

Staged learning of agile motor skills

by

Andrej Karpathy,

B.Sc., University of Toronto, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

May 2011

© Andrej Karpathy, 2011

Abstract

Motor learning lies at the heart of how humans and animals acquire their skills. Understanding of this process enables many benefits in Robotics, physics-based Computer Animation, and other areas of science and engineering. In this thesis, we develop a computational framework for learning of agile, integrated motor skills.

Our algorithm draws inspiration from the process by which humans and animals acquire their skills in nature. Specifically, all skills are learned through a process of staged, incremental learning, during which progressively more complex skills are acquired and subsequently integrated with prior abilities. Accordingly, our learning algorithm is comprised of three phases. In the first phase, a few seed motions that accomplish goals of a skill are acquired. In the second phase, additional motions are collected through active exploration. Finally, the third phase generalizes from observations made in the second phase to yield a dynamics model that is relevant to the goals of a skill.

We apply our learning algorithm to a simple, planar character in a physical simulation and learn a variety of integrated skills such as hopping, flipping, rolling, stopping, getting up and continuous acrobatic maneuvers. Aspects of each skill, such as length, height and speed of the motion can be interactively controlled through a user interface. Furthermore, we show that the algorithm can be used without modification to learn all skills for a whole family of parameterized characters of similar structure. Finally, we demonstrate that our approach also scales to a more complex quadruped character.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	vi
List of Figures	vii
Acknowledgements	ix
1 Introduction	1
1.1 Motivation	2
1.2 Why is motor control hard?	4
1.2.1 Modeling issues	4
1.2.2 Control issues	5
1.2.3 Simulation issues	6
1.3 Related work	6
1.3.1 Physics-based Computer Animation	7
1.3.2 Robotics	7
1.3.3 Kinesiology, Biomechanics, Neuroscience	8
1.3.4 Acrobot	9
1.4 Contributions	10
1.5 Thesis organization	10
2 System Overview	11
2.1 Characters	11
2.2 Control	13

Table of Contents

2.3	Skills	16
2.3.1	Hop	19
2.3.2	Flip	20
2.3.3	HopFlip	21
2.3.4	HopRoll	22
2.3.5	Stop	22
2.3.6	GetUp	23
2.4	Composite skills	23
2.4.1	Acrobatics	24
3	Learning	26
3.1	Overview	26
3.2	Trials	28
3.2.1	Motor Action reliability	30
3.3	Learning algorithm	31
3.3.1	Initialization	31
3.3.2	Phase 1: initial successful action generation	31
3.3.3	Phase 2: exploration	34
3.3.4	Phase 3: parameterization and generalization	37
3.3.5	Learning the recovery model for Hop	41
4	Results	42
4.1	Acrobot	42
4.1.1	Hop and Flip skills	43
4.1.2	Stop skill	45
4.1.3	HopFlip and HopRoll skills	46
4.1.4	Acrobatics and GetUp skill	52
4.1.5	Sample sequences of skills executed in sequence	54
4.1.6	Use of task parameters in planning	55
4.2	Quadruped character in 3D	57
5	Conclusions	61
5.1	Discussion	61
5.2	Limitations and future work	62

Table of Contents

Bibliography 67

List of Tables

2.1	Character diagram	13
2.2	Types of skills	18
3.1	Training sequences for all skills	29

List of Figures

1.1	Acrobot character	9
2.1	Character diagram	12
2.2	Character variations	12
2.3	System overview	14
2.4	Example Motor Action	16
2.5	Skills FSM	17
2.6	Hop visualization	19
2.7	Flip visualization	20
2.8	HopFlip visualization	21
2.9	HopRoll visualization	22
2.10	Stop visualization: Film strip	22
2.11	GetUp visualization	23
2.12	Acrobatics visualization	24
3.1	All stages of the learning algorithm visualized	27
3.2	An example of a trial	28
3.3	Learning: Phase 1 visualization	34
3.4	Learning: Phase 2 visualization	37
3.5	Learning: Phase 3 visualization	41
4.1	Capability chart for Hop and Flip skills of our characters	45
4.2	HopFlip capabilities 1	47
4.3	HopFlip capabilities 2	48
4.4	HopFlip capabilities 3	49
4.5	HopRoll capabilities 1	50

List of Figures

4.6	HopRoll capabilities 2	51
4.7	HopRoll capabilities 3	52
4.8	Phase 2 of Acrobatics	53
4.9	Sample run with moving camera	54
4.10	Sample runs with stationary camera	55
4.11	Sample run with terrain	56
4.12	Sample run with terrain 2	56
4.13	Dog character	57
4.14	Dog leaps	59
4.15	Dog leaps on platform	59
4.16	Dog demo	59
4.17	PCA analysis of dog experiences	60

Acknowledgements

I would like to thank my supervisor, Michiel van de Panne, for his continuous support, guidance, and his willingness to let me explore the research tangents that made my time here so enjoyable. Michiel's joyful excitement about the field is infectious, and I will cherish the times we spent reverse-engineering the human and animal motor system. The insights I gained through our discussions about academia have been particularly helpful and shaped me as a researcher.

I am also thankful to Nando de Freitas for contributing ideas to this project, and for helping me with this thesis. I had the opportunity to closely interact with Nando on many projects and endeavors over the last two years, and his support, excitement, and can-do attitude have been very helpful and inspiring.

I would also like to extend my thanks to David Lowe, Bob Woodham, and Kevin Murphy, all of whom I've come to know and admire over the last two years. I also had the pleasure of working with Steve Wolfman and Kimberly Voll as a TA on several occasions, and I am thankful to have witnessed their inspiring dedication to teaching.

My thanks go to my bullpen mates and lab mates Paul Vanetti, John Chia, Byron Knoll, Stelian Coros, Philippe Beaudoin, Ben Jones, and many others who I don't have enough space to name for contributing to a pleasant, fun working atmosphere and offering healthy distractions from work.

I wish to thank my sister for her unconditional support, and my parents, who sacrificed to bring us to Canada and made it possible for me to pursue my dreams.

Last but not least, I wish to thank Shelly, for giving me something to look forward to every day.

Chapter 1

Introduction

A basketball player takes a sharp turn through two defenders, catches the ball, instantly jumps up and dunks it into the basket. The agility with which humans and animals move through their environment remains unmatched by our state-of-the-art robots. The task of replicating these abilities has proven to be a difficult problem even for our physically simulated characters, where one is allowed to abstract away the myriad of difficulties involved when working with robotic hardware. Leaving physical implementation details aside, it is clear that we do not yet have a good understanding of how to control a complicated physical character, such as a human, to achieve demanding tasks in a complex environment with the agility and grace that we observe in nature.

It is difficult to develop a generic approach for simplifying the problem. Traditionally, much research has focused on controlling complex systems such as humanoid robots, but these solutions often restrict the repertoire of skills to ones that involve slow and deliberate looking motions. In contrast, this thesis works toward the ultimate goal described above from the other direction: Our goal is to produce quick and agile motions even if it comes at the immediate cost of having to work with simpler characters rather than characters of human complexity. In addition, we allow ourselves to concentrate on the theoretical problems associated with developing the appropriate representations and algorithms, and therefore work exclusively in a simulated environment. Physical simulation offers a fertile playground for modeling of skilled motion. It allows one to experiment with a wide variety of representations and hypotheses without being concerned with the limitations of robotic hardware. In addition, a simulation can be run much

faster than real time, which encourages the development of automated solutions.

Our work is inspired by the observation that humans and animals acquire their skills and abilities through a long process of incremental learning, during which progressively more complex skills are acquired and subsequently integrated with prior abilities. As an example, think of a child as it acquires the ability to run. It first figures out how to roll over, then crawl, then it learns to walk while being supported by furniture or parents, and only then does it slowly adjust its walk cycle into an agile running gait. Throughout the entire time, it is subject to supervision from its parents, who guide it along its development by contributing examples, giving it incrementally harder tasks to complete, and rewarding good attempts and strategies. Our approach is to model this process explicitly in order to learn a controller for several agile, integrated skills through a large collection of trial and error exercises. To simplify the framework and speed up the computation, we use a simple, planar character for most of our experiments. Later, we apply our learning algorithm to learn leaps for a simulated quadruped dog in order to demonstrate the scalability and generality of the approach.

1.1 Motivation

Imitation of skillful human and animal motion has applications in Computer Animation, Robotics, and many other areas of science and engineering.

For Computer Animation, the ability to create controllable physically realistic motion quickly and easily is desirable both in the film and video game industries. Current approaches to animation involve significant repetitive, manual labor of an artist, or processing of a large database of motion capture data. The former approach is a time-consuming and expensive process. In addition, if some specifications of the desired motion change, it may become necessary to redo large portions of the animation. The resulting

1.1. Motivation

motion is also not guaranteed to look and feel realistic. The use of motion capture can alleviate some of these shortcomings, but raises other concerns. For example, the actor who was used to collect the motion data may not have desirable physical characteristics such as body proportions, size, or style of motion. In addition, the final animation is limited in its capabilities by the available motion capture data. Having characters that are capable of performing controllable, physically-plausible and realistic-looking motions could aid with many aspects of the animation process.

Lessons learned in controlling physical characters in simulation also find applications in Robotics. While controllers built in simulation do not always transfer directly to real robots, they often provide a good initial guess. In general, it is a common practice to work in simulation first, and then attempt to transfer a solution to a real robot.

Finally, the study of how humans and animals execute skilled motion is a vast area of research that is of academic interest in many disciplines. Researchers from diverse fields, such as Anatomy, Control Theory, Robotics, Machine Learning, Biomechanics, Kinesiology and Neuroscience all study the problem. While many researchers approach the problem by reverse-engineering the human and animal motor system through experiments, it is a worthwhile goal to work in the other direction, and attempt to build computer systems that replicate it. In the end, we can only be certain that we understand the human or animal motor system if we are able to reproduce it in a simulation, or on a robot. Understanding of the principles that give rise to human and animal motion could also have far-reaching medical applications. It could lead to improvements in the design and functionality of prosthetics and exoskeletons. The models may also allow for better predictions of surgical outcomes regarding patient mobility.

1.2 Why is motor control hard?

The unintuitive difficulty of controlling a physical character is not apparent until one attempts to replicate our abilities in a physical simulation. After all, we learn to perform thousands of skilled motor actions in our lifetime without much effort. We walk around our environment without thinking about it, and perform many other manipulation tasks automatically. However, several types of issues arise when one sets out to implement a control solution for a physically-simulated character.

1.2.1 Modeling issues

Humans and animals are complex organisms made up of bones, flexible tendons and muscles, cartilage, and other tissue with a variety of physical properties. It is not yet feasible to model an entire body in its full complexity, and traditionally many simplifying assumptions are made. In particular, it is common to assume that a character is made up of some number of rigid bodies that are connected by idealized ball joints. Rigid bodies are convenient to work with mathematically, but lack many desirable properties, which may be detrimental to our efforts. Most notably, rigid bodies cannot cushion impacts as is done by the flesh on a foot, and cannot absorb and release energy as tendons do. Whether or not this is a crucial feature of our bodies that is necessary for natural movement is a subject of debate.

Additional simplifying assumptions are made when modeling the dynamics. Real bodies are powered by contractions of hundreds of antagonistic and synergistic muscles each with their own activation dynamics, but characters in simulation are typically controlled by directly applying desired torques at the joints. The dynamics of the movements are therefore, once again, slightly different as a result.

Modeling of the properties of the surrounding physical environment also poses challenges. Most often, ground is treated as a rigid plane with some

coefficient of friction. This approach is inadequate when trying to faithfully model the interactions of ground with feet of animals or humans. For example, animals can decide to sink their claws into the ground, providing additional friction when necessary. The most commonly used model of a foot, however, is a single rigid body.

1.2.2 Control issues

Part of the difficulty in modeling our motor system stems from the fact that most of the underlying complexity is hidden from our consciousness, and cannot be introspected. Even though a plethora experiments have been conducted on humans and animals to gain understanding of the control principles in use, there remains much disagreement in the scientific community on how to explain the data in a computational framework.

Much of the trouble also comes down to the sheer scale of the problem. Humans and animals are high-dimensional systems composed of hundreds of degrees of freedom. Even the simplest models of a human can therefore have as many as 30 degrees of freedom. Computing the torques for all joints over time such that the result is a stable walking or running gait is a challenging problem.

The unforgiving nature of physics introduces an additional layer of complexity. A small mistake in controlling a particular aspect of the motion may lead to disastrous consequences some time later. For example, if the foot of a human character gently scrapes the ground, friction may stop it in its path, thereby tipping the character unless the disturbance is immediately and appropriately corrected for. Conversely, many aspects of various motions need not be carefully controlled. For example, during a walking motion the arms of a character can move around without many restrictions.

1.2.3 Simulation issues

Even with an adequate model and controller, simulation may still not produce desired results. Sources of failure can be due to numerical artifacts in the simulation, or other issues pertaining to numerical stability. When simulating rigid bodies, it is easy to detect an overlap between two rigid bodies, but it is much harder to resolve it accurately. The simulation may also become unstable under a variety of conditions, such as when the controller asks for a rapid increase in the applied torques. Many of the popular physical simulation frameworks are explicitly designed to minimize errors due to numerical inaccuracies, rendering these concerns relatively minor. However, a programmer must keep these issues in mind when designing a suitable control solution.

1.3 Related work

The study of control laws that give rise to skilled motion has been a subject of research in various disciplines. However, every discipline tends to work on the problem from a different perspective, subject to different constraints, and usually with different goals in mind. Researchers in Computer Animation are primarily concerned with creating animations of characters for use in simulation scenarios, film and video games. In Robotics, the goal is to move a physical robot through an environment, subject to many hardware constraints. Finally, researchers in Kinesiology, Biomechanics and Neuroscience are specifically interested in modeling the control systems of humans.

The focus of this thesis, in particular, is on motor learning. How does one go about learning skills from experience? What is the structure of the learning process? What is the correct level of abstraction to work on, and what are the right representations for motion? How can one formalize the process in a learning algorithm? First, we discuss prior work on this topic from these disciplines. Later, we discuss existing control approaches for a character called *Acrobot*, which we use for most of our experiments.

1.3.1 Physics-based Computer Animation

In Computer Animation, there has been a growing trend of using physical simulation to achieve natural-looking motions. As far as animation is concerned, the downside of this approach is that it requires significant effort to design controllers that can accomplish even simple tasks, such as walking. However, over the last few years, significant progress has been made by several research groups [5, 9, 12, 14, 20] on the associated control problems. The prevalent approaches most often contain a significant hand-engineered component that is obtained from human insight into the problem. The parameters in the system are often either fine-tuned using optimization, or inferred from motion capture data. Even though it is now possible to control complicated physical systems such as humans for various tasks, it is still beyond our capabilities to generate rich, integrated, and agile motions such as a character playing basketball.

1.3.2 Robotics

The typical approach to motor control in Robotics has changed little over the last two decades [8]: Every particular task at hand is first modeled as accurately as possible, and then the roboticist develops a strategy for addressing that type of problem. If inaccuracies remain, all exceptions are handled using human understanding of the task. However, it is becoming increasingly clear that if robots are ever to leave factory floors and research environments, we will need to reduce or eliminate the strong reliance on hand-crafted models and skills.

Several research groups have therefore adopted the paradigm of learning motor skills from demonstration, which provides greater potential to scale to novel setups. The merit of this approach has been demonstrated in conjunction with Reinforcement Learning for several tasks, including learning a tennis swing for a robot[17], and helicopter flight[1].

Yet another more closely related alternative for learning in robots comes

from the newly emerging field of Developmental Robotics [13]. This branch of Robotics takes inspiration from biology and infant studies when designing algorithms for robot learning.

1.3.3 Kinesiology, Biomechanics, Neuroscience

A large number of experiments have been conducted on humans and animals to study motor learning [18]. One of the common hypotheses that has been drawn by the researchers from these experiments is that movements in humans are controlled by motor programs. Motor programs are parameterized, open-loop motions in abstract space that can be executed over time by the motor system. This hypothesis is supported by three reasons: the slowness of information processing states, the evidence for planning movements in advance, and the findings that deafferented animals and humans can often produce aspects of skilled actions without feedback. This is not to say that feedback is not used in movement, only that it is not strictly necessary.

Concerning the learning process, one of the non-disputed conclusions [18] is that learners appear to pass through various phases when acquiring a skill: a cognitive phase in which emphasis is on discovering what to do, an associative phase in which the concern is with perfecting the movement patterns, and an autonomous phase in which attentional requirements of the movement appear to be reduced or even eliminated. The actual theories of learning, however, are still the subject of disputes. One influential theory of motor learning is called the Schema Theory. It is based on the idea that slow movements are feedback based and rapid movements are motor program based. With learning, subjects develop rules (or schemas) that allow for generation of movements.

Despite suggesting high-level ideas and representations that can explain the accumulated experimental data, ideas in these fields lack the mathematical rigor and detail that is needed when one sets out to implement a solution.

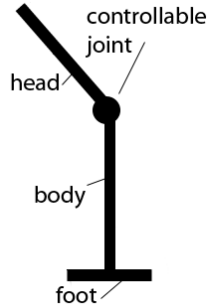


Figure 1.1: Acrobot character

1.3.4 Acrobot

We propose to control a planar, physically simulated character that consists of two cylindrical rigid bodies (links) connected by one actuated joint and an attached, fixed cylindrical foot, as shown in Figure 1.1. The joint between the body and the foot is held fixed. A character with this structure is commonly referred to in Computer Animation and Robotics as *Acrobot*. In previous work, the foot of the Acrobot is often held fixed in space above ground with the rest of its body hanging down, and the task is to use its actuated joint to swing up to a balanced position [3, 10, 19]. [2, 15] keep the foot free and develop an analytic solution for a hopping gait. However, analytic approaches to control laws were not demonstrated to scale to more agile motions such as flips and rolls. Instead, [11] achieves these motions by assuming perfect knowledge of forward dynamics, and formulating the control problem as a planning problem on move trees. The approach adopted in this work is not formalized as a planning problem and does not assume the a priori existence of a forward dynamics model. Instead, we gradually build a similar forward model by learning the dynamics through trial and error.

1.4 Contributions

The main contribution of this thesis is a learning algorithm that addresses the problem of learning whole-body motor skills. We designed our algorithm to explicitly model the process by which humans and animals acquire their motor skills. Specifically, the algorithm proceeds through three major phases: In the first phase, the character acquires a strategy for completing goals of a skill. In the second phase, the skill is perfected through many repeated trials. In the third, final phase, observations are generalized to form a compact model of the skill. We show that the algorithm can be used to learn a variety of controllable motor skills that are tightly integrated into a coherent system.

1.5 Thesis organization

Chapter 2 begins with description of a simple, planar character that we use for most of our experiments. This is followed by a description of the simulation environment and the control framework. We then describe the set of skills that the character will be trained for. In Chapter 3 we introduce the learning algorithm that is used to acquire every one of the skills. In Chapter 4 we present results of our experiments. We also present evidence for the scalability of our approach by applying the algorithm to a more complex character in 3D. Finally, conclusions, limitations, and future work are addressed in Chapter 5.

Chapter 2

System Overview

The goal of this work is to endow a character in a physical simulation with various skills through application of a learning algorithm. In §2.1 of this chapter, we introduce the parametrized family of characters that we use for this task. In §2.2 we discuss the control system that is used by to generate motion. Finally, in §2.3 we describe the set of skills that the characters will be trained for. We defer all details pertaining to the learning algorithm to Chapter 3.

2.1 Characters

We propose to control a planar, physically simulated character that consists of two cylindrical rigid bodies (links) connected by one actuated joint and an attached, fixed cylindrical foot, as shown in Figure 2.1. The joint between the body and the foot is held fixed. A character with this structure is commonly referred to in Computer Animation and Robotics as *Acrobot* [10]. Despite its apparent simplicity, Acrobot is capable of a wide variety of challenging, agile motions. Note that the simplicity of the character does not imply that the resulting control problem is easy. On the contrary, the character must use its single joint very precisely over time in order to accomplish all tasks.

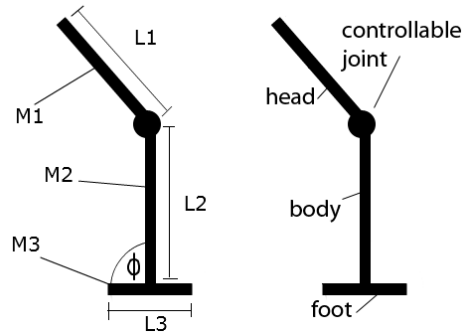


Figure 2.1: **Right:** The Acrobot character consists of three rigid body links: head, body and foot. The angle between the body and the foot is held fixed. The only controllable joint is the one between head and body link. **Left:** The character variations are obtained by varying the masses $M1$, $M2$, $M3$, the link lengths $L1$, $L2$, $L3$, and the angle between the foot and the body link, ϕ .

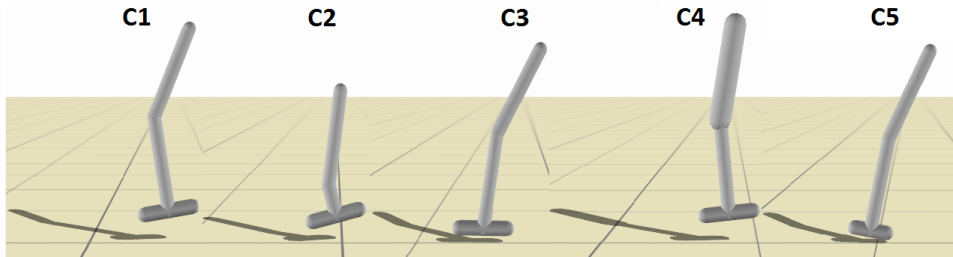


Figure 2.2: Character variations. This figure shows sample characters that we consider in this work. The first character on the left is our baseline character. The second character has a very short base link. The third character has a foot that is tilted by 0.11 radians to the right. The fourth character has a head link that is 1.5x heavier. The last character has a foot which is 33% shorter. We refer to our characters as C1 - C5, as indicated above each character.

The exact proportions of our character are left as free variables. Indeed, one of the strengths of the proposed framework is that it can learn controllers for a family of characters of this type. The characters we choose to experiment with are shown in Figure 2.2. Details of the parameters used for

2.2. Control

Name	Description	L1	L2	L3	M1	M2	M3	ϕ
C1	normal	0.6	0.6	0.3	5	5	1	0
C2	short base	0.6	0.2	0.3	5	5	1	0
C3	tilted base	0.6	0.6	0.3	5	5	1	0.11
C4	heavy head	0.6	0.6	0.3	8	5	1	0
C5	small foot	0.6	0.6	0.2	5	5	1	0

Table 2.1: Parameter values used for our characters. All lengths are given in meters, and all masses in kilograms.

each character are provided in Figure 2.1.

The configuration space of the Acrobot is described by $q \in \mathbb{R}^4$, $q = [x, y, \theta_1, \theta_2]$, where (x, y) is the absolute position of the head link, θ_1 is the absolute angle of the head link, and θ_2 is the relative angle between the head and body link. The state space is then described by $s \in \mathbb{R}^8$, $s = [q, \dot{q}]$.

Throughout this thesis, we will sometimes refer to the *rest state* of a character. This state describes the character being in the upright position, i.e. with head and base links vertical, and at rest. Whenever the character is re-initialized, or reset, its state is simply set to the rest state.

The **distance metric** $d_s(s_1, s_2)$ on the state space is a weighted L2 norm between two state vectors: $d_s(s_1, s_2) = \|w^T(s_1 - s_2)\|$. The weights are set manually through experimentation. We use $w = [1, 1, 1, 1, 0.1, 0.1, 0.1, 0.1]^T$. Even though it is difficult to justify any particular setting of the weights, our system is not very sensitive to this choice because of our scarce use of the state distance metric. In particular, it is required only when dealing with the recovery Hop mapping described in §2.3.1.

2.2 Control

Figure 2.3 shows a block diagram of the control system. The character moves by sequencing short, open-loop motions that we refer to as Motor

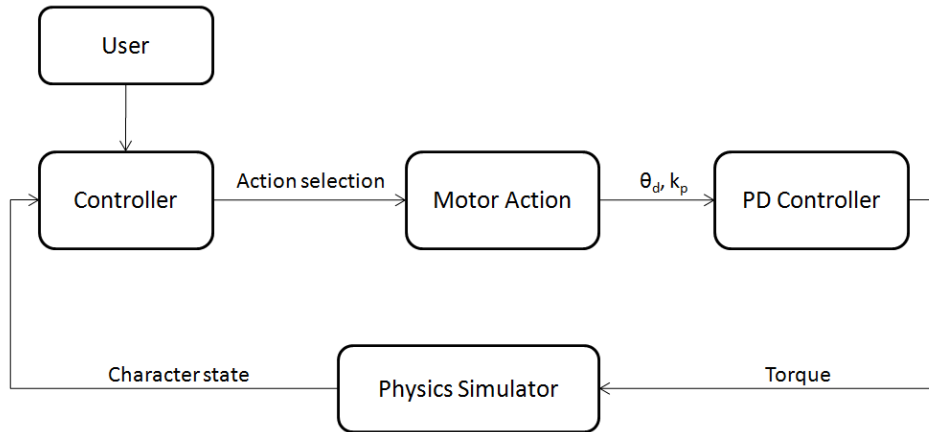


Figure 2.3: Block diagram of the control system.

Actions, or simply actions. Only one action can be active at any time. The Controller is responsible for initiating actions at the appropriate time based on what skill is currently active, based on events that occur in the world, and input received from the user. For example, when the user input indicates that the character should be hopping then the character controller initiates a new hop action every time when the foot hits the ground. While an action is active, it outputs two quantities over time: the desired angle between the head and body links (θ_d) and the force with which this value should be pursued (k_p). A PD-controller then calculates the torque that must be applied on the joint to meet the desired angle. Finally, the torque is applied as input to the Physical Simulator at each time step to produce motion. The simulator computes the accelerations using the equations of motion and these are then numerically integrated to updated the character state.

The PD-controller computes the torque on the actuated joint using

$$\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$$

where θ is the current angle of the joint, $\dot{\theta}$ is its instantaneous rate of change

2.2. Control

in time, θ_d is the desired angle, k_p is the spring coefficient, and k_d is the damping coefficient. Qualitatively, large values of k_p lead to stiff motions, and cause the character to exert large torques in order to meet the desired angle. It is desirable to add the second term in the equation above to prevent the system from oscillating around the desired value θ_d . To achieve this effect, the coefficient k_d penalizes large angular velocities. We fix $k_d = \sqrt{2 * k_p}$ in this work, as this ensures that the system is approximately critically damped.

An action can be formalized as a paired, piecewise constant function of time $A(t) : t \rightarrow (\theta_d, k_p)$. An example of an action is visualized in Figure 2.4. The number of pieces in every action is left as a design parameter, which we usually fix to be between 3 and 6 for simplicity. In general, fixing the number of pieces to be N allows us to think of every Motor Action as a $(3 \times N)$ -dimensional vector, because for every piece we need to specify its duration, as well as the (θ_d, k_p) over that time period. Thinking of Motor Actions in this way allows us to interpolate two actions by simply interpolating values separately along each dimension, as long as the number of pieces is the same.

The **distance metric** $d_a(a_1, a_2)$ on the action space is a weighted L2 norm between two action vectors: $d_a(a_1, a_2) = \|w^T(a_1 - a_2)\|$. The weights are set manually through experimentation. For values corresponding to time durations we use weights of 3, for desired angles we use weights of 1, and for k_p values we use 0.01. These choices were made by considering typical numerical ranges for these quantities.

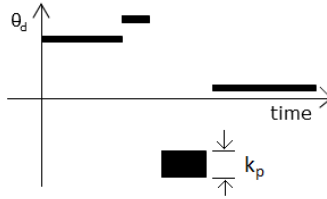


Figure 2.4: An example of a Motor Action with 4 pieces. The horizontal axis represents time, and vertical axis indicates the desired angle θ_d of the joint. The thickness of every piece corresponds to the spring coefficient k_p . This Motor Action causes one of our characters to hop when it is re-initialized by the controller every time when the character lands on the ground. Intuitively, the character first lowers its head (first 2 pieces), then stiffens up and quickly kicks its head backwards (third piece). During the fourth piece, the character relaxes as it anticipates impact with the ground.

2.3 Skills

We now give an overview of the skills that our characters will be trained for. We defer the description of learning to Chapter 3.

The majority of skills are parameterized according to a set of task parameters denoted by α , which specify aspects of the skills that the user has control over. The set of skills and their connectivity is shown in Figure 2.5. The base skill is the Hop, which allows the character to transition to a hopping motion from the rest state. The task parameter for the Hop skill, α_v , is the desired speed of the hop. Once the character is hopping, it can perform several other skills. The HopFlip skill allows the character to perform a back-flip with user-specified length, α_l , and height, α_h , and continue hopping immediately after landing. The HopRoll skill consists of the same set of task parameters, but results in a motion that more closely resembles a forward dive roll. The Stop skill allows our character to suddenly stop hopping and return to the rest state. From the rest state, the character can be commanded to hop again. Alternatively, while the character is in the rest state, it can also execute a Flip. The length of the flip is a user-controlled

2.3. Skills

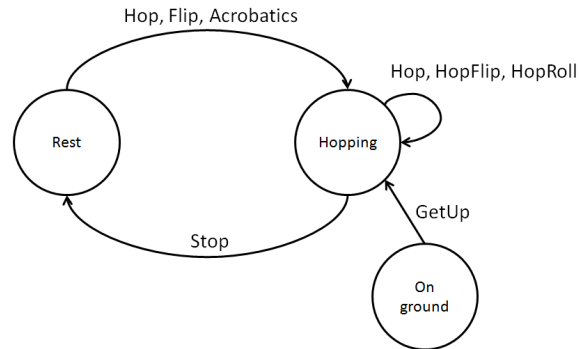


Figure 2.5: Skill transitions graph. Each node corresponds to a set of character states. Rest is the rest state. Hopping refers to any state where the character has just landed and made contact with left part of its foot. On ground refers to any state where the character is motionless on the ground. Every directed edge represents an action returned by a skill.

task parameter. If the character falls while executing any of the above skills, it can get back up using the GetUp skill. Finally, we consider a composite skill that we call Acrobatics. This skill can be executed from rest and involves a fixed sequence of the above skills executed in quick succession. Specifically, with Acrobatics the character starts from rest, and then uses Flip, followed immediately by HopRoll, and then HopFlip.

The function of a skill is to return a Motor Action that accomplishes the task parameters set by the user, while taking into account the current state of the character. For example, if the character is in the rest state and the user requests a Flip of length 2 meters, the character controller queries the skill for a Motor Action that accomplishes that task, and then initiates it. For some skills it is also necessary to know about the exact state of the character at the time when the action is about to be initiated. For example, when the character is hopping and the user requests a HopFlip of length 2 meters and height 2 meters, the appropriate action depends on the current state of the character while it is hopping. One of our insights is that it is not necessary to know the exact details of the state, because the set of states

2.3. Skills

it could be in is already constrained to a very particular part of the state space through the hopping skill. Therefore, the initial conditions of one skill can be summarized by the task parameters of the skill before it. In this particular example, the HopFlip skill must only know about the speed task parameter that is currently being used to generate the Hop.

Every skill can thus be formalized as a mapping $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$, where \mathbb{I} is the set of skill-specific parameters that describes the initial conditions, \mathbb{T} is a set of skill-specific task parameters, and \mathbb{A} is the Motor Action space. The mapping specifies the action that should be initialized by the controller in order to accomplish the goals given by the task parameters from the current initial conditions. Table 2.2 shows the initial conditions and task parameters used for every skill. Note that in this thesis the initial conditions set is either empty, in which case the character starts from the rest state, or it contains the speed of the hop, as motivated by the previous paragraph.

Skill	Initial Conditions	Task Parameters
Hop	$\{\}$	$\{\alpha_v\}$
Flip	$\{\}$	$\{\alpha_l\}$
GetUp	$\{\}$	$\{\}$
Stop	$\{\alpha_v\}$	$\{\}$
HopFlip	$\{\alpha_v\}$	$\{\alpha_l, \alpha_h\}$
HopRoll	$\{\alpha_v\}$	$\{\alpha_l, \alpha_h\}$
Acrobatics	$\{\}$	$\{\}$

Table 2.2: Initial Conditions and Task Parameters sets for all skills. α_v, α_l and α_h are labels for the speed, length, and height parameters, respectively.

2.3.1 Hop

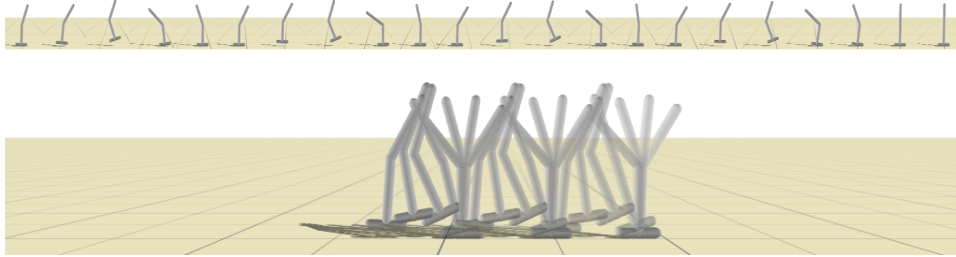


Figure 2.6: Hop skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The hop skill $H : \alpha_v \rightarrow \mathbb{A}$ returns an action that causes the character to hop if it is initialized every time when the foot strikes the ground. In addition, the hop skill can also be used from the rest state, in which case the character starts hopping. The user can change the speed of the hop interactively by changing the value of α_v . A value of $\alpha_v = 0$ makes the character hop slowly, and $\alpha_v = 1$ makes the character hop at the maximum speed. The exact speed achieved by the slowest and the fastest hop is not determined a priori. Instead, it depends on what the character acquires during the learning process.

It is the responsibility of the Hop skill to manage its own velocity to minimize the chance of falling. If the character is hopping at $\alpha_v = 0$, but suddenly the user changes the value to $\alpha_v = 1$, simply initializing the new action right away may cause the character fall over. To address this issue, we introduce a new variable β , which roughly corresponds to *carefulness* of the character. When $\beta = 0$, the character will execute a hop of any speed α_v that the user specifies. However, when $\beta = 1$, the character will ignore the user-defined α_v , and instead uses a recovery mapping $R : S \rightarrow \alpha_v$ to specify the value of α_v . The recovery mapping R predicts a value of α_v that most likely results in a successful hop, given the current state of the character. More generally, every time the character lands, the controller initiates a new

2.3. Skills

Hop action

$$H(\beta R(\vec{s}) + (1 - \beta)\alpha_v)$$

where \vec{s} is the state of the character and α_v is the user-defined desired speed.

A high-level controller can make effective use β to accomplish various tasks. For example, if it becomes very important that the character starts hopping as fast as possible, we can increase α_v , decrease β , and hope that the character does not fall. However, if we want the character to casually speed up, we can increase both α_v and β , and the character will slowly speed up without risking a fall. More importantly, whenever the character lands from a flip or a roll, the controller forces $\beta = 1$ regardless of the user-specified value for one hop. We refer to these hops as *recovery hops*, because they temporarily ignore the task parameters to maximize the probability of a successful recovery. In practice, we set $\beta = 0.05$ which is usually sufficient to guarantee that the character does not fall while hopping, even if the user quickly changes the value of α_v .

2.3.2 Flip

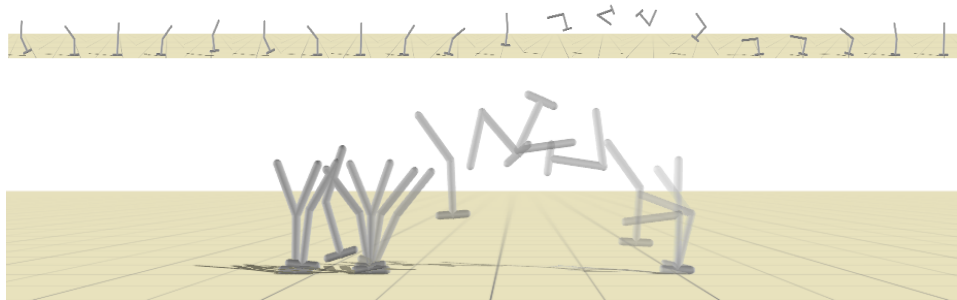


Figure 2.7: Flip skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The Flip skill $F : \alpha_l \rightarrow \mathbb{A}$ returns an action that causes the character to flip from the rest state. The length of the flip, $\alpha_l \in [0, 1]$ is controlled by the user. Low values of α_l cause the character to execute a short flip, while values close to 1 lead to long flips. On landing, the controller initializes a hop with $\beta = 1$ to force a single recovery hop.

2.3.3 HopFlip

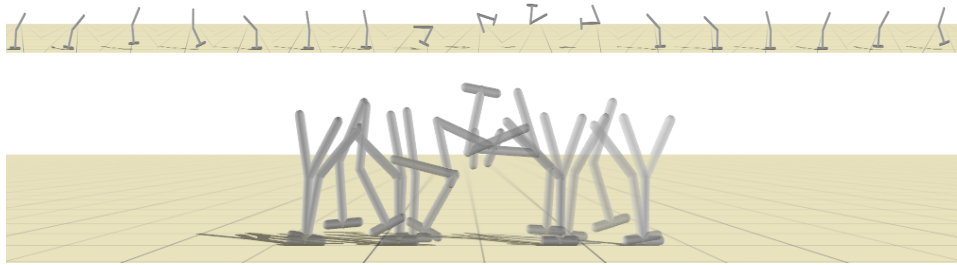


Figure 2.8: HopFlip skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The HopFlip skill $HF : \{\alpha_v, \alpha_l, \alpha_h\} \rightarrow \mathbb{A}$ returns an action that causes the character to flip while it is hopping at some speed α_v . The length of the flip, $\alpha_l \in [0, 1]$ and the height of the flip $\alpha_h \in [0, 1]$ are controlled by the user. As before, low values of α_l cause the character to execute a short flip, while values close to 1 lead to long flips. The user also has control over the height of the flips through the task parameter α_h , which works analogously. On landing, the controller initializes a hop with $\beta = 1$ to force a single recovery hop.

2.3.4 HopRoll

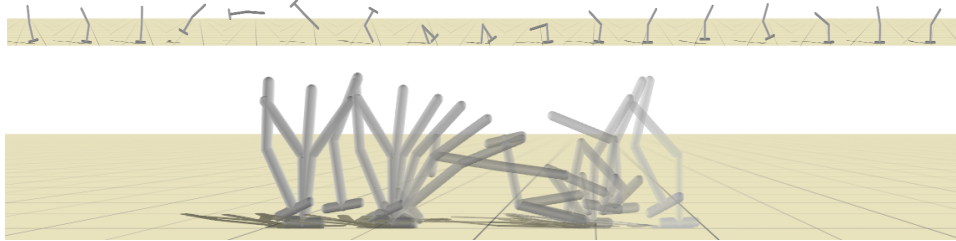


Figure 2.9: HopRoll skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The HopRoll skill $HF : \{\alpha_v, \alpha_l, \alpha_h\} \rightarrow \mathbb{A}$ works exactly as the HopFlip, which is described in 2.3.3. The only difference is in the style of the motion: Instead of a flip, the HopRoll skill causes the character to perform a forward dive-roll.

2.3.5 Stop



Figure 2.10: Stop skill. Character pose snapshots concatenated horizontally (from right to left) as the character stops.

The Stop skill $SS : \alpha_v \rightarrow \mathbb{A}$ returns an action that causes the character to stop from a hop of some speed given by α_v . If the character is hopping too quickly to stop in a single stop action, the controller first lowers the speed of the hop, and then initiates a Stop action once it becomes available. The highest speed limit from which a Stop action can be successfully executed is determined during the learning process.

2.3.6 GetUp

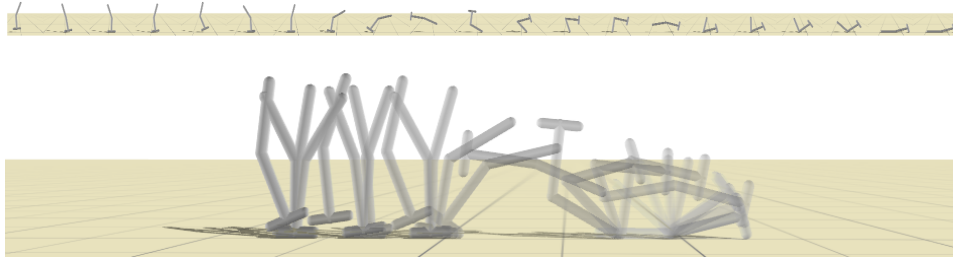


Figure 2.11: GetUp skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The GetUp skill can be used by the character to recover back into a hopping motion if it happens to fall. Even though the character can be in slightly different positions while it is on the ground, only a single Motor Action is required to bring it back to its feet. The character can achieve this by learning an action that consist of two parts: The first segment of the action usually brings the character to some specific static pose on the ground, and the second segment causes it to get up from that pose. Therefore, even if the character begins in a slightly different configuration, by the end of the first segment it will always be in the same state. The GetUp skill therefore consists of only two actions: one for getting up if the character falls to the left, and one for getting up if it falls to the right. The controller is programmed to execute the appropriate action by detecting if the character fell to the left or right.

2.4 Composite skills

It is possible to naturally extend the idea of skills into a higher level of abstraction to explore more complicated, composite skills. Every skill discussed in the previous section is of the form $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$, where the output is a

vector that specifies a Motor Action. The output of a composite skill is also a vector, but the numbers are instead interpreted as the initial condition and task parameters of other skills, which then get translated into Motor Actions accordingly. We only experimented with one composite skill and describe it in more detail below.

2.4.1 Acrobatics

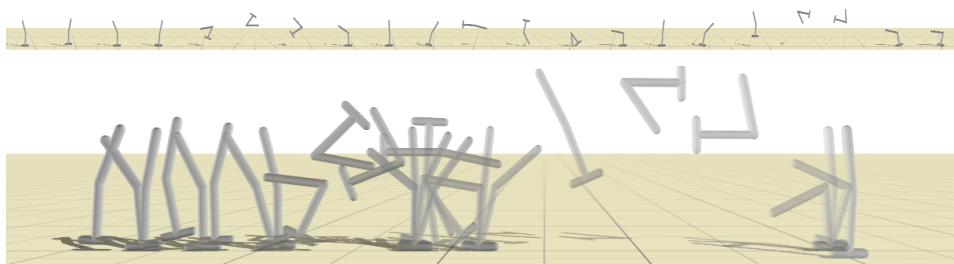


Figure 2.12: Acrobatics composite skill. Top: character pose snapshots concatenated horizontally from right to left. Bottom: character pose snapshots, with a stationary camera and overlaid frames.

The Acrobatics skill initializes a specific sequence of actions over time that cause the character to perform a continuous sequence of acrobatic maneuvers starting from rest. More specifically, Acrobatics initializes a Flip from rest, followed immediately by HopRoll, and then immediately by HopFlip. After landing from the HopFlip, the controller initiates one recovery hop.

Recall from Section 2.3 that the flip skill is defined as $\alpha_l \rightarrow \mathbb{A}$, and that HopRoll and HopFlip are both defined as $(\alpha_v, \alpha_l, \alpha_h) \rightarrow \mathbb{A}$. The Acrobatics skill consists of one composite action $(\alpha_l^1, \alpha_v^2, \alpha_l^2, \alpha_h^2, \alpha_v^3, \alpha_l^3, \alpha_h^3)$ where α_l^1 is used to execute the Flip, $\alpha_v^2, \alpha_l^2, \alpha_h^2$ are used for the subsequent HopRoll, and $\alpha_v^3, \alpha_l^3, \alpha_h^3$ are used for the HopFlip that immediately follows. The Motor Action therefore becomes the concatenation of $F(\alpha_l^1)$, $HR(\alpha_v^2, \alpha_l^2, \alpha_h^2)$, and $HF(\alpha_v^3, \alpha_l^3, \alpha_h^3)$ in time.

Note that both HopRoll and HopFlip are meant to work from a steady

2.4. Composite skills

hopping motion of some speed and are therefore not generally expected to work from the initial conditions that come up during the Acrobatics sequence. Specifically, the HopRoll must be executed right after landing from a Flip, and the HopFlip right after landing from the HopRoll. Nonetheless, we shall show that by choosing the parameters of both skills appropriately, it is almost always possible to successfully string these skills together. The computational advantage of formalizing the Acrobatics skill through parameters of other skills instead of simply one long Motor Action will become apparent when we discuss the learning algorithm in more detail. We offer additional remarks concerning this issue in the Results section.

Chapter 3

Learning

3.1 Overview

We take inspiration from nature when designing our learning algorithm. Animals and humans go through stages of exploration when mastering a new skill. Consider how a human could go about learning a flip: Through observation, we first get a sense of the gist of the motion. What does a flip look like? Approximately how could one go about doing a flip? We then begin to try many variations of some initial guess, until we finally accomplish our first flip after many iterations of trial and error. We then begin to explore small variations of our successful trial, until we converge on a motion that reliably causes us to flip. If our goal is to control a particular aspect of the flip, such as its length, we can proceed further. We could attempt variations of our initial flip, and try to come up with motions that lead to longer flips. Finally, after getting a sense of how the flip length varies with the motions we attempt, we can try to generalize and come up with a rule on how to accomplish longer flips. In our example, we must push off harder from the ground and extend our arms further while in the air in order to compensate for our higher angular momentum. Finally, suppose that our goal was to flip while running. The natural way to learn this skill would be start with what we know: we already know how to flip from rest. We can then slowly start to attempt flips while moving at incrementally quicker pace, until eventually learning how to flip while running quickly. In summary, attempting to learn how to flip while running is nearly impossible from scratch, but the task becomes significantly easier by going through incrementally harder stages of learning.

This thesis attempts to formalize the intuition described above in a staged learning algorithm, which we use to learn all skills described in Section 2.3 one by one. The skills are learned in the following order: Hop, Flip, GetUp, Stop, HopFlip, HopRoll, and finally Acrobatics. The learning is done in an online, active learning setting and proceeds as follows: every skill is initialized with a seed Motor Action that at least approximately accomplishes the goal of the skill. For the Flip skill, this action could be a flip that almost works. Phase 1 of learning runs a stochastic greedy local search to find the first Motor Action that accomplishes the goal of the skill without falling. In our example, this corresponds to a motion that successfully leads to a flip and a subsequent recovery through recovery hops. Phase 2 of learning explores similar motions and collects a large set of actions that are all successful, but possibly with different outcomes. Finally, in phase 3 we generalize from our observations by fitting linear models to the actions found in phase 2. This corresponds to finding a rule of thumb on how to accomplish flips of various sizes.

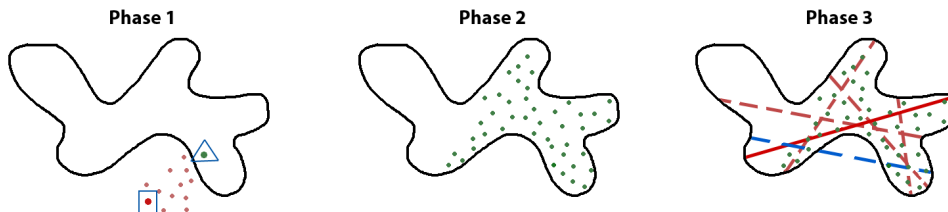


Figure 3.1: Stages of the Learning Algorithm. This diagram visualizes one execution of the learning algorithm for some skill. Bounded by the black contour is the set of Motor Actions that are successful from some initial parameters setting. The algorithm is initialized with a seed action (boxed in red dot). In phase 1, stochastic greedy local search finds the first successful action (green dot inside triangle). Starting with this action, phase 2 explores its variations to collect a large database of successful actions (green dots). Finally, phase 3 attempts to fit a model to this data to generalize from the observations. Specifically, in this thesis we consider linear models. Several candidate models (light red dashed lines) are collected and the best one is returned according to some skill-specific criteria (dark red line).

3.2 Trials

The character learns every skill by measuring outcomes of actions through repeated trials in simulation. A trial is labeled as being successful if the character completes a skill-specific training sequence without falling. A result of a successful trial is a tuple $E = (I, A, T)$ that we refer to as an Experience. It encodes the observation that starting from a character state generated by initial condition parameters $I \in \mathbb{I}$ and applying Motor Action $A \in \mathbb{A}$ results in task parameter outcome $T \in \mathbb{T}$.

An example of an Experience while learning HopFlip is a 3-tuple $(0.3, A, (1.2, 1.5))$, where A is some Motor Action. It states that when we hop with $\alpha = 0.3$ and then execute A the next time we land on the ground, it will cause us to flip 1.2 meters forward, and at the highest point of the motion we will be 1.5 meters off the ground. Furthermore, after landing from the flip, executing a few recovery hops will lead us to recover balance.

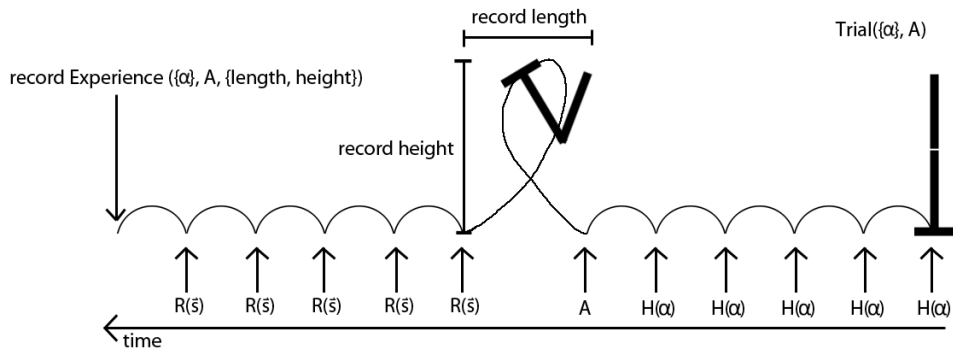


Figure 3.2: An example of a trial for learning the HopFlip skill. The character is initialized in the rest state and starts to execute skills according to the training sequence for HopFlip. First, the Hop mapping H is used to hop 5 times using the provided α parameter. Next, the HopFlip is attempted using the action that is currently being investigated. The last 5 hops are executed as recovery hops and use the Hop recovery mapping $R(\vec{s})$, where \vec{s} is the state of the character every time it lands. Arrows indicate the points at which a Motor Action is initialized. Since the character did not fall, the Trial is marked as being successful, and the resulting Experience is returned.

3.2. Trials

In general, trials can be thought of as a function $\mathbb{I}, \mathbb{A} \rightarrow \mathbb{T}$. In other words, they measure the outcome of an action from some initial conditions. During phase 2, the character accumulates a large database of Experiences $\mathbb{E} = \{E_i\}$ by conducting many trials. Collectively, the experiences form a tuple-based dynamics model, which is a common strategy for modeling dynamical systems in general [4, 6]. This database can later be used to perform high-level tasks, because it can be inverted to produce a model of the form $\mathbb{I}, \mathbb{T} \rightarrow \mathbb{A}$. In other words, given some initial conditions, what action should be initialized to meet some desirable goals specified in the task parameters? For example, if we find ourselves hopping at some speed and wanting to jump over a gap ahead using HopFlip, what action should we initialize to meet this goal?

The training sequence that describes the order of skills that should be executed during every trial is generally different for each skill. For example, as shown in Figure 3.2, the training sequence for HopFlip is to Hop 5 times at some speed, attempt a HopFlip, and then attempt 5 recovery hops on landing. A full listing of training sequences for every skill can be found in Table 3.1.

Skill	Training sequence
Hop	Hop 20 times
Flip	Flip, 5 recovery hops
GetUp	Hop 2 times, (push), Getup, 5 recovery hops
Stop	Hop 5 times, Stop
HopFlip	Hop 5 times, HopFlip, 5 recovery hops
HopRoll	Hop 5 times, HopRoll, 5 recovery hops
Acrobatics	Flip,HopRoll,HopFlip, 5 recovery hops

Table 3.1: Training sequences for all skills. If the character executes the entire sequence without falling, the trial is considered to be successful. For GetUp, (push) indicates that the character is suddenly pushed with significant force in a random direction. In this case, its objective is to recover back to hopping motion.

3.2.1 Motor Action reliability

A crucial element of our approach is that during the learning process we explicitly look for actions that not only successfully complete the training sequence, but actions that can reliably do so even if subjected to small disturbances in both the initial conditions and the action itself. For example, when learning a skill such as a flip, humans will explicitly look for motions that can make them land flat on both feet. This particular outcome is desirable because it is inherently robust toward small disturbances in initial conditions and errors in the motion itself. Similarly, a motion that only barely makes us flip and causes us to land off balance should likely undergo further refinement.

We explicitly incorporate the concept of reliability into our learning framework. While we cannot implement high-level intuitions for evaluating actions (such as detecting when we fall off-balance) it is still possible to approximate the reliability of an action by explicitly measuring the robustness of its outcome toward small changes in initial conditions and the action itself. As mentioned in the previous section, Trials enable us to test an outcome of some action under some initial conditions. If we define $Successful(A, I)$ that returns 1 if action A succeeds from initial conditions I and 0 if the character falls, then we can formalize this notion by defining reliability as:

$$Reliability(A, I) := \frac{\sum_{i=1}^N Successful(A + \Delta A, I + \Delta I)}{N}$$

where N is large, and ΔA , and ΔI , are drawn from an appropriate noise distribution. Given an Experience (I, A, T) , we could compute an estimate of its reliability according to the above definition by running many Trials with slightly different actions and initial conditions. However, in order to improve the efficiency of our algorithm, we will instead approximate the reliability of every action online, during phase 2 of the learning algorithm.

3.3 Learning algorithm

In this section we describe the core learning algorithm that is used to train every skill. The goals of each phase, using the terminology defined in the previous section are as follows:

- In **phase 1**, we find the first successful Experience $E_0 = (I_0, A_0, T_0)$.
- In **phase 2**, we extend the Experience into a large set of Experiences $\{E_i\} = \{(I_i, A_i, T_i)\}$ through exploration.
- In **phase 3**, we generalize from the observations made in phase 2 by fitting a model $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$ to the database of Experiences \mathbb{E} . This model can be later used to accomplish high-level tasks.

3.3.1 Initialization

The algorithm is initialized with a seed Motor Action A_{init} and some Initial Condition parameters I_0 that are needed for the particular training sequence. Ideally, I_0 should contain parameters that result in the easiest initial conditions. In the case of hopping, the easiest initial conditions are to hop as slowly as possible (i.e. $I_0 = \{\alpha_v = 0\}$).

3.3.2 Phase 1: initial successful action generation

The goal of phase 1 is to find the first successful Experience (I_0, A_0, T_0) . We assume that we are given A_{init} , I_0 , and a Phase 1 reward function for every skill. The Phase 1 reward function assigns a score to every failed trial, which helps guide the search. We run a stochastic greedy local search, starting with the seed action A_{init} as our initial guess. In this setting, conducting a single trial corresponds to one function evaluation. We keep attempting variations of A_{init} with the highest reward until we reach an action that is successful. The procedure is summarized in Algorithm 1.

The phase 1 reward functions are skill specific:

3.3. Learning algorithm

- For **Hop**, the reward function returns the number of hops that were successfully executed before a fall.
- For **Flip, HopFlip and HopRoll**, the reward function returns the difference of net rotation undergone by the character from a full circle.
- For **GetUp**, the reward function measures the angle of the body link and its velocity, and returns high rewards for near-vertical angles and low velocity at some point during the action.
- For **Stop**, the reward function measures the amount by which the Stop action decreases the velocity of the character.
- For **Acrobatics**, the reward function measures the number of skills that were successfully executed before a fall and returns this number as the reward.

3.3. Learning algorithm

Algorithm 1 Phase 1: Initial successful action generation

Input:

$I_0 \leftarrow$ Initial Parameters

$A_0 \leftarrow$ Seed action

Output:

Successful Experience (I, A, T)

```
1:  $(T, success, reward) \leftarrow Trial(I_0, A_0)$ 
2: if sucess then
3:   return  $(I_0, A_0, T)$ 
4: end if
5:  $reward\_best \leftarrow reward$ 
6:  $A \leftarrow A_0$ 
7: loop
8:    $A_{candidate} \leftarrow A + \Delta A$ 
9:    $(T, success, reward) \leftarrow Trial(I_0, A_{candidate})$ 
10:  if success then
11:    return  $(I_0, A_{candidate}, T)$ 
12:  end if
13:  if  $reward > reward\_best$  then
14:     $reward\_best \leftarrow reward$ 
15:     $A \leftarrow A_{candidate}$ 
16:  end if
17: end loop
```

In the above algorithm, $Trial(I, A)$ executes a single Trial, as described in 3.2. If the training sequence contains hops before the action that is being investigated, all hops are executed using the same speed parameter α_v provided in I_0 .

In practice, we restart the search if $reward_best$ does not improve for some number of iterations to minimize the chance of getting stuck in local optima. In addition, a small set of Experiences is collected in this phase instead of only a single one to decrease the probability of hitting a bad local optimum with the first successful Experience.

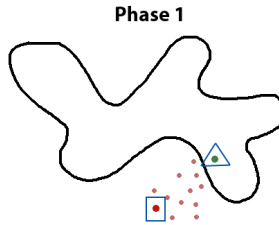


Figure 3.3: Phase 1 of the learning algorithm. The set of successful actions for some initial conditions is bounded by the given black contour. We are given a seed action (boxed in red dot), and explore its variations (light red dots), guided by the phase 1 reward function. Phase 1 ends when the first successful action is found (green dot inside triangle).

3.3.3 Phase 2: exploration

Having acquired the first successful Experience, phase 2 of the learning algorithm incrementally grows the set $\mathbb{E} = \{E_i\}$. This stage corresponds to exploring variations of successful motions and their effects on the task parameters.

The exploration process is shown in detail in Algorithm 2. First, we select a specific promising experience $E_k = (I_k, A_k, T_k) \in \mathbb{E}$. The details of this choice are discussed later. We then change the initial parameters I_k and the Motor Action A_k by adding sparse gaussian noise to obtain slightly different initial conditions I_{new} and action A_{new} . Next, a Trial is attempted using I_{new} and A_{new} . If the trial is successful, we record the outcomes of the trial in the task parameters T_{new} , and add the Experience $(I_{new}, A_{new}, T_{new})$ to the set of all Experiences \mathbb{E} . If the trial was unsuccessful, its results are discarded. One could imagine making use of this information in some way to avoid recomputing a similar trial in the future, but we defer this extension to future work. Finally, based on the outcome of a trial we also update our estimate of the reliability of E_k . Reliability is thus computed over time by simply keeping track of how many times a variation of some Experience was attempted, and how many times that variation was also successful. After updating the reliability statistics, we pick another promising experience and

3.3. Learning algorithm

repeat the procedure.

Algorithm 2 Phase 2: Exploration

Input:

$(I_0, A_0, T_0) \leftarrow$ Initial successful Experience from phase 1
 $N \leftarrow$ Number of Experiences to collect

Output:

Successful Experience set \mathbb{E}

```
1:  $\mathbb{E} = \{(I_0, A_0, T_0)\}$ 
2: while  $|\mathbb{E}| < N$  do
3:    $(I_k, A_k, T_k) \leftarrow$  pick promising Experience from  $\mathbb{E}$ 
4:    $N_{tried}_k \leftarrow N_{tried}_k + 1$ 
5:    $I_{new} \leftarrow I_k + \Delta I$ 
6:    $A_{new} \leftarrow A_k + \Delta A$ 
7:    $(T_{new}, success) \leftarrow Trial(I_{new}, A_{new})$ 
8:   if success then
9:      $\mathbb{E} \leftarrow \mathbb{E} \cup \{(I_{new}, A_{new}, T_{new})\}$ 
10:     $N_{successful}_k \leftarrow N_{successful}_k + 1$ 
11:   end if
12: end while
13: return  $\mathbb{E}$ 
```

A good definition of a promising Experience is crucial to the success of the algorithm. On one hand, it is desirable to explore experiences that are known to be reliable in hopes of discovering more. On the other hand, we also want to consider newly found experiences, as they may prove to be reliable in the future. We also want to collect Experiences from many initial conditions that result in a wide variety of task parameters. Lastly, we must be cautious to not over-explore a certain part of the space. Algorithm 3 addresses these tradeoffs.

3.3. Learning algorithm

Algorithm 3 Picking promising Experience.

Input:

$\mathbb{E} \leftarrow$ Successful Experience set
 $Ntried_i, Nsuccessful_i$ reliability statistics for each Experience in \mathbb{E}
 $q \in [0, 1] \leftarrow$ Exploration-Exploitation threshold
 $n \in \mathbb{N} \leftarrow$ threshold for what counts as new Experience
 $maxTry \in \mathbb{N} \leftarrow$ maximum number of times an Experience is explored
 $\sigma_1, a_1, \sigma_2, a_2 \in \mathbb{R}^+ \leftarrow$ parameters

Output:

Promising Experience E

```

1: if  $U(0, 1) < q$  then
2:   return Random Experience  $E_i \in \mathbb{E}$  such that  $Ntried_i < n$ 
3: end if
4:  $I_r \leftarrow$  Random initial parameters
5:  $T_r \leftarrow$  Random task parameters from region of interest
6: for  $E_i \in \mathbb{E}$  for which  $Ntried_i \leq maxTry$  do
7:    $Score_i \leftarrow \frac{Nsuccessful_i}{Ntried_i} + a_1 e^{-\frac{\|I_i - I_r\|^2}{2\sigma_1^2}} + a_2 e^{-\frac{\|T_i - T_r\|^2}{2\sigma_2^2}}$ 
8: end for
9:  $Roll_i \leftarrow U(0, Score_i)$ 
10: return  $\underset{E_i \in \mathbb{E}}{\operatorname{argmax}}(Roll_i)$ 

```

In the above algorithm, we usually set $q = 0.5$, $n = 4$, $maxTry = 30$, $\sigma_1 = 0.5$, $\sigma_2 = 0.5$, $a_1 \sim U(0, 0.3)$ and $a_2 \sim U(0, 0.3)$. $U(a, b)$ is the uniform random distribution. Overall, the algorithm tends to pick reliable Experiences as promising. The second and third terms in the equation for $Score_i$ ensure that we explore actions from all possible initial conditions that achieve rich set of task parameters. As per line 2, we also choose to explore actions that are new to gather significant statistics of their reliability. No action is explored too often due to the hard upper bound given by $maxTry$.

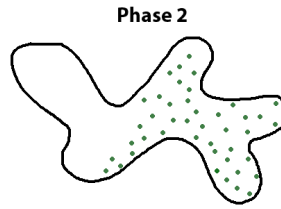


Figure 3.4: Phase 2 of the learning algorithm. The set of successful actions for some initial conditions is bounded by the given black contour. We explore the set of successful experiences starting from the first successful experience (dark green) by repeatedly trying variations of experiences in trials. Note that every point in this diagram can only be interpreted as a Motor Action for Flip, Hop, and Acrobatics because these skills don't have any initial condition parameters. However, in general $\mathbb{I} \times \mathbb{A}$ is the space that undergoes exploration in this phase. The reader can imagine the variables in I forming another axis that extends outwards, such that only a particular slice for some initial conditions is shown in this diagram.

3.3.4 Phase 3: parameterization and generalization

Once we have accumulated a large database of successful Experiences in phase 2, we can develop an inverse model, $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$. This mapping is desirable because it can be used by the character to accomplish high-level tasks. For example, given that we want to flip across a 2-meter gap, what Motor Action should be executed to accomplish this goal?

In this work we only experiment with 1-dimensional linear models for Hop and Flip skills in phase 3. That is, for the Hop skill we fit a linear model $\alpha_v \rightarrow \mathbb{A}$ such that $\alpha_v = 0$ and $\alpha_v = 1$ result in slow and fast hops, respectively. For Flip, we fit a linear model $\alpha_l \rightarrow \mathbb{A}$ such that $\alpha_l = 0$ and $\alpha_l = 1$ result in short and long flips, respectively. Since both skills don't have any initial parameters because they start from the rest state, this task can simply be viewed as that of fitting a line to data in the action space, as illustrated in Figure 3.5. More generally, for a set of N task parameters, a linear model can be constructed using either:

- a linear weighting of $N + 1$ examples, or

3.3. Learning algorithm

- defined according to $A = A_0 + M\vec{\alpha}$, where M is a matrix and $\vec{\alpha}$ is the vector of task parameters.

For every skill other than Hop and Flip, we stop after phase 2 and then construct a non-parametric model. This is done by first partitioning the volume of space $\mathbb{I} \times \mathbb{T}$ into cubes of some small size (we use 0.05). According to the collected experiences \mathbb{E} , we then map each cube to the most reliable Motor Action that was found in that part of space. In either case, the Experiences \mathbb{E} that were collected in phase 2 can then be discarded.

Some considerations should be made when fitting a model to these kinds of datasets. One difficulty is that there are usually many Motor Actions that all accomplish the goals of the skill, but do so with a particular style of motion. The output of phase 2 therefore contains a mix of many kinds of motions. To deal with the large number of outliers in the data, it becomes important to only consider robust fitting methods. In addition, instead of only fitting one model we collect a number of candidate models and later evaluate their performance to pick the best one.

Algorithm 4 Phase 3: Generalization

Input:

- $\mathbb{E} \leftarrow$ successful experiences from phase 2
- $N \leftarrow$ number of candidate models to collect

Output:

- Best model M

- 1: $L \leftarrow \{\}$
 - 2: **for** $i = 1$ to N **do**
 - 3: $M \leftarrow$ fit model $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$
 - 4: **if** M generalizes **then**
 - 5: $L \leftarrow L \cup \{M\}$
 - 6: **end if**
 - 7: **end for**
 - 8: **return** $\operatorname{argmax}_{M_i \in L}(\operatorname{Score}(M_i))$
-

3.3. Learning algorithm

The fitting procedure (line 3 in Algorithm 4) is carried out using a method that is inspired by the RANSAC algorithm to deal with the high number of outliers in our data. The fitting procedure constructs the model as follows: first, a line is constructed through two randomly chosen actions from the data. Then, the distance from every other action to this line is found, and the sum of the 10% of the lowest distances is computed to produce a robust estimate of the evidence for this model. In this context, small values for the final sum indicate more evidence for a model. We keep selecting pairs of actions and repeating this procedure some number of times (usually about 30), and finally set the output of the procedure to the line that achieved the lowest sum.

The next step is to test the generalization of a proposed model (line 4 in Algorithm 4). The test is carried out by systematically sampling points along the line. If any point somewhere in the middle between the two actions results in a failed trial, the generalization test fails. If the model passes the generalization test, we extrapolate along the line in both directions for as long as the resulting actions lead to successful trials, and in the end keep the two actions A_1, A_2 at the boundary of what was found to be successful in both directions. The final candidate model then becomes $A(\alpha_v) = \alpha_v A_1 + (1 - \alpha_v) A_2$. We repeat the entire procedure until a significant number of candidate models is gathered and then proceed to score them.

Since the scoring procedure is specific to each skill, we discuss them separately below.

Hop phase 3 model scoring function

The candidate models for the Hop skill are scored according to two desirable properties. First, a good model covers a large portion of the task parameters. In context of a Hop, this is equivalent to requiring that the difference between the slowest and fastest hop generated by the model is large. In addition,

3.3. Learning algorithm

a good model results in a Hop that is robust to changes in the speed. We investigate this robustness for every one of the candidate models as follows. The character starts hopping with $\alpha_v = 0.5$, and every hop the speed is changed according to $\alpha_v \leftarrow \min(1, \max(0, \alpha_v + N(0, 0.25)))$ with a 50% probability. Here, N is the Normal distribution. If the character ever falls, it is reset and the evaluation continues. A high score is given to the character that falls the least number of times during a fixed-length evaluation. In the end, the two contributions are weighted together, and the model with highest variance in task parameters and the highest robustness to change in speed is returned.

Flip phase 3 model scoring function

For Flip, models that cover a large portion of task parameters are also preferred. In context of a Flip, this corresponds to requiring that the difference between the smallest and largest flip is large. In addition, the transition of the Flip into a Hop on landing should be as robust as possible. As a good quantitative correlate of this intuition, we record the variance in the speed of the 5 recovery hops that follow the flip. If the variance is small, it is likely that the transition was successful and the action receives a large score. In the end, the two contributions are weighted together, and the model with highest variance in the task parameters and the best transition to a Hop is returned.

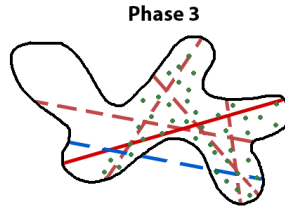


Figure 3.5: Phase 3 of the learning algorithm. In this phase, we attempt to discover a model of the form $(\mathbb{I} \times \mathbb{T}) \rightarrow \mathbb{A}$ from the experiences collected in phase 2. Every model is tested according to some skill-specific generalization criteria, and then scored. The highest scoring model is returned as output. In this work, we experimented with finding a linear manifold of successful actions. Linear models (dashed light red lines) are repeatedly fit to our dataset, and subsequently tested for generalization. The blue dashed line in this diagram is a model that fails to generalize. In the end, we evaluate all candidate models, and pick the best one (dark red line) as output according to skill-specific criteria.

3.3.5 Learning the recovery model for Hop

Unlike other skills, we also learn a recovery model for the Hop skill, which allows us to recover from arbitrary landings. As discussed in 2.3.1, the recovery model is a mapping $R : S \rightarrow \alpha_v$ that predicts the α_v of the Hop action that most likely leads the character toward a stable hopping motion from some state. To learn the recovery model for the Hop skill we proceed as follows. While the character is hopping using the learned Hop skill, the speed of the hop is changed according to $\alpha_v \leftarrow \min(1, \max(0, \alpha_v + N(0, 0.25)))$ with a 50% probability every hop. On every landing, we store the character's state together with the parameter α_v that will be used for the next hop. If the character ever happens to fall, we discard the latest 3 measurements. The resulting database of (\vec{s}, α_v) forms the non-parametric recovery model for the Hop skill. Intuitively, the character remembers the states that it usually encounters while hopping at different speeds. If it needs to transition to a hop from an arbitrary state, the best guess for the α_v that should be used during the recovery attempt can be extracted from the table by finding the nearest neighbor of the current state, and using the corresponding α_v .

Chapter 4

Results

We applied our algorithm to learn skills for two different characters. We first discuss results pertaining to the planar Acrobot character. We then introduce a more complicated 3D simulated quadruped and demonstrate that the algorithm can be used to train the character to perform parameterized leaps from a trotting gait.

We used a 2.7GHz computer for all of our experiments. The simulation-based system was built using the Open Dynamics Engine (ODE) as the physics engine. The time step frequency was set to $2000Hz$. This allows us to simulate the Acrobot character 30x faster than real time, and the dog character about 3x faster than real time. Given this setup, about 10-20 trials can be evaluated per second with the Acrobot character, and about 1-2 trials per second with the dog character.

4.1 Acrobot

We were able to successfully learn all skills on the base Acrobot character (C1) using our learning algorithm. The seed Motor Actions that must be provided as input to each skill in phase 1 took less than a few minutes to create in each case, although more complex characters might require more finely tuned actions. Afterwards, we were able to run the entire algorithm on all four variations of the base character (C2, C3, C4, C5) without any parameter changes, with the exception of having to change the seed Motor Action on a few occasions. We defer a more detailed treatment of these modifications to the Discussions section. The learning algorithm was run for about 10 hours on every character. In our experiments, phase 1 usually

completes in a few seconds and never lasts more than one minute. The majority of the computation is taken up by phase 2, and phase 3 for the Hop and Flip skills. As with many other online algorithms, the results progressively improve when the algorithm is allowed to run for longer periods of time. We experimented with shorter learning times by collecting a smaller set of actions in phase 2 and by restricting the number of candidate models that are evaluated in phase 3. Under these constraints, the learned skills become generally less reliable and tend to cover a smaller range of task parameters.

The end product of the learning algorithm can be summarized for every Acrobot character as consisting of:

- A model for the **Hop** skill, linearly parameterized for speed (α_v). This is constructed from two hop actions and their known speeds. In addition, the Hop skill contains the recovery model, which is represented as a table of (\vec{s}, α_v) pairs, where $\vec{s} \in S$ is a state vector.
- An analogous linear model of the **Flip** skill, constructed from two flip actions and their respective lengths.
- For **HopFlip**, **HopRoll**, **Stop**, the output is a lookup table that stores the most reliable action for cubes with side lengths 0.05 in the appropriate $\mathbb{I} \times \mathbb{T}$ space. For HopFlip and HopRoll, this space consists of the parameters $\alpha_v, \alpha_l, \alpha_h$. For Stop, this space only consists of one dimension, α_v .
- For **GetUp and Acrobotics**, the skill only contains the most reliable action that was found in phase 2.

We now proceed to discuss the details of the results pertaining to all skills.

4.1.1 Hop and Flip skills

The results from learning of the Hop and Flip skills are visualized Figure 4.1. The figure summarizes the extent of capabilities that were learned for

all characters. Looking at the achieved ranges of speeds and lengths for the Hop and the Flip, it is clear that the most challenging characters to control for these skills are C3 and C5. It is important to note that characters were often capable of performing each skill with task parameters outside of the ranges given in the figure. All results listed are merely the ones derived from the highest-scoring linear model that was found in each case. In particular, note that a simple linear model performs poorly for C5 on the Flip skill. This indicates that a linear model is too weak to capture the challenging dynamics of this skill for that character. In this case, it may be better to instead only use the non-parametric model that derives from the experiences collected in phase 2.

4.1. Acrobot

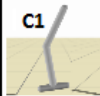
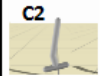

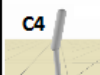
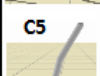
	HOP				FLIP		
	min speed [m/s]	max speed [m/s]	range [m/s]	speed change fail [%]	min length [m]	max length [m]	range [m]
 C1	0.53	1.92	1.39	0.24	1.6	2.63	1.03
 C2	0.35	1.47	1.12	0.08	1.72	2.73	1.01
 C3	0.47	1.06	0.59	0.33	1.18	1.87	0.69
 C4	0.26	1.08	0.82	2.2	2.11	3.34	1.23
 C5	0.49	0.88	0.39	0.19	2.31	2.75	0.44

Figure 4.1: Results for the Hop and Flip skill. For both skills and all characters, we fit a linear model in phase 3 and examine the range of task parameters that are covered by these models. On left, min and max speed for the Hop (corresponding to speeds of $\alpha_v = 0$ and $\alpha_v = 1$ respectively) is shown in meters per second. The computed range is also shown for viewer’s convenience. Speed change fail % is the percentage of hops that lead to a fall during the phase 3 candidate model scoring procedure discussed in §3.3.4, where robustness of the Hop to changes in speed is estimated. Analogously, on the right we show the smallest and largest Flip that each character can generate in meters.

4.1.2 Stop skill

The Stop skill was found to be relatively easy to learn for all of our characters. Phase 1 of the learning algorithm usually terminates after a single trial because to stop from a very slow hopping motion, the character can almost always simply stiffen up in the upright position and wait until it settles. Recall that the Stop skill is a single action that causes our characters to stop right away, without taking any additional hops. However, in some cases the character may be traveling too quickly for the Stop action to be successful under these constraints. We were able to learn the Stop skill

from any velocity for every character except for C1 and C5. For these two characters, the maximum stopping α_v is 0.7 and 0.65, respectively. If the controller detects that the character is hopping too quickly, it first lowers the speed and then executes one of the appropriate Stop actions.

4.1.3 HopFlip and HopRoll skills

In this section we document the results for both HopRoll and HopFlip for all of our characters. In each of the figures below, we show histograms and scatter plots of Experiences collected in phase 2. Specifically, for each Experience we examine its length α_l , height α_h , the initial condition parameter α_v , and reliability. Recall that the reliability is computed as a fraction, which results in the lines that can be observed in some figures. For example, a reliability of 0.5 is common because it can be a result of several distinct fractions, such as $\frac{1}{2}$, $\frac{2}{4}$, $\frac{3}{6}$, ... etc. In all of the figures below, we only plot Experiences that were picked as promising at least 6 times during the execution of phase 2. This increases the accuracy of the reliability estimate for each of the plotted Experiences.

4.1. Acrobot

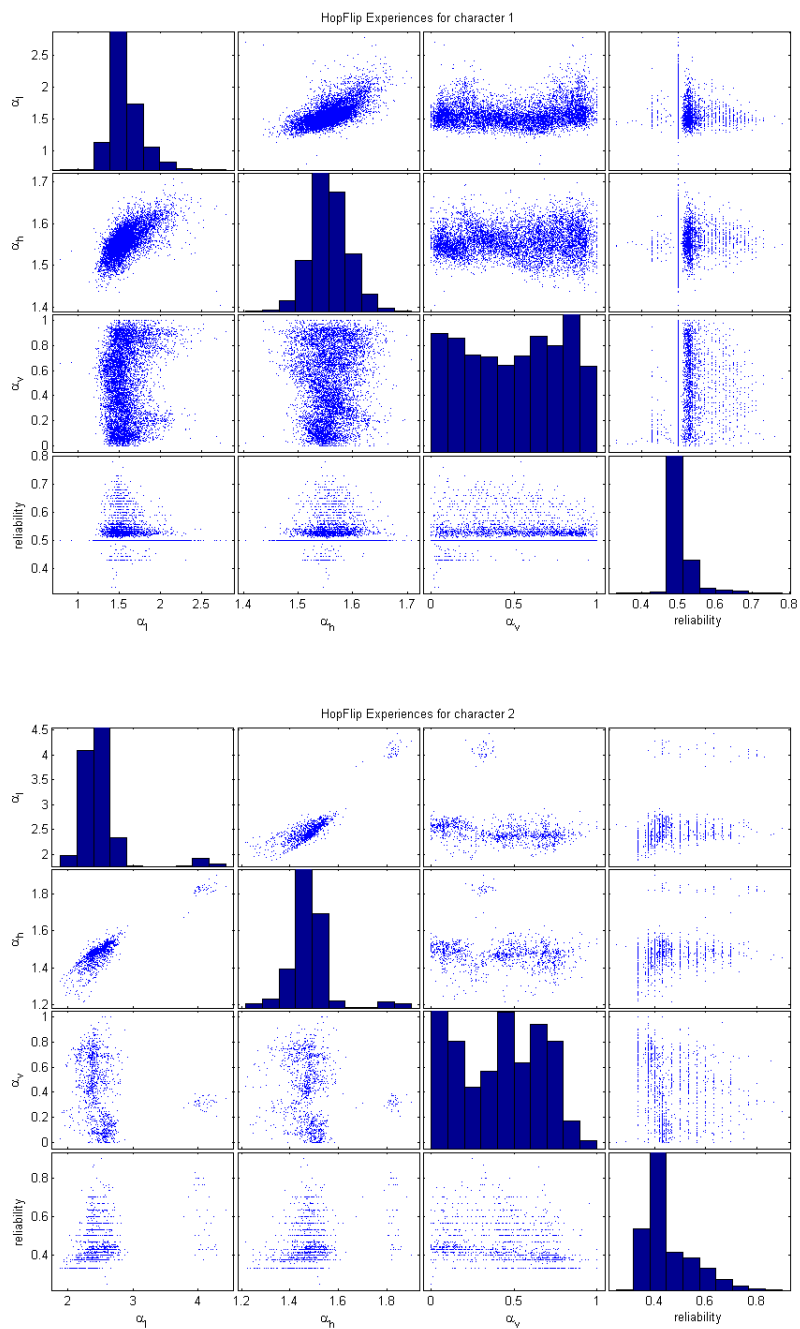


Figure 4.2: HopFlip capabilities for characters C1 (base character) and C2 (short base link).

4.1. Acrobot

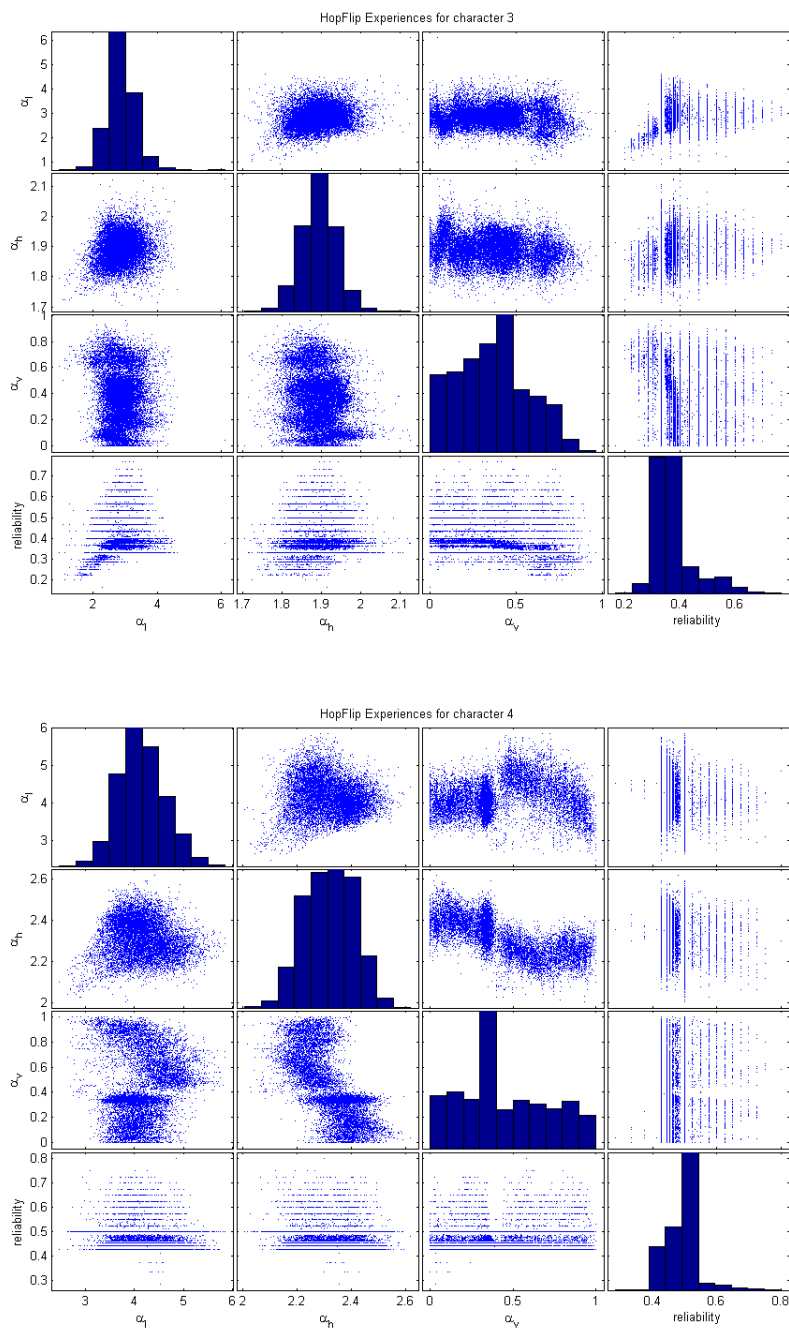


Figure 4.3: HopFlip capabilities for characters C3 (tilted base link) and C4 (heavy head link)

4.1. Acrobot

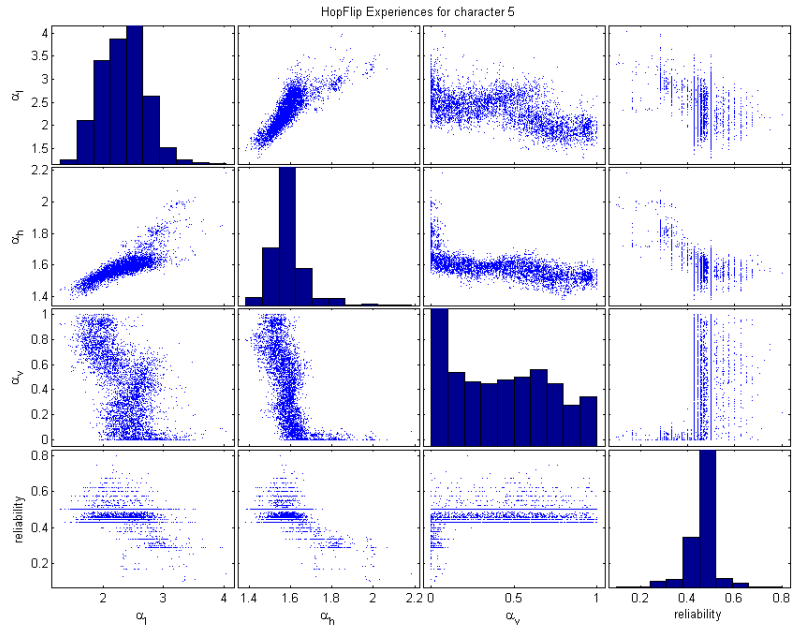


Figure 4.4: HopFlip capabilities for character C5 (small foot)

4.1. Acrobot

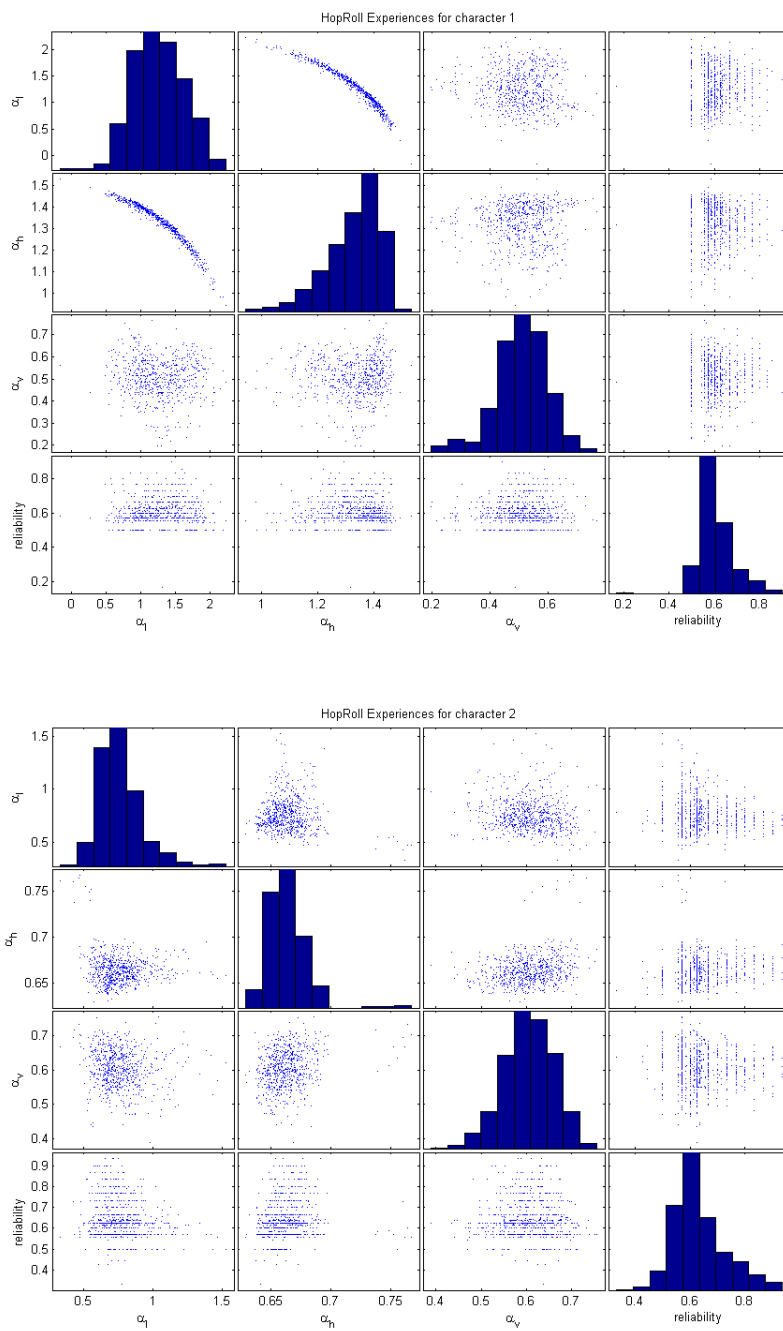


Figure 4.5: HopRoll capabilities for characters C1 (base character) and C2 (short base link).

4.1. Acrobot

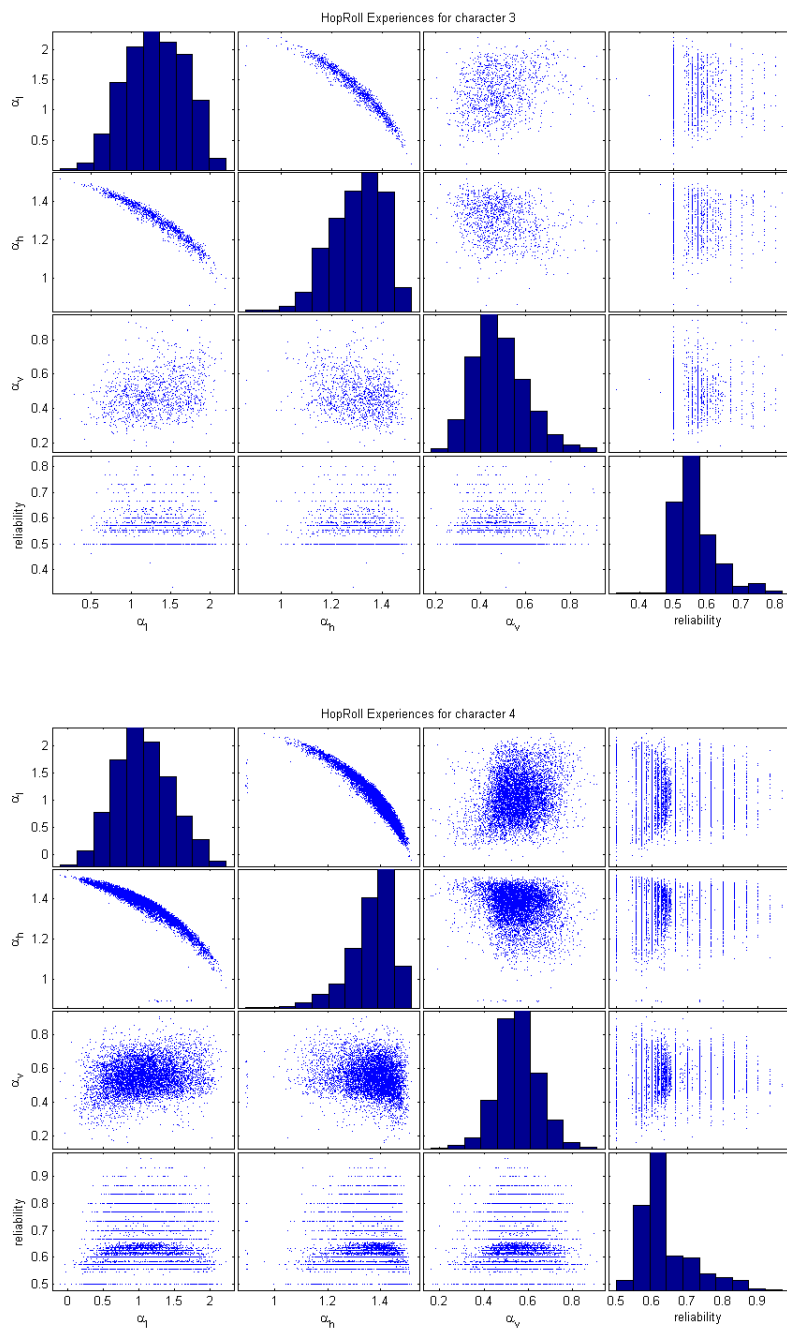


Figure 4.6: HopRoll capabilities for characters C3 (tilted base link) and C4 (heavy head link)

4.1. Acrobot

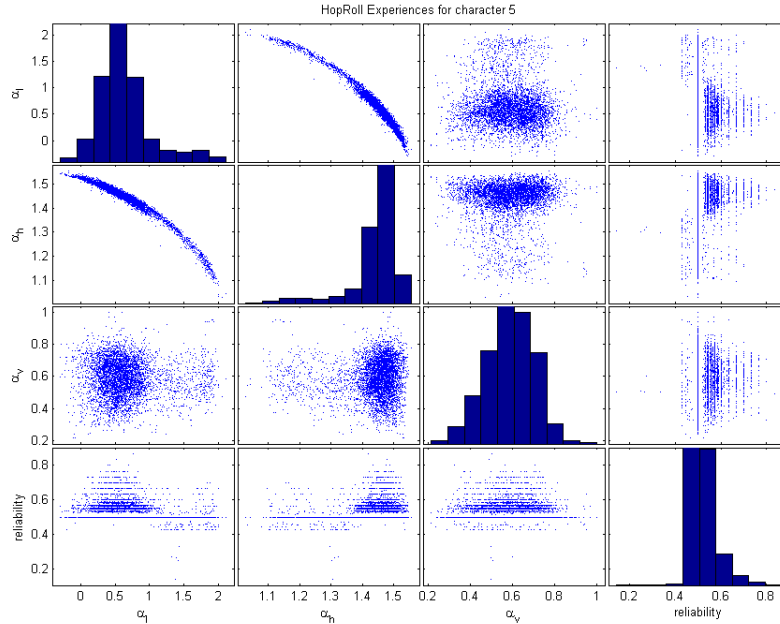


Figure 4.7: HopRoll capabilities for character C5 (small foot)

4.1.4 Acrobatics and GetUp skill

Recall that the Acrobatics skill searches for the parameters in $\mathbb{I} \times \mathbb{T}$ of Flip, HopRoll, and HopFlip, and Hop skills such that the resulting string of actions is successful. Despite the fact that both HopRoll and HopFlip are intended to work from a hop of some steady speed, we found that it was easy to find combinations of task parameters in the Acrobatics skill that still resulted in successful trials. We speculate that this robustness to initial conditions is a result of the fact that we explicitly maximize for reliability of every action in the final model of both HopFlip and HopRoll.

In Figure 4.8 we show the progress of phase 2 over time for the Acrobatics skill on all characters. The reliability value itself holds no information because it depends on the amount of noise that we add into the Motor Actions as we explore variations of each action in phase 2. However, a clear

4.1. Acrobot

trend emerges from the graph regarding the speed of convergence and the relative order of the characters in the graph. First, note that it only takes about 200 trials (roughly 10-20 seconds of computation) to find parameters that are close to what the algorithm eventually converges to. In addition, the task was easiest for C1, followed by C2, C3, C4, and finally C5.

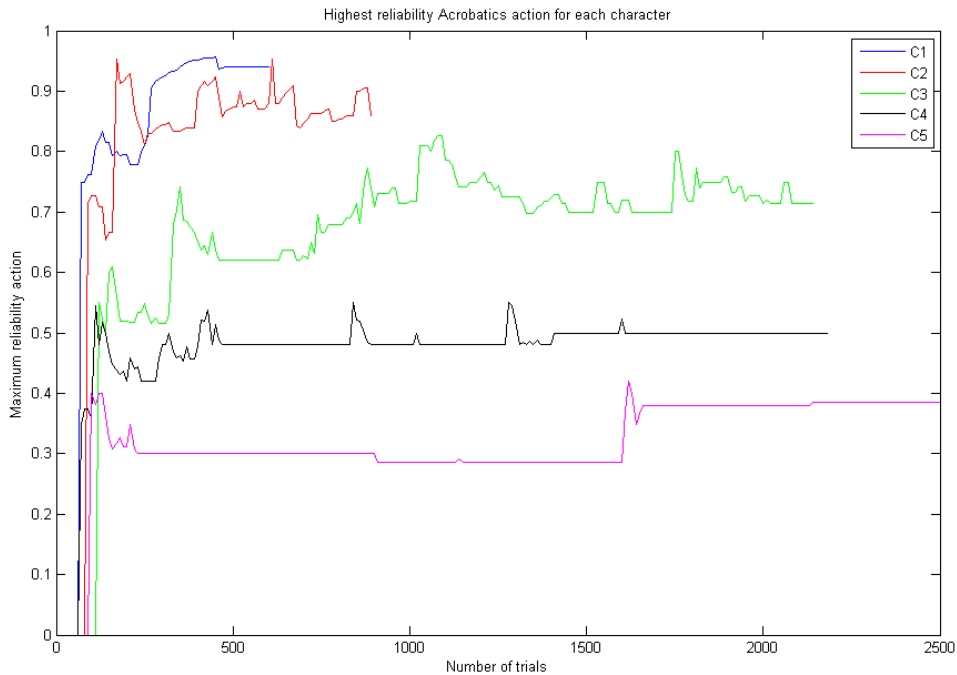


Figure 4.8: Maximum reliability over all Experiences collected in phase 2 of the Acrobatics skill as a function of number of trials. 2500 trials correspond to about 3 minutes of computation. The maximum reliability sometimes decreases as we accumulate more precise statistics on the reliability of each Experience.

The dynamics of the GetUp skill are easy for our character to master due to reasons described in 2.3.6. In practice, after only a few hundred trials during phase 2, the reliability of many GetUp actions is identically 1 for all of our characters.

4.1.5 Sample sequences of skills executed in sequence

The learned skills can be used to generate contiguous motion. Several aspects of the motion can be controlled by the user through task parameters of all skills. In the figures below, the user commands the character to execute various skills over time by pressing buttons in the user interface. The task parameters for each skill can be provided through sliders in the user interface. The user also has the option of not providing the task parameters. This functionality can be useful for situations where one wants to execute some motion without regard to its precise outcome. In these cases, the controller picks the task parameters that lead to the most reliable action.

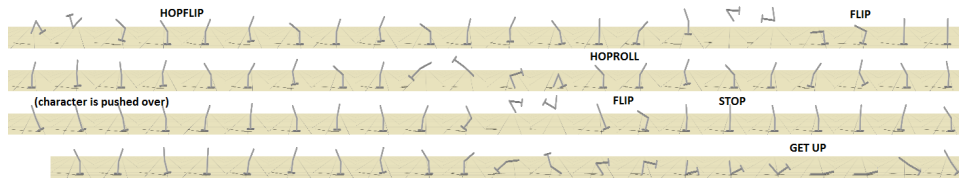


Figure 4.9: Sample contiguous sequence of skills, as executed by one of our characters. The sequence starts on top right, then goes across to the left, and downwards. Text above a film strip indicates what skill was initialized at that time.

4.1. Acrobot

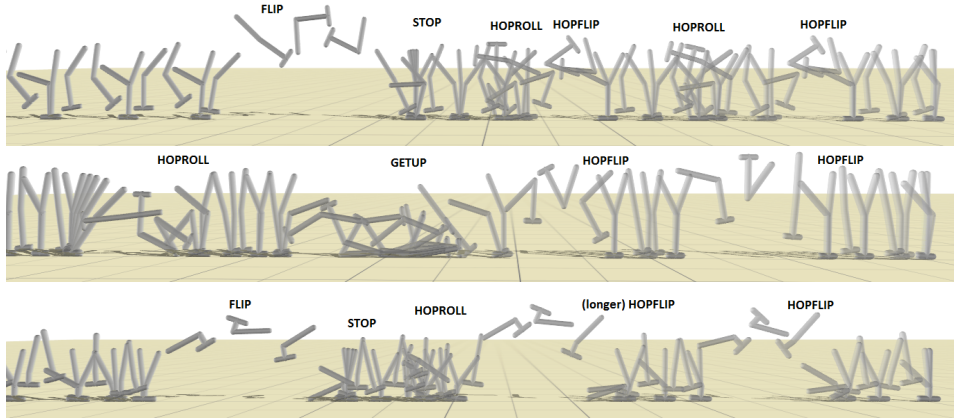


Figure 4.10: Additional example sequences with a stationary camera and overlapped frames for characters C1, C5 and C2 (from top to bottom). Note that in the first sequence, the character successfully completed a HopRoll right away after a HopFlip. This often works even though the character was trained to roll from a hopping motion. We believe that this robustness to initial conditions is a result of explicitly maximizing the reliability of our Motor Actions.

4.1.6 Use of task parameters in planning

The characters can accomplish various high-level tasks through appropriate use of task parameters in each skill. For example, with the skills defined in this work it is possible to do simple planning over terrain. Specifically, we experimented with gaps in the ground. For small gaps, the character can modulate its velocity to land close to the edge of the gap, and then execute a long HopFlip or HopRoll. For longer gaps, the controller uses the Stop skill to stop the character at an edge of the gap, and then uses the Flip skill to perform a large flip. If the character ever happens to fall, the controller detects this and uses the GetUp skill to recover.

4.1. Acrobot

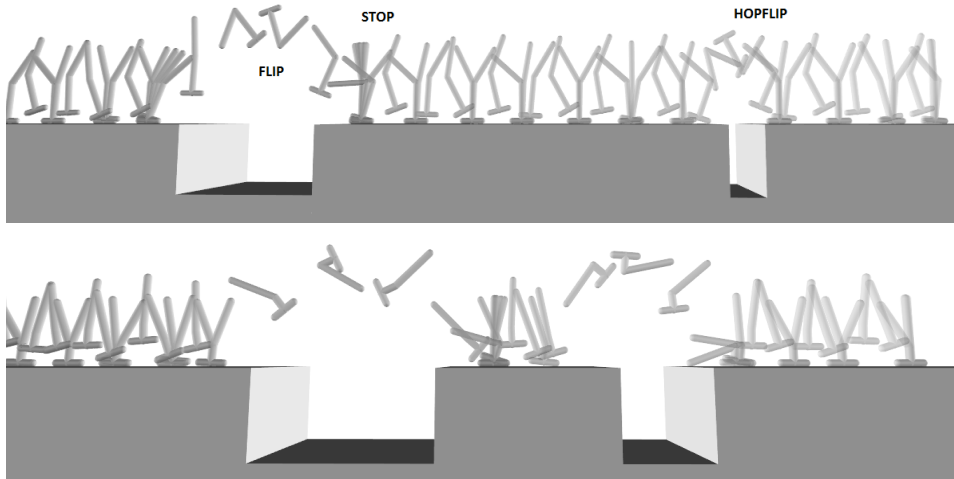


Figure 4.11: Sample runs through a terrain with two of our characters. The character goes from right to left. Snapshots are taken at regular time intervals. The character executes a HopFlip to get over the first gap because it is short enough for HopFlip. The second gap is too large, so the controller commands the character to hop to the edge, stop, and then execute a Flip.



Figure 4.12: Our characters are only trained on flat ground. However, they can still flip up on top of high platforms and then use the GetUp skill to continue. The learning algorithm generalizes easily to incorporate flips on platforms of different heights by adding a task parameter for height of the platform. We did not consider this extension in this work.

4.2 Quadruped character in 3D

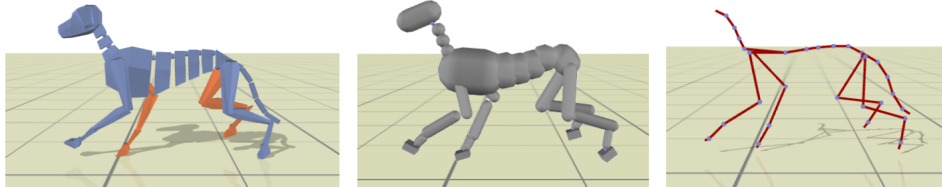


Figure 4.13: This image shows, from left to right, the display mesh, collision proxies, and the joint hierarchy of the dog character.

To demonstrate the scalability of our approach, we used our algorithm to learn parameterized leaps for a high-dimensional dog character in 3-D. Figure 4.13 shows the details of the character structure. The entire articulated figure is comprised of 30 links: 4 links for each leg, 6 for the back, 4 for the tail, and 4 for the neck-and-head. There are 67 internal degrees of freedom: 7 per leg, 15 for the spine, 12 for the neck-and-head, and 12 for the tail. In comparison, Acrobot only has 3 links and a single internal degree of freedom. Additional details about the character and many elements of the control strategy can be found in [7].

The controller for the dog character relies on two control abstractions to generate torques for every joint. Similar to this work, the first primary component of the final torques comes from PD-controllers. First, the controller specifies the desired end-effector positions over time for every leg. Inverse kinematics is then used to compute the exact pose that each leg should assume in order to meet the specified end-effector position. The output pose from Inverse kinematics is then used in the PD-controller to generate tracking torques. The second primary component derives from use of internal virtual forces [16]. Using this abstraction, the controller can specify forces on any link of the dog body. The method then allows the character to compute torques that it should apply on every joint such that the resulting motion behaves as if a given internal force was applied between the base link

and the point of application of the virtual force.

By appropriately controlling the positions of the legs and the virtual forces on links of the body over time, the character can accomplish a large variety of motions. In particular, the dog comes equipped with a controller for several common gaits such as trot, gallop, and walk. Details of the gaits and the control are described in more detail in [7].

We applied our learning algorithm to train the dog to leap to different heights and lengths, starting from a trotting gait at some speed. The dog leap skill is thus analogous to HopFlip: the task parameters of the skill are the length and height of the leap and the initial conditions contain a single parameter that describes the speed of the trotting gait that the dog is using immediately before the leap. While the learning algorithm did not have to undergo any changes, the Motor Actions take on a different interpretation. A skeleton of a single leap was first designed manually, but some of the exact timings, end-effector positions, and magnitudes of virtual forces were left as free parameters. These parameters then formed a 17-dimensional Motor Action space. The task of the learning algorithm thus became to find a setting of these parameters such that the result is leap of different lengths and heights from a range of initial velocities.

Similar to our Acrobot character, the dog can use the resulting parameterized leaping skill to accomplish various high-level tasks: it can navigate complex terrain by jumping over gaps of various lengths (Figure 4.16, top), it can jump on top of platforms of different heights (Figure 4.16, Figure 4.15), it can jump over barriers (Figure 4.16, bottom left), and finally, the dog is able to jump up and catch a sausage at an arbitrary reasonable position in space (Figure 4.16, bottom right).

4.2. Quadruped character in 3D



Figure 4.14: Film strip visualization of the dog as it leaps into the air and lands again. Horizontal distance is not to scale.



Figure 4.15: Film strip visualization of the dog as it leaps on top of a platform. Horizontal distance is not to scale.

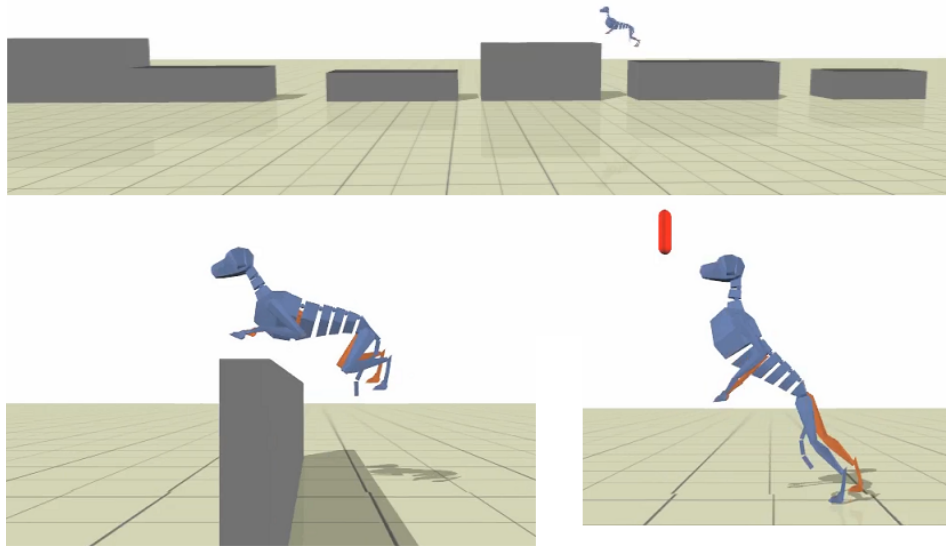


Figure 4.16: Illustrations of some of the capabilities of the dog after learning. Top: the dog can traverse a complicated environment by executing leaps with different length and height task parameters. Bottom Left: Jumping over barrier. Bottom right: Precise, targeted jump for sausage.

4.2. Quadruped character in 3D

As an alternative approach to the procedure outlined in phase 3, we develop an approach for generalization of the leaping skill by use of Principal Component Analysis. Specifically, we computed the principal components of all actions that produce a successful leap from a velocity in the range of $[0.45, 0.55]$. Next, we systematically sampled actions on the manifold spanned by the first two most significant principal components and subjected them to a trial. The resulting graph of outcomes can be found in Figure 4.17. It is interesting to note that the outcome varies smoothly with the relative displacement along each component. In addition, the first and second components correlate well with the height and length of the leap. Principal Component Analysis may therefore be a powerful heuristic for generating candidate models during phase 3 of the learning algorithm.

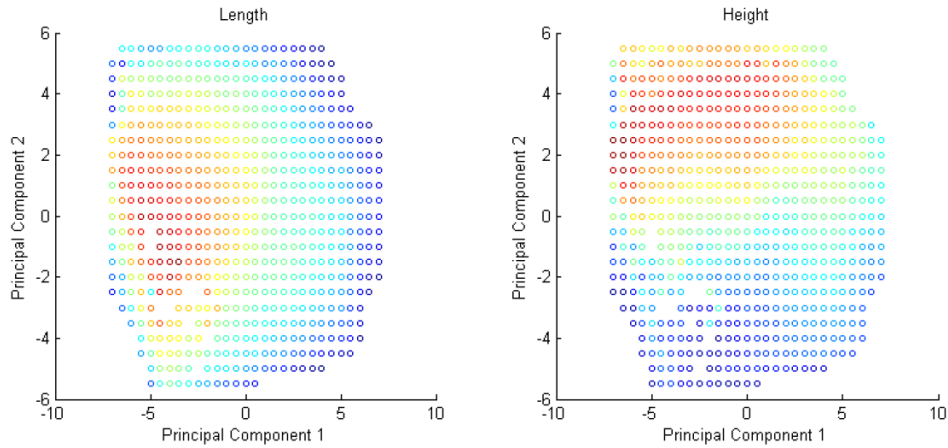


Figure 4.17: PCA analysis of dog leaps for some initial velocity. Visualization of the variance in length and height of leaps from one velocity, if we execute actions along the 2-dimensional manifold spanned by the first two principal component vectors. An absence of a circle at some coordinates indicates that the corresponding action led to an unsuccessful trial.

Chapter 5

Conclusions

We have presented a learning algorithm for agile, integrated whole-body skills of physically-simulated characters. The algorithm uses a nature-inspired online, active exploration of the character action space to find reliable motions that give rise to parameterized skills. We further demonstrated that our algorithm works for a family of simple characters without requiring any algorithm or parameter modifications. In addition, we experimented with a complex dog character in 3-D and showed that our approach generalizes to this character, given appropriate changes in the motor abstractions used during the learning process. Finally, we showed that the resulting parameterized skills can be effectively used for high-level tasks, such as traversing a terrain.

5.1 Discussion

While a learning approach to acquiring skills possesses many benefits, it also comes with its own set of limitations.

Mainly, we found that the learning process requires occasional supervision to ensure that the intended skills are actually being learned. For example in one case, a character learned a Flip motion that made it launch into air, fall down, and then get back up very quickly. On a different occasion, the character learned a double-flip instead of a flip. It thus became necessary to supervise the learning process and restart it on a few occasions. The majority of skill and character combinations (roughly 90%) did not require any interventions. We believe that these issues could be alleviated by better quantifying when a particular trial should count as being successful

for every skill.

In addition, the phase 1 reward functions may be difficult to specify in some cases. In particular, we found the GetUp skill to be the most troublesome. If the character fails at getting up after some Motor Action, how should one assign a score for how close the attempt was? Specifying a bad phase 1 reward function could lead to long computation times in phase 1, because the character is essentially left searching randomly in the motor space for a successful action. Even worse, the optimization in phase 1 could be repeatedly led astray with an inappropriately specified phase 1 reward function.

The main challenges for the quadruped character were aesthetic in nature. Unlike the Acrobot’s motions, a dog’s leap is a specific type of motion that we are all familiar with from nature. Even though the learning algorithm produced leaps that accomplished all desired goals, they did not always resemble leaps that one would expect to see from a real dog. For example, dogs exhibit a tendency to lift their front feet while in mid-flight, but this motion did not emerge in the actions that were produced by the algorithm. Instead, the dog left its front feet outstretched during the leap, producing a motion that felt qualitatively strange despite achieving all task goals. In the end, we opted to include these details into the leap controller manually to achieve a more familiar style of motion.

5.2 Limitations and future work

Even though the learning algorithm described in this thesis works well for our characters and the set of skills we considered, we make no claims to have addressed the general problem of motor learning. In this section, we discuss possible extensions of the proposed framework that can bring us closer toward the final goal of matching human or animal abilities.

To begin with, several immediate improvements can be made to the framework by addressing some of the simplifications that were made mostly

out of convenience.

Fine-grained Motor Actions. In our definition of piecewise defined Motor Actions, we manually selected the number of piecewise constant control actions for each skill. This simplifying assumption could be relaxed in later stages of the learning process. At some discrete points in time during the execution of the algorithm, we could take every piece in a Motor Action and split it in two distinct pieces in the middle. The added pieces could then be slowly refined as the algorithm continues its execution. Giving greater freedom to actions that can be learned could allow the character to develop better and more reliable skills.

Extensions to phase 3 model fitting In this work, we only considered fitting lines to data in phase 3 of the learning algorithm, but in principle this could be extended to hyperplanes that define a multi-dimensional task parameterization. However, we expect that higher-dimensional linear models may only cover small ranges of task parameters. In addition, good candidate models may be harder to find due to much larger number of possibilities. We investigated the use of Principal Component Analysis as a potential heuristic to address some of these issues. Finally, it may be possible to represent a skill using a set of local linear models to cover a greater breadth of task parameters.

More complicated environments. A subject of further research could be generalizing the framework to more complicated terrains, such as slopes of different angles, various friction coefficients of the ground, etc. The obvious way to handle these additional parameters is to include them as additional parameters in the initial conditions. For example, a HopFlip could be redefined as a mapping $(\alpha_v, \alpha_s, \alpha_f, \alpha_l, \alpha_h) \rightarrow A$ where α_s and α_f parameterize the slope and the friction coefficient, respectively. Clearly, as the number of parameters grows, the exploration will also become more difficult. It might therefore be advantageous to repeatedly toggle between phase 2 and phase 3. Phase 2 would find some Experiences in some region of the search space, and phase 3 could immediately attempt to generalize

them with a local model. One could then return back to phase 2, work on a different part of the space, and repeat.

Learning transition skills. More serious simplifications were made when designing the connectivity between skills. For example, the Hop skill has no setup action. In other words, the same actions are used to generate the hop not only from rest, but also while the character is already hopping. Getting rid of this simplification by learning the initial setup actions together with the skills is an interesting future direction. In general, we were able to do without having to learn transition motions between any of our skills, but it is clear that transition motions may be required in more complicated scenarios.

Composite Motor Actions. Another interesting direction for future research is to more closely investigate types of actions and their interactions. In this thesis, we used three separate interpretations for what an action is: For skills defined in section 2.3, an action is a piecewise-constant function that forms the input of a PD-controller. For the Acrobatics skill, an action is interpreted as the task parameters of other skills. Lastly, for the dog character, an action translated to virtual forces, end-effector positions, and relative timings. However, one could easily imagine other interpretations for an action. For example, an action could directly describe the raw torques that should be applied at some joints over time. Instead of choosing an action representation for every skill, it would be interesting to use all of them at once because they all have advantages and disadvantages depending on the type of task at hand. One could further imagine learning what the right type of action is for a skill, and even mixing different types of actions together to successfully execute a single skill.

Incorporating Feedback into Motor Actions. The representation of Motor Actions, as defined in 2.3 lacks concept of a continuous feedback mechanism. One direction to expand upon in the future is to add optional local feedback terms to Motor Actions, and then learn the feedback laws

for every skill in a new, fourth phase of the learning algorithm. This phase could even further increase the robustness of every skill to perturbations in initial conditions, and the actions themselves. For example, an algorithm could learn that while it is executing a Hop action, then if it is ever leaning too much to the front, it should kick back further to recover. A possible way to achieve this is as follows: We could push the character while it is hopping to make it fall. Then, the character could attempt variations of the action until it does not fall from that same push. If we remembered results of this form for many pushes, we could use them to predict compensations to novel pushes. As we did in phase 3, we could further attempt to generalize from this data to arrive at a local linear feedback law.

Parallel execution of Motor Actions. We assumed that only a single Motor Action can be played at a time. An obvious extension is to allow several Motor Actions to be played at the same. This could allow, for example, a human character to walk forward, while simultaneously picking up an object from a nearby table.

Learning skills and their connectivity. The framework presented above describes a general learning algorithm for a every one of our skills, but no attempts are made to learn the skills themselves. Every skill also comes with pre-defined goals, initial parameters, and its connectivity with other skills. There are other hard-coded elements in the framework. For example, the salient common event of foot touching the ground is at the center of every skill transition, but the event is hard-wired for this particular character. Eliminating these aspects of the framework is another interesting direction for future work.

Parallelizing the computation. Implementation aspects of the framework could also be immediately improved on. For example, there is much potential for parallelizing the execution of the learning algorithm. Not only is there room for parallelizing the individual phases, but it is also possible to parallelize entire skills because some of them are mutually independent.

For example, GetUp, Stop, and Flip are all independent given that the Hop has been learned. Similarly, HopFlip and HopRoll are independent given Hop and Flip.

Transfer learning to the real world. One of the promises of this algorithm is that it could eventually be used in Robotics. One possible way of accomplishing this is to run it in simulation to train several skills, and then attempt to transfer them onto a real robot. This transfer learning process from simulation to real world is arguably feasible because the solutions that work inside the simulation may only need to be slightly adjusted.

Improvements aimed toward Robotics. It could also be possible to have the algorithm run on a real robot from the beginning, without the need of a high-fidelity simulated environment. In its present form, it is not practical for immediate use on a robot. However, it is possible to address this by explicitly incorporating some of the limitations that real robots are also subject to. For example, real robots cannot magically reset themselves to a rest state. Real robots also cannot run faster than real-time, so the algorithm should be modified with the explicit goal of minimizing the number of trials that must be attempted. Finally, robotic hardware is expensive and subject to damage. The algorithm could respect this limitation by carefully considering actions that may result in high-velocity impacts and not attempting them unless it is sure that they will succeed with a high probability. Even though the algorithm currently ignores all of the above issues, they are not theoretically insurmountable, and present interesting challenges for future research.

Bibliography

- [1] Pieter Abbeel, Adam Coates, and Andrew Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 2010.
- [2] M.D. Berkemeier and R.S. Fearing. Sliding and hopping gaits for the underactuated acrobot. *Robotics and Automation, IEEE Transactions on*, 14(4):629–634, aug 1998.
- [3] G. Boone. Minimum-time control of the acrobot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 3281–3287 vol.4, apr 1997.
- [4] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 28(5):Article 170, 2009.
- [5] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transactions on Graphics*, 29(4):Article 130, 2010.
- [6] Stelian Coros, Philippe Beaudoin, KangKang Yin, and Michiel van de Panne. Synthesis of constrained walking skills. *ACM Trans. Graph. (Proc. Siggraph Asia)*, 27(5), 2008.
- [7] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Revert, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *SIGGRAPH 2011*, 2011.
- [8] J.J. Craig. Introduction to robotics: Mechanics and control. 2005.

- [9] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Trans. Graph.*, 29:131:1–131:10, July 2010.
- [10] John Hauser and Richard M. Murray. Nonlinear controllers for non-integrable systems: the acrobot example. In *American Control Conference, 1990*, pages 669–671, may 1990.
- [11] Pedro S. Huang and Michiel van de Panne. A planning algorithm for dynamic motions. *Eurographics Workshop on Computer Animation and Simulation*, 1996.
- [12] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. *ACM Trans. Graph.*, 29:129:1–129:8, July 2010.
- [13] Max Lungarella. Developmental robotics: a survey. *Connection science*, 15:151, 2003.
- [14] Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Trans. Graph.*, 28:80:1–80:8, July 2009.
- [15] M. Miyazaki, M. Sampei, M. Koga, and A. Takahashi. A control of underactuated hopping gait systems: acrobot example. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4797–4802 vol.5, 2000.
- [16] Chew Pratt J., Torres, Dilworth, and Pratt G. Virtual model control: An intuitive approach for bipedal locomotion. *Int'l Robotics Research*, 2001.
- [17] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions: Biological Sciences*, 358(1431):pp. 537–547, 2003.
- [18] Richard Schmidt and Tim Lee. Motor control and learning-4th edition: A behavioral emphasis. 2005.

Bibliography

- [19] M.W. Spong. The swing up control problem for the acrobot. *Control Systems, IEEE*, 15(1):49–55, feb 1995.
- [20] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.