

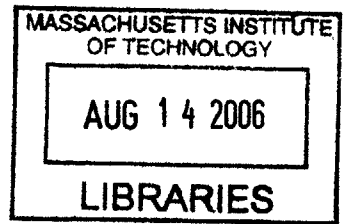
Policy Search Approaches to Reinforcement Learning for Quadruped Locomotion

by
Antonio R. Jimenez

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

June 2006
May 2006

© Antonio R. Jimenez, MMVI. All rights reserved.



The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author _____
Department of Electrical Engineering and Computer Science
May 12, 2006

Certified by _____
Leslie P. Kaelbling
Professor of Computer Science and Engineering, MIT
Thesis Supervisor

Certified by _____
Member of _____
Paul A. DeBitetto
Laboratory Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

THIS PAGE INTENTIONALLY LEFT BLANK

Policy Search Approaches to Reinforcement Learning for Quadruped Locomotion

by

Antonio R. Jimenez

Submitted to the
Department of Electrical Engineering and Computer Science
May 12, 2006
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Legged locomotion is a challenging problem for machine learning to solve. A quadruped has 12 degrees of freedom which results in a large state space for the resulting Markov Decision Problem (MDP). It is too difficult for computers to completely learn the state space, while it is too difficult for humans to fully understand the system dynamics and directly program the most efficient controller. This thesis combines these two approaches by integrating a model-based controller approach with reinforcement learning to develop an effective walk for a quadruped robot. We then evaluate different policy search approaches to reinforcement learning. To solve the Partially Observable Markov Decision Problem (POMDP), a deterministic simulation is developed that generates a model which allows us to conduct a direct policy search using dynamic programming. This is compared against using a nondeterministic simulation to generate a model that evaluates policies. We show that using deterministic transitions to allow the use of dynamic programming has little impact on the performance of our system. Two local policy search approaches are implemented. A hill climbing algorithm is compared to a policy gradient algorithm to optimize parameters for the robot's model-based controller. The optimal machine-learned policy achieved a 155% increase in performance over the hand-tuned policy. The baseline hill climbing algorithm is shown to outperform the policy gradient algorithm with this particular gait.

Thesis Supervisor: Leslie P. Kaelbling
Title: Professor of Computer Science and Engineering, MIT

Thesis Supervisor: Paul A. DeBitetto
Title: Member of the Technical Staff, Charles Stark Draper Laboratory

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

This thesis was prepared at the Charles Stark Draper Laboratory, Inc., under internal Research and Development.

Publication of this thesis does not constitute approval by the Draper Laboratory or any sponsor of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

I would like to thank Leslie Kaelbling for the guidance she has provided and the opportunities she has given me. I would also like to thank Paul DeBitetto for allowing me to work in his group and opening the doors of robotics research for me.

I could not have accomplished this research without the rest of the Draper Laboratory Cognitive Robotics group: Greg Andrews, Jeff Miller, Adam Rzepniewski, Scott Rasmussen, and Chris Rakowski. Their engineering efforts and important insights were essential to this work. Also, I must thank Rick Cory and Russ Tedrake for key conversations that were immensely helpful.

Finally, I thank my family for all their support and help. Liz, for always believing in me. Katie, for the immense motivation she gave me and being a bundle of joy.

Antonio R. Jimenez

May 12, 2006

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 15 |
| 1.1 | Motivation | 16 |
| 1.2 | Thesis Approach | 16 |
| 1.3 | Thesis Roadmap | 18 |
| 2 | Background | 19 |
| 2.1 | Reinforcement Learning | 19 |
| 2.2 | Markov Decision Processes | 20 |
| 2.2.1 | The Fully Observable Case | 20 |
| 2.2.2 | Partially Observable Markov Decision Processes | 21 |
| 2.3 | Solving Reinforcement Learning Problems | 23 |
| 2.3.1 | Q-learning | 23 |
| 2.3.2 | Policy Search | 24 |
| 3 | Policy Search by Dynamic Programming | 27 |
| 3.1 | PEGASUS | 27 |
| 3.1.1 | The Deterministic Model | 27 |
| 3.1.2 | Function Approximation | 28 |
| 3.2 | PSDP | 29 |
| 3.2.1 | Non-Stationary Policies | 30 |
| 3.2.2 | Performance Bounds | 30 |
| 3.3 | Problem Abstraction | 31 |

| | | |
|----------|--|-----------|
| 3.3.1 | Performance Bounds with this Problem Abstraction | 32 |
| 3.3.2 | Applying Deterministic Models to the Problem Abstraction . . | 34 |
| 4 | Implementation | 37 |
| 4.1 | Controlling a Quadruped | 37 |
| 4.1.1 | The Designed Gait | 37 |
| 4.1.2 | Step Control | 38 |
| 4.2 | Learning a Better Walk | 40 |
| 4.3 | Simulation of an Aibo | 42 |
| 5 | Results and Analysis | 43 |
| 5.1 | Results | 43 |
| 5.1.1 | Simulation Results | 43 |
| 5.1.2 | Hardware Verification | 45 |
| 5.2 | Analysis | 48 |
| 5.2.1 | Comparing PGA and Hill Climbing | 48 |
| 5.2.2 | Comparing Models | 51 |
| 6 | Conclusion and Summary | 55 |
| 6.1 | Conclusion | 55 |
| 6.2 | Future Work | 56 |
| 6.3 | Summary | 57 |
| A | Simulation Data | 59 |
| A.1 | Joint Positions | 59 |
| A.2 | Torque Control | 61 |
| B | Acronyms | 63 |

List of Figures

| | | |
|-----|--|----|
| 2-1 | Overview of Reinforcement Learning | 20 |
| 2-2 | A Partially Observable Markov Decision Process | 23 |
| 3-1 | Degrees of Freedom per Leg | 31 |
| 4-1 | The Aibo Robot Model ERS-7 | 37 |
| 4-2 | The Designed Gait | 38 |
| 4-3 | Controlling the Step | 39 |
| 5-1 | Nondeterministic Test Results with the Quasi-Greedy Algorithms | 45 |
| 5-2 | Deterministic Test Results with the Quasi-Greedy Algorithms | 46 |
| 5-3 | Nondeterministic Test Results with the Completely Greedy Algorithms | 46 |
| 5-4 | Deterministic Test Results with the Completely Greedy Algorithms | 47 |
| 5-5 | The Learned Gait | 48 |
| 5-6 | Comparison of Nondeterministic and Deterministic models with PGA | 52 |
| 5-7 | Comparison of Nondeterministic and Deterministic models with Hill-Climbing | 53 |
| A-1 | Shoulder Position | 59 |
| A-2 | Upper Leg Position | 60 |
| A-3 | Lower Leg Position | 60 |
| A-4 | Shoulder Torque | 61 |
| A-5 | Upper Leg Torque | 62 |
| A-6 | Lower Leg Torque | 62 |

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

| | | |
|-----|---|----|
| 3.1 | Problem abstraction | 32 |
| 4.1 | The initial control parameters and the modified ϵ values | 39 |
| 5.1 | The best scores found with each algorithm | 43 |
| 5.2 | The optimized control parameters | 44 |
| 5.3 | Hardware verification | 47 |

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Much research has been conducted in autonomous robotic control. Legged, ground robots pose much promise in allowing the ability to operate in rough terrain. In situations requiring humans to carry increasingly heavier loads through terrains such as mountains, larger legged robots would be able to serve as pack mules. Smaller legged robots would also be able to operate in areas where humans are unable to easily enter, such as through rubble. The challenge then becomes to try to develop intelligent controllers to be able to quickly and effectively operate in challenging terrain.

Reinforcement learning can be used to adapt to the environment or optimize a robot's controller for a specific terrain. Directly searching for the best policy can be very effective in designing a controller [16].

The goal of this research is to create a better controller for quadruped locomotion, movement across the ground for a four-legged robot. For the scope of this work, the focus is on the speed of the walk, with the goal of walking as fast as possible without falling over. In this case, reinforcement learning is applied to optimize the controller.

1.1 Motivation

Two approaches have been used to solve this problem. It is possible to completely hand design a controller. The parameters of the controller are then set by hand to attempt to maximize the efficiency of the controller.

The second approach is for reinforcement learning to completely control all aspects of the robot's movement. This would mean controlling the angle for each of the joints. The balance of an inverted pendulum which controls a joint with one degree of freedom (DOF) is a common example problem for reinforcement learning. However, a quadruped is especially challenging since it has 12 degrees of freedom. This creates a rather large state space which has not even taken into account other sensors on the robot yet.

Each of these approaches has inadequacies. With many parameters, it is too difficult for humans to directly design the most efficient controller. In addition, it is not practical for computers to completely learn the state space of the reinforcement learning problem.

The current solution is a combination of these two approaches. This combines a human designed model with reinforcement learning to either learn parameters for the model-based controller or some aspect of the state space. It is possible to view the model-based controller as a method to abstract the state space of the problem to a solvable space for the reinforcement learning algorithms.

1.2 Thesis Approach

This thesis approaches the reinforcement learning problem using the policy search set of algorithms. Our goal is to compare two optimization algorithms [10] that will

attempt to maximize the performance of the policy. In addition, we will frame the problem according to the standard Markov model for reinforcement learning. This will allow the analysis of the problem for algorithms for more complex belief state spaces [15] [3]. It will also provide insight and a development of the intuition for the performance bounds of these algorithms.

A theoretical analysis of the problem moves us in the direction of solving the additional aspects of the overall problem of controlling a walk over rough terrain. By producing insight into the performance bounds of these algorithms using a more complex belief state space, this work may help to integrate reinforcement learning into a larger scope of the controller.

Specifically, this thesis produces insight into these algorithms by analyzing the use of deterministic models for policy search. Using deterministic models allows the use of dynamic programming directly on the policies, rather than on the states, which is how dynamic programming is traditionally used for reinforcement learning. By investigating the use of deterministic models, we will look at its impact on performance in the simulated and real worlds.

The designed model-based controller, or gait, is created for future use in rough terrain. It uses a single-step gait, meaning only one leg is off the ground at a time, as opposed to using two diagonal legs in phase with one another, which would maximize the speed over flat terrain. Each step is defined by a simple quadratic trajectory. The model-based controller with learning shows a large speed increase over the original hand-tuned controller. This demonstrates the value of using policy search to optimize a quadruped's controller.

1.3 Thesis Roadmap

In this thesis I demonstrate the use of policy search on reinforcement learning. I will compare two policy search algorithms and show the impact of deterministic models on the rate of learning and performance in the real world.

The chapters of this thesis are organized as follows:

- Chapter 2 gives an overview of reinforcement learning and the strategies used to solve this class of problems. This chapter presents the fundamental framework used for reinforcement learning problems by detailing Markov Decision Processes (MDPs) which define the states and actions of the system.
- Chapter 3 goes in depth into the policy search set of algorithms for reinforcement learning. Specifically, it addresses recently developed algorithms that allow the use of dynamic programming directly on the policies. This chapter also provides an analysis that can serve as a theoretical basis for future work.
- Chapter 4 presents the implementation of the system. The model-based controller is described in detail and the optimization algorithms are presented. The experiments are described along with the framework of the simulation.
- Chapter 5 presents the results from the experiments in simulation and the hardware verification of these results. The analysis shows the verification of the theoretical results presented in Chapter 3.
- Chapter 6 discusses the conclusions from this research and expands on future work for quadruped locomotion in rough terrain in dealing with the corresponding challenges.

Chapter 2

Background

2.1 Reinforcement Learning

In this chapter, the basics of reinforcement learning are reviewed and the notation for the rest of the thesis is presented.

Reinforcement learning is the problem of learning behavior through trial and error while interacting with the environment. This sequential decision making needs to take into account a delayed reward. A good action now may not result in receiving a high reward until much later. In the same manner, penalties for current actions may not be received until later. A future state of the agent is also conditionally independent of past states given the current state. It is solely dependent on the actions taken from the current state.

To formalize this system, a Markov Decision Process provides the basis for reinforcement learning.

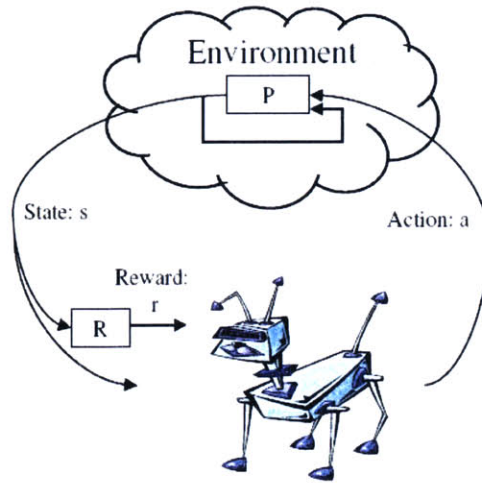


Figure 2-1: An overview of the reinforcement learning model. The state is determined by the environment with the transition probabilities, P . The reward is determined by the Reward Function, R .

2.2 Markov Decision Processes

2.2.1 The Fully Observable Case

A Markov Decision Process is defined as a model with the following components:

- A set of states, S
- A set of actions, A
- A reward function, $R : S \times A \rightarrow \mathfrak{R}$
- A transition probability function, $P : S \times A \rightarrow \mathcal{P}(S)$, where $\mathcal{P}(S)$ is a probability distribution over the states of the environment.

The transition probability function determines the chances that a particular action from a particular state will take the agent to another state. This basically means that the result of an action is nondeterministic. The reward function provides the expected reward received from a state while taking an action. The task in solving an MDP

is to create a policy π for choosing actions which will maximize the agent's lifetime reward. This is shown as

$$Q = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots , \quad (2.1)$$

where γ is defined as the discount factor for rewards in future states. This allows this value to converge. This Q value can be defined by value iteration for a finite horizon H as

$$Q_H(s, a) = R(s, a) + \gamma \sum \left[p(s'|s, a) \max_{a'} Q_{H-1}(s', a') \right] . \quad (2.2)$$

The policy that will solve this reinforcement learning problem is just

$$\pi(s) = \arg \max_a Q(s, a) . \quad (2.3)$$

In summary, the process of executing a MDP is to observe the state in time step t , to choose an appropriate action, to receive the reward corresponding to that state and action, and to change the state according to the transition probabilities.

2.2.2 Partially Observable Markov Decision Processes

There are many sources of uncertainty in the real world. With any sensor for a robot, it is important to model the uncertainty caused by the sensor's noise. The robot never has perfect knowledge of the environment around it, but it can maintain a belief of the environment, based upon its sensors.

For the Markov Decision Process, this means that it is not possible to fully observe the state the agent is in. Instead, at each step in the decision process, the agent receives an observation and then has a belief of which state it is in, based upon the observation probabilities for the states and the transition model.

Therefore, let us define a Partially Observable Markov Decision Process (POMDP) as having the following components:

- A set of states, S
- A set of actions, A
- A reward function, $R : S \times A \rightarrow \mathfrak{R}$
- A transition probability function, $P : S \times A \rightarrow \mathcal{P}(S)$
- A set of observations, O
- A set of observation probabilities, $p(O|S)$
- An initial state distribution, μ_0

Let us also define μ_t as the state distribution at a given time t . μ_0 would be the initial state distribution, but D is also used in Chapter 3 to reference the initial state distribution. In that chapter these state distributions are used to provide an expectation on the value of policies.

The belief state generated from the observations is not independent of past observations. It can be defined as

$$b^t = p\left(s^t | o^t, a^t, o^{t-1}, a^{t-1}, \dots, o^0, a^0, s^0\right) . \quad (2.4)$$

While updating the belief state is easy, finding the optimal k -step policy is doubly exponential in k . For n states, the optimal policy must be found over an $n - 1$ dimensional belief state. This causes the finding of the optimal policy for a POMDP to be a rather difficult task.

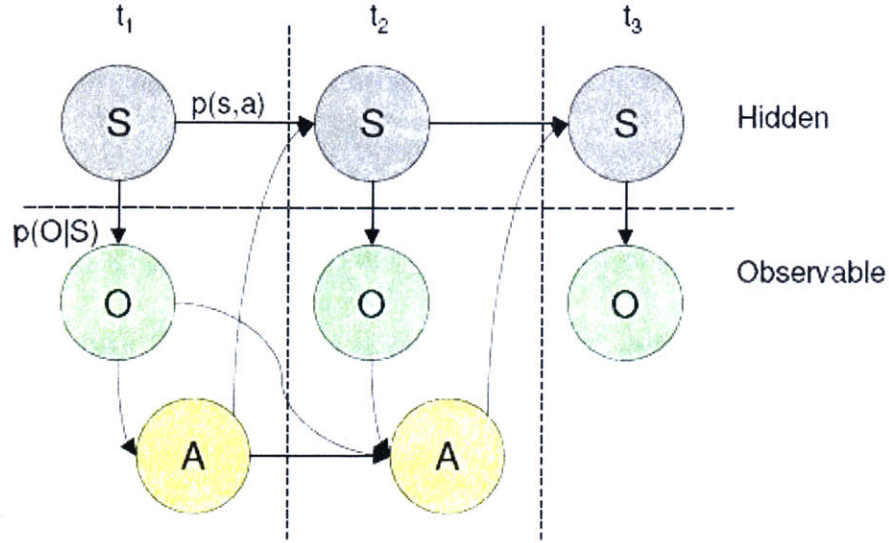


Figure 2-2: An overview of a Partially Observable Decision Process. The states are hidden from the agent, which must choose its actions just from its observations.

2.3 Solving Reinforcement Learning Problems

There are two general approaches to solving reinforcement learning problems. The first is to use dynamic programming to estimate the utility of taking actions for specific states in the world. Q-Learning is an example of this method and is presented here for completeness. The second approach is to directly search the space of policies for a policy that results in the best performance. The main focus of this thesis is on these policy search methods.

2.3.1 Q-learning

The Q-value of a state can be determined if the P and R are known. If they are not known, then the Q-value can be trained directly through a method called Q-Learning.

Q-Learning is called a model-free method since it does not need a model for either learning or action selection. Instead, the Q-values are learned for each state. Q-

Learning is an iterative process that will eventually compute exact Q-values. The value iteration for a state is given by Equation 2.2. The update equation for Q-Learning is given by

$$Q(s, a) := Q(s, a) + \alpha \left(R() + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) , \quad (2.5)$$

where α is the learning rate. α must be decayed appropriately. Then if each action is executed in each state an infinite number of times, the agent's Q-values will converge to the exact Q-values of the system.

Calculating the optimal policy is then straightforward since it is given by choosing the actions that will maximize the Q-value.

2.3.2 Policy Search

Policy search is the straightforward solution to the reinforcement learning problem, by focusing on what we truly care about, the behavior of the agent.

A form of policy search can be summarized as

$$\pi(s) = \max_a \hat{Q}(s, a) . \quad (2.6)$$

The general idea is to continuously adjust the policy to improve the performance and then stop. With the proper parameterization, this problem can be formulated to be solved by using standard local function optimization. As with all local searches, deciding when to stop or finding ways to move out of local optima is a challenge. There are further challenges in dealing with function discontinuity which is common with policies. We present two algorithms in Chapter 4 that attempt to solve a local policy search that deals with these issues.

If the policy is determined by Q-functions as in Equation 2.6, then policy search results in a process that learns the Q values of a state. Dynamic programming is used to save these values. However, this is different than Q-learning, since the goal of Q-learning is to find a \hat{Q} that is close to the optimal Q-function, Q^* . Instead, policy search is solely focused on the policy that results in good performance, even though the the Q-values may be greatly different than the true underlying Q-value of a state.

Russell [19] presents a good example of this difference. The approximate Q-function defined by $\hat{Q}(s, a) = Q^*(s, a)/10$ will give an optimal policy, but it remains far from Q^* .

However, this method of policy search is still using dynamic programming on the Q-values of a state. In the next chapter, we will present a method which will allow the use of dynamic programming directly on the policies.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Policy Search by Dynamic Programming

3.1 PEGASUS

3.1.1 The Deterministic Model

The model that we are referring to with the phrase “the deterministic model”, is the generative model that allows us to try actions from different states. This will generate our scores for each of the actions. In a practical sense, this model is the simulation of the robot used to test a controller. This use of the term “model” should not be confused with the overall Markov or POMDP model presented in Chapter 2.

Russell [19] provides a clear example for the benefit of using a deterministic model. He asks us to consider the task of determining which of two blackjack programs is the best. An initial approach is to have each program play against the same dealer and then base their score on the amount of money they earn (or lose). However, their score will vary depending upon the cards that each of them receive. We are left wondering whether one program was just lucky, or if it is truly the better blackjack program.

A solution to this problem is to have each program receive the same set of cards. The idea is to eliminate the error due to the cards that are received. To apply this to our problem, the system randomness is found in our sensor noise. This noise impacts the placement of each of the joints. Even though a joint is commanded to a certain angle, it only achieves that angle with a certain amount of error.

To make a deterministic model for the quadruped simulation, we set aside a pre-determined set of random numbers which is used to determine the system noise. Therefore each policy is scored with the same sensor noise model. This is referred to as a deterministic simulative model [15]. We will compare this to a nondeterministic model.

What results from this transformation is a modified POMDP where taking an action in a state will always result in transitioning to some fixed state. It is this deterministic simulative model that provides the new Markov model with which PEGASUS can operate on. In general, this deterministic model gives the ability to store the value of an action for a given state into a table. In this way, we are able to use dynamic programming directly on policies.

3.1.2 Function Approximation

PEGASUS presents a function approximation for the value of policies. In general, the value of a policy can be defined by the following equation with respect to some initial state distribution D

$$V(\pi) = E_{s_0 \sim D} [V^\pi(s_0)] \quad , \quad (3.1)$$

where $s_0 \sim D$ means that s_0 is drawn according to D for the expectation. $V^\pi(s_0)$

returns the estimated sum of the rewards for executing π from s_0 .

An approximation to $V(\pi)$ can be defined as

$$V(\pi) \approx \frac{1}{m} \sum_{i=1}^m V^\pi(s_0^{(i)}) , \quad (3.2)$$

where $s_0^{(i)}$ is a drawn sample of m initial states according to D . Then since the transitions from each state can be exactly determined, this sum can also be exactly determined. The simulative model will determine the states from the policy and sum the rewards with a discount factor γ . The horizon, H , is set as $H = \log_\gamma(\epsilon(1-\gamma)/2R_{max})$ which will give an $\epsilon/2$ approximation for the function.

The final approximation is given as

$$\hat{V}(\pi) = \frac{1}{m} \sum_{i=1}^m R(s_0^{(i)}) + \gamma R(s_1^{(i)}) + \dots + \gamma^H R(s_H^{(i)}) , \quad (3.3)$$

where m scenarios are used to define $\hat{V}(\pi)$ for all π . Since we are using deterministic transitions, this is a deterministic function and standard optimization techniques can be used to find the optimized policy. Two optimization algorithms are presented in Section 4.2. They are implemented and tested with the problem abstraction presented later in this chapter.

3.2 PSDP

Fundamentally, Policy Search by Dynamic Programming (PDSP) and PEGASUS are very similar. In fact, PEGASUS is, itself, a form of Policy Search by Dynamic Programming. However, the PSDP algorithm presented in [3], specifies the use of non-stationary policies and does not specify the function approximation method, while PEGASUS uses stationary policies with a specific approximation function.

3.2.1 Non-Stationary Policies

PSDP as presented in [3] is summarized in Algorithm 1 where μ_t is the state distribution at time t . This would mean that $\mu_0 = D$ where D is defined in Section 3.1.2. This algorithm states that the policy can change over time, T and the algorithm maximizes this non-stationary policy, Π .

Algorithm 1 PSDP. Given T , μ_t , and Π

for $t = T - 1, T - 2, \dots, 0$ **do**
 Set $\pi_t = \operatorname{argmax}_{\pi' \in \Pi} E_{s \sim \mu_t} [V_{\pi', \pi_{t+1}, \dots, \pi_{T-1}}(s)]$
end for

3.2.2 Performance Bounds

The theorem for the performance bounds of this algorithm for an optimized policy π calculated with a function approximation with an error ϵ is defined as

$$V_{\pi}(s_0) \geq V_{\pi_{ref}}(s_0) - T\epsilon - Td_{var}(\mu, \mu_{\pi_{ref}}) , \quad (3.4)$$

where π_{ref} is any possible policy in Π and where $d_{var}(\mu, \mu')$ is defined as the average variational distance between two distributions. This is given as

$$d_{var}(\mu, \mu') = \frac{1}{T} \sum_{t=0}^{T-1} \sum_{s \in S} |\mu_t(s) - \mu'_t(s)| . \quad (3.5)$$

$\mu_{\pi_{ref}}$ is defined as the future state distribution caused by following policy, π_{ref} . Defined explicitly it is

$$\mu_{\pi,t}(s) = Pr(s_t = s | s_0, \pi) . \quad (3.6)$$

Using the average variational distance assumes that the policy values are calculated

based upon the expectation of the policy sampled against its future state distribution.

Proofs for the theorem presented in Equation 3.4 can be found in [2], [3] and [9]. However, with the following problem abstraction, we will show an intuitive understanding of the correctness of this theorem.

3.3 Problem Abstraction

To apply reinforcement learning to a quadruped, the state space has 12 degrees of freedom (DOF) for the positions of the joints for the legs. This is created by 2 dimensions for a shoulder and 1 dimension for an elbow as seen in Figure 3-1. In addition, there is a 2 dimensional stability vector and a 2 dimensional velocity vector. This gives a 16 dimensional state space and the space of actions within the 12 DOF joint positions.

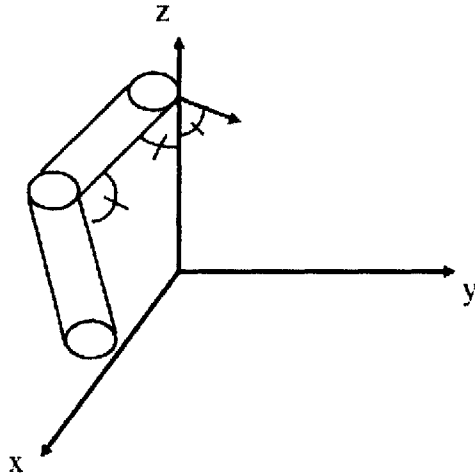


Figure 3-1: This demonstrates the three degrees of freedom per leg for the quadruped. This gives a total of 12 degrees of freedom for the four legs.

An underlying Markov model of this size is not practical to base a reinforcement

learning system on, so we use an approach that uses model-based control to abstract the state and action spaces. The controller has a model of a stable walk and step pattern. It then uses 6 control parameters to govern this model. We define the resulting problem abstraction from the underlying POMDP model for use with this problem formulation.

The resulting model for the reinforcement learning problem is as follows: Let the state space S be defined as a subset of the observations of the underlying POMDP model, so that S is the position of the robot in the y -direction of a plane. Let the actions A be defined as the 6 control parameters. Let s_0 be set to 0 and D or $\mu_0 = \{1, 0, 0, \dots\}$. This means that the robot will always be initialized at 0. This allows us to define $R(s)$ as the magnitude of the state which is the distance in the y -direction, $|s|$. We will only measure the results of the action after 10 seconds. This means that $T = 1$ time step of 10 seconds. The discount factor γ is not needed with this setting of T . The horizon H is also set to 1.

| | |
|--------------------------------------|-------------------------------------|
| S : Position in the y -direction | A : Control Parameters for Gait |
| $s_0 = 0$ | D or $\mu_0 = \{1, 0, 0, \dots\}$ |
| $R(s) = s $ | $P(s, a)$: Defined by simulation |
| T : 1 Time Step of 10 seconds | γ : Not needed |

Table 3.1: An abstraction of the underlying POMDP for use with our model-based controller. S is a subset of the observations of the underlying POMDP model.

3.3.1 Performance Bounds with this Problem Abstraction

Taking this model and applying it to PEGASUS and PSDP will show that this is a special case for both algorithms. Since the time step, T and horizon H is set to 1, non-stationary policies have no additional meaning since there are no time steps for the policy to change. Also, the function approximation for policy values in PEGASUS ends up with a single term. The remaining parts of the equation drop out when the

horizon is set to 1.

This has a direct impact on Equation 3.4. Since μ_0 is defined to be $\{1, 0, 0, \dots\}$ and the state transitions are deterministic, the future state distributions will also have a complete certainty that we will be in one particular state. This means that the average variational distance for all policies π_{ref} will be zero. In other words, $d_{var}(\mu, \mu_{\pi_{ref}}) = 0$.

This trivializes Equation 3.4 as the following for our problem formulation

$$V_{\pi}(s_0) \geq V_{\pi_{ref}}(s_0) - \epsilon . \quad (3.7)$$

This states that the optimal policy will be greater than all other policies within the error rate ϵ . Since the value of any policy is just the reward of one run in the simulation, it should be clear that this error rate ϵ is just the error rate of the simulation, which is the observation error from the underlying POMDP. Therefore by definition, the optimal policy will be greater than all other policies within the error rate of the simulation.

Similar to the proof by induction presented in [2], we can extend this to a larger time scale. It will be necessary to calculate $d_{var}(\mu, \mu_{\pi_{ref}})$. Bagnell [3] states that it is well known that for any function $f(s)$ bounded in absolute value B the following is true

$$|E_{s \sim \mu_1}[f(s)] - E_{s \sim \mu_2}[f(s)]| \leq B \sum_s |\mu_1(s) - \mu_2(s)| . \quad (3.8)$$

In our case, this is bounded by $B = 1$. This means that the average variational distance will be greater than any distance between two expectations of the policy values when sampled against their future state distributions. Therefore, since an optimal policy is derived based upon these expectations, an optimal policy will be greater than

all other policies minus the error rate and this average variational distance. This is exactly what is stated in Equation 3.4.

3.3.2 Applying Deterministic Models to the Problem Abstraction

With deterministic models, the question arises, does a deterministic simulation learn an optimal policy faster? The follow-up question is, by what standard? According to a “real world” measurement, the policy being learned will actually perform worse in the real world dependent upon the complexity of the model [15].

However, in the simulation world, the deterministic simulation should learn an optimal policy faster according to certain circumstances. To understand why, and to see how it applies with our particular problem abstraction, we need to look at exactly what we are doing.

In using a deterministic simulative model, what we end up doing with our Markov model is setting our transitions from the transition probabilities to be certain. This means that the variance from the results is zero which then means that the system is certain of its results and can react accordingly. This reduced variance is compounded with each step in the state space. In other words, the system does not have to “hedge” its bets.

However, our problem formulation has the system only take one step in the state space. While we search through the policy space, we only ever take one step in the state space. It is not possible to exploit the lower variance for future states since none ever occur.

The k-armed bandit problem [8] demonstrates the tradeoff in reinforcement learning systems between exploration and exploitation of current knowledge of the world. However, the presented system is focused on exploration of the possible policies as opposed to exploitation of the known states. It is not setup to exploit the rewards of the accumulated states. It is due to this lack of exploitation of accumulated or future states that we would expect the deterministic model to not have an impact on the rate of learning in the simulation world.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Implementation

4.1 Controlling a Quadruped



Figure 4-1: This Aibo robot, model ERS-7, was used as the development platform. Tests were conducted in simulation and verified on hardware.

4.1.1 The Designed Gait

The objective of this system is to walk over complex terrain. As opposed to a gait designed for speed, our designed controller needs to place more weight on stability. A gait designed for speed can use diagonal legs synchronously. However, our designed gait uses a single step walk. This will allow it to maintain balance over uneven terrain, since three legs will always be in contact with the ground.

Figure 4-2 shows the step ordering for the controller. Front and back steps are alternated. This designed gait has been implemented with software primitives developed at Draper Laboratory for use with other legged robots. It as been extended for use with this quadruped.

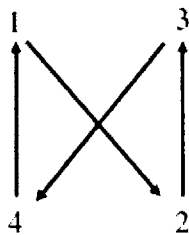


Figure 4-2: This is the gait that was used for control. Each step occurred sequentially according to the numbers in this figure.

4.1.2 Step Control

In contrast to the half ellipses used by [11], the leg movement is controlled by a quadratic equation. This is governed by 2 y-intercepts for the front and rear legs (referred to as step distances) and a uniform height for each leg. The parameters are listed in Table 4.1. The aft and fore step distances are reversed for the rear legs to make a front to back symmetrical walk. Figure 4-3 shows these parameters in reference to a leg.

The initial values used were originally from a simpler gait that were then optimized using genetic algorithm coupled with a feed-forward neural network. The values were then hand-tuned slightly.

| Parameter | Initial Value | ϵ |
|--------------------|---------------|------------|
| Shoulder Height | 130 | 3 |
| Speed | 2000 | 100 |
| Leg Lift Height | 57 | 3 |
| Fore Step Distance | -20 | 1 |
| Aft Step Distance | 4 | 1 |
| Leg Width | 6 | 1 |

Table 4.1: The control parameters along with the initial values used and the ϵ values that modify the parameters. ϵ was chosen through experimentation to give the smallest possible delta for the next parameter while still showing a measurable change in results.

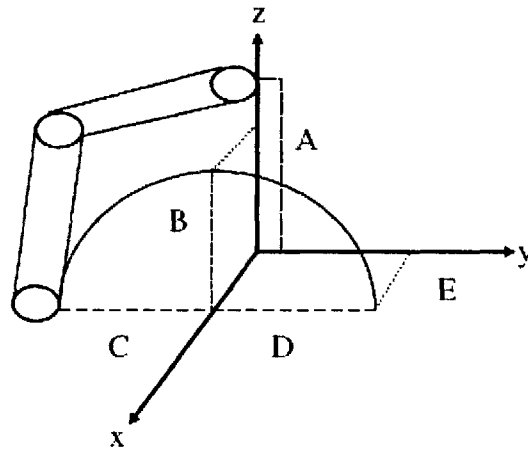


Figure 4-3: This illustrates how the parameters match against the actual Aibo leg. **A** is the shoulder height, **B** is the leg lift height, **C** is the forward step distance, **D** is the aft step distance, and **E** is the leg width.

4.2 Learning a Better Walk

The two algorithms are presented in [10]. The first, a hill climbing approach, is a straightforward, greedy local search on the policy and it serves as our baseline approach. The pseudocode is shown in Algorithm 2. Two versions of the algorithm are implemented. The first allows the new policy that is generated to possibly worsen the current score. This will occur if all the randomly sampled policies happen to worsen the score of the current policy. This, in effect, provides a limited ability to get out of local optima. A more standard greedy approach is implemented in the second version of this algorithm, where only a policy that improves upon the current score will be used as the new policy. Both of these versions of the algorithm were implemented with $t = 10$. These algorithms will be referred to as Hill Climbing 1 and Hill Climbing 2 respectively.

Algorithm 2 The greedy hill climbing algorithm. t policies are sampled near π by the ϵ given in Table 4.1.

```
 $\pi \leftarrow \text{InitialPolicy}$   
loop  
   $\{R_1, R_2, \dots, R_t\} = t$  random perturbations of  $\pi$   
  evaluate( $\{R_1, R_2, \dots, R_t\}$ )  
   $\pi \leftarrow \text{getHighestScoring}(\{R_1, R_2, \dots, R_t\})$   
end loop
```

The second algorithm presented by [10] is an N-dimensional policy gradient algorithm shown in Algorithm 3. After evaluating the t randomly generated policies surrounding the current policy, the gradient is calculated. The policy is moved along the gradient by a learning factor η . A correction that I have made to this algorithm as presented in [11] and [10] is also multiplying the adjustment vector A by the ϵ vector. This is absolutely necessary and can be easily seen as such when the parameter and ϵ values are far greater than one. Otherwise, A will always be a unit vector and the policy will hardly change. Kohl and Stone([10],[11]) were able to avoid being impacted by this since their ϵ values were all less than one and pretty much uniform. In their

implementation they, in effect, have a learning rate of 6 instead of the learning rate of 2 that they assume.

Again, two versions of this algorithm were implemented. The first allowed the policy to take an action that decreases its score as described in [10], while the second would only update the policy if the new policy actually improved the score. These algorithms will be referred to as Policy Gradient 1 and Policy Gradient 2 respectively.

Algorithm 3 The N -dimensional policy gradient algorithm. t policies are sampled near π to estimate the gradient around π . Then π is moved along the gradient by a learning factor η .

```

 $\pi \leftarrow \text{InitialPolicy}$ 
loop
   $\{R_1, R_2, \dots, R_t\} = t$  random perturbations of  $\pi$ 
  evaluate( $\{R_1, R_2, \dots, R_t\}$ )
  for  $n = 1$  to  $N$  do
     $Avg_{+\epsilon, n} \leftarrow$  average score for all  $R_i$  with  $+\epsilon$  added in dimension  $n$ 
     $Avg_{+0, n} \leftarrow$  average score for all  $R_i$  that have no change in dimension  $n$ 
     $Avg_{-\epsilon, n} \leftarrow$  average score for all  $R_i$  with  $-\epsilon$  added in dimension  $n$ 
    if  $Avg_{+0, n} \geq Avg_{+\epsilon, n}$  and  $Avg_{+0, n} \geq Avg_{-\epsilon, n}$  then
       $A_n \leftarrow 0$ 
    else
       $A_n \leftarrow Avg_{+\epsilon, n} - Avg_{-\epsilon, n}$ 
    end if
  end for
   $A \leftarrow \frac{A}{|A|} * \epsilon * \eta$ 
   $\pi \leftarrow \pi + A$ 
end loop

```

To develop an intuition for the difference between these two algorithms, it should be noticed that the hill climbing algorithm randomly samples policies around the current policy and then takes the best of the tested set. In contrast, the policy gradient algorithm develops an estimate of the gradient around the current policy from the tested set. The policy is then moved along that gradient.

Both of these versions of the algorithm were implemented with $t = 10$. This was

chosen through experimentation to show a reasonable space around the policies. Too many iterations is expensive for the algorithms, while too few will not provide a sufficient estimation of the space around the current policy. It is the same value that is used by Nate [10]. Also, the *evaluate* function for the policy gradient algorithm was implemented to score each random policy three times as recommended by Kohl [11]. This was done to help compensate for randomness in the system. This corresponded to 30 evaluations per trial for the non-deterministic PGA in contrast to Hill Climbing's 10 evaluations. The deterministic PGA does not need multiple evaluations per trial since by definition, it returns the same result each time.

4.3 Simulation of an Aibo

The four algorithms were implemented and then executed on an Aibo simulator with the initial values from Table 4.1. The simulator was built on the Yobotics Simulation Construction Set [23] and my research group has developed the Aibo models. The construction set is physics based and allows the use of terrain models. A standard trial time of 10 seconds was used and the policies were scored by the distance traveled in that time.

Chapter 5

Results and Analysis

5.1 Results

5.1.1 Simulation Results

The policy search algorithms found a 297% faster gait using the deterministic and nondeterministic models with the Hill Climbing algorithm that allowed steps to decrease the score, also referred to as Hill Climbing 1. The initial score in simulation was 67 mm/sec while Hill Climbing 1 was able to find a score of 266 or 267 mm/sec. The remaining scores are shown in Table 5.1.

| | | |
|------------------|-----------------|-------------------|
| Model | Hill Climbing 1 | Policy Gradient 1 |
| Nondeterministic | 267 | 236 |
| Deterministic | 266 | 236 |
| Model | Hill Climbing 2 | Policy Gradient 2 |
| Nondeterministic | 126 | 226 |
| Deterministic | 200 | 155 |

Table 5.1: The best scores found in simulation with each algorithm. The scores are presented in mm/sec. Hill Climbing 1 and Policy Gradient 1 allowed policies that may reduce the current score, while the other two algorithms online allowed new policies that improved the current score. Each of the algorithms were run for 200 iterations.

The optimized values for the parameters are presented in Table 5.2. This is the

approximate policy that Hill Climbing 1 converged to with both nondeterministic and deterministic models. The Policy Gradient Algorithm was never able to find the expected optimized region for the policy.

| Parameter | Initial Value | Optimal Value |
|--------------------|---------------|---------------|
| Leg Height | 130 | 112 |
| Speed | 2000 | 11000 |
| Leg Lift Height | 57 | 33 |
| Fore Step Distance | -20 | -34 |
| Aft Step Distance | 4 | 5 |
| Leg Width | 6 | 40 |

Table 5.2: The approximate optimized control parameters found by both the nondeterministic and deterministic models with Hill Climbing 1. Once the simulation was within a small difference of these expected optimal values, they had very close scores.

The joint positions for one leg with the optimal parameters are presented in Figure A-1 to A-3. This was captured for the right front leg. The corresponding torque levels used by the simulated motors are presented in Figure A-4 to A-6.

Figures 5-1 to 5-4 show the policy score for each algorithm per iteration of the simulation. This is used to demonstrate the rate of learning in the simulation world for each algorithm. The standard hill climbing algorithm consistently outperforms the Policy Gradient Algorithm. In Figure 5-2 the Policy Gradient Algorithm is shown to have a high degree of fluctuation with the deterministic model, however it is still not able to outperform the Hill Climbing algorithm as Hill Climbing rises to the 260's by the end of the trial.

The completely greedy versions of the algorithms are never able to find the expected optimal policy region as seen in Figures 5-3 and 5-4. While some have success in getting above the 200 score, they are still unable to find the policy region of the first set of algorithms. Most runs remain stuck in the lower scores. While 200 itera-

tions of the algorithms were run, the completely greedy versions remain stuck in local optima after 120 iterations.

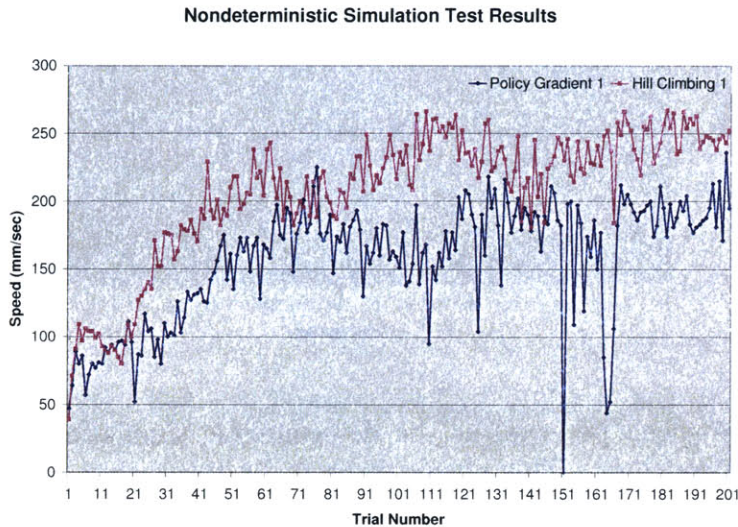


Figure 5-1: The speed per trial as the nondeterministic simulation executes the first version of the algorithms. This allows steps to be taken that may hurt the score. There were 30 evaluations per trial for the policy gradient algorithm and 10 evaluations per trial the greedy hill climbing algorithm.

5.1.2 Hardware Verification

The optimal policy was tested on the actual Aibo hardware. It was compared against the initial hand-tuned parameters. During testing, it was found that the Draper developed software primitives were unable to handle the higher leg speeds, so it had to be turned down to 5000 from 11000. None of the parameters governing the stance were modified. With these parameters, this showed a 155% increase in speed on the actual hardware. This comes out to be 0.75 leg lengths per second which passes Phase II speed requirements for DARPA's Learning Locomotion program.

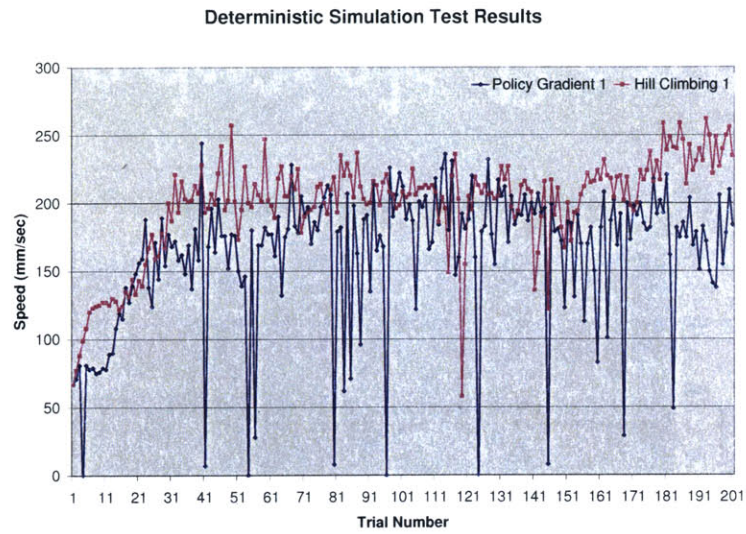


Figure 5-2: The speed per trial as the deterministic simulation executes the first version of the algorithms. This allows steps to be taken that may hurt the score. There were 10 evaluations per trial for both the policy gradient algorithm and the greedy hill climbing algorithm.

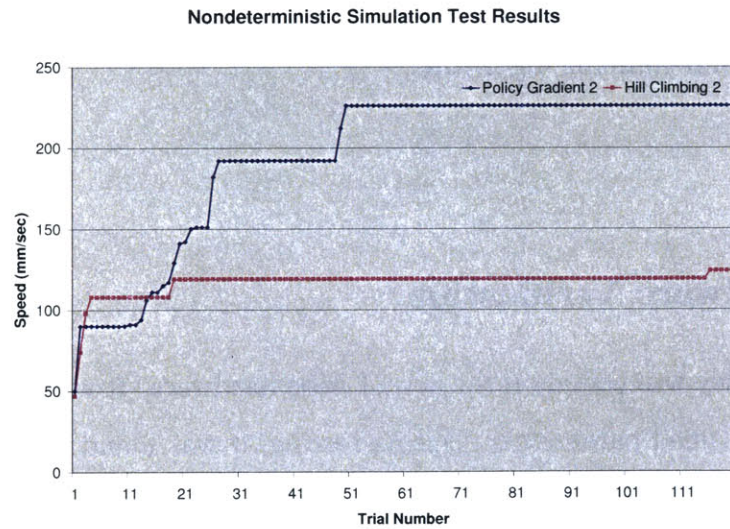


Figure 5-3: The speed per trial as the nondeterministic simulation executes the second version of the algorithms. This only allows steps that will improve the score. There were 30 evaluations per trial for the policy gradient algorithm and 10 evaluations per trial for the greedy hill climbing algorithm.

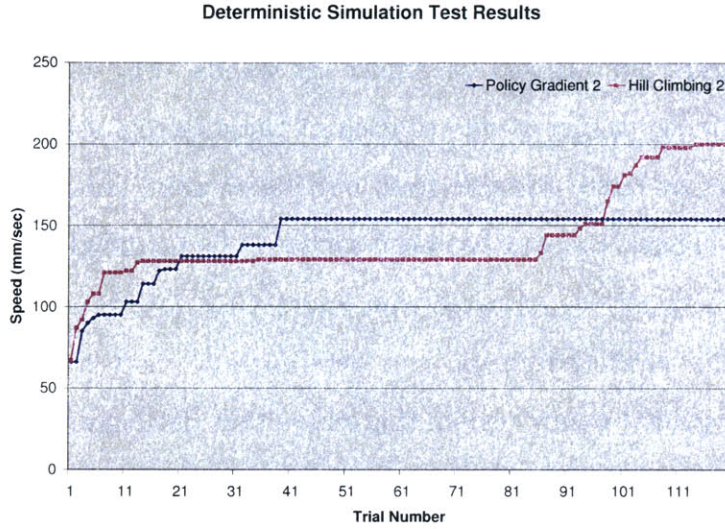


Figure 5-4: The speed per trial as the deterministic simulation executes the second version of the algorithms. This only allows steps that will improve the score. There were 10 evaluations per trial for both the policy gradient algorithm and the greedy hill climbing algorithm.

| Policy | mm/sec | inches/sec | leg lengths/sec |
|-----------|--------|------------|-----------------|
| Initial | 41 | 1.622 | 0.26 |
| Optimized | 105 | 4.138 | 0.75 |

Table 5.3: These are the scores from hardware verification. The parameters were slightly modified to handle the leg speeds allowed by the software controller. This shows a 155% increase in speed with the real hardware.

5.2 Analysis

Hill Climbing 1 found a gait that had a wider step width, a short aft step distance, and a much longer forward step distance as seen in Table 5.2. This broader stance, illustrated in Figure 5-5, allowed for higher leg speeds. The challenge for a human to find this particular stance is due to each of these factors separately causing the walk to be slower or more unstable from our initial gait. However, with the right combination of changes, the gait has been shown to be modified to be much faster.

Since both the nondeterministic and deterministic models for Hill Climbing 1 found the same optimal policy region, this shows that deterministic models with the complexity of our system do not have an impact on the real-world performance of the optimized policy.

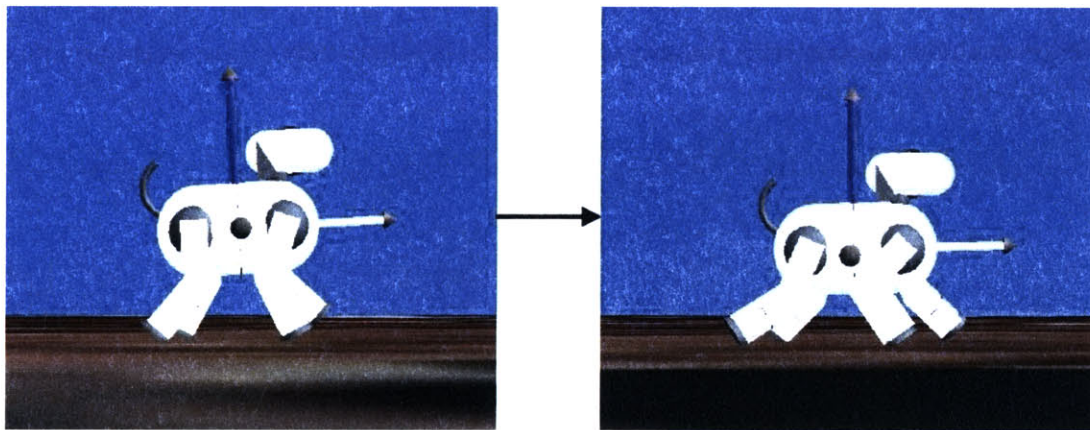


Figure 5-5: The optimally learned gait had a wider stance with a longer forward step distance. This allowed for higher leg speeds.

5.2.1 Comparing PGA and Hill Climbing

Our results show that the Policy Gradient Algorithm (PGA) is consistently outperformed by the standard hill climbing algorithm. In Figure 5-1 PGA is running three

times as slow as Hill Climbing, yet it always lags behind Hill Climbing in its score per trial. Higher learning rates do not help with PGA since it causes the gradient to be overly applied and creates even higher discontinuities in the graph. This demonstrates the challenges in calculating the policy gradient for complex models. These results show that for complex models, the discontinuities in the policy function cause hill climbing to comparatively be a much stronger tool.

When trying to calculate the gradient of the policy space, penalized scores for unstable gaits will skew the gradient. Upon inspection of the policy gradient algorithm, it can be seen that the adjustment vector takes into account the difference between the two sets of modified scores. Therefore if one of the modified scores is overly penalized, the gradient will not be accurate.

As a simple illustration for how a policy can be overly penalized and skew the gradient, imagine a policy that is walking on the edge of stability. Due to an amount of randomness in the system, this policy is equally likely to fall at 4 seconds or 8 seconds. For this particular set of policy scores, the negative score may fluctuate randomly as much as 100% of the lowest score. When it is very easy to stumble across these failed policies, the gradient estimation suffers from the high amount of randomness.

To attempt to alleviate the discontinuities, a higher number of evaluations can be done for each policy in the nondeterministic model. In these tests, I used the recommended 3 evaluations per policy [11]. While I can attempt to further increase the number of evaluations per policy, it can be shown that this average score will go to the expectation of the probability distribution of when the gait becomes unstable. In the case of a uniform distribution, this will give an expected penalty score of $\frac{1}{2}$ the normal score. This is still quite a bit and will likely still skew the gradient. It may make more sense to assume that it is a geometric distribution with an expected

penalty score of $\frac{1}{p}$ the normal score, where p is the probability that it will fail at any given time. The challenge then becomes estimating this p in a reasonable amount of time and deciding what a 'normal' score should be. In contrast, the hill climbing algorithm simply ignores these penalized scores and only takes the best policy found.

With the deterministic model, it does not make sense to sample the policy more than once. Since the state transitions are deterministic, the resulting state for each action will always be the same. While this increases the performance in time of the algorithm, it does not help with the rate of learning the optimal policy.

Kohl and Stone[10] were able to show that the Policy Gradient Algorithm and hill climbing algorithm greatly outperform other reinforcement approaches to this problem. In their case, the policy gradient approach slightly edged out the hill climbing approach. However, they are using a gait that is optimized for speed over a flat Robocup soccer field and probably much less likely to fall over than our gait.

There is definitely an art to finding the proper initial parameters. While the ranges for the parameters were known, certain combinations of the parameters in the proper ranges would not work at all, causing the robot to fall over. Because of this, common methods for dealing with local optima were not available. Using a random restart for the policy was likely to cause the controller not to work at all. The parameters had to have some initial intelligence in their selection. Simulated annealing could be made to work but can initially cause the same effect if the starting amount of added randomness was too great.

In analyzing the differences in optimal policies between Policy Gradient 1 and 2, and then Hill Climbing 1 and 2, it shows the limited ability to escape local optima provides better performance for the gait. This method forced a move from a position even if a better policy was not found. Instead the best policy or the best

gradient was taken of those sampled nearby regardless of the outcome. Without any ability to escape local optimal, the second version of the algorithms performed poorly.

Finally, the graph of the policy gradient results in Figure 5-2 demonstrates why it is so difficult for humans to set these parameters. The outliers on the graph are caused by the discontinuity of this function. The policy is set to the best expected result along the gradient, which may end up drastically decreasing the actual score. Just having general knowledge of the expected gradient, in this case that a wider, longer stance should be faster and more stable, does not help when trying to tweak these parameters by hand. The original hardware score of 1.6 inches per second was not improved upon by hand because the research team became bogged down by the discontinuities in this high dimensional policy space.

5.2.2 Comparing Models

These results show that there is little impact on policy scores when using deterministic models. Figure 5-6 shows that trial runs closely follow each other. While initially with the Hill Climbing algorithms, the deterministic models look like they are performing better than the nondeterministic models, the functions quickly converge together and eventually move up to the optimal policy region. The same holds true for the Policy Gradient Algorithms, however, neither converge to the optimal policy region.

This corresponds to the expectation that there will be little benefit to the learning rate in the simulation world from using deterministic models as stated in Section 3.3.2. This is in regard specifically to the problem abstraction of the underlying Markov model. Those as discussed earlier, a more complex Markov model may be able to take advantage of the deterministic transitions. Even though the learning rate was not improved, there is a benefit of using deterministic transitions since it allows the use of dynamic programming which speeds up the overall execution time of the system. Dynamic programming allows us to use a table to lookup the results of an

action from a state if it has been calculated once, instead of having to simulate the 10 seconds with the nondeterministic noise.

The impact on performance in the real world was not seen since Hill Climbing 1 was able to find the optimal region with both models. With a more complex Markov model, this factor should continue to be watched.

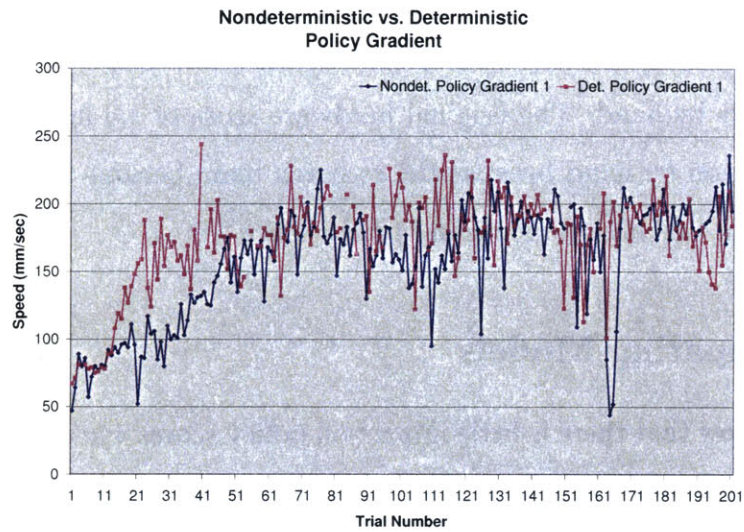


Figure 5-6: This is a comparison of the trial runs with nondeterministic and deterministic models with the Policy Gradient Algorithm. Outliers have been removed for clarity.

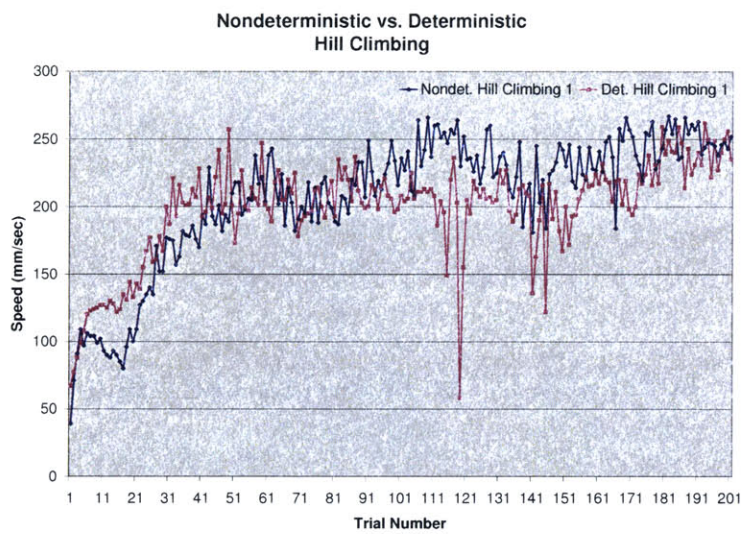


Figure 5-7: This is a comparison of the trial runs with nondeterministic and deterministic models using the Hill-Climbing algorithm.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Conclusion and Summary

6.1 Conclusion

Policy Search is an effective Reinforcement Learning approach for optimizing controllers. This work demonstrates that using a standard hill climbing approach can be more rewarding than trying to generate a policy gradient approach. While performance may be achieved by smoothing out discontinuities or generating more intelligent methods for calculating the gradient, hill climbing is still difficult to beat.

The penalty in time of the policy gradient approach and the high performance of hill climbing suggests that hill climbing should be considered first when designing policy search for highly complex models.

The benefit of using dynamic programming on policies can greatly speed up the performance of the algorithms in computational speed. Implementing deterministic transitions to allow the use of dynamic programming has little impact on the performance of our system with the problem abstraction of the Markov model.

We also show that our abstraction is a special case for PEGASUS and PSDP. This special case allows an intuitive understanding of the performance guarantees of this

set of algorithms. The abstraction of the Markov model simplifies the state space for use with our model-based controller.

6.2 Future Work

While this work has been able to complete one requirement of the DARPA Phase II requirements, the other requirement is still a challenge. The quadruped is required to be able to step over an obstacle at two thirds its leg length.

To overcome obstacles, the consensus is to integrate reactive control to maintain stability. The challenge will be to integrate this stability control with a reinforcement learning system to intelligently control steps. This work contributes towards that goal by investigating the impact on the use of deterministic transitions on learning performance.

As step control is integrated with stability control, it will be necessary to redesign the abstraction of the Markov model for the reinforcement learning system. The new model should make better use of the deterministic transitions to improve the rate of learning while in simulation. This new model would be able to make better use of the more advanced algorithms in Policy Search such as PEGASUS and PSDP.

A part of the challenge in using these algorithms will be to calculate the initial state distribution for use in estimating the value of a policy. To overcome this, passive learning of the state distribution of a real animal quadruped may be useful. Once this initial state distribution is calculated, dynamic programming for policies will be even more powerful which would allow a better integration of reinforcement learning with stability control and allow the robot to take more intelligent steps.

6.3 Summary

This thesis shows that Policy Search is an effective means of optimizing a controller. We demonstrated a 155% increase in performance on hardware. This work also demonstrates that standard hill climbing works well in comparison to a policy gradient approach for complex models. Deterministic transitions allow the use of dynamic programming to speed up system performance at little impact to the overall learning performance both in simulation and the real world.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Simulation Data

A.1 Joint Positions

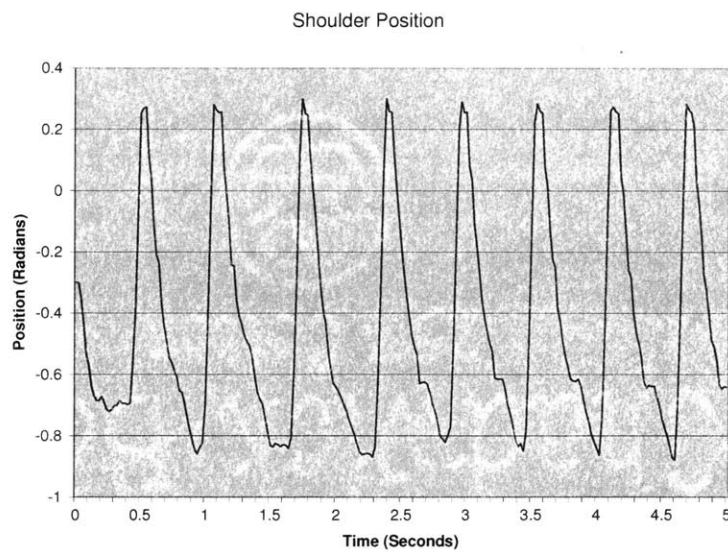


Figure A-1: The position of the shoulder joint for the right front leg over 5 seconds. This position is commanded with the optimal found policy.

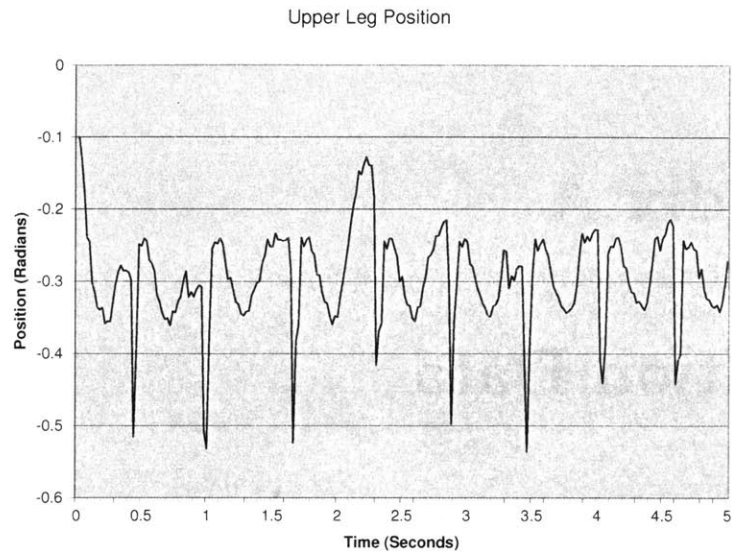


Figure A-2: The position of the upper leg joint for the right front leg over 5 seconds. This position is commanded with the optimal found policy.

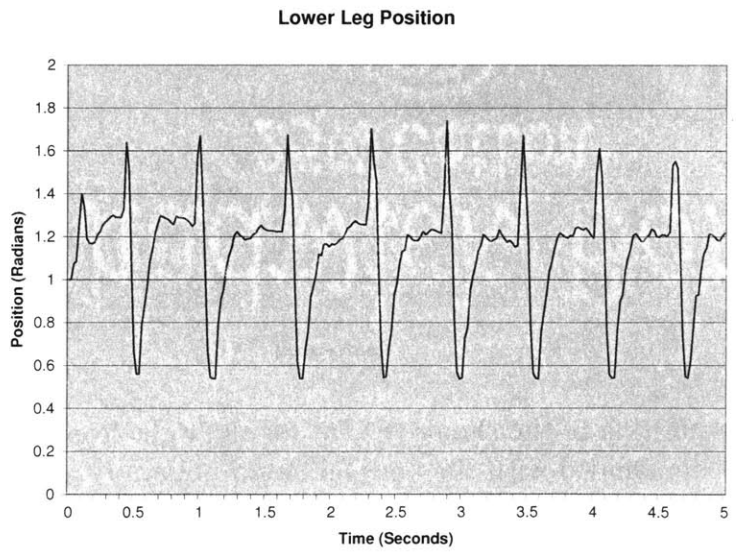


Figure A-3: The position of the lower leg joint for the right front leg over 5 seconds. This position is commanded with the optimal found policy.

A.2 Torque Control

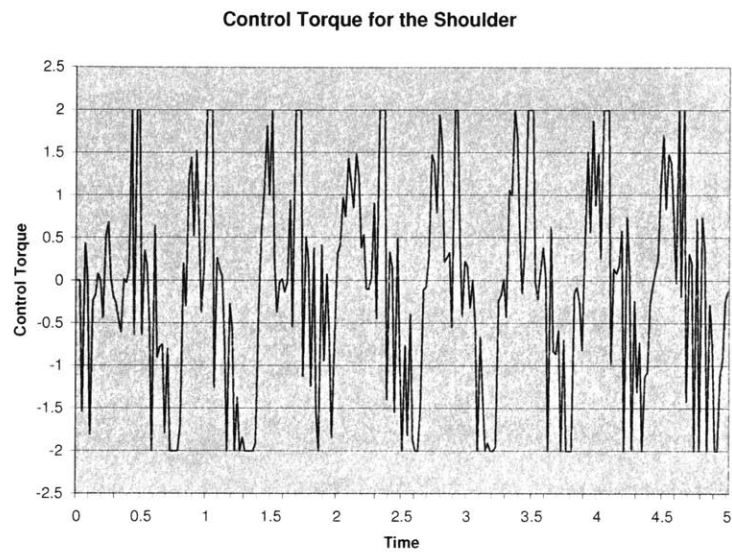


Figure A-4: The torque of the shoulder joint for the right front leg over 5 seconds. This torque is controlled by the optimal found policy.

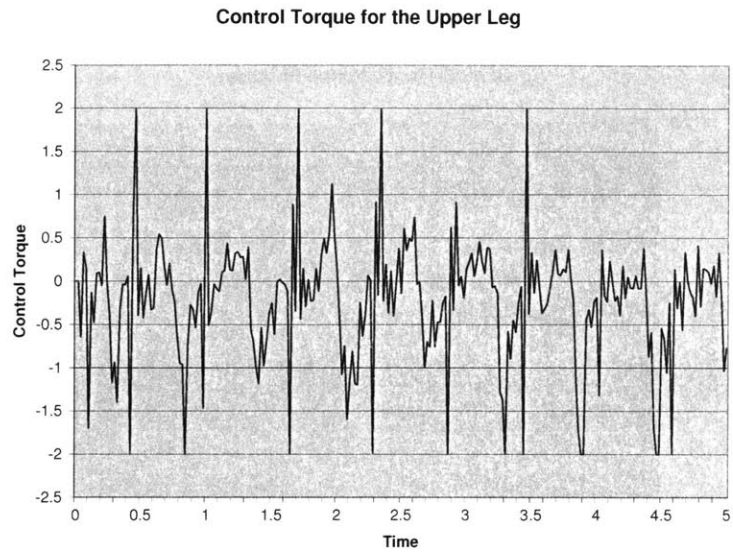


Figure A-5: The torque of the upper leg joint for the right front leg over 5 seconds. This torque is controlled by the optimal found policy.

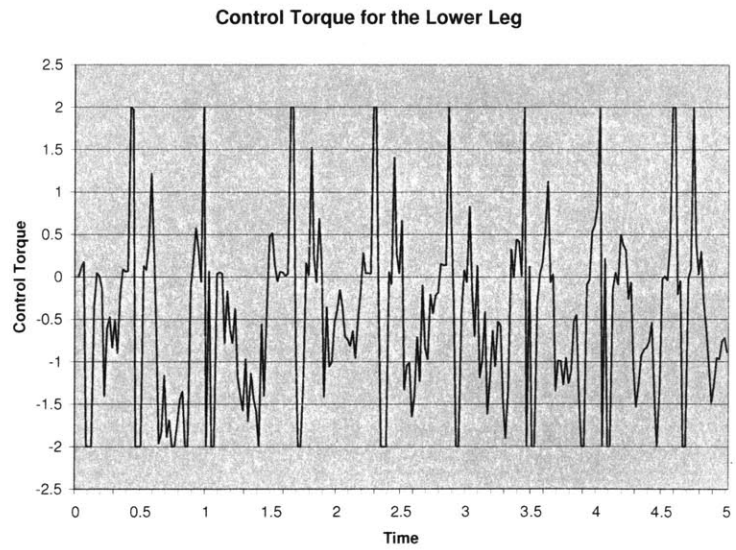


Figure A-6: The torque of the lower leg joint for the right front leg over 5 seconds. This torque is controlled by the optimal found policy.

Appendix B

Acronyms

DOF Degrees of freedom

MDP Markov Decision Process

PEGASUS Policy Evaluation of Goodness And Search Using Scenarios

PGA Policy Gradient Algorithm

POMDP Partially Observable Markov Decision Process

PSDP Policy Search by Dynamic Programming

RL Reinforcement Learning

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

- [1] Pieter Abbeel and Andrew Y. Ng. Learning first-order markov models for control. In *Advances in Neural Information Processing Systems 17 (NIPS-04)*, 2005.
- [2] J. Andrew Bagnell. *Learning Decisions:Robustness, Uncertainty, and Approximation*. PhD thesis, Carnegie Mellon University, 2004.
- [3] J. Andrew Bagnell, Sham Kakade, Andrew Y. Ng, and Jeff Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, Vancouver, 2004.
- [4] J. Andrew Bagnell and Jeff Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation*, May 2001.
- [5] J. Andrew Bagnell and Jeff Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, August 2003.
- [6] Tao Geng, Bernd Porr, and Florentin Woergoetter. Fast biped walking with a reflexive controller and real-time policy searching. In *Advances in Neural Information Processing Systems 18 (NIPS-05)*, 2006.
- [7] Leemon C. Baird III. *Reinforcement Learning through Gradient Descent*. PhD thesis, Carnegie Mellon University, May 1999.

- [8] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [9] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.
- [10] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, July 2004.
- [11] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.
- [12] Ivo Kwee, Marcus Hutter, and Jurgen Schmidhuber. Gradient-based reinforcement planning in policy-search methods. In *Proceedings of the 5th European Workshop on Reinforcement Learning*, 2001.
- [13] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics*, 2004.
- [14] Gerhard Neumann. The reinforcement learning toolbox: Reinforcement learning for optimal control tasks. Master’s thesis, Graz University of Technology, Austria, 2005.
- [15] Andrew Y. Ng and Michael I. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, 2000.
- [16] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, Vancouver, 2004.

- [17] Leonid Peshkin. *Reinforcement Learning by Policy Search*. PhD thesis, Massachusetts Institute of Technology, 2001.
- [18] Nicholas Roy. *Finding Approximate POMDP Solutions Through Belief Compression*. PhD thesis, Carnegie Mellon University, 2003.
- [19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter 17. Prentice Hall, 2003.
- [20] Scott Sanner and Craig Boutilier. Approximate linear programming for first-order MDPs. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, 2005.
- [21] Russ Tedrake, Teresa Weirui Zhang, and H. Sebastian Seung. Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, 2005. Yale University.
- [22] G. Theodorou and L. Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, Vancouver, 2004.
- [23] Yobotics, Inc. *Yobotics Simulation Construction Set*. <http://www.yobotics.com/simulation/simulation.htm>, 2005 edition.