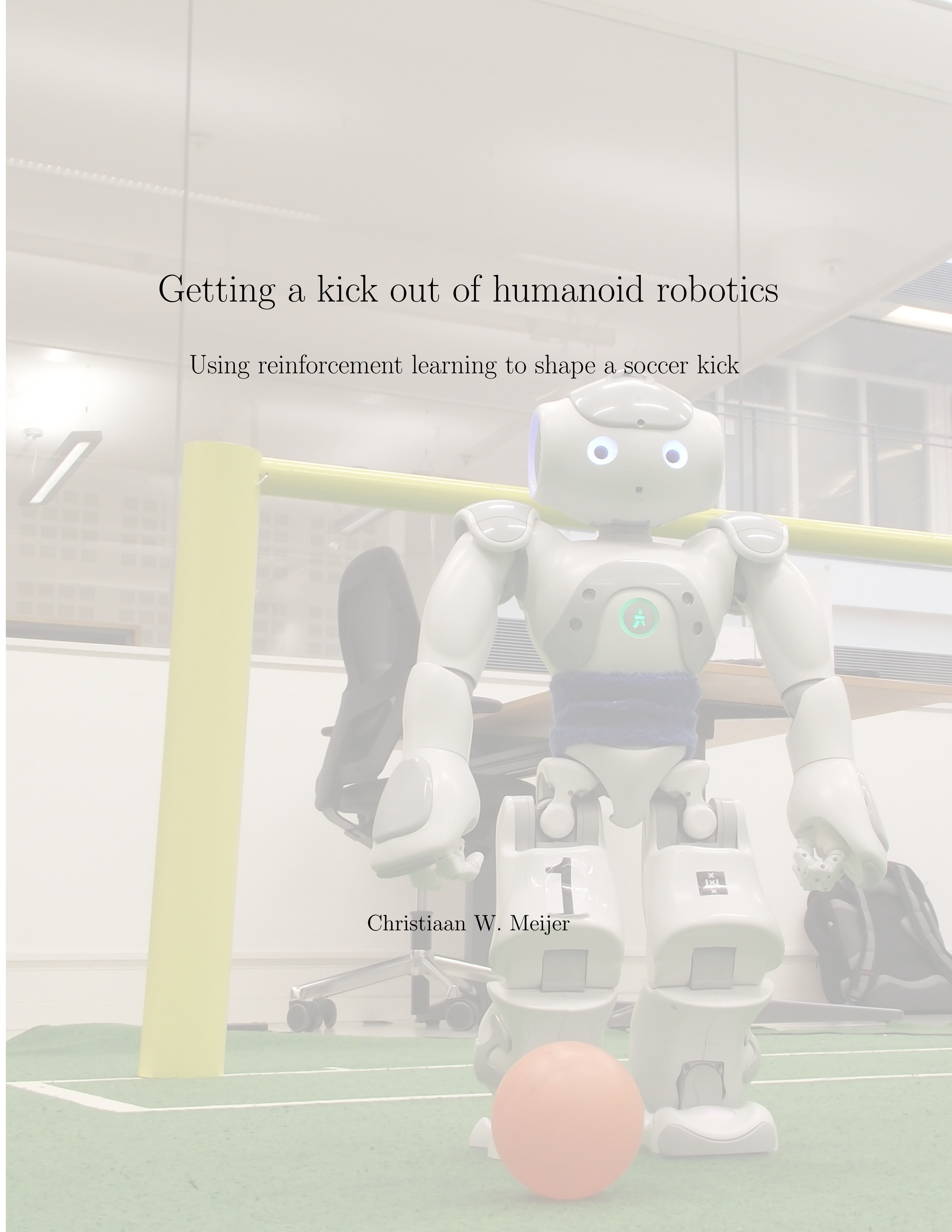


Getting a kick out of humanoid robotics

Using reinforcement learning to shape a soccer kick

Christiaan W. Meijer

Getting a kick out of humanoid robotics

Using reinforcement learning to shape a soccer kick

Christiaan W. Meijer
0251968

Master thesis

Master Artificial Intelligence, track Intelligent Systems

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisors
dr. S. A. Whiteson
dr. A. Visser

Intelligent Systems Lab Amsterdam
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July 7th, 2012

Abstract

Robots at home and in other uncontrolled environments need to be versatile. Time designing behavior for each task can be drastically reduced when robots can optimize behavior or learn it from scratch. Using reinforcement learning, a simulated humanoid robot, the NAO, is trained to perform a soccer kick. The used policy maps poses in joint space, encoded in policy parameters, via splines to motor settings. Shaping was applied by using different reward functions for different stages of the learning process. Direct policy search algorithms were used for learning. More specifically, the used learning algorithms are Finite Difference, Vanilla, Episodic Natural Actor Critic and a Genetic Algorithm. Results were compared. Using Finite Difference and the Genetic algorithm yielded the best results. Poor performance of Vanilla and Episodic Natural Actor Critic algorithms was probably due to error in the estimation of the gradient of the policy with respect to the parameters. Estimating that gradient is not needed for Finite Difference and the Genetic algorithm. Shaping was applied in two different approaches. In one, shaping resulted in reduced performance. In the other, performance was improved by shaping. While a good soccer kick was learned from scratch, the best soccer kick was the result of optimizing an already existing policy.

Acknowledgements

I want to thank my supervisors for their support and patience. With their knowledge and expertise, I was able to do the research and write this thesis.

I want to thank Jan Peters for taking the time to answer many specific questions concerning Episodic Natural Actor Critic which helped me understand the algorithm.

I want to thank my girlfriend who helped me to form solutions for practical problems and to make schedules.

I want to thank my parents for their financial and emotional support. I also want to thank my mother's husband who lent me his laptop for over half a year. Without it, the experiments in this thesis could not have been done.

Last but not least, I want to thank my fellow students for being such good sparring partners. They made these final months that were sometimes difficult, also a joy.

Contents

Abstract **i**

Acknowledgements **iii**

1 Introduction **1**
1.1 Related work 3
1.2 Research questions 5
1.3 Thesis overview 5

2 Background **7**
2.1 Reinforcement Learning 7
2.2 Policy Gradient 9
2.2.1 Base algorithm 10
2.2.2 Finite Difference 11
2.2.3 Vanilla 12
2.2.4 Episodic Natural Actor Critic 15
2.3 Genetic Algorithm 18

3 Solution Approaches and Problem Setting **21**
3.1 Problem setting 21
3.2 Policy 22
3.2.1 Parameterization 22
3.2.2 Exploration 24
3.2.3 Cubic interpolation 25
3.2.4 Estimating the gradient of the policy 27
3.3 Starting parameters 28
3.4 Reward functions 29
3.5 Shaping a kick 32

4 Experiments **35**
4.1 Simulator 35
4.2 Comparative experiments 37
4.2.1 Learning a subset of parameters using all algorithms . 37
4.2.2 Learning all parameters using only finite difference and genetic algorithm 41
4.3 Approach validation experiments 42

5 Discussion and Future Work **47**
5.1 What algorithm is most effective for learning a soccer kick? . . 47
5.2 What is a good approach for learning a soccer kick on a humanoid robot using reinforcement learning? 49
5.3 Future research 50
5.4 Conclusion 51

A Finding the optimal step size using Lagrangian multipliers **53**

B Episodic Natural Actor Critic Derivation **55**

Chapter 1

Introduction

Figure 1.1: Romeo is a prototype under development, designed to assist people at home.

In the cartoon The Jetsons from the 60s, the starring family was assisted by a versatile cleaning robot. This idea is more and more becoming reality. Many robots are designed to do a specific task like vacuum cleaning or lawn mowing. Others are designed to perform a greater range of tasks with the purpose of assisting or entertaining people. One such robot is the humanoid NAO by Aldebaran Robotics. This robot, designed for entertainment and research, is too small to be able to assist people around the house. A bigger humanoid robot, a project led by Aldebaran Robotics, is the French Romeo in Figure 1.1. This robot is a 1.43 meter tall prototype, that is still under development. Romeo is being designed to help elderly and other individuals who need help in their daily actions. This robot will be able to open doors, grasp items like key chains, and help fallen people. Designing these individual

actions by hand is time-consuming. If robots can learn to optimize such actions, or even learn them from scratch, the time needed to implement many actions would be reduced drastically.

An environment where the NAO needs to be versatile is that of the robot soccer competitions organized by RoboCup. RoboCup is an international initiative to stimulate research in the field of robotics. RoboCup organizes annual soccer tournaments for robots since 1997 called RoboCup Soccer. The famous ultimate goal of the RoboCup organization that it has set for itself, is as follows:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.

Through the years, RoboCup has expanded by also organizing competitions of other games for robots. Besides RoboCup Soccer, RoboCup currently organizes competitions in three other types of games. In RoboCup Rescue, teams of robots need to navigate through and search simulated disaster zones. In RoboCup @Home, robots compete in tests that focus on abilities needed for assisting people in home environments. RoboCupJunior is organized to introduce robotics to young students.

Like most RoboCup competitions, RoboCup Soccer has several leagues. In Middle Size and Small size leagues, robots usually on wheels play with the focus on team play and strategy. In the Humanoid league, human-like robots of different sizes, designed by the competing teams themselves, compete with each other. In the Standard Platform league, all robots are identical, meaning that purely the software determines who wins. The robot currently used for this league is the humanoid NAO, discussed below. In the past, Sony's Aibo dog-like robot fulfilled this role. Another league is the simulation league in which models of the NAO compete in a virtual world.

The NAO is a humanoid robot created by Aldebaran Robotics [1]. Several versions have been available. The first version that was released was the RoboCup version which became available in 2008. The robot had 21 degrees of freedom. Later that year, an academic version was released that had additional degrees of freedom in the hand area of the robot. In December

[1] http://www.aldebaran-robotics.com/

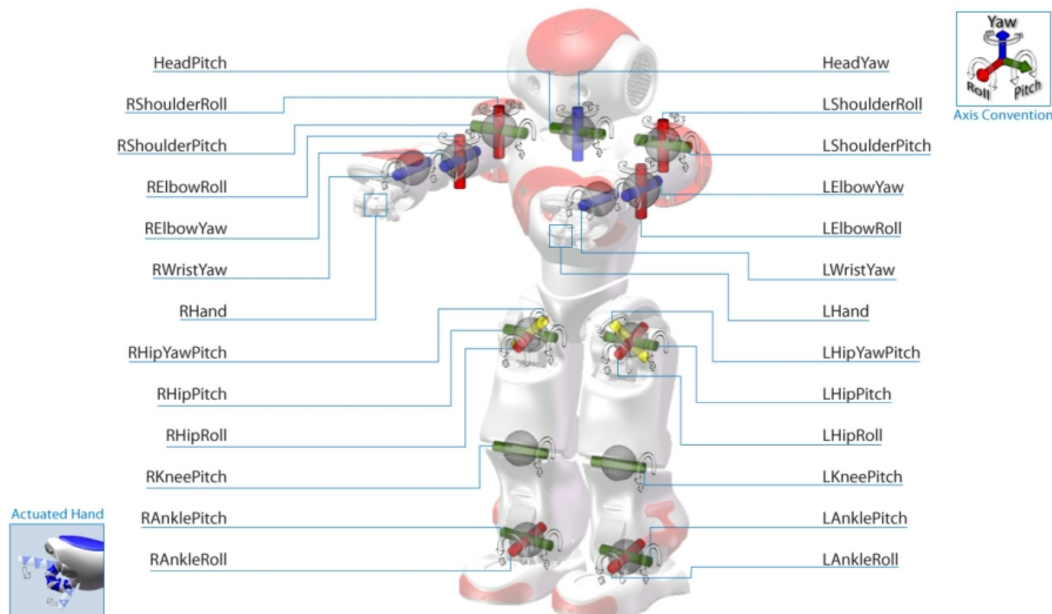


Figure 1.2: The degrees of freedom of the humanoid NAO robot by Aldebaran Robotics. [20]

2011, a new generation of the NAO was announced, the NAO 4.0 [2] which is currently the latest model. Figure 1.2 shows the NAO and the position of its joints.

Because of RoboCup Soccer, there is a large amount of literature about developing soccer behavior. To build on this literature, the research of current thesis focuses on teach a NAO to kick a ball. This was done using reinforcement learning.

1.1 Related work

RoboCupSoccer is an often used environment in which to test reinforcement learning methods. Research to improve soccer play have led to many publications. Robots have learned to dribble [19] or to walk as fast as possible [12, 13]. Often, reinforcement learning has been done to fine tune parameters

[2] http://www.engadget.com/2011/12/10/aldebaran-robotics-announces-nao-next-gen-humanoid-robot-video/

of some existent behavior, often designed by hand [23]. For instance, a robot had learned to adapt its soccer kick to small variations in the ball's location [11].

In research by Hausknecht and Stone [10], quadrupedal robots learned to pass a ball without an initial working policy of a kick. To do this, Hausknecht and Stone used a policy that consisted of a number of poses which were in turn defined by the desired position of all the joints of the robot. In other words, they defined the pose of the robot at several moments during the kicking motion. Because a quadrupedal robot was used, it was assumed that effective kicks would be symmetrical. By only including symmetrical policies in the search, the parameter space could be reduced. To further reduce the number of parameters that had to be learned, some joints that did not seem relevant for kicking the ball, like those in the tail of the robot, were fixed and not included in the search. Parameters were added for the timing of the poses. Hausknecht and Stone [10] used a Policy Gradient algorithm to find parts of the 66-dimensional parameter space that resulted in an effective kick. These behaviors were fine tuned afterwards using a hillclimbing algorithm. Learning on a real robot resulted in a powerful kick. The robot had to try out 650 different kicking techniques during the Policy Gradient stage and another 135 for the hillclimbing stage.

Hausknecht and Stone used a 'vanilla' [26, 3, 16] Policy Gradient algorithm to successfully learn a kick. Peters et al. derived an algorithm based on this Vanilla Policy Gradient called Episodic Natural Actor Critic [16]. Peters and Schaal have compared learning performance of Vanilla and Episodic Natural Actor Critic [15], and a Finite Difference algorithm on a single degree of freedom motor control task. All three algorithms are described in Section 2. When learning to pass through a point using splines, the Episodic Natural Actor Critic outperformed both other algorithms, while Finite Difference performed better than the Vanilla Policy Gradient algorithm.

Searching for policies can be facilitated by shaping. The term shaping was invented by the psychologist Skinner [24]. He used the term for teaching animals behavior by rewarding them for increasingly difficult tasks. This behavior would have been too complex for animals without experience to learn directly by rewarding only the ultimately targeted behavior. In the field of artificial intelligence, the idea of shaping has been applied to facilitate machine learning. By learning on tasks with increasing difficulty, a policy for controlling a robot hand was learned that could not be found without shaping [9]. Shaping has accelerated the search for a policy for controlling

a simulated bike [18]. For an overview of shaping methods for in artificial intelligence see [7].

1.2 Research questions

Section 1.1 described that in the research of Hausknecht and Stone, a policy for a soccer kick for a quadrupedal robot was learned from scratch. The research of the thesis focuses on learning a soccer kick on a humanoid robot instead of a quadrupedal robot. A bipedal, humanoid robot is a more difficult subject to learn behavior on as more effort, compared to a quadrupedal, is needed to prevent the robot from falling over. Also, a bipedal robot's kicking technique will have to be asymetrical and will therefore have more parameters that need to be learned. While parameters of soccer kicks in bipedals have been tuned using machine learning, to the best of the author's knowledge, no kick has been learned without an initial working policy. Learning this new behavior using existing methods can therefore validate methods described in the literature. For these reasons, the first research question of this thesis is as follows. What is a good approach for learning a soccer kick on a humanoid robot using reinforcement learning?

An important part of the approach is the choice of a learning algorithm. In the research of Hausknecht and Stone a soccer kick was learned on a quadrupedal robot without a working policy using Vanilla Policy Gradient [10]. The Vanilla Policy Gradient, Episodic Natural Actor Critic and Finite Difference algorithms have been compared on a single degree of freedom motor control task [15]. A comparison on a complex task like learning a soccer kick has not yet been done. The second research question of this thesis is therefore as follows. What algorithm is most effective for learning a soccer kick?

Note that thoughout this thesis, the word *approach* is reserved for the set of design choices that need to be made to develop a soccer kick and the word *algorithm* is used to refer to a reinforcement learning algorithm.

1.3 Thesis overview

In Chapter 2 reinforcement learning and Markov decision process, an often used framework for describing control tasks in, are introduced. Learning al-

gorithms used in the research of this thesis are also described in this section. In Chapter 3 the problem setting is defined and details about the used policy model are described as well as reward functions that were used. In Chapter 4, experiments are discussed that compare the performance of the used algorithms. Also, the experiments are discussed that test the approach described in Chapter 3. Results are discussed and future research is suggested in Chapter 5.

Chapter 2

Background

In this chapter, the necessary theoretical background is described to understand the approaches taken to learn a soccer kick described in Chapter 3. First, the field of reinforcement learning is briefly introduced in Section 2.1 and the reinforcement learning algorithms that have been used in the research of this thesis are described. Policy Gradient learning algorithms are discussed in Section 2.2 and Genetic Algorithms in Section 2.3.

2.1 Reinforcement Learning

In this section, the concept of reinforcement learning is described. In reinforcement learning problems, an agent is typically learning to perform some task. This is often described as a Markov Decision Process. At each timestep, the agent observes the current state $\mathbf{x}$ from all possible states X and takes an action $\mathbf{u}$, a real valued vector, from all possible actions U. It then receives a reward signal r from $\mathbb{R}$ according to reward function f which maps state $\mathbf{x}$ to a real value. A transition function maps the state and action to a probability distribution from which the state of the next time step is drawn. The process is illustrated in Figure 2.1.

A finite horizon is used, which means that the process is stopped after a constant finite number of steps. It is the agent's goal to maximize the return which is the sum of the rewards, received until the horizon H, which marks the end of the task. The agent has a policy $\pi_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{u})$ which is a mapping from the current state $\mathbf{x}$, action $\mathbf{u}$ and the policy parameters $\boldsymbol{\theta}$ to the probability of taking action $\mathbf{u}$ given state $\mathbf{x}$ and policy parameters $\boldsymbol{\theta}$. The goal

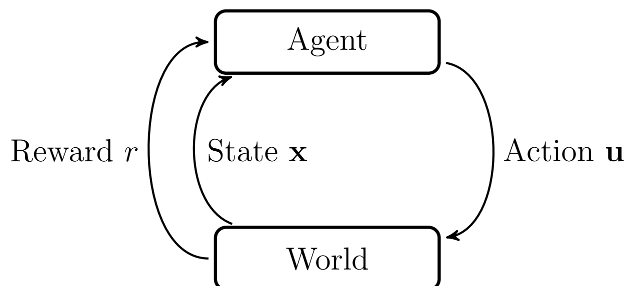


Figure 2.1: Markov Decision Process. The agent performs an action after which the world generates a new state and reward which are both observed by the agent.

of reinforcement learning is to find policy parameters for which the expected return J is maximal. For a more detailed description of the Markov Decision Process, see [5].

Note that the only feedback the agent receives is the reward. This is an indirect form of feedback. It is indirect in the sense that the agent is never given the actions it should have taken at each timestep. It is up to the agent to analyze rewards and to deduce which actions were better than others. The agent is also never given information about what actions specifically caused it to receive a reward. All actions before receiving a reward might have contributed to receiving it.

A distinction can be made between the ways how learning algorithms collect data. When a current policy is used to perform rollouts and collect data, this is called on-policy learning. An alternative for this is to generate random states or to use data collected by performing rollouts with a different policy. When the policy that is learned and the policy that is used for collecting data is not the same, this is called off-policy learning [6]. In the research of this thesis data needs to be collected by the robot itself. For this reason, it is most practical to use the policy that is learned, to perform rollouts with. This thesis focuses therefore on on-policy learning algorithms.

Traditionally, the field of reinforcement learning has focused on developing methods to learn tasks with discrete state and action spaces. This means that a finite number of states and actions exist. Examples of such tasks are games like chess, checkers and backgammon. On the other hand, tasks in the field of robotics generally have continuous state and action spaces. This

means that an infinite number of states and actions exist and a single state is generally not encountered twice by an agent. In theory, every continuous dimension can be discretized by dividing its values over a finite set of bins. Choosing these bins can be problematic. Also, the number of bins needed to accurately classify the state, grows exponentially with respect to the number of dimensions of the state space. For these reasons, many reinforcement learning methods are not easily applied to tasks with continuous state and action spaces.

Algorithms can be divided over three categories. In Value Iteration methods, an estimate of the value of each state is iteratively improved. An example of such an algorithm is the widely used Q-Learning [29]. Policy Iteration, also uses estimates the value of states to generate optimal policies given this estimated value function. An example of this family is the SARSA [22] algorithm. Both families of algorithms depend on learning a value function and learning a transition function. Because of the high dimensial state and action space associated with robotics with a humanoid robot, members of these families of algorithms are difficult to apply. A family that can be applied without difficulty to continuous state and action spaces with many dimensions is the Policy Search family. These algorithms are model free and no value function has to be learned [6]. The thesis focuses therefore on Policy Search algorithms.

Two important Policy Search methods are the Policy Gradient algorithms and Genetic Algorithms. Policy gradient methods are designed for continuous state and action spaces, which makes them appropriate for the research in this thesis. In fact, for Policy Gradient methods, continuous state spaces have to be assumed. Other algorithms which have proven to be succesful in continuous state and actions spaces are Genetic Algorithms. This thesis focuses on the Policy Gradient family of algorithms which is discussed in section 2.2 and Genetic Algorithms which are discussed in section 2.3.

2.2 Policy Gradient

In this section Policy Gradient methods, a specific family of algorithms used for reinforcement learning tasks, are described. In section 2.2.1, the general idea of policy gradients is explained. After that, three policy gradient algorithms are discussed. The *Finite Difference* algorithm, which will serve as a base method, is described in Section 2.2.2. In section 2.2.3 the Vanilla gra-

dient algorithm is described. This algorithm forms the basis for the *Episodic Natural Actor Critic* algorithm which is explained in section 2.2.4.

2.2.1 Base algorithm

The goal of policy gradient methods is to find the column vector of parameters $\boldsymbol{\theta}$ of a policy that maximizes the total expected reward $J(\boldsymbol{\theta})$ of an agent applying the policy.

$$\boldsymbol{\theta}_* = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmax}}\, J(\boldsymbol{\theta}) \tag{2.1}$$

The idea is to start with initial parameters $\boldsymbol{\theta}_{start}$. Let us assume that $J(\boldsymbol{\theta})$ is continuous over the set of all possible parameters Θ. For each element of $\boldsymbol{\theta}$ we could determine whether to increase or decrease it to improve the policy. Similarly, we could calculate the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Because $J(\boldsymbol{\theta})$ is continuous, a sufficiently small step into the direction of the gradient, improves the expected reward $J(\boldsymbol{\theta})$. More formally, the update step can be written

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) \tag{2.2}$$

where α is the step size, or learning rate, and t the timestep. The above process is repeated until some predefined stopping criteria are met. The algorithm is summarized in Algorithm 1.

Algorithm 1 Policy gradient

1: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{start}}$
2: **repeat**
3:    estimate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$
4:    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$
5: **until** stopping criteria are met
6: **return** $\boldsymbol{\theta}$

Stopping criteria can be convergence of the gradient or a fixed number of iterations. Note that following the gradient until convergence does not guarantee to find the globally optimal parameters. It is possible to converge to, or to get stuck at, a local optimum. The choice of the stepsize is also important. While a small stepsize can result in slow learning, a too large stepsize can cause the search to overshoot and not find the optimum. Whether the

algorithm will find the global optimum depends on the stepsize, the shape of $J(\boldsymbol{\theta})$ as well as the parameters with which the process initialized.

Now that just the main loop has been described, a way must be found to get gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. This thesis focuses on three algorithms that estimate this gradient. Finite Difference, Vanilla, and Episodic Natural Actor Critic algorithms return gradients $\mathbf{g}_{\text{FD}}$, $\mathbf{g}_{\text{GMDP}}$ and $\mathbf{g}_{\text{NG}}$ respectively. These gradients can be inserted as $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in the main policy gradient loop. In the following sections, these three different algorithms for estimating gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ are described.

2.2.2 Finite Difference

The Finite Difference method is a simple way to estimate the gradient of the expected return with respect to policy parameters, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. The method has been known since the 1950s [15]. The task is performed by the agent, called a rollout, n times. For each rollout the current parameters are perturbed. In other words, for each rollout, a small random column vector, $\Delta\boldsymbol{\theta}$, is added to the current parameters. The total received reward in the rollout is considered an estimator of the expected return $\hat{J}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})$.

To estimate the gradient, the system can be treated as a linear regression problem. This results in:

$$[\mathbf{g}_{\text{FD}}^T \hat{J}(\boldsymbol{\theta})]^T = (\Delta\Theta^T \Delta\Theta)\Delta\Theta^T \hat{J} \tag{2.3}$$

where $\mathbf{g}_{\text{FD}}$ is the estimated gradient by the finite difference algorithm, $\hat{J}(\boldsymbol{\theta})$ is the estimation of the expected total reward using current parameters, $\hat{J}$ is the vector of the expected total reward per rollout, and

$$\Delta\Theta = \begin{bmatrix} \Delta\boldsymbol{\theta}_1 \ ... \ \Delta\boldsymbol{\theta}_n \\ 1 \ ... \ 1 \end{bmatrix}^T \tag{2.4}$$

where n is the total number of rollouts. The row of ones is appended to the matrix to both find the gradient $\mathbf{g}_{\text{FD}}$ as well as the estimated expected reward for the current parameters J_{ref}. The gradient $\mathbf{g}_{\text{FD}}$ is an estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ and can be inserted into the base algorithm in Section 2.2.2. The algorithm is shown in Algorithm 2.

Note that the random vectors $\Delta\boldsymbol{\theta}$ need to be sufficiently small to ensure that all rollouts are performed in the same neighborhood of the parameter space as the current parameters $\boldsymbol{\theta}$. The reason for this is that the gradient

Algorithm 2 Finite Difference

1: $\Delta\boldsymbol{\theta}_1 ... \Delta\boldsymbol{\theta}_n \leftarrow$ generate random vectors
2: **for** i=1 to n **do**
3: perform rollout and obtain $\hat{J}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i)$
4: **end for**
5: $\Delta\Theta \leftarrow \begin{bmatrix} \Delta\boldsymbol{\theta}_1 \; ... \; \Delta\boldsymbol{\theta}_n \\ 1 \; ... \; 1 \end{bmatrix}^T$
6: $[\mathbf{g}_{\mathrm{FD}}^T \hat{J}(\boldsymbol{\theta})]^T \leftarrow (\Delta\Theta^T \Delta\Theta)\Delta\Theta^T \hat{J}$
7: **return** $\mathbf{g}_{\mathrm{FD}}$

$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ needs to be constant by approximation in this region. If the vectors $\Delta\boldsymbol{\theta}$ are too large, the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})$ is different for each rollout and it will be unlikely that a linear solution approximates the actual gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

This section described a relatively simple algorithm for estimating the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. However, two aspects of the information are not used. The first is the known mapping of the parameters to the actions and its gradient $\nabla_{\boldsymbol{\theta}} \pi(\mathbf{x}, \mathbf{u})$, and the second is the specific time a reward was given in the rollout. In section 2.2.3, a gradient estimator is described that uses this information to make learning more efficient.

2.2.3 Vanilla

In this section the Vanilla [16] gradient estimator will be described. First it is explained how the known gradient of the policy with respect to policy parameters $\nabla_{\boldsymbol{\theta}} \pi(\mathbf{x}, \mathbf{u})$ can be used in order to estimate the gradient of the expected return with respect to the parameters $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

To find $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, the expected return $J(\boldsymbol{\theta})$ can be written as

$$J(\boldsymbol{\theta}) = \int_T p(\boldsymbol{\tau}|\boldsymbol{\theta}) R(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \tag{2.5}$$

where τ is a single trajectory out of the set of all possible trajectories T and R is a function that maps trajectories to a return. Using the chain rule and the derivative of the logarithm,

$$\frac{\mathrm{d}\log(\mathbf{x})}{\mathrm{d}\mathbf{x}} = \frac{1}{\mathbf{x}} \tag{2.6}$$

the gradient that needs to be known can now be written as follows.

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int_T p(\boldsymbol{\tau}|\boldsymbol{\theta}) R(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && (2.7) \\ &= \int_T p(\boldsymbol{\tau}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}|\boldsymbol{\theta}) R(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && (2.8) \\ &= E\{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}|\boldsymbol{\theta}) R(\boldsymbol{\tau})\} && (2.9) \end{aligned}$$

The probability of following a trajectory, $p(\boldsymbol{\tau}|\boldsymbol{\theta})$, is not known. This is because $p(\boldsymbol{\tau}|\boldsymbol{\theta})$ contains two components. One component consists of the probabilities of taking given actions in given states. These probabilities define the policy which is known. The other component consists of the state transition probabilities which are not known. By expanding and rewriting, these components can be split.

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}|\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \log \left(p(\mathbf{x}_0) \prod_{k=0}^{H} p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k) \right) && (2.10) \\ &= \nabla_{\boldsymbol{\theta}} \left(\log p(\mathbf{x}_0) + \sum_{k=0}^{H} \log p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) + \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k) \right) \\ &= \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k) && (2.11) \end{aligned}$$

The resulting expression does no longer contain state transition probabilities and is known. Filling in this result in Equation 2.9 yields the following.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E\{\nabla_{\boldsymbol{\theta}} \sum_{k=0}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k) R(\boldsymbol{\tau})\} \qquad (2.12)$$

To minimize the variance of the estimator, an arbitrary constant baseline can be subtracted. Subtracting a constant bias would result in subtracting the following component from $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

$$\begin{aligned} E\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) b\} &= b \int_T p(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p_(\boldsymbol{\tau}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\tau} && (2.13) \\ &= b \nabla_{\boldsymbol{\theta}} 1 && (2.14) \\ &= 0 && (2.15) \end{aligned}$$

Therefore, subtracting a constant baseline, b, does not introduce a bias to the gradient estimator.

This results in the gradient estimator

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E\{\nabla_{\boldsymbol{\theta}} \sum_{k=0}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)(\sum_{l=0}^{H} a_l r_l - b)\} \tag{2.16}$$

where a_l is the weight of receiving a reward r_l at timestep l.

This estimator is called the REINFORCE algorithm and was first described by Williams in 1992 [30]. The Vanilla gradient is an extension on this estimator [15]. To minimize variance further, the insight is used that the received reward cannot be altered by future actions. In other words, although it is not sure which actions have caused the agent to receive a certain reward, it is sure that actions performed after receiving the reward cannot have caused it. This means that, when estimating the policy gradient, actions can only be dependant on future rewards.

This results in the gradient estimator by Baxter and Barlett [3],

$$\mathbf{g}_{\text{GMDP}} = E\{\nabla_{\boldsymbol{\theta}} \sum_{l=0}^{H} \sum_{k=0}^{l} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)(a_l r_l - b)\} \tag{2.17}$$

The optimal baseline b for each element of the gradient estimator minimizes the variance of the estimator. Solving for this constraint results in

$$\mathbf{b}_k^h = \frac{\langle (\sum_{\kappa=0}^{k} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k))^2 a_k r_k \rangle}{\langle (\sum_{\kappa=0}^{k} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k))^2 \rangle} \tag{2.18}$$

A similar estimator, policy gradient, was published by Sutton et al. [26]. For more information about the relation between these two similar gradient estimators, or for details regarding above derivations, see Peters and Schaal [15]. The algorithm is summarized in Algorithm 3. Note that the original algorithm calculates $\mathbf{g}_{\text{GMDP}}$ and b within the loop while here it is done afterwards because a fixed number of rollouts is used to estimate $\mathbf{g}_{\text{GMDP}}$.

Both Finite Difference and the Vanilla gradient described above are dependant on the used units of the parameters. As the unit of each parameter is arbitrary, it is not optimal for the gradient to be dependant of the unit. This problem is as well as a solution for this problem is further described in section 2.2.4.

Algorithm 3 Vanilla gradient

1: **for** i=1 to n **do**
2: perform rollout and obtain r, $\mathbf{x}$ and $\mathbf{u}$ foreach step
3: **end for**
4: $\mathbf{g}_{\text{GMDP}} = E\{\nabla_{\boldsymbol{\theta}} \sum_{l=0}^{H} \sum_{k=0}^{l} \log \pi_{\boldsymbol{\theta}}(u_k|x_k)(a_l r_l - b)\}$
5: where $b_k^h = \frac{\langle(\sum_{\kappa=0}^{k} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k))^2 a_k r_k\rangle}{\langle(\sum_{\kappa=0}^{k} \log \pi_{\boldsymbol{\theta}}(\mathbf{x}_k,\mathbf{u}_k))^2\rangle}$
6: **return** $\mathbf{g}_{\text{GMDP}}$

2.2.4 Episodic Natural Actor Critic

Policy gradient using the Vanilla gradient estimator can be slow due to plateaus of the expected return in the parameter space. These are areas of the parameter space where the gradient of the expected return is close to zero and where often the direction of the gradient is hard to measure. A distinction between two kinds of plateaus must be made. A plateau can be formed because change in parameters does not significantly change the behavior, the distribution over trajectories. Therefore there is no change in expected return. In the other kind of plateau, changes in parameters do change the behavior. The changed behavior however, does not cause a change in the expected return. The problem of the first kind of plateau could be solved by taking steps in the parameter space of variable size, with the restriction that each step causes the same change in distribution over trajectories. This way, each learning step causes an equal change in behavior.

Another difficulty using the Vanilla gradient estimator lies in the parameterization. Even linear transformations on the parameters can change the gradient in such a way that it changes the path of the search through parameter space when the gradient is followed. This means that the policy that will be found is dependant on such a transformation. Following the natural gradient instead of the conventional gradient solves this issue.

The parameter space in general is not orthonormal. This can easily be seen because the units in which components of the policy parameters are measured can each be chosen arbitrarily and do not necessarily have any relationship to each other. This can cause the problem that a chosen learning rate, or stepsize, is too small for effective learning on one dimension and simultaneously too large for another. Similarly, taking a step in one dimension can result in a smaller change in trajectory as taking a step of the same size in another dimension of the parameter space. It would be desirable to do a

gradient descent where each step results in an equal change of distribution of trajectories $p(\boldsymbol{\tau})$. To be able to do this, first a metric must be found that measures the divergence between two distributions. This divergence can be measured using different metrics. According to Peters and Schaal [15], many of these metrics, such as the Kullback-Leibler divergence, share the same second order Taylor expansion,

$$d_{\mathrm{KL}}(p(\boldsymbol{\tau}_{\boldsymbol{\theta}})||p(\boldsymbol{\tau}_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}})) \approx \frac{1}{2}\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta} \tag{2.19}$$

where $\boldsymbol{F}_{\boldsymbol{\theta}}$ is the Fischer information matrix,

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \langle \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})^T \rangle \tag{2.20}$$

which can be estimated by

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \langle \sum_{l=0}^{H} a_l \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{x}_l, \mathbf{u}_l) \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{x}_l, \mathbf{u}_l)^T \rangle \tag{2.21}$$

See Peters and Schaal [15] for the derivation.

The optimal step $\Delta\boldsymbol{\theta}$ under the constraint that the step results in a change of trajectory distribution of a fixed size ϵ can be found by solving

$$\max_{\Delta\boldsymbol{\theta}}(J\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx \max_{\Delta\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}^T \nabla J(\boldsymbol{\theta}) \tag{2.22}$$

under the constraint

$$d_{\mathrm{KL}}(p(\boldsymbol{\tau}_{\boldsymbol{\theta}})||p(\boldsymbol{\tau}_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}})) \approx \frac{1}{2}\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta} = \epsilon \tag{2.23}$$

Solving this system can be done using Lagrange multipliers. For a detailed description of this method in general, see [2]. For a description of application of the method in the current context, see Appendix A. The result shows that gradient descent using this metric can be done by changing parameters

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha_n F_{\boldsymbol{\theta}_n}^{-1} \nabla J(\boldsymbol{\theta}_n) \tag{2.24}$$

where the learning rate is

$$\alpha = \lambda^{-1} \tag{2.25}$$

and

$$\lambda = \sqrt{\frac{\Delta\boldsymbol{\theta}^T F_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta}}{2\epsilon}} \tag{2.26}$$

Algorithm 4 Episodic Natural Actor Critic gradient

1: **for** i=1 to n **do**
2: perform rollout and obtain r, $\mathbf{x}$ and $\mathbf{u}$ for each step
3: **end for**
4: Policy derivatives $\boldsymbol{\psi}_k = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)$
5: Fisher matrix $\boldsymbol{F}_{\boldsymbol{\theta}} = \langle \sum_{k=0}^{H} (\sum_{l=0}^{k} \boldsymbol{\psi}_{\boldsymbol{l}}) \boldsymbol{\psi}_k^T \rangle$
6: Vanilla gradient $\mathbf{g} = \langle \sum_{k=0}^{H} (\sum_{l=0}^{k} \boldsymbol{\psi}_l) a_k r_k \rangle$
7: Eligibility matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_K]$
where $\boldsymbol{\phi}_h = \langle \sum_{k=0}^{h} \boldsymbol{\psi}_k \rangle$
8: Average reward vector $\bar{\mathbf{r}} = [\bar{\mathbf{r}}_1, ..., \bar{\mathbf{r}}_K]$
where $\bar{\mathbf{r}}_h = \langle a_h r_h \rangle$
9: Baseline $\mathbf{b} = \boldsymbol{Q}(\bar{\mathbf{r}} - \boldsymbol{\Phi}^T \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \mathbf{g})$
where $\boldsymbol{Q} = \boldsymbol{M}^{-1}(\mathbf{I}_K + \boldsymbol{\Phi}^T (\boldsymbol{M} \boldsymbol{F}_{\boldsymbol{\theta}} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1} \boldsymbol{\Phi})$
10: Natural gradient $\mathbf{g}_{\text{NG}} = \boldsymbol{F}_{\boldsymbol{\theta}}^{-1}(\mathbf{g} - \boldsymbol{\Phi}\mathbf{b})$
11: **return** $\mathbf{g}_{\text{NG}}$

A different learning rate can be used because doing so only changes the stepsize and does not change the direction of the search.

The algorithm in matrix form can be derived by solving the standard regression form, in which the sum of squared residuals is minimized,

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} (Y - X\boldsymbol{\beta})^T (Y - X\boldsymbol{\beta}) \tag{2.27}$$

where $\boldsymbol{\beta}^{*T} = \begin{bmatrix} \mathbf{w}^T & \mathbf{b} \end{bmatrix}$ and

$$X^T = \begin{bmatrix} \boldsymbol{\phi}_1^1 & \boldsymbol{\phi}_2^1 & \dots & \boldsymbol{\phi}_n^1 & \boldsymbol{\phi}_1^2 & \dots & \boldsymbol{\phi}_n^m \\ \mathbf{e}^1 & \mathbf{e}^2 & \dots & \mathbf{e}^n & \mathbf{e}^1 & \dots & \mathbf{e}^n \end{bmatrix} \tag{2.28}$$

where $\boldsymbol{\phi}_n^j = \sum_{t=1}^{n} \nabla_\theta \log \pi(\mathbf{u}_t^j|\mathbf{x}_t^j)$ for the j-th rollout and $\mathbf{e}_i$ is the i-th unit vector basis function with length n. The rewards are in

$$Y = \begin{bmatrix} r_1^1 & r_2^1 & \dots & r_n^1 & r_1^2 & \dots r_n^m \end{bmatrix} \tag{2.29}$$

The complete algorithm is summarized in Algorithm 4. Eligibility matrix $\boldsymbol{\Phi}$ contains the average gradients of the logarithm of probabilities of taking the same actions as have been done in the rollouts. Column k of the eligibility matrix is the gradient of taking the same actions untill timestep k, disregarding all actions taken after k. Note that lines 4 to 10 are originally

executed within the loop. Because a fixed number of rollouts is used to estimate $\mathbf{g}_{\mathrm{NG}}$, this calculation is could be taken out of the loop in this thesis. A derivation of the algorithm is described in Appendix B. More information about this algorithm can be found in [15] and [17].

2.3 Genetic Algorithm

In this section Genetic Algorithms are described. These algorithms are inspired by the process of natural selection as is seen in nature. There are many variants of Genetic Algorithms. This section discusses the implementation used in the research of this thesis. In this kind of algorithms, a population of individuals is maintained. In the case of learning policy parameters in a reinforcement learning, each individual is a set of policy parameters $\boldsymbol{\theta}$. The algorithm contains two steps that are performed repeatedly.

In the first step, the fitness of each individual is determined by a fitness function. In this case, $\exp \hat{J}(\boldsymbol{\theta})$ was used. Selecting individuals is done by roulette wheel selection. This means that the probability for each individual to be represented in the next generation, either as a parent or as itself, is proportional to this fitness. Therefore, the fitness cannot be negative. This is the reason that the exp operator was used in the fitness function.

In the second step, a new population is created through several different mechanisms. First, individuals can be transferred directly from the previous generation to the new, current population. This mechanism is often referred to as eliteness. Second, individuals can be transferred from the previous population after being mutated by a mutation function. This mutation function permutates a single element of $\boldsymbol{\theta}$ by adding a random, normally distributed, value. A third mechanism is the crossover function that takes two individuals from the previous generation and combines them to form a new individual. It involves taking a random set of elements from parameters $\boldsymbol{\theta}_i$ from one individual i, and the rest from parameters $\boldsymbol{\theta}_j$ of the other individual j. The new generation of individuals is filled with using each mechanism for each available position in the population.

For each of the mechanisms to produce an individual for the new population, one or two individuals from the previous population need to be selected. Each individual's probability of being selected is proportional to its fitness.

The probability that a single individual i is selected is therefore

$$p_i = \frac{\exp \hat{J}(\boldsymbol{\theta}_i)}{\sum_{j=1}^{n} \exp \hat{J}(\boldsymbol{\theta}_i)} \tag{2.30}$$

where n is the number of individuals in the population from which an individual is selected.

In order for the algorithm to search effectively, the population needs to be diverse. Because elements of fit individuals are being overrepresented in the next population with respect to individuals that are less fit, individuals of a population can easily come to occupy the same region of the parameter space. To prevent this from happening, similar individuals can be made to share their fitness. The shared fitness f_s for an individual i is

$$f_s(i) = \frac{\exp \hat{J}(\boldsymbol{\theta}_i)}{\sum_{j=1}^{n} \mathrm{sh}(i,j)} \tag{2.31}$$

where sh is a function that is 1 for individuals that are similar to each other and 0 for individuals that are not. In the case of parameters $\boldsymbol{\theta}$ the Euclidian distance can be used as a measure of similarity which would result in

$$\mathrm{sh}(i,j) = \begin{cases} 1, & \text{if } |\boldsymbol{\theta}_i - \boldsymbol{\theta}_j| < t \\ 0, & \text{otherwise} \end{cases} \tag{2.32}$$

where t is some predefined threshold. See Stanley and Miikkulainen [25] for a more information about shared fitness.

Genetic Algorithms can be applied in various fields. In this section, it was described how it is applied as a reinforcement learning algorithm. In this case the algorithm is a policy search algorithm in the sense that a group of policies is selected, and new policies are derived from the more valuable ones, while others are thrown away. For a more detailed overview of evolutionary algorithms see Beasley, Bull and Martin [4].

This chapter has described theoretical background required for the research of this thesis. In the next chapter, which builds on this background, the problem of learning a soccer kick is formalized and the approaches to solve it are described.

Chapter 3

Solution Approaches and Problem Setting

This chapter defines the problem setting of learning a kicking motion and discusses design choices for finding a solution. Section 3.2 discusses the policy, the mapping from policy parameters to actions. Section 3.3 describes the policy parameters that were used to start the learning process with. Section 3.4 describes the reward functions that were used and Section 3.5 discusses how these reward functions are combined to shape the complex behaviour of a soccer kick.

3.1 Problem setting

The aim of this study is to find a policy for a robot to kick a ball. The exact conditions are described, after which a description of measurements of success are given.

Before each trial, the ball is placed on a flat surface. The robot is placed in front of the ball. Because the humanoid robot needs to kick with one leg and stand on the other, the robot is placed in such a way that the ball is in front of the left foot. This means that the robot is placed 5 cm to the right of the center. In Figure 3.1 the situation is illustrated.

There are two aspects of the success of a policy. The aim is to find a powerful kick. Therefore the first aspect is the mean distance the ball is kicked over a number of trials. Falling is dangerous to both robot and the environment. It also prevents the robot from performing other tasks after

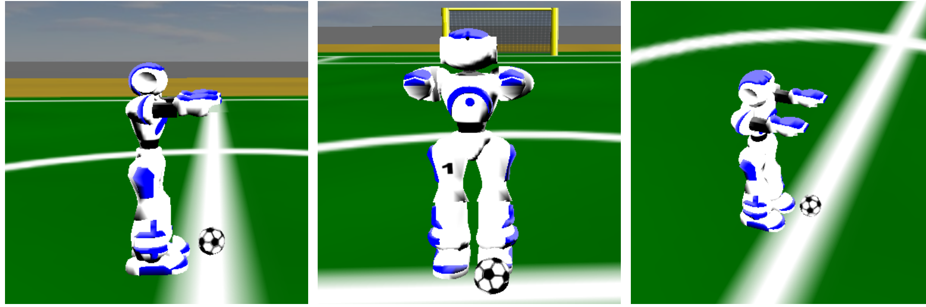


Figure 3.1: The initial position of the robot in the SimSpark simulator. The ball is placed in front of the left foot of the NAO.

a kick. For these reasons, the second aspect of the success is the portion of kicks in which the robot manages to keep its balance and ends the kick in a standing position over a number of trials.

3.2 Policy

The policy is the mapping from policy parameters and state to actions taken by the agent. The different aspects of this process are described here. In Section 3.2.1 the outline of this mapping is discussed. In Section 3.2.2 the way exploration was implemented is described. In section 3.2.3, cubic interpolation is described that is part of the process of mapping to actions. In Section 3.2.4, estimating the gradient of the policy with respect to policy parameters is discussed which is necessary information for using Vanilla and Episodic Natural Actor Critic algorithms.

3.2.1 Parameterization

Analogous to the policy used in Hausknecht and Stone [10], the parameterization was composed of a number of poses. At each pose, the desired angle of each joint of the robot was defined. The model of the robot that was used, the NAO, has 20 degrees of freedom in the Simspark simulator. See section 4.1 for considerations about using the simulator. The degrees of freedom of the real NAO that are missing in the used model represent the joints in the neck and hands, which are not relevant when learning to kick a ball. The model of the NAO has a different hip joint than the real NAO.

To keep learning on the model as close as possible to learning on the real NAO, the hip joints of the model were fixed. This results in a total of 18 joints. Therefore, each pose consists of 18 parameters, indicating the position of each joint. Contrary to Hausknecht and Stone's research, the time the robot needs to take to move from one pose to the next was equal for all poses. This way no time variables need to be optimized. Therefore all parameters resemble an angle so the unit each parameter is measured in is the same. Episodic Natural Actor Critic is designed to be able to effectively learn parameters whose units differ from parameter to parameter. For the other algorithms, however, parameters measured in different units can cause problems. Because 4 poses were used, the parameters θ consist of a total of $18 \times 4 = 72$ parameters. A fixed stable standing pose was added before and after the movement so every possible motion would both start and end in the same position. The exact pose that was used for this is discussed in Section 3.3. By choosing the parametrization this way, many different motions can be modeled. Also, when given some arbitrary existing motion, like a kicking motion from a RoboCup team, it is relatively easy to transform it to key poses and ultimately to policy parameters.

To calculate the trajectory, cubic interpolation over the poses was done. A single element of the pose, that resembles the angle of a single joint, is called a via point. For each joint independently, a curve through all via points was calculated. This means that the set of via points for a single joint i contained one point from each pose. A smooth function, called a spline, S_i through these via points was then calculated. Calculating the spline is discussed in section 3.2.3. The derivative of this spline $S_i(t)$ with respect to the time t is the angular velocity with which the joint i should move in order to stay on the trajectory. The element u_i^* of the action $\mathbf{u}^*$ for joint i is therefore

$$u_i^* = \frac{\mathrm{d}S_i(t)}{\mathrm{d}t} \tag{3.1}$$

Because of friction in the movements of the robot, the joint position might drift from the trajectory. To reduce this drift, instead of using all the via points for a joint, only the via points in the future were used for the interpolation. The robots sensors were used to measure the current position of each joint. This position was then used as a starting point of the spline. This is illustrated in Figure 3.2.

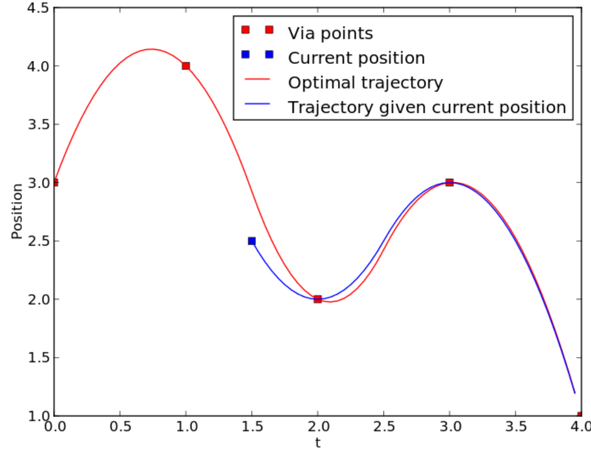


Figure 3.2: The optimal trajectory given the viapoints and the trajectory given a drifted current position.

3.2.2 Exploration

To find better policies, exploration is needed. This was done in two seperate ways for two different kinds of algorithms used in the research of this thesis. The Genetic Algorithm and the Finite Difference gradient estimator make small random changes in the policy parameters $\boldsymbol{\theta}$ to do exploration. The Vanilla and Episodic Natural Actor Critic algorithms on the other hand, work under the assumption that exploration is part of a stochastic policy. Exploration was implemented as follows.

The action that is taken at each time step was the sum of the optimal action $\mathbf{u}^*$ described above, and an exploration term $\boldsymbol{\epsilon}$. Exploration vector $\boldsymbol{\epsilon}$ was drawn from a normal distribution with zero mean. For each dimension, or joint, of the action, the same variance σ^2 was applied. This σ^2 is a predefined constant. This means that the covariation matrix was of the form

$$\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_n \tag{3.2}$$

where n is the dimensionality of the action $\mathbf{u}$. One constant was chosen for all dimensions of action $\mathbf{u}$ because all dimensions represent angular velocities of a single joint and are therefore measured in the same units. The many small variations can cancel eachother out resulting in very little variation in

the movement as a whole. For this reason, analogous to [15], the same error term was used for multiple timesteps before a new error term was drawn. The number of times that the same error term was used, was drawn from a uniform probability distribution over the range of 1 to 200 timesteps. Because there were 50 timesteps per second, the mean time that the same error term was used, was 2 seconds.

3.2.3 Cubic interpolation

A spline is a function that is formed by segments described by polynomial functions. For this purpose, cubic polynomials was used because this method is often used in trajectory calculations. Each via point is connected to two segments with the exception of the first and the last via point. This means that for n via points (including start and end points) $n-1$ segments are to be calculated. For each segment s_i of spline $S(t)$, a polynomial function needs to be calculated of the form

$$s_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \tag{3.3}$$

Where a, b, c, and d are the variables that need to be determined for each segment i to be able to fully describe the spline. For n via points, a total of $n-1$ segments times 4 variables per segment, results in $4(n-1)$ variables of which the value needs to be determined to know the spline. Conditions for these variables are as follows:

1. The spline needs to go through each via point (including start and end points)
2. At each via point (excluding the start and end points) the first and second derivative of the spline with respect to the time needs to be continuous.
3. At the end point, the first and second derivative need to be zero.

Condition 1 results in two constraints for each segment as its start and end are fixed. Condition 2 yields two constraints for each segment that is not the end segment. Condition 3 yields two constraints for the end segment. This results in a total of $4(n-1)$ constraints. With as many constraints as there are variables to be solved, a single exact solution can be calculated. This results in a description of the full spline.

For both the Vanilla and Episodic Natural Actor Critic algorithms, the gradient of the policy with respect to its parameters $\nabla_{\boldsymbol{\theta}}\pi(\mathbf{x}, \mathbf{u})$ needs to be known, where $\mathbf{x}$ is the state and $\mathbf{u}$ is the action. Using the chain rule, the derivative with respect to each element i of $\boldsymbol{\theta}$ is

$$\frac{\partial p(\mathbf{u}|\mathbf{x})}{\partial \theta_i} = \frac{\partial \mathrm{pdf}(\mathbf{u}|\mathbf{u}^*, \boldsymbol{\Sigma})}{\partial \theta_i} = \frac{\partial \mathrm{pdf}(\mathbf{u}|\mathbf{u}^*, \boldsymbol{\Sigma})}{\partial \mathbf{u}^*} \frac{\partial \mathbf{u}^*}{\partial \theta_i} \tag{3.4}$$

where $\mathrm{pdf}(\mathbf{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the probability density function of the normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ at $\mathbf{u}$.

To be able to calculate $\frac{\partial \mathbf{u}^*}{\partial \theta_i}$, an implementation of cubic spline interpolation was needed that could calculate the derivative with respect to via points. Because no standard package was found with this feature, an implementation had to be created.

To obtain spline $S_(t)$, equations following the constraints discussed above had to be solved. For this purpose, the system was expressed in an augmented matrix. Because derivatives of the resulting spline with respect to the via points were needed, matrix entries could either be constants or expressions containing variables. Gauss elimination was used to solve the system. This method is known to be computationally efficient [8].

In Figure 3.3 the performance of the implementation is shown. As can be seen in the figure, the time it takes to solve a matrix with variables increases exponentially when the size n, of the $n \times n$ matrix, increases. Solving a 6×6 matrix takes around 2 minutes to solve. The figure also shows that solving a matrix without variables, only containing constants, is much quicker. In practice, each row contains at least one variable, as a row without a variable does not represent a constraint on any variables and is therefore useless. For this reason the figure also shows time to solve a matrix with one variable on each row a lower bound on the time that would be needed when using this implementation for calculating a spline. In the current policy, a spline through 5 via points needs to be calculated. This means that the spline is formed out of 4 segments. To calculate each 4 variables of each segment, a matrix of 16×16 needs to be solved. Based on the data in Figure 3.3 time to solve a matrix of that size would take around 10^{11} seconds (over 3000 years). For this reason, this implementation to calculate splines could not be used.

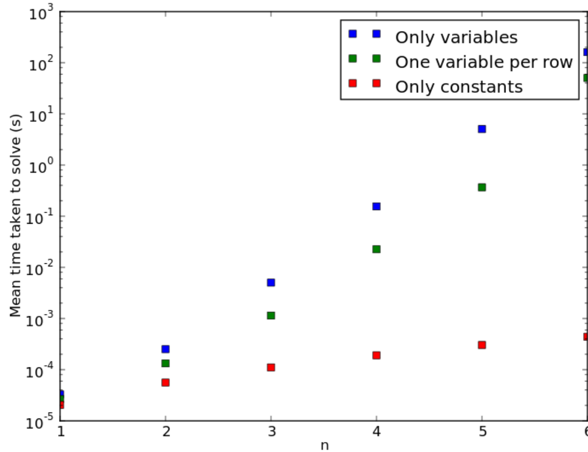


Figure 3.3: Time taken to solve an $n \times n$ matrix with respect to n. Solving matrices containing variables instead of constants takes considerably more time.

3.2.4 Estimating the gradient of the policy

Because standard software packages had to be used to calculate splines, calculating the exact derivatives of the policy with respect to policy parameters was not possible. Vanilla and Episodic Natural Actor Critic, however, depend on this information. An alternative had to be found to be able to use these algorithms. An alternative for calculating the exact derivatives of the policy is to estimate it,

$$\frac{\partial p(\mathbf{u}|\mathbf{x}, \theta_i)}{\partial \theta_i} \approx \frac{p(\mathbf{u}|\mathbf{x}, \theta_i + \Delta\theta_i) - p(\mathbf{u}|\mathbf{x}, \theta_i)}{\Delta\theta_i} \tag{3.5}$$

where $\Delta\theta_i$ is a small constant. Although estimating the gradient solves the performance issue discussed above, it raises the problem of having to select an appropriate $\Delta\theta_i$. Choosing too small a $\Delta\theta_i$ results in underflow as $p(\mathbf{u}|\mathbf{x}, \theta_i + \Delta\theta_i) - p(\mathbf{u}|\mathbf{x}, \theta_i)$ can be too small to store accurately in the computer's memory. To decrease the risk of losing valuable information due to underflow, instead of estimating the derivative of $p(\mathbf{u}|\mathbf{x}, \theta_i)$, the derivative of the logarithm of $p(\mathbf{u}|\mathbf{x}, \theta_i)$ can be estimated directly:

$$\frac{\partial \log p(\mathbf{u}|\mathbf{x},\theta_i)}{\partial \theta_i} \approx \frac{\log p(\mathbf{u}|\mathbf{x},\theta_i+\Delta\theta_i) - \log p(\mathbf{u}|\mathbf{x},\theta_i)}{\Delta\theta_i} \tag{3.6}$$

This way, $p(\mathbf{u}|\mathbf{x},\theta_i+\Delta\theta_i) - p(\mathbf{u}|\mathbf{x},\theta_i)$ never has to be evaluated.

Underflow problems when calculating probability densities of the normal distribution were not expected. However, the logarithm of these probabilities was also calculated directly to be sure such problems would not occur.

$$\log \mathrm{pdf}(x|\mu,\sigma^2) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp -\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.7}$$

$$= -\frac{(x-\mu)^2}{2\sigma^2} - \log\sqrt{2\pi\sigma^2} \tag{3.8}$$

Choosing too large a $\Delta\theta_i$ can also result in an inaccurate estimate when the derivative at $\theta_i + \Delta\theta_i$ is no longer approximately equal to the derivative at θ_i. It is dependant on the specific problem what too large means in this context. A $\Delta\theta_i$ of 0.01 was chosen as the elements of $\boldsymbol{\theta}$ represent joint angles measured in degrees. It was assumed that the derivative does not change significantly at this small scale.

While exact derivatives of the policy with respect to policy parameters could not be calculated, the Vanilla and Episodic Natural Actor Critic algorithms could still be by using this estimation of the gradient.

3.3 Starting parameters

Before learning can start, some set of policy parameters has to be taken as an initial policy. Two different initial policies were used in the various experiments in Section 4 of this thesis. One policy was designed to be an already working kick that could be optimized. Designing the movement manually would have been time consuming. Instead, the kicking motion of the RoboCanes team [1] of 2010 was used. This kick was chosen out of the kicks for which the code was available, because the way movement was defined was the most compatible with the policy used in the research of this thesis. RoboCanes had defined the joint angles of all joints of the NAO 50 times per second. Mere subsampling resulted in four poses that could be used as policy parameters for the used policy in the research of this thesis. Note that by subsampling, most of the information of the actual kicking motion had been lost asthis is similar to using a low-pass filter. The result was a

motion that was leaning to right and lifted its left foot but did not touch the ball. This is probably because the moment the ball is kicked, the lower leg is moved forwards abruptly. Information about abrupt movements is lost the most when subsampling. The policy was therefore tweaked by hand untill the robot softly kicked the ball.

The second initial policy was designed to keep the robot standing still. One option would be to use the default position of the NAO in the simulator where all joint poses are set to 0. This however represents a completely upright pose which is not very stable and not a realistic pose for a robot soccer player. The starting pose of the RoboCanes kick is a more stable pose. Therefore, as the standing still policy, this pose was used at all timesteps.

3.4 Reward functions

Different reward functions were needed for different tasks. This section describes each reward function used in the experiments in Section 4.

To learn the agent to kick the ball as far as possible, a reward function could be used that rewards the agent the kicked distance in meters. Such a reward function would require each rollout to end after the ball had stopped moving. This often takes several seconds in which the agent has finished its movement and is waiting for the rollout to end without taking any more actions. Because of the great number of rollouts needed for the learning process, having to wait several seconds each rollout is infeasible. The distance the ball is going to travel can be predicted by measuring the velocity of the ball after it has been kicked. For this reason, $f_{\text{max_ball_velocity}}$ rewards the agent the maximum velocity, in meters per second, the ball has had in the forward direction during the rollout. This way, the rollout can be stopped before the ball has stopped rolling. The timestep this reward is given, is the time the maximum velocity was measured. Note that the maximum velocity can only be determined after the rollout. This means that the reward can only be given after the rollout has ended so the agent can only be given the height and time of receiving the reward after the rollout is ended. Because the learning algorithms used are learning offline, meaning that no changes are made to the policy during the rollout, this does not form a problem.

For test purposes, 10 policies were trained using $f_{\text{max_ball_velocity}}$. As expected, all resulting policies resulted in a fall, after the ball was kicked, everytime the kick was executed. The agent had to be taught to keep the

robot in balance. For this purpose, the accelerometer of the NAO was used to estimate the body position. This was done because the absolute angular position was not easily accessible in the simulator. Also using the same sensors that are available on the real NAO simulates learning on the real NAO more closely. Similarly to the $f_{\text{max_ball_velocity}}$ reward function, waiting long enough to ensure that the robot had fallen or was keeping its balance would take a lot of time. To be able to keep rollouts short, a prediction for falling was needed. A large deviation of the vertical position of the torso at the end of the rollout implies that the robot is falling over. This can be used to make a prediction of whether the robot will fall. By penalizing the agent based on this prediction, there is no need to continue the rollout after this prediction is done. The $f_{\text{torso_angle}}$ is based on this idea. This reward penalizes the agent at the final timestep of the rollout. Because greater angles however would increase the chances of falling over, a continuous reward signal was used, as opposed to a discrete one. For this reason the reward function returns a negative reward the size of the difference between the vertical component of the acceleration measured a_z, and the vertical acceration which would be measured had the robot been standing still in a vertical position. Vertical acceleration a_z is the vertical component, relative to the torso, of the measured acceration of the torso. When standing still in a vertical position, the acceleration measured would be the gravitation acceleration g. The reward function yields some negative reward for every position other than vertical results, while deviating from the vertical position by only a small angle results in only small negative rewards. This results in

$$f_{\text{body_position}}(a_z, t) = \begin{cases} \mathrm{g} - a_z, & \text{if } t = t_{\text{end}} \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

where t is the time. Note that this reward function takes a_z and t as arguments instead of the state $\mathbf{x}$. Although this leads to confusion in the formulation, in practice this is not a problem as both a_z and t are both components of the state $\mathbf{x}$.

To teach a policy to kick a ball while not falling over, the weighted sum of $f_{\text{torso_angle}}$ and $f_{\text{max_ball_velocity}}$ might be used as a combined reward function. Doing this would result in the following problem. Consider learning the complete kicking motion from a standing still policy. In the first stage of the learning process, little or no reward would be received from the $f_{\text{max_ball_velocity}}$ while any attempt to explore policy variations would result in increased like-

liness to fall and to receive a negative reward from the $f_{\text{torso_angle}}$ component. If the latter component is too large, the agent would be discouraged from exploring which is needed to ever kick a ball. If the penalty component is too small, exploration would be possible and a kick would be learned. In this case a problem arises in the later stages of the learning process. The agent would have learned how to receive large rewards from the $f_{\text{max_ball_velocity}}$ component. This would result in the $f_{\text{torso_angle}}$ component being relatively small and it could be ignored by the agent. For this reason, to be able to learn effectively in both the start as well as later stages of the learning process, the penalty for falling over should be proportional to the reward for the kicking distance. The resulting reward function is

$$f_{\text{proportional}}(x) = f_{\text{max_ball_velocity}}(x)(1 + \mathrm{w} f_{\text{torso_angle}}(x)) \tag{3.10}$$

where w is a weighting contstant.

To learn a policy to stand on one leg, the $f_{\text{foot_pressure}}$ was used. This function uses the information from the pressure sensors of both feet of the robot. For the right foot, a positive reward is given proportional to the measured pressure because standing on one leg means that all the robot's weight is must be on this foot. For the left foot, a negative reward is given proportional to the measured pressure. The robot starts each rollout with its weight evenly distributed over both feet. To start transferring weight to the right foot, an effective strategy would be to push its weight away by applying extra pressure to the left foot. To make sure that such a strategy would not be discouraged, $f_{\text{foot_pressure}}$ only assigns above rewards between 1 and 1.5 seconds in the rollout. Outside this interval, zero reward is given. The resulting reward function is

$$f_{\text{foot_pressure}}(p_l, p_r, t) = \begin{cases} p_r - p_l, & \text{if } 1.0 < t < 1.5 \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

where p_l is the pressure on the left foot, p_r the pressure on the right foot and t the time past since the start of the rollout.

This section described the reward functions that were used in the research of this thesis. Section 3.5 describes how reward functions can be combined in order to learn more effectively.

3.5 Shaping a kick

Some learning tasks can be learned directly using a single reward function. More complex tasks can be learned more effectively when shaping is used, and sometimes cannot be learned without it [18, 9]. It was hypothesized that learning to kick the ball starting with a policy of just standing still could benefit from shaping. This is because while gradually changing a policy of standing still to something that looks like a soccer kick, the learner would have to pass policies that do not seem promising. In other words, the Policy Gradient algorithms try to find the local maximum and can not pass a valley in the fitness landscape. Using shaping, the agent is trained before it is given the task of ultimate interest. This way, the agent has gained experience before it starts learning that task. Erez and Smart [7] have enumerated methods for gaining this experience. Methods that are applicable to the task of the research of this thesis are discussed below.

The dynamics of the task can be modified. The robot could for instance be suspended so it cannot fall. It would be easier for the robot to learn to kick the ball this way. It would however learn a kick that does not take the possibility of falling into account. It was hypothesized that a soccer kick that does take falling into account would differ significantly from a kick from a suspended position. The experience gained would therefore not be useful.

The algorithm parameters could be modified. A sensible modification would be to decrease exploration rates or learning step sizes when a relatively good policy has been learned and the current policy needs to be fine tuned. To apply this method, rules need to be invented that dictate the exploration rate or learning step size at each point in the learning process. Because such rules are not trivial to define, it was chosen to use a different shaping method.

The action space could be modified. Some joints of the robot could be fixed. For this to work, it would be necessary to decide which joints would not contribute much and could therefore stay fixed initially.

Finally, the reward function could be modified. Rewards can be given for behavior that seems promising, even though the ball might not have been kicked at all. Because this is the most intuitive method in the context of the research of this thesis, this method was applied.

Two learning stages were used. In the first stage, the learner is rewarded the sum of the $f_{\text{foot_pressure}}$ and the $f_{\text{torso_angle}}$. In this stage a policy is being learned that results in the robot standing on one foot. The resulting policy serves as the starting policy for the second stage. In the second stage, the

agent is rewarded for the rolling distance of the ball and for keeping balance according to $f_{\text{proportional}}$. It is tested if using these two stages to learn a policy results in a more efficient kick.

In this section, the approach of the research of this thesis has been described. In Section 4 experiments are described to test the validity of the approach.

Chapter 4

Experiments

Experiments were done to compare the performance of algorithms discussed in Section 2 and to test the discussed approach. The experiments were done using a simulator. Section 4.1 discusses considerations for choosing the simulator, Section 4.2 discusses experiments that compare the learning using the discussed algorithms, and Section 4.3 discusses experiments that test the discussed approach.

4.1 Simulator

Because experiments involve many iterations of the robot interacting with the world, it is infeasible to work with a real robot. It was therefore chosen to use a physics simulator. There are various simulators that can be used for simulating a robot in a physical environment. The most relevant simulators are listed here.

The simulator that is used often in research is the commercial simulator WeBots [11]. Robots can be modeled from scratch using components like servos that can be programmed to move toward some desired position with a given maximum torque. For WeBots a set of complete models of robots including the NAO are available. Physics are handled by the ODE physics engine. The simulator also includes important features like a programmable supervisor and an option to simulate scenarios with accelerated time. These two features are both valuable when it comes to machine learning experiments. This simulator however is costly and was for this reason not available for the research of this thesis.

USARSim (Urban Search and Rescue Simulator) was designed for simulating search and rescue scenarios. It was based on an Unreal Tournament game engine. Both graphics and physics are used from this engine. Because of these graphics, the simulator is suitable for experiments with computer vision. During the research of this thesis, a model was created for the NAO robot [28]. It was however not yet available for the research of this thesis.

A simulator specifically designed to simulate the NAO robot is NAOSimPro by Cogmation Robotics. It will work closely with the Aldebaran Choregraph tool that comes with the NAO robot. This simulator however, was not yet available as it was still under development.

Another simulator specifically designed for the NAO, but can simulate other models as well, is SimRobot [21]. This simulator is part of software written by a RoboCup team called the GermanTeam and further developed by the BHuman team. The software contains the simulator, as well as the complete decision making software used by the team during RoboCup matches, and debugging tools for this software. It's implementation is completely done in C++.

The simulator that is used for official simulated RoboCup matches is Simspark. Besides being used for simulated soccer matches, Simspark has also been used for research [27]. Earlier versions of this simulator simulated a simple 2-Dimensional enviroment to be able to let users focus on soccer teamplay. Since 2006 it was upgraded to simulate 3-Dimensional environments. In 2008 a model, which is currently used in matches, of the NAO robot was included in the simulator. Note that this model differs one degree of freedom with the real NAO robot. This extra degree of freedom makes the hip in the simulated NAO more flexible than that of the real NAO [31]. The simulator is split into components that can be run seperately and on different computers to be able to distribute the load of the complete simulation. The server simulates the physics using the ODE physics engine and provides perceptions on request to agents that can set effectors in joints for example that simulate servos. Agents can be implemented in any language as long as they are able to communicate with the server via sockets or as a plugin. This makes it possible to develop agents more efficiently using higher level programming languages like Python. While repeated experiments can be done using this simulator, it cannot do them in accelerated time.

The most suitable simulators for the research of this thesis are SimRobot and SimSpark as they both include a NAO model and were available. Because of its status as simulator for official RoboCup matches, and because of the

accesible way agents can be developed for it, the research of this thesis was done using SimSpark.

The total time running the simulator for experiments discussed in Sections 4.2 and 4.3 was over 50 days. Because of this, the use of a supercomputer was considered to run both simulator and agent. The used simulator however, uses sockets to connect agent and simulator programs which could form a problem configuring properly when multiple instances of the programs were run simultaneously. For this reason, configuration on a supercomputer was believed to take more time than would be gained by using it. Therefore, experiments were run on two PCs.

4.2 Comparative experiments

In this section learning using the four algorithms in section 2 are compared. Learning the complete set of policy parameters described in Section 3.2 experiments is time consuming. Therefore only a subset of parameters was learned using all four algorithms. The results are described in section 4.2.1. Using the two best performing learning algorithms, another experiment was done in which the full set of parameters is learned. This is described in Section 4.2.2.

4.2.1 Learning a subset of parameters using all algorithms

To explore which learning algorithm can most efficiently learn a soccer kick in the current setting a subset of the policy parameters was optimized. In this experiment as a initial policy, the Robocanes based soccer kick described in Section 3.3 was used. The eight parameters that were needed to be optimized resembled the position of the left knee, and forward-backward degree of freedom of the hip, at four poses. These parameters were chosen as the knee and hip joints play a key role in kick the ball and therefore improvements can be made by optimizing only this small subset of eight parameters. The other parameters were kept fixed at the initial policy. The reward function used was $f_{\text{proportional}}$ which is described in Section 3.4. The same number of rollouts per Policy Gradient iteration, 20 rollouts, was used as the size of the population in the Genetic Algorithm. The same number of Policy Gradient iterations was used as the number of generations. This way the same num-

ber of rollouts were done when using the Policy Gradient methods as when using the Genetic Algorithm. Each run was repeated 10 times. The resulting means, with error bars representing twice the standard error, are shown in Figure 4.1.

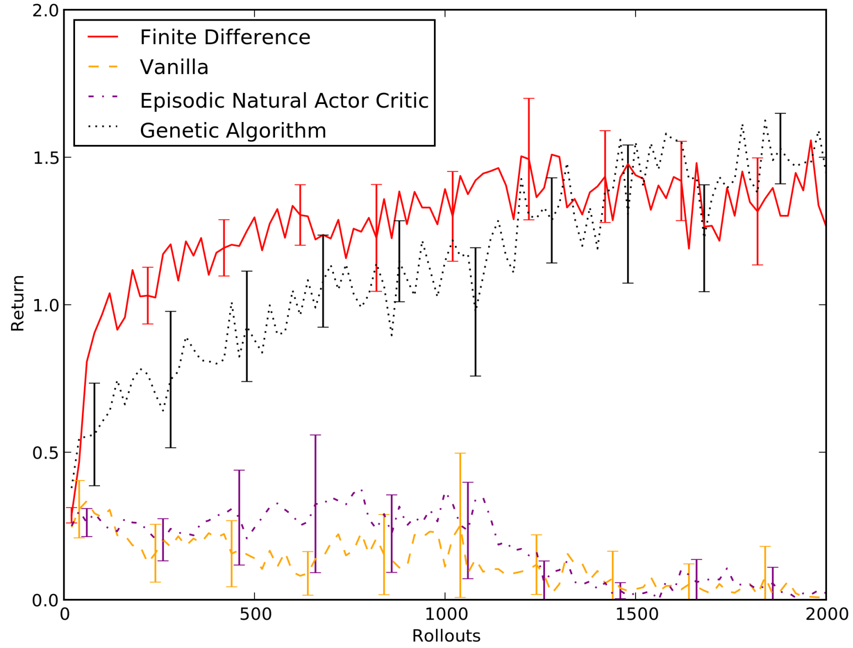


Figure 4.1: Mean return versus rollouts. Finite Difference and Genetic Algorithm manage to increase return while Vanilla and ENAC perform below expectations.

The results show that both Finite Difference and the Genetic Algorithm have increased return significantly. Using Episodic Natural Actor Critic and Vanilla gradients, no improvements were made at all. This result was not expected considering earlier research [15]. One hypothesis is that current implementations of these algorithms are not correct. Another hypothesis is that the gradient of the policy with respect to the policy parameters, $\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{u})$, was not estimated accurately. As discussed in Section 3.2.4 this gradient could not be calculated as was done in research in the literature

[15] and had to be estimated. A problem with this estimation could explain the results above because only Vanilla and ENAC algorithms use $\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{u})$ to estimate the gradient of the return $\nabla_\theta J(\theta)$ while the Finite Difference and Genetic Algorithm do not use $\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{u})$. To test these hypotheses two experiments were done.

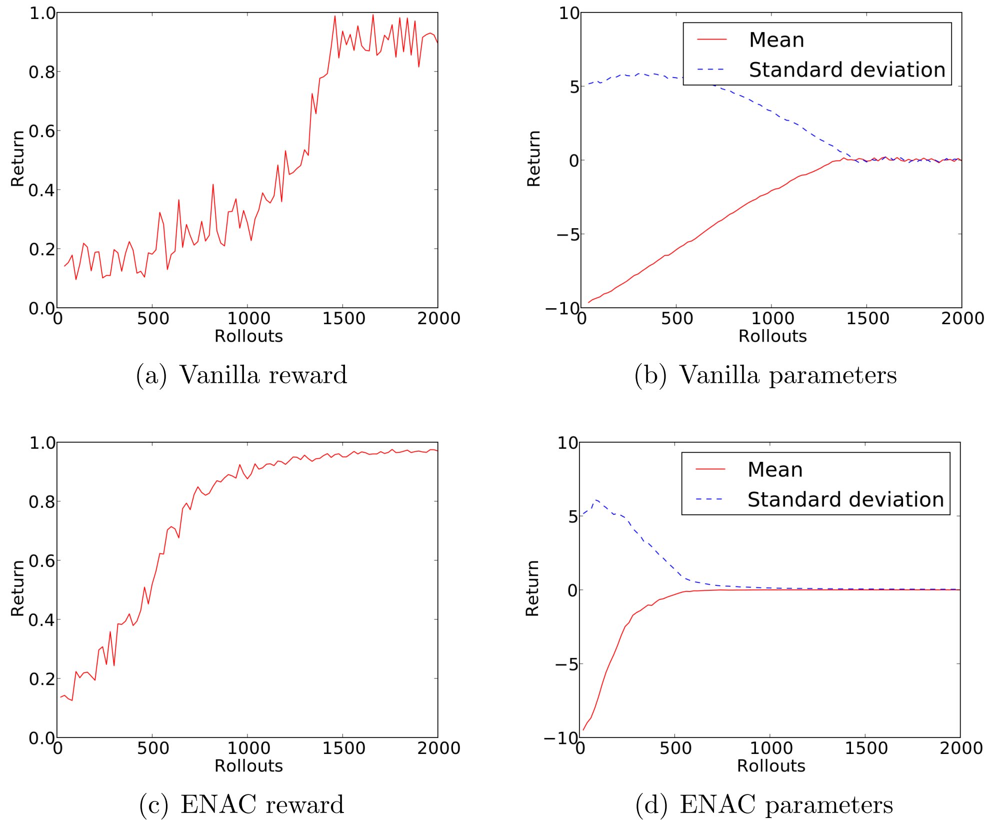
To investigate the first hypothesis that current implementations of Vanilla and ENAC are not correct, these algorithms were tried to learn a simpler task. This task is unrelated to learning a soccer kick or robotics in general. The task is as follows. Each rollout has only one timestep. The action is returning a single real number. The reward function is

$$f_{\text{simple}}(u) = \frac{1}{1 + \text{abs}(u - c)} \tag{4.1}$$

where u is the scalar action and c a constant value, in this case 0. The highest possible reward 1 is returned when $u = c = 0$. The policy for this task is defined as follows. Each action is drawn from a normal distribution with its mean and standard deviation being the only policy parameters. Parameters are optimal when both mean and standard deviation are zero, which results in consistantly drawing $u = 0$ which results in the highest possible reward. Result of two single runs are shown in Figure 4.2.

Results show, as expected, a difference in way the optimal parameters, zero mean and zero standard variation, are found. It can be seen that the size of the natural gradient estimated with the Episodic Natural Actor Critic algorithm decreases when parameters are near optimal. These results show that implementations work as expected when applied to a very simple learning task.

To investigate the second hypothesis that the gradient of the policy with respect to the policy parameters, $\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{u})$, was not estimated accurately, algorithms were applied to another simple task. This task however, involved the model of the NAO in the SimSpark simulater used in earlier experiments. The same policy was used as described in Section 3.2. However, only one parameter was learned, which represented a single pose of backward-forward degree of freedom of the left hip joint. A simple reward function was used. At each timestep, the current position of the left hip joint, in degrees, was given as a reward. Increasing the parameter will increase the position of the joint and therefore increase reward. To decrease variance, the torso position of the robot in the simulator was fixed some distance in the air so the robot

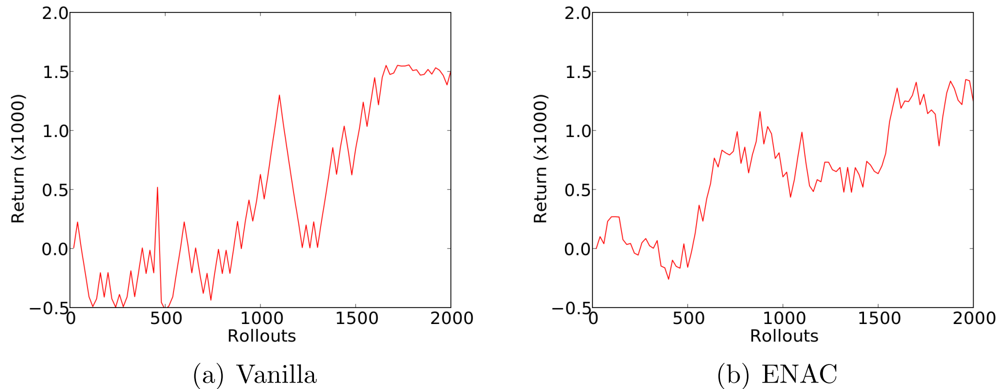


(a) Vanilla reward
(b) Vanilla parameters
(c) ENAC reward
(d) ENAC parameters

Figure 4.2: Return at simple task. Both Vanilla and ENAC algorithms perform as expected.

could not fall or touch the ground. Per policy gradient, 20 rollouts were done. Results are shown in Figure 4.3.

The results show a slight increase of return with respect to policy gradient iterations. The actual gradient of this single-dimensional space is positive everywhere and no significant noise in the return is present in this controlled setting. Decreasing the return can therefore only be a result of decreasing the parameter while an increase in return can only be the result of an increase of the parameter. Going back and forth in the parameter space indicates that the algorithm cannot reliably estimate the gradient. Analysis shows that 54 out of 100 estimated gradients had the correct sign when Vanilla was used. When using ENAC, 50 out of 100 estimated gradients, had the



(a) Vanilla

(b) ENAC

Figure 4.3: Return on the simple increase joint pose task. Both methods show some increased return although not as much as was expected.

correct sign. This is remarkable given that the task is simple and only single-dimensional policy parameters are to be optimized. Because performance was worse than expected, it might be the case that the estimation of $\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{u})$ using this method is accurate enough to result in some improvement during learning, but not accurate enough to result in effective learning when used with Episodic Natural Actor Critic and Vanilla gradient estimators in higher dimensional parameter spaces.

It can be concluded that both finite difference policy gradient and the genetic algorithm perform best. Therefore a full scale run was done using these two algorithms.

4.2.2 Learning all parameters using only finite difference and genetic algorithm

Using the two best performing algorithms, the Robocanes based soccer kick was optimized using all policy parameters described in Section 3.2. The reward function $f_{\text{proportional}}$ was used. The number of rollouts per Policy Gradient iteration and size of population was both 200. The number of iterations and generations was both 200. Figure 4.4 shows the mean learning curves of 10 runs with bars indicating twice the standard error.

The results show similar mean learning curves when using both algo-

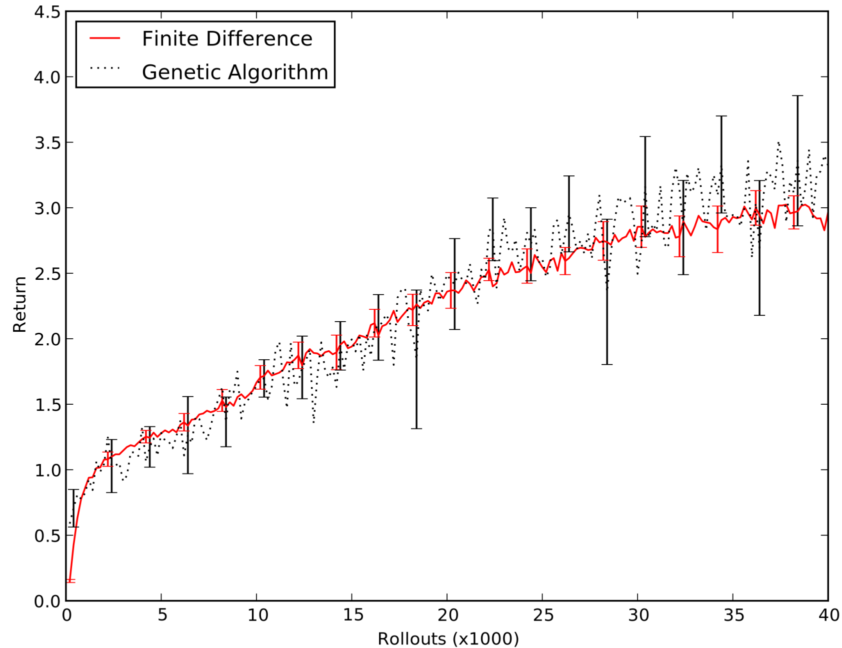


Figure 4.4: Mean return versus the number of rollouts during the optimization of the Robocanes based policy.

rithms. While the mean learning curve when using the Genetic Algorithm seems to end higher than when using Finite Difference, the difference between the two is smaller than the standard error of the Genetic Algorithm curve. Note the high variance in results when using the Genetic Algorithm. High variance in return within a single generation was also observed where the exploration of the Finite Difference method resulted in similar looking, predictable movements. For these reasons, the Finite Difference algorithm was used in method validaton experiments.

4.3 Approach validation experiments

To answer the question of what would be a good approach for learning a soccer kick, a kick was learned using three different approaches. The first

approach was by providing no working initial policy, in the sense that the ball is never kicked, and by using a single reward function, $f_{\text{proportional}}$. The initial policy is the standing still initial policy. The second approach was using the same, not working, policy and using shaping over two learning stages as described in Section 3.5. The third approach was by optimizing an already working policy using the reward function $f_{\text{proportional}}$. The setup of this approach is identical to the one in Section 4.2.2. Those results for this approach are therefore reused here. The number of rollouts per Policy Gradient iteration was 200 and the number of iterations was 200. Mean results of 10 runs for each method with error bars indicating twice the standard error are shown in Figure 4.5.

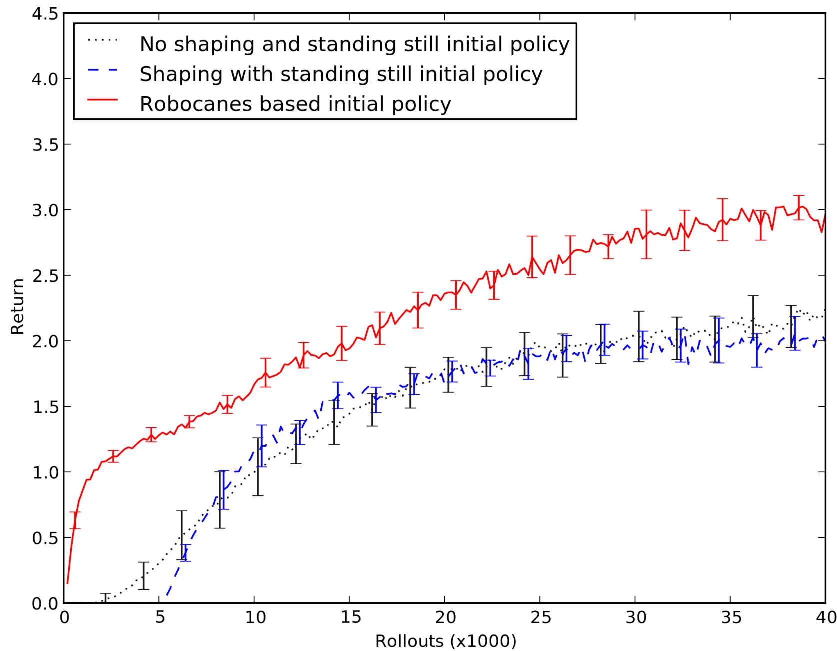


Figure 4.5: Mean return versus Policy Gradient iterations

Note that the first 25 iterations of the learning curve when using shaping has a zero return. This is because the first 25 iterations, a different reward function was used. The reward shown here is $f_{\text{proportional}}$. The height of the

mean return of $f_{\text{foot_pressure}}$ during the first 25 iterations is shown in a different graph, Figure 4.6, to avoid confusion. Also note that the mean returns of the first iterations are zero of the learning curve where no working initial policy or shaping is used. This is because the learning process has not yet

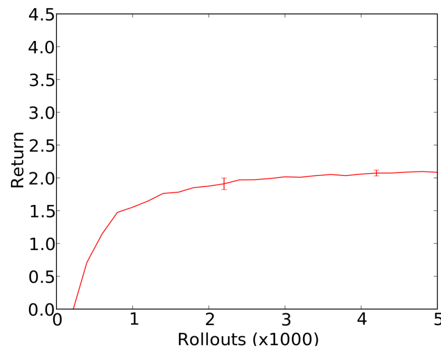


Figure 4.6: Mean return versus Policy Gradient iterations during first shaping stage. The return shown is that of the same reward function as was used for learning.

reached a policy that results in a movement that moves the ball. During these iterations, the robot received no non-zero reward signal at all because it did not manage to touch the ball. The search was therefore unguided and it is only by chance that a movement that touched the ball was developed.

Results show that the mean return at the last Policy Gradient iterations was a bit higher when shaping was not used. The highest mean return was received when a working kick was used as initial policy.

These three methods resulted each in 10 optimized policies. Each policy was executed 10 times to measure both the actual mean rolling distance of the ball, as well as how often the robot falls when performing the kick. Results of the methods are in the first three rows of Table 4.1.

The policy learned with the Robocanes based initial policy results in the kick with the greatest mean distance. However, the movement is the most unstable because the robot falls in 87% of the times it is performed.

Because the method that resulted in the greatest kicking distance was not stable, an attempt was made to correct this. It was hypothesized that the reason for being able to learn such an unstable kick was that falling did not generate enough negative reward for the agent. The used reward function

Method	Distance		Fall ratio
	Mean	Std	
No example or shaping	3.0	0.7	0.66
No example with shaping	2.8	0.9	0.27
Robocanes based initial policy	3.7	1.0	0.87
Stabilized Robocanes based initial policy	3.8	0.7	0.30

Table 4.1: Mean kicking distance by method

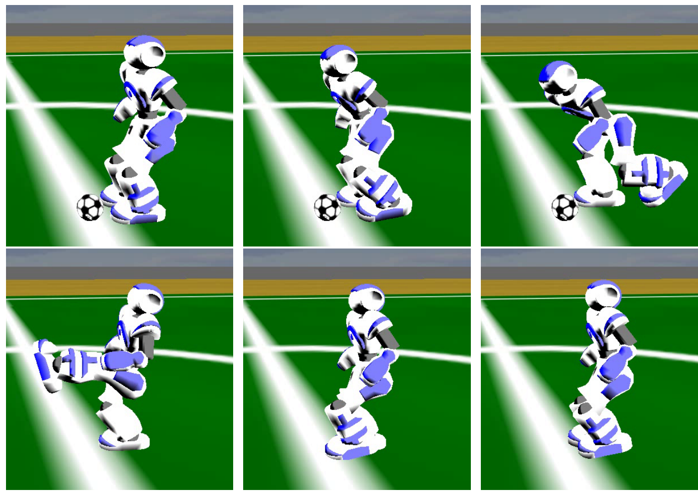


Figure 4.7: One of the final resulting kicks. This motion was learned by stabilizing a kick that was the result of optimizing the Robocanes based initial policy.

makes a prediction on whether the robot will fall over so each rollout can be stopped without waiting to see if the robot would keep its balance (see Section 3.4). It is possible that this prediction was not accurate enough. For this reason, 25 additional policy gradient iterations were done, with a longer rollout of 4 seconds. Reward function and rollouts per iteration were unchanged. The resulting policies had a mean kicking distance of 3.8 with a standard deviation of 0.7 and had a combined fall ratio of 0.30. Note that one out of the ten resulting policies fell every time, which meant a fall ratio of 1.0. Without this outlier, the combined fall ratio of the remaining nine kicks was 0.22.

Figure 4.7 illustrates the kick that was learned with the Robocanes based initial policy and stabilized as described above. The complete motion is performed in under two seconds. Most of this time is spent preparing for the actual kick and returning to a stable stance afterwards. All kicks that were trained using this same approach appear similar and are in fact hard to distinguish from each other.

Chapter 5

Discussion and Future Work

In Sections 5.1 and 5.2 research questions introduced in the Section 1.2 are answered. Future research is suggested in Section 5.3 after which this thesis is concluded in Section 5.4.

5.1 What algorithm is most effective for learning a soccer kick?

It was unexpected that learning with using Vanilla and Episodic Natural Actor Critic algorithms during the comparative experiments would result in such poor results. Extra experiments showed that learning a single-dimensional parameter in a simple task was possible using the implementations for this thesis. It was also shown that a one-dimensional parameter in the SimSpark simulator could be learned while using the estimated gradient of the policy with respect to policy parameters. This learning process however was slow, and learning eight parameters, to optimize the existing kicking motion was unsuccesful. The literature shows that for a similar policy, also using poses and splines, 66 parameters can be learned using Vanilla [10]. Also, the Episodic Natural Actor Critic algorithm had been used to learn to hit a ball with a baseball bat on a robot arm with seven degrees of freedom using motor primitives [16]. Both these experiments had calculated the gradient of the policy with respect to the parameters instead of estimating it like was done in this thesis. This suggests that poor performance was due to the estimation of the policy with respect to policy parameters, instead of the algorithms themselves. This gradient might not have been estimated

accurately enough to facilitate learning in the eight-dimensional parameter space, while it did suffice for learning a one-dimensional parameter.

The best learning performance was achieved using Finite Difference Policy Gradient and Genetic Algorithm methods. While both methods had similar performance in terms of learning speed and resulting policy, the Finite Difference method is more suitable for robotics because the parameter variations that are tried during the learning process are relatively similar and predictable. When using the Genetic Algorithm, although the end result was a good policy, the variations that were tried during the learning process were unpredictable and sometimes violent. This increases chances on damaging the robot when learning on a real robot is done.

A concern when using Policy Gradient methods is that the search can get stuck in a local optimum. Repeating the same experiment multiple times has given some insight about this problem in the current context. If the found policies would have been very diverse, this would have been some evidence that the searches had all got stuck in different local optima, and therefore a global optimum, at least in most cases, had not been found. However, kicks that were learned using the same, most successful, approach using Policy Gradient appeared were hard to distinguish from each other purely by looking at the motion. Although no evidence of getting stuck in a local optimum was found, it cannot be concluded that the searches had all led to a global optimum as all searches might have found the same local optimum.

When learning this number of parameters using Finite Difference or Genetic algorithms, many rollouts were needed to learn a good policy. When using a real robot to learn a soccer kick, it would be necessary to put the robot and ball in the starting position hundreds of thousands of times. It would therefore be infeasible to use either of these algorithms to learn a movement like kicking a ball with this many parameters on a real robot. Optimizing another task however, like walking fast for example, might be feasible. When walking, the ending position of one rollout can be used as a starting position for the next as long as the robot does not fall, and an already working policy is used. Note that in this case, the distributions over start states would change after each learning iteration. If variations are tried out in random order, variation in the start state can be treated like any other noise and is not likely to form a problem. Walking variations could even be tried out without having to stop the walking motion after each variation has been tried. This way, the robot could easily optimize its walk while doing

other tasks. Optimizing a policy while performing other tasks is only feasible when using a learning algorithm that changes policy parameters smoothly like Policy Gradient methods do.

5.2 What is a good approach for learning a soccer kick on a humanoid robot using reinforcement learning?

The result that the strongest kick was learned using an example as a starting policy was expected. Because the starting policy was already a working kick, all learning iterations could be used to optimize the performance of this kick. Learning with shaping did not result in a more powerful kick than learning without it. By using two different reward functions for two different stages in the learning process, the knowledge was used that, for kicking, the robot would have to stand on one leg. The idea is to facilitate learning by moving the search into a part of the parameter space in the first stage, from which it is easier to find a good policy in the second stage. This comes at the cost of spending learning iterations optimizing with respect to a reward function in the first stage that does not represent the final goal of the learning process. Choosing an unsuitable reward function for the first stage can lead to a detour in the parameter space. This seems to have happened in this thesis when it comes to learning a powerful kick.

Learning with shaping resulted in a more stable kick than learning without it. This was an expected result. Unexpected, however, was that learning using an example resulted in the least stable kick. One possible reason for this unexpected result is that falling was not observed directly during learning, but was predicted. This made it possible for unstable policies to be learned while the agent did not receive negative rewards for it. When learning with a working kick as a starting policy, all iterations were used to search in a good part of the parameter space. It is possible that, because of this, policies could be developed that exploited the reward function in this unintentional way.

Shaping has not improved learning on each criterion. The kick with the best properties was created by using an example as an initial policy. To make this kick stable, episode length had to be extended to make sure that the agent would receive a negative reward for falling. Changing the length of the episode, effectively changed the expected return for a policy. So this also

is a form of shaping. The same resulting kick might have been learned when the extended episode length was used during the complete learning process instead of just the final stage. This would, however, have resulted in twice the time needed to perform all rollouts.

5.3 Future research

Future research can focus on applying the techniques used in the research of this thesis on a real robot. A great number of rollouts is required to learn a kick without a good initial policy. By learning mainly in simulation and only optimize a transferred policy on the real robot, this difficulty could be dealt with.

Defining a policy by defining key poses like was done in this thesis and earlier by Hausknecht and Stone [10] has proven to be successful. In the research of Hausknecht and Stone, the actions of the robot were only dependent on the time in the rollout and not on the current position of the robot. In this thesis, the current position was taken into account to calculate the trajectory to ensure smooth movements. Stability might not have been a problem in the research of Hausknecht and Stone because movements were performed by a quadrupedal robot. In this thesis, where a humanoid robot was used, the best learned policy for a soccer kick still results in falling in one out of five tries. Stability can be improved by making the policy dependent on the position and accelerometers of the robot. This way the robot could learn to keep its balance. One way this could be done would be to introduce an extra parameter c_i for each joint i. After an action u_i^* has been calculated, like explained in Section 3.2, a correction can be added based on the accelerometer. This correction could be a linear combination of c_i and a measure of how much the robot is leaning in one direction l. This could be done to correct falling over to either side, x-direction, but also for the forward-backward or y-direction. The components of the final action $\mathbf{u}^f$ would then be

$$u_i^f = u_i^* + l_x c_i^x + l_y c_i^y \tag{5.1}$$

This correction could be used when learning to kick, but also when learning other tasks like walking.

In this thesis, criteria for the quality of a soccer kick were the stability of the kick and the distance the ball had traveled, or the power of the kick. Future research could use different criteria. Some important properties of a

soccer kick that were not taken into account in the research of this thesis are accuracy of the kick, and time it takes from deciding to kick to the actual kicking of the ball.

5.4 Conclusion

Finite Difference Policy Gradient and Genetic Algorithms are effective for finding good policies for humanoid robotics tasks. Although many iterations are needed to learn, no gradient of the policy needs to be available which makes these algorithms applicable to a wide range of situations. Shaping can also be a helpful method. Not only can it help to find good policies in specific situations, but can also be applied to reduce time needed to learn. Future research has been suggested to improve learning approaches. These methods will improve the ability to play soccer, but can also help make robots better personal assistants in future.

Appendix A

Finding the optimal step size using Lagrangian multipliers

Gradient descent with step $\Delta\boldsymbol{\theta}$ under the constraint that the step results in a change of trajectory distribution of a fixed size ϵ can be found by solving

$$\max_{\Delta\boldsymbol{\theta}}(J\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx \max_{\Delta\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}^T \nabla J(\boldsymbol{\theta}) \tag{A.1}$$

under the constraint

$$\frac{1}{2}\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta} = \epsilon \tag{A.2}$$

The Lagrangian becomes

$$L(\Delta\boldsymbol{\theta}, \lambda) = \Delta\boldsymbol{\theta} J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}^T \nabla J(\boldsymbol{\theta}) + \lambda(\epsilon - \frac{1}{2}\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta}) \tag{A.3}$$

$$\frac{\partial L(\Delta\boldsymbol{\theta}, \lambda)}{\Delta\boldsymbol{\theta}} = \nabla J(\boldsymbol{\theta}) + \frac{1}{2}\lambda\Delta\boldsymbol{\theta}^T(\boldsymbol{F}_{\boldsymbol{\theta}} + \boldsymbol{F}_{\boldsymbol{\theta}}^T) \tag{A.4}$$

Because $\boldsymbol{F}_{\boldsymbol{\theta}}$ is symmetrical

$$\frac{\partial L(\Delta\boldsymbol{\theta}, \lambda)}{\Delta\boldsymbol{\theta}} = \nabla J(\boldsymbol{\theta}) + \lambda\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \tag{A.5}$$

Setting the partial derivative to zero yields

$$\Delta\boldsymbol{\theta} = \lambda^{-1} \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \nabla J(\boldsymbol{\theta}) \tag{A.6}$$

Filling this in into the partial derivative of the Lagrangian with respect to λ and setting it to zero results in

$$\frac{\partial L(\Delta\boldsymbol{\theta}, \lambda)}{\lambda} = \epsilon - \frac{1}{2}\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta} = 0 \tag{A.7}$$

where

$$\lambda = \sqrt{\frac{\Delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta}}{2\epsilon}} \tag{A.8}$$

Appendix B

Episodic Natural Actor Critic Derivation

Compatible basis functions are

$$f_{\mathbf{w}}(\boldsymbol{\xi}_{0:k}^{j}) = \left(\sum_{\tau=0}^{k} \nabla \log \pi(\mathbf{u}_{\tau}^{j}|\mathbf{x}_{\tau}^{j})\right)^{T} \mathbf{w} \tag{B.1}$$

This can be brought to standard regression form, in which the sum of squared residuals is minimized,

$$\boldsymbol{\beta}^{*} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})^{T}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) \tag{B.2}$$

where $\boldsymbol{X}^{T} = \begin{bmatrix}\boldsymbol{X}_1 & \boldsymbol{X}_2\end{bmatrix}$ and $\boldsymbol{\beta}^{*T} = \begin{bmatrix}\mathbf{w}^{T} & \mathbf{b}\end{bmatrix}$. Before plugging in the expressions specific to current context the solution of this standard problem will be described.

The solution of Equation B.2 is

$$\boldsymbol{\beta}^{*} = \begin{bmatrix}\mathbf{w} \\ \mathbf{b}\end{bmatrix} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{Y} \tag{B.3}$$

Let $\boldsymbol{T} = \boldsymbol{X}_1^{T}\boldsymbol{X}_1$, $\boldsymbol{U} = \boldsymbol{X}_1^{T}\boldsymbol{X}_2$ and $\boldsymbol{W} = \boldsymbol{X}_2^{T}\boldsymbol{X}_2$. The Matrix Inversion Theorem can then be applied which results in

$$(\boldsymbol{X}^{T}\boldsymbol{X})^{-1} = \begin{bmatrix}\boldsymbol{T} & \boldsymbol{U} \\ \boldsymbol{U}^{T} & \boldsymbol{W}\end{bmatrix}^{-1} = \begin{bmatrix}\boldsymbol{T}^{-1} + \boldsymbol{T}^{-1}\boldsymbol{U}\boldsymbol{Q}\boldsymbol{U}^{T}\boldsymbol{T}^{-1} & -\boldsymbol{T}^{-1}\boldsymbol{U}\boldsymbol{Q} \\ -\boldsymbol{Q}\boldsymbol{U}^{T}\boldsymbol{T}^{-1} & \boldsymbol{Q}\end{bmatrix} \tag{B.4}$$

where $\boldsymbol{Q}^{-1} = \boldsymbol{W} - \boldsymbol{U}^T\boldsymbol{T}^{-1}\boldsymbol{U}$. Applying Sherman-Morrison formula [14] yields

$$\boldsymbol{Q} = \boldsymbol{W}^{-1} + \boldsymbol{W}^{-1}\boldsymbol{U}^T(\boldsymbol{T} - \boldsymbol{U}\boldsymbol{W}^{-1}\boldsymbol{U}^T)^{-1}\boldsymbol{U}\boldsymbol{W}^{-1}) \tag{B.5}$$

Putting these results in Equation B.3 yields

$$\begin{aligned} \mathbf{w} &= \boldsymbol{T}^{-1} + \boldsymbol{T}^{-1}\boldsymbol{U}\boldsymbol{Q}\boldsymbol{U}^T\boldsymbol{T}^{-1}\boldsymbol{X}_1^T\boldsymbol{Y} - \boldsymbol{T}^{-1}\boldsymbol{U}\boldsymbol{Q}\boldsymbol{X}_2^T\boldsymbol{Y} && \text{(B.6)} \\ &= \boldsymbol{T}^{-1}(\boldsymbol{X}_1^T\boldsymbol{Y} - \boldsymbol{U}\mathbf{b}) && \text{(B.7)} \\ \mathbf{b} &= \boldsymbol{Q}(\boldsymbol{X}_2^T\boldsymbol{Y} - \boldsymbol{U}^T\boldsymbol{T}^{-1}\boldsymbol{X}_1^T\boldsymbol{Y}) && \text{(B.8)} \end{aligned}$$

Filling in the definitions of $\boldsymbol{T}$, $\boldsymbol{U}$ and $\boldsymbol{W}$ results in

$$\mathbf{w} = (\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}\boldsymbol{X}_1^T(\boldsymbol{Y} - \boldsymbol{X}_2\mathbf{b}) \tag{B.9}$$

$$\mathbf{b} = \boldsymbol{Q}\boldsymbol{X}_2^T(\boldsymbol{Y} - \boldsymbol{X}_1(\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}\boldsymbol{X}_1^T\boldsymbol{Y}) \tag{B.10}$$

where

$$\begin{aligned} \boldsymbol{Q} &= (\boldsymbol{X}_2^T\boldsymbol{X}_2)^{-1} \\ &+ (\boldsymbol{X}_2^T\boldsymbol{X}_2)^{-1}\boldsymbol{X}_2^T\boldsymbol{X}_1(\boldsymbol{X}_1^T\boldsymbol{X}_1 - \boldsymbol{X}_1^T\boldsymbol{X}_2(\boldsymbol{X}_2^T\boldsymbol{X}_2)^{-1}\boldsymbol{X}_2^T\boldsymbol{X}_1)^{-1}\boldsymbol{X}_1^T\boldsymbol{X}_2(\boldsymbol{X}_2^T\boldsymbol{X}_2)^{-1} \end{aligned} \tag{B.11}$$

Now the solution to Equation B.2 is known, $\boldsymbol{X}_1$, $\boldsymbol{X}_2$ and $\boldsymbol{Y}$ need to be defined for the current context.

$$\boldsymbol{X}^T = \begin{bmatrix} \boldsymbol{\phi}_{1:1}^1 & \boldsymbol{\phi}_{1:2}^1 & \dots & \boldsymbol{\phi}_{1:n}^1 & \boldsymbol{\phi}_{1:1}^2 & \dots & \boldsymbol{\phi}_{1:n}^m \\ \mathbf{e}^1 & \mathbf{e}^2 & \dots & \mathbf{e}^n & \mathbf{e}^1 & \dots & \mathbf{e}^n \end{bmatrix} \tag{B.12}$$

where $\boldsymbol{\phi}_{1:n}^j = \sum_{t=1}^n \nabla_\theta \log \pi(\mathbf{u}_t^j|\mathbf{x}_t^j)$ for the j-th rollout and $\mathbf{e}_i$ is the i-th unit vector basis function with length n.

$$\boldsymbol{Y} = \begin{bmatrix} r_1^1 & r_2^1 & \dots & r_n^1 & r_1^2 & \dots r_n^m \end{bmatrix} \tag{B.13}$$

We insert Fischer information matrix $\boldsymbol{X}_1^T\boldsymbol{X}_1 = \boldsymbol{F}_2$, Eligibility matrix $\boldsymbol{X}_1^T\boldsymbol{X}_2 = \overline{\boldsymbol{\Phi}}$ and the Identity matrix $\boldsymbol{X}_2^T\boldsymbol{X}_2 = m\mathbf{I}_n$ into Equations B.9, B.10 and B.11 which yields

$$\mathbf{w} = \boldsymbol{F}_2^{-1}\mathbf{g}_2 \tag{B.14}$$

$$\mathbf{b} = \boldsymbol{Q}(\bar{\mathbf{r}} - \overline{\boldsymbol{\Phi}}^T\boldsymbol{F}_2^{-1}\mathbf{g}) \tag{B.15}$$

$$\boldsymbol{Q} = (m\mathbf{I}_n)^{-1} + (m\mathbf{I}_n)^{-1}\overline{\boldsymbol{\Phi}}^T(\boldsymbol{F}_2 - \overline{\boldsymbol{\Phi}}(m\mathbf{I}_n)^{-1}\overline{\boldsymbol{\Phi}}^T)^{-1}\overline{\boldsymbol{\Phi}}(m\mathbf{I}_n)^{-1} \tag{B.16}$$

$$= m^{-1}(\mathbf{I}_K + \overline{\boldsymbol{\Phi}}^T(m\boldsymbol{F}_2 - \overline{\boldsymbol{\Phi}}\overline{\boldsymbol{\Phi}}^T)^{-1}\overline{\boldsymbol{\Phi}}) \tag{B.17}$$

where

$$\boldsymbol{F}_2 = \sum_{j=0}^{m}\sum_{i=0}^{n}\boldsymbol{\phi}_{1:i}^{j}(\boldsymbol{\phi}_{1:i}^{j})^T \tag{B.18}$$

$$\overline{\boldsymbol{\Phi}} = \sum_{j=0}^{m}\sum_{i=0}^{n}\boldsymbol{\phi}_{1:i}^{j}(\mathbf{e}_i)^T \tag{B.19}$$

$$\bar{\mathbf{r}} = \sum_{j=0}^{m}\sum_{i=0}^{n} r_i^j(\mathbf{e}_i)^T \tag{B.20}$$

$$\mathbf{g}_2 = \sum_{j=0}^{m}\sum_{i=0}^{n}(r_i^j - \mathbf{b}_i)\boldsymbol{\phi}_{1:i}^{j} \tag{B.21}$$

$$\mathbf{g} = \sum_{j=0}^{m}\sum_{i=0}^{n} r_i^j\boldsymbol{\phi}_{1:i}^{j} \tag{B.22}$$

Note that, for better readability, symbols used in Algorithm 4 differ from the symbols used here. The following relations apply. $\boldsymbol{\phi}_k = m^{-1}\boldsymbol{\phi}_{1:n}^{j}$, $\boldsymbol{F}_{\boldsymbol{\theta}} = m^{-2}\boldsymbol{F}_2$ and $\boldsymbol{\Phi} = m^{-1}\overline{\boldsymbol{\Phi}}$.

Bibliography

[1] S. Abeyruwan, V. Diaz, A. Locay, N. Sneij, A. Seekircher, J. Stoecker, and U. Visser. *RoboCanes - Team Description Paper*. 2010.

[2] R.A. Adams and C. Essex. *Calculus: Several Variables*. Pearson Education Canada, 2009.

[3] J. Baxter, P. L. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

[4] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993.

[5] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *CoRR*, abs/1105.5460, 2011.

[6] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Approximate reinforcement learning: An overview. In *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, pages 1–8, Paris, France, April 2011.

[7] T. Erez and W.D. Smart. What does shaping mean for computational reinforcement learning? In *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 215 –219, aug. 2008.

[8] J.B. Fraleigh, R.A. Beauregard, and V.J. Katz. *Linear Algebra*. World student series edition. Addison-Wesley, 1995.

[9] V. Gullapalli and A.G. Barto. Shaping as a method for accelerating reinforcement learning. In *Intelligent Control, 1992., Proceedings of the 1992 IEEE International Symposium on*, pages 554 –559, aug 1992.

[10] M Hausknecht and P. Stone. Learning powerful kicks on the aibo ers-7: The quest for a striker. In *Proceedings of the RoboCup International Symposium 2010*. Springer Verlag, 2010.

[11] T. Hester, M. Quinlan, and P. Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

[12] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

[13] J. Kulk and J. Welsh. A low power walk for the nao robot. preprint, available at, 2008.

[14] T. K. Moon and W. C. Stirling. *Mathematical Methods and Algorithms for Signal Processing*, volume 204. Prentice Hall, 2000.

[15] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients, 2008.

[16] J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. In *ECML*, pages 280–291, 2005.

[17] J. R. Peters. *Machine learning of motor skills for robotics*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 2007. AAI3262746.

[18] J. Randlv and P. Alstrm. Learning to drive a bicycle using reinforcement learning and shaping, 1998.

[19] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Auton. Robots*, 27:55–73, July 2009.

[20] Aldebaran Robotics. Naoware documentation, 2009. Technical Specifications Document.

[21] T. Röfer, T. Laue, J. Müller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. De Haas, A. Härtl, D. Honsel, P. Kastner, T. Kastner, B. Markowsky, M. Mester, J. Peter, O. J. L. Riemann, M. Ring, W. Sauerland, A. Schreck,

I. Sieverdingbeck, F. Wenk, and J. Worch. B-human team report and code release 2010. unpublished, available at http://www.b-human.de/file_download/33/bhuman10_coderelease.pdf, 2010.

[22] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

[23] M. Saggar, T. D'Silva, N. Kohl, and P. Stone. Autonomous learning of stable quadruped locomotion. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 98–109. Springer Verlag, Berlin, 2007.

[24] B.F. Skinner. *The behavior of organisms: an experimental analysis*. Century psychology series. D. Appleton-Century Company, incorporated, 1938.

[25] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[26] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Information Processing Systems*, 12:1057–1063, 2000.

[27] S. G. van Dijk. Practical Hierarchical Reinforcement Learning in Continuous Domains. Master's thesis, University of Groningen, 2008.

[28] S. Van Noort. Validation of the dynamics of a humanoid robot in usarsim. Master's thesis, University of Amsterdam, the Netherlands, 2012.

[29] C. J. C. H. Watkins and P. Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992.

[30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 10.1007/BF00992696.

[31] Y. Xu, H. Mellmann, and H. Burkhard. An approach to close the gap between simulation and real robots. In *Lecture Notes in Artificial Intelligence*, volume 6472, 2010.