LEARNING DYNAMIC MOTOR SKILLS FOR 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 Computer Science

Georgia Institute of Technology
December 2015

LEARNING DYNAMIC MOTOR SKILLS FOR VIRTUAL AND REAL HUMANOIDS

Approved by:

Professor C. Karen Liu, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Jun Ueda
School of Mechanical Engineering
Georgia Institute of Technology

Professor Greg Turk
School of Computer Science
Georgia Institute of Technology

Professor Katsu Yamane
Disney Research Pittsburgh

Professor Jarek Rossignac
School of Computer Science
Georgia Institute of Technology

Date Approved: 14 August 2015

To myself,

Sehoon Ha,

the only person worthy of my company.

ACKNOWLEDGEMENTS

I want to "thank" my committee, without whose ridiculous demands, I would have graduated so, so, very much faster. Can I?

TABLE OF CONTENTS

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **viii**

LIST OF FIGURES . . . . . **ix**

SUMMARY . . . . . **xi**

I INTRODUCTION **1**

1.1 Falling Strategies for Humanoids 4

1.1.1 Falling and Landing Motion Control for Virtual Characters . 4

1.1.2 Multiple Contact Planning for Humanoid falls 5

1.2 Learning of Dynamic Controller for Characters 6

1.2.1 Iterative Design of Dynamic Controllers 6

1.2.2 Optimization with Failure Learning 7

1.2.3 Optimization for Parametrized Motor Skills 7

1.3 Model-based Learning for Virtual and Real Characters 7

1.4 Contributions 8

II RELATED WORK **10**

2.1 Physics-based simulation of dynamic motor skills 10

2.1.1 Physics-based animation 10

2.1.2 Physics-based animation of highly dynamic motions 11

2.2 Control of falling motions 12

2.2.1 Falling detection techniques 12

2.2.2 Falling damage reduction strategies 12

2.2.3 Falling strategies of animals 13

2.3 Interactive interfaces 14

2.3.1 Parameter selection interfaces 14

2.3.2 Human-in-the-loop interfaces 14

2.4 Policy search algorithms 15
2.4.1 Model-free policy search algorithms 15
2.4.2 Model-based policy search algorithms 15
2.4.3 Policy search algorithms for parametrized tasks 17

III FALLING AND LANDING MOTION CONTROL FOR VIRTUAL CHARACTERS 19

3.1 Motivation 20
3.2 Overview 22
3.3 Landing Strategy 23
3.4 Airborne Phase 26
3.5 Landing Phase 29
3.5.1 Impact Stage 30
3.5.2 Rolling Stage 32
3.5.3 Getting-Up Stage 33
3.6 Results 34
3.6.1 Evaluation 38
3.6.2 Limitations 39
3.7 Discussion 40

IV MULTIPLE CONTACT PLANNING FOR HUMANOID FALLS 42

4.1 Motivation 42
4.2 The Problem 44
4.3 The Algorithm 45
4.3.1 Abstract Model 45
4.3.2 Multiple Contacts 47
4.4 Experiments 53
4.4.1 Simulation Results 53
4.4.2 Hardware Results 56
4.4.3 Limitations 57
4.5 Conclusion 58

V MODEL-BASED LEARNING FOR VIRTUAL AND REAL CHARACTERS . . . 59

5.1 Motivation . . . 59

5.2 Overview . . . 61

5.3 Learning the Dynamics Model . . . 62

5.3.1 Dynamics Bias Formulation . . . 62

5.3.2 Gaussian Process . . . 63

5.3.3 Learning . . . 64

5.3.4 Prediction . . . 65

5.4 Data-Efficient Reinforcement Learning . . . 66

5.5 Results . . . 67

5.5.1 Bongoboard Balancing . . . 67

5.5.2 Dynamics Bias Learning . . . 71

5.5.3 Policy Search . . . 72

5.5.4 Policy Performance . . . 73

5.6 Conclusion and Future Work . . . 74

APPENDIX A — SOME ANCILLARY STUFF . . . 76

REFERENCES . . . 77

INDEX . . . 87

VITA . . . 88

LIST OF TABLES

1 Control parameters. 32

2 Initial conditions of the examples shown in the video (in order of appearance) 35

3 The initial conditions and the results of BioloidGP simulations. . . . 54

4 The initial conditions and the results of Atlas simulations. 56

5 Parameters used for the experiments. 70

6 Average number of experiments required at different noise levels and inertial parameter errors. 73

LIST OF FIGURES

1 A falling motion of Parkour. 5

2 Hardware of BioloidGP robot. 5

3 The proposed learning frame uses human readable instructions to teach motions. 6

4 Bongo Board balance toy. 7

5 A cat is able to right itself as it falls to land on its feet, irrespective of its initial orientation. 13

6 Difference between a real robot and its simulation model results different motions from same controllers. 16

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. 19

8 Three stages in the landing phase. 23

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. 24

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. 25

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. 26

12 Among 16 poses in $\mathbf{Q}$, pose 1, 2, 9, and 13 are frequently selected by the airborne controller 29

13 Landing phase controller. 30

14 Two-step impact stage for the feet-first strategy. 31

15 Hands-first landing motion. 34

16 Left: The character model used for most examples. Right: A character with a disproportionately large torso and short legs. 37

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. 38

18 Feet-first landing motion. 40

19 The abstract model consists of a telescopic inverted pendulum and a massless stopper. 45

20 Contact graphs 48

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. 53

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. 55

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). 57

24 Framework of our approach. 61

25 Direct policy search. 62

26 Robot balancing on a bongoboard. 69

27 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. 71

28 Velocity field of the learned dynamics model. Cyan: training data; red: prediction; blue: ground truth. 72

29 Change of cost function value in Box2D simulations over iterations. . 73

30 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. 75

SUMMARY

Summary goes here

CHAPTER I

INTRODUCTION

Over the last decades, humanoids become more available in academia, industry, and our daily life. They have played important roles as indefatigable workers in factories, surrogate rescue parties in disaster relief scenes, emotional friends in home [PEPPER], or virtual star sport players in computer games[EA]. And the recent development of software and hardware provides an opportunity to enrich their motion vocabularies with dynamic motor skills, which is essential for efficient and agile task achievements. One notable benefit of dynamic motions is that a humanoid can advance to desired locations as swift as possible using only its body, like a practitioner of free-running. Moreover, a humanoid can operate on the irregular terrain by overcoming obstacles with jumping and falling, which is often observed in disaster paces. Development of physics-based controllers for virtual and real humanoids are popular but yet challenging topics in many disciplines due to their large momentum and frequent changes of contacts. An in-depth understanding of mechanisms of dynamic motions and their controller would shed light on general motor control problems of humanoids.

Virtual characters and real robots are two main subjects of motor control problems in computer graphics and robotics, which have both common and different properties. To name a few common properties, control problems in both areas have non-linear objective function, under-actuated characters, high-dimensional control parameters. There are also different assumptions and limitations on sensors, actuators, contacts, control mechanisms, and so on. It is often possible to apply the existing principles and algorithms developed in one domain to the other and expedite the design process of the controller, if they are robust enough to handle different assumptions. For instance,

a virtual simulation of a robot is often used as a testbed for developing hardware compatible controllers due to the expensive cost and time-consuming trials [,,]. The techniques that are designed to control noisy hardware systems can be applied to virtual characters for creating more robust controllers. In this dissertation, I will develop control techniques for both virtual and real humanoids to demonstrate how the algorithms in two different systems can benefit each other.

Highly dynamic motor skills are difficult tasks for humanoids. Execution of dynamic motions accompanies abrupt accelerations and decelerations of momentum, frequent changes of contacts, and explosive usage of torques near limitations. Therefore, control becomes a very sensitive problem because small errors can quickly accumulate and generate a disastrous result to the humanoid, such as loosing it balance and hitting the ground severely. One of challenging motions is safe falling of a humanoid, which is a fundamental motor skill that protects the subject from severe injuries and connects the previous and next actions for fluent transitions. A falling motion requires very accurate control because a robot must decelerate huge vertical momentum within a fraction of a second, and a minor failure will cause huge damage to the body parts. In addition, a wide range of initial conditions must be considered for robustness because falling can be initiated by unexpected situations. The development of falling controllers will make virtual characters and robots to execute motions safely and smoothly, and its principles can be applied to the other highly dynamic motions with huge momentum.

There are several issues when a user develops physics-based controllers for difficult dynamic motor skills. The first problem arises when the user designs objectives, control mechanisms, and control parameters, which require a lot of prior knowledge. Since a large portion of controller designs remains unknown until testing the implementation, the design process is often driven by trial-and-errors. This can be very

tedious and time-consuming, especially in the traditional monolithic optimization setting that may take several hours to days. Thus, it is important to develop a simple and intuitive design process that allows the user to train controllers within a short amount of time. An iterative learning can be more efficient due to short feedback loops that can easily test and modify the controller, and more intuitive because it is similar how human learn motor skills through progressive process of coaching and practicing.

Optimizing control parameters for dynamic motions is another time-consuming step that requires a lot of computing power. Typically, whole-body dynamic tasks typically have a cost function that is multimodal, non-linear, non-convex, and discontinuous due to an under-actuated system and discrete contacts. Further, control parameters are likely to be in a high dimensional space with small feasible regions that does not generate undesired behaviors. These difficulties often require the most robust optimization algorithm. In computer animation, a robust black-box sampling-based method, Covariance Matrix Adaption Evolution Strategy (CMA-ES) [], has been frequently applied to discontinuous control problems, such as biped locomotion [], parkour-style stunts[], or swimming []. However, CMA-ES is not the most effective algorithm for optimizing dynamic motor skills when they have small feasible regions or high-dimensional control space due to the parametization. This motivated me to work on improving the performance of the baseline algorithm, CMA-ES, for more difficult tasks with many user constraints I further extend CMA-ES for a parametrized motor skill, which is essential for operating a robot in the unpredictable environment.

Unlikely the optimization for virtual characters, a control policy search for hardware with many trials is often infeasible because conducting hardware experiment can be expensive and time-consuming. Moreover, an execution of a bad controller on a robot can potentially cause disastrous damage to the robot and enormous cost to repair. To reduce the number of trials on the hardware, a virtual simulation is

used as a practical solution that provides a fast and safe evaluation of the control parameters. However, it suffers from *simulation bias* in which controllers developed for a virtual character do not work on hardware due to differences in the two systems. The *simulation bias* is hard to explicitly model because it can be caused by many reasons, such as different mass-distributions, sensor and actuator noises, command delays, and more. Therefore, a data-driven model-based policy search, which iteratively updates the simulation using collected hardware data, is a promising method to model the simulation bias, which will be discussed in this dissertation.

I will present the following identified problems for developing controllers for highly dynamic motor skills.

1.1 Falling Strategies for Humanoids

Highly dynamic motions often accompany the abrupt momentum changes, which can cause large contact forces to characters. Therefore, how to manage falls is a fundamental motor skill to reduce damages to humanoids and achieve fluent transitions between motor skills. In this dissertation, I will discuss two different falling scenarios, for virtual and real humanoids. For a virtual character, I will describe a general controller that allows the character to fall from a wide range of heights and initial speed, which are inspired by falling of traceurs. For a real robot, a general falling strategy for handling various external perturbations is introduced, which is feasible to be executed by actual hardware. The effectiveness of the presented strategies will be validated 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, I will show how to create an on-line 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.

Inspired by falling skills of Parkour(Figure 1), I formulate the falling problem with three phases, *airborne*, *impact*, and *rolling* based on the contact states. 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. I will demonstrate that the motion generated by the proposed controller induces smaller joint stress, which is still four times lower than a rag-doll motion at the worst cases.

Figure 1: A falling motion of Parkour.

1.1.2 Multiple Contact Planning for Humanoid falls

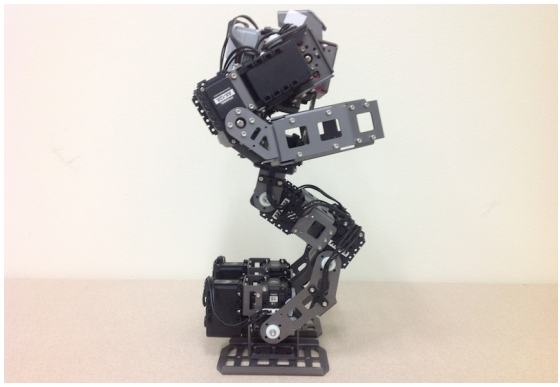


Figure 2: Hardware of BioloidGP robot.

Chapter 4 will describe 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 multiple contact planning provides a unified framework that can represent many existing falling techniques [,,]. Then, I will show how to efficiently optimize the multiple contact falling strategy to the given initial state using a simplified model and dynamic programming. Finally, various scenarios will be tested on simulated humanoids and the actual hardware (Figure 2) to show that our algorithm plans various falling strategies with different contact sequences.

1.2 Learning of Dynamic Controller for Characters

Teaching a physically simulated character a new motor skill requires a lot of efforts from the controller designer, from the design of the control mechanism to the tweaking of low-level control parameters. To simplify the learning process, I will introduce an intuitive and interactive framework for developing dynamic controllers that is inspired by how humans learn dynamic motor skills through a iterative process of coaching and practicing. Further, we propose two optimization techniques that can extend the popular policy search algorithm, CMA-ES, to accelerate the convergence rate and to optimize a parametrized objective function.

1.2.1 Iterative Design of Dynamic Controllers

In Chapter 5, I will describe an iterative framework to design dynamic controllers using high-level, human-readable instructions, inspired by a training process of athletes that consists of interactive coaching and repetitive practices (Figure 3) To enable interactive coaching, I introduce "control rigs" as an intermediate layer of control module to facilitate the mapping between human instructions and low-level control parameters. 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 will be shown with example parkour motions.

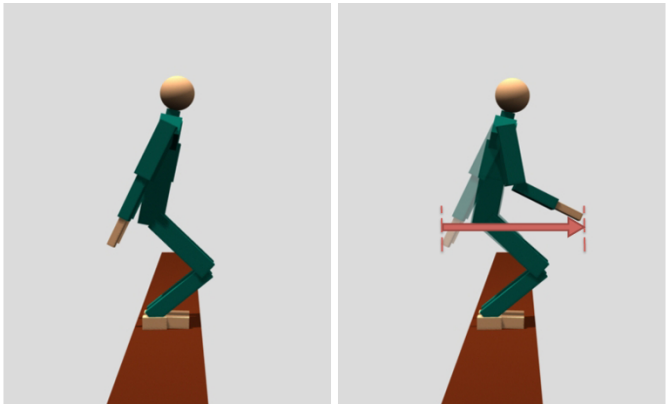


Figure 3: The proposed learning frame uses human readable instructions to teach motions.

1.2.2 Optimization with Failure Learning

A controller with many user constraints is difficult to optimize due to the relatively small feasible region. In Chapter 6, I will describe a new optimization algorithm for highly-constrained problems based on the observation of humans ability to learn 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 the quality of the samples, resulting a faster convergence than the standard CMA-ES.

1.2.3 Optimization for Parametrized Motor Skills

In Chapter 7, I will explain the optimization of parametrized motor skills. The parametrization of the learned motor skills is an essential ability because a robot can reinterpret the skill to a new situation, without learning from scratch. Instead of maintaining a single Gaussian distribution, the algorithm reduces the number of samples by evolving a parametrized probability distribution for a range of skills. I will test the algorithm on a simulated humanoid robot learning three parametrized dynamic motor skills, including vertical jump, kick a ball, and walk.

1.3 Model-based Learning for Virtual and Real Characters

Figure 4: Bongo Board balance toy.

In Chapter 8, I will describe an iterative approach for for learning hardware models and optimizing control policies with as few hardware experiments as possible. Instead of learning hardware models from scratch, the proposed approach only learns the difference from a simulation model. Similarly to the previous

work, Gaussian Process is used to model the difference between dynamics of virtual and real characters based on the collected hardware data. To prove the concept, I will validate the algorithm on two different simulation models, one with perfect contacts and one with realistic contacts, by finding a balancing controller for a 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 community. These contributions are as follows:

- **A falling and landing strategy for virtual characters** The falling strategy presented in the dissertation 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.

- **A multiple contact falling strategy for robots** I also introduce a new falling strategy that can optimize a sequence of contacts, which optimizes the number and locations of contacts for the given initial state.

- **An iterative learning framework for dynamic motor skills** Unlikely previous monolithic design processes in the literature, I proposed an iterative and interactive learning framework using human readable instructions. Starting from a basic controller, the proposed framework allows a user to easily train complex dynamic motion controllers within minutes, with only a few high-level instructions from the user.

- **An optimization technique for highly constrained problems** I 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 Supported Vector Machines.

- **An optimization technique for parametrized tasks** I introduce an efficient evolutionary optimization algorithm for learning parametrized skills to achieve whole-body dynamic tasks, which is much faster than the baseline algorithm, CMA-ES.

- **A model-based policy search for reducing hardware experiments** I propose an iterative approach for learning hardware model and optimizing policies with as few hardware experiments as possible by learning the difference between a simulation model and hardware.

In the next chapter, I will discuss the related work conducted by other researchers to address similar problems.

CHAPTER II

RELATED WORK

In this section, I will review relevant previous work done in computer graphics, robotics, and bio-mechanics. In Section 2.1, I will start with a brief introduction on popular animation techniques to generate highly dynamic motions of biped characters in the physics-based simulation. In Section 2.2, I will review previous methods for controlling falling motions of humanoids to reduce damage to body parts. In Section 2.3, I will briefly summarize prior interactive interfaces for selecting parameters, especially under the *human-in-the-loop optimization* paradigm, which is adopted in this dissertation for incorporating interactive user interventions in a controller design process. In Section 2.4, I will review various policy search techniques and optimization algorithms in both computer graphics and robotics that find optimal control parameters for humanoids.

2.1 Physics-based simulation of dynamic motor skills

2.1.1 Physics-based 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 [34, 108] 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 [115, 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 [18, 68]. Combination of PD servos and a specialized balance controller driven by a simple state machine was a very successful strategy [111], which enabled much follow-on work in biped control [101, 12, 50, 38]. Global planning of momentum has also been applied to a wide range of motion from standing balance [57] to locomotion [64, 110] to highly dynamic motion [31, 55, 7, 114] Coros *et al.* adopted Jacobian transpose control from robotics literature [95] to generate stable biped and quadruped locomotion [12, 13]. Ha *et al.* further demonstrated the effectiveness of the Jacobian transpose control on dynamic stunts [56, 31, 30].

2.1.2 Physics-based animation of highly dynamic motions

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.* [34, 108] showed that carefully crafted control algorithms can simulate highly athletic motions, including diving, tumbling, vaulting, and leaping. Faloutsos *et al.* [22] composed primitive controllers to simulate more complex motor skills, such as a kip-up move or a dive down stairs. Liu *et al.* [56] 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 [113] 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 [58, 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 [54, 23, 84, 94]. This thesis

the approach of physical simulation, but I 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 falling motions

2.2.1 Falling detection techniques

To activate a falling controller, a robot first predicts a fall and try to recover the balance if it is possible. Various machine learning techniques has been proposed to detect falls, such as Principal Component Analysis [41] or Supported Vector Machine [43]. Horn and Gerth [35] 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

When a robot is pushed hard and falling is inevitable, various techniques has been proposed to minimize damage on humanoid. Fujiwara *et al.* [26, 28, 25, 24] proposed falling techniques inspired by Japanese martial arts (*Ukemi*). Ogata *et al.* [73, 72] evaluates the risk of falling with predicted ZMP and optimizes COM trajectories to reduce damage. Ruiz-del-Solar *et al.* [83, 82] designed low damage falling sequences for soccer robots and verified them in the simulation. Wang *et al.* [104] formulated an optimization of whole body trajectories as a nonlinear programming problem and solved it with heuristics. Lee and Goswami [49] proposed a control strategy that reorients the robot to fall with a backpack for absorbing shock. Yun and Goswami [112] 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, [29] proposed a fall direction-changing strategy that utilizes foot placement and inertia shaping.

In contrast to the related work for falls caused by external perturbations, there are more works that focus on falls from higher places. In those cases, control strategies during long airborne phase become critical for safe landing. I draw 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 [21, 36]. In robotics, Bingham *et al.* [9] 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.

2.2.3 Falling strategies of animals

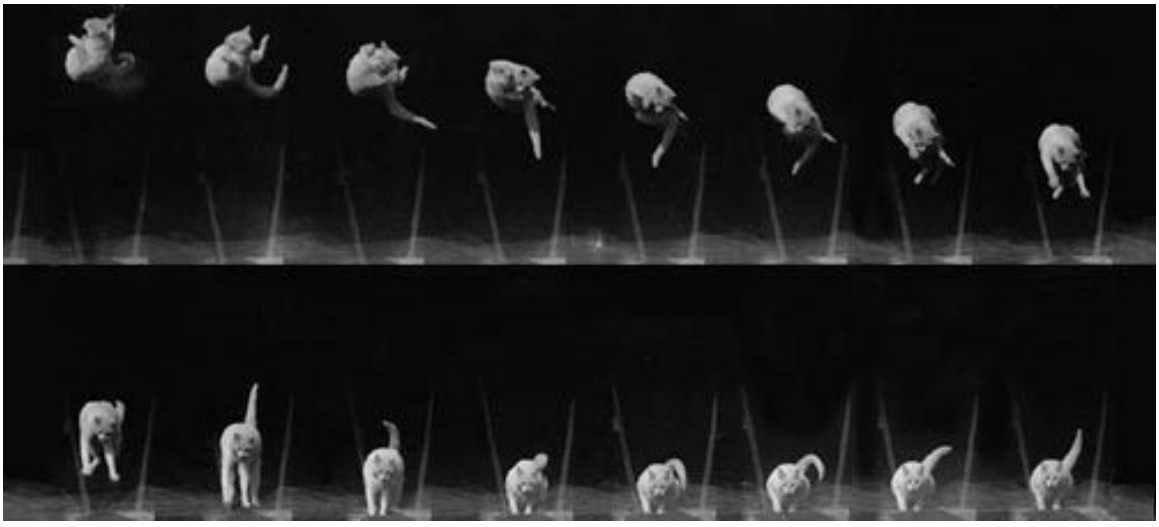


Figure 5: A cat is able to right itself as it falls to land on its feet, irrespective of its initial orientation.

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 [40, 62, 85] (Figure 5). Lizards swing their tails

to stabilize their bodies during a leap [51]. 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 Interactive interfaces

2.3.1 Parameter selection interfaces

A common approach in parameter selection interfaces is to present the parameter space (or a collection of samples thereof) in an explorable way, through 2D layout of results [59]; careful selection of sliders [74, 52]; or (in a physics context) direct manipulation [76] or in-situ visualization [99]. Both [105, 106] represent simulations as tracks (or word lines) where each parameter change corresponds to branches that spawn new tracks, and [105] describes how to incorporate uncertain parameter values into this explorable visualization. Meanwhile [10] clusters outcomes from different timelines to suppress minor variations and to highlight entire outcome categories. These methods help a designer understand the effects of parameter variation on a single set of initial conditions.

2.3.2 Human-in-the-loop interfaces

Without human guidance, fully automated optimization algorithms sometimes produce undesired solutions due to unexpected factors or situations. To fill the gap, researchers have developed semi-automatic systems which involve a human in the process to provide prior knowledge and guidance to the optimization. [86]. This type of optimization systems, called *human-in-the-loop* (HITL) optimization, have proven effective for various problems, such as vehicle planning [107] 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, I develop a new controller design framework that exploit the nature of HITL computation paradigm.

2.4 Policy search algorithms

2.4.1 Model-free policy search algorithms

In robotics, a common techniques for searching optimal policy parameters is model-free policy optimization, where the policy is improved through a number of hardware trials [65, 47]. 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 [70] or to provide a good initial trajectory by human demonstration [8]. 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) [32], has been frequently applied to discontinuous control problems, such as biped locomotion [101, 102, 103], parkour-style stunts[55, 30], or swimming [97]. To compensate the expensive cost of sampling-based algorithm, different approaches have been proposed, including exploiting the domain knowledge [101, 102, 103], shortening the problem horizons [89], or using a classifier to exclude infeasible samples [30]. Based on the previous success of CMA-ES, I proposed new sampling-based algorithms that resemble the evolution process of distribution.

2.4.2 Model-based policy search algorithms

Difference between a robot and its simulation model becomes a serious problem when we try to use controllers obtained by model-based optimization or tuned in simulation

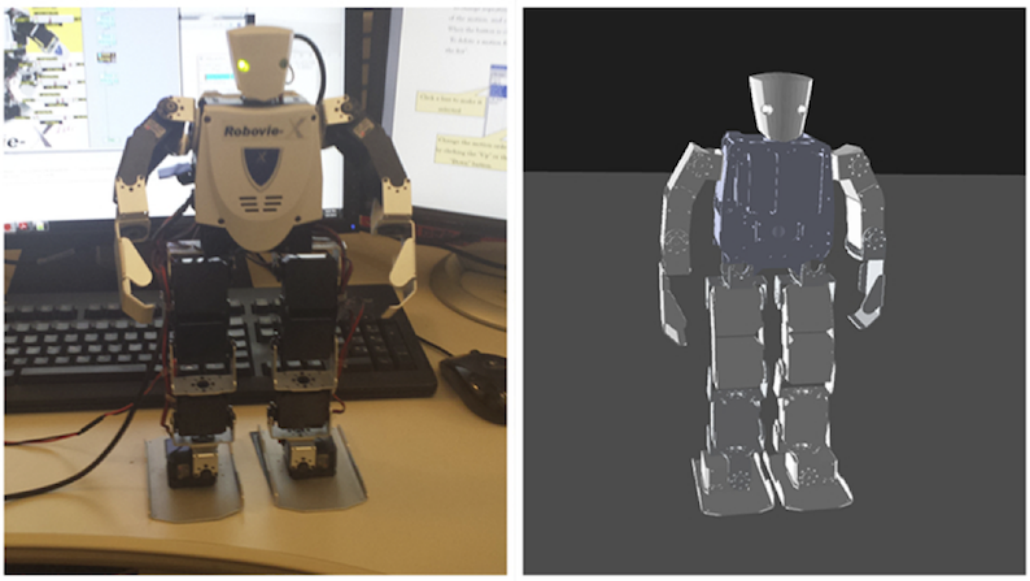


Figure 6: Difference between a real robot and its simulation model results different motions from same controllers.

(Figure 6). Classical parameter identification techniques [42] 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 [109].

A number of researchers have attempted to overcome the drawbacks of these approaches by combining simulation and real-world data [96, 63, 75]. Abbeel et al. [5] used an inaccurate model to estimate the derivative of the cost with respect to the policy parameters. Ko et al. [44] 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. [20] also used Gaussian Process for learning the dynamics model from scratch. Similarly, Morimoto et al. [66] 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. [14] 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 [46] 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. da Silva *et al.* [17, 15, 16] 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 [37]. A DMP defines a form of control policies which consists of a feedback term and a feedforward forcing term. Ude *et al.* [100] 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.* [67] proposed a mixture of DMPs and used a gate network to activate the appropriate primitive for the given target parameters. Kober *et al.* [48] 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.* [60] 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.* [71] 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 [15]. 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.* [33] 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, I 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

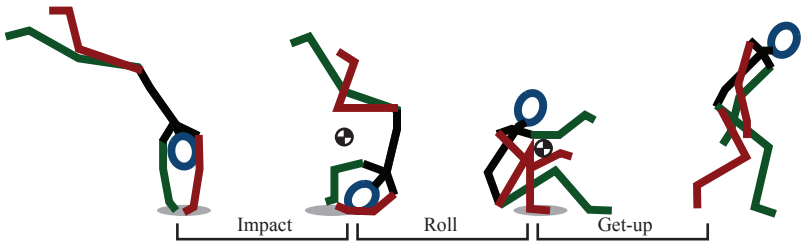


Figure 8: Three stages in the landing phase.

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

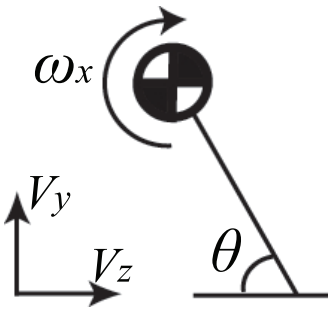
At the beginning of a fall, the character first decides on a landing strategy. During the airborne phase, the character optimizes its moment of inertia to achieve the ideal landing condition. The landing phase is divided into three stages: impact, rolling, and getting-up (Figure 8). The impact stage 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.* [11] 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

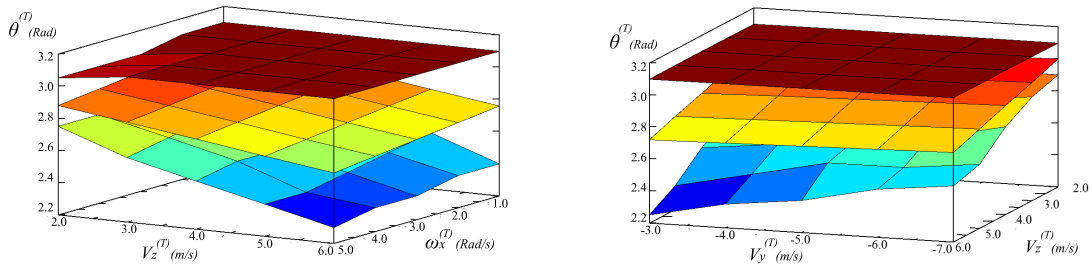


Figure 10: Samples for hands-first landing strategy. Successful samples are bounded between top and bottom planes along $\theta^{(T)}$ axis. The middle plane, average of the two, indicates the linear relation of the ideal landing condition.

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).

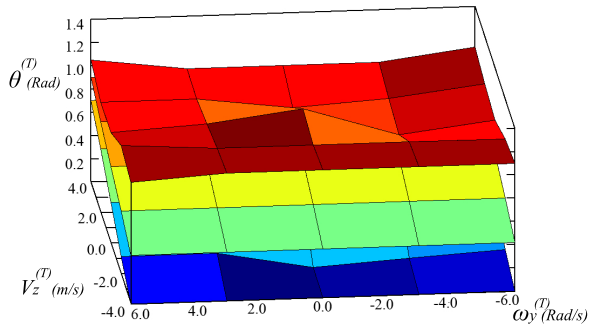


Figure 11: Samples in the space of $v_z^{(T)}$, $\omega_y^{(T)}$, and $\theta^{(T)}$. The spinning velocity $\omega_y^{(T)}$ has minimal effect on the success of a sample.

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

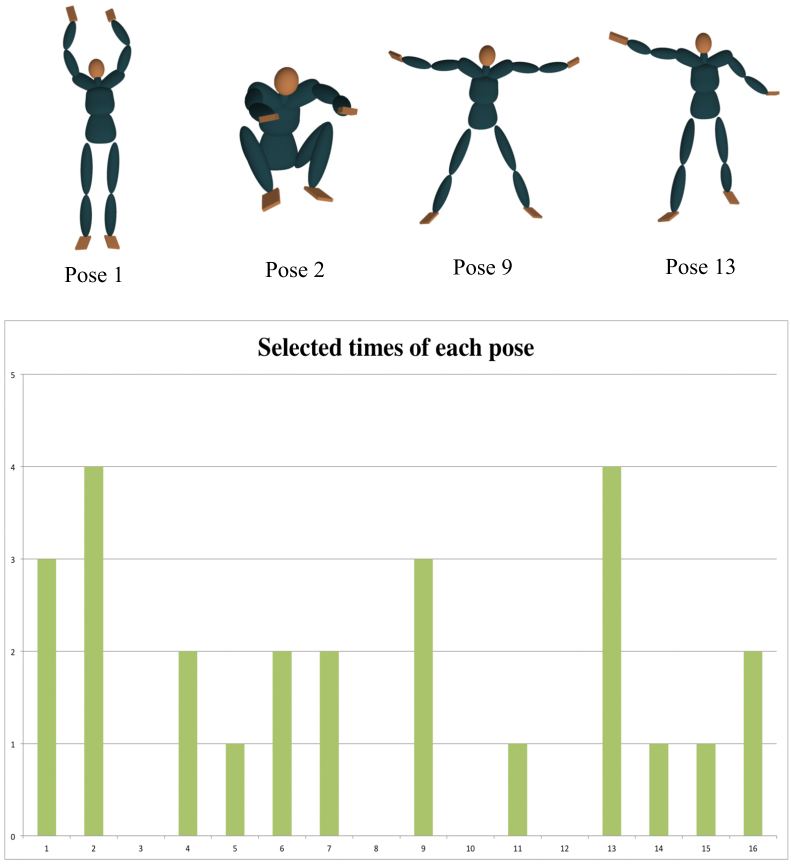


Figure 12: Among 16 poses in **Q**, pose 1, 2, 9, and 13 are frequently selected by the airborne controller

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

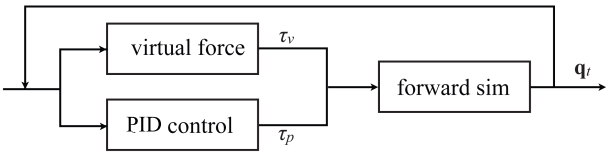


Figure 13: Landing phase controller.

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).

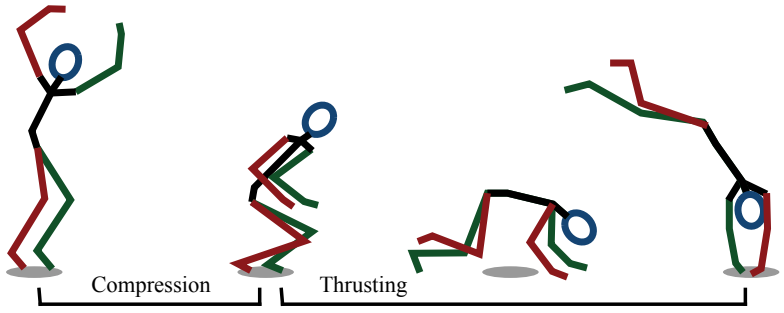


Figure 14: Two-step impact stage for the feet-first strategy.

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
$\bar{c}_y$	$\dot{\bar{c}}_y$	$\dot{\bar{c}}_{x/z}$	k_v	k_p (Eq 8)	ω_{MAX}
0.4m	0.0m/s	4.0m/s	500	800	3.3 Rad/s

where $\dot{\bar{c}}_{x/z}$ 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.

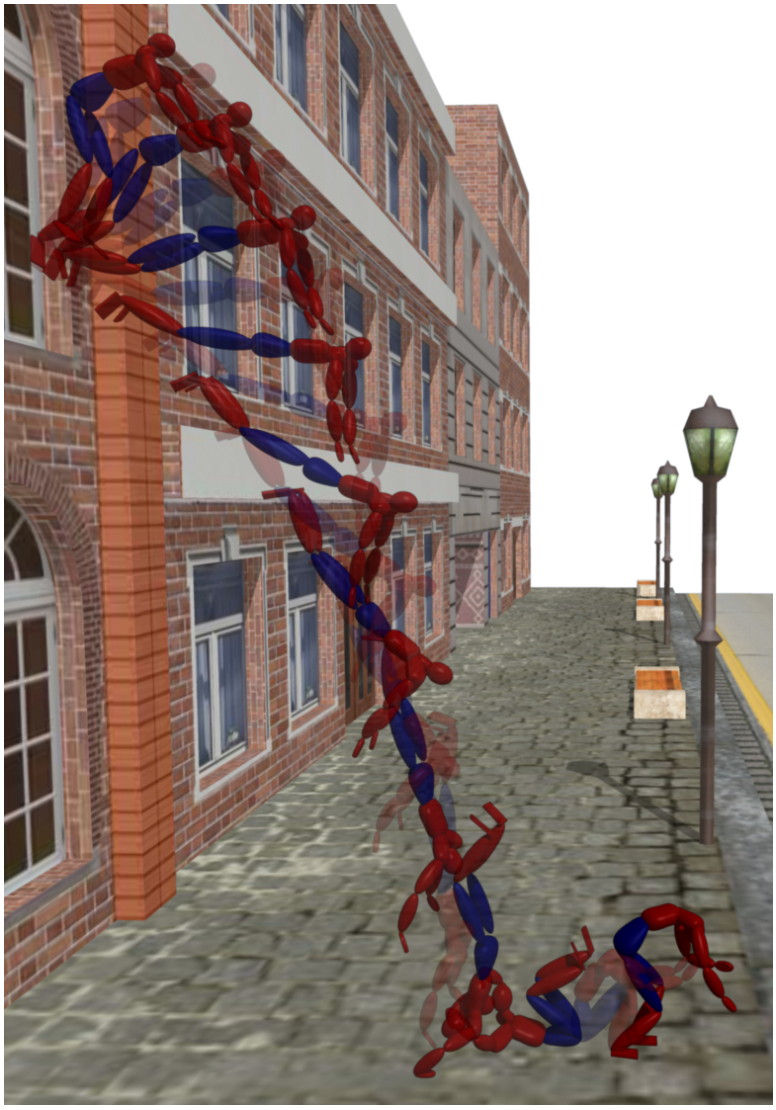


Figure 15: Hands-first landing motion.

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

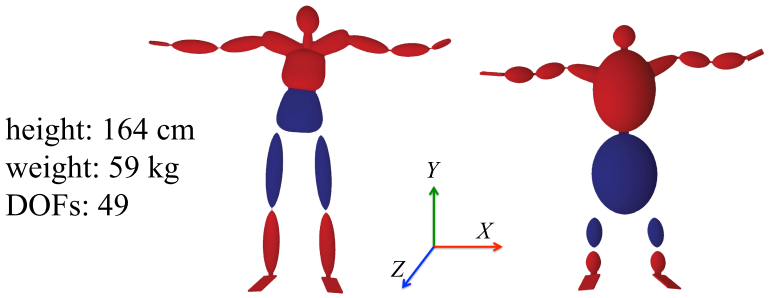


Figure 16: Left: The character model used for most examples. Right: A character with a disproportionately large torso and short legs.

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.

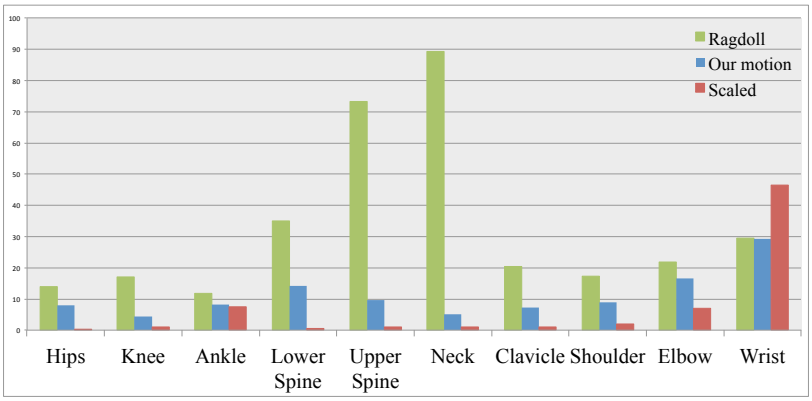


Figure 17: Maximal stress for each joint from a hands-first landing motion. Results are quantitatively similar across all of our simulations. Green: Ragdoll motion. Blue: Our motion. Orange: Joint stress scaled by mass.

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 ([6]). 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 [61, 45]. 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.*

[26, 28, 27, 25, 24] proposed fall strategies inspired by *Ukemi*, a set of techniques used in *Judo*. Ogata *et al.* [73, 72] 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.* [83, 82] designed falling sequences for simulated robots playing soccer. Wang *et al.* [104] directly solved joint trajectories of a three-link robot subject to the full-body dynamics. Lee and Goswami [49] proposed a control strategy that reorients the robot to fall on its backpack. Yun and Goswami [112] 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.* [29] 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.* [73] employed a single contact at hands. Fujiwara *et al.* [24] 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 [4].

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

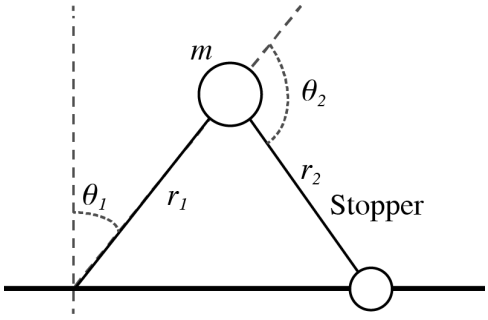


Figure 19: The abstract model consists of a telescopic inverted pendulum and a massless stopper.

The falling motion of biped humanoids has been modelled by a simple 2D inverted pendulum with a massless telescopic rod [28, 25]. 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

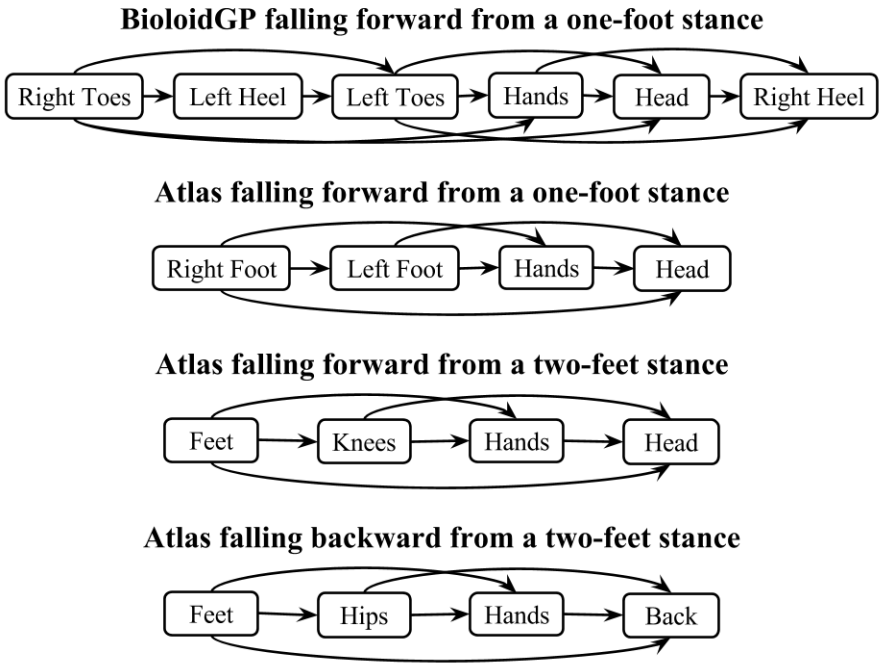


Figure 20: Contact graphs

employ. For example, a path from feet to knees to hands in Figure 20 represents the falling strategy described in [26].

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 [19, 53] 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.

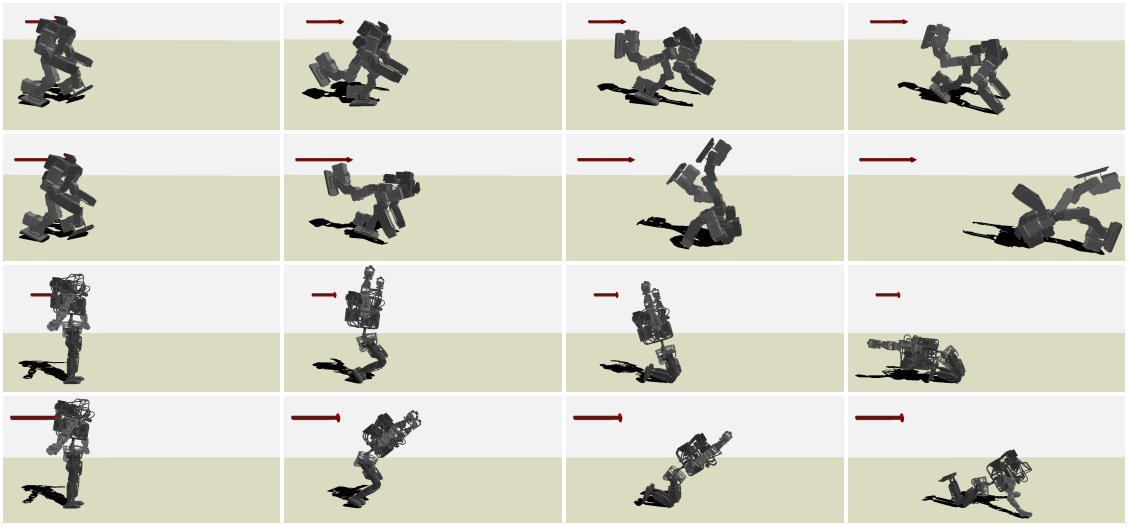


Figure 21: First row: BioloidGP forward falling from a one-foot stance due to a 5.0N push. Second row: BioloidGP forward falling from a one-foot stance due to a 8.0N push. Third row: Atlas forward falling from a two-feet stance due to a 1000N push. Fourth row: Atlas forward falling from a two-feet stance 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 [112]. When we increased the magnitude of the push to 8.0N, the rolling strategy, effective for breaking high speed falls [4], 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).

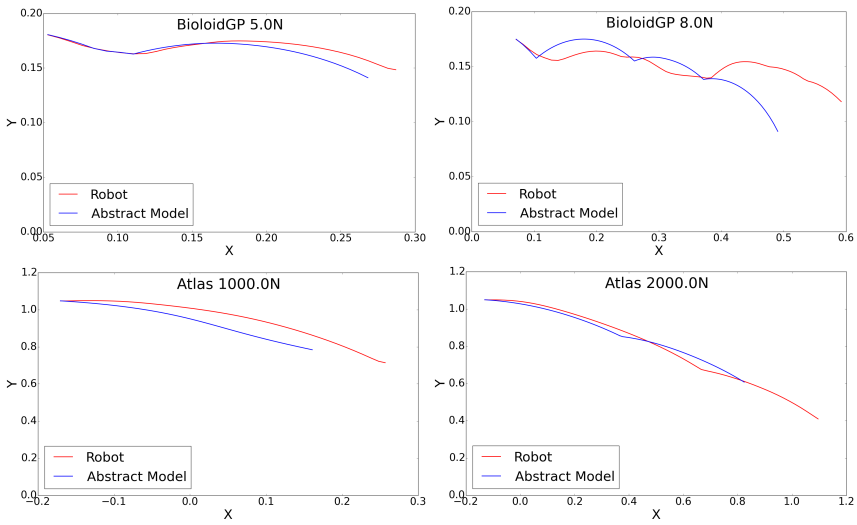


Figure 22: COM trajectories between the abstract model (Blue) and the robot (Red). Top left: BioloidGP forward falling from a one-foot stance due to a 5.0N push. Top right: BioloidGP forward falling from a one-foot stance due to a 8.0N push. Bottom left: Atlas forward falling from a two-feet stance due 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 [27]. When the external force became stronger ($2000N$), the robot utilized an additional contact with hands, similar to the strategy reported in [24, 72]. 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 [26]. 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

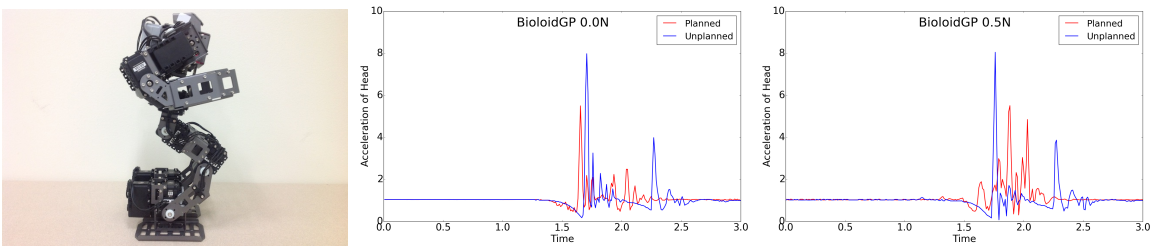


Figure 23: We measured the acceleration at the head of BioloidGP (Left). For both 0.0N (Middle) and 0.5N (Right) cases, the planned motions (Red) yielded about 68% of the maximum acceleration of the unplanned motions (Blue).

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 [112] and [29] 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 [29].

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

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.

5.1 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 [46] 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. 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 contact model. These models are different enough that a policy optimized for the former cannot stabilize the latter.

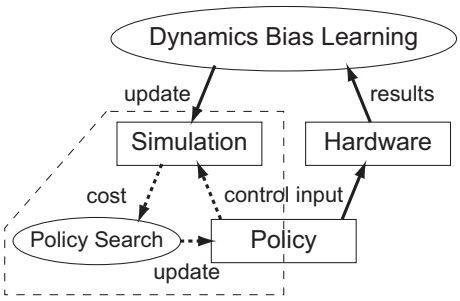


Figure 24: Framework of our approach.

The rest of the paper is organized as follows. In Section **??**, we first review the related work in machine learning literature in the context of robot control. Section 5.2 gives an overview of our framework, followed by more details on the model learning in Section 5.3 and policy optimization in Section 5.4. Section 5.5 presents simulation results and analysis. We finally conclude the paper in Section 5.6.

5.2 Overview

We developed an iterative reinforcement learning process to alternatley refine the model and policy. Figure 24 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 [44], we use Gaussian Process to model the difference between a dynamics model and actual

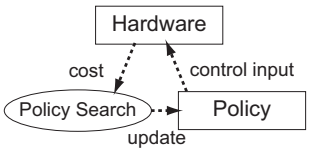


Figure 25: Direct policy search.

robot dynamics. On the other hand, we also adopt the iterative learning scheme as in [5] 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 [20].

We compare our framework with conventional direct policy search represented in Figure 25. 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.

5.3 Learning the Dynamics Model

5.3.1 Dynamics Bias Formulation

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

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{20}$$

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:

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) + \boldsymbol{g}_D\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{21}$$

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.

5.3.2 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:

$$k\left(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}'\right) = \alpha^2 \exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)^T \boldsymbol{\Lambda}^{-1}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)\right) + \delta_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{x}}'}\sigma^2 \tag{22}$$

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 y_i $(i = 1, 2, \ldots, N)$, we can predict the output for a new input $\tilde{\boldsymbol{x}}^*$ by

$$y^* = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y} \tag{23}$$

with variance

$$\sigma^2 = k(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*) - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_* \tag{24}$$

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.

5.3.3 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:

$$\boldsymbol{\Delta}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1} - \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right). \tag{25}$$

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.

5.3.4 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.,

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) + \exp\left(-d|\boldsymbol{\sigma}^2|^2\right) \bar{\boldsymbol{\Delta}}_t \tag{26}$$

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 (26) is nearly zero, meaning that we mostly use the prediction by the model.

5.4 Data-Efficient Reinforcement Learning

Algorithm 3 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

$$Z = c(T - t_{fail}) + \max_{1 \le t \le T} \boldsymbol{x}_t^T \boldsymbol{R} \boldsymbol{x}_t + \sum_{t=0}^{T} \boldsymbol{u}_t^T \boldsymbol{Q} \boldsymbol{u}_t \tag{27}$$

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 $\boldsymbol{R} \in \Re^{n \times n}, \boldsymbol{Q} \in \Re^{m \times m} \geq 0$ 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 5.3.3. 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 [39] works best for our problem. Theoretically, the DIRECT algorithm is capable of finding the globally optimal solution relatively quickly. We terminate the algorithm when the relative change in the cost function value in an optimization step is under a threshold

Algorithm 3: 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

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.

5.5 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.

5.5.1 Bongoboard Balancing

The task we consider is balancing on bongoboard of a simple legged robot shown in Figure 26(a). Specifically, we apply the output-feedback controller developed by Nagarajan and Yamane [69] 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 26(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 26(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

$$u = \boldsymbol{H}\boldsymbol{z} \tag{28}$$

where $\boldsymbol{H} = (h_1\ h_2\ \ldots\ h_5)$ 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

$$h_i = \begin{cases} \exp(\hat{h}_i) - 1 & \text{if } \hat{h}_i \geq 0 \\ -\exp(-\hat{h}_i) + 1 & \text{if } \hat{h}_i < 0 \end{cases} \tag{29}$$

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 24.

The first model is derived by the Lagrangian dynamics formulation as described in [69]. 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 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.

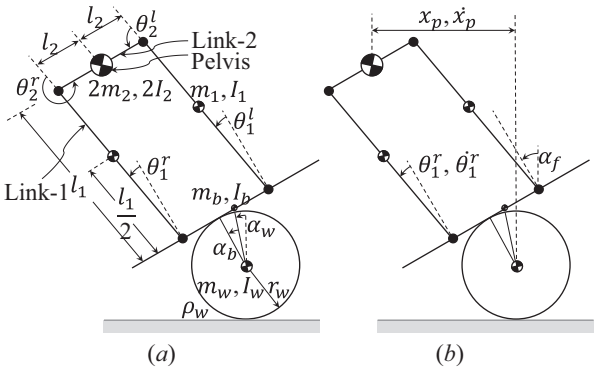


Figure 26: Robot balancing on a bongoboard.

- 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 27 show an example of using a policy optmized for the Lagrangian and Box2D models for both the Box2D model 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 5 summarizes the parameters we used for the experiments.

Table 5: 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

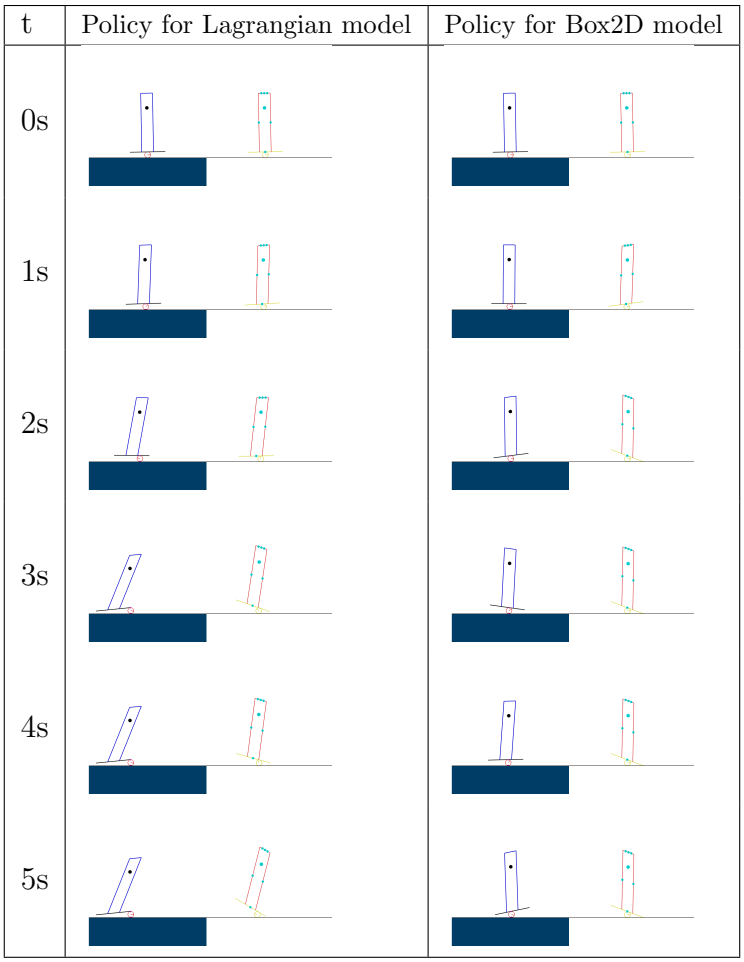


Figure 27: Simulation result of a policy optimized for the Lagrangian model (left column) and Box2D model (right column). In each snapshot, the left and right figures are the Box2D and Lagrangian model simulations respectively.

5.5.2 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 (α_w α_b θ_1^r) of the state spate. An example is shown in Figure 28, 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.

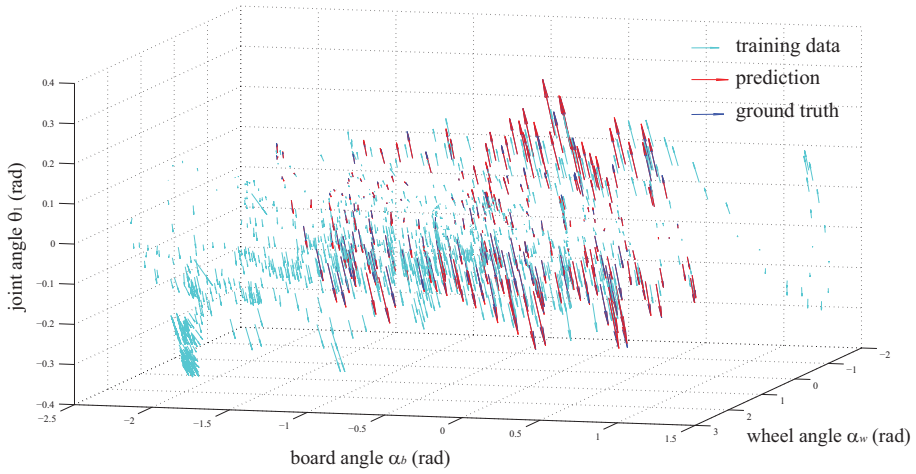


Figure 28: Velocity field of the learned dynamics model. Cyan: training data; red: prediction; blue: ground truth.

5.5.3 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 6 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, 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.

Table 6: 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

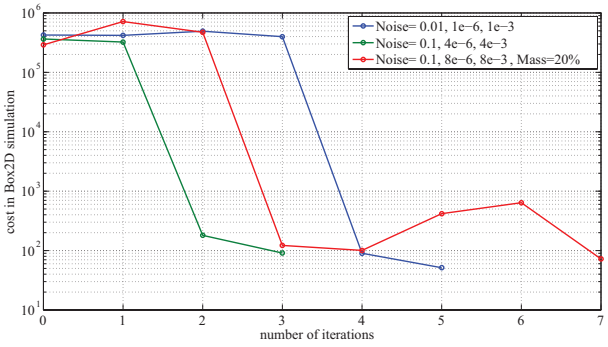


Figure 29: Change of cost function value in Box2D simulations over iterations.

Figure 29 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.

5.5.4 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 30 depicts the balancing success rates starting from various wheel and board

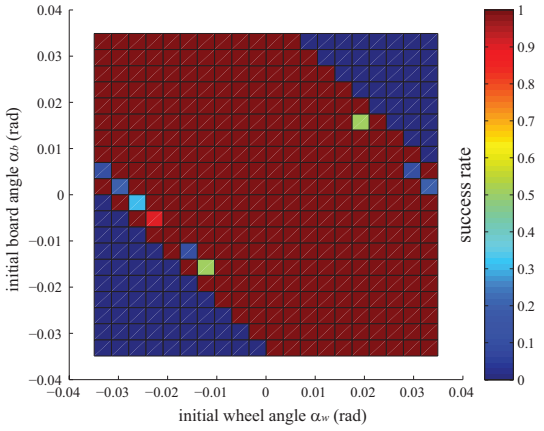
angles, using a policy optimized with Box2D simulation without noise (a) and with noise (b). This result clearly shows that the policy optimized in noisy environment can successfully balance the robot from a wider range of initial states under noisy actuator and sensors.

5.6 Conclusion and Future Work

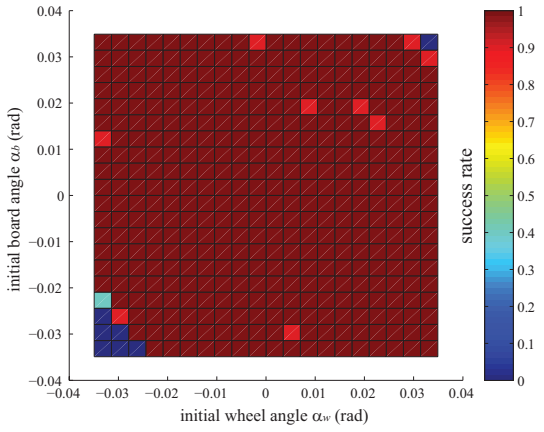
This paper presented a framework for model learning and policy optimization of robots that are difficult to conduct experiments with. The key idea is to learn the difference between a model and hardware rather than learning the hardware dynamics from scratch. We also employ an iterative learning process 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 30: 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.

APPENDIX A

SOME ANCILLARY STUFF

Ancillary material should be put in appendices, which appear just before the bibliography.

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

[5] 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.

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

[7] 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.

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

[9] 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.

[10] Bruckner, S. and Moller, T., "Result-driven exploration of simulation parameter spaces for visual effects design," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, pp. 1468 –1476, nov.-dec. 2010.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] 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.

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

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

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

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

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

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

[38] 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.

[39] 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.

[40] 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.

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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] Lindow, N., Baum, D., and Hege, H.-c., "Perceptually Linear Parameter Variations," *EUROGRAPHICS 2012*, vol. 31, no. 2, 2012.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

[58] 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.

[59] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J. K., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., and Shieber, S., "Design galleries: A general approach to setting parameters for computer graphics and animation," in *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pp. 389–400, Aug. 1997.

[60] 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.

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

[62] 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.

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

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] Ovsjanikov, M., Li, W., Guibas, L., and Mitra, N. J., "Exploration of continuous variability in collections of 3d shapes," *ACM Trans. Graph.*, vol. 30, pp. 33:1–33:10, July 2011.

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

[76] Popović, J., Seitz, S. M., Erdmann, M., Popović, Z., and Witkin, A., "Interactive manipulation of rigid body simulations," in *Computer Graphics*

(Proceedings of SIGGRAPH 2000), Annual Conference Series, pp. 209–218, ACM SIGGRAPH, July 2000.

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

[83] 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.

[84] 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.

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

[86] Scott, S., Lesh, N., and Klau, G., "Investigating human-computer optimization," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2002.

[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] Twigg, C. and James, D. L., "Many-worlds browsing for control of multibody dynamics," *ACM Transactions on Graphics (TOG)*, 2007.

[100] 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.

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

[102] 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.

[103] 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.

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

[105] Waser, J., Fuchs, R., Ribicic, H., Hirsch, C., Schindler, B., Bloschl, G., and Groller, M., "World lines," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, pp. 1458 –1467, nov.-dec. 2010.

[106] Waser, J., Ribicic, H., Fuchs, R., Hirsch, C., Schindler, B., Bloschl, G., and Groller, M., "Nodes on ropes: A comprehensive data and control flow for steering ensemble simulations," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, pp. 1872 –1881, dec. 2011.

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

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

[109] 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.

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

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

[112] 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.

[113] 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.

[114] 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.

[115] 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.

INDEX

VITA

Perry H. Disdainful was born in an insignificant town whose only claim to fame is that it produced such a fine specimen of a researcher.

Learning Dynamic Motor Skills
for Virtual and Real Humanoids

Sehoon Ha

88 Pages

Directed by Professor C. Karen Liu

This is the abstract that must be turned in as hard copy to the thesis office to meet the UMI requirements. It should *not* be included when submitting your ETD. Comment out the abstract environment before submitting. It is recommended that you simply copy and paste the text you put in the summary environment into this environment. The title, your name, the page count, and your advisor's name will all be generated automatically.