly, we also need to parameterize the covariance matrix $\mathbf{C}$ with the task interpolation parameter as $\mathbf{C}(w)$. Unlike the *mean segment*, we do not represent $\mathbf{C}(w)$ as a continuous function because accurate estimation of continuous $\mathbf{C}(w)$ requires a large number of sample evaluations. We choose a simpler way to estimate the covariance of the parameterized probability function by maintaining one covariance matrix $\mathbf{C}(w_i)$ for each discretized task interpolation parameter w_i.

Three questions arise when we extend the mean from a single point to a segment in the policy parameter space. How do we draw samples from the current distribution, how do we evaluate and select samples, and how do we update the distribution?

7.3.2.1 Drawing samples

To draw samples from the current parameterized mean $\mathbf{m}_{\boldsymbol{\phi}}(w)$ and covariance matrix $\mathbf{C}(w)$, we define the following probability distribution:

$$\mathbf{x}_k \sim \frac{1}{M} \sum_{w_i \in \mathbf{w}} \Big(\mathbf{m}_{\boldsymbol{\phi}}(w_i) + \sigma \mathcal{N}(0, \mathbf{C}(w_i)) \Big) \tag{40}$$

where k is the sample index, σ is the global step size that controls the scale of the distribution (the details will be explained later), and w_i is one of the M discretized task interpolation parameters. This formulation is equivalent to drawing samples from a mixture of Gaussian models with a uniform weight.

7.3.2.2 Selecting samples

We draw λ samples from the current probability distribution and mix the new samples with μ samples from the previous generation. From this pool of samples, we select the best ν samples for each task w_i by evaluating the task cost using Equation (37). We make $\nu = \mu/M$ so that the size of elite sample set remains μ from generation to generation. We denote each elite sample as $\bar{\mathbf{x}}_i^k$, the k^{th} sample for the task w_i.

Note that we use a set of various objective functions (Equation (37)) with different w_i to select elite samples, instead of a single objective function that sums up the costs over the range of the task. We prefer specialized samples (excellent for a particular task but mediocre on other tasks) rather than "well-rounded" samples (adequate for all tasks), because we need to keep the diversity in the elite sample pool in order to evolve the entire range of tasks. If we only keep "well-rounded" samples, the sample pool will become more and more assimilated due to the single objective function. Eventually, the mean segment will converge to a single point instead of a curve segment.

7.3.2.3 Updating the model

After sample selection step, we can proceed to use these μ elite samples to update the probability model. If the new model results in lower objective cost, we accept the new model and move on to the next generation. Otherwise, we reject the new model and repeat the steps of drawing samples and selecting samples described above.

To compute a new mean segment, we apply regression on the μ elite samples to solve for the mapping parameters $\boldsymbol{\phi}$: $\mathbf{m}_{\boldsymbol{\phi}}(w)$.

$$\boldsymbol{\phi} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{w_i \in \mathbf{w}} \sum_{k=1}^{\nu} \|\bar{\mathbf{x}}_i^k - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\|. \tag{41}$$

However, using *all* the selected samples tend to yield suboptimal $\mathbf{m}_{\boldsymbol{\phi}}(w)$ because some samples might have relatively high task costs (Equation (37)). Alternatively, we could use only the best sample for each task to fit $\mathbf{m}_{\boldsymbol{\phi}}(w)$, but these best samples might not be well aligned with the assumed function form of $\mathbf{m}_{\boldsymbol{\phi}}(w)$, resulting in huge error in regression and a suboptimal mean segment.

We propose a new scheme to update the mean segment $\mathbf{m}_{\boldsymbol{\phi}}(w)$. Instead of using *all* the elite samples or only the *best* elite sample for each task, we search for the best combination of samples such that it minimizes the task cost and the regression error at the same time. Our algorithm first combines samples into groups of M by randomly selecting one sample for each task w_i, i.e. selecting one from $\bar{\mathbf{x}}_i^1, \cdots, \bar{\mathbf{x}}_i^\nu$. Once a group $\mathbf{g}$ is formed, we use the following function to evaluate the cost of the group.

$$f_{group}(\mathbf{g}) = \sum_{w_i \in \mathbf{w}} f(\bar{\mathbf{x}}_i^{g_i}; w_i) + \alpha \min_{\boldsymbol{\phi}} \sum_{w_i \in \mathbf{w}} \|\bar{\mathbf{x}}_i^{g_i} - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\| \tag{42}$$

where g_i indicates the index of selected sample for task w_i. α (=10.0) adjusts the weights between the task cost and the regression error. Note that each evaluation of Equation (42) involves solving a regression for $\boldsymbol{\phi}$, but the computational cost for regression is fairly low.

Because the number of possible groups can be potentially very large ($= \nu^M$), our algorithm only tests 100 randomly selected groups. The function parameters $\boldsymbol{\phi}$ that yield the best group $\mathbf{g}^*$ in Equation (42) are used to define the new mean segment:

$$\boldsymbol{\phi} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{w_i \in \mathbf{w}} \|\bar{\mathbf{x}}_i^{g_i^*} - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\|. \tag{43}$$

The new mean segment will be compared to the current mean by evaluating its cost (Equation (38)). If the new mean has a lower cost, we accept it, increase the step size, and move on to updating the covariance matrix. Otherwise, we discard the new mean, decrease the step size, and try again with another set of samples redrawn from the current model.

The step size update is based on the standard 1/5-success-rule.

$$\sigma = \begin{cases} \sigma \cdot exp(1/3) & \text{if a new model is accepted} \\ \sigma / exp(1/3)^{(1/4)} & \text{otherwise} \end{cases} \tag{44}$$

Finally, to adapt the parameterized covariance matrices, we apply the update rules used in (1+1)-CMA-ES. For each covariance $\mathbf{C}(w_i)$, we call the procedure **update-Cov** with the newly accepted mean and the previous mean segments.

7.4 Result

We compared our algorithm and the baseline algorithm, CMA-ES, on three dynamic motor control problems and four parameterized CEC'15 problems. The baseline algorithm uses CMA-ES to search for the mapping parameter $\boldsymbol{\phi} \in \mathcal{F}$ that minimizes the objective function $\hat{f}(\boldsymbol{\phi})$ (Equation (38)), while our algorithm follows the procedure described in Section 7.3.2.

For all problems, we measured the number of sample evaluations and/or the quality of the final solution. Evaluating a sample in a motor control problem involves simulating a sequence of motion, which is typically the bottleneck of using a sampling-based optimizer to solve a control problem. Because of the stochastic nature of our

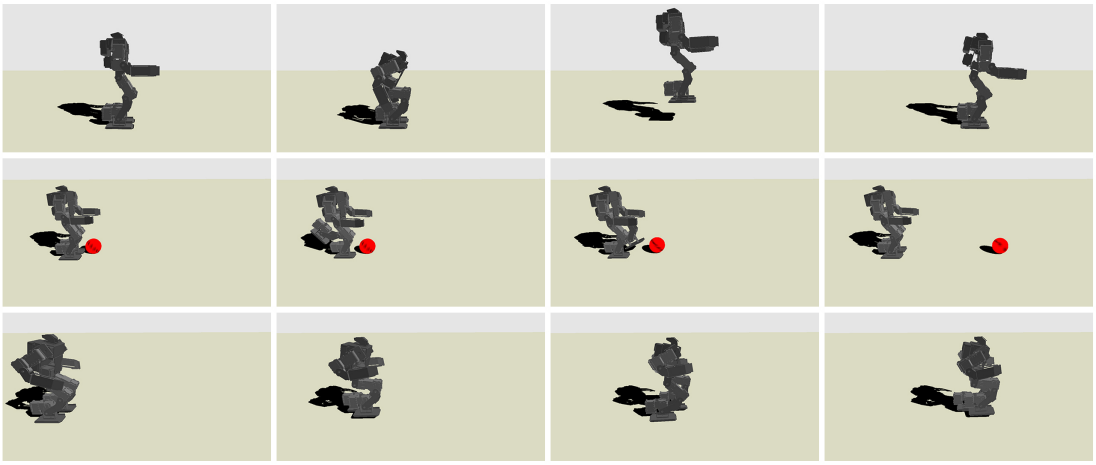


Figure 31: Top: Vertical jump with parameterized target height, $3cm$ to $8cm$ ($8cm$ is shown). Middle: Kick with parameterized target distance, $0.3m$ to $0.6m$ ($0.6m$ is shown). Bottom: Walk with parameterized target speed, $6.7cm/s$ to $13.3cm/s$ ($13.3cm/s$ is shown).

algorithm, for each problem we ran nine optimization trials with different initial seeds and reported the average results of seven trials, excluding the best and the worst trials. Our algorithm generates $\lambda = 16$ samples at each iteration and maintains $\mu = 48$ elite samples, while CMA-ES generates $\lambda_{cma} = 16$ samples each iteration and selects $\mu_{cma} = 8$ elite samples.

7.4.1 Parameterized Motor Skills

The primary goal of this work is to learn controllers for parameterized dynamic motor skills. We experimented three dynamic tasks, vertical jump, kick a ball, and walk (Figure 31), on a small humanoid, BioloidGP [1] ($34.6cm$ and $1.6kg$). All motions were simulated using an open source physics engine, DART [20, 56], with $0.0005s$ as the simulation time step. We set the joint angle limits at $\pm 150^\circ$ and the torque limits at $0.6Nm$. Contact and collision were handled using implicit time-stepping, velocity-based LCP (linear-complementarity problem) to guarantee non-penetration, directional friction, and approximated Coulomb friction cone conditions.

The parameterized walking skill was also demonstrated on the hardware. Please

see the supplementary video for simulated motion sequences and recorded video footages.

We used two simple control mechanisms to define the policy parameter space and let our algorithm find the optimal policy. The first control mechanism is to use PD servos to track target poses. The second one is to control the desired force a body link exerts to the world using Jacobian transpose [95]. The definitions of cost function vary by tasks and will be described in the following subsections.

7.4.1.1 Vertical Jump

Our goal is to learn a parameterized vertical jump skill ranging from $3cm$ to $8cm$. We designed two objective terms to achieve desired motion: the balance term and the apex height of center of mass.

$$f_{jump}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + (h_{apex}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{h}^0_{apex} + w\hat{h}^1_{apex}))^2 \quad (45)$$

where $h_{apex}(\mathcal{S}(\mathbf{x}))$ evaluates the height of center of mass at apex of the jump. We set $\hat{h}^0_{apex} = 0.22$ and $\hat{h}^1_{apex} = 0.27$ since the robot's center of mass height is $0.19m$ at the rest pose. $\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))$ evaluates the state of balance by computing the squared sum of horizontal distances between the center of mass and the center of pressure over the entire motion.

We broke a vertical jump into three phases: preparing, thrusting, and landing. The policy π_{jump} is then defined by six parameters: the target hip, knee, and ankle angles during preparing phase, the intended magnitude of force exerted to the ground during thrusting phase, and the target hip and knee angles during landing phase.

Because the humanoid hardware is not sufficiently powerful to perform vertical jumps, we remove the torque limits during the thrusting phase.

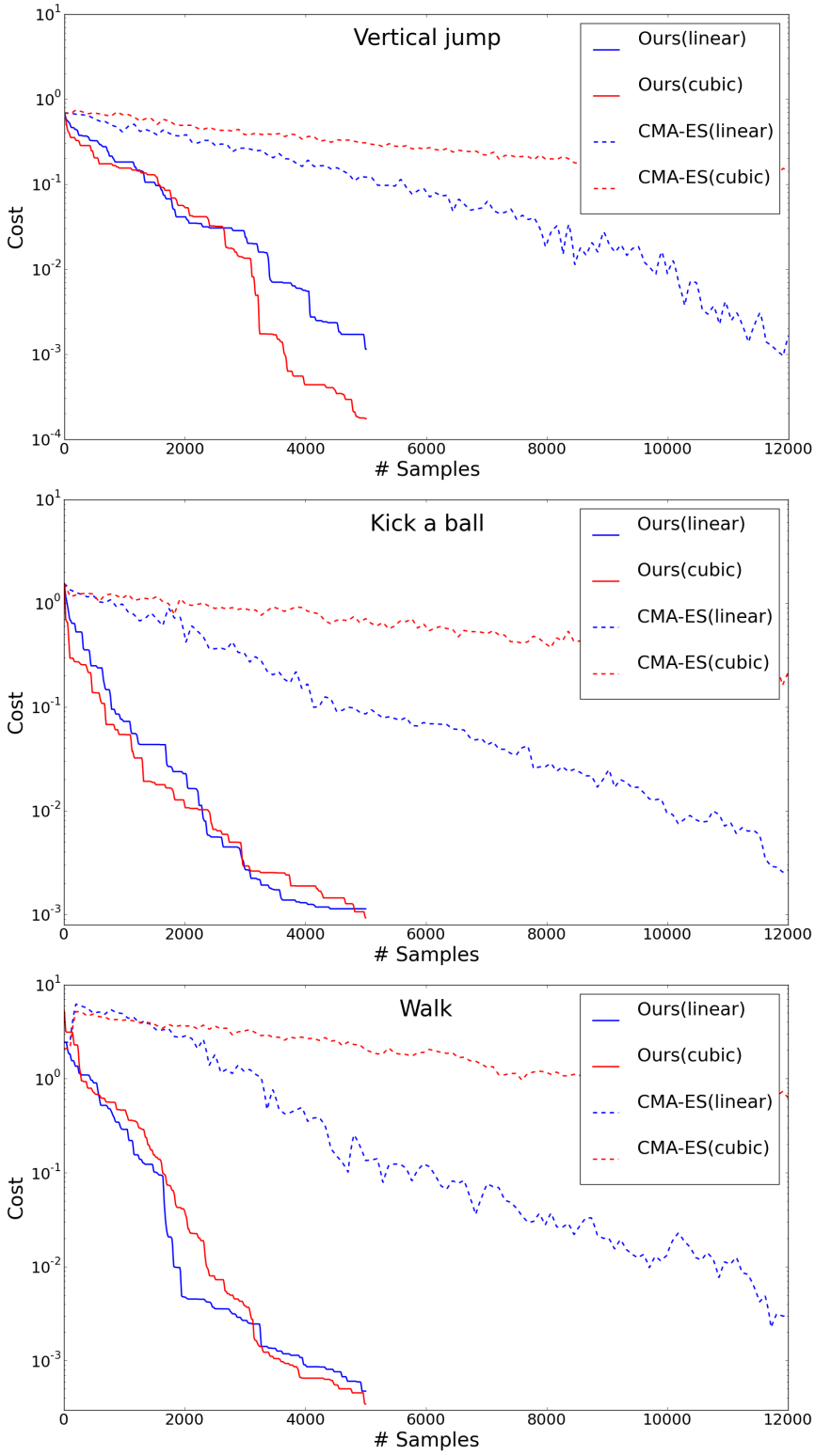


Figure 32: Comparison on three parameterized control problems. The cost (Equation (38)) is computed by averaging seven optimization trials. In all problems, our algorithm converges faster than CMA-ES, especially when the parameterized skill function is of cubic form.

7.4.1.2 Kick a Ball

Inspired by the work on quadrupeds playing soccer [36], the goal of this example is to learn a parameterized skill to kick a ball such that it travels a desired distance ranging from $0.3m$ to $0.6m$. Although in previous work a quadruped was allowed to fall after kicking a ball, we required our biped humanoid to maintain an upright balanced posture throughout the entire motion. The objective function is defined as follows:

$$f_{kick}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + \|\mathbf{h}_{ball}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{\mathbf{h}}^0_{ball} + w\hat{\mathbf{h}}^1_{ball})\|^2 \quad (46)$$

where $\mathbf{h}_{ball}(\mathcal{S}(\mathbf{x}))$ is the final position of the ball, $\hat{\mathbf{h}}^0_{ball}$ is $[0, 0.03, 0.3]^T$, and $\hat{\mathbf{h}}^1_{ball}$ is $[0, 0.03, 0.6]^T$. We broke the motion into three phases: leg lifting, backward leg swing, and forward leg swing, with six parameters for policy π_{kick}: the target hip angle during leg lifting, the target hip and knee angles during backward leg swing, and the target hip, knee, and ankle angles during forward leg swing.

7.4.1.3 Walk

The goal is to learn a locomotion controller such that the humanoid can walk at different speeds ranging from $6.7cm/s$ to $13.3cm/s$. We measured the walking speed over three seconds of simulation. The objective function is defined as

$$f_{walk}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + \|\mathbf{h}_{com}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{\mathbf{h}}^0_{com} + w\hat{\mathbf{h}}^1_{com})\|^2 \quad (47)$$

where $\mathbf{h}_{com}(\mathcal{S}(\mathbf{x}))$ is the center of mass position at the final frame, $\hat{\mathbf{h}}^0_{com}$ is $[0, 0.19, 0.15]^T$, and $\hat{\mathbf{h}}^1_{com}$ is $[0, 0.19, 0.40]^T$. One step of walk (half gait cycle) consists of three phases: take-off (foot leaving the ground) swing (leg swinging forward), and contact (the other foot touching the ground). We defined seven parameters for policy π_{walk} including the duration of the swing phases, the target hip, knee, and ankle angles for the take-off phase, and the target hip, knee, and ankle angles for the swing phase. We started from a manually designed policy which produces successful walk at $8.5cm/s$. Based

on the parameter values of this manual controller, we set the bounds for the policy parameters in the optimization.

7.4.1.4 Results

We compared our algorithm and CMA-ES with the linear parameterized skill function, as well as the cubic one. Our algorithm outperforms CMA-ES in all three problems (Figure 32). When solving linear parameterized skill functions, CMA-ES required approximately 2.5 times more samples than our algorithm. When solving cubic parameterized skill functions, our algorithm yielded a better solution with almost no slowdown in convergence. CMA-ES, on the other hand, became extremely slow and could not find any good solution when terminated at maximal iterations (12000). Comparing the difference in linear and cubic parameterized skill functions on our algorithm, an interesting observation is that, among three motor skills, cubic parameterization results in significant improvement only for the vertical jump problem. We conjecture the reason being that the intended force exerted to the ground during the thrusting phase has a highly nonlinear relationship with the resulting height of the jump. A cubic parameterized skill function captures this nonlinearity better than the linear one.

In addition, we investigated the impact of task discretization on the convergence rate. Specifically, we solved the vertical jump problem using different numbers of discrete tasks, $M = 6$ and $M = 21$, and compared the results in Figure 33. Although finer discretization ($M = 21$) should result in a better solution, it slows down the convergence because more samples evaluations are required. Since CMA-ES directly optimizes $\hat{f}(\boldsymbol{\phi})$, it becomes three to four times slower when $M = 21$. On the other hand, Figure 33 shows that our algorithm does not suffer from slower convergence rate when the number of discrete tasks increases.

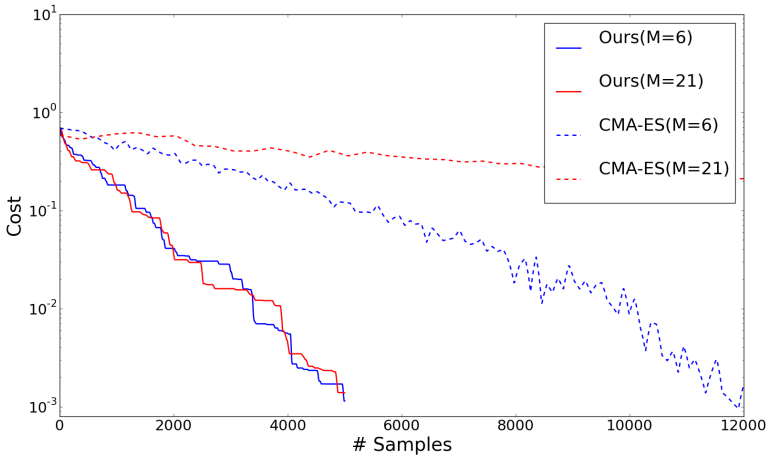


Figure 33: The impact of task discretization on convergence. More discrete tasks slow down the convergence of CMA-ES significantly, while it has negligible impact on our algorithm.

Table 11: Results on Parameterized CEC'15 Problems

Function	Dim	Ours(evals)	CMA-ES(evals)	Ratio
Sphere	5	6309.7	7151.1	0.88
Sphere	10	9460.3	13404.9	0.71
Sphere	20	16395.4	23539.7	0.70
Bent-Cigar	5	1402.6	2666.6	0.53
Bent-Cigar	10	3092.0	5724.9	0.54
Bent-Cigar	20	5824.3	11196.9	0.52
Function	Dim	Ours(cost)	CMA-ES(cost)	Ratio
Weierstrass	5	0.00093	0.03549	0.026
Weierstrass	10	0.00105	0.15398	0.007
Schefel	5	0.00444	0.07025	0.063
Schefel	7	0.00848	0.25131	0.034

7.4.2 Parameterized CEC'15 Problems

We tested the performances of our algorithm and the baseline algorithm on four parameterized problems from the benchmark CEC'15 [11] which was designed for testing evolutionary optimization algorithms. We selected two unimodal problems (Sphere and Bent-Cigar) and two multimodal problems (Weierstrass and Schwefel) from the benchmark set. However, the objective functions in CEC'15 were designed for testing standard single-task optimizations rather than for parameterized optimization problems. Therefore, we took the original objective function $f(\mathbf{x})$ and parameterized it by shifting, rotating, and scaling it with the parameter w:

$$f(\mathbf{x}; w) = s_w f(\mathbf{R}_w(\mathbf{x} - \mathbf{t}_w)) \tag{48}$$

where $\mathbf{t}_w, \mathbf{R}_w, s_w$ are linear functions of w and represent the shift, rotation, and scale parameters for the task w. The parameterized skill function is assumed linear in w for all four problems.

The results are shown in Table 11. For unimodal functions, we compared the average number of samples required to reach the convergence threshold ($= 0.001$). Our algorithm requires $52\% - 88\%$ samples comparing to CMA-ES on Sphere and Bent-Cigar problems. For multimodal problems, comparing the number of samples is not informative because one algorithm might use fewer samples but returns a very bad local minimum, while the other spends more samples to find a better local minimum or even the global one. Therefore, we compared the cost of the solution found by each algorithm instead of the number of samples used. For Weierstrass and Schwefel, the cost of solution for our algorithm is $0.7\% - 6.3\%$ of that of CMA-ES. From the four problems we evaluated, the performance gain increases as the dimension of the problem grows, although further investigation with more problem sets is needed to verify this trend.

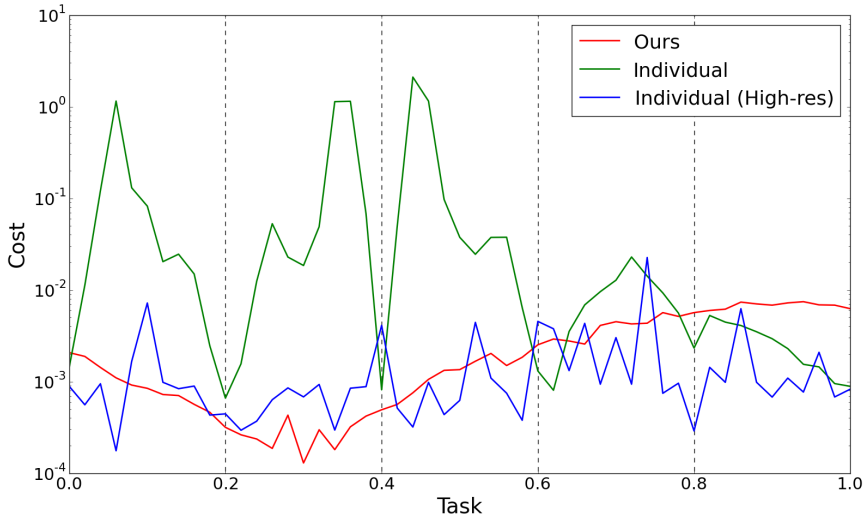


Figure 34: Comparison between our algorithm and the individual learning approach. The quality of the low-resolution policy (shown in green) is comparable with the high-resolution one (shown in blue) for those six tasks used for training (dotted vertical lines). However, for those tasks corresponding to interpolated policy parameters, there is a significant discrepancy between the quality of low-resolution and high-resolution policies. In contrast, our policy (shown in red) learned with only six tasks (M = 6) is comparable to the high-resolution one.

7.4.3 Comparison with Individual Learning Approach

Alternatively, a parameterized policy can be trained by defining a set of tasks, learning a policy for each task separately, and interpolating the policies using a regression model. We refer this method as *individual learning approach*. We solved the vertical jump problem using individual learning approach with two different resolutions of task discretization. In the low-resolution setting, we individually learned six tasks evenly across the entire task range (i.e. $M = 6$) and applied linear regression on the learned policy parameters. We repeated the same process in the high-resolution setting except that this time we individually learned 51 tasks (i.e. $M = 51$). The quality of the low-resolution policy is comparable with the high-resolution one for those six tasks used for training (dotted vertical lines in Figure 34). However, for those tasks corresponding to interpolated policy parameters, there is a significant discrepancy between the quality of low-resolution and high-resolution policies. In

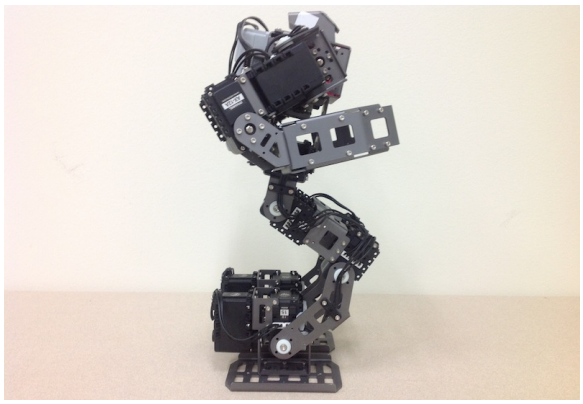


Figure 35: BioloidGP hardware.

contrast, our policy learned with only six tasks ($M = 6$) is comparable to the high-resolution one.

7.4.4 Hardware experiment

Our evaluation also includes deploying the learned walking policy on the hardware (Figure 35). The only difference in the parameterized policy for the real robot is that we decreased the bounds of the policy parameters during the optimization. This reduction results in more stable walk but decreases the maximal target speed from $13.3cm/s$ to $10.0cm/s$.

7.5 Conclusion

We presented a new evolutionary optimization algorithm for learning parameterized dynamic motor skills. Instead of individually acquiring optimal policies for each task, our algorithm simultaneously learns the policies for the entire range of tasks. The key insight of our algorithm is to sample in the space of policy parameters rather than directly sample in the high-dimensional space of parameterized skill function parameters. Since the solution in a parameterized problem is a curve segment rather than a point, our approach maintains a parameterized probability distribution along the mean segment and evolves it using selected elite samples. We demonstrated that our algorithm shows faster convergence when comparing to the baseline algorithm,

CMA-ES, especially when using a cubic parameterized skill function.

Although our algorithm optimizes parameterized tasks automatically, it is the user's responsibility to set a feasible task range achievable by the given parameterization of the control policy. If the range is too wide, the optimization will not converge to a good solution due to the intrinsic limitations of the space of policy parameters.

For future consideration, we plan on extending the task interpolation parameters into higher dimensions, which may afford greater flexibility of the resultant motor skills considerably. For example, currently we parameterize the jump controller to act in the vertical direction only, which obviates its use in general navigation applications. A consequence of increasing the dimension of the task parameter is that it will require the mean function be extended to the hyperplane in this new parameter space instead of a segment.

CHAPTER VIII

MODEL-BASED LEARNING FOR VIRTUAL AND REAL CHARACTERS

Conducting hardware experiment is often expensive in various aspects such as potential damage to the robot and the number of people required to operate the robot safely. Computer simulation is used in place of hardware in such cases, but it suffers from so-called simulation bias in which policies tuned in simulation do not work on hardware due to differences in the two systems. Model-free methods such as Q-Learning, on the other hand, do not require a model and therefore can avoid this issue. However, these methods typically require a large number of experiments, which may not be realistic for some tasks such as humanoid robot balancing and locomotion. This paper presents an iterative approach for learning hardware models and optimizing policies with as few hardware experiments as possible. Instead of learning the model from scratch, our method learns the difference between a simulation model and hardware. We then optimize the policy based on the learned model in simulation. The iterative approach allows us to collect wider range of data for model refinement while improving the policy.

8.1 Motivation

Conducting hardware experiments is a cumbersome task especially with large, complex and unstable robots such as full-size humanoid robots. They may require multiple people to operate to ensure safety of both operators and the robot; control failures can cause major damage; and even a minor damage is difficult to troubleshoot due to complexity.

For this reason, simulation is often used to replace hardware experiments. Unfortunately, it is difficult to obtain accurate simulation models, and therefore it suffers from so-called simulation bias [50] in which policies tuned in simulation cannot realize the same task with the hardware system due to differences in the two systems.

This paper presents an iterative approach for model learning and policy optimization using as few experiments as possible. Instead of learning the hardware model from scratch, our method reduces the number of experiments by only learning the difference from a simulation model, which provides estimations for unobserved states. The policy is then optimized through simulations using the learned model. We repeat this process iteratively so that we can refine the model because the improved policy is more likely to realize wider range of motions.

The assumption is that three things are essential to policy learning for complex robots:

- Learning only the difference from a model is essential to reduce the number of hardware experiments. The model can also be used for optimizing the initial policy.
- Iterative process is important for inherently unstable robots because we cannot collect enough data using a policy trained only in simulation.
- The learned model should be stocastic so that it can model sensor and actuator noises.

Our target task in this paper is balancing of bipedal robot on a bongoboard. To prove the concept, and to better control the noise conditions, we shall use two simulation models instead of a simulation model and a hardware system. One of the models is derived by Lagrangian dynamics assuming perfect contact conditions, while the other model is based on a 2D physics simulation engine with a more realistic

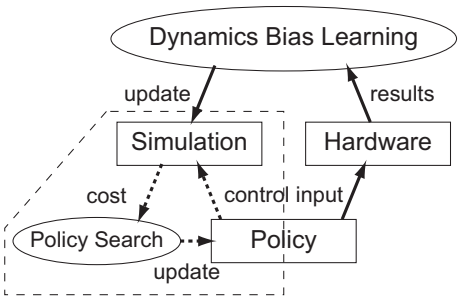


Figure 36: Framework of our approach.

contact model. These models are different enough that a policy optimized for the former cannot stabilize the latter.

The rest of the paper is organized as follows. Section 8.2 gives an overview of our framework, followed by more details on the model learning in Section 8.3 and policy optimization in Section 8.4. Section 8.5 presents simulation results and analysis. We finally conclude the paper in Section 8.6.

8.2 Overview

We developed an iterative reinforcement learning process to alternatley refine the model and policy. Figure 36 illustrates the approach.

The three main components are simulation, hardware, and policy. *Simulation* is based on a model of the robot hardware, and cheap to run. *Hardware* is the real robot and therefore more expensive to run. Both simulation model and robot hardware are controlled by control inputs computed by the *policy*.

The framework includes two iteration loops that run with different cycles. The outer loop (solid arrows) is the *dynamics bias learning* process that uses the experimental data from hardware to train the simulation model. The inner loop (dashed arrows) is the *policy search* process that uses the simulation model to optimize the policy based on a given cost function.

Our framework adapts some of the ideas used in prior work. Similarly to [48], we

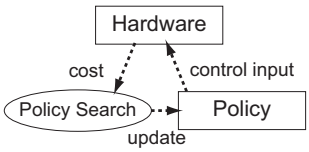


Figure 37: Direct policy search.

use Gaussian Process to model the difference between a dynamics model and actual robot dynamics. On the other hand, we also adopt the iterative learning scheme as in [6] because the performance of the initial controller is usually not good enough to learn accurate dynamics model. We also chose to directly optimize the policy parameters instead of learning the value function, as in [21].

We compare our framework with conventional direct policy search represented in Figure 37. This approach only has the policy search loop that uses the hardware directly to obtain the control cost for policy search. It usually requires a large number of hardware trials, which is unrealistic for our target robots and tasks.

The goal of this work is to reduce the number of dynamics bias learning loops that involve hardware experiments. On the other hand, we can easily run many policy search loops because we only have to run simulations.

8.3 Learning the Dynamics Model

8.3.1 Dynamics Bias Formulation

A general form of dynamics of a system with n states and m inputs can be written as

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{49}$$

where

$$\begin{aligned} \boldsymbol{x} \in \Re^n &: \text{robot state} \\ \boldsymbol{u} \in \Re^m &: \text{input} \\ \boldsymbol{f} \colon \Re^n \times \Re^m \to \Re^n &: \text{system dynamics function.} \end{aligned}$$

The goal of learning is to obtain $\boldsymbol{f}$ such that the model can accurately predict the system's behavior. In this paper, we employ one of the non-parametric models, Gaussian Process (GP) model. Learning $\boldsymbol{f}$ without prior knowledge, however, is expected to require a large amount of data to accurately model the system dynamics.

For many robots, we can obtain an approximate dynamics model by using, for example, Lagrangian dynamics. We denote such model by $\boldsymbol{f}'$. Instead of learning $\boldsymbol{f}$ that requires a large amount of data, our idea is to learn the difference between $\boldsymbol{f}'$ and the real dynamics:

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) + \boldsymbol{g}_D\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{50}$$

where $\boldsymbol{g}_D \colon \Re^n \times \Re^m \to \Re^n$ is the difference model to be learned and D represents the set of data used for learning the model. In this paper, we call $\boldsymbol{g}_D$ as dynamics bias.

Our expectation is that $\boldsymbol{f}'$ is a good approximation of the system dynamics, and therefore learning $\boldsymbol{g}_D$ requires far smaller data set than learning $\boldsymbol{f}$ from scratch.

8.3.2 Gaussian Process

Gaussian Process (GP) [78] is a stochastic model that represents the relationship between r inputs $\tilde{\boldsymbol{x}} \in \Re^r$ and a scalar output y. For the covariance function, we use the sum of a squared exponential and noise functions:

$$k\left(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}'\right) = \alpha^2 \exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)^T \boldsymbol{\Lambda}^{-1}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)\right) + \delta_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{x}}'}\sigma^2 \tag{51}$$

where α^2 is the variance of the latent function, σ^2 is the noise variance, and $\boldsymbol{\Lambda}^{-1}$ is a positive-definite matrix. Assuming that $\boldsymbol{\Lambda}^{-1}$ is a diagonal matrix whose elements are $\{l_1, l_2, \ldots, l_r\}$, the set of parameters $\boldsymbol{\theta} = (l_1, l_2, \ldots, l_r, \alpha^2, \sigma^2)$ is called hyper-parameters.

With N pairs of training inputs $\tilde{\boldsymbol{x}}_i$ and outputs $\boldsymbol{y} = [y_1 y_2 \ldots y_N]^T$, we can predict the output for a new input $\tilde{\boldsymbol{x}}^*$ by

$$y^* = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y} \tag{52}$$

with variance

$$\sigma^2 = k(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*) - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_* \tag{53}$$

where $\boldsymbol{K} = \{k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)\} \in \Re^{N \times N}$ and $\boldsymbol{k}_* = \{k(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}_i)\} \in \Re^N$.

The hyper-parameters are normally optimized to maximize the marginal likelihood of producing the training data. In our setting, however, optimizing hyper-parameters often results in over-fitting due to the small number of training data. We therefore manually adjust the hyper-parameters by looking at the policy optimization results. Once we determinte the hyper-parameters, we apply the same hyper-parameters for all testing scenarios.

8.3.3 Learning

We collect the input and output data from hardware experiments to train the dynamics bias model. For multiple-output systems, we use one GP for each dimension and train each GP independently using the outputs obtained from the same set of inputs.

The inputs to the GP models are the current state and input, $\tilde{\boldsymbol{x}}_t = \left(\boldsymbol{x}_{t-1}^T \ \boldsymbol{u}_{t-1}^T\right)^T$, while the outputs are the difference between the measured state and the prediction of the simulation model:

$$\boldsymbol{\Delta}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1} - \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right). \tag{54}$$

We collect a set of input and output pairs from hardware experiments.

The computational cost for learning increases rapidly as the training data increases. We therefore remove some of the samples from learning data set. First, we downsample the data because similar states do not improve model accuracy. We then remove the samples where the robot and board are no longer balancing on the wheel. Next, we discard the samples whose states are too far away from the static equilibrium state or too difficult to recover balance since designing a controller in such areas of the state space does not make much sense.

Finally, we discard the frames that are far from the prediction by the simulation model in order to remove outliers that may happen due to sensor erros in hardware experiments.

To summarize, samples with the following properties are not included in the training data:

1. The board touches the ground.
2. The board and wheel are detached.
3. The distance from the static equilibrium state is larger than a threshold.
4. The global angle of the robot body exceeds a threshold.
5. The global angle of the board exceeds a threshold.
6. The norm of the velocity exceeds a threshold.
7. The distance from the state predicted by the Lagrangian model is larger than a threshold.

8.3.4 Prediction

In policy search, we use the dynamics bias model to predict the next state $\boldsymbol{x}_t$ given the current state $\boldsymbol{x}_{t-1}$ and input $\boldsymbol{u}_{t-1}$. The GP model predicts the mean $\bar{\boldsymbol{\Delta}}_t$ and variance $\boldsymbol{\sigma}_t$ of the output, and the mean value is commonly used as the prediction. A problem with this method is that the prediction is not accurate if the input is far from any of the training data, especially when the traning data is sparse as in our case. Here, we take advantage of the system dynamics model $\boldsymbol{f}'$ by weighing the prediction of the GP such that we rely on the model as the prediction variance becomes larger, i.e.,

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) + \exp\left(-d|\boldsymbol{\sigma}^2|^2\right) \bar{\boldsymbol{\Delta}}_t \tag{55}$$

where $d > 0$ is a user-defined coefficient. If $\left(\boldsymbol{x}_{t-1}^T \ \boldsymbol{u}_{t-1}^T\right)^T$ is far away from any learning data, then the last term of (55) is nearly zero, meaning that we mostly use the prediction by the model.

8.4 Data-Efficient Reinforcement Learning

Algorithm 5 summarizes our framework. The algorithm starts from an empty learning data set $D = \emptyset$ and the assumption that the simulation model is accurate, i.e., $\boldsymbol{g} = 0$. At each iteration, we first search for an optimal policy using the simulation model $\boldsymbol{f}' + \boldsymbol{g}$. If the optimal policy does not give satisfactory results with the simulation model, we clear the model and restart from scratch. Otherwise, we evaluate the policy by a few hardware experiments to obtain the maximum cost as well as a new data set D_i for learning. If the policy successfully achieves the control objective on hardware, we terminate the iteration. Otherwise, we append D_i to the existing data set and re-learn the dynamics bias model $\boldsymbol{g}$ and repeat the same process until the maximum number of iterations is reached.

The cost function for policy optimization is

$$Z = c(T - t_{fail}) + \max_{1 \le t \le T} \boldsymbol{x}_t^T \boldsymbol{R} \boldsymbol{x}_t + \sum_{t=0}^{T} \boldsymbol{u}_t^T \boldsymbol{Q} \boldsymbol{u}_t \tag{56}$$

where c is a user-defined positive constant, T is the number of simulation frames, t_{fail} is the frame at which the simulation failed, and $\boldsymbol{R} \in \Re^{n \times n}, \boldsymbol{Q} \in \Re^{m \times m} \ge 0$ are user-defined weight matrices. We set $c = Z_{max}$ to make sure that the cost function value always exceed Z_{max} if a policy fails to keep the robot balanced for T frames. The first term penalizes policies that cannot balance the model for at least T frames. To determine failure, we use the criteria 1)–6) described in Section 8.3.3. The second term tries to minimze the maximum distance from the static equilibrium state. The third term considers the total energy consumption for control.

Any numerical optimization algorithm can be used for optimizing the policy p using the simulation model. We have found that the DIRECT algorithm [43] works

Algorithm 5: Data-efficient reinforcement learning

Require: nominal model $\boldsymbol{f}$
initialize $D = \emptyset$ and $\boldsymbol{g} = 0$
$i \leftarrow 0$
while $i < N_{out}$ **do**
  $p \leftarrow$ policy optimized for $\boldsymbol{g}$
  $Z_g \leftarrow$ evaluate policy p on $\boldsymbol{g}$
  if $Z_g > Z_{max}$ **then**
    initialize the simulation model: $D = \emptyset$ and $\boldsymbol{g} = 0$
  end if
  $Z_r, D_i \leftarrow$ evaluate p with hardware experiments
  if $Z_r < Z_{max}$ **then**
    break
  end if
  $D \leftarrow D \cup D_i$
  $\boldsymbol{g} \leftarrow \boldsymbol{g}_D$
  $i \leftarrow i + 1$
end while

best for our problem. Theoretically, the DIRECT algorithm is capable of finding the globally optimal solution relatively quickly. Because our optimization problem has many local minima, we selected the DIRECT algorithm instead of CMA-ES, which has been commonly used in this dissertation. We terminate the algorithm when the relative change in the cost function value in an optimization step is under a threshold ϵ, or the number of cost function evaluations exceeds a threshold N_{in}. DIRECT also requires the upper and lower bounds for each optimization parameters.

8.5 Results

While the final goal of this work is to optimize a policy for hardware systems, this paper focuses on proof of concept and uses two different simulation models in place of a simulation model and hardware. Using a well-controlled simulation environment also gives us the opportunity to explore different noise types and levels.

8.5.1 Bongoboard Balancing

The task we consider is balancing on bongoboard of a simple legged robot shown in Figure 38(a). Specifically, we apply the output-feedback controller developed by Nagarajan and Yamane [71] and attempt to optimize the gains through model learning and policy search. The state of the system is $\boldsymbol{x} = (\alpha_w \ \alpha_b \ \theta_1^r \ \dot{\alpha}_w \ \dot{\alpha}_b \ \dot{\theta}_1^r)^T$ (see Figure 38(a)), and the outputs we use for feedback control are $\boldsymbol{z} = (x_p \ \dot{x}_p \ \theta_1^r \ \dot{\theta}_1^r \ \alpha_f)^T$ as indicated in Figure 38(b).

The system has three degrees of freedom, and the only input is the ankle torque. Therefore the number of states is $n = 6$ and the number of inputs to the model is $m = 1$. Then the number of inputs to the GP becomes $r = n + m = 7$.

The output-feedback controller takes the five outputs of the models and compute the ankle torque by

$$u = \boldsymbol{H}\boldsymbol{z} \tag{57}$$

where $\boldsymbol{H} = (h_1 \ h_2 \ \dots \ h_5)$ is the feedback gain matrix. The policy search process computes the optimal values for the five elements of the gain matrix. In our implementation, we optimize a different set of parameters $\hat{\boldsymbol{h}}$ that are mapped to the elements of $\boldsymbol{H}$ by

$$h_i = \begin{cases} \exp(\hat{h}_i) - 1 & \text{if } \hat{h}_i \geq 0 \\ -\exp(-\hat{h}_i) + 1 & \text{if } \hat{h}_i < 0 \end{cases} \tag{58}$$

instead of directly optimizing h_i.

As mentioned above, we use two models in this paper, one corresponding to the *simulation* and the other corresponding to the *hardware* blocks for Figure 36.

The first model is derived by the Lagrangian dynamics formulation as described in [71]. This model assumes perfect contact condition, i.e. no slip or detachment of contacts between the floor and wheel, the wheel and board, as well as the board and robot feet.

The second model, used in lieu of hardware, is based on a 2D physical simulation

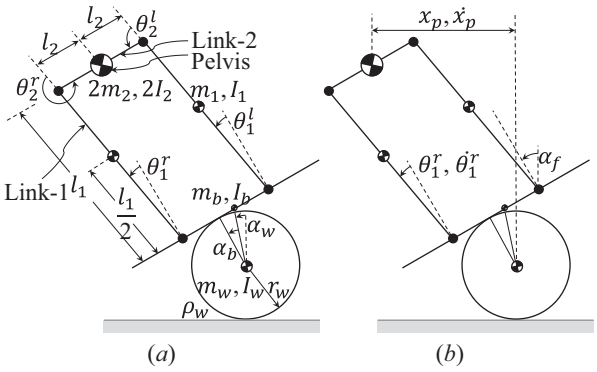


Figure 38: Robot balancing on a bongoboard.

engine called Box2D [3], which uses maximal (Eulerian) coordinate system and a spring-and-damper contact model. To make the simulation realistic, we add three types of noise:

- Torque noise: a zero-mean gaussian noise of variance σ_τ^2 is added to the robot's ankle joint torque.
- Joint angle noise: a zero-mean Gaussian noise of variance σ_p^2 is added to the wheel (α_w), board (α_b), and robot (θ_1^r) angles used for feedback control.
- Joint velocity sensor noise: a zero-mean Guassian noise of variance σ_v^2 is added to the wheel ($\dot{\alpha}_w$), board ($\dot{\alpha}_b$), and robot ($\dot{\theta}_1^r$) anglular velocities.

We also randomly choose the initial states in Box2D simulations for collecting training data for dynamics bias model learning because it is impossible to set exact initial states in hardware experiments.

Even though both are simulation, the results may be different due to different contact models and coordinate systems. In fact, a policy optimized for the Lagrangian model does not always balance the robot in the second model, which justifies the need for our framework even in this simple setup. Figure 39 show an example of using a policy optmized for the Lagrangian and Box2D models for both the Box2D model

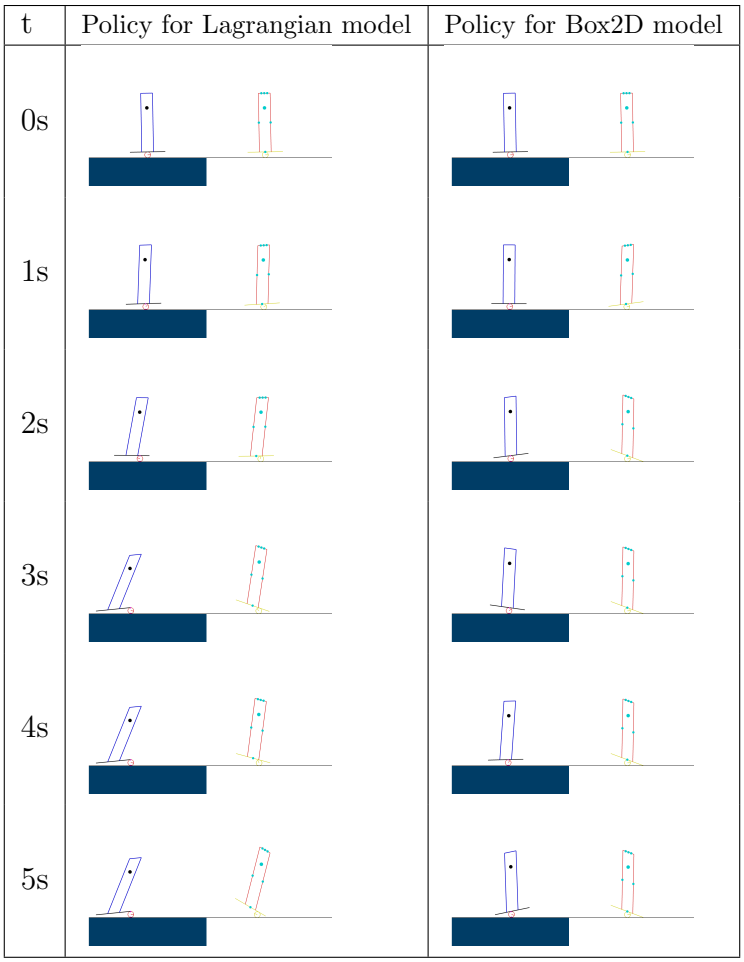


Figure 39: Simulation result of a policy optimized for the Lagrangian model (left column) and Box2D model (right column). In each snapshot, the left and right figures are the Box2D and Lagrangian model simulations respectively.

and the Lagrangian model. Both policies can successfully balance the model for which they are designed, but not the other model. With the policy designed for the Lagrangian model, the Box2D simulation fails before $t = 3$ sec when the board leaves the wheel. The Lagrangian model simulation with the policy designed for Box2D model fails when the board hits the ground before $t = 2$ sec.

Table 12 summarizes the parameters we used for the experiments.

Table 12: Parameters used for the experiments.

Dynamics Bias Model	
$\mathbf{\Lambda}^{-1}$	$diag(1, 1, 1, 1, 1, 1)$
α^2	1
σ^2	e^{-4}
d	1.0
Policy Optimization	
c	200
T	5000
Q	10^{-6}
$\boldsymbol{R}$	$diag(10, 10, 10, 0.1, 0.1, 0.1)$
N_{out}	10
Z_{max}	200
experiments per iteration	2
DIRECT parameters	
parameter bounds	$-10 \leq \hat{h}_i \leq 10$
N_{in}	1000
ϵ	10^{-6}
Simulation Setting	
maximum torque	100 Nm
timestep	0.001 s

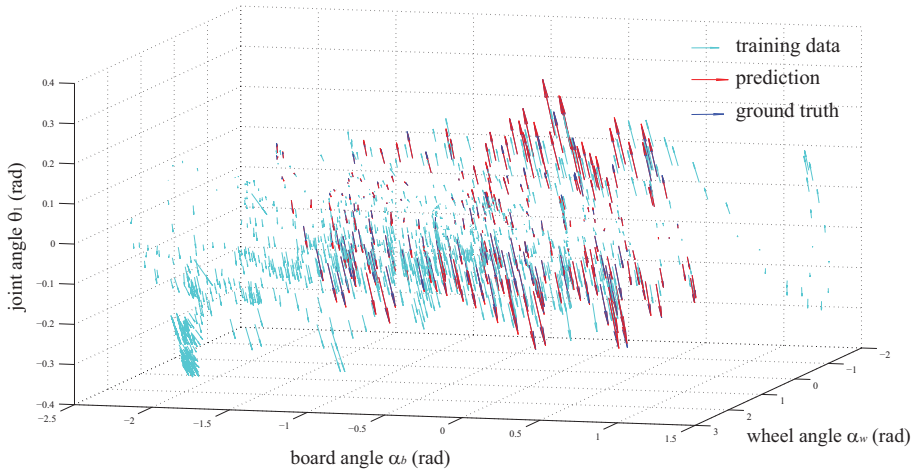


Figure 40: Velocity field of the learned dynamics model. Cyan: training data; red: prediction; blue: ground truth.

8.5.2 Dynamics Bias Learning

To ensure that the GP models can accurately predict the dynamics bias, we draw the vector field in the 3-dimensional subspace $(\alpha_w\ \alpha_b\ \theta_1^r)$ of the state spate. An example is shown in Figure 40, where the cyan arrows represent the training data and red and blue arrows depict the prediction and ground truth computed at different states. This example uses 571 samples obtained from four Box2D simulations. As shown here, the corresponding red and blue arrows match well, indicating that the GP models can accurately predict the dynamics bias.

8.5.3 Policy Search

We run our method for different noise levels and inertial parameter error magnitudes to investigate the relationship between the number of experiments required and the discrepancy between the model and hardware. Furthermore, to test the robustness against model errors, we conducted the same set of experiments when the inertial parameters of the Box2D model are 20% larger than those in the Lagrangian model.

Table 13 shows the average number of experiments required to obtain a policy that can successfully balance the robot in Box2D simulation for 5 seconds. For reference,

Table 13: Average number of experiments required at different noise levels and inertial parameter errors.

Torque σ_τ^2	Position σ_p^2	Velocity σ_v^2	# of experiments	
			no error	20% error
0	0	0	6.4	2.8
0.001	0	0	7.3	3.5
0.01	0	0	9.5	4.8
0.1	0	0	5.5	2.5
0.1	1.0×10^{-6}	1.0×10^{-3}	7.5	3.5
0.1	2.0×10^{-6}	2.0×10^{-3}	4.4	4.4
0.1	4.0×10^{-6}	4.0×10^{-3}	7.0	3.3
0.1	8.0×10^{-6}	8.0×10^{-3}	9.6	5.0
0.1	1.6×10^{-5}	1.6×10^{-2}	4.6	5.5
0.1	3.2×10^{-5}	3.2×10^{-2}	4.0	3.5
0.1	6.4×10^{-5}	6.4×10^{-2}	6.0	4.0
0.1	1.28×10^{-4}	1.28×10^{-1}	4.0	3.6

a 12-bit rotary encoder combined with a 50:1 gear measures the output joint angle at a resolution of 3.1×10^{-5} rad.

The results do not show any clear relationship between the noise level and the number of experiments required, which implies that larger noise or error does not necessarily require more experiments. Also, it is interesting that the numbers of experiments with inertial parameter errors are generally lower than their counterparts without errors. We suspect that the larger inertia lowered the natural frequency of the system, making the control easier in general.

Figure 41 shows three examples of cost function value change in Box2D simulation. The cost generally remains flat for a few iterations and then declines rapidly, probably when the dynamics bias model becomes accurate enough.

8.5.4 Policy Performance

Since the Box2D simulation includes noise, simulation results vary even if the robot starts from the same initial state and uses the same policy. We therefore compute the success rate from various initial states to evaluate the performance of a policy.

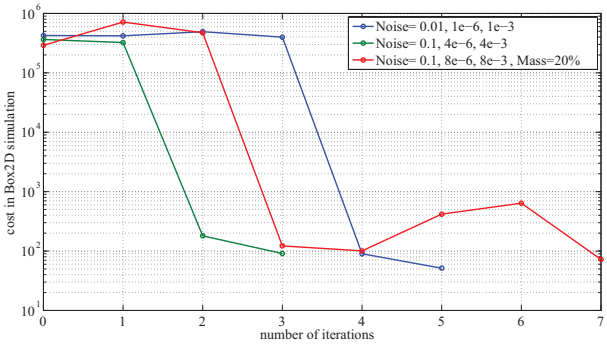


Figure 41: Change of cost function value in Box2D simulations over iterations.

Figure 42 depicts the balancing success rates starting from various wheel and board angles, using a policy optimized with Box2D simulation without noise (a) and with noise (b). This result clearly shows that the policy optimized in noisy environment can successfully balance the robot from a wider range of initial states under noisy actuator and sensors.

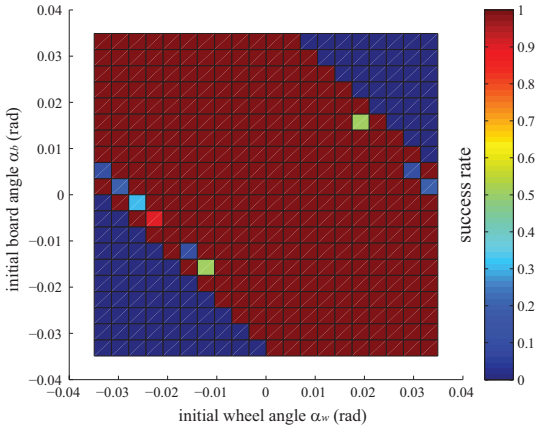
8.6 Conclusion and Future Work

This paper presented a framework for model learning and policy optimization of robots that are difficult to conduct experiments with. The key idea is to learn the difference between a model and hardware rather than learning the hardware dynamics from scratch. We also employ an iterative learning process to improve the model and policy This approach is particularly useful for tasks such as humanoid balancing and locomotion where a dynamics model is necessary to obtain a controller to collect the initial set of data.

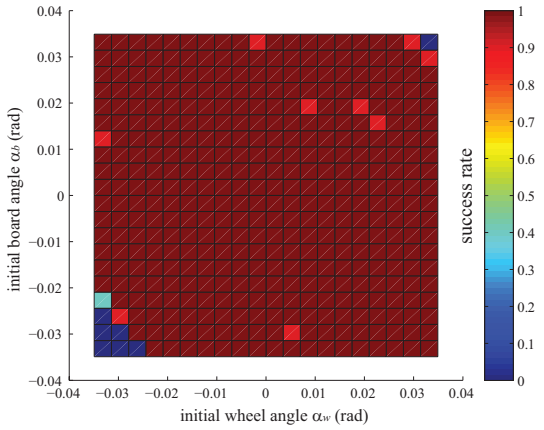
We conducted numerical experiments through bongoboard balancing task of a simple bipedal robot, and demonstrated that the framework can compute a policy that successfully completes the test task with only several hardware experiments. The policy obtained from noisy simulation proved to have higher balancing performance than the one obtained from clean simulation. The number of hardware experiments

did not show clear correlation with the noise level or magnitude of inertial parameter error.

Future work besides experiments with actual hardware system includes establishing a guideline for determining the hyper-parameters of GP and extension to more complex robot models. Another interesting direction would be to explore different representation of dynamics bias instead of the additive bias considered in this paper.



(a)



(b)

Figure 42: Balancing success rate in Box2D simulation with noise, starting from various initial wheel and board angles. (a) The policy has been optimized with Box2D simulation without noise. (b) The policy has been optimized with Box2D simulation with noise.

CHAPTER IX

CONCLUSION

Within this dissertation, we have presented a set of algorithms and frameworks for developing agile motor skills on virtual and real humanoids. In this final chapter, we will summarize the proposed techniques and preliminary results, and suggest future research directions that could provide the next steps toward more effective systems for real humanoids.

9.1 Summary

The dissertation started from designing controllers for the specific falling strategies on simulated characters and finished with general computational tools for developing various motor skills on real humanoids. Using the proposed techniques, various motor skills can be intuitively designed in virtual simulation and easily transferred to physical systems. Several optimization algorithms were also designed to develop more versatile and robust physics-based controllers for humanoids within a short amount of time. Further, various falling scenarios of virtual and real humanoids were extensively studied to ensure safety of humanoids and to achieve smooth transitions between motor skills.

Chapter 3 described a method to generate agile falling and landing motions of virtual characters in real-time via physical simulation without using motion capture data or pre-scripted animation. By designing novel controllers based on three landing principles informally developed in Parkour community, we can develop a robust controller that allows the character to fall from a wide range of heights and initial speeds, roll on the ground, and get back on its feet, without inducing large stress on joints at any moment.

Chapter 4 introduced a new planning algorithm to minimize the damage of humanoid falls by utilizing multiple contact points. Instead of selecting among a collection of manually designed control strategies, we proposed a novel algorithm which plans for appropriate falling motions to a wide variety of falls. Our algorithm covers various falling strategies from a single step to recover from a gentle nudge, to a rolling motion to break a high-speed fall.

The iterative learning framework of Chapter 5 allows users to intuitively develop dynamic controllers for virtual characters only using high-level, human-readable instructions. We introduced control rigs that formulate an intermediate layer for facilitating mapping between high-level instructions and manipulating multiple low-level control variables. The control rigs are design for utilizing the human coach's knowledge and reducing the search space for control optimization.

The optimization problems formulated in Chapter 5 can be efficiently solved using a new sampling-based optimization method, Covariance Matrix Adaptation with Classification (CMA-C), explained in Chapter 6. Inspired by the human ability to learn from failure, CMA-C utilizes the failed simulation samples to approximate infeasible regions, resulting in a faster convergence than the standard CMA-ES.

Chapter 7 shows how to optimize a parameterized motor skill which is essential for autonomous robots operating in an unpredictable environment. Our algorithm achieves a faster convergence rate by evolving a parameterized probability distribution for the entire range of tasks.

The method described in Chapter 8 provides us a framework for reducing hardware experiments which is usually very time-consuming and costly. The goal of this method is to learn the difference between simulation and hardware systems, and optimize a control policy that works on the target system. Instead of learning the model from scratch, our method learns only the difference between virtual and real system and reduces numbers of hardware experiments.

9.2 Future work

This section presents a few interesting future directions for improving the proposed techniques in this dissertation.

9.2.1 Real-time controllers for non-planar falls

In Chapter 4, we presented a falling strategy that breaks a fall by utilizing multiple contact points to reduce damage to humanoids. However, the proposed strategy requires two major revisions in order to be deployed on real robots: achieving real-time performance and handling non-planar falls.

Real-time planning of falls. Planning of falling motions must be very efficient to provide enough time for executing planned motions. However, the current implementation of the algorithm takes a few seconds, which is far from real-time so that a controller can be deployed to hardware of robots. One possible option is to compute falling motions for various initial cases in the pre-processing stage. When a fall is predicted, we can simply select the optimal falling motion that is designed for the closest scenario to the current state, instead of computing it online.

Planning of non-planar falls. Although we previously focused on planar falls in the sagittal plane as a proof of concept for the multiple contact falling strategy, humanoid falls can accompany all rotations in pitch, yaw, and roll axes. An intuitive approach for handling non-planar cases is to use a more complex abstract model, such as an inertia-loaded pendulum, that can account inertia and momentum in all axes. Because the proposed algorithm is not confined to the specific choice of the abstract model, we can easily incorporate different models by augmenting state and action spaces. However, the discretization of spaces and computation of dynamic programming must be improved for efficiency because more dimensions would increase the computation time tremendously, which is so called a curse of dimensionality.

9.2.2 A more intuitive learning interface

In Chapter 5, we presented the learning framework for teaching virtual characters how to execute dynamic motor skills using only high-level human-readable instructions. However, the current design of the framework still requires a long and repetitive sequence of instructions for complex motions with a large number of target angles, which is not straightforward for novice users. Therefore, designing a more intuitive interface will help users to efficiently develop more complex motions.

Motion capture systems. Extending the framework with motion capture systems may reduce the number of instructions to describe the motion. With the motion capture system, a user can act out the motion instead of teaching the motion with several instructions. Note that this is different from simply reconstructing the capture motions because a user may not be agile enough to perform the athletic target motion such as a back-flip, or the captured motion cannot be executed by humanoids due to differences between a user and a humanoid including body dimensions or joint structures.

Hierarchical composition of controllers. Another technique to prevent the repetition of similar instructions is to formulate a hierarchical structure of controllers that further extends the current concept of "control rigs". Control rigs formulate an intermediate control layer that can manipulate multiple low level controllers. Similarly, the key idea is to adopt a tree structure to hierarchically decompose a complex motor skill to primitive movements. For instance, a drop-and-roll motion can be decomposed into a jump, a transition, and a roll that are previously trained, and it can be served as another component for an even more complex motion. This hierarchical structure allows users to describe a motion in higher level of instructions such as "jump further" or "roll faster". In addition, re-optimizing controllers in this representation can be

done much more efficiently than learning new controllers from scratch, by conducting the optimization in the reduced parameter space.

9.2.3 Dynamics bias learning for humanoids

We previously demonstrated a framework for learning dynamic bias for a simple legged robot in Chapter 8, but it is not fully validated on a full-scale humanoid. Because humanoid robots usually have much higher degrees of freedom than simple robots, a naïve approach will require a tremendous amount of data infeasible to be obtained. There are several possible approaches for resolving this issue. For instance, a high dimensional parameter space can be projected into a lower dimensional space using a simplified model, which is a common approach to decrease the number of dimensions [93]. Another possibility is to employ the concept of active learning. Instead of executing the current best policy, we can find the most informative policy that maximizes the amount of data we can get from a single hardware experiment [17]. In addition, we can use multiple simulators as the work of Cutler and his colleagues [15] that efficiently solves reinforcement problems in the discretized grid setting. Although it is required to extend the approach for continuous spaces, optimizing the policy with multiple simulators will give us a chance to obtain more robust controllers that work for a wider range of scenarios.

REFERENCES

[1] *BioloidGP, http://en.robotis.com/.*

[2] *Boston Dynamics, http://www.bostondynamics.com.*

[3] "Box2d — a 2d physics engine for games." http://box2d.org/.

[4] *Fukushima Daiichi nuclear disaster, https://en.wikipedia.org/wiki/Fukushima_Daiichi_nuclear_disaster.*

[5] *Zenpo Kaiten Ukemi, http://en.wikipedia.org/wiki/Uke_(martial_arts).*

[6] Abbeel, P., Quigley, M., and Ng, A., "Using inaccurate models in reinforcement learning," in *Proceedings of the 23rd International Conference on Machine Learning*, pp. 1–8, 2006.

[7] *Advanced Parkour Roll Techniques, http://youtu.be/bbs7wDqViY4*, 2011.

[8] Al Borno, M., de Lasa, M., and Hertzmann, A., "Trajectory optimization for full-body movements with complex contacts.," *IEEE Trans. on visualization and computer graphics*, 2013.

[9] Atkeson, C. and Schaal, S., "Robot learning from demonstration," in *International Conference on Machine Learning*, pp. 12–20, 1997.

[10] Bingham, J. T., Lee, J., Haksar, R. N., Ueda, J., and Liu, C. K., "Orienting in mid-air through configuration changes to achieve a rolling landing for reducing impact after a fall," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3610–3617, Sept. 2014.

[11] Chen, Q., Liu, B., Zhang, Q., and Liang, J., "Evaluation Criteria for CEC 2015 Special Session and Competition on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization," *CEC*, 2015.

[12] Coros, S., Beaudoin, P., and van de Panne, M., "Robust task-based control policies for physics-based characters," in *ACM Trans. Graph*, 2009.

[13] Coros, S., Beaudoin, P., and van de Panne, M., "Generalized biped walking control," *ACM Trans. Graph.*, vol. 29, pp. 130:1–130:9, July 2010.

[14] Coros, S., Karpathy, A., Jones, B., Reveret, L., and Van De Panne, M., "Locomotion skills for simulated quadrupeds," *ACM Transactions on Graphics (TOG)*, pp. 1–11, 2011.

[15] Cutler, M., Walsh, T. J., and How, J. P., "Reinforcement learning with multi-fidelity simulators," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3888–3895, 2014.

[16] da Silva, B. C., Baldassarre, G., Konidaris, G., and Barto, A., "Learning parameterized motor skills on a humanoid robot," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5239–5244, May 2014.

[17] da Silva, B. C., Konidaris, G., and Barto, A., "Active Learning of Parameterized Skills," *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, 2014.

[18] da Silva, B. C., Konidaris, G., and Barto, A. G., "Learning Parameterized Skills," *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[19] da Silva, M., Abe, Y., and Popović, J., "Interactive simulation of stylized human locomotion," in *ACM SIGGRAPH 2008 papers*, pp. 82:1–82:10, 2008.

[20] DART, *Dynamic Animation and Robotics Toolkit, http://dartsim.github.io/.*

[21] Deisenroth, M. and Rasmussen, C., "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning*, pp. 465–472, 2011.

[22] Edwardes, D., *The Parkour and Freerunning Handbook.* It Books, August 2009.

[23] Faloutsos, P., van de Panne, M., and Terzopoulos, D., "Composable controllers for physics-based character animation," in *SIGGRAPH*, pp. 251–260, Aug. 2001.

[24] Fang, A. C. and Pollard, N. S., "Efficient synthesis of physically valid human motion," *ACM Trans. on Graphics (SIGGRAPH)*, pp. 417–426, July 2003.

[25] Forte, D., Gams, A., Morimoto, J., and Ude, A., "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, pp. 1327–1339, Oct. 2012.

[26] Fujiwara, K., Kajita, S., Harada, K., Kaneko, K., Morisawa, M., Kanehiro, F., Nakaoka, S., and Hirukawa, H., "An optimal planning of falling motions of a humanoid robot," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 456–462, IEEE, 2007.

[27] Fujiwara, K., Kajita, S., Harada, K., Kaneko, K., Morisawa, M., Kanehiro, F., Nakaoka, S., and Hirukawa, H., "Towards an optimal falling motion for a humanoid robot," *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pp. 524–529, Dec. 2006.

[28] Fujiwara, K., Kanehiro, F., Kajita, S., Kaneko, K., Yokoi, K., and Hirukawa, H., "UKEMI: Falling motion control to minimize damage to biped humanoid robot," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, pp. 2521–2526, IEEE, 2002.

[29] FUJIWARA, K., KANEHIRO, F., KAJITA, S., and HIRUKAWA, H., "Safe knee landing of a human-size humanoid robot while falling forward," *Intelligent Robots and Systems, 2004*, pp. 503–508, 2004.

[30] Fujiwara, K., Kanehiro, F., Kajita, S., Yokoi, K., Saito, H., Harada, K., Kaneko, K., and Hirukawa, H., "The first human-size humanoid that can fall over safely and stand-up again," *IEEE-RSJ International Conference on Intelligent Robots and Systems*, no. October, pp. 1920–1926, 2003.

[31] Goswami, A., Yun, S.-k., Nagarajan, U., Lee, S.-H., Yin, K., and Kalyanakrishnan, S., "Direction-changing fall control of humanoid robots: theory and experiments," *Autonomous Robots*, vol. 36, no. 3, pp. 199–223, 2014.

[32] Ha, S. and Liu, C. K., "Iterative training of dynamic skills inspired by human coaching techniques," *ACM Transactions on Graphics*, vol. 33, 2014.

[33] Ha, S., Ye, Y., and Liu, C. K., "Falling and landing motion control for character animation," *ACM Trans. Graph*, vol. 31, no. 6, p. 155, 2012.

[34] Hansen, N. and Kern, S., "Evaluating the CMA evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature - PPSN VIII*, vol. 3242 of *LNCS*, pp. 282–291, 2004.

[35] Hansen, N., Müller, S., and Koumoutsakos, P., "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[36] Hausknecht, M. and Stone, P., "Learning powerful kicks on the aibo ers-7: The quest for a striker," *RoboCup 2010: Robot Soccer World Cup XIV*, no. June, 2010.

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

[38] Hohn, O. and Gerth, W., "Probabilistic Balance Monitoring for Bipedal Robots," *The International Journal of Robotics Research*, vol. 28, pp. 245–256, 2009.

[39] *How to Land a Jump in Parkour, http://www.wikihow.com/Land-a-Jump-in-Parkour*, 2011.

[40] Igel, C., Suttorp, T., and Hansen, N., "A Computational Efficient Covariance Matrix Update and a (1 + 1) -CMA for Evolution Strategies," *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO)*, pp. 453—-460, 2006.

[41] Ijspeert, A. J., Nakanishi, J., and Schaal, S., "Learning attractor landscapes for learning motor primitives," *Advances in neural information processing systems*, 2002.

[42] Jain, S. and Liu, C. K., "Controlling physics-based characters using soft contacts," *ACM Trans. Graph. (SIGGRAPH Asia)*, vol. 30, pp. 163:1–163:10, Dec. 2011.

[43] Jones, D., Perttunen, C., and Stuckman, B., "Lipscitzian optimization without the Lipschitz constant," *Journal of Optimization Theory*, vol. 79, no. 1, pp. 157–181, 1993.

[44] Kane, T. R. and Scher, M. P., "A dynamical explanation of the falling cat phenomenon," *Int J Solids structures*, no. 55, pp. 663–670, 1969.

[45] Karssen, J. G. D. and Wisse, M., "Fall detection in walking robots by multi-way principal component analysis," *Robotica*, vol. 27, 2008.

[46] Khalil, W. and Dombre, E., *Modeling, identification and control of robots*. London, U.K.: Hermès Penton, 2002.

[47] Kim, J., Kim, Y., and Lee, J., "A machine learning approach to falling detection and avoidance for biped robots," *SICE Annual Conference (SICE)*, pp. 562–567, 2011.

[48] Ko, J., Klein, D., Fox, D., and Haehnel, D., "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *IEEE International Conference on Robotics and Automation*, pp. 742–747, 2007.

[49] Kobayashi, K., Yoshikai, T., and Inaba, M., "Development of humanoid with distributed soft flesh and shock-resistive joint mechanism for self-protective behaviors in impact from falling down," *IEEE International Conference on Robotics and Biomimetics*, pp. 2390–2396, 2011.

[50] Kober, J. and Bagnell, J.A. amd Peters, J., "Reinforcement learning in robotics: A survey," *the International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[51] Kober, J. and Peters, J., "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems*, pp. 849–856, 2008.

[52] Kober, J., Tübingen, M. P. I., and Peters, J., "Reinforcement Learning to Adjust Robot Movements to New Situations," *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, pp. 2650–2655, 2010.

[53] Lee, S.-H. and Goswami, A., "Fall on Backpack: Damage Minimization of Humanoid Robots by Falling on Targeted Body Segments," *Journal of Computational and Nonlinear Dynamics*, vol. 8, 2012.

[54] Lee, Y., Kim, S., and Lee, J., "Data-driven biped control," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 29, July 2010.

[55] Libby, T., Moore, T. Y., Chang-Siu, E., Li, D., Cohen, D. J., Jusufi, A., and Full, R. J., "Tail-assisted pitch control in lizards, robots and dinosaurs," *Nature*, vol. advance online publication, January 2012.

[56] Liu, C. K. and Jain, S., "A short tutorial on multibody dynamics," Tech. Rep. GIT-GVU-15-01-1, Georgia Institute of Technology, School of Interactive Computing, 08 2012.

[57] Liu, C. K. and Popović, Z., "Synthesis of complex dynamic character motion from simple animations," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 21, pp. 408–416, July 2002.

[58] Liu, L., Yin, K., van de Panne, M., and Guo, B., "Terrain runner: control, parameterization, composition, and planning for highly dynamic motions," *ACM Trans. Graph*, vol. 31, no. 6, p. 154, 2012.

[59] Liu, L., Yin, K., van de Panne, M., Shao, T., and Xu, W., "Sampling-based contact-rich motion control," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 128, 2010.

[60] Macchietto, A., Zordan, V., and Shelton, C., "Momentum control for balance," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, p. 80, 2009.

[61] Majkowska, A. and Faloutsos, P., "Flipping with physics: motion editing for acrobatics," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Aire-la-Ville, Switzerland, Switzerland), pp. 35–44, 2007.

[62] Matsubara, T., Hyon, S.-H., and Morimoto, J., "Learning parametric dynamic movement primitives from multiple demonstrations.," *Neural networks*, vol. 24, pp. 493–500, June 2011.

[63] Missura, M., Wilken, T., and Behnke, S., "Designing effective humanoid soccer goalies," *RoboCup 2010: Robot Soccer World Cup XIV*, pp. 374—-385, 2011.

[64] Montgomery, R., "Gauge theory of the falling cat," in *Dynamics and Control of Mechanical Systems* (Enos, M. J., ed.), pp. 193–218, American Mathematical Society, 1993.

[65] Moore, A. and Atkeson, C., "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[66] Mordatch, I., de Lasa, M., and Hertzmann, A., "Robust physics-based locomotion using low-dimensional planning," *ACM Trans. Graph.*, vol. 29, pp. 71:1–71:8, July 2010.

[67] Morimoto, J. and Doya, K., "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[68] Morimoto, J., Atkeson, C. G., Endo, G., and Cheng, G., "Improving humanoid locomotive performance with learnt approximated dynamics via Gaussian processes for regression," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4234–4240, 2007.

[69] Muelling, K., Kober, J., and Peters, J., "Learning table tennis with a Mixture of Motor Primitives," *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 411–416, Dec. 2010.

[70] 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.

[71] Nagarajan, U. and Yamane, K., "Universal balancing controller for robust lateral stabilization of bipedal robots in dynamic, unstable environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 6698–6705, 2014.

[72] Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., and Kawato, M., "Learning from demonstration and adaptation of biped locomotion," *Robotics and Autonomous Systems*, vol. 47, pp. 79–91, 2004.

[73] Neumann, G., Daniel, C., Kupcsik, A., Deisenroth, M., and Peters, J., "Information-theoretic motor skill learning," *Proceedings of the AAAI Workshop on Intelligent Robotic Systems*, 2013.

[74] Ogata, K., Terada, K., and Kuniyoshi, Y., "Real-time selection and generation of fall damage reduction actions for humanoid robots," *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pp. 233–238, 2008.

[75] Ogata, K., Terada, K., and Kuniyoshi, Y., "Falling motion control for humanoid robots while walking," *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pp. 306–311, Nov. 2007.

[76] Peng, J. and Williams, R., "Incremental multi-step Q-Learning," *Machine Learning*, vol. 22, pp. 283–290, 1996.

[77] Quiroz, J., Louis, S., and Dascalu, S., "Interactive evolution of XUL user interfaces," *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, p. 2151, 2007.

[78] Rasmussen, C. and Kuss, M., "Gaussian Processes in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[79] Renner, R. and Behnke, S., "Instability Detection and Fall Avoidance for a Humanoid using Attitude Sensors and Reflexes," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2967–2973, 2006.

[80] Ros, I. G., Bassman, L. C., Badger, M. A., Pierson, A. N., and Biewener, A. A., "Pigeons steer like helicopters and generate down- and upstroke lift during low speed turns," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 108, no. 50, 2011.

[81] Ross, S. and Bagnell, J., "Agnostic system identification for model-based reinforcement learning," in *International Conference on Machine Learning*, 2012.

[82] RTQL8, *RTQL8, http:bitbucket.org/karenliu/rtql8*, 2012.

[83] Ruiz-del Solar, J., "Fall detection and management in biped humanoid robots," *Robotics and Automation (ICRA), IEEE International Conference on*, pp. 3323–3328, 2010.

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

[85] Safonova, A., Hodgins, J. K., and Pollard, N. S., "Synthesizing physically realistic human motion in low-dimensinal, behavior-specific spaces," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 23, no. 3, pp. 514–521, 2004.

[86] *ScienceTweets, http://sciencetweets.eu/photochemistry/archive/fullsize/cat-falling_fc51ab5347.jpg*, 2015.

[87] Shapiro, a., Pighin, F., and Faloutsos, P., "Hybrid control for interactive character animation," *Computer Graphics and Applications*, pp. 455–461, 2003.

[88] Sims, K., "Artificial evolution for computer graphics," *SIGGRAPH Comput. Graph.*, vol. 25, pp. 319–328, July 1991.

[89] Sok, K. W., Kim, M., and Lee, J., "Simulating biped behaviors from human motion data," *ACM Trans. Graph*, vol. 26, no. 3, 2007.

[90] Sok, K. W., Yamane, K., Lee, J., and Hodgins, J., "Editing dynamic human motions via momentum and force," *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2010.

[91] Sreevalsan-Nair, J., Verhoeven, M., Woodruff, D., Hotz, I., and Hamann, B., "Human-guided enhancement of a stochastic local search: Visualization and adjustment of 3d pheromone," *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, vol. 4638, pp. 182–186, 2007.

[92] Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O., "Learning Compact Parameterized Skills with a Single Regression," *Proc. IEEE-RAS International Conference on Humanoid RObots - HUMANOIDS*, 2013.

[93] Sugimoto, N. and Morimoto, J., "Trajectory-model-based reinforcement learning : Application to bimanual humanoid motor learning with a closed-chain constraint," *IEEE-RAS International Conference on Humanoid Robots*, 2013.

[94] Sulejmanpašić, A. and Popović, J., "Adaptation of performed ballistic motion," *ACM Trans. on Graphics*, vol. 24, no. 1, 2004.

[95] Sunada, C., Argaez, D., Dubowsky, S., and Mavroidis, C., "A coordinated jacobian transpose control for mobile multi-limbed robotic systems," in *ICRA*, pp. 1910–1915, 1994.

[96] Sutton, R., "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the 7th International Conference on Machine Learning*, pp. 216–224, 1990.

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

[98] Tangkaratt, V., Mori, S., Zhao, T., Morimoto, J., and Sugiyama, M., "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation.," *Neural networks*, vol. 57, Sept. 2014.

[99] Ude, A., Gams, A., Asfour, T., and Morimoto, J., "Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives," *IEEE Transactions on Robotics*, vol. 26, pp. 800–815, Oct. 2010.

[100] Wang, J. M., Fleet, D. J., and Hertzmann, A., "Optimizing walking controllers," *ACM Trans. Graph*, vol. 28, no. 5, 2009.

[101] Wang, J. M., Fleet, D. J., and Hertzmann, A., "Optimizing walking controllers for uncertain inputs and environments," *ACM Trans. Graph*, vol. 29, no. 4, 2010.

[102] 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, no. 4, p. 25, 2012.

[103] Wang, J., Whitman, E. C., and Stilman, M., "Whole-body trajectory optimization for humanoid falling," *American Control Conference (ACC), 2012*, pp. 4837—-4842, 2012.

[104] Waters, C. D. J., "Interactive Vehicle Routeing," *The Journal of the Operational Research Society*, no. 9, 1984.

[105] Wooten, W. L., *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. PhD thesis, Georgia Institute of Technology, 1998.

[106] Yamane, K., "Practical kinematic and dynamic calibration methods for force-controlled humanoid robots," in *Proceedings of IEEE-RAS International Conference on Humanoids Robots*, (Bled, Slovenia), p. (in press), October 2011.

[107] Ye, Y. and Liu, C. K., "Optimal feedback control for character animation using an abstract model," *ACM Trans. Graph*, vol. 29, no. 4, 2010.

[108] Yin, K., Loken, K., and van de Panne, M., "Simbicon: simple biped locomotion control," in *SIGGRAPH*, p. 105, 2007.

[109] Yun, S.-k. and Goswami, A., "Tripod Fall : Concept and Experiments of a Novel Approach to Humanoid Robot Fall Damage Reduction," *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2799–2805, 2014.

[110] Zhao, P. and van de Panne, M., "User interfaces for interactive control of physics-based 3d characters," *I3D: ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, 2005.

[111] Zordan, V., Riverside, U. C., Brown, D., and Columbia, B., "Control of Rotational Dynamics for Ground and Aerial Behavior," *Transactions on Visualization and Computer Graphics*, 2014.

[112] Zordan, V. B., Majkowska, A., Chiu, B., and Fast, M., "Dynamic response for motion capture animation," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 24, pp. 697–701, July 2005.

VITA

Sehoon ha was born in Geoje island, Korea in 1985. He graduated from Gyungnam Science High School in 2003. He attended Korea Advanced Institute of Science and Technology in Daejon, Korea, where he majored computer science. During his undergraduate years, Sehoon participated ACM International Collegiate Programming Contests and won several awards including 3rd in 2005 Asia Regional and 13th in 2006 World Final. In his senior year at KAIST, Sehoon performed research on reconstructing motion capture data in the Theory of Computation lab with Dr. Sungyong Shin. After two years of military service, he graduate from KAIST in 2009.

After visiting Georgia Institute of Technology as an exchange undergraduate student in 2009, he decided to pursuit his doctorate degree in computer graphics lab at Georgia Tech, under the supervision of Professor C. Karen Liu. In 2010, Sehoon visited University of Southern California in Los Angeles, California to collaborate with Professor Eva Kanso. In 2012, Sehoon worked at Adobe Research in Boston, Massachusetts and Seattle, Washington where he collaborated with Dr. Jovan Popović and Dr. Jim McCann. In 2014, Sehoon worked at Disney Research in Pittsburgh, Pennsylvania where he collaborated with Dr. Katsu Yamane. In the Fall of 2015, Sehoon completed all the requirements for the doctorate degree in computer science.