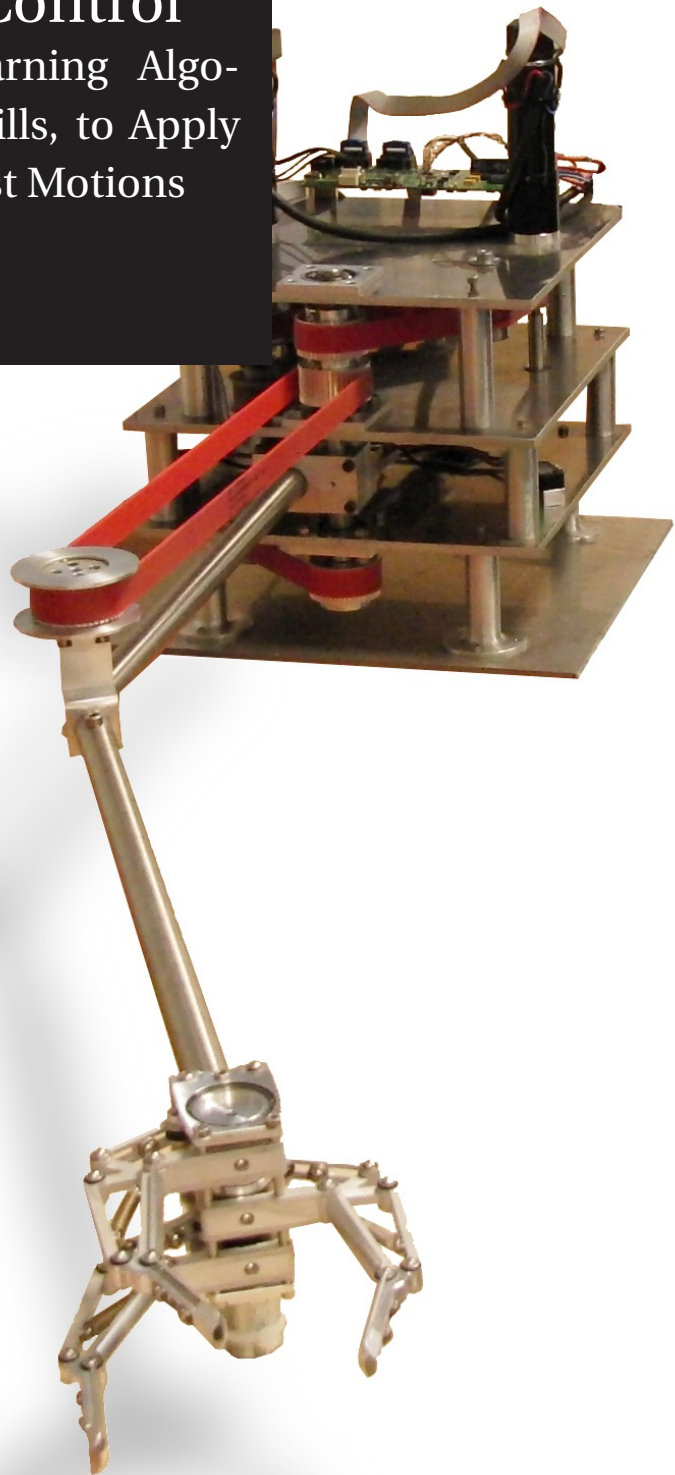


Learning Versatile Control

Evaluating Supervised Learning Algorithms in Parameterized Skills, to Apply the Algorithm on Rest to Rest Motions

A. D. Koelewijn

Technische Universiteit Delft



LEARNING VERSATILE CONTROL

EVALUATING SUPERVISED LEARNING ALGORITHMS IN PARAMETERIZED SKILLS, TO APPLY THE ALGORITHM ON REST TO REST MOTIONS

by

A. D. Koelewijn

in partial fulfillment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Delft University of Technology,
to be defended publicly on Wednesday July 16, 2014 at 14:00.

Supervisor:	Ir. W. Wolfslag	
Thesis committee:	Prof. dr. ir. P. P. Jonker,	3ME, TU Delft
	Dr. Ir. M. Wisse,	3ME, TU Delft
	Ir. W. Wolfslag,	3ME, TU Delft
	Prof. dr. R. Babuska	DCSC, TU Delft

This thesis is confidential and cannot be made public until July 16, 2016.

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

PREFACE

This report describes the findings of my master graduation project at Delft University of Technology. This project was carried out in the department of Biomechanical Engineering of the faculty of Mechanical, Maritime and Materials Engineering. The aim of this project was to analyse and improve a reinforcement learning algorithm that can be used to learn many tasks.

This goal provided a year long journey through literature and MATLAB. It was very interesting to learn about all the different algorithms that exist and the research groups that work on this topic around the world. It was even more exciting to be able to actually create and analyse an algorithm by myself. It was a great challenge, and I really enjoyed these first steps in the world of academia, even though it was quite stressfull from time to time.

I would like to thank my supervisors, Wouter Caarls and Wouter Wolfslag, for their help and support throughout this year. Also, Martijn Wisse should be mentioned for providing me the contacts that I needed to find an interesting graduation project. Finally, all master and PhD students present at the RAP meetings were a great source of inspiration.

This thesis ends six years of studying in Delft. It is incredible to look back and realize how much I have learned and grown in these years. I have had so much fun thanks to so many people, but especially Michelle and Irene, who helped me finish this last year, and my parents, because without them I would not have gotten this far yet.

*A. D. Koelewijn
Delft, July 2014*

ABSTRACT

Service robots require versatile control to be able to operate in environments that can change. Classic control methods require costly retuning of parameters if the environment changes. Therefore, it is necessary to use a different approach when designing a controller that can be used in a changing environment. An option is to automate the process of designing the controller. This can be done using reinforcement learning. In reinforcement learning, a robot interacts with the environment, to learn to perform a task such that a user-defined reward function is optimized.

However, this method is rather slow if many tasks should be learned, since this process should be repeated for each task. A speed up can be achieved by exploiting similarities between tasks. Parameterized Skills, introduced by Da Silva *et al.* [1] is a state-of-the-art reinforcement learning algorithm that does this. It aims to learn a function that describes the control policy, or sequence of actions that should be executed from start to goal, as a function of the task, using the assumption that similar tasks yield a similar control policy. The control policy is parameterized, so Parameterized Skills trains a controller to find these parameters as a function of the task.

This controller is found in two steps. First, training tasks are defined. The optimal control policy is learned for these tasks using reinforcement learning. Then, one obtains a data set consisting of the training tasks and their optimal parameters. In the second step, a function is fitted through this data using supervised learning. This function should return optimal parameters for all tasks.

My aim is to apply this algorithm on a two degree of freedom robot to learn to move from any start to any goal position. To do so, three issues should be solved. First, the performance of the controller should be improved, since the performance found by Da Silva *et al.* was not as good as the performance found using reinforcement learning only. Second, the time required to construct the model should be decreased. The supervised learning step was performed using support vector regression (SVR) by Da Silva *et al.*. This algorithm uses an optimization procedure to find a vector with the length equal to the number of training tasks that is used, which makes it rather slow. Finally, it should be checked if one can implement the algorithm to learn a task that varies in more than two dimensions, since it has only been used to learn a task with only a varying goal position.

Four supervised learning algorithms will be implemented in Parameterized Skills and it is evaluated which of these yield the best performance in terms of accuracy and energy efficiency and in terms of required time. These supervised learning algorithms are SVR, polynomial regression, a neural network algorithm and locally weighted linear regression.

It is expected that polynomial regression, the neural network algorithm and locally weighted linear regression are faster than SVR. Next to this, polynomial regression and the neural network algorithm can yield better performance due to the flexibility of these algorithms. Locally weighted linear regression can also yield better performance since it matches well with the assumption that similar tasks yield similar parameters. This algorithm uses only a local region to find the output for the current target.

These algorithms are used to learn a task with two dimensions. Their performance is analyzed and the supervised learning algorithm that yields the best performance will be applied to a task with four dimensions. The size of the data set should be increased if a more complex task is learned. It is evaluated how well the data set size scales with an increased dimension.

Parameterized Skills will be applied to a two degree of freedom robot arm. My aim is to train this arm to move to any goal position from a fixed start position and to learn to move from any start position to any goal position. This arm has two links which are connected using joints, allowing for a rotation around one axis for each arm. These two states are controlled using current control.

A dynamic movement primitive (DMP) is used to define the control policy applied to the arm. A DMP consists of an attractor system and a nonlinear system. The attractor system attracts the robot arm to the desired goal. The nonlinear system adds a forcing term to this movement such that any kind of motion towards the goal can be achieved. The forcing term has parameters that define the strength of the forcing term. These

parameters are dependent on the task that is currently executed.

Parameterized Skills aims to find a function that describes these parameters as a function of the task. First, parameters are learned for training tasks using PoWER, a state-of-the-art reinforcement learning algorithm. Three changes were made to the standard implementation of PoWER to yield good learning.

First, the exploration around the mean policy is defined as a function of the parameter, instead of as a function of time. This yields that all sampled trajectories can be reproduced using a single policy without exploration.

Second, the algorithm is altered to take into account the difference between the parameters of the trajectory and the parameters of the current mean policy. This ensures that trajectories sampled under previous policies are taken into account correctly.

Third, the importance weight is defined as the ratio between the current reward and the reward of the trajectory. Originally, the probability of occurrence under the current policy was used. However, this yielded bad results if the first update step was not in the correct direction.

The training data, consisting of training tasks and their optimal parameters, is fed to the supervised learning algorithms. Then, controllers are found using each algorithm. These controllers are tested using random testing tasks. The performance in terms of accuracy and energy efficiency yielding from the controllers is compared using statistical tests. Also, the time required to construct each controller is compared.

First, Parameterized Skills is used to train the robot on a task with a fixed start position and a varying goal position. It is found that locally weighted linear regression yields similar performance as SVR and is slightly faster than SVR. However, the speed up is negligible when looking at the total time. The neural network algorithm performs significantly worse than SVR and locally weighted linear regression. Polynomial regression is not suited to use in Parameterized Skills, since the shape of a polynomial does not fit the training data well.

Next, SVR and locally weighted linear regression are applied to learn a task with a varying goal and start position. The performance of the algorithms is expected to be similar to the performance on the task with a varying goal state if the size of the data set increases quadratically. It is found that the performance of locally weighted linear regression was indeed similar. The results found with SVR are also similar in value in terms of accuracy, but not in energy efficiency. Also, the variance is also higher in the problem with a varying start and goal position.

Hence, it is concluded that locally weighted linear regression scales better in terms of the required size of the data set used to construct the controller. Locally weighted linear regression is also faster than SVR, a speed up of 2% was achieved already on the task with a varying start and goal position, so with four dimensions.

The construction time of SVR increases exponentially with the size of the data set. Therefore, it is not feasible to use SVR to train a controller for a robot arm when it should perform a task that varies in more than four dimensions. This would yield an infeasible construction time for the SVR model. Locally weighted linear regression can be used in Parameterized Skills to find a controller for the robot arm, since it does not require construction time, while the performance is similar to SVR.

CONTENTS

Abstract	v
1 Introduction	1
1.1 Reinforcement Learning	2
1.2 Parameterized Skills	4
1.3 Research Question	5
1.4 Contents of this Thesis	6
1.4.1 Supervised Learning Algorithms	7
1.4.2 Method to Answer Research Question	9
1.5 Summary	12
2 Implementation of Parameterized Skills	15
2.1 Implementation of Parameterized Skills	15
2.1.1 Implementation of Reinforcement Learning	15
2.1.2 Finding the Number of Subspaces in the Policy Space	20
2.1.3 Implementation of Supervised Learning in Parameterized Skills	20
2.2 Implementation of the Control Policy	22
2.2.1 Attractor System	22
2.2.2 Nonlinear System	22
2.3 Summary	23
3 Verification of the Implementation of Parameterized Skills	25
3.1 Verification of Reinforcement Learning Algorithm	25
3.1.1 Simple Problem Used in Experiments	26
3.1.2 Experiment 1: Verification that the Algorithm Learns	26
3.1.3 Experiment 2: Verification that the Algorithm Optimizes for the Correct Variables	27
3.1.4 Experiment 3: Verification that the Result is Optimal	29
3.2 Verification of the Supervised Learning Algorithms	32
3.2.1 Experiment 4: Verification of Supervised Learning Methods	32
3.2.2 Results of Experiment 4	32
3.3 Summary	34
4 Comparison of Supervised Learning Algorithms on a Task with Two Dimensions	35
4.1 Threshold on Accuracy and Energy Efficiency	35
4.1.1 Set Up of Experiment 5	36
4.1.2 Results and Analysis of Experiment 5	36
4.2 Analysis of Performance of Supervised Learning Algorithms	36
4.2.1 Set Up of Experiment 6	37
4.2.2 General Results of Experiment 6	37
4.2.3 Selection of Data Set Sizes	45
4.2.4 Comparison of Performance of the Models	47
4.3 Comparison of Time Required of Supervised Learning Algorithms	48
4.3.1 Set Up of Experiment 7	48
4.3.2 Results and Analysis of Experiment 7	49
4.3.3 Comparison of Total Time Required by Supervised Learning Algorithms	50
4.4 Summary	50

5	Parameterized Skills Applied to a Task with Four Dimensions	53
5.1	Performance of Parameterized Skills on a Task with Four Varying Dimensions	53
5.1.1	Set Up of Experiment 8.	54
5.1.2	General Results of Experiment 8	54
5.1.3	Comparison of Results, Scaled for Increased Complexity.	57
5.2	Comparison of Required Time for Each Algorithm	59
5.2.1	Set Up of Experiment 9.	59
5.2.2	Results and Analysis of Experiment 9	59
5.2.3	Conclusion.	59
5.3	Summary	61
6	Conclusion	63
6.1	Limitations	64
6.2	Recommendations for Future Work.	64
	Bibliography	67
A	Boxplots of Accuracy and Difference Between Used Torque and Threshold of Experiment 6	69

1

INTRODUCTION

Service robots¹ are required to operate in many different and changing environments to enable its wide-spread use. This means that a controller should be designed that supports this versatile use. Currently, a controller that is designed for a specific environment can achieve a high level of versatility. However, a change in the environment in which the controller operates can be very expensive. Often, it is required to hire a control expert to retune parameters of the controller such that the robot is able to operate in this new environment.

An example is presented by Singh *et al.* [2], which was introduced by Barto and Crites [3]. They describe a system of three elevators for which a controller should be installed to ensure that all passengers are transported as fast as possible. This is a very time-consuming task for a programmer, which does not necessarily yield the best result. However, the programmer should be able to find at least a semi-optimal solution.

However, the system needs to be retuned completely if a fourth elevator is installed, or if a faster motor is installed in the elevator. This requires the programmer to design a new controller completely from scratch. Therefore, it would be advantageous to automate this process, such that the elevator system is able to come up with a new controller by itself. This controller aims to optimize certain variables that are defined by the user.

Reinforcement learning is a technique that automates the process of designing a controller. It aims to find a controller that optimizes a user-defined reward function. Reinforcement learning is used by Jonker *et al.* [4] on a soccer robot to learn to dribble with the ball. The use of reinforcement learning was advantageous since hand coding would be very complex, due to the many empirical factors that are present. Also, the behavior can be changed if the environment changes, for example if the ball should be handled differently than usual or if the friction of the ball on the ground is different.

Reinforcement learning is also very important for wide-spread application in household robots. All households are different, requiring a different controller for each robot. Therefore, Schuitema *et al.* [5] designed a robot which uses reinforcement learning to learn to walk. This will enable the future use of household robots in many households without requiring an expert present in each house to tune the robot for the specific house.

Finally, reinforcement learning can provide easy installation for adaptable robots. Small and medium enterprises (SMEs) currently do not often operate robots, since it is very costly to use them for short-term batch process. The installation takes a couple of weeks, which makes it very inefficient to install robots for such a task. However, if a robot is adaptable and can learn a task in a day, it can be applied to do several of these short-term batch processes during the year. The factory-in-a-day project aims to develop this kind of adaptable robots [6]. Reinforcement learning can be used to learn to perform a new task, such that it is not necessary to have an expert present when a new task should be learned. Also, it is possible to save multiple tasks in the memory of the robot.

However, reinforcement learning is a very time-consuming process. It uses interactions with the environment to learn a task. If many tasks should be learned, this process should be repeated many times, making it infeasible to use reinforcement learning in practice. Therefore, it is desired to speed up the process. A literature

¹Here defined as robots that can move their position and orientation in a three dimensional world.

study was performed on this topic and it was found that it is possible to use similarities between tasks to find a controller that can perform many tasks [7]. Using these similarities will decrease the time required to learn.

A state-of-the-art method that uses similarities is Parameterized Skills by Da Silva *et al.* [1]. This algorithm provides an automated approach to learning a set of related tasks, such that a robot can be used in an environment that is allowed to change. It does so by using reinforcement learning to learn some tasks of the set. The similarities between these tasks are used to define how each task in the set can be performed. This is done using supervised learning.

In this thesis, Parameterized Skills will be further examined on a two degree of freedom robot arm using several supervised learning algorithms. This introduction will start with explaining reinforcement learning and Parameterized Skills more thoroughly in section 1.1 and section 1.2, respectively. Then, section 1.3 presents the research questions that are answered in this thesis. This is followed by section 1.4, where the contents of this thesis are presented. The supervised learning algorithms that will be evaluated are introduced, as well as the experiments that are used to answer the research question and the problem on which these experiments are conducted. Section 1.5 summarizes this chapter.

1.1. REINFORCEMENT LEARNING

The goal of reinforcement learning is to find a control policy, π , that maximizes the reward function. The control policy denotes the sequence of control actions that is performed to move from the start position to the goal position, so it defines which action u an agent should take in a certain state x . In reinforcement learning, the control policy can be stochastic or deterministic. The stochastic policy includes the random exploration around the mean, or deterministic policy. The mean policy denotes the control policy that is the optimal solution in the current iteration. This section introduces reinforcement learning and PoWER [8], the state-of-the-art algorithm that will be used in this thesis.

The environment in which the reinforcement learning algorithm operates is modeled as a Markov decision problem, MDP, a discrete time stochastic control process. It is a mathematical system that transitions from a state to a different state due to some action. It is a memoryless process, meaning that the action only depends on the current state and not on the previous states.

Mathematically, an MDP is described as a 4-tuple $(X, U, P(\cdot, \cdot, \cdot), R(\cdot, \cdot, \cdot))$ where X is a set of states, U is a set of actions, $P_u(x, x') = Pr(x_{t+1} = x' | x_t = x, u_t = u)$ is the probability that taking action u in state x yields state x' and R is the set of rewards, where $r_u(x, x') = r(x, u)$ is the (expected) immediate reward after transitioning from state x to state x' . There can also be some reward for reaching a terminal state. Then, the reward R is equal to some cumulative function of the rewards $r(x, u)$ obtained at each state and the reward for reaching the terminal state. The optimal policy is the argument of the maximum of R over all control policies.

PoWER, introduced by Kober and Peters [8], is a state-of-the-art algorithm that directly searches the space of policies to find the optimal policy. This algorithm uses trajectories, sampled from a stochastic policy, to determine the direction in which the policy parameters should be updated. It is based on the theory of expectation maximization (EM), which was introduced in 1977 by Dempster *et al.*

The policy update rule yielding from EM is simplified when the policy is represented by a normal distribution [10] [11]. This means that the mean of the stochastic policy is equal to a set of basis functions multiplied with the policy parameters, which are updated. Gaussian noise is added to the mean policy to perform exploration around the mean policy.

EM is a maximum likelihood method which can be used to estimate the underlying probability density for a set of data X using parameters θ . It is assumed that there is a hidden variable $u \in U$ under each point $x \in X$. This hidden variable can for example be data which is dependent on the value of x . The hidden variables are dependent on parameters θ according to the known probability $p(u|\theta)$. The goal of the algorithm is to find the parameters θ maximizing the sum over all points $x \in X$ of the log of the distribution of $p(x|\theta)$. This performance function is given in equation 1.1, which uses the distribution given in equation 1.2 [12].

$$\theta^* = \arg \max_{\theta} \sum_{x \in X} \log p(x|\theta) \quad (1.1)$$

$$\text{with } p(x|\theta) = \sum_{u \in U} p(x|u, \theta) p(u|\theta) \quad (1.2)$$

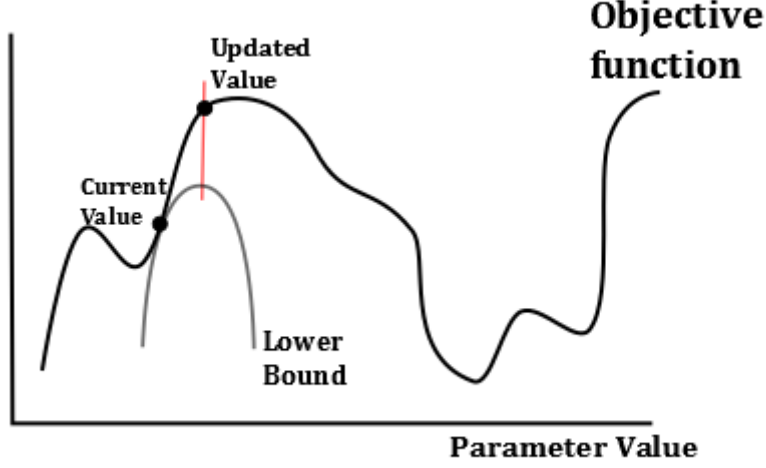


Figure 1.1: Illustration of an EM algorithm. In the E-step a lower bound is computed that touches the objective function at the current parameter value, which is depicted by the parabola. In the M step, the value of the parameters is updated to the location where the lower bound of the function is maximized.

An EM algorithm iteratively updates the policy parameter θ using a lower bound of the expected return. The lower bound simplifies the expected return by making it a function of both the current and the updated parameters. Therefore, it is easier to find the optimal parameters. The lower bound is maximized while it touches the original function [13]. Figure 1.1 illustrates this.

The update consists of two steps. In the Expectation step, or E-step, the so-called responsibilities are updated, which denote how the lower bound is dependent on the current parameters. The responsibilities, $p(u|x, \theta)$, represent the probability of having a certain hidden variable u , given a certain data point x and parameters θ . The E-step defines the shape and the location of the parabola in figure 1.1.

Then, the parameters θ are updated in the Maximization step, or M-step. The parameters are updated such that they maximize the lower bound, as illustrated in figure 1.1. $p(u|x, \theta)$ is kept fixed in this step. The E- and M-step are given in equation 1.3 and equation 1.4, respectively [12].

$$p(u|x, \theta) = \frac{p(u|\theta)p(x|u, \theta)}{\sum_{z \in U} p(z|\theta)p(x|z, \theta)} \quad (1.3)$$

$$\theta' = \arg \max_{\theta} \sum_{x \in X} \sum_{u \in U} p(u|x, \theta) \log[p(x|\theta)] \quad (1.4)$$

In reinforcement learning, the hidden variables can be seen as the actions. Then, the distribution $p(x|u, \theta)$ denotes the expected reward of executing some action u .

PoWER uses the EM steps in an episodic fashion. This means that the reward is only evaluated at the end of a trajectory τ . Then, the parameters are updated using the reward of the full trajectory and previous trajectories.

The policy improvement of PoWER is found by taking the derivative of the lower bound of the expected return, setting the result to zero and solving for the parameters θ . The derivative is given in equation 1.5. $Q^\pi(x, u, t)$ denotes the state-action value function, which is determined using equation 1.6. The resulting update rule depends on the stochastic policy that is used [8].

$$\partial_{\theta'} L_\theta(\theta') = E \left\{ \sum_{t=1}^T \partial_{\theta'} \log \pi(u_t|x_t, t) Q^\pi(x, u, t) \right\} \quad (1.5)$$

$$\text{with } Q^\pi(x, u, t) = E \left\{ \sum_{\tilde{t}=t}^T r(x_{\tilde{t}}, u_{\tilde{t}}, x_{\tilde{t}+1}, \tilde{t}) | x_t = x, u_t = u \right\} \quad (1.6)$$

A policy which consists of a mean policy $u = \theta^T \phi(x, t)$ with parameters θ and basis functions $\phi(x, t)$ will be used. This type of policy yields a simple update rule, as was introduced by Peters and Schaal [10].

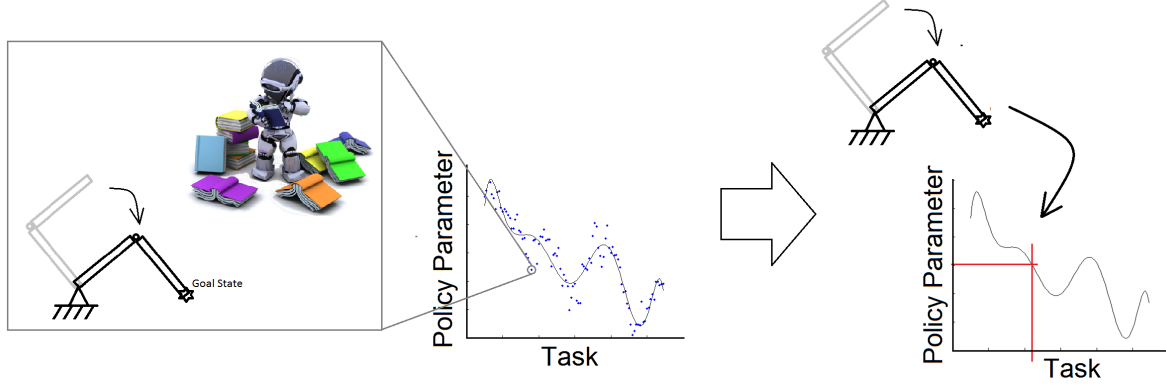


Figure 1.2: Schematic overview of Parameterized Skills. The left graph plots the training tasks versus the optimal parameters. The blue dots depict the training tasks, for which optimal parameters are learned using reinforcement learning. Then, the black line is found using supervised learning. This line depicts the function that defines the parameters as a function of the task. This use of this line is depicted on the right side. It shows how the function can be used to find optimal parameters for a given task.

Exploration is performed by applying Gaussian white noise around the mean policy with a certain variance. It was decided to apply the Gaussian noise to the basis functions $\phi(x, t)$, instead of to the actions. It was found by [Kober and Peters](#) that applying the noise to the latter yielded unrecognizable perturbations.

Inserting this policy in the derivative of the lower bound (equation 1.5) yields equation 1.7 [8]. In this equation, E denotes the expected value, h denotes the current iteration and ε_t denotes the Gaussian noise that is applied at each timestep t . T denotes the final timestep. The term $W(x, t)$ reduces to a diagonal matrix and cancels out.

$$\theta_{h+1} = \theta_h + E \left\{ \sum_{t=1}^T Q^\pi(x, u, t) W(x, t) \right\}^{-1} E \left\{ \sum_{t=1}^T Q^\pi(x, u, t) W(x, t) \varepsilon_t \right\} \quad (1.7)$$

$$\text{with } W(x, t) = \frac{\phi(x, t) \phi(x, t)^T}{(\phi(x, t)^T \phi(x, t))} \quad (1.8)$$

The EM method can only be used if all rewards are greater than or equal to zero. Therefore, a nonlinear transformation of the reward, $u_v(r)$, is performed to make sure this requirement is met. This transformation should be strictly monotonic with respect to r and it should also hold that $\int_0^\infty u_v(r) dr = c^{st}$ [10].

POWER finds parameters that optimize the reward for a certain task. This process should be repeated very often if one aims to use the controller to perform many tasks. To speed up learning, similarities between these tasks can be exploited. Parameterized Skills [1] is a state-of-the-art algorithm that uses similarities to learn to perform a set of related tasks.

1.2. PARAMETERIZED SKILLS

Parameterized Skills was introduced by [Da Silva et al.](#). It provides a framework that defines how reinforcement learning can be used to find optimal policy parameters for training goals and how supervised learning can be used to fit a model through the data obtained using reinforcement learning.

Figure 1.2 shows how this algorithm works graphically. The left side shows how the algorithm trains the controller and the right side shows how the controller can be used. The blue dots in the left graph depict the training tasks, for which optimal parameters are learned using reinforcement learning. The black line denotes the function that is fitted through this data. This function can be used to find the parameter that should be used for a given task, as shown on the right side. Next, Parameterized Skills will be introduced in more detail.

Parameterized Skills aims to maximize the expected reward over the whole distribution of possible tasks. This is equivalent to maximizing the expected reward as given in equation 1.9, where for each task the expected reward of following policy $\pi_{\Theta(\tau)}$ in task τ is determined, which is multiplied with the probability of executing this task and integrated over all tasks [1].

$$\int P(\tau) J(\pi_{\Theta(\tau)}, \tau) d\tau \quad (1.9)$$

$\pi_{\Theta(\tau)}$ denotes the policy parametrized by the model $\Theta(\tau)$. This model defines the policy parameters as a function of the task, τ is a task parameter vector drawn from a $|T|$ -dimensional continuous space, T , and $J(\pi_{\Theta(\tau)}, \tau)$ is the expected reward that will be obtained when executing policy $\pi_{\Theta(\tau)}$ in task τ . The probability $P(\tau)$ defines the probability of a task occurring [1].

The model $\Theta(\tau)$ maps the task parameters to policy parameters, so the policy parameters are chosen based on the task which is currently performed. This is described in equation 1.10 [1].

$$\Theta : T \rightarrow \mathbb{R}^N \quad (1.10)$$

This model can be found using a set K of pairs $\{\tau, \theta_\tau\}$, with τ being a set of task parameters and θ_τ denote the parameters which maximize the reward for this specific task. Reinforcement learning is used to find these parameters [1].

The model will then be created using these pairs. Several assumptions are made about the structure of the model before this is done. It is assumed that the tasks, for which the model is constructed, are related. Therefore, it is assumed that they lie on some lower-dimensional surface in \mathbb{R}^N and their parameters vary smoothly. It is also possible that the policies lie on two or more disjoint lower-dimensional surfaces [1].

This is a reasonable assumption if the policy is differentiable with respect to the parameters. This means that the gradient of the reward function, $J(\theta)$ defines the direction of the update such that the reward is maximized locally. For example, if one optimizes the policy for how accurate it reaches its goal, parameters can be found that yield a goal slightly different from the current goal. The gradient then defines the direction of the update, yielding a smooth update of the parameters. However, these parameters can also be used to define the optimal policy towards a different goal [1].

These assumptions yield the following procedure to construct the model. First, one should draw $|K|$ training tasks from T and construct K , the set of the training tasks and the parameters that maximize the reward for these tasks. Next, K can be used to estimate the geometry and topology of the policy space. This is done to define the number of surfaces, D , present in the policy space, on which the policies lie. This is important since a separate model should be constructed for each surface. Third, a classifier, χ , should be trained to map all elements of T to one of the surfaces, $[1, \dots, D]$. Finally, a set of $(N \times D)$ supervised learning models, $\Phi_{ij}, i \in [1, \dots, D], j \in [1, \dots, N]$, should be found which define the policy parameters as a function of the task [1].

As said, a separate model is created for each surface. The full model consists of the vector of all supervised learning models of one surface and is given in equation 1.11 [1].

$$\Theta(\tau) = [\Phi_{\chi(\tau),1}, \dots, \Phi_{\chi(\tau),N}]^T \quad (1.11)$$

1.3. RESEARCH QUESTION

Parametrized Skills is a very versatile algorithm. It can be used in combination with any reinforcement learning and supervised learning algorithm. However, three issues should be solved before Parameterized Skills can be applied on a two degree of freedom robot arm. They will be explained next.

- The controller found with Parameterized Skills does not perform well enough
- The supervised learning algorithm used by Da Silva *et al.* [1] requires much time to construct the model, especially for a large data set
- The algorithm has never been applied to learn a task with more than two varying dimensions

Da Silva *et al.* investigated the performance yielding from the controller that was learned. They found that extra reinforcement learning iterations were required to meet the threshold set on the performance, meaning that the performance found using Parameterized Skills is not as good as the performance that is found using reinforcement learning.

Parameterized Skills uses support vector regression (SVR) to determine the relation between the goal and the required control action. SVR uses an optimization procedure to construct the model, which is a slow method to find a function. Additionally, this optimization procedure aims to find a vector of which the length equals the size of the data set, which means that the required time increases even further for a larger data set.

Third, the application of the algorithm should be extended. Parameterized Skills has only been applied to learn a task with a varying, two-dimensional goal position. It should be checked if it is possible to apply Parameterized Skills to learn a task with more dimensions. In this study, the algorithm will be implemented to a task with a varying two-dimensional start position and a varying two-dimensional goal position, meaning that the task has four dimensions. It is expected that the size of the data set that is used in the supervised learning algorithms should be increased quadratically to obtain the same performance.

My aim is to solve these first two issues by implementing a different supervised learning algorithm in Parameterized Skills. Polynomial regression, a neural network algorithm and locally weighted linear regression will be compared against SVR. These algorithms will be implemented on a task with a varying goal position and the performance of Parameterized Skills is compared for the different methods. The algorithm will be selected that yields better or similar performance than SVR, while requiring less time to do so. This algorithm is then used to analyze the performance of Parameterized Skills on a task with a varying start and goal position.

Three research questions and some hypotheses are defined to formalize the issues. The main research question that is answered in this thesis is as follows:

Which supervised learning algorithm provides the best performance of Parameterized Skills in terms of accuracy and energy efficiency of the controller found and in terms of required time and when the algorithm is used to learn a more complex task, namely to learn rest to rest motions?

Two steps should be taken to answer this research question. First, the performance of Parameterized Skills is evaluated on a task with only a varying goal state, as was done previously. Second, it is evaluated if the algorithm can be applied to train a controller to perform a more complex task, with up to four dimensions. Therefore, two subquestions will be answered in this thesis.

The first question is: **Which supervised learning algorithm provides the best performance of Parameterized Skills in terms of accuracy and efficiency and in terms of time required if a different supervised learning algorithm than support vector regression is used?**

It is expected that polynomial regression and locally weighted linear regression will improve the speed of the algorithm, since they do not require an optimization procedure. Also, these algorithm and the neural network algorithm do not aim to find a vector with the length equal to the size of the data set. Therefore, the time required to use these three methods will not increase as much when a larger data set is required, compared to SVR.

Next to this, it is expected that polynomial regression and the neural network algorithm yield better performance due to the flexibility of these algorithms. It is expected that locally weighted linear regression provides better performance since it matches well with the assumption of Parameterized Skills that similar tasks yield similar parameters. This algorithm uses only a local region to find the output for the current target.

The second question that can be deduced from the research question is: **Does the performance of the supervised learning algorithms scale in terms of the data set size required to construct the model with an increased task dimension?**

Since the dimension of the input space is larger, the data set should be bigger to obtain the same performance, because of the increased complexity of the function. It is expected that the size of the data set should increase quadratically if the size of the input space is doubled.

1.4. CONTENTS OF THIS THESIS

This thesis aims to solve the issues that exist when applying Parameterized Skills by implementing a different supervised learning algorithm. Four algorithms will be evaluated and experiments are used to assess their performance. This section introduces the algorithms that will be used in the evaluation. Next to this, the experiments that are conducted are introduced. These experiments are used to verify that the implementation of all reinforcement and supervised learning algorithms is correct and to answer the research question. The problem that is used in these experiments is also introduced in this section.

1.4.1. SUPERVISED LEARNING ALGORITHMS

Supervised learning is used in Parameterized Skills to find a function that should return parameters that optimize the reward for a specific task. [Da Silva et al.](#) use support vector regression (SVR). Three algorithms are compared against SVR to evaluate their performance and time required. These algorithms are polynomial regression, a neural network algorithm and locally weighted linear regression. This section starts with an introduction of SVR, followed by these three algorithms, respectively.

SUPPORT VECTOR REGRESSION

Support vector regression (SVR) is a supervised learning algorithm originating from the framework of statistical learning. It aims to find a function that fits the data as smooth as possible, while allowing a maximum error for outliers. This problem can be converted to a dual problem, where an inner product of the input data is maximized. Many different options exist, since the input can be processed using many different kernel functions, special mappings which use some inner product of the input [\[14\]](#).

SVR aims to find a function that is as smooth as possible while outliers should not exceed a maximum difference with respect to the function output. This function is found using a minimization problem, which aims to minimize the parameters that define the smoothness of the function. The maximum error between the outliers and the function output, ϵ , is a constraint on this problem. This constraint is generally not implemented as rigid as stated here, since this problem would be a very difficult to solve. Therefore, slack variables are introduced, which are added to ϵ to stretch the constraint on the difference between the output of the function and the test data.

These slack variables are also minimized. This means that the objective function, which is minimized, is equal to the sum of the smoothness parameters and these slack variables, multiplied with a constant, C . This constant defines the trade off between minimizing the flatness of the function and minimizing the value of the slack variables [\[14\]](#).

The aforementioned problem can be converted to a dual problem using Lagrangian multipliers a_i and a_i^* for the constraint on the original problem. Smola and Schölkopf [\[14\]](#) show that the following dual function can be minimized instead of the original optimization problem, where $\langle \cdot, \cdot \rangle$ denotes the dot product and x_i and y_i an input-output set of the test data, which has a total of l input-output sets [\[14\]](#).

$$\max \quad -\frac{1}{2} \sum_{i,j=1}^l (a_i - a_i^*)(a_j - a_j^*) \langle x_i, x_j \rangle - \epsilon \sum_{i=1}^l (a_i + a_i^*) + \sum_{i=1}^l y_i (a_i + a_i^*) \quad (1.12)$$

$$\text{s. t.} \quad \sum_{i=1}^l (a_i - a_i^*) = 0 \quad \text{and} \quad a_i, a_i^* \in [0, C] \quad (1.13)$$

In equation [1.12](#), a kernel function $k(x, x')$ can be inserted instead of the dot product [\[15\]](#). This way, many different functions can be constructed using SVR. The resulting function is shown in equation [1.14](#). This function uses the Lagrangian multipliers a_i and a_i^* , which are optimized in the maximization problem, the original inputs of the test data and, if present, the constant term b .

$$f(x) = \sum_{i=1}^l (a_i - a_i^*) k(x_i, x) + b \quad (1.14)$$

It is expected that the performance of SVR is not as good as desired due to its assumption that the function should be as smooth as possible and that outliers are allowed to exist. The data used to create the model consists of optimal parameters that are learned using reinforcement learning. Therefore, it is expected that there will not be outliers in the data and that the data will be quite smooth.

Next to this, SVR is a slow method, since it uses an optimization procedure to find a vector of which the length is equal to the size of the data set. An optimization procedure is slower than the fitting procedures used in polynomial regression and locally weighted linear regression. Next to this, the time required to construct this vector increases further when the size of the data set increases, which can yield a large increase in the required time when the algorithm is applied to a task with a varying start and goal position.

POLYNOMIAL REGRESSION

Polynomial regression assumes that a polynomial function fits the test data. The degree of this polynomial is chosen specifically for the data. A polynomial function of degree k has the structure given in equation [1.15](#),

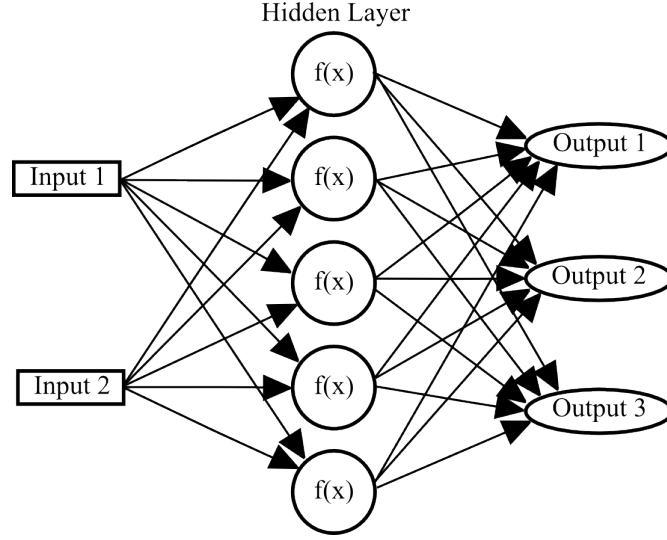


Figure 1.3: Example of An Artificial Neural Network

where x denotes the input data and a_0 to a_k denote unknown variables [16].

$$y = a_0 + a_1x + a_2x^2 + \dots + a_kx^k \quad (1.15)$$

The polynomial, which was given in equation 1.15, should fit the data correctly. This fit is found by adjusting the variables a_0 to a_k using least square fitting. This is done using a matrix that contains the input data, a vector that contains the unknown variables and a second vector that contains the outputs, as given in equation 1.16. Here, n defines the number of training sets of input and output are used [16].

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^k \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix} \quad (1.16)$$

This problem is solved for each row, which can all yield a different result for the unknown variables. Therefore, least square fitting is used to find the set of variables for which the error between the test data output and model data output is least. This means that the total squared error, given by $\sum_{i=1}^n [P(x_i) - y_i]^2$, is minimized, where $P(x_i)$ is the output of the polynomial function and y_i is the output found in the training data [17].

The equations presented above are applicable to problems with a single input. The extension to multiple inputs is straightforward. Note that cross-terms should also be taken into account when there is more than one input to the problem.

Polynomial regression is very easy to use, since only one parameter should be tuned, the degree of the polynomial. When this degree is chosen sufficiently high, it is possible to create any type of function using polynomial regression, since it can be seen as the Taylor expansion of the original function. It is one of the methods that is used most often for curve fitting [18] [19]. The reputation of this method yielded its selection to be used as a supervised learning algorithm in Parameterized Skills.

NEURAL NETWORK ALGORITHM

An artificial neural network is an input-output system which is inspired by the biological process in the human neural network, where a neuron collects signals through dendrites, a large structure of inputs. The neuron then transfers this signal to an output signal which is transported to a large structure of outputs.

This biological process can be transformed to an artificial process. In an artificial neural network, there are usually three layers of neurons, the input layer, the hidden layer and the output layer. Figure 1.3 gives an example of an artificial neural network. The input layer feeds the inputs, multiplied with some weight, to the hidden layer, where there is some nonlinear function applied to the input. The output of the nonlinear function is multiplied with a weight and fed to the output layer.

It is possible to have multiple hidden layers. Then, the output of the first hidden layer will be fed to the second and so on until the final hidden layer feeds the output to the output layer. The number of neurons in the input layer is equal to the number of inputs and the number of neurons in the output layer is equal to the number of outputs. The hidden layers can have any number of neurons.

An artificial neural network can be used to fit data in a supervised learning algorithm. This means that the weights of the connections are adjusted such that the output of the neural network fits the output of the test data. This is done using backpropagation. This is a method that starts at the output and defines the gradient of the weights with respect to the difference between the desired output and output found by the neural network. This gradient is used to update the weights.

A neural network can fit into many different types of training data due to its versatility [20]. Therefore, it is expected that a neural network algorithm can find a good fit for the training data in Parameterized Skills.

LOCALLY WEIGHTED LINEAR REGRESSION

Locally weighted linear regression is a supervised learning algorithm where a local linear model is constructed around each test input τ_{des} . Therefore, the model is not constructed separately. The local linear model is constructed using the following steps, following the method introduced by Atkeson *et al.* [21].

First, k nearest neighbors are found around the test goal, where k is a tuneable parameter. The inputs and outputs of the nearest neighbors are stacked in N_I and N_O , respectively. Next, the distances d_i between the nearest neighbors and the test goal are determined using the 2-norm and their weight w is determined using equation 1.17.

$$w_i = \exp \left[- \left(\frac{d_i}{h} \right)^2 \right] \quad (1.17)$$

$$\text{with } h = \max_i d_i \quad (1.18)$$

Now, the weighted input and output matrices are defined as follows, where a vector of ones is added to the input matrix to account for a constant term in the regression [21].

$$A = w[N_I, \mathbf{1}] \quad (1.19)$$

$$B = wN_O \quad (1.20)$$

Equation 1.21 is solved using matrices A and B to find the prediction of the parameter for the test goal.

$$\theta = \tau_{des} X \quad (1.21)$$

$$\text{with } X = (A^T A)^{-1} A^T B \quad (1.22)$$

Locally weighted linear regression is implemented in Parameterized Skills since it connects well with the assumption made in Parameterized Skills. It is assumed that the tasks are related and therefore that they lie on a lower dimensional subspace in the policy space. Locally weighted linear regression only takes into account a local subset of the training data when it finds parameters for a test goal. Therefore, these assumptions hold better than when a model is created for the complete data set, since the used tasks are less different. An additional advantage is that it is not necessary to analyze the geometry and topology of the policy space using this algorithm, since the algorithm uses only nearest neighbors, which will all lie on the same subspace as the target.

1.4.2. METHOD TO ANSWER RESEARCH QUESTION

Experiments are conducted to answer the research questions that are given in this introduction. Figure 1.4 presents an overview of all experiments that are conducted in this study. Experiments 1 to 4 are conducted to verify that the implementation of the reinforcement learning algorithms and the supervised learning algorithms is correct. This implementation is presented in chapter 2. Chapter 3 presents the results of these experiments.

Experiments 5 to 9 are conducted to answer the research question. Chapter 4 evaluates the four supervised learning algorithms using experiments 5, 6 and 7. These experiments are used to analyze the performance of Parameterized Skills on a problem with only a varying goal state. Models are constructed using data sets of

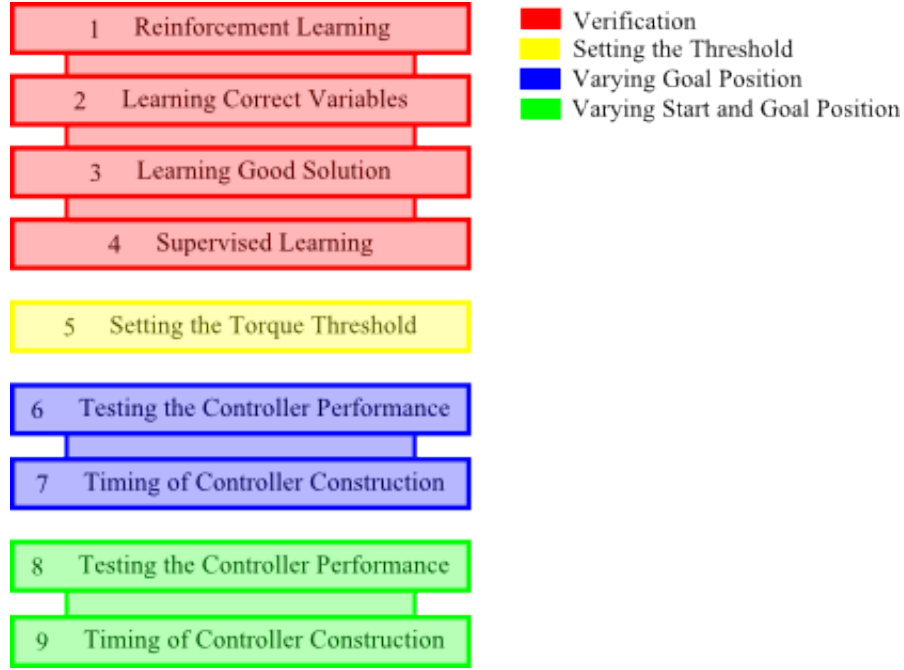


Figure 1.4: Overview of all experiments conducted in this thesis.

different sizes. These models are evaluated using testing tasks. The minimum required data set size is found and the performance yielding from models using a data set of this size is compared. Also, the time required to construct the controller is compared, since it is required to achieve a speed up.

Chapter 5 evaluates the performance of the supervised learning algorithms on a task with a varying start and goal state. This chapter will only analyze the supervised learning algorithms that perform best on a task with only a varying goal position. Experiment 8 is used to analyze if the performance of Parameterized Skills is similar when the size of the data set is increased quadratically. Experiment 9 is used to compare the time required by the different algorithms.

These experiments aim to find an optimal control policy to perform tasks on a two degree of freedom robot arm. Therefore, this section introduces the two degree of freedom robot arm that is used in this thesis, as well as the control policy that is optimized.

TWO DEGREE OF FREEDOM ROBOT ARM

Figure 1.5 shows the robot arm which is used in these experiments. All experiments are applied on a simulation of this arm. It consist of two links which are connected using joints. There are two degrees of freedom, namely the rotation of the links around the joints. These rotations are performed in the same plane. The direction of gravity is perpendicular to the robot arm, which means that the effect of gravity is negligible.

Table 1.1 shows the weights and dimensions of the arm as well as its range of motion for each state. The robot has four states, namely the angles at the joints, which are drawn in figure 1.5 (b) and their velocities.

A torque can be applied to both degrees of freedom to control the robot arm. This torque is applied using current-control, which means that the applied torque is equal to the product of a motor constant, the gearbox ratio and the current that is running through the motor [22]. This is equivalent to applying open-loop torque control. The next section will introduce the control policy which is used to define the torque that should be applied to the robot arm. Two policies are required, since each is applied to one of the controlled joints.

CONTROL POLICY

Parameterized Skills aims to find optimal control policies for many tasks by exploiting similarities between the tasks. A control policy that is very suited for this task is a dynamic movement primitive (DMP) [23]. DMPs were introduced by Ijspeert *et al.* [24] and represent a policy using parameters, such that a function can be found that relates the task to these parameters. These parameters can be learned using reinforcement learning and uniquely describe the control actions which are taken.

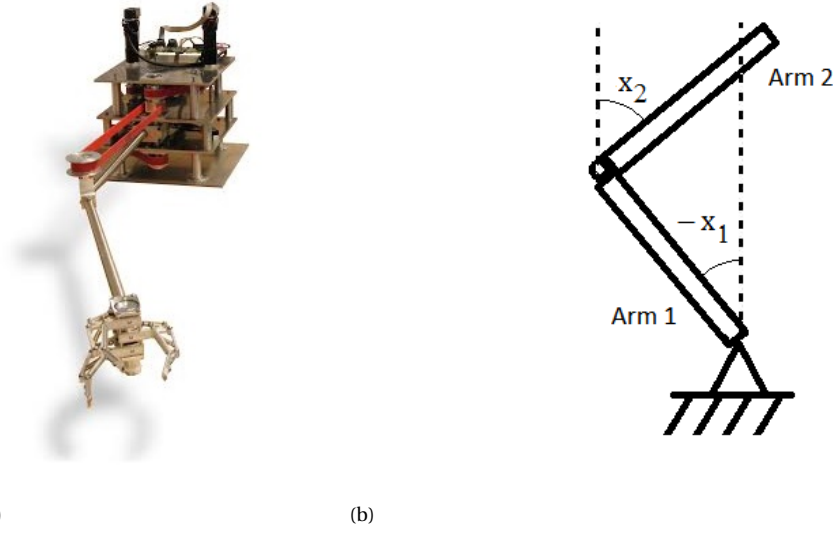


Figure 1.5: Figure (a) shows a picture of the robot arm that is used. Figure (b) depicts a schematic drawing of this robot arm.

Table 1.1: Overview of the weights, dimensions and range of motion of the robot arm. The range of motion defines the goal and start position space to which the controllers can be applied which are learned using Parameterized Skills

Parameter	Symbol	Value
Length of Arm 1	l_1	0.41 [m]
Length of Arm 2	l_2	0.45 [m]
Location of Center of Mass of Arm 1	l_{CoM_1}	0.070 [m]
Location of Center of Mass of Arm 2	l_{CoM_2}	0.3247 [m]
Weight of Arm 1	m_1	0.809 [kg]
Weight of Arm 2	m_2	1.599 [kg]
Friction Coefficient of Arm 1	μ_{C_1}	0.1939 [Nm]
Friction Coefficient of Arm 2	μ_{C_2}	0.25 [Nm]
Range of Motion of Arm 1	$[x_{1_{min}} \ x_{1_{max}}]$	$[-1.25 \ 1.25]$
Range of Motion of Arm 2	$[x_{2_{min}} \ x_{2_{max}}]$	$[-2.5 \ 2.5]$

The DMP determines the desired state, $[x_{des}^T(t), v_{des}^T(t)]^T$, that should be visited in the next timestep to move to the goal. The torque which is applied to the robot arm is determined using a PD controller, as given in equation 1.23, where K_P and K_D denote the position and velocity gain, respectively. The state, $[x^T(t), v^T(t)]^T$, that yields from applying torque are inserted in the DMP to find the next desired state.

$$M(t) = K_P \cdot (x_{des}(t) - x(t)) + K_D \cdot (v_{des}(t) - v(t)) \quad (1.23)$$

A DMP consists of two separate dynamic systems, an attractor system and a nonlinear system. These systems are connected such that the attractor system drives the nonlinear system. The attractor system attracts the robot to the goal state. The nonlinear system adds a forcing term to the attractor system, to allow any kind of movement towards the goal. The systems have a unique equilibrium point at the desired goal.

The attractor system is a canonical system that drives the nonlinear system, $f(s)$, for each degree of freedom k . These nonlinear systems define the so-called forcing term. This term is added to the attractor system, which has a shape similar to a PD controller. The dynamics of the attractor system are given below in equations 1.24 and 1.25 Da Silva *et al.* [1]. In these equations, κ denotes a scaling factor, which is set to one, g denotes the desired goal position and x_0 denotes the start position.

$$\kappa \dot{v}_{des}(t+1) = K(g - x(t)) - Qv(t) + (g - x_0)f(s) \quad (1.24)$$

$$\kappa \dot{x}_{des}(t+1) = v(t) \quad (1.25)$$

The nonlinear system consists of a set of nonlinear basis functions Ψ_i , which are activated depending on the phase s and their parameters w_i . The phase is a time-dependent variable which monotonically decreases from 1 to 0 during the execution of the movement. It is computed by integration $\kappa \dot{s} = -\alpha s$, where α is a factor that depicts how fast the phase decreases. The forcing term is determined using equation 1.26, with N equal to the total number of basis functions. The basis functions are determined using equation 1.27, where h_i and c_i denote the height and the center of basis function i , respectively. Kober and Peters [25].

$$f(s) = \sum_{i=1}^N \Psi_i(s) w_i \quad (1.26)$$

$$\text{with } \Psi_i = \frac{\exp -h_i(s - c_i)^2}{\sum_{j=1}^N \exp -h_j(s - c_j)^2} \quad (1.27)$$

A reinforcement learning algorithm can be used to learn the parameters w_i of the nonlinear system for a given task. These optimal parameters are used to create a forcing term that yields optimal reward. The next section introduces how a model can be created which defines the parameters w_i as a function of the goal.

1.5. SUMMARY

This thesis will evaluate the effect on the performance of Parameterized Skills in terms of accuracy and energy efficiency and in terms of time required when different supervised learning algorithms are implemented and when the dimension of task is increased. Therefore, this introduction presented all theories and algorithms that will be used in this thesis, namely reinforcement learning, Parameterized Skills, the supervised learning algorithms that are evaluated, the problem on which experiments are conducted and the control policy that is used.

Reinforcement learning is used to learn to perform a task such that a user-defined reward function is maximized. The robot interacts with the environment and collects the reward. The reward is used to define the interaction that is performed in the next iteration, such that a higher reward is obtained.

Parameterized Skills uses reinforcement learning to learn to perform a set of related tasks. It assumes that the optimal policies for related tasks lie on a lower-dimensional surface in the policy space and that the optimal policy parameters vary smoothly. Therefore, it is possible to find a model over the full task space that defines the control policy as a function of the goal. This model is constructed using training data, consisting of training tasks and their optimal control policy.

This model is found using supervised learning. Three supervised learning algorithms will be compared against support vector regression (SVR): polynomial regression, a neural network algorithm and locally weighted linear regression. SVR aims to optimize variables of a function such that the function is as smooth as possible while not having too much outliers. Polynomial regression aims to fit a polynomial function through the

data. A neural network algorithm aims to train the weights of a neural network using backpropagation. Locally weighted linear regression constructs local linear models using k nearest neighbors for a certain input.

The experiments that are conducted in this study are conducted on a two degree of freedom robot arm. This arm can perform planar motions. It consists of two links that are connected with joints that allow rotation around a single axis. A torque can be applied to both degrees of freedom using current-control, so the applied torque is equal to the product of a motor constant, the gearbox ratio and the current that is running through the motor.

The control policy that is applied to this arm and learned using reinforcement learning is represented using a dynamic movement primitive (DMP). A dynamic movement primitive consists of an attractor system which attracts the robot to the goal state. The attractor system drives a nonlinear system, which applies a forcing term to the movement defined by the attractor system, such that in principle any motion can be achieved from the start to the goal state.

2

IMPLEMENTATION OF PARAMETERIZED SKILLS

The previous chapter introduced the theories and algorithms that are used in this study as they are presented in literature. This chapter will introduce how these algorithms are implemented in Parameterized Skills. Certain parts of the reinforcement learning algorithm are altered to improve the learning process. Next to this, the reward function should be defined such that learning is achieved for the correct variables. Next to this, an algorithm should be implemented that is able to find the number of subspaces that are spanned in the policy space. Finally, the supervised learning algorithms and the control policy required tuning of certain parameters.

This chapter starts with an introduction of the implementation of Parameterized Skills. First, the implementation of the reinforcement learning algorithm is presented. This includes the alterations made to the algorithm and the reward function that is used. Second, this section introduces ISOMAP. This algorithm is used to find the number of subspaces that are spanned in the policy space. Third, it is introduced how certain variables are tuned in the supervised learning algorithms that are compared in this thesis. Then, section 2.2 introduces the implementation of the control policy. A dynamic movement primitive (DMP) is used to represent the control policy. Several variables should be set in the DMP such that the behavior of the control policy is as expected. Section 2.3 summarizes the chapter.

2.1. IMPLEMENTATION OF PARAMETERIZED SKILLS

This section will present the implementation of Parameterized Skills. Parameterized Skills consists of four steps that should be implemented. First, optimal parameters are found for training goals using reinforcement learning. Next, the geometry and topology of the policy space is analyzed to find the number of subspaces spanned by the optimal policies. The third step is to find in which subspace each training goals lies. Finally, models can be constructed using supervised learning. Section 1.2 introduced Parameterized Skills and the reinforcement learning and supervised learning algorithms that will be used in this study. ISOMAP is used to perform the second and third step of the algorithm.

This section introduces how these algorithms are implemented. Section 2.1.1 presents the implementation of the reinforcement learning algorithm, as well as the implementation of the reward function. Then, section 2.1.2 presents ISOMAP, which is used to perform the second and third step. Finally, section 2.1.3 presents the implementation of the supervised learning algorithms.

2.1.1. IMPLEMENTATION OF REINFORCEMENT LEARNING

Parameterized Skills uses reinforcement learning to find optimal parameters for training goals. Section 1.1 introduced PoWER [8], the reinforcement learning algorithm that will be used. This algorithm finds optimal parameters for the control policy using exploration around the mean policy. A reward is given to each trajectory that is explored based on its performance, which is used to find the direction in which the parameters should be updated.

This section explains how reinforcement learning is implemented. It was required to change the algorithm on three aspects to improve the learning. First, the exploration around the mean policy was implemented

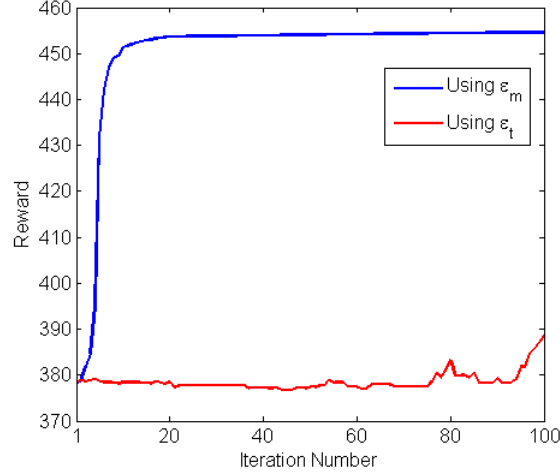


Figure 2.1: Reward obtained when learning to move from $[x_1, x_2] = [0, 0]$ to $[x_1, x_2] = [\pi/2, \pi]$ on the robot arm that was introduced in section 1.4.2, using time-dependent exploration (red line) and trajectory-dependent exploration (blue line). One can see that the trajectory-dependent exploration yields a much better result. The reward increases and converges to a significantly higher value. The reward improves shows more oscillation and does not yield a very high reward.

differently from the original algorithm. Secondly, the update rule was altered to ensure convergence of the algorithm. Finally, it will be explained how the importance weight is implemented such that previously sampled trajectories improve the policy update.

The reward function is very important for the learning process, since reinforcement learning aims to optimize the reward function. Therefore, this function should be defined such that the algorithm optimizes for the correct variables, namely energy and accuracy. This section will also introduce the implementation of the reward function.

EXPLORATION

Exploration is performed to evaluate the reward of policies which are close to the current policy. This information is used to update the policy in a direction such that the updated policy yields a higher reward. This exploration can be applied to the policy in different ways. PoWER, as introduced in section 1.1, uses different exploration on each timestep. However, it was found that the learning process can be improved by applying exploration to a trajectory. This also alters the update rule.

In PoWER, exploration is applied to the policy parameters as white Gaussian exploration, $\varepsilon(x, t) = \varepsilon_t^T \phi(x, t)$ with $[\varepsilon_t]_{ij} \sim N(0, \sigma_{ij}^2)$, which is applied on each time step in the trajectory. This exploration is dependent on the time step, which is clear by the subscript t on ε .

In the update rule, which was given in equation 1.7, the action-state value is determined for each timestep as the sum of the rewards from the current to the final timestep. However, these next timesteps have exploration which is different from the current exploration. This means that different parameters are used in the next timesteps. Actually, different parameters are used in every timestep. Therefore, it is not possible to reproduce this exact trajectory using a deterministic policy, which has fixed parameters over the full trajectory. So, the action-state values found in such trajectory cannot be reproduced by any policy.

Figure 2.1 shows that this causes bad results. This figure was obtained when reinforcement learning was applied to the two degree of freedom robot arm, which was presented in section 1.4.2, to learn to move from $[x_1, x_2] = [0, 0]$ to $[x_1, x_2] = [\pi/2, \pi]$. The motion should optimize the reward function presented in section 2.1.1. The red line shows the result that is found using time dependent exploration.

One can see that the algorithm showed difficulties in learning correct policy parameters. Therefore, exploration was implemented differently. Instead of time-dependent exploration ε_t , it was decided to have trajectory-dependent exploration ε_m . This exploration was also implemented as white Gaussian noise, but the exploration has a single value for each parameter over the full trajectory. So, the samples were created with the same parameters at every timestep. This also means that the resulting state-action values can be reproduced by a deterministic policy.

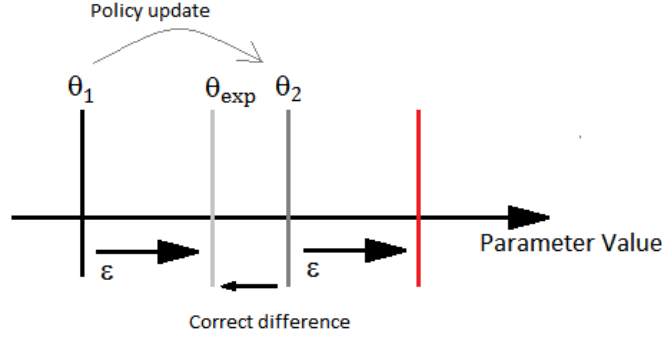


Figure 2.2: Illustration of implementation of the update rule. If one would use the original exploration, ϵ , one would take into account the incorrect, red parameter value for the light grey exploratory policy, sampled under policy 1. Instead, one should take into account the difference between policy 2 and the light grey parameter value which was used in the original sampling.

One can see that the algorithm is able to find a better solution when the exploration is trajectory dependent. The red line, that shows the result using time-dependent exploration, yields hardly any increase in reward. On the other hand, the reward is increased significantly when trajectory-dependent exploration is used, as shown by the blue line.

This simplifies the update rule. The original update rule (see equation 1.7) determines the change in policy parameters for each sample by summing a product of the time-dependent exploration and the action-state values over time and normalizing this using the sum of the action-state values. Since the exploration is now policy dependent, it is possible to move the exploration out of the sum. This yields the update rule given in equation 2.1, where $\omega(m)$ denotes the importance weight of trajectory m . M denotes the set of trajectories that are sampled.

$$\theta_{h+1} = \theta_h + \sum_{m=1}^M \epsilon_m \cdot \omega(m) \cdot \sum_{t=1}^T Q^\pi(x, u, t) / \sum_{m=1}^M \omega(m) \cdot \sum_{t=1}^T Q^\pi(x, u, t) \quad (2.1)$$

UPDATE RULE

The update is based not only on the trajectories sampled using the current policy, but also on trajectories which were sampled under previous policies. The update rule given in equation 2.1 implies that the exploration, that was applied to the trajectories when they were sampled using their original policy, should be used to determine the update of the current policy. This section explains why the learning does not converge properly when this approach is used.

Figure 2.2 illustrates this. The parameters of the current policy are equal to θ_1 . A trajectory is sampled with exploration noise equal to ϵ , yielding parameters θ_{exp} . This trajectory, among others, is used to update the current policy to the policy with parameters θ_2 . The sampled trajectories are used again. However, if one would use ϵ to take into account the trajectory sampled with θ_{exp} , the parameter value depicted by the red line would be used.

However, the trajectory found using θ_{exp} is not found using this parameter value, therefore the result would be incorrect. Instead, one should use the difference between the current parameters θ_2 and the parameters θ_{exp} , since this corresponds to the correct parameter value. This means that the update rule should be altered to equation 2.2, where θ_m denote the parameters used to sample trajectory m . θ_m is equal to the sum of the policy parameters of the mean policy when the trajectory was sampled and the exploration which was used to sample this trajectory, ϵ .

$$\theta_{h+1} = \theta_h + \sum_{m=1}^M (\theta_m - \theta_h) \cdot \omega(m) \cdot \sum_{t=1}^T Q^\pi(x, u, t) / \sum_{m=1}^M \omega(m) \cdot \sum_{t=1}^T Q^\pi(x, u, t) \quad (2.2)$$

IMPORTANCE SAMPLING

Importance Sampling is a technique that enables the use of trajectories, sampled under a previous policy, in the current policy update. The use of policies that were sampled previously increases the number of samples

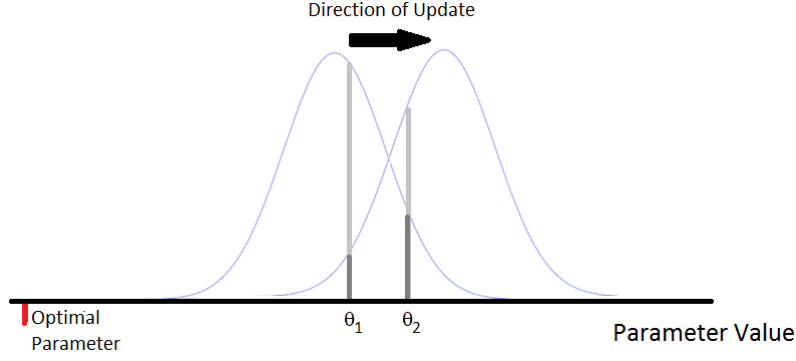


Figure 2.3: One dimensional illustration of the disadvantage of using the likelihood of occurrence for certain parameters. The red line denotes the location of the optimal parameter. The dark and light grey line denote the likelihood of occurrence under the current and previous policy parameter, respectively. The blue graph indicates the normal spread using which the exploration is defined. One can see that the previous parameter update was performed to the right, while the optimal parameters are located left of the current parameters. One can see that parameter θ_1 will receive a lower importance weight than parameter θ_2 , since the likelihood of occurrence of this parameter is lower for the current parameters. Therefore, θ_1 will receive less weight during the update, even though this parameter is closer to the optimum than θ_2 .

on which the expected reward is based, which means that the estimation of the expected reward will be more accurate. However, old trajectories can have a negative effect on the policy update if it is not likely that they are sampled under the current policy. For example, a trajectory that was sampled before the reward was improved significantly will negatively influence the update due to its low reward. Therefore, it is necessary to discard such policies. This section introduces how this is done.

The importance weight used by [Kober and Peters](#) in PoWER was interpreted such that the trajectories all receive an importance weight based on the chance that a trajectory will be sampled under the current policy. This importance weight is equal to the ratio between the likelihood of occurrence in the original policy and the likelihood of occurrence in the current policy.

It was found that this importance weight does not take into account the reward of the trajectories. It is possible that the update is executed in a direction yielding a lower reward if the number of sampled trajectories is low in the initial stages of the learning process. If this happens, trajectories which yielded a low reward will now receive a higher importance weight than trajectories which yielded a higher reward. This means that worse trajectories will have more weight in the next update.

Figure 2.3 illustrates the problem when a single parameter is learned. The location of the optimal parameter is denoted by the red line. Parameters θ_1 and θ_2 were sampled in the previous iteration and the update which was found after this iteration is in the wrong direction with respect to the optimal parameter.

Now, when the importance weight is determined for these parameters, θ_1 will receive a lower importance weight, since its likelihood of occurrence for the new policy parameters is lower. Therefore, this parameter will also receive less weight during the update, even though its location is closer to the optimal parameter than the location of θ_2 . Therefore, it is possible that the update will be in the wrong direction again and the algorithm will not be able to find the correct optimal value.

Therefore, the importance weight was determined based on the state-action value of the sampled trajectory, Q^π in my implementation, instead of on the likelihood of the trajectory. During each iteration, the importance weight was determined as the ratio of the state-action value of the sampled trajectory Q_m^π and the state-action value of the current policy Q_h^π . This ratio was found using equation 2.3.

$$\omega(m) = \frac{Q_m^\pi}{Q_h^\pi} \quad (2.3)$$

Figure 2.4 the effect of the importance weight. Reinforcement learning is applied to the two degree of freedom robot arm, which was introduced in section 1.4.2, to learn to move from $[x_1, x_2] = [0, 0]$ to $[x_1, x_2] = [\pi/2, \pi]$. This is repeated five times.

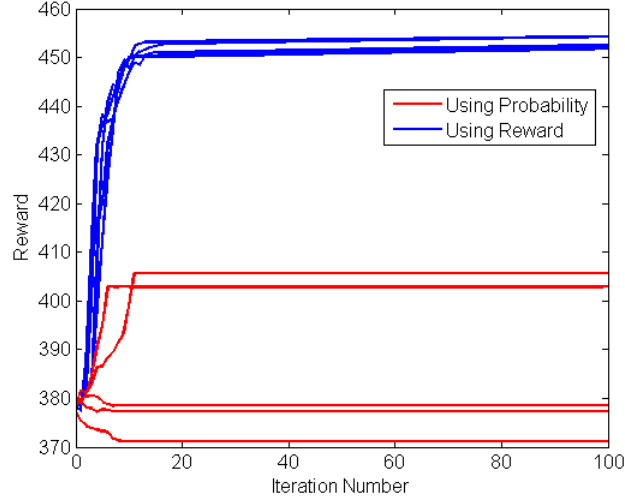


Figure 2.4: Reward found when reinforcement learning is applied on the two degree of freedom robot arm (see section 1.4.2) to learn to move $[x_1, x_2] = [0, 0]$ to $[x_1, x_2] = [\pi/2, \pi]$, repeated five times. The red line shows the result found using the importance weight based on probability. The blue line shows the result found using the importance weight based on reward. One can see that this yields a higher reward and a more robust result.

The red line shows the results that are obtained using the importance weight based on the probability, while the blue line shows the results that are obtained using the importance weight based on the reward. One can see that the reward is only decreased in some cases, due to the effect which was explained before. Next to this, the result is not very robust, since a different result is obtained in the different repetitions. Also, the red line yields a lower optimum.

The blue line shows the results yielding from the importance weight based on the reward. One can see that the behavior is robust, since approximately the same reward is obtained in each iteration. Also, this reward is higher than the reward obtained using the importance weight based on the probability.

REWARD FUNCTION

The reward is a very important aspect in a reinforcement learning algorithm, because it is the function which is optimized using the policy. This section introduces how the reward function is determined. It should be specified such that the policy is optimized for the correct variables. My aim is to find control policies yielding accurate and energy efficient trajectories for a robot arm. Next to this, the reward function should fulfill the requirement, which is set for reinforcement learning algorithms based on Expectation Maximization. This requirement states that the reward should always be positive [10] (see section 1.1).

The latter requirement can be met using a nonlinear transformation on the reward function, $u(r)$. This transformation function should not change the underlying problem. This means that the transformation is strictly monotonic with respect to the original reward function.

This transformation is applied to an untransformed reward function which is a cost function, which means that the best policy will obtain the lowest, possibly negative, reward. Then, equation 2.4 is applied to this reward function to obtain a reward function which is always positive and gives the highest reward for the optimal policy. This is the reward that is used to find the policy update, where ρ is a constant which is dependent on the value of r .

$$u(r) = \exp(-\rho r) \quad (2.4)$$

Still, the untransformed reward function should be designed. This reward function should be defined such that a low reward is given if the used energy is low and the accuracy is high. Three variables should be included in the reward function. The square of the torque is used instead of the energy, since the torque is directly dependent on the current in the motor [22]. Since the goal is to minimize the energy of the complete trajectory, the reward will be equal to the integral of the square of the torque over the full trajectory.

Next to this, the difference between the desired and actual end state should be as low as possible. Therefore, the difference between the goal position and the actual end position, and the speed at the end of the trajectory

will be inserted in the reward function. The speed is equal to the angular velocity present. The difference between the desired and actual end position, or the accuracy $h(x)$ of the end position, is determined using equation 2.5, where g_i denotes the desired end position of arm i and $x_i(T)$ denotes the actual end position at time step T .

$$h(x) = \left\| \begin{bmatrix} \left(\sum_{i=1}^2 \cos g_i \cdot l_i \right) \\ \left(\sum_{i=1}^2 \sin g_i \cdot l_i \right) \end{bmatrix} - \begin{bmatrix} \left(\sum_{i=1}^2 \cos x_i(T) \cdot l_i \right) \\ \left(\sum_{i=1}^2 \sin x_i(T) \cdot l_i \right) \end{bmatrix} \right\|_2 \quad (2.5)$$

This yields the reward function given in equation 2.6. $M(t)$ denotes the torque executed at time t . One can see that there are three weights, a , b and c . These weights will be tuned in chapter 3 such that the optimization is performed correctly.

$$r(x, u) = \exp - \rho \left(a \cdot h(x(T)) + b \cdot \dot{x}(T) + c + \sum_{t=0}^T M(t)^2 \right) \quad (2.6)$$

2.1.2. FINDING THE NUMBER OF SUBSPACES IN THE POLICY SPACE

An important step of Parameterized Skills is the analysis of the geometry and topology of the policy space. This is done to find the number of subspaces spanned by the optimal policy parameters that are found using reinforcement learning, since a separate model should be found for each subspace. This section will introduce ISOMAP, the algorithm which is used for this purpose, and how this algorithm is applied.

ISOMAP is a dimensionality reduction algorithm that was introduced by Tenenbaum *et al.* [26]. Its main purpose is to find lower dimensional structures in a high dimensional space. Meanwhile, it also analyses the number of connected elements of its input and defines to which element each point belongs. The number of connected elements is equal to the number of subspaces present in the high dimensional space. Therefore, this algorithm can be used for the second and third step of Parameterized Skills.

The input of ISOMAP is a matrix with entries equal to the distances between all points in the data set. Then, all points are connected to the points within a fixed radius or to a fixed number of nearest neighbors. These distances are saved and the distances to points further away are changed to a very large number. Then, the geodesic distances between all points are determined using a shortest path algorithm. This algorithm will use only the connections found in the first step. Multi-dimensional scaling is used to construct an embedding of this data in a d -dimensional Euclidean space [26].

Parameterized Skills requires this algorithm to find the number of subspaces spanned by the optimal policy parameters in the policy space. Therefore, only the first two steps of the algorithm are used. The Euclidean distances between the optimal policy parameters are fed to the algorithm. The first two steps of the algorithm are executed, so the shortest path algorithm is used to define the distances between all points. Two or more subspaces exists if it is not possible to find a path between two points, which are not nearest neighbors, that is shorter than the large value to which it was set in the first step.

2.1.3. IMPLEMENTATION OF SUPERVISED LEARNING IN PARAMETERIZED SKILLS

Parameterized Skills uses supervised learning to find the models that describe the parameters of the control policy as a function of the task. These models are found using the data obtained with reinforcement learning. A separate model is defined for each basis function and for each of the subspaces in the policy space.

This section introduces the implementation of the supervised learning algorithms. It will introduce the nonlinear function that is implemented in support vector regression and the neural network algorithm and the procedure that is followed to tune variables in polynomial regression and the neural network algorithm. These algorithms require tuning of variables separately for each model. This would be very time-consuming, since a model is created for each basis function and for each subspace. Therefore, a procedure is designed such that this is performed automatically.

SUPPORT VECTOR REGRESSION

Support vector regression (SVR) was used originally in Parameterized Skills by Da Silva *et al.*. This algorithm optimizes parameters such that a function is found that is as flat as possible and does not have outliers greater than a certain boundary. The kernel function should be chosen, as well as the variable C that defines the trade off between flatness and the outliers of the parameters and ϵ , the maximum value of the difference between the function output and the outliers.

It was decided to use a Gaussian kernel [15], since this type of kernel was also used by Da Silva *et al.*. Equation 2.7 defines the Gaussian kernel. This kernel function has one tuneable variable, the width of the kernel σ . This variable should be tuned based on the input data. When the width of the kernel is too small, the function will overfit the data. When the width of the kernel is chosen too high, the function will become almost linear, so it will not fit the data well.

$$k(x, x') = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad (2.7)$$

This yields that three variables should be tuned in SVR when a Gaussian kernel is used. σ , C and ϵ . These variables are tuned once for the problem. It is not necessary to tune the variables separately for each model that is constructed. Therefore, it is not necessary to define a special procedure.

POLYNOMIAL REGRESSION

Polynomial regression assumes that a polynomial function can fit the data, as was introduced in section 1.4.2. The variables of the polynomial function define the exact shape of the function. They can be found using least square fitting.

Polynomial regression has one unknown, namely the degree of the polynomial, k . This degree can be different for each basis function and it is cumbersome to find the appropriate degree by hand for the data sets of all parameters. This section introduces an approach to find the degree automatically.

This is done as follows. First, 80% of the input data is selected randomly. This data is used to create the polynomial model. Then, the remaining 20% of the data is used to verify if the model is correct. The verification is done using the coefficient of determination, or R^2 value, which is determined using equation 2.8, where \bar{y} denotes the mean of the output of the data set.

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2} \quad (2.8)$$

The coefficient of determination is a measure of how good the fit is. If all data points are matched exactly by the model, the value is equal to one. If the fit is really bad, the coefficient of determination can become as low as zero. This function does not account for overfitting, therefore the data is split up into training data and verification data. The latter is not used to construct the model.

An iterative process was used to obtain a model with a coefficient of determination with a floor value, which is tuned. So, the algorithm starts by fitting a polynomial model of degree one. Then, the degree is increased, up to a value of 10, until the coefficient of determination is as high as the floor. The tuning of this floor value is not cumbersome, since it is equal for all models.

NEURAL NETWORK ALGORITHM

An artificial neural network is an input-output system which is inspired by the biological process in the human neural network. An artificial neural network consists of three layers of neurons, the input layer, the hidden layer and the output layer, which are connected using weighted connections. This section introduces the procedure used to find the number of neurons in the hidden layer and the nonlinear function that is used in the hidden layer.

The number of input neurons is equal to the number of inputs to the function. This is equal to two, if only the goal position is varied, and four if the goal and start position are varied. The number of output layers was equal to one, since a model is constructed for each parameter separately. The number of hidden neurons is a free parameter, which should be tuned separately for each model that is constructed. This is done using a procedure similar to the one used in polynomial regression.

The data is split up in training data and verification data. The neural network starts training with a low number of hidden neurons. This number is increased and the training is repeated until the coefficient of determination reaches its floor value. A maximum number of hidden layers was defined to prevent the algorithm from ending in an infinite loop. The floor value should be defined, but this number is equal for all models that will be created.

The nonlinear function used in the hidden layer should also be defined. It was chosen to use a so-called tan-sigmoid transfer function, which is the default in MATLAB. An example of such a function is given in equation 2.9, where n denotes the input to the hidden layer and a the output.

$$a = \frac{2}{1 + \exp - 2n} - 1 \quad (2.9)$$

LOCALLY WEIGHTED LINEAR REGRESSION

Locally weighted linear regression is a supervised learning algorithm that constructs a local linear model around the current target. It uses a specific number of nearest neighbor around the target. This number should be tuned. This is an integer, which yields that the tuning is simple. Therefore, no special procedure should be defined to tune this variable.

2.2. IMPLEMENTATION OF THE CONTROL POLICY

The control policy is represented using a dynamic movement primitive (DMP), as was presented in section 1.4.2. A DMP consists of two systems, an attractor system which defines the movement from the start position to the goal position and a nonlinear system, which adds a forcing term to the trajectory provided by the attractor system. Both systems should be implemented such that they behave as expected.

The attractor system should yield a stable motion. The nonlinear system should be tuned such that each basis function adds a forcing term to a specific part of the trajectory. This section presents how several variables of the DMP are tuned such that its behavior is as expected, starting with the attractor system in section 2.2.1, followed by the nonlinear system in section 2.2.2.

2.2.1. ATTRACTOR SYSTEM

The attractor system is similar to a PD controller. The difference is that the forcing term $f(s)$ is added to the PD equation. There are two parameters that should be tuned in this system, the position gain K , and the velocity gain Q . It is not required to change the value of these parameters when a slight change is applied to the environment, which would be required for a PD controller.

It was given by Ijspeert *et al.* [27] that for a stable motion, $K = \frac{1}{4}Q^2$ should be used. Next to this, the gains should be set such that the forcing term can influence the motion significantly. Therefore, Q was set to a low value. This will allow the PD part of the attractor system to yield results in the same order of magnitude as the forcing term.

2.2.2. NONLINEAR SYSTEM

The nonlinear system defines the forcing term. The forcing term is equal to the sum of the nonlinear basis functions, which are activated depending on the phase s . The phase is a time-dependent variable which monotonically decreases from 1 to 0 during the execution of the movement. It is computed by integration $\kappa \dot{s} = -\alpha s$, where α is a factor which depicts how fast the phase decreases. The level of activation is dependent on the height and centers of the basis functions and the value of the parameters w_i , which are learned using reinforcement learning.

Several variables of the nonlinear system should be set, namely h_i , the heights of the basis functions, c_i , the centers of the basis functions and α , the factor which determines the speed of the decrease of the phase variable. These variables should be set such that the reinforcement learning algorithm can improve the performance on the controller by changing the parameters. Therefore, center and height of the basis functions are set such that all basis functions are activated at a distinct location in time, with overlap only at the edges of the basis functions.

The centers of the basis functions denote the location where the activation of the basis function is highest. These centers are defined as a function of the phase. It is desired to spread the basis functions evenly over time, while the centers should be found with respect to the phase variable s . The procedure given by DeWolf [28] is used to find the locations of the centers in terms of the phase such that the basis functions are spread evenly over time.

First, the locations in time are determined which yield an even spread of the trajectory over time. For example, when ten basis functions are used, it is determined that there should be 44.4 time steps between two centers, $c_i(t)$, if 400 time steps are used. These indices are inserted in equation 2.10 to find the location of the centers in terms of the phase, $c_i(s)$. Note that this equation is obtained by integrating the phase equation $\kappa \dot{s} = -\alpha s$.

$$c_i(s) = \exp\left(-\frac{\alpha}{\kappa} c_i(t)\right) \quad (2.10)$$

Next, the heights of the basis functions should be determined. These are also dependent on the phase of the trajectory. Therefore, these heights must be defined in such a way that the activation will decrease to zero

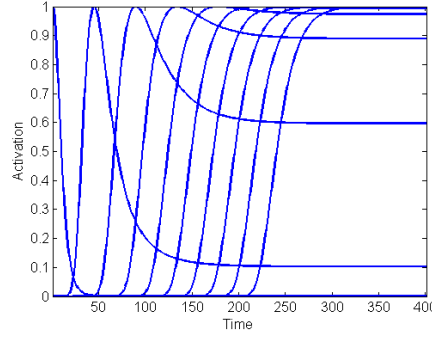
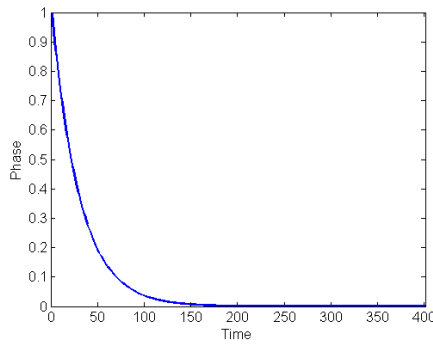
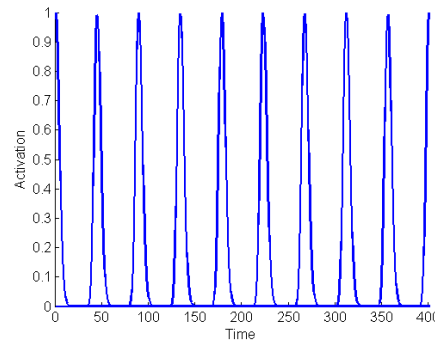


Figure 2.5: An example of a wrong value for the height. One can see that the activation of almost all basis function does not decrease to zero. Therefore, they will have effect on a large part of the trajectory and learning will be difficult.



(a)



(b)

Figure 2.6: Figure (a) denotes the phase variable as a function of time. Figure (b) denotes an example of a correct choice of the height and the centers. One can see that they are spread evenly over time and their activation has a similar maximum.

over time. Otherwise, an increasing amount of basis functions is active over time. This would make learning of the parameters very difficult.

Figure 2.5 illustrates the effect of a bad value for the heights. One can see that almost all basis function are active at the end of the trajectory, which means that many parameters corresponding to these basis functions affect this part. Then, it is very difficult to learn the parameters correctly. Empirically, it was found that for heights around $h_i(s) = 4N/c_i(s)^2$ the heights of the basis function would have the desired shape.

The factor α should allow the phase to decrease to a value very close to zero during the trajectory. Therefore, α is dependent on the number of time steps of the trajectory. The value of α also influences the maximum value of the activation. Therefore, a value was chosen which yielded that the maximum activation of all basis functions was similar, while the phase at the end of the trajectory was very close to zero. This is supported with figure 2.6. Figure (a) shows the phase variable as a function over time. Figure (b) shows the corresponding basis functions. The variables are now tuned correctly, such that the activation decreases to zero for each basis function.

2.3. SUMMARY

This chapter introduces the implementation of the algorithms presented in chapter 1. The implementation of the policy representation was presented, as well as the implementation of PoWER, the reinforcement learning algorithm that was used, the algorithm used to analyze the geometry and topology of the parameters in the policy space and the implementation of the different supervised learning algorithms.

PoWER is the reinforcement learning algorithm that finds optimal policy parameters for training goals. Three alterations were made to the algorithm described by Kober and Peters [8]. Also, the reward function should be implemented such that the algorithm will optimize for a trajectory which is energy efficient and accurate.

The exploration was performed such that a specific value was determined for a sampled trajectory, instead of for each time step. Also, the update rule would not take into account the exploration of the trajectory when this trajectory was sampled under a previous policy. Instead, the difference between the policy parameters used for a specific trajectory and the current mean trajectory was taken into account for the update rule. The final alternation was applied to the importance sampling. The importance weight was determined as the ratio of the reward between the current mean policy and the reward of the sampled trajectory, instead of the likelihood of occurrence under the current policy.

ISOMAP is used to analyze the geometry and topology of optimal policies in the policy space. The first and second step of the algorithm are used. These steps define the number of connected elements of a matrix consisting of the Euclidean distances between the data points in a data set. Also, it is given to which element each data point belongs.

Finally, the implementation of the supervised learning methods was introduced. SVR and the neural network algorithm required choice of a nonlinear function. All algorithms required tuning of certain variables.

SVR uses a Gaussian kernel, as was done by [Da Silva *et al.*](#). The neural network used a tan-sigmoid transfer function in its hidden layer. Polynomial regression and the neural network algorithm required tuning of a variable specific for each model. This was done by increasing the value of this variable until the coefficient of determination was at least equal to a floor value. Locally weighted linear regression and SVR required tuning of parameters given the problem, which will be done after a data set is obtained.

A dynamic movement primitive (DMP) is used to describe the control policy. The gains of the attractor system are tuned such that they yield a stable motion. Next to this, the phase variable and the heights and centers of the basis functions should be chosen such that each basis function has an effect on a specific part of the trajectory.

3

VERIFICATION OF THE IMPLEMENTATION OF PARAMETERIZED SKILLS

The goal of this thesis is to improve the performance of Parameterized Skills, using one of the supervised learning algorithms introduced in chapter 1. Chapter 2 presented the implementation of the reinforcement learning algorithm and of the supervised learning algorithms in Parameterized Skills. This chapter will verify if these algorithms are implemented correctly. A correct version of the algorithm used to analyze the geometry and topology of the policy space (see section 2.1.2) is made available by [Tenenbaum *et al.*](#), therefore it is not necessary to verify this algorithm.

Four experiments are used to verify the reinforcement learning and supervised learning algorithms. Figure 3.1 presents an overview of all experiments that are conducted in this study. The colored text indicates the application of the experiment. Experiment 1 to 4 are used for the verification, which is clear by the red color of the text.

It is clear from figure 3.1 that three things should be verified for the reinforcement learning algorithm. It should be verified if the algorithm actually yields learning of the control policy. Next, it should be verified that the correct thing is learned using the reward function given in chapter 2. Finally, it should be verified that a good results is learned.

For the supervised learning algorithms, it should be verified that the algorithms are able to construct a model. This will be verified using a data from a trial function, since data of the type which will be used during the experiments is not available. Also, the time required to construct the models using the different supervised learning algorithms is compared, to find if polynomial regression, the neural network algorithm and locally weighted linear regression are faster than support vector regression (SVR).

Section 3.1 of this chapter presents the experiments that were used to verify PoWER. Also, the results and analysis of these experiments are given in this section. Next, in section 3.2 the experiments are presented that are performed to verify the supervised learning part of Parameterized Skills. Finally section 3.3 summarizes this chapter.

3.1. VERIFICATION OF REINFORCEMENT LEARNING ALGORITHM

Three steps are conducted to verify that the implementation of the reinforcement learning algorithm is correct. First, it is checked if the reinforcement learning algorithm is able to learn on a simple problem. The second experiment is conducted to verify that the algorithm optimizes the correct variables. This means that the policy that optimizes the reward function yields a trajectory that is accurate, while minimizing energy. Third, it was verified that the a good control policy is learned. This was done by tuning several variables of the reinforcement learning algorithm such that the reward is maximized.

This section will start with an introduction of the simple problem which is used for the first two experiments. Then, the experimental set up and the results of the experiments are given. These results will show that the algorithm learns, that the algorithm optimizes the correct parameters and that the policy found by the algorithm yields an optimal reward.

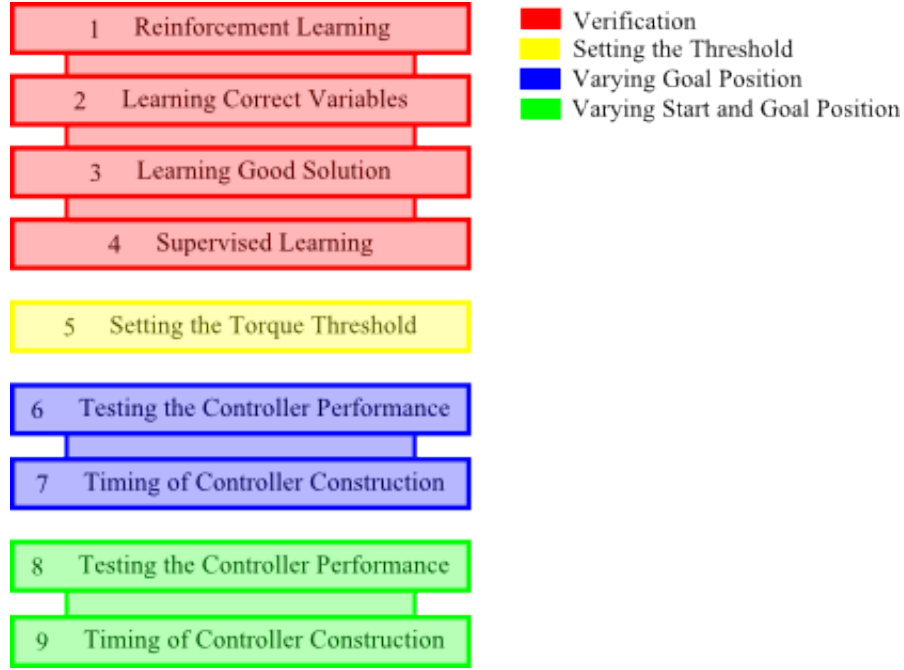


Figure 3.1: Overview of all experiments conducted in this thesis. This chapter will introduce the set up and results of experiments 1 to 4. These experiments are used to verify the reinforcement learning algorithm (experiments 1 to 3) and the supervised learning algorithm (experiment 4).

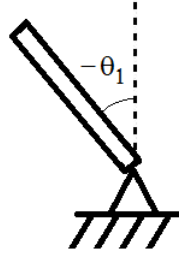


Figure 3.2: Drawing of the simple problem

3.1.1. SIMPLE PROBLEM USED IN EXPERIMENTS

The simple problem is designed to verify if the reinforcement learning algorithm is correct. This problem should be simple, such that the relation between the input and the output is clear. A one degree of freedom robot arm was used for this purpose. This robot arm has the same dimensions and mass as the arm 1 of the two degree of freedom robot arm, which was presented in section 1.4.2. This yields the problem shown in figure 3.2. This robot arm rotates around the joint connecting the arm to the ground.

3.1.2. EXPERIMENT 1: VERIFICATION THAT THE ALGORITHM LEARNS

The first experiment uses the simple arm to verify that the implementation of the reinforcement learning algorithm is correct. This is verified using an experiment for which the result is known. The result found in the experiment should match this expected result. Experiment 1 aims to find the policy that minimizes the required energy. The optimal results would be a policy where the forcing term counteracts the acceleration found by the attractor system. This means that a reward function should be minimize the energy. Then, the optimal result as a function of this reward function should be defined.

Minimizing the energy is the same as minimizing the square of the torque, as was explained in section 2.1.1. This yields the reward function given in equation 3.1, where M denotes the executed torque. Note that

the transformation was performed to obtain a reward function that is always positive.

$$r(x, u) = \sum_{t=0}^T \exp(-\rho M(t)^2) \quad (3.1)$$

This reward function should yield a trajectory that does not use energy. This means that the used torque should be equal to zero. When the torque is zero everywhere, this means that the reward after each timestep will be equal to one. The complete reward should be equal to the number of steps taken, so 400.

This experiment was conducted several times with a different number of basis functions. The simplest problem only uses one basis function. Then, the search is conducted in a one-dimensional space, so it is a simple line search and it should be easy to find the optimal policy. Tests were also performed with two, five and ten basis functions to see if there is an effect on the quality of the result when the dimension of the parameter space is increased.

RESULTS OF EXPERIMENT 1

Figure 3.3 shows the results of experiment 1 for one, two, five and ten basis functions, respectively. The goal of the experiment is to find the policy which yields a reward of 400, which means that no energy is used during the complete trajectory. One can see that the the problem with one and two basis functions yield the correct result with the maximum possible reward.

However, the solution is not yet optimal for the case when five or ten basis function are used. One can see that the reward still increases, so it is expected that eventually the optimal result will be found. This is harder when a larger number of basis functions is used, since the problem is more complex. It was found that the algorithm requires over 1000 iterations to converge using five and ten basis functions.

3.1.3. EXPERIMENT 2: VERIFICATION THAT THE ALGORITHM OPTIMIZES FOR THE CORRECT VARIABLES

Experiment 2 is conducted to verify that the algorithm finds a trajectory that optimizes for energy efficiency and accuracy. The goal of this experiment is to move the simple robot arm 180° while using minimum energy. This means that the reward function should be defined such that the policy will be optimized for both energy and accuracy.

The reward function will be defined as given in section 2.1.1 (see equation 3.2). This function is build up of immediate reward and endpoint reward. The immediate reward is given on the torque, to minimize the energy. The endpoint reward will only be given after the final time step of the trajectory and is dependent on the accuracy $h(x)$ of the actual end position with respect to the desired end position.

$$r(x, u) = \exp - \rho \left(a \cdot h(x(T)) + b \cdot \dot{x}(T) + c + \sum_{t=0}^T M(t)^2 \right) \quad (3.2)$$

Three weights should be tuned in the reward function such that the reward function optimizes correctly, namely a , b and c given in equation 3.1. It might be possible that the function becomes non-convex if the reward function is dependent on more than one variable. This means that there can be several local maxima in the function. Then, it is possible that the reward function converges but this solution does not yield a trajectory that was desired.

For example, if the reward on the energy would be too high, the algorithm might still find a policy that does not yield any movement, since the optimal result is a policy that applies a forcing term to counteract the attractor of the dynamic movement primitive (DMP), such that no energy is used. The algorithm will receive a penalty on the accuracy, but this is not significant with respect to the high reward given for the energy.

This experiment was performed several times while varying the number of basis functions. It is expected that more than one parameter was required to solve a problem which aims to optimize two variables.

RESULTS OF EXPERIMENT 2

Figure 3.4 shows the results for the experiments where the reward function was dependent on the energy and the final position. The weights a , b and c were equal to 200, 100 and -50 , respectively. One can see that for one and two basis functions, the algorithm is not able to find a correct solution. Instead, it finds a solution where

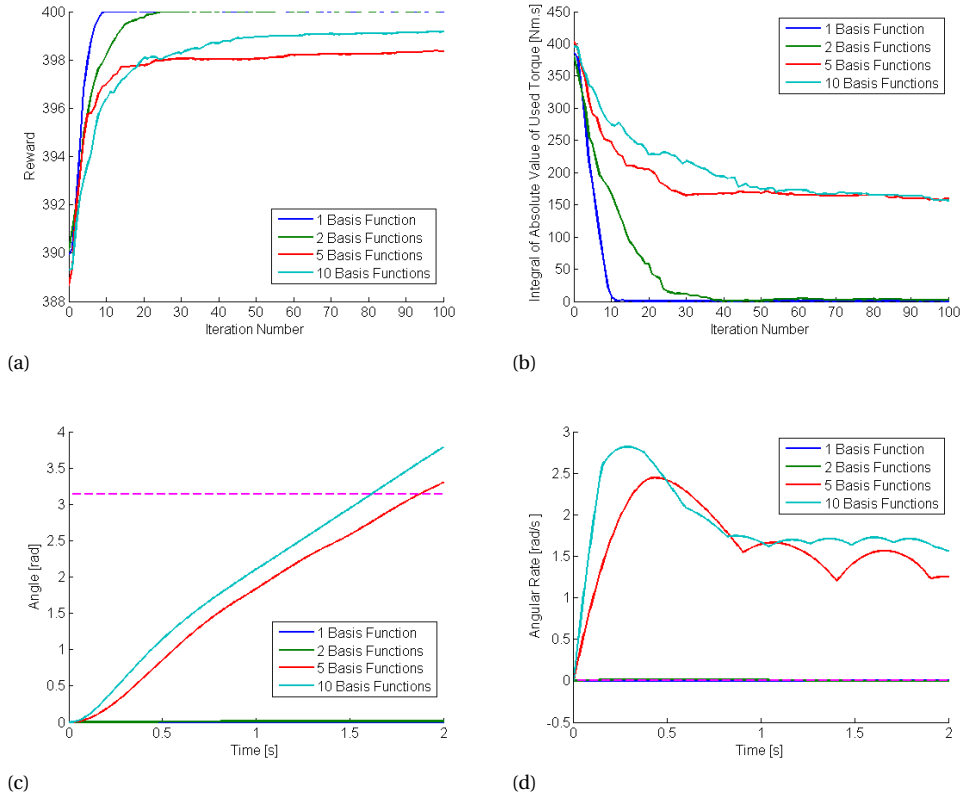


Figure 3.3: Results of experiment 1, showing that the reinforcement learning algorithm is able to learn. The goal of this experiment is to find the policy that does not require any energy. (a) shows the reward found using one, two, five and ten basis functions, (b) shows the integral of the torque used during each iteration, (c) shows the trajectory of the optimal policy and (d) shows the speed of the trajectory of the optimal policy. One can see that the correct result is obtained when one or two basis functions are used. One can see that the problem is more difficult with a higher number of basis function, since a higher number of iterations is required with 2 basis functions. The results have not yet converged for the case using five or ten basis functions, due to the increased complexity of the problem. More than 1000 iterations are required for the algorithm to converge.

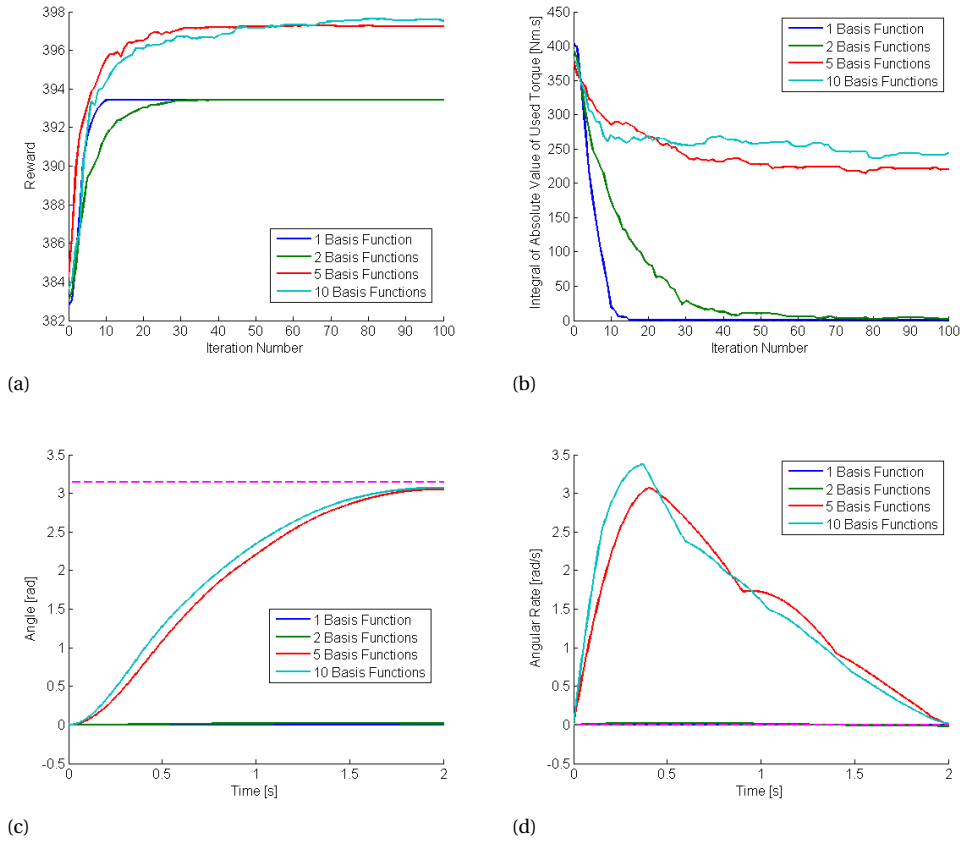


Figure 3.4: Results of using PoWER to find the policy which minimizes energy and maximizes accuracy for one, two, five and ten basis functions. (a) shows the rewards, (b) shows the torques over the iterations, (c) shows the trajectory of the optimal policy and (d) shows the speed of the trajectory of the optimal policy. One can see that when one or two basis functions are used, the algorithm finds a local minimum and is not able to optimize for both the required energy and the accuracy. The policy found with five or ten basis functions yields a solution which is optimal on the required energy and the accuracy.

the used energy is minimized only. Apparently, the forcing term requires a shape which cannot be achieved when only one or two basis function are used.

The algorithm is able to find a correct solution if the number of basis functions is increased. Then, it is possible to find the correct shape of the forcing term since the forcing term can have more variation. One can see that when there are five or ten basis functions used, the algorithm is able to decrease the torque while being accurate, so the difference between the actual and desired end position is small.

3.1.4. EXPERIMENT 3: VERIFICATION THAT THE RESULT IS OPTIMAL

It is now verified that the implementation of the reinforcement learning enables learning towards a trajectory that optimizes for the correct variables, namely accuracy and energy efficiency. Still, there are several variables of the reinforcement learning algorithm that should be tuned such that a good solution is found by the algorithm, meaning that the reward is maximized. Experiment 3 is used to tune these variables.

This experiment is conducted on the problem presented in section 1.4.2, since this problem will be used in all further experiments, which are used to answer the research question. I aim to find the optimal parameters on this problem, therefore the algorithm will be tuned for this problem.

Due to time limits, it was decided to run the algorithm for 800 iterations. This means that I aim to find variables out of a set that yield the best result after 800 iterations. It might be possible that a different number of iterations yields a different value for the variables. The values that are tried during tuning were chosen based on previous experience with the algorithm, which showed that values in the chosen range yielded good results.

The variables that are tuned are the standard deviation of the stochastic policy, which denotes the exploration of the algorithm, the dimension of the parameter space where the algorithm searches for an optimal solution,

the minimum importance weight and the number of trajectories that are sampled each learning iteration. The tuning was performed by trying several values for these variables. The value yielding the highest reward was chosen. This was repeated ten times, to account for variations in the results due to the stochasticity of reinforcement learning.

First, experiment 3a is conducted to find the standard deviation of the exploration which yielded the highest reward. The exploration will be farther away from the current mean policy if the standard deviation is higher. This is advantageous because then the algorithm will find a result quicker. However, when the policy is updated with large steps, the algorithm might step over a maximum in the policy space, so the quality of the result can be lower.

Therefore, it is necessary to find a value for the standard deviation such that the exploration yields the highest reward. It was decided to have a standard deviation of at least $\sigma = 3$ for the first fifty iterations for all of the tests to speed up the learning when a low standard deviation was used.

Secondly, the dimension of the parameter space, or the number of basis function, is determined using experiment 3b. This number denotes the number of locations where a forcing term can be applied to the DMP (see section 2.2). When this dimension is too low, the algorithm will not find the best solution, since there are not enough possibilities to add a forcing term to improve the trajectory. When the dimension is too high, the search space of the reinforcement learning algorithm will be very large and there is a high chance that the algorithm will settle in a local minimum because of the complexity of such a high-dimensional space. This was already seen in the simplest experiment on the simple problem (see section 3.1.2, results of experiment 1).

Experiment 3c tunes the minimum value of the importance weight ω . Any trajectory will be deleted if its importance weight is lower than this minimum. In this way, 'bad' trajectories will not influence the result in a negative way. However, care should be taken that this threshold is not too high. This might yield that the number of samples used in the update is too small, which can result in a bias. The maximum value for which was tested was equal to one. At this value all trajectories which are used in the update, have a state-action value which is at least equal to the state-action value of the current policy.

The variable that is tuned in experiment 3d is the number of new trajectories that are sampled each iteration. A high number of new trajectories yields a lower bias in the policy update, but this can increase the time required to find the optimal solution. Therefore, I aim to find the lowest number of samples which yields a good result.

RESULT OF EXPERIMENT 3

First, the standard deviation of the exploration was tuned using experiment 3a. The exploration is a random Gaussian process, so the standard deviation is a measure of the difference between the exploratory policies and the current mean policy.

Figure 3.5 (a) shows the results of experiment 3a, where the reward is plotted against the number of iterations. The number of reinforcement learning iterations performed was defined based on time limits. The first ten iterations were performed with a standard deviation of at least 3 to speed up learning. One can see that, when the standard deviation during the remainder of the learning process is equal to 0.1, the result yielding the highest reward was found.

Figure 3.5 (b) shows the result of experiment 3b. The dimension of the parameters space, or the number of basis functions used is tuned in this experiment. One can see that the highest reward is obtained with five basis functions.

Thirdly, experiment 3c was conducted to find the importance weight ω , which denotes how many previous trajectories are used in the current update. One can see in figure 3.5 (c) that the highest reward is obtained when the importance weight is equal to one, which means that all trajectories that yielded a lower reward than the reward of the current policy are discarded.

Finally, in figure 3.5 (d) the results of experiment 3d are shown. This experiment was performed to tune the number of new trajectories sampled during each iteration. One can see that the reward is highest when 200 new trajectories are sampled each iteration. One can also see that when 200 new trajectories are sampled, the algorithm converges to an optimal solution already after around 500 iterations. Therefore, in the next tests, the number of iterations used is decreased.

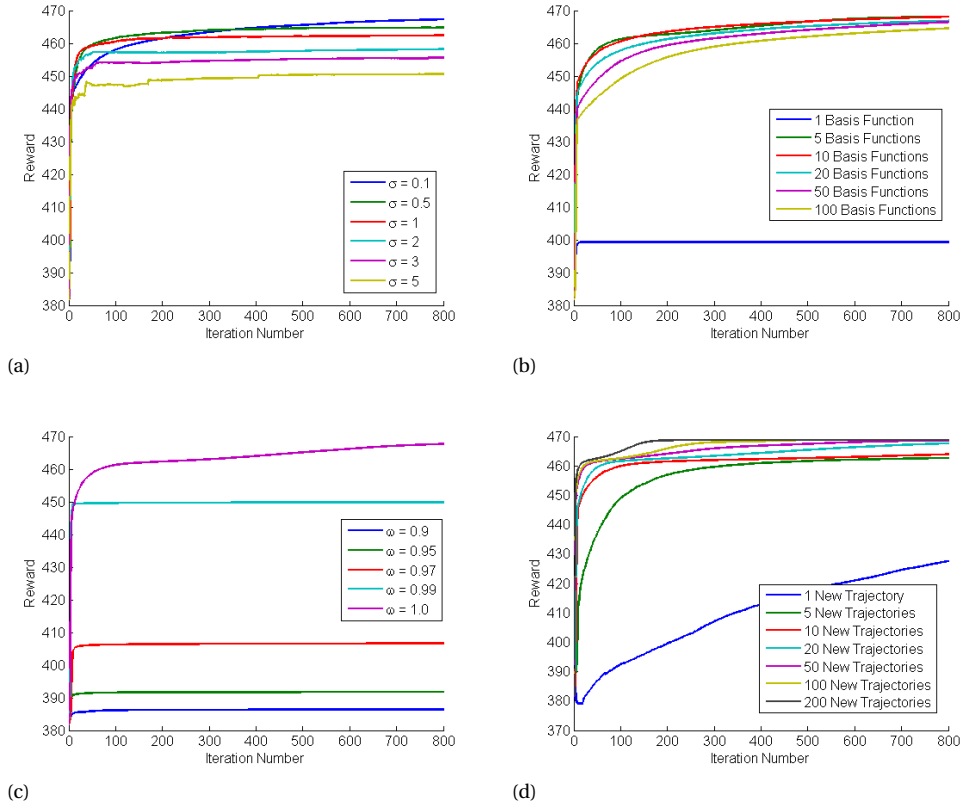


Figure 3.5: Results of the experiment 3, used to tune the algorithm. Figure (a) denotes the results of the experiment with a varying standard deviation, figure (b) denotes the results of the experiment on the dimension of the parameter space, figure (c) shows the result of the experiment on the minimum importance weight and figure (d) shows the result of the experiment on the number of trajectories sampled each iteration. One can see that a standard deviation should be 0.1, the dimension of the parameters space should be 5, the minimum importance weight should be 1 and the number of new trajectories should be 200 to obtain the best result. Then, only 500 iterations are required for the algorithm to converge.

Table 3.1: Coefficients of determination for the supervised learning algorithms. Polynomial regression finds the best fit, which is expected. Locally weighted linear regression performs second best, followed by SVR. The neural network algorithm shows the worst performance.

Algorithm	Coefficient of Determination
Polynomial Regression	1
SVR	0.9876
Neural Network Algorithm	0.9789
Locally Weighted Linear Regression	0.9989

3.2. VERIFICATION OF THE SUPERVISED LEARNING ALGORITHMS

This section introduces the experiment that is conducted to verify if the implementations of the supervised learning algorithms are correct. This experiment is performed for two reasons. It is checked if the implementation of the algorithms is correct and it is checked if polynomial regression, the neural network algorithm and locally weighted linear regression are faster than SVR.

The trial function requires different values for the tuneable variables in the supervised learning algorithms than the data set obtained with reinforcement learning. Therefore, it cannot be verified if it is possible to create a model for the actual data set. It is only verified that the implementation is correct.

3.2.1. EXPERIMENT 4: VERIFICATION OF SUPERVISED LEARNING METHODS

Experiment 4 uses a data set obtained from a trial function to verify the supervised learning algorithms, since actual data from reinforcement learning is not yet available. It was chosen to use a third order polynomial with Gaussian noise, since it is also unknown what shape the data will have. However, it is expected that the data is noisy, due to the stochasticity of reinforcement learning. Therefore, Gaussian noise is added. Note that this function is only used to check if the supervised learning algorithm can create a model and does not represent the actual data in any way.

Equation 3.3 denotes the trial function used in this experiment. 200 random data points were constructed using this trial function. The output was determined for these random inputs, with added Gaussian noise with a standard deviation of 5.

$$f(x, y) = x^3 + 3y^2 - 2x^2 + 4x + 2y \quad (3.3)$$

3.2.2. RESULTS OF EXPERIMENT 4

Figure 3.6 shows the models found using the data set obtained using equation 3.3 with added Gaussian noise. One can see that all methods find a function which matches the original trial function well, meaning that their implementation is correct. The polynomial regression algorithm found the best match. This is expected since the trial function is also a polynomial function. One can also see that locally weighted linear regression is influenced most by the noise. This is expected, since the model only uses nearest neighbors and the noise added locally is not necessarily spread evenly. The other algorithms use the complete data set and the noise is spread more even over this data.

Table 3.1 shows the coefficient of determination for each of the models. This coefficient is equal to one if the model fits the data very well and decreases to zero when the fit is not good. This table supports the conclusion that the polynomial model finds the best fit.

Next to this, the time required to construct the models was recorded to verify that polynomial regression, the neural network algorithm and locally weighted linear regression are faster than SVR. Table 3.2 shows the time required to construct the model for the polynomial model, the neural network and the SVR model. Locally weighted linear regression does not construct a model, therefore the time required to find the values of the data points used to create figure 3.6 (e) was saved.

One can see that SVR is significantly slower the other algorithms. Polynomial regression is fastest, followed by locally weighted linear regression. Note that this time is dependent on the size of the mesh used to create this plot since it is the time required to calculate these outputs. The recorded time for the other models did not take into account the time required to determine the outputs of the model for the mesh.

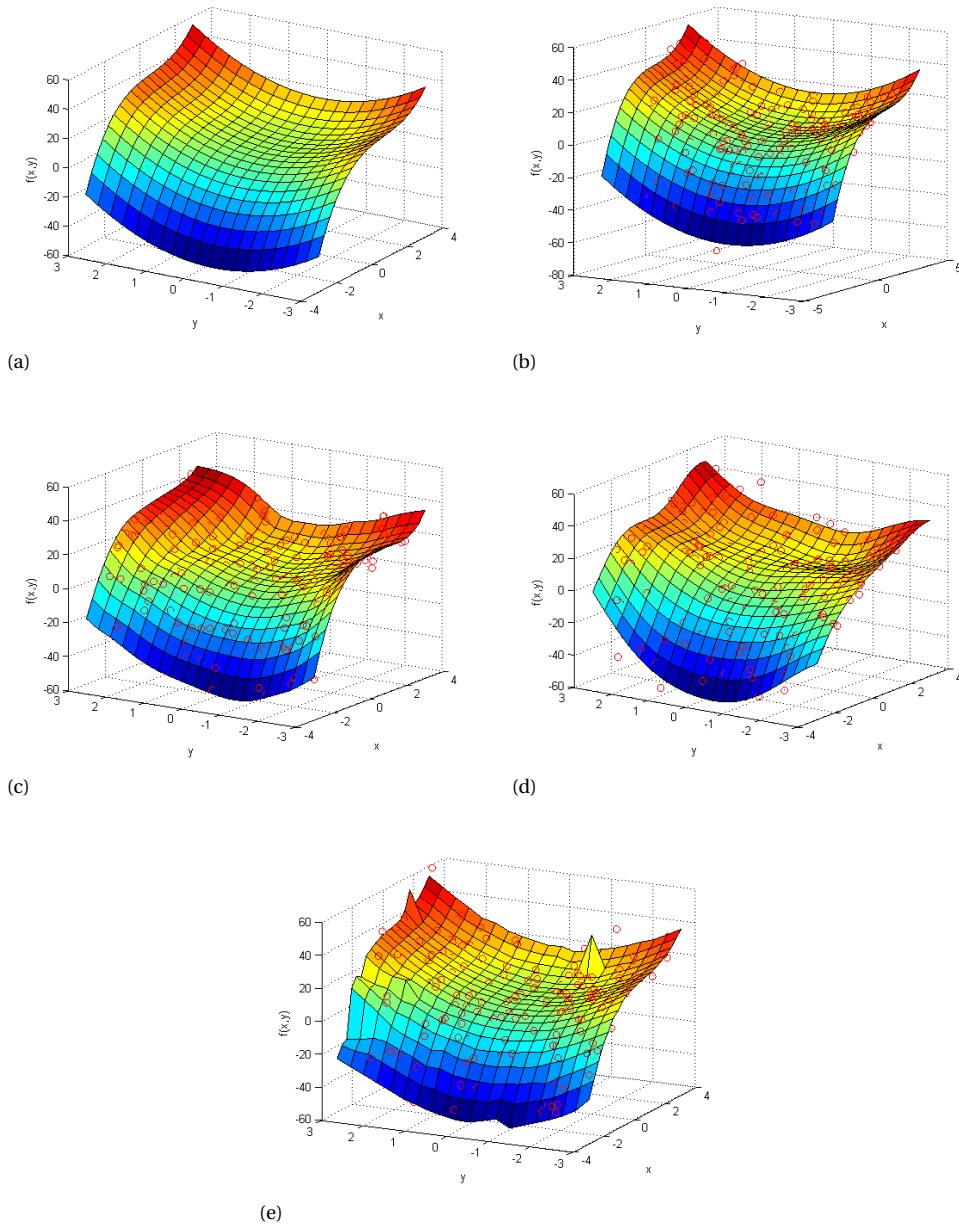


Figure 3.6: Testing the implemented supervised learning algorithms. Figure (a) shows the original function, (b) to (e) show the resulting function found with supervised learning, where the red circles show the test data. (b) is found using polynomial regression, (c) using a neural network algorithm, (d) using SVR and (e) a mesh plot of the solutions found with locally linear regression

Table 3.2: Overview of Elapsed Time during the construction of the supervised learning models. For the outputs found using locally weighted linear regression, the process of finding output values for all test inputs was timed, since an actual model is not constructed. One can see that polynomial regression is fastest, followed by locally weighted linear regression. The SVR model is significantly slower than all other methods.

Algorithm	Elapsed time [s]
Polynomial Regression	$9.134 \cdot 10^{-2}$
Neural Network Algorithm	1.666
SVR	110.6
Locally Weighted Linear Regression	0.2472

3.3. SUMMARY

This chapter verifies that the implementation of Parameterized Skills is correct. This is done for the reinforcement learning and supervised learning algorithms. It is not required to verify the correctness of ISOMAP, since a correct version is available.

Four experiments were conducted in this verification. First, experiments were performed to verify the implementation of the reinforcement learning algorithm. Next, the supervised learning algorithms were tuned using a trial function.

Three things were verified for the reinforcement learning algorithm. First, it was verified that the algorithm yielded learning of the control policy. Next, it was verified that the reward function and algorithm were implemented such that the learning yielded a solution that was optimized for the correct variables. Finally, several variables were tuned to ensure that the solution was the optimal solution possible.

Next, the implementation of the supervised learning algorithms was verified using a third order polynomial with added Gaussian noise. The output found using the different supervised learning algorithms was compared to the original output. All methods were able to find a model which matched the original well. It was clear that locally weighted linear regression was most sensitive to the noise. It was also verified that polynomial regression, the neural network algorithm and locally weighted linear regression are faster than SVR.

4

COMPARISON OF SUPERVISED LEARNING ALGORITHMS ON A TASK WITH TWO DIMENSIONS

The previous chapters provided background information necessary to understand the research performed in this thesis. First, the algorithms that will be used were presented theoretically in chapter 1. Then, it was explained how they are implemented in chapter 2 and it was verified that the implementations were correct in chapter 3. This chapter is the first chapter which provides information that is used to answer the research question.

In this chapter, two issues that exist when applying Parameterized Skills to train a controller of a robot arm are analyzed. These issues are the following: the control policy that was found in a previous application of Parameterized Skills did not meet the threshold set on the performance and much time was required to construct the controller using support vector regression (SVR). SVR is a slow method since it requires an optimization algorithm and aims to find a vector with the length equal to the size of the data set.

My aim is to solve these two problems by implementing another supervised learning algorithm. Four supervised learning algorithms are implemented and compared to find an algorithm that performs better than SVR. If similar performance is found, the faster algorithm will be selected.

This chapter will use experiments 5, 6 and 7 of the overview presented in figure 4.1 to compare the supervised learning algorithms. Experiment 5 is used to set a threshold on the accuracy and required energy. Experiment 6 is used to compare the performance of the models found using the supervised learning algorithms. This experiment is also used to find the minimum size of the data set required for the supervised learning algorithms. Experiment 7 compares the time required by the algorithms to construct the model.

This chapter will first introduce how the threshold was set on the required energy using experiment 5. This threshold is used in experiment 6. Section 4.2 will present the set up of experiment 6 and presents and analyses the results of this experiment. These results will be compared against the results found by [Da Silva et al.](#), striking results are analyzed. Also, the minimum data set size required for each supervised learning algorithm is found and the performance of the different algorithms is compared. Then, section 4.3 presents the results of experiment 7, which are used to find the time required for each supervised learning algorithm. Finally, this chapter is summarized in section 4.4.

4.1. THRESHOLD ON ACCURACY AND ENERGY EFFICIENCY

A threshold should be set for the accuracy and energy efficiency to define when the performance is good. The threshold on the accuracy is set to a fixed value, 2 cm. This value is based on the results found during the verification phase and previous experience on the robot arm. The required energy is dependent on the square of the used torque. Therefore, a threshold was set on the used torque. This threshold is dependent on the path length of the trajectory. This dependency is found using experiment 5. First, the set up this experiment is given, after which the results are presented and analyzed and the threshold is set.

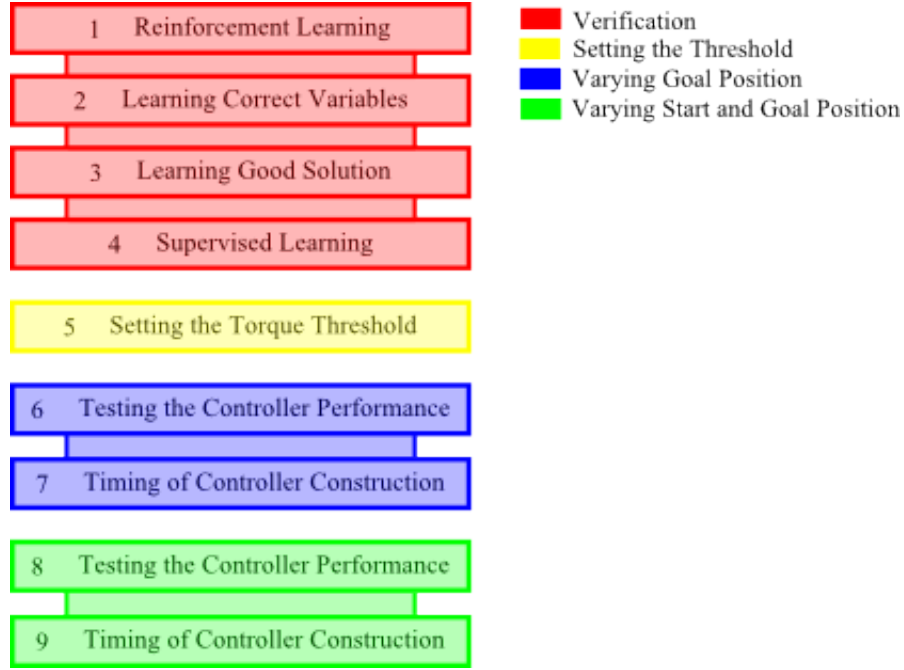


Figure 4.1: Overview of all experiments conducted in this thesis. This chapter will introduce the set up and results of experiments 5, 6 and 7.

4.1.1. SET UP OF EXPERIMENT 5

This experiment is conducted to find a threshold for the performance of the supervised learning algorithms in terms of energy efficiency. The energy is directly dependent on the square of the torque. Therefore, it is decided to set a threshold on the integral of the absolute torque used during a trajectory. The used torque is dependent on the length of the path of the robot arm. This experiment is conducted to find this dependency.

A data set is generated of hundred training goals with a fixed starting position. This data set consists of the goals and optimal parameters for these goals that are learned using reinforcement learning. The path length of the trajectories is determined, as well as the used torque yielding from optimal parameters. Then, a first order polynomial is fitted through this data. This function defines the relation between the torque and the path length.

It is expected that the relation between the used torque and the path length will be not be smooth. Therefore, the threshold will be equal to the sum of the value yielding from the first order polynomial and the standard deviation of the difference between this value and the torque yielding from the optimal parameters.

4.1.2. RESULTS AND ANALYSIS OF EXPERIMENT 5

Figure 4.2 shows the integral of the used torque as a function of the length of the path of the robot for each training goal. One can see that there is a relation between the length of the path and the used torque. The relation is described using a first order polynomial. This polynomial is shown using the red line in figure 4.2.

This threshold is not met by the all optimal trajectories that were found for the training goals, since the relation is not smooth. Therefore, it is not expected that a model constructed using these training goals will meet this threshold. Thus, the threshold on the used torque will be set slightly higher. The standard deviation of the difference between the polynomial and the used torque yielding from the optimal parameters is determined. This value is added to the polynomial. This yields the threshold shown by the pink line in figure 4.2.

4.2. ANALYSIS OF PERFORMANCE OF SUPERVISED LEARNING ALGORITHMS

This section analyses the performance of the supervised learning algorithms. This is done to find if any of the algorithms finds a controller that yields performance that meets the threshold, to find the minimum size of the data set required for each algorithm and to find the algorithm that yields the best performance. So, this section analyses the first issue that was found when Parameterized Skills is applied to train a controller of a robot arm.

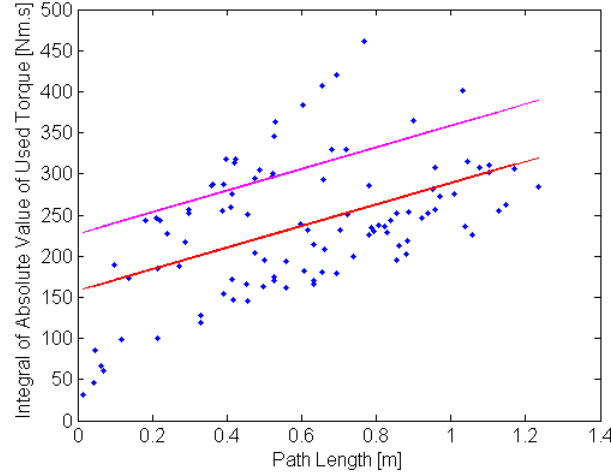


Figure 4.2: Plot of the integral of the used torque versus the path length. The red line shows the linear function fitted through this data. The pink line shows the final torque threshold, where the standard deviation of the noise between the data and the linear function is added to the linear function.

This section will first determine the minimum data set size required for each algorithm, since it is required to know this size to compare the performance yielding from the algorithms and to compare the time required by each algorithm.

This section will first present the set up of experiment 6. Then, the general results are presented in section 4.2.2. This section provides some expectations and initial conclusions. Additionally, in this section, the results are compared to the results found by Da Silva *et al.* [1] and striking results are explained. Section 4.2.3 uses the results of experiment 6 to find the minimum size of the data set required for each supervised learning algorithm. The performance of the supervised learning algorithms is compared in section 4.2.4.

4.2.1. SET UP OF EXPERIMENT 6

This experiment aims to find if polynomial regression, the neural network algorithm or locally weighted linear regression yields better or similar performance as SVR. Therefore, the performance of the models constructed using different supervised learning algorithms is compared. To do so, the minimum data set size required to construct a good model is determined for each algorithm. The comparison is performed using the data set of this size.

The experiment consists of three steps. First, a data set is obtained consisting of hundred training goals and optimal parameters. Optimal parameters are found for these training goals using reinforcement learning. These optimal parameters can span one or multiple subspaces in the policy space. This is checked using the ISOMAP algorithm [26]. Multiple models should be created if multiple subspaces are spanned.

Second, models are created using this data set and using subsets of the data set consisting of three, five, seven, nine, ten, twenty and up to ninety training goals. This is done for all supervised learning models. Note that the supervised learning algorithms find models for each of the parameters separately.

Third, these models are tested using 35 testing goals. Parameters are found for the testing goals using each model. The performance of the trajectory yielding from these parameters should meet the threshold set in experiment 5. If the threshold is not met, reinforcement learning iterations are used to improve the performance until the performance meets the threshold. If the threshold was not met after 100 reinforcement learning iterations, it was called a failed test.

4.2.2. GENERAL RESULTS OF EXPERIMENT 6

The general results are presented to get an impression of the performance yielding from the supervised learning algorithms and to define some expectations and initial conclusions. The analysis is performed as follows. First, expectations on the performance of the different algorithms are constructed using the shape of the models and their coefficients of determinations. Then, the mean performance yielding from the testing goals is analysed to check if the results are as expected. Also, the performance of the model found using SVR is compared to the results found by Da Silva *et al.* [1] to check if the results are as expected. Finally, a more in depth

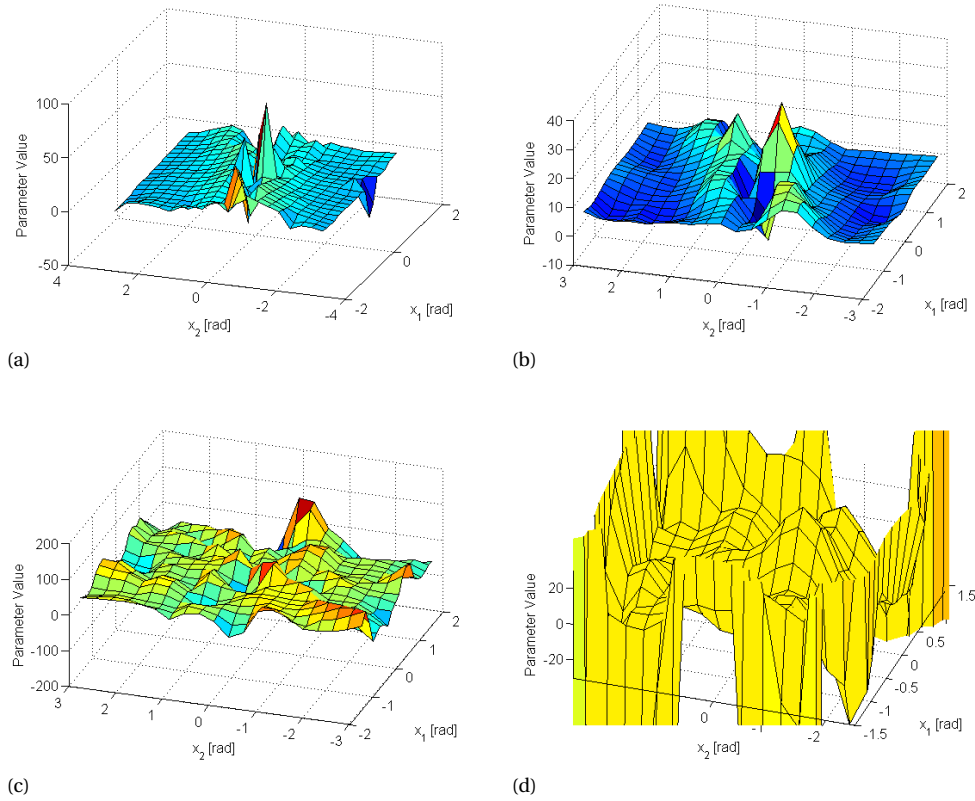


Figure 4.3: Typical Model found using locally weighted linear regression (a), support vector regression (SVR) (b), a neural network algorithm (c) and polynomial regression (d). One can see that the shapes of the locally linear and SVR model are similar, so their performance will probably also be similar. The shapes of the neural network and polynomial models are different, which will probably yield different results for the different supervised learning methods. The polynomial model even returns parameters values out of the range of this plot for certain goals, so it is expected that this model will have very bad performance.

analysis of several striking results is given.

MODELS CONSTRUCTED USING SUPERVISED LEARNING ALGORITHMS

Examples of the models constructed using the different supervised learning algorithms are plotted to see if they have a similar shape. This would imply that the results will also be similar. The coefficient of determination is used to assess how well the models fit the data and infer some expectation on the performance. Finally, this section presents a disadvantage of the neural network algorithm.

Figure 4.3 shows typical examples of the models found using the different supervised learning methods for a specific parameter. These models were found using optimal parameters of hundred training goals. One can see in figure 4.3 (a) and (b) that the models found using locally weighted linear regression and SVR have a similar shape. Therefore, it is expected that their results are similar. The neural network and the polynomial model both have different shapes. Therefore, it is expected that the performance of these models will be different.

Table 4.1 shows the coefficient of determination for the models. This value ranges from zero to one, where one means that there is no difference between the model output and the data and zero means that the fit is bad. One can see that the coefficient of determination is highest for locally weighted linear regression, followed by SVR. Therefore, it is expected that the models found using locally weighted linear regression and SVR will perform similar and better than the neural network and polynomial model. The latter two have different shapes and a lower coefficient of determination, therefore it is expected that their performance is different and worse.

The construction phase of Parameterized Skills showed that the neural network algorithm is not a very robust method. The coefficient of determination was used to see how well the model fit the data. Different models

Table 4.1: Coefficients of determination for all models constructed with 100 training goals. 1 means a very good fit and 0 means a bad fit. The coefficient is highest for locally weighted linear regression, so it is expected that this model has the best performance.

Parameter	1	2	3	4	5	6	7	8	9	10
Neural Network	0.7703	0.3837	0.8378	0.8104	0.5236	0.5138	0.6553	0.1957	0.6577	0.8735
SVR	0.8171	0.8457	0.6138	0.8408	0.8869	0.6376	0.6054	0.8513	0.8285	0.8595
Polynomial	0.8265	0.8241	0.9091	0.7344	0.8135	0.6491	0.6216	0.8030	0.7963	0.7754
Locally Linear	0.8749	0.9442	0.9782	0.9436	0.9551	0.9286	0.8603	0.9564	0.9421	0.9223

were found when the neural network algorithm was ran several times using the same parameter setting. The coefficient of determination of these models ranged from 0, meaning a very bad fit, to 0.8 or 0.9, meaning a pretty good fit. This means that neural network training is not very robust, which is important when Parameterized Skills is used in practice.

MEAN RESULTS OF EXPERIMENT 6

This section introduces the mean results of experiment 6. Parameters are found for 35 testing goals using models constructed with the complete data set or a subset of this data set. Trajectories are created and simulated using these parameters and it is checked if the performance of these trajectories meets the threshold in terms of accuracy and energy efficiency. If it does not meet the threshold, reinforcement learning iterations are performed until the performance meets the threshold. If more than hundred iterations are required, the test is called failed.

The mean results present the average performance over the testing goals. Four performance measures are analyzed: the number of learning iterations, the number of failed tests over the testing goals, the accuracy and the difference between the used torque and the threshold. These results are used to check if the behavior is as expected. Next to this, the results are compared to the results found by [Da Silva et al.](#). Also, striking results are noted and will be explained later.

Figure 4.4 shows the results of these performance measures against the size of the data set used to create the model. The results are as expected for SVR, the neural network algorithm and locally weighted linear regression. Their performance improves if a larger data set is used to construct the model. The performance of the polynomial model is bad and unexpected, since it does not improve with a larger data set. Therefore, the tests are repeated five times for the polynomial model and the average and standard deviation over these five repetitions is plotted.

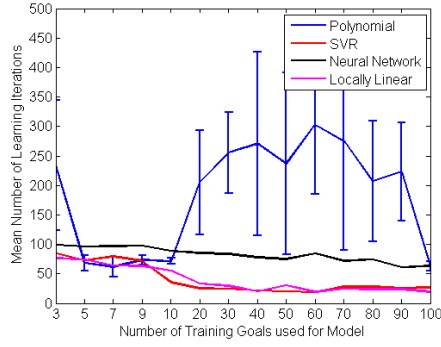
Note that in figure 4.4 (d) the polynomial model is not shown. This was done because the difference between the torque and torque limit using the polynomial model was up to three orders of magnitude larger. The difference between the other models would be invisible if this line was included in the figure. It is more important to show this difference since these models perform better.

One can see that locally weighted linear regression and SVR yield the best performance, which was expected (see section 4.2.2). These models require less learning iterations and have less failed test than the polynomial model and the neural network using all data set sizes. Also, the accuracy is higher and difference between the torque and its threshold is lower.

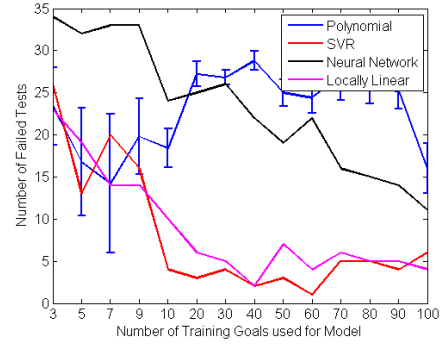
Two striking results were found, which will be explained in section 4.2.2. The number of learning iterations was rather high. All models, even the ones constructed with the largest data sets, yielded failed tests. Next to this, the performance of polynomial regression was bad and unexpected, since it does not improve when a larger data set is used to construct the model.

COMPARISON TO RESULTS OF DA SILVA *et al.*

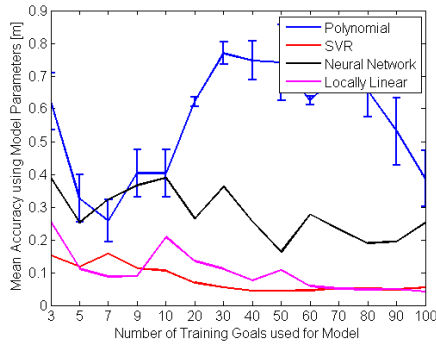
The results of experiment 6 are compared to the results found by [Da Silva et al. \[1\]](#). This is done to verify that the performance of Parameterized Skills is similar to its previous application. We will look at the relative decrease in learning iterations required when the SVR model is used. So, the relative decrease in number of reinforcement learning iterations required when starting from random initial parameters and when starting from parameters found using the SVR model is compared. A similar relative decrease implies that the advantage yielding from the SVR model is similar.



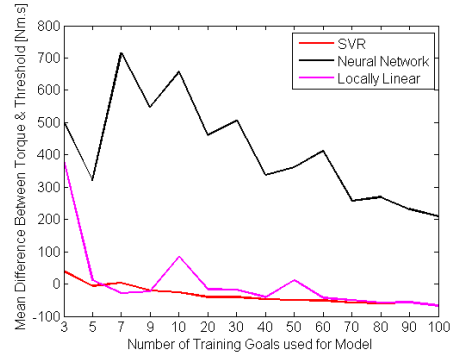
(a)



(b)



(c)



(d)

Figure 4.4: Mean Results of experiment 6. Figure (a) denotes the mean number of learning iterations required before the performance meets the threshold on accuracy and torque. Figure (b) denotes the number of failed tests. Figure (c) denotes the mean accuracy of the trajectory using the parameters found with the model. Figure (d) shows the mean difference between the torque and the threshold of the torque. This plot does not show the results for the polynomial model, since these were of a different order of magnitude. All plots show the size of the data set on the x-axis. The results are as expected, except for polynomial regression. Therefore, the test was repeated five times for this algorithm to show the range of possible results. It should be investigated why the polynomial model behaves so bad. Also, it should be investigated why tests fail for all supervised learning models.

It was found that the advantage of the SVR model is similar in the results presented here and the results presented by Da Silva *et al.* [1]. 22 learning iterations were required without a model in the application given by Da Silva *et al.* [1]. This decreased to 2 learning iterations when a model was created using 20 training tasks. This means that the number of learning iterations required decreases 91%. 170 learning iterations were required on the robot arm to learn performance that meets the threshold on the robot arm. 25 iterations are required using the parameters given by the SVR model that was created using 20 training tasks. This is a decrease of 85 %.

This means that the performance yielding from the SVR model is as expected. The number of learning iteration required is higher for this application, but this is due to the fact that a higher number of iterations is required starting from random initial parameters too.

There are two possible explanations for why the number of iterations is higher over all. The problem might be harder, meaning that if the threshold was set lower, it would be possible to use less iterations. Also, the learning process might be slower due to a different implementation of the reinforcement learning algorithm or a different parameter setting in the algorithm.

EXPLANATION OF STRIKING RESULTS

This section explains the striking results that were visible in figure 4.4. These results were unexpected and it is necessary to explain these results before conclusions can be derived from this data. There were two striking results. All supervised learning algorithms yielded failed tests and polynomial regression yielded bad results. This section will analyse both results.

It was found that it was impossible to find parameters yielding performance that met the threshold for some training goals, even after using 100 extra learning iterations. This was also the case for SVR and locally linear regression, which performed best. It should be determined if this is due to the fact that a bad model is created. It is checked if there is a consistency among the locations for which the controller returns parameters that do not yield good performance, even after 100 iterations. First, we will look at the results found for SVR. Then, we will look at the results found for locally weighted linear regression.

Figure 4.5 shows common locations of the failed tests for SVR in figure (a). Figure 4.5 (b) and (c) show the accuracy and difference between the used torque and the threshold as a function of the goal location. It is checked if the locations of figure (a) match locations of bad performance in figure (b) or (c).

The performance of the model in terms of accuracy and energy efficiency is not very good near the edges of the goal space. This explains the failed tests located at the right side of figure 4.5. One can see that the accuracy and the used torque show a larger difference with the threshold close to the edge of the goal space. This means that the data set used does not span the edges well.

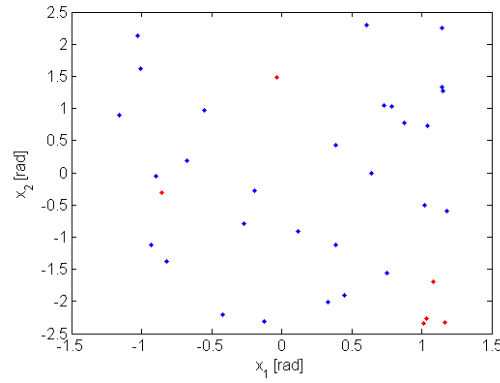
Figure 4.5 (b) shows that the performance in terms of accuracy is less good in certain locations in the goal space. This explains the failed test in the left side of figure 4.5 (a). This testing goal is located on a part where the accuracy yielding from the model shows a peak.

The failed test, located in the upper middle part of figure 4.5 (a), can be explained by the desired configuration corresponding to this goal. Figure 4.5 (b) and (c) show that the performance of the models is not very bad for the goal $[x_1, x_2]$ equals $[0, \pi]$ radians. However, the performance yielding from reinforcement learning does not improve the accuracy of the trajectory. This is shown in figure 4.6.

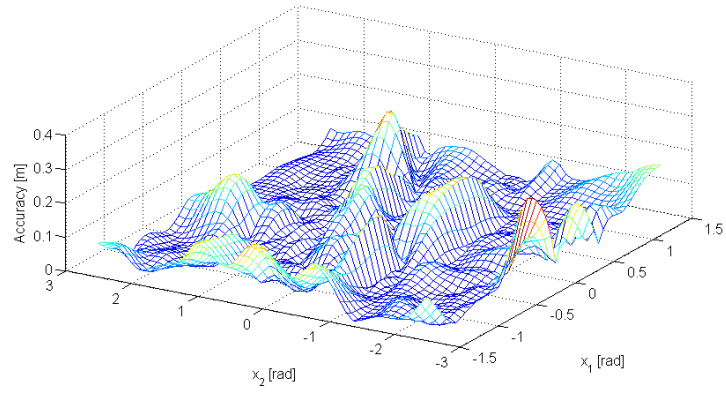
This figure shows the maximum of the threshold and the accuracy and difference between the used and threshold found after using extra learning iterations. The accuracy now meets the threshold for almost all goals, except for the configuration where x_1 is zero and x_2 equals π or $-\pi$ and for trajectories where the path length is small. The torque is not yet learned for the locations where the difference between the used torque and threshold was also highest without any learning iterations, as shown in figure 4.5 (c).

Next, the results found for locally weighted linear regression are analysed. Figure 4.7 (a) shows common locations of failed tests found using locally weighted linear regression. Figure 4.5 (b) and (c) show the accuracy and difference between the used torque and the threshold as a function of the goal location for locally weighted linear regression. One can see that the performance of the model is worse at the corners of the goal space. Also, there is slightly worse performance for the configurations where x_1 equals x_2 . This explains the locations of the failed tests found with locally weighted linear regression.

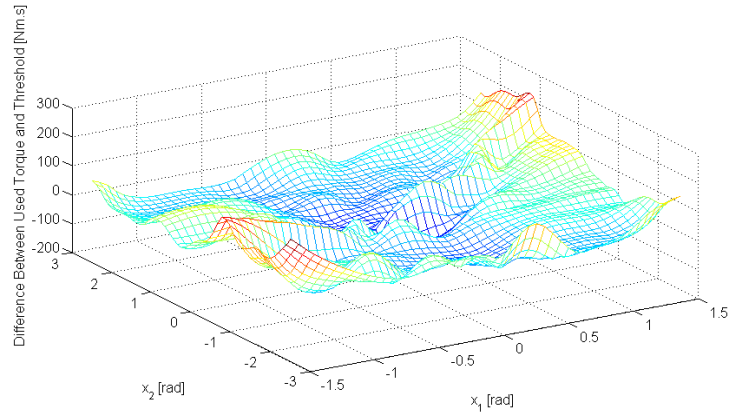
It can be concluded that there are certain locations in the goal space for which the model performs worse.



(a)



(b)



(c)

Figure 4.5: Plot of common locations of failed testing goals for SVR in figure (a), shown by the red dots. Figure (b) and (c) show the accuracy and difference between the torque and the threshold yielding from the parameters found for the SVR model constructed with 100 training goals. One can see that many tests fail at the corner in figure (a). Figures (b) and (c) show that the model performs worse at these corners. Also, a test, located in the left of figure (a) fails at a location where the accuracy is not correct. Also, tests fail when the angle of the second arm is equal to π , while the angle of the first arm is equal to 0, which is an difficult configuration for the robot.

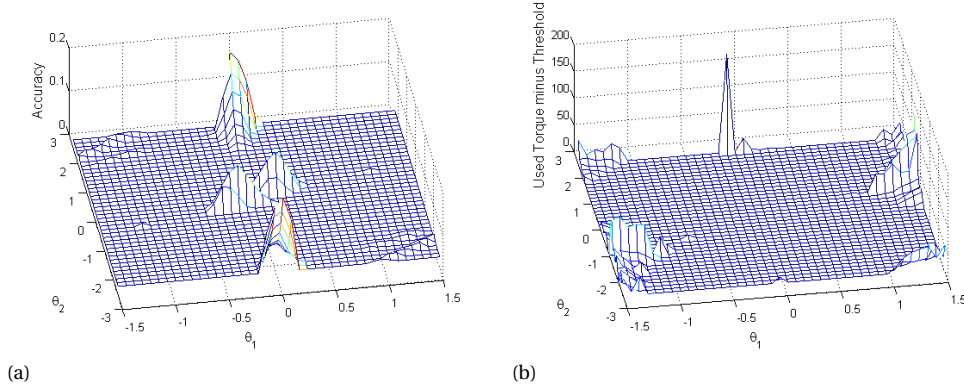


Figure 4.6: Plot accuracy and difference between torque and threshold found after learning iterations for locations where the threshold was not met. One can see that the accuracy does not meet the threshold for small motions and when x_1 is zero and x_2 goes to π . The torque does not meet the threshold for the location where the controller performed bad, as was shown in figure 4.5.

Therefore, special care should be taken for these locations. It is necessary to define the training goals such that they span the goal space well. This would make sure that testing goals located close to the edge of the goal space do not fail. Next to this, specific configurations of the arm require extra attention.

Secondly, it was found that the performance of the polynomial model did not improve if the size of the data set used to construct the model was increased. Next to this, the parameters found with the polynomial model would sometimes yield a trajectory with such high velocities, that the output of the simulation was 'not a number'. This means that it will not be possible to use the model, since this would yield dangerous situations when applied in practice.

It is hypothesized that the nature of the polynomial function causes the bad results. A polynomial function will always go to positive or negative infinity. This happens increasingly fast when the degree of the polynomial is increased. Therefore, the function could yield very high values if the randomly defined training goals do not span the entire goal space. This is also expected after analyzing figure 4.3.

This hypothesis also explains why the results do not improve. If a polynomial function yields a bad fit for the data set, it will not matter if the size of the data set is increased, since it is not possible to improve the results.

This hypothesis was verified by comparing the locations of the tests yielding 'not a number' to the locations of the training goals and peaks in the polynomial model. Figure 4.8 shows the location of these tests (red dots) and the polynomial model, created with 100 training tasks. One can see that the location of this testing goal is not covered by the training goals. This confirms the hypothesis that the nature of the polynomial model does not fit the data set.

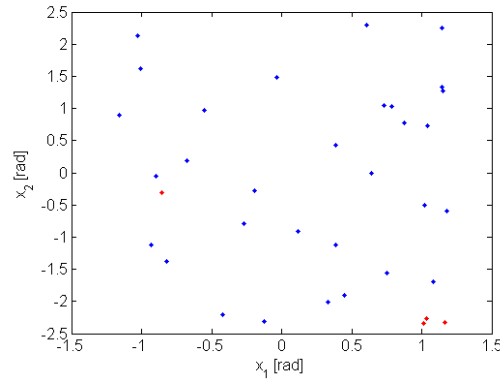
CONCLUSION

This section presented the general results of experiment 6. These results were analysed to check if the results were as expected and to provide some initial conclusions, which are presented next.

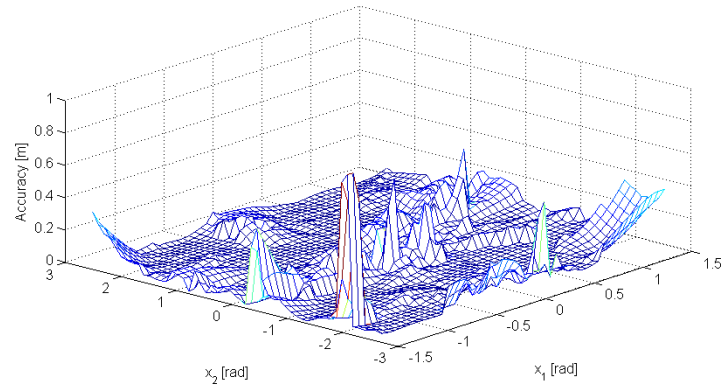
The results shown in figure 4.4 showed that none of the algorithms yielded performance that meets the threshold. Locally weighted linear regression and SVR yielded the best performance, followed by the neural network. Polynomial regression was not able to find a good fit to the data. This is due to the shape of the polynomial, which does not match the desired shape of the model. Therefore, it is decided not to take into account the results of this algorithm in further comparisons.

The performance of the SVR model was as expected. The results of SVR were compared to the results found by Da Silva *et al.* [1]. It was found that the use of supervised learning models caused a similar decrease in number of learning iterations.

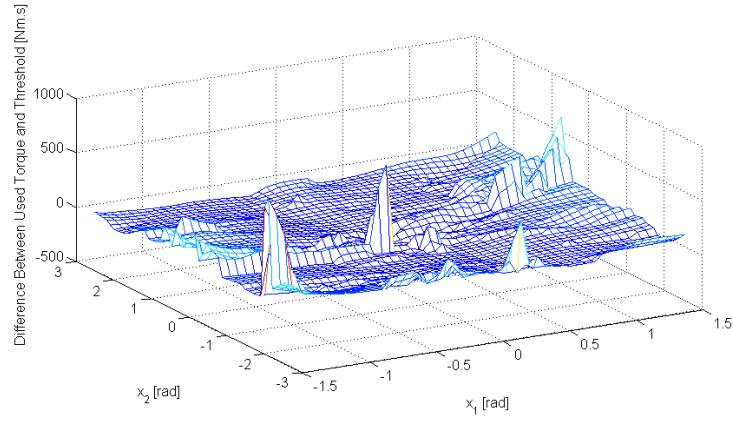
A final conclusion is that care should be taken for several goals. It is important that the training goals span the complete goal space well, including the edges, such that the model performs well in all locations. Also, specific locations require extra attention, since it is difficult to move to these locations.



(a)



(b)



(c)

Figure 4.7: Plot of common locations of failed testing goals for SVR in figure (a), shown by the red dots. Figure (b) and (c) show the accuracy and difference between the torque and the threshold yielding from the parameters found for the SVR model constructed with 100 training goals. One can see that three tests fail at the corner in figure (a). Figures (b) and (c) show that the model performs worse at these corners. The final failed test is located in the middle left of figure (a). One can see in figure (b) and (c) that the performance is worse here in terms of accuracy and torque. The performance is consistently slightly worse for the configurations where x_1 equals x_2

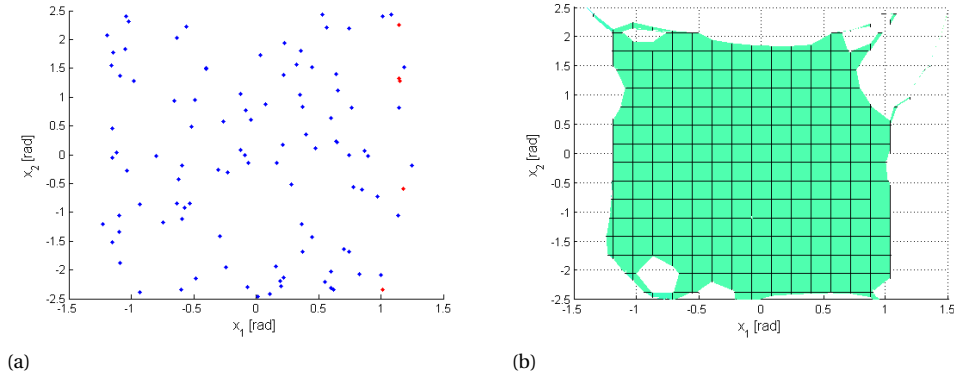


Figure 4.8: Plot of the location of training goals (blue) and testing goal yielding not a number (red) in figure (a) using the model constructed with 90 training goals. Figure (b) shows the part of the goal space (in green) where the parameters found with the polynomial model range between $[-100, 100]$. One can see that their location is at the boundary of the goals space and is spanned by the part of the polynomial model yielding very high parameter values, which explains why the results are very bad for these testing goals.

4.2.3. SELECTION OF DATA SET SIZES

Experiment 6 is used to find the minimum size of the data set required for each of the supervised learning methods. This size is used to compare the performance of the supervised learning algorithms and to find the time required to construct the models. These evaluations are performed using the data set size that is found in this section.

Boxplots will present the performance and the variance in the performance as a function of the size of the data set used to create the model. These boxplots plot the median, the 25th and 75th percentile, the range of the data and outliers. An ANOVA test is used to find the smallest data set for which the performance is not significantly different from the performance found for the largest data set.

An ANOVA test is a statistical test analyses if there is a significant difference between two or ore data sets with a 95% confidence level. The test assumes that the observations of the data set are independent. Furthermore, it is assumed that the variance of the data is normal and equal. This last assumption is not important if the data sets are of the same size.

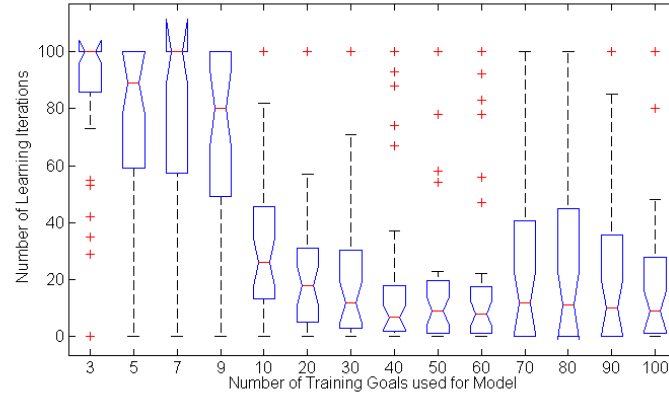
This test was performed for three performance measures that were presented in the general results of experiment 6. First, the minimum data set is defined for which the number of learning iterations is not different. Then, this test is repeated for the accuracy and the difference between the used torque and threshold. The minimum required size is equal to the maximum found in these three tests.

Figure 4.9 shows boxplots of the results of experiment 6. The number of learning iterations yielding from SVR, the neural network algorithm and locally weighted linear regression are plotted in figure (a), (b) and (c), respectively. This test was not performed for polynomial regression. One can see that all models show a decrease of the number of learning iterations required if a larger data set is used to create the model.

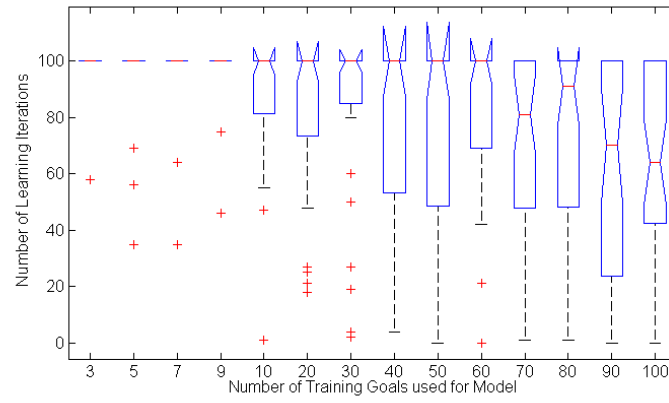
This data was used in the ANOVA test to find the smallest required data set. It was found that SVR required only ten training tasks to create the model, since there was no significant difference between the mean number of training tasks found with ten or more training tasks. It was found that locally weighted linear regression required twenty training tasks and the neural network required seventy.

These test were repeated for the accuracy and the differences between the used torque and the threshold. The results are shown in appendix A. The minimum data set size yielding from the results on the accuracy was equal to twenty for SVR and locally weighted linear regression. There was no significant difference between any of the models using the neural network algorithm. The difference between the used torque and the threshold required a data set size of ten for SVR and twenty for locally weighted linear regression and the neural network algorithm.

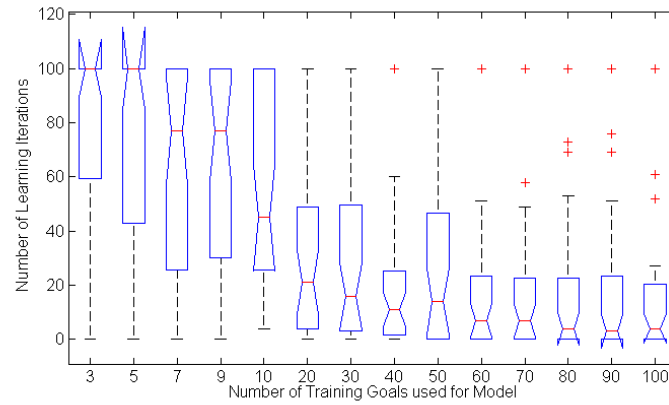
Hence, it was concluded that SVR and locally weighted linear regression both required a data set of size 20, since there is no improvement in the performance in terms of accuracy and number of learning iterations when a greater data set is used. The neural network requires 70 training goals to construct its model, since the number of learning iterations decreased until a data set of 70 was used, while the accuracy did not show any



(a)



(b)



(c)

Figure 4.9: Boxplot of the number of learning iterations found using SVR (a), the neural network algorithm (b) and the locally weighted linear regression (c). No significant difference was present between the results yielding from SVR using ten and hundred training tasks, meaning that only ten training tasks are required. Seventy training tasks are required for the neural network and twenty for locally weighted linear regression.

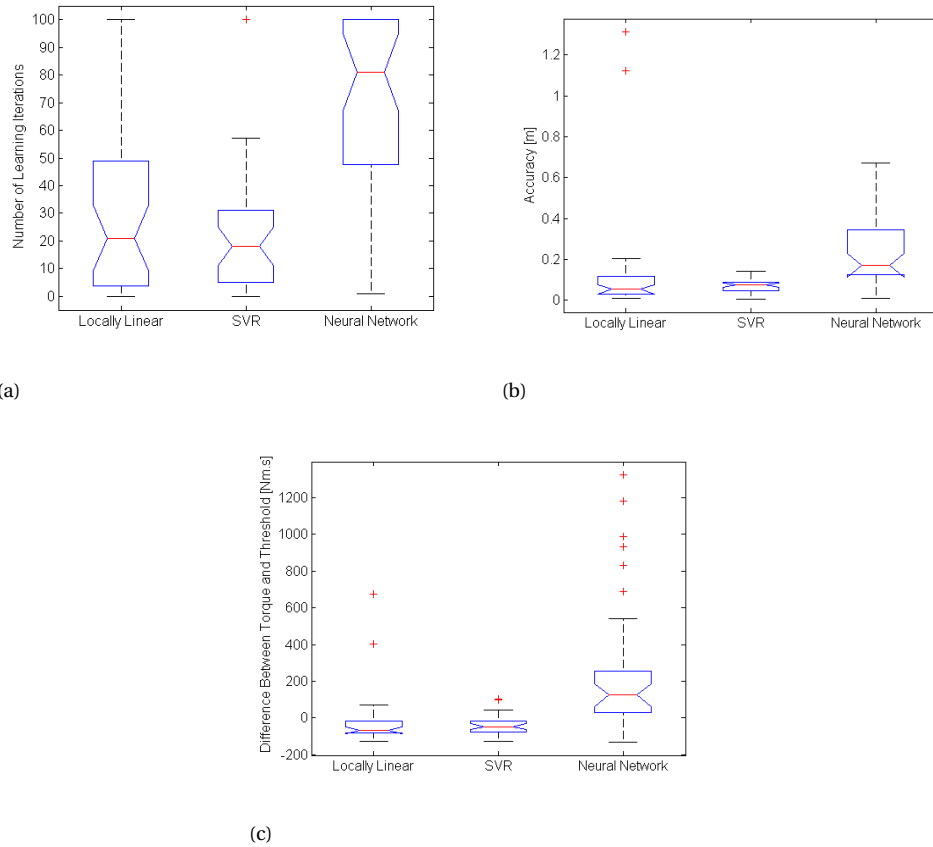


Figure 4.10: Boxplot of the performance of the different supervised learning methods using the minimum number of training tasks required. Figure (a) shows the results for the number of learning iterations required while figure (b) shows the results for the accuracy and (c) the results for the difference between the used torque and the threshold. The polynomial model was not taken into account since its variance was different, violating the assumptions of the test. It was found that locally weighted linear regression and SVR yield the best performance, without significant difference. There is significant difference between both methods and the neural network, which performs worse.

significant improvement and there were only ten tasks required in the difference between the torque and the threshold.

4.2.4. COMPARISON OF PERFORMANCE OF THE MODELS

Experiment 6 aims to compare the performance of the different supervised learning algorithms. This comparison will evaluate if there is any supervised learning algorithm that performs better than or similar to SVR. The mean results of experiment 6 already showed that polynomial regression is not a good fit to the data, therefore this model was not taken into account in this comparison.

The algorithms are compared using ANOVA tests. These tests are used to see if the performance on the three performance measures differs significantly between the supervised learning algorithms, with a 95% significance level. The performance measures that will be compared are the number of learning iterations, the accuracy and the difference between the used torque and the threshold. Based on the results for the individual performance measures, it is decided if the performance of any of the supervised learning algorithm is better than or similar to the performance of SVR.

Figure 4.10 shows a boxplot of the number of learning iterations (a), accuracy (b) and difference between the used torque and threshold (c) which were found for each supervised learning algorithm. One can see that the mean accuracy and mean difference between the used torque and the threshold is lowest using locally weighted linear regression, while SVR performs second best, followed by the neural network algorithm. The mean number of learning iterations is lowest using SVR, followed by locally weighted linear regression and the neural network algorithm.

It was checked if the differences between the means were significant. It was found that the performance yielding from locally weighted linear regression and SVR was not significantly different. The difference in the performance yielding from SVR and the neural network algorithm was significant. Therefore, it is concluded that the neural network algorithm performs significantly worse than SVR and that locally weighted linear regression and SVR yield similar performance.

4.3. COMPARISON OF TIME REQUIRED OF SUPERVISED LEARNING ALGORITHMS

Finally, the time required to construct models using the supervised learning algorithms is compared. This is done to find any of the supervised learning algorithms is faster than SVR. Experiment 7 is used to obtain information on the time required by the supervised learning algorithms. This experiment uses the data set of the sizes which were found in section 4.2.3. First, the set up of experiment 7 is presented, after which the results are presented and analysed. Finally, the total required time is compared in the conclusion.

4.3.1. SET UP OF EXPERIMENT 7

Experiment 7 times three phases of Parameterized Skills: the learning time, the construction time and the model time. The learning time consists of the time required to learn the data set. The construction time denotes the time required to construct the model. The model time denotes the time required to find the parameters given a testing goal. The model time is compared as well since locally weighted linear regression does not construct a full model, but constructs a local model during the model time.

Experiment 7 determines the learning time, construction time and model time of each supervised learning algorithm. The latter two directly depend on the supervised learning algorithm that is used. Therefore, it is expected that a speedup will be achieved here. The learning time is used to determine the significance of the speed up.

The learning time is found by timing the reinforcement learning process. This is done for the data set size which was found in experiment 6. Due to time constraints this is timed only once. The learning time is used only to find the significance of the speed up achieved during construction of the supervised learning models.

The construction time is found by timing the process of constructing the supervised learning models. This is repeated three times to account for variance in the results. I will look at the effect of the size of the data set to the construction time by plotting the constructing time versus the size of the data set. The relation between the construction time and size of the data set is important when Parameterized Skills is applied to a task with a varying start and goal state, since then a large data set is required. This figure will also show the standard deviation. The construction time is determined for each supervised learning method is equal to the time required to construct a model using the size of the data set found in experiment 6.

The model time denotes the time required to find parameters given a testing goal, using the model. This is included since locally weighted linear regression method also spends this time to construct the locally linear model around the testing goal. Therefore, the model time required for locally weighted linear regression might be higher, such that the model time is equal to the combined model and construction time required by another supervised learning algorithm.

The model time is found by determining the average of the time required to find parameters for 35 testing goals. The model time will also be plotted against the size of the data set, as was done for the construction time. The model time required for the supervised learning algorithms is based on the size of the data set found in experiment 6.

After the three times are determined, they are added to find the total time required to construct and use the models. It is expected that a speed up is achieved during the construction time and the model time. The difference in total time will show the significance of this speed up. The plots of the construction time and model time against the size of the data set used to construct the model is also used to define some expectations on the time required in experiment 9, where a controller is trained to move from any start position to any goal position.

Table 4.2: Times required to create a data set consisting of 20 or 70 training tasks. The SVR, locally linear and polynomial model require a data set of 20 training goals. The neural network requires 70 training goals.

Size of data set	Time required to construct data set
20	4216 s
70	9497 s

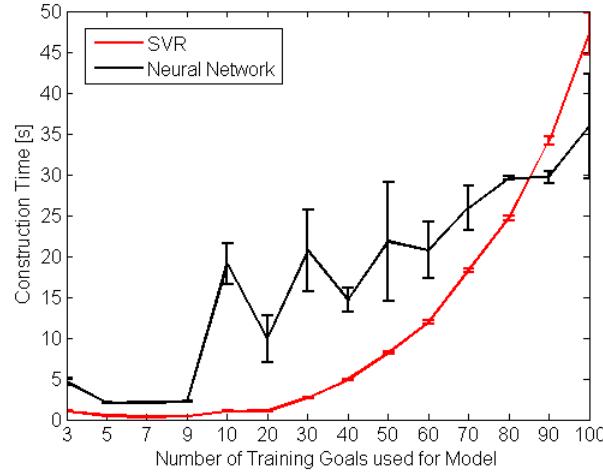


Figure 4.11: Plot of the construction time versus the size of the data set used to construct the model. The locally linear regression method is not taken into account since this method does not construct a model. The polynomial method is fastest in constructing its model. The SVR method and neural network method perform worst and it is clear that a large data set will yield very large construction times for both methods. It was found that the SVR method requires 1.05 seconds to construct a model using 20 training goals. The neural network requires 25.9 seconds to construct a model using 70 training goals.

4.3.2. RESULTS AND ANALYSIS OF EXPERIMENT 7

This section presents the learning time, construction time and model time for SVR, locally weighted linear regression and the neural network algorithm. The fastest method will be identified in terms of construction time and model time. The times are summed in the conclusion to find the total time and to evaluate the significance of the difference.

The learning time was determined by timing the reinforcement learning process. The SVR model and locally linear model both require twenty training tasks, while the neural network algorithm requires seventy. Table 4.2 denotes how much time is required when creating a data set of these sizes. It was found that 1658 s are required to find optimal parameters for twenty training goals and 5196 s to find optimal parameters for seventy training tasks.

Figure 4.11 shows the construction time as a function of the size of the data set used for the different supervised learning methods. One can see that the construction time of SVR and the neural network algorithm both increase with a larger data set. SVR has a higher slope than the neural network algorithm. Also, SVR has a lower variance. This is due to the fact that the neural network algorithm is less robust. Different neural networks are found if the algorithm is ran twice, yielding that the construction time also shows more variation than the construction time required for SVR, since this method finds a much more similar model.

Next, the actual construction time for the SVR model and the neural network is determined using the minimum size of the data set found in experiment 6. The SVR model has a construction time of 1.05 seconds for a model consisting of 20 training goals, while the neural network requires 25.9 seconds to construct the model using 70 training goals.

Finally, the model time is analyzed. Figure 4.12 shows the model time as a function of the size of the data set used. One can see that the neural network is much slower in producing its outputs than the other models, therefore figure 4.12 (b) shows the same graph but without the neural network. The high model time for the neural network can be explained because the neural network has a more complex structure than the SVR

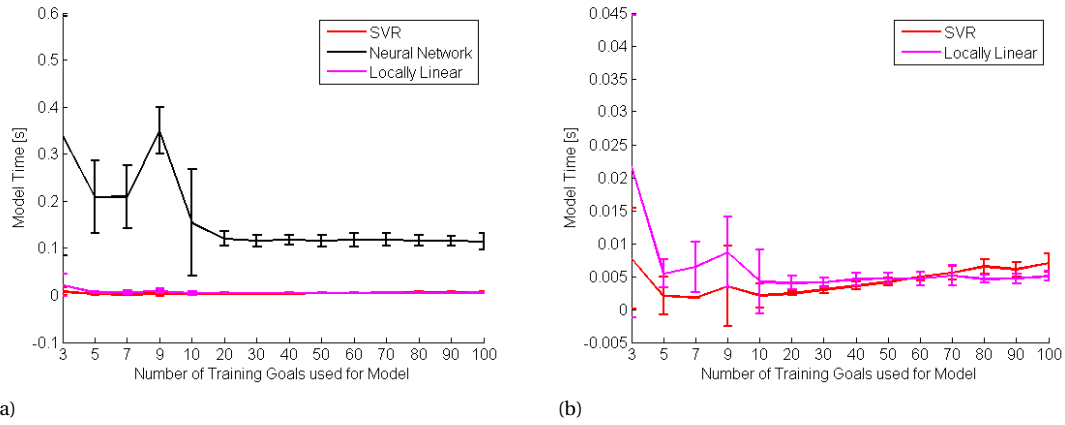


Figure 4.12: Plot of the model times versus the number of training goals used. One can see that the neural network is much slower when finding its output, therefore figure (b) plots the same information without the neural network, to see the differences between the other models more clearly. The model time is dependent on the size of the model for locally weighted linear regression and SVR, while SVR has a higher slope. It is important to notice that locally weighted linear regression does not require a larger model time to constructs its model during this part of Parameterized Skills. The model times for SVR, the neural network and locally weighted linear regression are $2.4 \cdot 10^{-3}$ s, 0.118 s and $4 \cdot 10^{-3}$ s, respectively.

model and the locally linear models.

One can see a bump in the graph when the model is used which was created using ten training goals. It was found that the first iteration in MATLAB required more time when the model was implemented. The algorithm was run twice, once for the models created using three, five, seven and nine training tasks and once using the other models. Therefore, the model for the model created using three and ten training goals is higher. This is not important in the actual comparison, since none of the methods use three or ten training goals to create the models.

It is important to note that locally weighted linear regression is not slower than the other models. This was expected since the locally linear algorithm also constructs the local model during this part of the algorithm and has zero construction time. Therefore, a speed up is achieved when locally weighted linear regression is used instead of SVR.

4.3.3. COMPARISON OF TOTAL TIME REQUIRED BY SUPERVISED LEARNING ALGORITHMS

Experiment 7 was conducted to evaluate the time required to construct a controller using the different supervised learning algorithms. It was found that locally weighted linear regression is faster than SVR in construction time and model time. Next, we will look at the effect of this speedup on the full time required by Parameterized Skills.

Table 4.3 gives an overview of the times for each of the models and the total time required. One can see that the locally linear regression method and SVR method are fastest in terms of learning time, since they require the smallest data sets. Comparing the learning time to the construction time and model time, it can be seen that the speedup in construction time is negligible, since the learning time is much larger than the construction time and model time.

It is expected that when Parameterized Skills is expanded to the case where the start and goal position are varied, the required size of the data set will increase quadratically. The construction time of an SVR model increases exponentially with an increased size of the data set, while the learning time increases less fast. Therefore, it is expected that there will be a significant difference for this case.

4.4. SUMMARY

This chapter aims to answer the first subquestion of the research question: **Which supervised learning algorithm yields the best performance of Parameterized Skills in terms of accuracy and efficiency and in terms of time required if a different supervised learning algorithm than support vector regression is used?**

It was expected that a faster method was found when a method was used different from support vector regres-

Table 4.3: Overview of the learning time, construction time, model time and total time required by each method. It was found that there was no significant difference between the total time required for SVR and locally weighted linear regression, since the learning time was by far the highest. Therefore, it was found that a negligible advantage was gained using locally weighted linear regression.

Supervised Learning Method	Learning Time [s]	Construction Time [s]	Model Time [s]	Total Time [s]
SVR	1658	1.05	$2.4 \cdot 10^{-3}$	1659
Neural Network	5196	25.9	0.118	5222
Locally Weighted Linear Regression	1658	0	$4 \cdot 10^{-3}$	1658

sion (SVR). It should also be checked if the model performed better than SVR, such that the performance found by the model meets the threshold. It was found that locally weighted linear regression and SVR yield the best performance in terms of accuracy and efficiency. Locally weighted linear regression is slightly faster. However, this difference is negligible.

Several steps were taken to answer this question. First, a threshold was set that defined when the performance of the models was good. Then, the performance of the models was compared. Finally, the time required to construct the models was compared.

The threshold on the accuracy was set to a fixed value. The difference between the desired and actual end-point should not exceed two centimeter. The threshold on the used energy was set on the integral of the used torque over the trajectory, since the used energy is directly dependent on the square of the torque. The used torque is dependent on the distance, therefore a relation was found that defined the torque limit as a function of the distance traveled.

Next, experiment 6 was performed and the results of this experiments were analyzed. All supervised learning algorithm required extra learning iterations to meet the threshold. SVR and locally weighted linear regression yielded the best performance, followed by the neural network algorithm. Also, it was found care should be taken to define training goals that span the goal space well, otherwise the controller does not yield good performance at the edge of the goal space. Finally, it was found that the nature of the polynomial function did not fit the model data well, so the results of the polynomial model were not taken into account in further comparisons. Also, it was found that the results were comparable to the previous implementation of Parameterized Skills.

Next, the algorithms were compared to find the one that performs best. First, the minimum size of the data set required to construct the model was determined. It was found that 20 training goals were required for SVR and locally weighted linear regression, while the neural network required 70 training goals. Using these data set sizes, the performance yielding from SVR and locally weighted linear regression did not differ significantly. The neural network showed significantly worse performance.

The last comparison was performed on the time required to construct the models. It was found that most time was spent on learning the data set for the models. Therefore, the difference in construction time was not significant, even though locally weighted linear regression was faster than SVR. However, it is expected that a speed up is achieved when a larger data set is required, since the time required for SVR increases exponentially for a larger data set.

5

PARAMETERIZED SKILLS APPLIED TO A TASK WITH FOUR DIMENSIONS

Three issues should be solved to enable the use of Parameterized Skills to train a versatile controller. Chapter 4 analysed the first two issues and evaluated the performance and time required of Parameterized Skills when applied to learn a two dimensional task. It was found that the performance yielding from locally weighted linear regression and support vector regression (SVR) is similar and better than the performance yielding from the neural network algorithm and polynomial regression.

This chapter will evaluate the performance of Parameterized Skills when it is used to learn a task with four dimensions, namely to learn to move from any start position to a any goal position. My aim is to analyze if the required number of training tasks scales with the increased complexity of the task. The dimension of the tasks is doubled, meaning that the size of the data set used to construct the models should be increased quadratically with respect to the data set required on a two-dimensional task.

It is checked if this is the case by comparing the performance on the four dimensional task to the performance on the two dimensional task that was learned in experiment 6. The performance will be compared in terms of accuracy and energy efficiency of the controller but also on the time required to construct the model. It is expected that a significant speed up is achieved using locally weighted linear regression, since the size of the data set is larger.

This analysis is performed using two experiments that are given in the overview of experiments in figure 5.1, experiment 8 and 9. Experiment 8 will analyze the performance of Parameterized Skills in terms of accuracy and energy efficiency when it is used to learn a task with a varying start and goal position on the robot arm. This experiment is used to check if the expectation is correct that the size of the data set should be increased quadratically to yield performance that meets the threshold. Two supervised learning algorithms will be implemented and analyzed: SVR and locally weighted linear regression, since these performed best when learning a task with a varying goal position (see chapter 4).

Experiment 9 will compare the time required to construct the models for both algorithms. SVR optimizes a vector with the length equal to the size of the data set to find a model that describes a parameter as a function of the task. The increased complexity requires a larger data set, therefore it is expected that locally weighted linear regression will yield a significant speed up when learning this task. This algorithm does not use any construction time, since it only constructs a local model when it is required to find parameters for some task.

This chapter is build up as follows. First, section 5.1 introduces experiment 8. The experimental set up will be given, as well as the results and an analysis of these results. Then, section 5.2 presents the set up and results experiment 9, as well as an analysis of these results. Section 5.3 summarizes this chapter.

5.1. PERFORMANCE OF PARAMETERIZED SKILLS ON A TASK WITH FOUR VARYING DIMENSIONS

This section analyses the performance of Parameterized Skills on task with four dimensions. It is expected that the size of the data set used to construct the model should be increased quadratically to get similar performance as on a task with only a varying goal position. Experiment 8 will verify if this is true. This is done

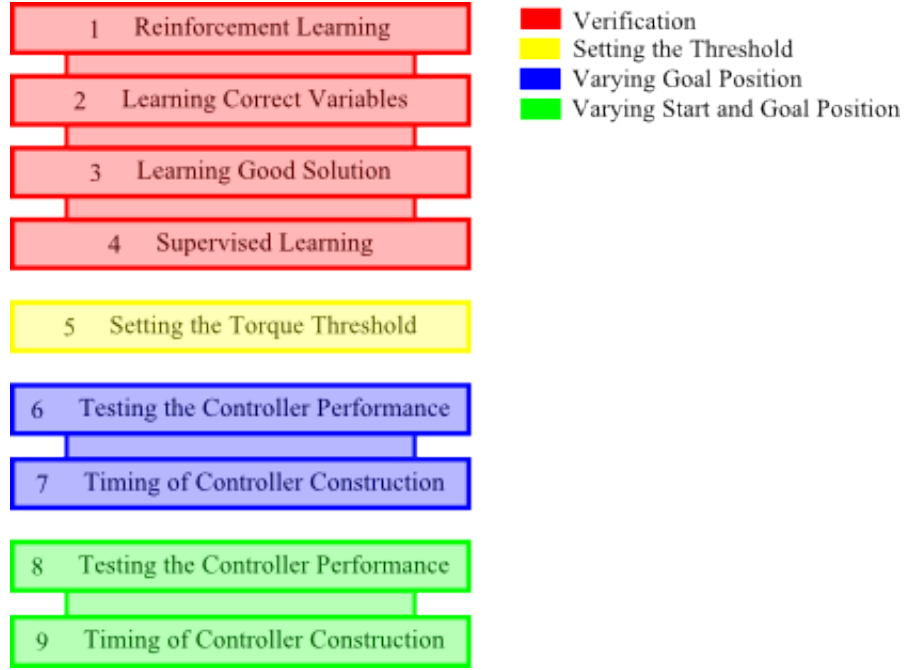


Figure 5.1: Overview of all experiments conducted in this thesis. This chapter will introduce the set up and results of experiments 8 and 9.

by applying Parameterized Skills to learn a task with a varying start and goal position on the two degree of freedom robot arm.

Section 5.1.1 introduces the set up of experiment 8. Then, section 5.1.2 introduces the general results of this experiment. This section analyses if the performance in terms of accuracy and energy efficiency is as expected. This is the case when the performance improves if a larger data set is used. The results are also used to check if the performance yielding from SVR and locally weighted linear regression is similar. In section 5.1.3, the results of experiment 8 are compared to the results of experiment 6 to evaluate if Parameterized Skills scales well with an increased dimension.

5.1.1. SET UP OF EXPERIMENT 8

This section introduces the set up of experiment 8. This experiment is conducted to evaluate the performance of the controller found using Parameterized Skills for a task with a varying start and goal state. The following procedure is followed to create and test the controller.

The controller is constructed using a data set consisting of optimal parameters for training tasks. Reinforcement learning is used to find optimal parameters for thousand training tasks. The number of subspaces that are spanned by these parameters in the policy space is checked. Then, models are created for each of the subspaces. These models are created using the full data set and subsets of three, five, seven, nine, ten, twenty, up to hundred training tasks and using two hundred, three hundred and up to nine hundred training tasks.

These models are analysed using fifty testing tasks. These tasks consist of a random start and goal position. The models are used to find parameters for these testing tasks. Trajectories are created using these parameters and the performance of the trajectories is analyzed. It is checked if the accuracy and used torque meet the threshold. If this is not the case, extra reinforcement learning iterations are performed until the threshold is met. A test is called failed if more than hundred iterations are required.

5.1.2. GENERAL RESULTS OF EXPERIMENT 8

This section presents the general results of experiment 8. These results are used to check if the performance is as expected. This means that the performance of the models improves if a larger data set is used. These results are also used to check if SVR and locally weighted linear regression yield similar performance, and to evaluate if there are any striking results. The mean performance of four performance measures is plotted: the number of learning iterations, the number of failed tests, the accuracy yielding from the model parameters and the difference between the used torque and the threshold yielding from the model parameters.

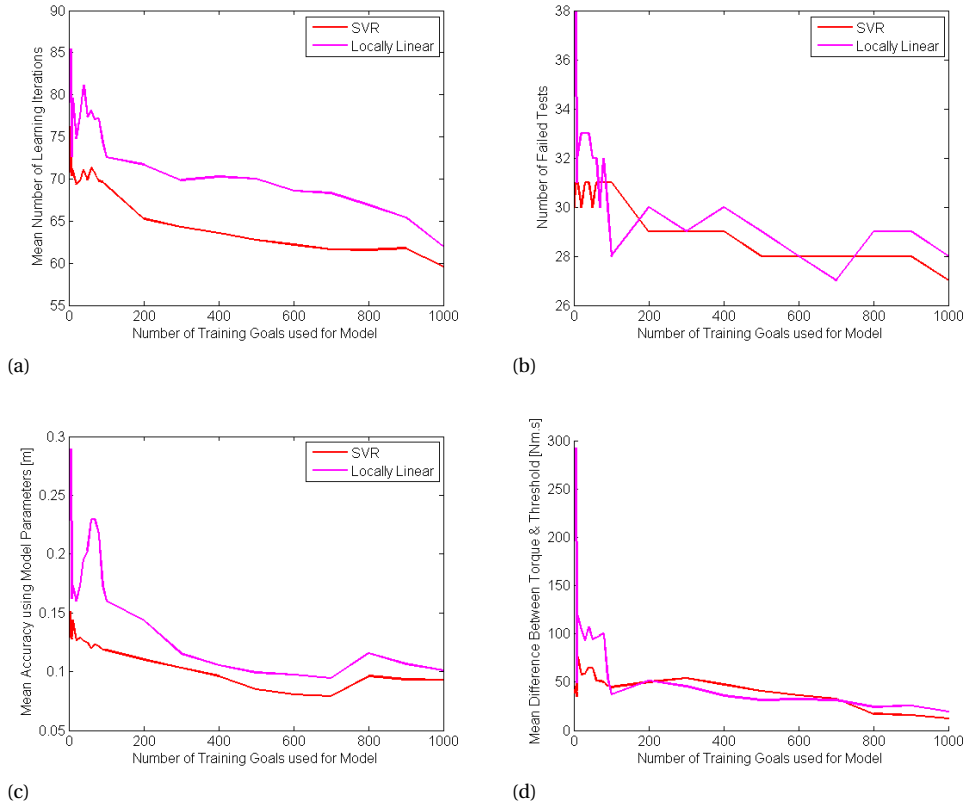


Figure 5.2: Mean Results of experiment 8. Figure (a) denotes the mean number of learning iterations required before the performance meets the threshold on accuracy and torque. Figure (b) denotes the number of failed tests. Figure (c) denotes the mean accuracy of the trajectory using the parameters found with the model. Figure (d) shows the mean difference between the torque and the threshold of the torque. One can see that the performance of the models is similar, but the SVR model is slightly better. The shape of the results are as expected since the performance improves for a larger data set.

Figure 5.2 shows the general results of experiment 8. One can see that the performance of the models is as expected, since the performance improves for a larger data set. ANOVA tests confirm that the performance of SVR and locally weighted linear regression is similar.

It is striking that in figure 5.2 (d) the difference between the used torque and the threshold is high. The threshold was determined using experiment 5 (see section 4.1.1). Here, the path length of all training tasks was plotted against the used torque and a first order polynomial was fitted through this data. The range of path lengths is larger when the start position is also varied. Therefore, this threshold might not fit the data found in experiment 8 well, especially for larger path lengths.

Figure 5.3 shows that this is the case. The black line shows the threshold that was set in experiment 5. One can see that the threshold on the used torque is high for trajectories with a small path length and low for trajectories with a large path lengths. This explains the high difference between the used torque and the threshold in the results of experiment 8, since it is very challenging to meet this threshold for trajectories with a large path length.

A new threshold can be defined using the data set obtained in experiment 8. This threshold is set in the same way as was done in experiment 5, using the data obtained in experiment 8. This means that a first order polynomial function is fitted through the data presented in figure 5.3. The standard deviation of the difference between the output of the polynomial function and the actual used torque is added to the function to obtain the threshold. This threshold is shown by the red line in figure 5.3.

Figure 5.4 shows the results using this new threshold. One can see that the mean number of learning iterations decreased around twenty iterations and that the number of failed tests decreases with ten. Also, the difference between the torque and the threshold is less.

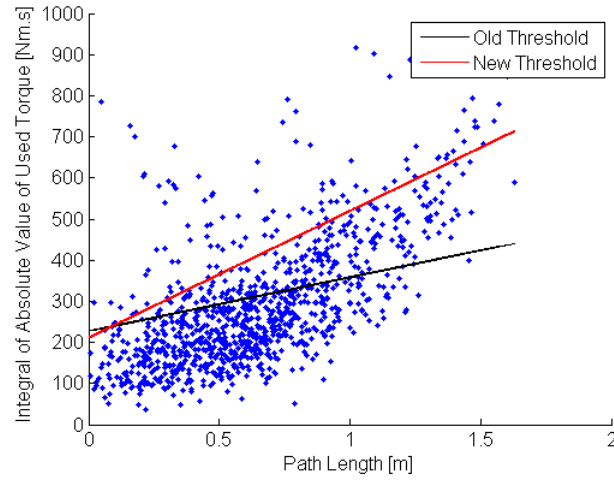


Figure 5.3: Plot of the used torque versus the path length yielding from the optimal parameters of the training goals. One can see that the threshold (black line) that was set in experiment 5 is not a good fit to the data when the path length is longer. Therefore, a new threshold is set, which is shown by the red line.

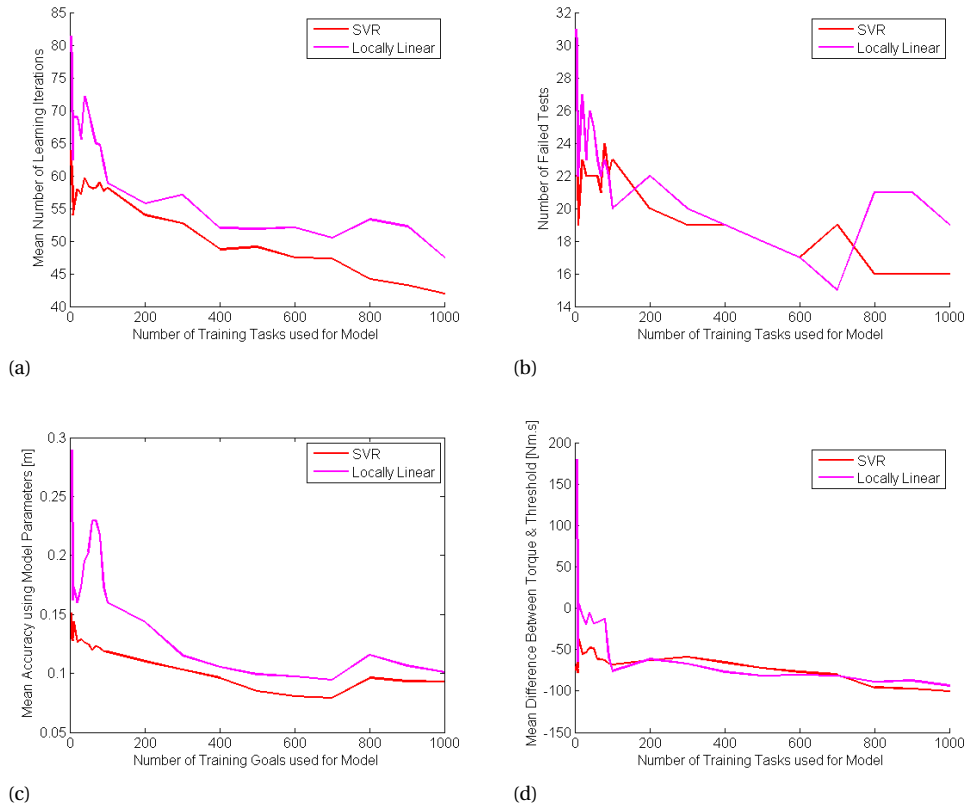


Figure 5.4: Mean Results of Experiment 8 using the updated threshold. The number of learning iterations has decreased with twenty using this threshold and the number of failed tests decreased by ten. Also, the difference between the used torque and the threshold is now as expected. These results will be used in further analysis.

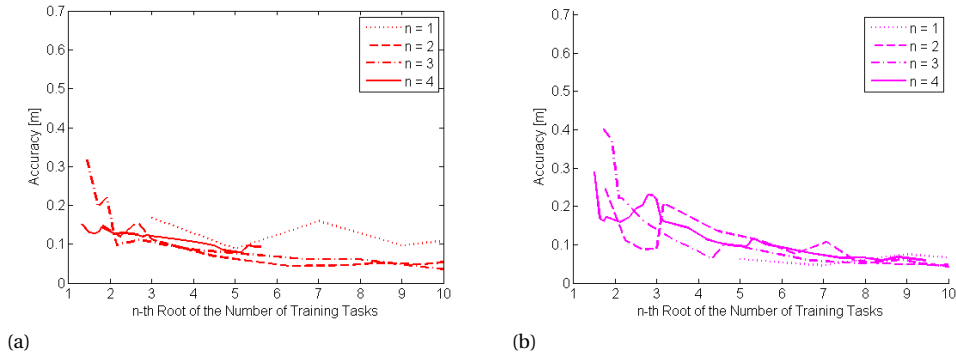


Figure 5.5: Plot of the accuracy found using SVR (a) and locally weighed linear regression (b), scaled for the task dimension. These figures plot the accuracy found for a task with one, two, three and four dimensions, where n is the dimension of the task. One can see that SVR and locally weighted linear regression have very similar performance when the dimension of the task is varied. SVR seems to have slightly more variation in the performance.

5.1.3. COMPARISON OF RESULTS, SCALED FOR INCREASED COMPLEXITY

My aim is to analyse if the size of the data set required in Parameterized Skills scales well with an increased complexity. Therefore, the results of experiment 8 will be compared to the results of experiment 6, where only the goal position is varied. Problem 1 denotes the task with two dimension, which was learned in experiment 6. Problem 2 denotes the task with a varying start and goal position. It is expected that the size of the data set for problem 2 should be equal to the square of the size of the data set used in problem 1. Problem 1 required at least twenty training tasks to find a good model. Therefore, this comparison is performed using a model that was constructed using twenty training tasks in problem 1 and a model that was constructed using 400 training tasks in problem 2.

This comparison is done as follows. First, the accuracy yielding from tasks with one to four dimensions is plotted as a function of the number of training tasks and scaled for the task dimension. Second, the variance of problem 1 and 2 will be compared using a Brown-Forsythe's test. A difference in the variance implies that the performance is also different. Then, an ANOVA test will be used to compare the overall performance of the data sets found in experiment 6 and 8. The variance of the performance and the overall performance are compared for three performance measures: the number of learning iterations, accuracy and difference between used torque and threshold, where the new threshold is used.

Figure 5.5 plots the performance found using SVR (a) and locally weighted linear regression (b), scaled for the dimension of the task n . This figure plots the performance of the models for tasks with one, two, three and four dimensions. For the task with one dimension, only the goal angle of the first arm, g_1 was varied. For the task with three dimensions, the goal position was varied, as well as the first dimension of the start position of the arm, x_{0_1} .

One can see that SVR and locally weighted linear regression scale well when the task of the dimension is varied. The lines overlap for both algorithms. Also, the slope of the lines is similar for the different tasks. Next, statistical tests are used to see if there are any significant difference in the performance of the tasks with two and four dimensions.

Table 5.1 gives an overview of the variances found on the performance measures for the models constructed using SVR and locally weighted linear regression. The bold text indicates that the variance is similar for these performance measures. This was the case for the accuracy and the difference between the used torque and the threshold using locally weighted linear regression. A significant difference in variance was found for the number of learning iterations for locally weighted linear regression and for SVR in all performance measures.

Figure 5.6 shows boxplots of the results of problem 1 and problem 2. One can see on the right side of figure 5.6 that the results are similar for problem 1 and 2 for locally weighted linear regression. The ANOVA test confirmed that there was no significant difference for the locally weighted linear regression algorithm. There was a significant difference in the used torque between problem 1 and 2 for SVR. The performance was similar on the accuracy.

One can see in figure 5.6 (e) and (f) that the performance in the number of learning iterations is significantly

Table 5.1: Overview of the variances found for the performance measures. Bold text indicates that the assumption of homoscedasticity is met, so the variance is not different. This was the case for the difference between the used torque and the threshold and the accuracy using locally weighted linear regression.

Performance Measure	SVR		Locally Linear	
	Problem 1	Problem 2	Problem 1	Problem 2
Number of Learning Iterations	778	1906	1314	1568
Accuracy	0.0011	0.0061	0.0764	0.0109
Torque	1567	12 918	27 215	12 585

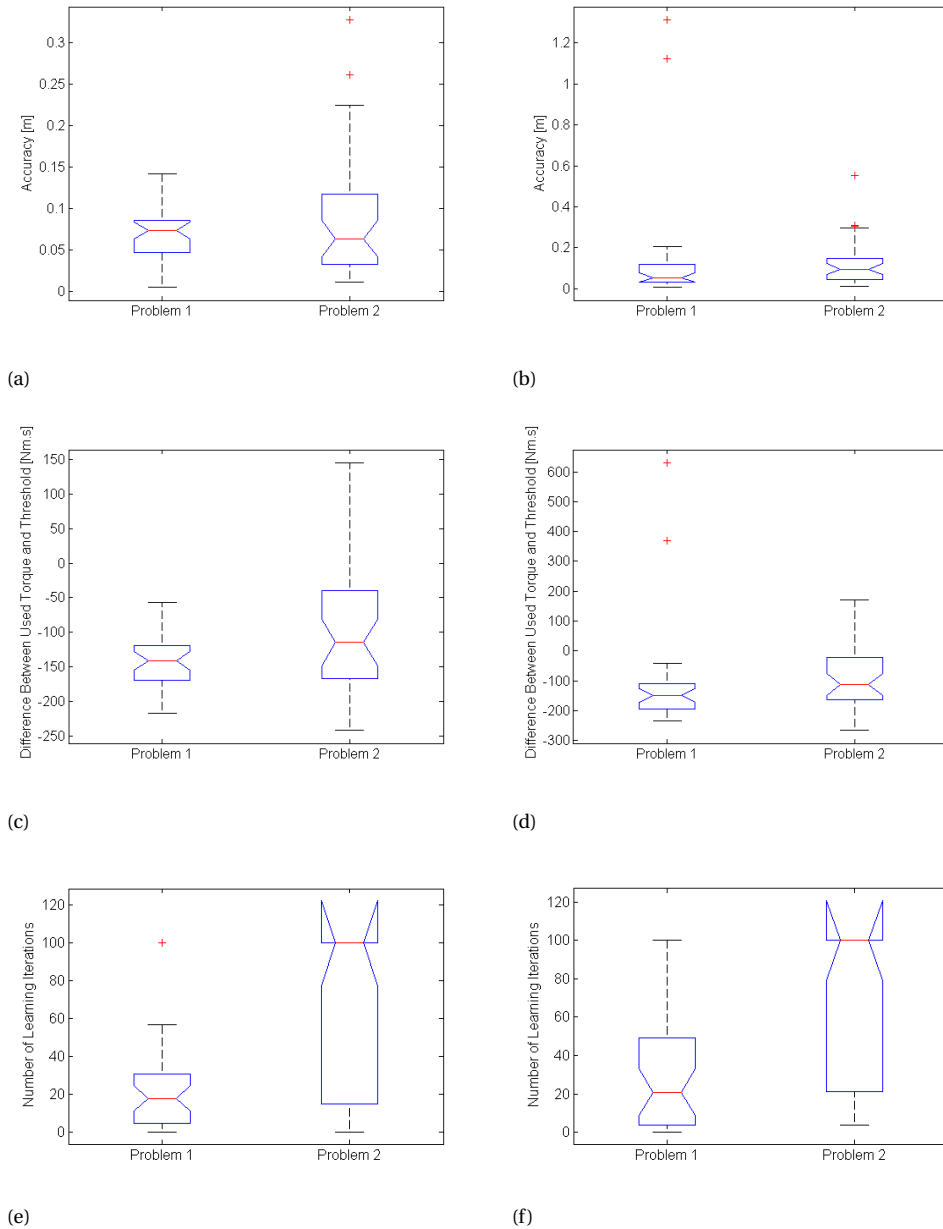


Figure 5.6: Boxplots of the comparison of problem 1, with only a varying goal position, and problem 2, with a varying start and goal position. The plots on the left are the results found using SVR, while the results on the right are found using locally weighted linear regression. A significant difference was found for the number of learning iterations for both algorithms, meaning that the number of learning iterations is significantly worse in problem 2. There is also a significant difference in the used torque for SVR, meaning that SVR performs slightly better in problem 1. However, there is no significant difference between the accuracy and used torque found for problem 1 and 2 using locally weighted linear regression

worse when the start and goal state is varied. This is due to the increased number of iterations required to learn this task using reinforcement learning. More than 600 learning iterations were required to learn parameters that yield performance meeting the threshold for the training tasks in problem 2, compared to an average of 170 learning iterations for problem 1. Therefore, these results are not used to assess how well the algorithm scales with an increased dimension. Instead, the performance in terms of accuracy and energy efficiency is used.

It is concluded that locally weighted linear regression scales better with an increased dimension than SVR. The results and the variance of the results of problem 1 and 2 are similar for locally weighted linear regression, while there is a significant difference between the results and the variance of the results in problem 1 and 2 using SVR. For SVR, the variance which is found using 20 training tasks in problem 1 is found using 900 training tasks in problem 2.

5.2. COMPARISON OF REQUIRED TIME FOR EACH ALGORITHM

The time required for locally weighted linear regression is compared against the time required for SVR. It is expected that a speed up is achieved using locally weighted linear regression since SVR aims to find using a vector with the length equal to the size of the data set an optimization algorithm. A large data set is required for a task with a varying start and goal position. This section presents the set up of experiment 9 and analyses its results.

5.2.1. SET UP OF EXPERIMENT 9

This experiment compares the time required by SVR and locally weighted linear regression separately for three phases of Parameterized Skill. These phases are the reinforcement learning process, the construction of the model and the use of these models on testing tasks. These time required for these phases are called the learning time, the construction time and the model time. A speed up is expected in the construction time and the model time. The learning time is recorded to find the significance of the speed up.

The learning time is found by recording the time required to perform reinforcement learning for 400 training tasks.

The construction time is found by recording the time required to construct the model. This is done for the models constructed using the complete data set and subsets of the data set. Locally weighted linear regression does not construct a separate model, therefore this is done only for SVR. Due to the time required to construct the model, this phase is only recorded once.

The model time is recorded for all testing tasks created in experiment 8. This is done for the models that were created using the complete data set and subsets of this data set. The model times are averaged over the testing tasks to find the model time of SVR and locally weighted linear regression.

5.2.2. RESULTS AND ANALYSIS OF EXPERIMENT 9

This section will present the learning time, construction time and model time for SVR and locally weighted linear regression. These are compared to find the speed up that is achieved using locally weighted linear regression. The learning time is equal to 49 977 s, approximately 14 hours.

Figure 5.7 shows the construction time as a function of the size of the data set used to construct the model. This plot only shows the construction time for SVR, since locally weighted linear regression does not require any learning time. One can see that the construction time for SVR increases very fast with an larger data set size. 400 training tasks are required to find the best model. This corresponds to a model time of 856 seconds, approximately 14 minutes. Locally weighted linear regression does not require any construction time.

Figure 5.8 shows the model time required for the models constructed using SVR and locally weighted linear regression. One can see that the model time of SVR increases linearly with the size of the data set. The locally weighted linear regression. The time required by locally weighted linear regression remains approximately constant with an increasing data set size. This is expected since the procedure does not change for a larger data set. Note that the higher model time for the first model is also present in this experiment. This was also seen in experiment 7 and was due to slower processing in MATLAB in the first iteration.

5.2.3. CONCLUSION

Experiment 9 was conducted to compare the time required to construct a model for a task with a varying start and goal position. A speed up of around 14 minutes is achieved using locally weighted linear regression. This

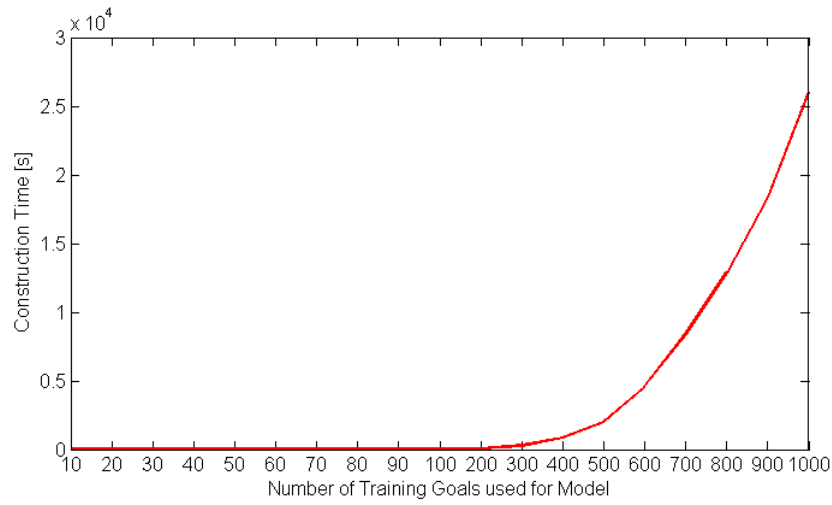


Figure 5.7: Construction Time versus size of the data set used to construct the model for SVR. One can see that the time required to use the SVR model increases exponentially in time. Locally weighted linear regression does not require construction time, therefore a large speed up is achieved in this phase of the algorithm.

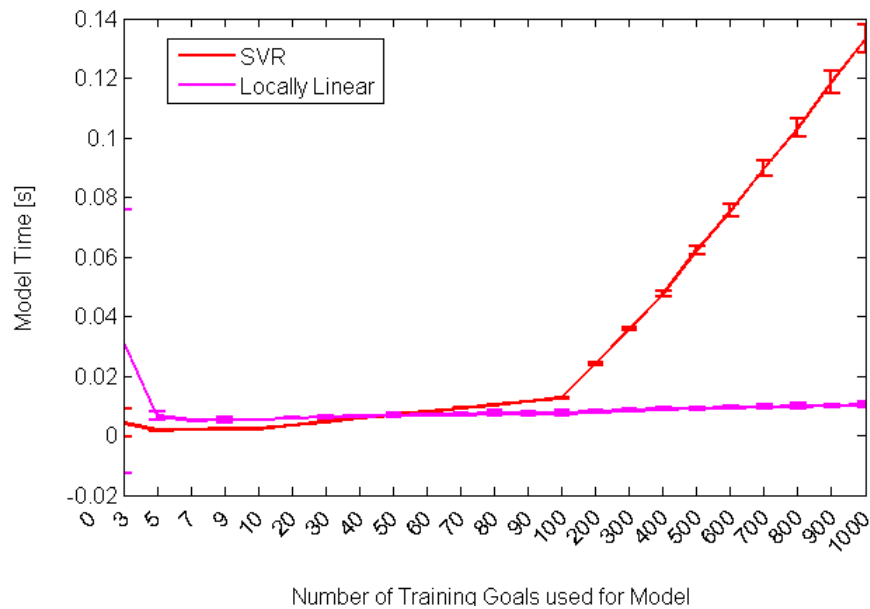


Figure 5.8: Model Time versus size of the data set used to construct the model. One can see that the time required to use the SVR model increases linearly in time.

corresponds to a speed up of 2% of the total time required to construct the algorithm.

This speed up is not yet very significant. However, if the dimension of the task is increased even further, the speed up will be significant, since the time required for SVR increases exponentially, meaning that model time increases with a factor ten if the dimension of the task is increased to five, while the learning time increases linearly. Next to this, it was found that the model time is also decreased if locally weighted linear regression is used. This means that a speed up is also achieved when the controller is used to perform the tasks it has learned.

5.3. SUMMARY

This chapter aims to answer the second subquestion of this study: **Does the performance of Parameterized Skills scale in terms of the data set size required to construct the model with an increased task dimension?** Parameterized Skills has not been applied to learn a task with more than two dimensions. It was checked if the number of training tasks required to construct the model scales when the task dimension is increased, meaning that the required data set size increases quadratically when the task dimension is increased to four. It was also expected that a significant speed up would be achieved when locally weighted linear regression is used instead of SVR.

The performance of a model constructed using twenty training tasks on a task with only a varying goal position was compared to a model constructed using 400 training tasks on a task with a varying start and goal position. It was found that Parameterized Skills scales better with locally weighted linear regression. There was no significant difference between the results and the variance of the results for locally weighted linear regression in the accuracy and used torque. SVR yielded a higher variance and a significantly worse performance in used torque on the task with a varying start and goal position.

A speedup was achieved in the construction time using locally weighted linear regression. SVR required a construction time of 14 minutes, which means that a speed up of 14 minutes is achieved, yielding a speed up of two percent over the construction and learning time. The model time was also higher for SVR, which means that it is also faster to use locally weighted linear regression to use the controller.

6

CONCLUSION

This thesis is conducted to improve Parameterized Skills in such a way that it can be applied to learn tasks with more than two dimensions. This chapter will answer the research question of this thesis, give some limitation of this answer and state some future work. The research question is given below:

Which supervised learning algorithm provides the best performance of Parameterized Skills in terms of accuracy and energy efficiency of the controller found and in terms of required time and when the algorithm is used to learn a more complex task, namely to learn rest to rest motions?

The following two subquestions were derived from the research question:

1. **Which supervised learning algorithm provides the best performance of Parameterized Skills in terms of accuracy and efficiency and in terms of time required if a different supervised learning algorithm than support vector regression is used?**
2. **Does the performance of the supervised learning algorithms scale in terms of the data set size required to construct the model with an increased task dimension?**

Figure 6.1 illustrates the answer to the research question. Locally weighted linear regression performs best over the three criteria of performance, namely accuracy and energy efficiency, required time and scaling to higher dimensions. The performance yielding from the algorithm scales best with an increased dimension, it is the fastest method and its performance is better than the neural network algorithm and polynomial regression, and similar to the performance of support vector regression (SVR).

Polynomial regression did not yield a good fit to the data set at all. The shape of the polynomial yield that high negative or positive values are returned if the training goal was located close to the edge of the goal space. This caused the trajectories for goals at these locations to blow up.

Locally weighted linear regression scales better than SVR when the dimension of the task is increased. The performance and the variance in the performance for locally weighted linear regression are similar when the goal position is varied and when the start and goal position are varied. This is the case when the size of the data set used when the start and goal position are varied is equal to the square of the size of the data set used when the goal position is varied. The SVR model yielded a higher variance and a significant difference in the used torque between the two tasks.

The construction time of SVR increases exponentially with the size of the data set. Therefore, it is only feasible to use SVR when Parameterized Skills is used on tasks with up to four varying dimensions. A speed up of 14 minutes is already achieved on a task with a varying start and goal position on a two degree of freedom robot arm, which is 2% of the total time required to learn the controller.

Many tasks have more than four varying dimensions. For example, when one aims to learn a controller on a robot arm with three degrees of freedom. This would require 8000 training tasks. Then, it is infeasible to use SVR, since the construction time required for a data set of this size is equal to over 100 days. So the speed up achieved by locally weighted linear regression becomes increasingly significant when the complexity of the task increases.

It is found that the performance found using Parameterized Skills is worse than the performance found when

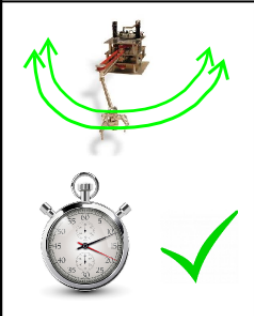
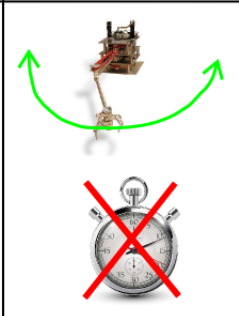
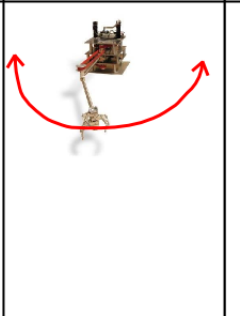

Locally Weighted Linear Regression	SVR	Neural Network Algorithm	Polynomial Regression
			

Figure 6.1: Illustration of the conclusion. A green line indicates that the controller is able to move from any position to any position in the configuration space. The red line indicates that the performance is worse. A red cross indicates that the behavior is very bad on this measure.

using a reinforcement learning algorithm to learn each task separately. However, there is a high advantage gained using Parameterized Skills. It was shown that the number of learning iterations required decreases with around 90% on a task with a varying start and goal position when the parameters found using Parameterized Skills are used as initial parameters in a reinforcement learning algorithm instead of the random initial parameters. This yields that a great speed up is achieved by exploiting the similarities of the tasks if many tasks have to be performed by a robot arm.

6.1. LIMITATIONS

This section will discuss limitation of this answer.

A first limitation of this study was that only one subspace was spanned by the policies on the problem used in this thesis. An additional advantage of locally weighted linear regression might be that it is not necessary to analyse the number of subspaces present in the policy space, as was stated in section 1.4.1. However, it was not possible to verify this, since only one subspace was spanned.

Second, my aim was to compare several supervised learning algorithm. I chose four algorithms that are commonly known and that have a good reputation. However, there might be another supervised learning algorithm that can achieve better performance than the algorithms that were tested.

Third, the algorithm was applied to two problems, which yielded similar performance. However, the applicability of these results towards other problems should be analysed. It is expected that a controller found using locally weighted linear regression will always yield good performance due to the fact that it takes into account only local information. This relates well with the assumption, made by [Da Silva et al.](#), that similar tasks have similar policy parameters. If this assumption is met by the problem that is learned, locally weighted linear regression will be able to find a good output using local training data. Also, the time required to construct and implement the model will not differ significantly due to the simplicity of locally weighted linear regression.

Finally, the answer given to the second subquestion is limited by the time required to construct a model using SVR. Therefore, it was not possible to analyse the performance of SVR using a data set consisting of more than thousand training tasks.

6.2. RECOMMENDATIONS FOR FUTURE WORK

This section introduces recommendations for further research into Parameterized Skills. My aim was to improve the performance by implementing a different supervised learning algorithm. However, further research into other parts of the algorithm might improve the performance further.

Research can be conducted into the parameters that are learned in reinforcement learning. These parameters define the control policy that is executed. Research can be performed into the manifold that is spanned by the optimal policies in the policy space. The smoothness of this manifold defines how well the model that is found using Parameterized Skills can perform.

Secondly, further research should be performed to lower the time required using reinforcement learning.

I already achieved a speed up using locally weighted linear regression instead of SVR. However, the learning time was also very significant, and also increases when the complexity of the task increases. Therefore, it can be very advantageous to investigate methods that can speed up the reinforcement learning process.

Thirdly, Parameterized Skills can be applied to other problems, which are more difficult or more complex. For example, it could be applied to a problem with a varying start and goal state, so including the velocity, as is done by Ezair *et al.* [29]. They designed a trajectory planning algorithm which returns trajectories from any start state to any goal state using a more classic control method that does not require learning.

Learning such a task would double the dimension of the input to the supervised learning algorithms. Therefore, it would not be feasible to use SVR on this task. Also, a different policy representation than a dynamic movement primitive (DMP) should be chosen, since this representation does not allow a nonzero endpoint velocity.

Another application is to learn a spring mechanism to further increase the energy efficiency of the robot arm. A spring mechanism was designed by Plooijs and Wisse [30] for the robot arm that was used in this thesis. The mechanism reduces the required energy on a single, repetitive task using a specific characteristic. It would be interesting to investigate if it is possible to use Parameterized Skills to learn a spring characteristic for every task, such that it is possible to further reduce the energy used by the robot.

BIBLIOGRAPHY

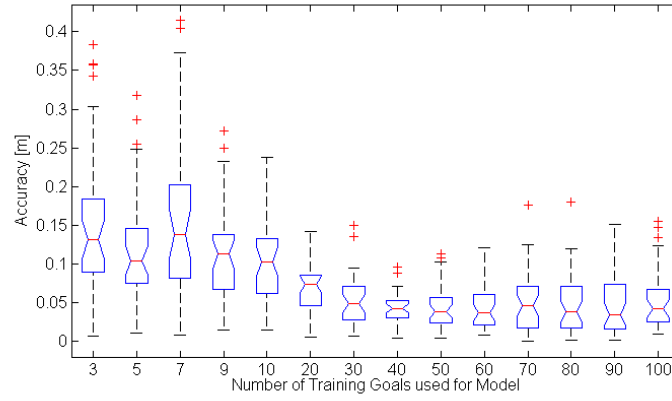
- [1] B. Da Silva, G. Konidaris, and A. Barto, *Learning parametrized skills*, arXiv preprint arXiv:1206.6398 (2012).
- [2] S. Singh, P. Norvig, and D. Cohn, *How to make software agents do the right thing: An introduction to reinforcement learning*, Adaptive Systems Group (1996).
- [3] A. Barto and R. Crites, *Improving elevator performance using reinforcement learning*, Advances in neural information processing systems **8**, 1017 (1996).
- [4] P. Jonker, B. van Driel, J. Kuznetsov, and B. Terwijn, *Algorithmic foundation of the clockwork orange robot soccer team*, in *Algorithmic Foundations of Robotics VI* (Springer, 2005) pp. 17–26.
- [5] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, *The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning*, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2010) pp. 3238–3243.
- [6] M. Wisse, *Factory in a day*, (2013).
- [7] A. Koelewijn, *Learning to move anywhere*, (2014), literature study TU Delft.
- [8] J. Kober and J. Peters, *Policy search for motor primitives in robotics*, Machine Learning **84.1-2**, 171 (2011).
- [9] A. Dempster, N. Laird, and D. Rubin, *Maximum likelihood from incomplete data via em algorithm*, Journal of the Royal Statistical Society, Series B **39.1**, 1 (1977).
- [10] J. Peters and S. Schaal, *Reinforcement learning by reward-weighted regression for operational space control*, Proc. of the 24th Intl Conf on Machine Learning, 745 (2007).
- [11] T. Rückstieß, M. Felder, and J. Schmidhuber, *State-dependent exploration for policy gradient methods*, in *Machine Learning and Knowledge Discovery in Databases* (Springer Berlin Heidelberg, 2008) pp. 234–249.
- [12] P. Dayan and G. Hinton, *Using em for reinforcement learning*, Neural Computation **9.2**, 271 (1997).
- [13] T. Minka, *Expectation-maximization as lower bound maximization*, Tutorial published on the web at <http://www-white.media.mit.edu/tpminka/papers/em.html> (1998).
- [14] A. Smola and B. Schölkopf, *A tutorial on support vector regression*, Statistics and Computing **14**, 199 (2004).
- [15] C. Souza, [Kernel functions for machine learning applications](#), (2010), month visited: April, 2014.
- [16] E. Weisstein, [Least square fitting – polynomial](#), Month visited: April, 2014.
- [17] D. Stronger and P. Stone, *Polynomial regression with automated degree: A function approximator for autonomous agents*, .
- [18] J. Geaghan, *Polynomial regression*, (2005), syllabus of course EXST7034 : Regression Analysis, Department of Experimental Statistics, Louisiana State University.
- [19] R. Chen, *Polynomial regression models*, Institute of Statistics, National University of Kaohsiung.
- [20] C. Stergiou and D. Siganos, [Neural networks](#), Month Visited: June, 2014.
- [21] C. Atkeson, A. Moore, and S. Schaal, *Locally weighted learning for control*, in *Lazy learning* (Springer, 1997) pp. 75–113.

- [22] M. Plooi, M. de Vries, W. Wolfslag, and M. Wisse, *Optimization of feedforward controllers to minimize sensitivity to model inaccuracies*, in *2013 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)* (IEEE, 2013).
- [23] E. Rückert and A. d'Avella, *Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems*, *Frontiers in computational neuroscience* **7** (2013).
- [24] A. Ijspeert, J. Nakanishi, and S. Schaal, *Movement imitation with nonlinear dynamical systems in humanoid robots*, in *Proc. of the ICRA/IEEE Intl. Conf. on Robotics and Automation* (IEEE, 2002) pp. 1398–1403.
- [25] J. Kober and J. Peters, *Learning motor primitives for robotics*, in *IEEE Intl. Conf. on Robotics and Automation* (IEEE, 2009) pp. 2112–2118.
- [26] J. Tenenbaum, V. De Silva, and J. Langford, *A global geometric framework for nonlinear dimensionality reduction*, *Science* **290**, 2319 (2000).
- [27] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, *Dynamical movement primitives: Learning attractor models for motor behaviors*, *Neural computation* **25** (2013).
- [28] T. DeWolf, *Dynamic movement primitives part 1: The basics*, (2013).
- [29] B. Ezair, T. Tassa, and Z. Shiller, *Planning high order trajectories with general initial and final conditions and asymmetric bounds*, *The International Journal of Robotics Research* **33.6**, 898 (2014).
- [30] M. Plooi and M. Wisse, *A novel spring mechanism to reduce energy consumption of robotic arms*, in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (IEEE, 2012) pp. 2901–2908.

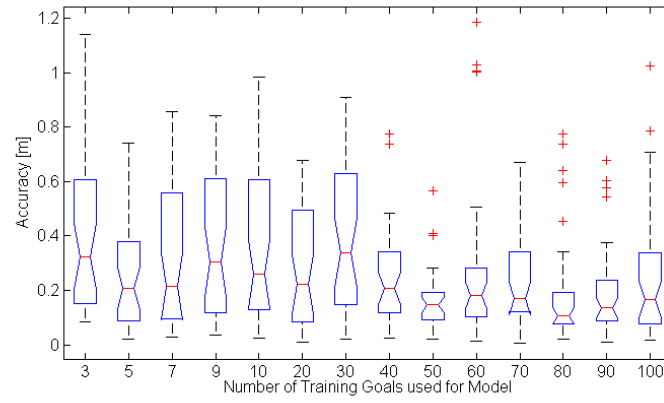
A

BOXPLOTS OF ACCURACY AND DIFFERENCE BETWEEN USED TORQUE AND THRESHOLD OF EXPERIMENT 6

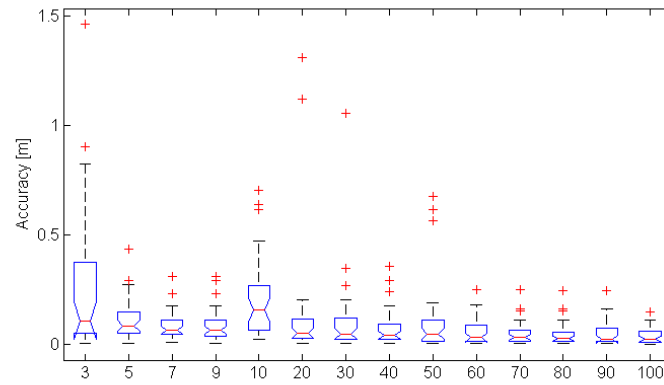
This appendix shows the boxplots of the results on the accuracy and difference between used torque and threshold found in experiment 7. This data is used in the ANOVA tests performed to find the minimum size of the data set in terms of the accuracy and the used torque in section 4.1. Figure A.1 shows the results for the accuracy, while figure A.2 shows the results for the difference between the used torque and the limit.



(a)

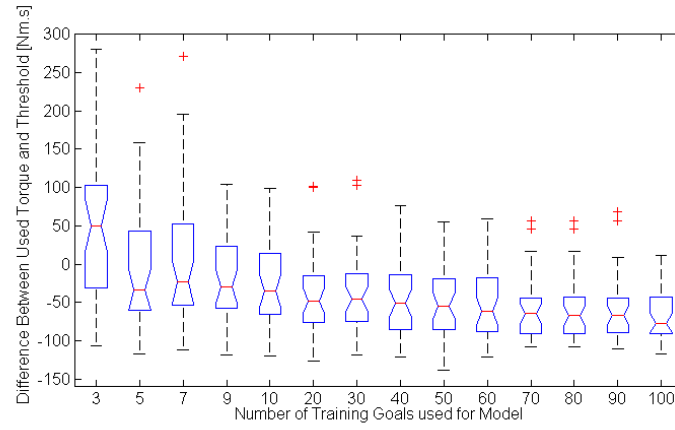


(b)

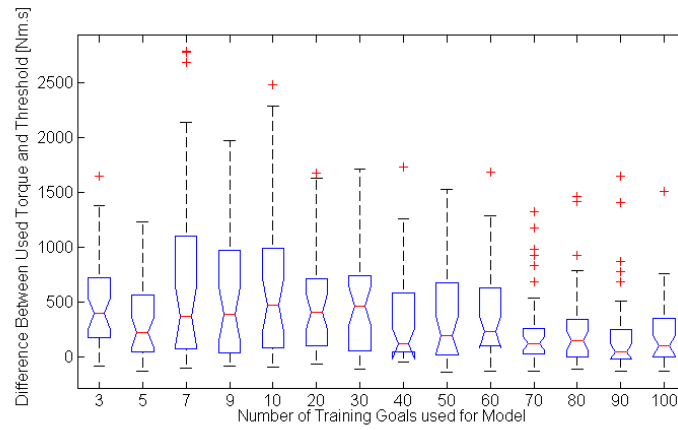


(c)

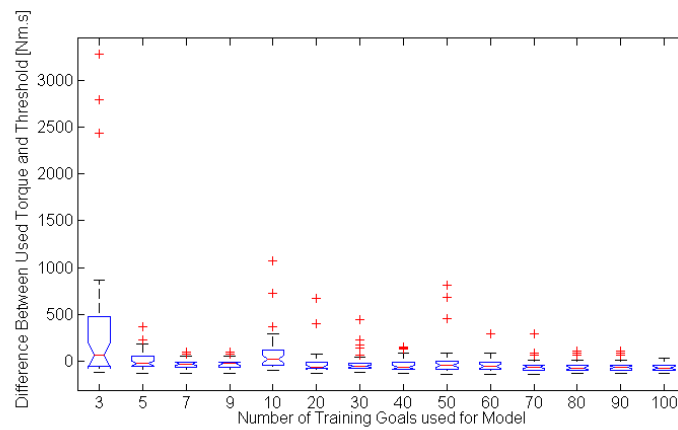
Figure A.1: Boxplot of the accuracy found using SVR (a), the neural network algorithm (b) and locally weighted linear regression (c). It was found that SVR and locally weighted linear regression required 20 training tasks. There was no significant difference between any of the results found for the neural network.



(a)



(b)



(c)

Figure A.2: Boxplot of the difference between the used torque and the threshold yielding from SVR (a), the neural network algorithm (b) and locally weighted linear regression (c). It was found that SVR and locally weighted linear regression require 20 training tasks. There was a no significant difference between the means for the neural network.