Reinforcement Learning for a Dexterous Manipulation Task

Reinforcement Learning für eine geschickte Manipulationsaufgabe
Bachelor-Thesis von Valerian Marg
Tag der Einreichung: 25. August 2016

Gutachten: Prof. Jan Peters
Betreuung: Herke van Hoof

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Reinforcement Learning for a Dexterous Manipulation Task
Reinforcement Learning für eine geschickte Manipulationsaufgabe

Vorgelegte Bachelor-Thesis von Valerian Marg

Gutachten: Prof. Jan Peters
Betreuung: Herke van Hoof

Tag der Einreichung: 25. August 2016

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, August 25, 2016

(Valerian Marg)

Erklärung zur Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, 25. August 2016

(Valerian Marg)

Abstract

In robotics, dexterous grasp- and manipulation tasks for unknown objects are still a major challenge. Methods based on conventional control engineering need accurate models of the object or the environment. If these are not available, those methods are not applicable. Reinforcement learning instead makes use of experience from direct interaction and learns a policy to accomplish a particular task. This thesis presents a method, based on reinforcement learning, with which manipulation primitives can be learned. In this method the task specifications are defined in a reward function in advance. To learn a value function from data *least-square temporal difference learning* (LSTD) is used. The policy is improved in smooth updates with *actor-critic relative entropy policy search* (AC-REPS) that balance exploration and exploitation of experience. This thesis inspects the individual components of the approach in more detail, presents a stabilization task in a realistic experiment and evaluates the method in a simulated robotic setup. The results show that the presented approach is feasible and suitable for the proposed manipulation task.

Zusammenfassung

Im Bereich der Robotik sind komplexe Greif- und Manipulationsaufgaben mit unbekannten Objekten nach wie vor eine große Herausforderung. Herkömmliche Methoden aus der Regelungstechnik benötigen exakte Modelle vom Objekt bzw. der Umwelt. Wenn diese nicht vorhanden sind, können diese Methoden nicht verwendet werden. Reinforcement Learning nutzt hingegen die Erfahrungswerte aus direkter Interaktion und lernt eine Handlungsstrategie um eine spezielle Aufgabe zu erfüllen. Die vorliegende Arbeit stellt eine Methode vor, mit welcher Manipulationsprimitive basierend auf Reinforcement Learning gelernt werden können. In dieser Methode werden die Zielvorgaben der Manipulationsaufgabe in einer Belohnungsfunktion definiert. Um aus den Daten eine Nutzenfunktion zu lernen, wird *least-square temporal difference learning* (LSTD) verwendet. Die Handlungsstrategie wird mit *actor-critic relative entropy policy search* (AC-REPS) in weichen Schritten aktualisiert, wobei zwischen Erkundung und Ausnutzung von Erfahrungswerten abgewägt wird. Diese Arbeit untersucht die einzelnen Bestandteile der Methode, präsentiert eine Stabilisierungsaufgabe in einem realistischen Versuch und bewertet die Methode in einer simulierten Umgebung. Die Ergebnisse zeigen, dass die vorgestellte Methode eine mögliche und brauchbare Lösung für die vorgeschlagene Manipulationsaufgabe bietet.

Contents

1 Introduction **1**
- 1.1 Related Work . . . 2
- 1.2 Overview . . . 3

2 Foundations **4**
- 2.1 Fundamental Principals of Reinforcement Learning . . . 4
- 2.2 State Representation and Reward Function . . . 5
- 2.3 Function Approximation with Radial-Basis Functions . . . 7
- 2.4 Least-Square Methods in Reinforcement Learning . . . 9
- 2.5 Policy Improvement with AC-REPS . . . 12

3 The Methods and Preliminary Studies **15**
- 3.1 The Principal Approach . . . 15
- 3.2 A Closer Inspection of the Methods . . . 16
  - 3.2.1 Impact of Data Size on LSTD . . . 17
  - 3.2.2 Policy Iteration and Learning Progress . . . 18

4 The Experiment and the Results **20**
- 4.1 Platform for Experiments – The Robot Hand . . . 20
- 4.2 Learning a Manipulation Primitive – Stabilization . . . 21
- 4.3 Collecting Samples and Exception Handling . . . 22
- 4.4 Learning the Policy by Simulation . . . 24
- 4.5 Learning Progress and the Results . . . 25

5 Discussion and Outlook **28**

References **30**

Chapter 1

Introduction

Robots will definitely play a large role in future life. In the past fifty years, a remarkable industrial change has taken place. Industrial robots and machine tools allow producing parts in an once impossible accuracy and velocity. In the next time surely highly specialized robots will take their place. These robots can perform those tasks autonomously, precisely and reliably for which they are built for. For instance, recently great progress can be noticed in the area of robot-assisted surgery. But what is definitely still up in the air, are multi-talented robots, capable to interact with their unknown environment – reactive and robust. These robots orient themselves and can perform complicated tasks, for instance, use various tools in a dynamic environment. But grasping and manipulation of objects still rank amongst the most challenging interactions, even though a lot of research is being done in this area.

Several works make use of classical methods for manipulation and craft sophisticated control rules [1][2][3]. Those rules are highly specialized, working only in a well known environment or for a small number of tasks. For this reason reinforcement learning in manipulation is definitely worth further research. Reinforcement learning uses the experience the system gains in action and uses the cumulative knowledge to improve its control strategy. This area of machine learning does not need a model of the environment or the system dynamic, as it takes samples of occurring situations frequently and values them respectively. Moreover, reinforcement learning has the ability to generalize, with the result that the system can take proper actions in previously unknown situations.

1.1 Related Work

In context of robotics and manipulation, complex abilities of grippers, commonly inspired by human hands, are often denoted with the term of dexterous manipulation. For this term there are several definitions [4]. In this thesis the following definition is used: Dexterous manipulation is the capability of changing the position and orientation of an object to another certain different configuration [5]. This capability has also a simple gripper, that is mounted on a robot arm with the respective degrees of freedom. The authors in [4] reason, that in this case the configuration space is restricted by the arm and therefore hand dexterity should be preferred in particular cases, as it increases the workspace. In addition they note, that hand dexterity can increase precision and allows object reorientation within the hand. During manipulation, there are several objectives, that have to be considered. In [6], the manipulation task is categorized hierarchical in low-, mid- and high-level controls, that are interacting with each other. Another perspective for managing complex robotic tasks, especially dexterous manipulation, is the idea of task decomposition into motion primitives. In [7] a taxonomy of different human grasps is introduced and various motion primitives are identified. Two possible primitives, for example, are acquiring an optimal grasp or turning a grasped object about one axis.

An early implementation of a control scheme using motion primitives is demonstrated in [8]. The suggested method utilizes sequences of motor commands and local feedback loops. In this sense, the authors of [9] propose a hardware distributed control structure for manipulation tasks. Here, the task is formulated as a combination of primitive modules and flow-control modules. A method based on skill primitive nets is presented in [10]. This method for task planning, especially assembly planning of industrial robots, considers uncertain environments.

However in an assembly line, there is prior knowledge about the environment, the robot and the manipulated object. Once dexterous manipulation should be possible in an unknown environment, and only with little or no knowledge about the object, additional approaches are required. As already mentioned, reinforcement learning is a promising area of machine learning, that meets those requirements. In the following section a small extract of various works about reinforcement learning for manipulation is given.

The authors of [11] present an approach based on reinforcement learning, focusing on holonomic constraints between the robot and the object. In a simulation they apply the presented method to stabilize an object with one degree of freedom. In [12] reinforcement learning is used to learn primitives – more precisely, policy improvement with path integrals is used to learn

dynamic motion primitives. The authors of [13] deals with model-free reinforcement learning for complex robot tasks, where only little data is available. A robot should be able to react to unexpected situations during manipulations. To handle such situations, a method is proposed in [14] to make the system reactive. The authors suggest a database of trajectories, that can adapt to new environment changes through weighting the trajectories based on learned metrics.

Tactile feedback plays an important role for manipulation tasks, as it offers great opportunities. With additional information from tactile sensors, the robustness and the ability to react can be improved by detecting instabilities, disturbances or slippage [15][16].

1.2 Overview

This thesis presents a method for learning a manipulation task and is structured mainly in three parts. It depends on the basic understanding of reinforcement learning and makes use of several selected methods. Therefore an introduction is given in the first place in Chapter 2, presenting all required basics for the remaining parts. Secondly, in Chapter 3 the principal approach is suggested, joining the prior concepts together. Also, two preliminary tests are conducted to review the suggested method. In the third place, an experiment is presented in Chapter 4, in which a stabilization primitive is learned in practice. In Chapter 5 possible improvements of the proposed method are discussed and the insights are summarized in a brief conclusion.

Chapter 2

Foundations

This thesis is based on the understanding of the background knowledge of reinforcement learning. The relevant basics are covered in this chapter. Section 2.1 indicates the used notations and focuses on the reinforcement learning framework. This section also introduces policy iteration, which provides the basic approach for this thesis. The following section 2.2 explains the general idea behind the proposed state representation. In addition, a generalized template is described to design reward functions for manipulation tasks. To keep the computation feasible, in section 2.3 the principals of function approximation are shown and Gaussian radial basis functions are introduced. The section 2.4 deals with least-square methods in reinforcement learning. It proposes two approaches that are used in this thesis for policy evaluation. Furthermore, this section also focuses on regularization, since it plays an important role in least-square methods. The section 2.5 introduces policy improvement and explains the trade-off between exploration and exploitation. It presents the method that is used for policy improvement with smooth updates.

2.1 Fundamental Principals of Reinforcement Learning

Reinforcement Learning is an area of machine learning, where a strategy for an arbitrary task in unknown environment can be learned autonomously. This strategy includes decision-making on the basis of past interactions with the environment. The whole process is assumed to be time-discrete, more precisely it is modeled as *Markov decision process* (MDP). The learning agent obtains a *reward* R in every step as a feedback, with which it can score his *state* $\boldsymbol{s}$ and *actions* $\boldsymbol{a}$. In every step, the current state transitions into a *next*

state $\boldsymbol{s}'$, which is influenced by the chosen actions and potentially by other effects like noise or external disturbances.

The learning is autonomous, because the actions to be performed to complete a task are not explicitly given. The chosen actions are determined by a *policy* π, which is learned while the agents experience different situations and value them regarding the given reward. A deterministic policy $\pi(\boldsymbol{s})$ is a function that returns particular action $\boldsymbol{a}$ for a given state $\boldsymbol{s}$. Whereas a stochastic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is a probabilistic distribution that describes the probability of possible actions $\boldsymbol{a}$ given a state $\boldsymbol{s}$.

In addition a *value function* $V^\pi(\boldsymbol{s})$ can serve as a basis of decision-making. It quantifies the expected future rewards, following a policy π starting from state $\boldsymbol{s}$. In this thesis only the state-action value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ is used, which is often mentioned as *Q-function*. It also estimates future rewards, while in addition taking actions into account. The correlation between both value functions is given by $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi(R|\boldsymbol{s}, \boldsymbol{a}) + \gamma V^\pi(\boldsymbol{s}')$, where $\mathbb{E}_\pi(R|\boldsymbol{s}, \boldsymbol{a})$ is the expected reward under the policy π and γ a discount factor that is later introduced in more detail.

The goal of learning is to archive a policy that maximizes the total reward in the long run. The concept of *policy iteration* is common in a lot of methods. This iteration is geared towards finding the optimal policy, alternating between *policy evaluation* and *policy improvement*. Both parts cannot be determined at once. Policy improvement is impossible without a sufficient estimate of the value function under the current policy. Since the improvement step changes the policy itself, evaluation has to be done repeatedly. This strategy converges for well-formed learning tasks to the optimal policy. Here the policy evaluation refers to estimating an approximation of the value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$.

What is finally learned depends most notably on the reward. It is the critical factor that prioritizes different goals, rations efforts and sets constraints. In a control task the objectives can be explicitly formulated by designing a reasonable *reward function* $r(\boldsymbol{s}, \boldsymbol{a})$. This approach is used in this thesis and thus the next section covers this topic in more detail.

2.2 State Representation and Reward Function

The state $\boldsymbol{s}$ is a tuple of variables that describes all relevant aspects of the system state. The state variables are given by the system, influenced by the actions the agent does, and normally cannot be set directly.

As this thesis is about performing in-hand manipulations with a robot gripper, reasonable variables can be the fingertip positions. Likewise, other

variables are conceivable, such as joint positions, pressure location and contact normals on the fingertip or even the power consumption of the servo drives. Often such state variables are interdependent. The difficulty lies in finding a state representation that covers all relevant parts in an efficient and sufficiently accurate way. State variables can be scaled or transformed to fit better the learning problem. Furthermore, they can actually be any combination of various other variables[1]. Usually there are many reasonable representations depending heavily on the specific task. This issue is revisited later in Chapter 4, where a concrete task is discussed.

According to the definition in chapter 1, dexterous manipulation is about turning an object to a particular position and orientation. Hence there are configurations more favorable than others and the objective is to reach those or respectively keep them. These configurations depend on the current goal, which can change during the task. A reasonable approach can be the use not only the variables of the current configuration $\boldsymbol{x}$ in the state $\boldsymbol{s}$, but also of context variables $\boldsymbol{x_d}$:

$$\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{x_d})\,. \tag{2.1}$$

Those context variables represent the goal or the desired configuration. Technically it is not a state in the proper sense, since the goal configuration is not necessarily given by the system. For instance it could be set by the user or by a hierarchical control system. In many cases it is not necessary to have all variables modifiable in the end. These can be fixed to a specific value, reducing the learning complexity, since only the specific case is learned. An example for a possible state representation with a fixed goal is given in Figure 2.1.

As mentioned in Section 2.1 the final policy depends notably on the reward. With the previously defined state representation it is simple to design a reasonable reward function. States close to the goal should obtain a higher reward than the states far away. Because of this, the reward function should penalize the difference between the current and desired state variables. In addition a penalty on the actions is reasonable to prevent unnecessarily high actions and to ensure convergence in the end. The following equation is a convenient way of designing the reward function for a manipulation task:

$$r(\boldsymbol{s}, \boldsymbol{a}) = -(\boldsymbol{x} - \boldsymbol{x}_d)^\intercal \boldsymbol{W}_s (\boldsymbol{x} - \boldsymbol{x}_d) - \boldsymbol{a}^\intercal \boldsymbol{W}_a \boldsymbol{a}. \tag{2.2}$$

Both parts of this term can be weighted independently with the weight matrices $\boldsymbol{W}_s$ and $\boldsymbol{W}_a$. Setting values only for the diagonals of the weight matrices is satisfactory in most cases The ratio between the entries of both matrices

[1] An example for preprocessed variables is given in figure [4.1]. The signals of all 19 electrodes in the tactile sensor are used to calculate pressure point and normal.

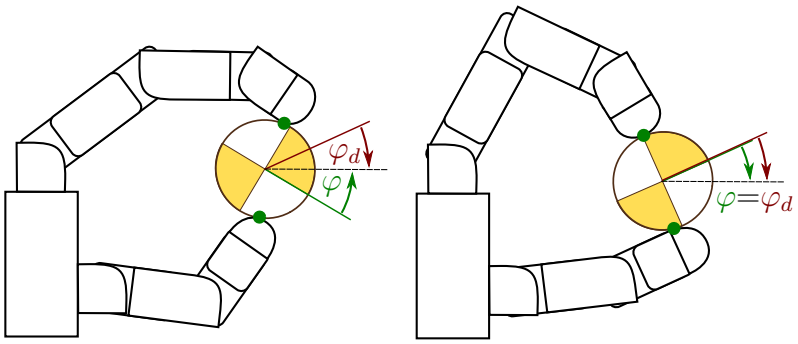


Figure 2.1: Example of a rolling task, where the gripper should roll an object by a certain angle. A possible state representation can be $\boldsymbol{s} = (\varphi, p, \varphi_d, p_d)$. φ denotes the angle and p the total pressure on the finger tips. The angle can be measured by an external sensor for example. To reduce the complexity φ_d can be set to zero and p_d to some empirical value.

have an effect on the final policy. For instance, relatively low weights on the actions can lead to oscillation and noisy behavior.

2.3 Function Approximation with Radial-Basis Functions

The methods, introduced in Section 2.1 involve saving values of the value function for various state-action pairs. Since manipulation tasks have continuous states and actions, it is impossible to save all values for every state-action pair. Even if states and actions are discretized in adequate intervals, the high dimensionality of the state and action space leads to problems. The vast needed memory size to store such Q-function sufficiently is not the only problem. The amount of state-action pairs rises rapidly with the number of dimensions and intervals. Besides the computational complexity, a discretized space leads to impractical discontinuities. Furthermore, the use of data is very inefficient and learning requires relatively large data sets. Every single state-action pair needs at least some samples to learn, but those can only be used for a particular interval. The major problem of discretization is the lack of generalization.

The agent should also handle unknown situations, by considering the experience from related situations.

In consequence of the disadvantages listed above, some kind of gener-

alization is mandatory. More precisely a way of function approximation is needed which matches the target data properly with a feasible amount of parameters. A metric for how close an approximated function matches the target data is the *mean squared error* (MSE):

$$\mathrm{MSE} = \frac{1}{N} \left\| \boldsymbol{y} - \hat{\boldsymbol{y}} \right\|^2, \tag{2.3}$$

where N is the number of observed samples, $\hat{\boldsymbol{y}}$ are observed values of the target function and $\boldsymbol{y}$ are the respective approximated values.

Generally there are many function approximation methods, that can be combined with reinforcement learning. The most commonly used methods in function approximation are linear in the parameters [17]. Those methods are well studied and used extensively in practical applications, due to their advantage of closed form solution. To be exact, for linear methods there is one optimal solution, where the MSE is minimal. The linear function approximation does not need to be linear in the data, but instead it makes use of nonlinear transformations ϕ on the input data $\boldsymbol{x}$ to fit it to a linear model.

The general linear form is given in the following equation:

$$y(x) = \boldsymbol{\phi}(\boldsymbol{x})^\intercal \boldsymbol{\theta} = \sum_{j=1}^{D} \phi_j(\boldsymbol{x}) \theta_j, \tag{2.4}$$

where $\boldsymbol{\phi}(x)$ is a $D{\times}1$ vector, containing the transformed input data $\boldsymbol{x}$, which is denoted from now on as present *features*. The parameter vector is denoted with $\boldsymbol{\theta}$.

There are many different transformations also referred to as *basis functions*. One important group of basis functions are *radial basis functions* (RBF). These functions depend only on the distance from the function center $\boldsymbol{c}$ and are well suited for approximation, because of their easily adjustable smoothness and their powerful convergence properties [18]. In this thesis multivariate Gaussian functions are used that meet the criteria of RBF:

$$\phi_j(x) = \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{c}_j)^\intercal \boldsymbol{\Sigma}_j^{-1} (\boldsymbol{x} - \boldsymbol{c}_j) \right). \tag{2.5}$$

The center, where the function has its maximum, is denoted with $\boldsymbol{c}$. The covariance matrix $\boldsymbol{\Sigma}$ is used to define a bandwidth that determines the shape and size of the features and therefore how much they overlap.

There are several possibilities how to choose the feature centers $\boldsymbol{c}$ and the covariance matrix $\boldsymbol{\Sigma}$. One possibility is spacing the feature centers equally in a grid. The advantage of this method is its simplicity. It just estimates the

boundaries of the data, chooses the number of features for every dimension and generate the grid. In contrast, there is the disadvantage that with the dimension growth the total number of features also grows exponentially. In addition there are cases, in which regions of little interest and with no data points are still covered with features. In these cases, the Q-function is not learned properly for these regions, due to lack of data.

As the ranges of various grid dimensions differ, a relative bandwidth b can be used to ensure a consistent overlapping of the features. In this case all entries of $\mathbf{\Sigma}$ are zero except the diagonal which is $\mathbf{\Sigma} = \text{diag}(\sigma_1^2, \ldots, \sigma_K^2)$, where the elements σ_k^2 are calculated as follows:

$$\sigma_k^2 = \frac{x_k^{\max} - x_k^{\min}}{d_k - 1} b. \tag{2.6}$$

So the absolute bandwidth could be calculated with the boundaries $\boldsymbol{x}^{\max}$ and $\boldsymbol{x}^{\min}$, the number of features d_k in each dimension and the relative bandwidth b.

Another method centers the features on the data points themselves. The same number of features as the data points can be used and the centers of the features are exactly where the data points are. Here there is the advantage that only the space is covered, where the data actually exist. To choose suitable bandwidths $\mathbf{\Sigma}$, for instances, the median distance between all data points to each other can be used. There are also methods to reduce the number of features and optimize the bandwidths directly, but this is not covered in this thesis.

2.4 Least-Square Methods in Reinforcement Learning

A popular approach is to adapt the theory of least-squares function approximation to reinforcement learning. One Example is the so-called *least squares temporal difference* (LSTD) learning, since it targets to minimize a so-called *temporal difference* (TD) error in a least-square sense. The main advantage of LSTD methods in comparison to classical TD(λ) [19] is the efficient use of data. Even though the computation of LSTD is more complex, in most cases LSTD converges faster and better than TD(λ) [20]. Another advantage is the lack of control parameters, such as a learning rate or trace, because these possibly lead to bad results, if not properly chosen. LSTD is generally used off-line, using a previously collected sample set. But there are also methods which can be used on-line, such as the incremental LSTD [21][22][23]. This

thesis presents one way of estimating the Q-function $Q(s, a)$ with least-square methods and derive this method step by step.

2.4.1 Closed Form Solution

The Q-function under the policy π can be defined recursively using the *Bellman function*:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}\left[(Q^{\pi}(s', a')|s, a)\right]. \tag{2.7}$$

Hence the value function equals the reward plus the expected next value of the value function, discounted by a discount factor γ. In doing so, the value function accumulates the total expected future rewards. However, by using a discount factor, it takes a lower account into further away events and thus reduces the effect of more uncertain values. Since the real Q-function is unknown, the TD error δ_{TD} describes the deviations between the current estimate and the real data:

$$\delta_{TD} = Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - \left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma Q^{\pi}(\boldsymbol{s}', \boldsymbol{a}')\right]. \tag{2.8}$$

That error reflects how good a Q-function fits the data. In the end, the Q-function should be approximated linear by a vector of parameters, as it is introduced in Section 2.3:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \approx \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})^{\intercal}\boldsymbol{\theta}. \tag{2.9}$$

Calculation of the Q-function based on a data set is not directly a least-square regression, due to the target being unknown [24]. It is rather finding a fixed point, involving a least-square optimization.

To learn from data, a previously collected set of samples $\mathcal{D}$ is used. Those samples are simply a set of tuples, containing state $\boldsymbol{s}$, action $\boldsymbol{a}$, reward R, next state $\boldsymbol{s}'$ and next action $\boldsymbol{a}'$:

$$\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, R_i, \boldsymbol{s}'_i, \boldsymbol{a}'_i)|i = 1, 2, \ldots, N\}. \tag{2.10}$$

Most methods make no use of the next action $\boldsymbol{a}'$, but here it is used to have implicitly the current policy in a convenient way.

In the first step, an optimal parameter vector $\boldsymbol{u}$ is calculated that approximates the target $\hat{\boldsymbol{y}}$ in the following error function:

$$\text{MSTDE} = \frac{1}{N}\left\|\delta_{TD}\right\|^2 = \frac{1}{N}\left\|\boldsymbol{\Phi}\boldsymbol{u} - \underbrace{(\boldsymbol{R} + \gamma\boldsymbol{\Phi}'\boldsymbol{\theta})}_{\hat{\boldsymbol{y}}}\right\|^2, \tag{2.11}$$

where the $N \times D$ matrix $\boldsymbol{\Phi} = (\boldsymbol{\phi}(\boldsymbol{s}_1, \boldsymbol{a}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_N, \boldsymbol{a}_N))^\intercal$ contains all current state-action features of the data set. The matrix $\boldsymbol{\Phi}'$ contains respectively all next state-action features. The objective in Equation 2.11 is the mean square TD error and is abbreviated as MSTDE in the following. It is not only used for the derivation, but also for evaluating the value function later in Section 3.2.1.

The optimal parameter vector $\boldsymbol{u}^\star$ can be calculated in closed form, by setting the derivative of the MSTDE to $\mathbf{0}$:

$$\tilde{f}(\boldsymbol{\theta}) = \boldsymbol{u}^\star = \arg\min_{\boldsymbol{u}} (\text{MSTDE}) \tag{2.12a}$$

$$\Rightarrow \frac{\partial(\text{MSTDE})}{\partial \boldsymbol{u}} \stackrel{!}{=} \mathbf{0} \tag{2.12b}$$

$$\boldsymbol{u}^\star = \left(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\intercal \left(\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\theta}\right). \tag{2.12c}$$

Afterwards, the fixed point for $\boldsymbol{\theta} = \tilde{f}(\boldsymbol{\theta}) = \boldsymbol{u}^\star$ is determined:

$$\begin{aligned} \boldsymbol{\theta} &= \left(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\intercal \left(\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\theta}\right) \\ &= \underbrace{\left(\boldsymbol{\Phi}^\intercal \left(\boldsymbol{\Phi} - \gamma \boldsymbol{\Phi}'\right)\right)}_{\hat{\boldsymbol{A}}}{}^{-1} \underbrace{\boldsymbol{\Phi}^\intercal \boldsymbol{R}}_{\hat{\boldsymbol{b}}} = \hat{\boldsymbol{A}}^{-1} \hat{\boldsymbol{b}}. \end{aligned} \tag{2.13}$$

2.4.2 Regularization

As in any regression problem, over-fitting can occur. In this case the approximating function fits the given data well, but performs badly on unseen data. In LSTD, such as in supervised learning, regularization can also be used to prevent this. Preventing bad generalization is not the only reason why regularization is reasonable. Calculating $\boldsymbol{\theta}$ in Equation 2.13 requires that $\hat{\boldsymbol{A}}$ is invertible. Especially when the number of samples N is not much greater than the number of features D, chances are high that $\hat{\boldsymbol{A}}$ is ill-conditioned.

Regularization can be done by restricting the magnitude of the parameters. Adding an ℓ_2 penalty on the parameters to the error function is the most simple case. This method is used in this thesis, as it is easily comprehensible and furthermore a closed form solution is still determinable. Including the penalty gives the new error function

$$\left\|\delta_{TD}\right\|^2 = \left\|\boldsymbol{\Phi}\boldsymbol{u} - \left(\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\theta}\right)\right\|^2 + \lambda \left\|\boldsymbol{\theta}\right\|^2 \tag{2.14}$$

with its fixed point

$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^\intercal \left(\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\theta}\right) + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{R} = \left(\hat{\boldsymbol{A}} + \lambda \boldsymbol{I}\right)^{-1} \hat{\boldsymbol{b}}. \tag{2.15}$$

However using ℓ_1 norm is also possible. The ℓ_1 regularization prefers sparse solutions [25] and performs better on learning problems with a high amount of irrelevant features [26]. However, there is no analytical solution in finding the ℓ_1 regularized fixed point, but it is possible to solve it with numerical methods, such as the least angle regression algorithm [27].

Having regularization has great advantages, but it also introduces a new parameter, which has to be set carefully. A simple and commonly used method is applied in this thesis to find an optimal constant λ. The data set is split in a training set $\mathcal{D}^{\text{train}}$ and a testing set $\mathcal{D}^{\text{test}}$. The closed form $\boldsymbol{\theta}(\lambda)$ is used with the training set and inserted in the following error function that makes use of the testing set:

$$\left\| \hat{\boldsymbol{A}}\boldsymbol{\theta}(\lambda) - \hat{\boldsymbol{b}} \right\|^2 . \tag{2.16}$$

Minimizing this error with respect to λ gives the optimal regularization constant regarding to the MSTDE.

Another method to optimize λ, for example, is cross validation. In this case, the data set is split in a number of bins and the optimization is done several times, using different bins as testing set each time. For this reason, cross validation is computational expensive, as it includes multiple optimizations. But the main advantage is its efficient use of data and it can be considered when only little data is available. However, for the reason of additional complexity, computational expense and the sufficient availability of data, in this thesis the previously proposed method is preferred to cross validation.

2.5 Policy Improvement with AC-REPS

While learning, the agent comes upon varying states, taking different actions and receiving corresponding rewards. In the beginning, there is little knowledge about what states are appealing with respect to the reward and what actions lead to them. This knowledge is gained by approaching new situations. Section 2.1 introduces policy iteration that includes the idea of an iterative policy improvement which aims to an optimal policy. Taking always random actions to come upon new situations contradicts the idea of converging to an optimal policy. This policy would try to explore any possible situation, but does not exploit past experiences. On the other hand, taking always the most promising actions is fraught with problems in a learning situation. This strategy is called *greedy policy*, as it always takes the action that maximizes the value function. However, the value function is an estimate, that is determined in the policy evaluation step and is also improving iteratively. Because of this, the greedy policy ignores potentially better, but undiscovered, situations during learning.

Considering the arguments above and following the idea of policy iteration, the agent has to trade off exploration of new situations with exploitation of known appealing situations. There are several modifications of the greedy policy to achieve the trade-off between exploration and exploitation. For instance, the ε-greedy policy, that takes random actions with a probability of ε and otherwise chooses the most promising action. Another modification is the use of softmax action selection. This policy is stochastic, meaning it chooses the action from a distribution with probabilities that are related to how promising an actions is.

This thesis makes use of *actor-critic relative entropy policy search* (AC-REPS) [28], which delivers the above-mentioned distribution in an advanced way. This method provides smooth policy updates and controls the amount of exploration. It belongs to the family of actor-critic methods, which store and use the value function and the policy separately. In such an architecture, the policy is the basis for selecting actions and is therefore denoted as *actor*. The value function is described as *critic*, because it is used to evaluate the current policy [17].

Such a structure has the advantage that it represents the policy directly in parametrized form [29]. Here, the stochastic policy is modeled as a standard distribution. To approximate the mean of the distribution the same methods are used, as introduced in Section 2.3. Therefore, the policy is represented directly. Particularly the parameter vector $\boldsymbol{\theta_\mu}$ describes the mean and the covariance $\boldsymbol{\Sigma}$ the uncertainty of the policy.

$$\pi\left(\boldsymbol{a}|\boldsymbol{s}\right) = \mathcal{N}\left(\boldsymbol{a}|\boldsymbol{\phi}(\boldsymbol{s})\boldsymbol{\theta}_\pi, \boldsymbol{\Sigma}\right) \tag{2.17}$$

In contrast, the greedy policy has to find a maximum of the value function in every step to deliver the most promising action. Having the policy directly in parametrized form, bypasses this computationally expensive optimization. Thus, this form simplifies the policy execution, where the agent just follows the learned policy.

The AC-REPS allows learning the policy in smooth updates by combining two optimizing objectives. On the one hand, this method targets finding an optimal policy with respect to the value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$. But on the other hand, it constrains the policy update. More precisely, the divergence between the observed state-action distribution $q(s, a)$ to the distribution $p(\boldsymbol{s}, \boldsymbol{a}) = \mu(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ is limited by the *Kullback-Leibler* (KL) divergence $\mathrm{KL}(p \parallel q) \leq \epsilon$ [30]. The state distribution $\mu(\boldsymbol{s})$ ensures that the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is learned where data actually exists. With this constraint optimization AC-REPS provides good policies with an appropriate ratio of exploration and

exploitation. Solving it can be done by minimizing the dual function $g(\eta, \boldsymbol{v})$:

$$\min g\left(\eta, \boldsymbol{v}\right) = \epsilon\eta + \boldsymbol{v}^\intercal \bar{\boldsymbol{\phi}} + \eta \log \sum_i \frac{1}{N} \underbrace{\exp\left(\frac{Q^\pi(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{v}^\intercal \phi(\boldsymbol{s}_i)}{\eta}\right)}_{\boldsymbol{w}_i}, \quad (2.18)$$

where η and $\boldsymbol{v}$ are Langrangian multipliers. The vector containing the average of all state features in the samples, is denoted by $\bar{\boldsymbol{\phi}} = \frac{1}{N}\Sigma_{i=1}^{N}\phi(\boldsymbol{s}_i)$. The utilized Q-function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ is determined in the prior policy evaluation step that is described in Section 2.4. With the optimized $\eta^\star$ and $\boldsymbol{v}^\star$ importance weights $\boldsymbol{w} = (w_1, \ldots, w_N)^\intercal$ can be obtained, which specify how the new distribution $p(\boldsymbol{s}, \boldsymbol{a})$ can be obtained from the observed distribution $p(\boldsymbol{s}, \boldsymbol{a})$:

$$p(\boldsymbol{s}, \boldsymbol{a}) \propto q(\boldsymbol{s}, \boldsymbol{a}) w(\boldsymbol{s}, \boldsymbol{a}). \quad (2.19)$$

To determine the parameters $\boldsymbol{\theta}_\mu$ of the new policy, linear regression is used with the sampled actions $\hat{\boldsymbol{Y}} = (\boldsymbol{a}_i, \ldots, \boldsymbol{a}_N)^\intercal$ as target. While $\left(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\intercal \hat{\boldsymbol{Y}}$ would determine the mean of the current policy $q(\boldsymbol{a}|\boldsymbol{s})$ in the data, the improved policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is obtained with the recently calculated importance weights $\boldsymbol{w}$. These weights are applied on the regression problem:

$$\boldsymbol{\theta_\mu} = \left(\boldsymbol{\Phi}^\intercal \boldsymbol{W} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{W} \hat{\boldsymbol{Y}}, \quad (2.20)$$

where the weight matrix $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w}) / \sum_{i=1}^{N} w_i$ contains the normalized weights on its diagonal. In the same way, the covariance $\boldsymbol{\Sigma}$ is obtained from the calculated parameters $\boldsymbol{\theta_\mu}$, using importance weighting:

$$\begin{aligned} \boldsymbol{\Sigma} &= \left(\hat{\boldsymbol{Y}} - \mathbb{E}(\pi)\right)^\intercal \boldsymbol{W} \left(\hat{\boldsymbol{Y}} - \mathbb{E}(\pi)\right) \\ &= \left(\hat{\boldsymbol{Y}} - \boldsymbol{\theta_\mu} \boldsymbol{\Phi}\right)^\intercal \boldsymbol{W} \left(\hat{\boldsymbol{Y}} - \boldsymbol{\theta_\mu} \boldsymbol{\Phi}\right). \end{aligned} \quad (2.21)$$

As the policy iteration is converging to an optimal policy, the covariance diminishes, due to decreasing policy updates. For this reason it can also be used as a metric for the progress of policy iteration.

Chapter 3

The Methods and Preliminary Studies

This chapter focuses on the principal approach that is used and presents two preliminary studies. The first section 3.1 shows the complete approach as the process of policy iteration, using the concepts explained in the previous chapter. In the second Section 3.2 a simple task is proposed that is well suited for further inspections of the methods, due to its simplicity and similarities with the experiment in Chapter 4. While Section 3.2.1 deals with the impact of the amount of training data on LSTD, Section 3.2.2 inspect the whole iteration and shows the learning progress of the proposed task.

3.1 The Principal Approach

In the first instance, a state representation has to be defined. Possibly, some data has to be preprocessed first, to be suitable. The objective of the control task is stated by designing an appropriate reward function. In the end a policy should be learned which enables performing the control task in the intended way. By evaluating collected data, the agent gains knowledge about coherences between states, actions and rewards under current policy that is reflected by the value function. This policy evaluation step uses LSTD to determine parameters $\boldsymbol{\theta}_Q$ that approximates the value function. In the policy improvement step, AC-REPS is used to perform a smooth policy update, utilizing the recently calculated parameters $\boldsymbol{\theta}_Q$. As an actor-critic structure is used, the policy is modeled separately in parametrized form. Here it is assumed to be a normal distribution: $\pi(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\Phi}\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\Sigma})$, where the parameters of the mean $\boldsymbol{\theta}_{\boldsymbol{\mu}}$ and the covariance matrix $\boldsymbol{\Sigma}$ are learned. The improved policy can be used to collect additional data. The past experiences

should be preserved and therefore the newly collected samples are appended to the former set. How this can be done without computational redundant expense, is shown in Section 3.2.2.

Policy evaluation and improvement are repeated, while new data is added, until the policy converges. The covariance matrix $\boldsymbol{\Sigma}$ reflects the uncertainty of the policy. If the uncertainty is very small, the policy will not change that much any more. Because of this, it is suitable as termination criterion for the iteration. The entire process is shown in Figure 3.1.

procedure PolicyIteration
  $\pi \leftarrow \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$
  $\mathcal{D} \leftarrow \emptyset$
  repeat
    $\mathcal{D} \leftarrow \mathcal{D} \cup$ collectSamples (π)
    $\boldsymbol{\theta}_Q \leftarrow$ LSTD $(\mathcal{D})$ ▷ *policy evaluation*
    $\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\Sigma} \leftarrow$ AC-REPS $(\boldsymbol{\theta}_Q)$ ▷ *policy improvement*
    $\pi \leftarrow \mathcal{N}(\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\Sigma})$
  until $\mathrm{tr}(\boldsymbol{\Sigma}) \approx 0$

Figure 3.1: Policy iteration is a process of alternating evaluation and improvement, that targets to find the optimal policy in the data $\mathcal{D}$.

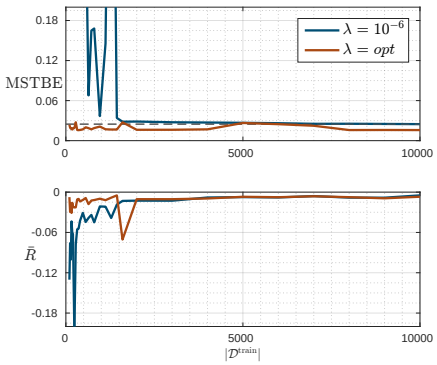
The process of collecting new samples is denoted with collectSamples in the algorithm and is performed in several roll-outs, using different initial states. The first data is obtained without knowledge about the system and so a random policy can be used $\pi(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$. The initial covariance matrix $\boldsymbol{\Sigma}_0$ should be chosen in a way that the policy covers the entire range of possible actions.

3.2 A Closer Inspection of the Methods

To get a feeling for the algorithms and to see how they perform, some small experiments were made, assuming a simple system. The system dynamic is assumed as:

$$\boldsymbol{s}' = \begin{bmatrix} x \\ x_d \end{bmatrix} + \begin{bmatrix} a + \mathcal{N}(0, \sigma^2) \\ 0 \end{bmatrix}. \tag{3.1}$$

It can be interpreted as a joint position x which should reach a desired position x_d taking changes in position as action, neglecting the impact of gravity and inertia. As reward function of the system the formulation in



Experiment Setup	
feature grid	$5 \times 5 \times 5$
discount factor γ	0.7
steps per episode	10
validation set size $\lvert\mathcal{D}^{\text{val}}\rvert$	10^5
state penalty $\boldsymbol{W}_s$	1
action penalty $\boldsymbol{W}_a$	1
range of actions $[\boldsymbol{a}^{\min}, \boldsymbol{a}^{\max}]$	$[-0.2, 0.2]$
range of states $[\boldsymbol{s}^{\min}, \boldsymbol{s}^{\max}]$	$[0, 1] \times [0, 1]$

Figure 3.2: On the left are the results of the learning without regularization – the right side shows the results with optimized λ.

Equation 2.2 is used, as it is suggested in Section 2.2. This dynamic has rather low practical use, but because of its simplicity, it is suitable for further investigations of the policy iteration.

3.2.1 Impact of Data Size on LSTD

In the first experiment the impact of the data size on the quality of the policy evaluation is examined. The direct performance could be measured with the MSTDE, because it is the error minimized by the proposed LSTD algorithm in Section 2.4. In addition, the value function shows the expected reward on the long run. As mentioned in Section 2.5, the greedy policy utilizes the value function directly. In this experiment, new samples are collected by applying the greedy policy directly on the evaluated value function. These new samples could be examined regarding their reward, which is an indicator of how good the policy evaluation works in the first place. The average of all rewards of newly collected samples is used as metric. The size of the training sets is altered while a validation set of $100k$ data points is used to evaluate the MSTDE. In the first run only a small fixed regularization constant was used, to make at least the inversion in LSTD possible. Subsequently, in the second run, the regularization constant was altered by optimizing it with respect to the MSTDE. The optimization was done with a testing set $\mathcal{D}^{\text{test}}$ which size was equal to the according training set $\mathcal{D}^{\text{train}}$. The setup of the experiment and the corresponding results are shown in Figure 3.2.

In conclusion, the results shows that the non-regularized method needs a particular training set size to converge to the minimal MSTDE. In this case, at least a training set size $\lvert\mathcal{D}^{\text{train}}\rvert$ of 1500 samples is needed which

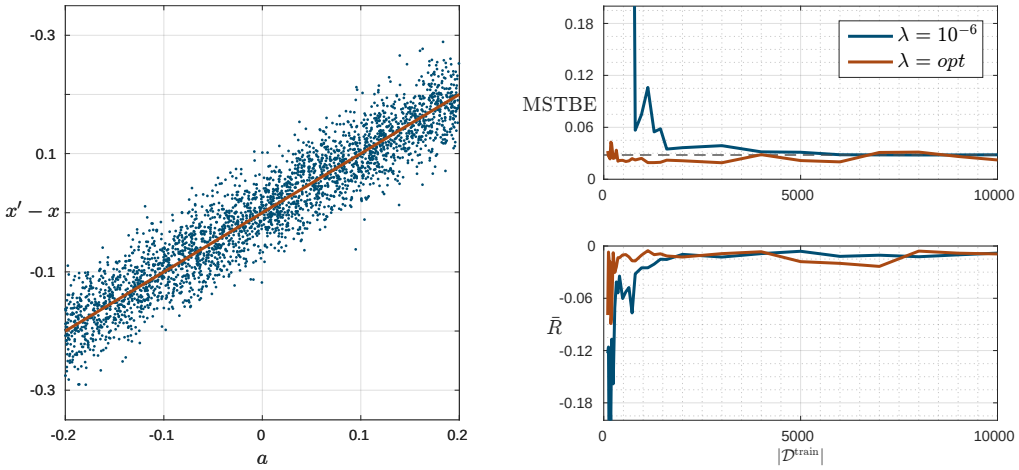
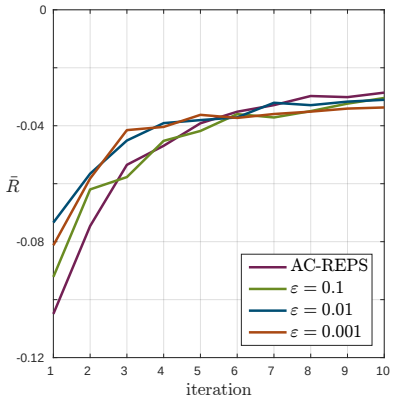


Figure 3.3: The results show no remarkable difference to the experiment without noise in Figure 3.2.

corresponds approximately to the tenfold amount of features $D = 125$. With enough data, this method converges well. In contrast, the regularized method performs well, even with little data. But with increasing data, the results do not get necessarily better. Both, MSTDE and the average reward, are not even better than those obtained using the non-regularized method in all cases. It is to be noted that in this first experiment the action variance σ^2 in Equation 3.1 is zero and hence there is a small chance, that over-fitting occurs. To examine the case where the state transitions of x are affected by noise a second experiment was performed. This experiment has the same setup, but uses a variance of $\sigma^2 = 0.016$. The results shown in Figure 3.3 indicate that little noise in the transitions has no remarkable influence on the learning quality.

3.2.2 Policy Iteration and Learning Progress

The complete approach, as it is presented in Section 3.1, particularly in Figure 3.1, is inspected in the next experiment. The system dynamic and the reward function is the same, as in the experiment before. At the beginning an initial sample set is collected, using a random policy. As policy evaluation method LSTD is used without regularization. In the policy improvement, AC-REPS is compared with the ε-greedy policy, as introduced in Section 2.5. For comparison, in each iteration, the actual policy is evaluated by



Experiment Setup	
feature grid	$3 \times 3 \times 3$
relative bandwidth b	3.0
discount factor γ	0.7
KL divergence bound ϵ (REPS)	0.1
steps per episode	5
training set size $\lvert\mathcal{D}^{\text{train}}\rvert$	30
state penalty $\boldsymbol{W}_s$	1.0
action penalty $\boldsymbol{W}_a$	1.0
range of actions $[\boldsymbol{a}^{\min}, \boldsymbol{a}^{\max}]$	$[-0.2, 0.2]$
range of states $[\boldsymbol{s}^{\min}, \boldsymbol{s}^{\max}]$	$[0, 1] \times [0, 1]$

Figure 3.4: The learning curve shows the progress of the policy improvement, utilizing different methods.

running long episodes[1] and estimating the average reward.

As already mentioned, during the iteration, past experience should be preserved, so that the policy is learned for the entire state space, and not locally for the recently collected data. If new samples are simply appended to the existing training set, the computational expense would grow during the iteration. In LSTD the parameter vector $\boldsymbol{\theta}_Q$ is obtained by solving $\hat{\boldsymbol{A}}^{-1}\hat{\boldsymbol{b}}$, as it is derived in Section 2.4. Because $\hat{\boldsymbol{A}}$ has the dimension $N \times N$, the costs for the containing matrix inversion rise rapidly[2] with the number of samples. To evade this problem, two data structures, $\hat{\boldsymbol{A}}^{\text{train}}$ and $\hat{\boldsymbol{b}}^{train}$, can be stored and updated in each step t, by adding the recently calculated $\hat{\boldsymbol{A}}_t$ and $\hat{\boldsymbol{b}}_t$ [31]:

$$\begin{aligned} \hat{\boldsymbol{A}}_{t+1}^{\text{train}} &= \hat{\boldsymbol{A}}_t^{\text{train}} + \hat{\boldsymbol{A}}_t \\ \hat{\boldsymbol{b}}_{t+1}^{\text{train}} &= \hat{\boldsymbol{b}}_t^{\text{train}} + \hat{\boldsymbol{b}}_t \end{aligned} \tag{3.2}$$

The results and the setup of the experiment are shown in Figure 3.4. To obtain fairly representative results, the experiment was repeated 50 times and the curves where averaged. The experiment shows, that AC-REPS and the ε-greedy policies are converging relatively equally. It is quite possible that the results are substantially different if the parameters are altered, notably the training set size $\mathcal{D}^{\text{train}}$ or the relative bandwidth b. This experiment should rather be seen as a proof of concept, than an in-depth analysis of the policy improvement.

[1] The episodes for evaluating the policy have 50 steps/episode in contrast to 5 steps/episode in the training data.

[2] The Gauss–Jordan elimination has the complexity of $\mathcal{O}(n^3)$.

Chapter 4

The Experiment and the Results

This chapter presents in an experiment how a stabilization primitive can be learned. In Section 4.1 the robot gripper and the tactile sensors are presented which are the platform for the experiment. A description of the task with the proposed state representation and reward function is given in Section 4.2. The Section 4.3 deals with the practical perspective of collecting samples, especially exception handling and feature coverage. In Section 4.4 a strategy to learn an initial policy is suggested. Finally, Section 4.5 presents the results of the learning progress.

4.1 Platform for Experiments – The Robot Hand

As platform for the experimental part of the thesis the so-called Allegro hand is used. This robot gripper is inspired by the human hand and is specially designed for grasping and manipulation research [32]. All 4 fingers are fully actuated, with 4 torque controlled joints each. Furthermore, on the finger tips tactile sensors[1] are attached. Hereby, the fingers can measure the total pressure on the finger tip. This tactile feedback offers great opportunities, as it can improve the grasp, by detecting instabilities, disturbances or slippage [15][16]. It also allows a stable grasp during the manipulation. Instead of learning tasks for one particular object, the tactile feedback allows a certain degree of generalization. Thus, for example a policy can be learned once and subsequently be applied on objects of the same shape, but alternating slightly in size.

Additionally, the sensors have also 19 electrodes on different locations in

[1] BioTac, SynTouch [33]

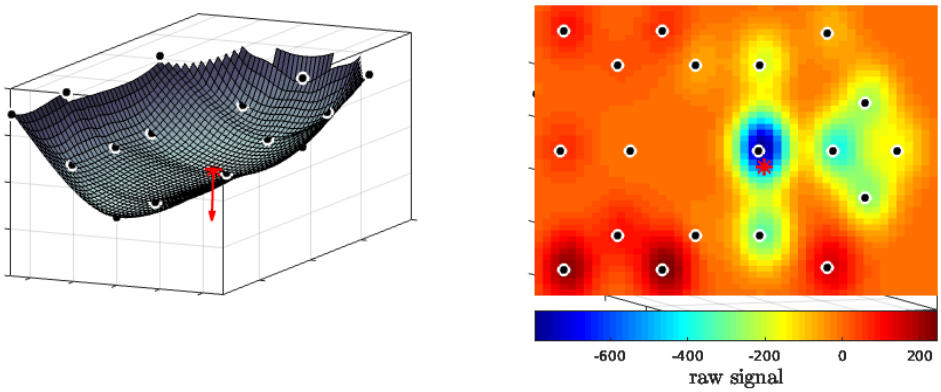


Figure 4.1: The pressure point and the contact normal can be calculated by estimating the center of the electrode signals. But the results are not reliable and therefore not used in the thesis.

the finger tip. With these, changes of the resistance of the conductive fluid under the skin can be detected. With some preprocessing, this data can be used to calculate pressure points and contact normals.

Having this data appears promising, as it can help to detect grasp instabilities or slippage. But some test runs have shown, that the raw electrode signals are not reliable enough for a simple preprocessing. If the pressure point and contact normal are simply calculated by the center of the signal, the results are prone to error and are impractical for learning. Therefore better heuristics or preprocessing strategies are needed to obtain reliable signals. However, this goes beyond the scope of this thesis and thus is not considered onwards.

4.2 Learning a Manipulation Primitive – Stabilization

The goal of this task is to hold an object in a particular pose, respectively keeping the objects in this pose that is defined by a mutable context variable. Such a control primitive is very useful in various situations. For example, this primitive can be applied, when a robot should hand over a full cup of coffee. The stabilization prevents the coffee from spilling out, while the hand follows a certain trajectory.

Before learning the policy, some assumptions have to be made and a prac-

tical state representation has to be chosen. To narrow the problem the task is learned for two dimensional space, meaning that only the x and z coordinate is considered. Moreover, it is assumed, that the manipulated object can be held in a pinch grasp. On the one hand, the pressure of the grasp should never be too low, so that the object slips out of place or even falls down. On the other hand, to protect the sensors or in some cases the manipulated object, the pressure should never be to high. Anyhow, the pressure plays an important role and this is considered in the state representation. The orientation of the object cannot be measured directly without the addition of external sensors. Therefore, it is assumed that the relative tip position to each other, directly affects the object orientation. Accordingly, as reverse conclusion, it is assumed that the object orientation can be estimated by the tip position. As suggested in Section 2.2, a context variable is introduced controlling the tip positions and thus affecting the object orientation. Taking all these things together, the following state representation is proposed:

$$\boldsymbol{s} = (\Delta x, \tilde{p}, \Delta x_d) \tag{4.1}$$

The variable Δx denotes the relative distance in x-direction between the finger tip. The according context variable, the desired relative distance in x-direction, is denoted with Δx_d. The pressure variable $\tilde{p}$ is preprocessed and describes the derivation of the desired grasp pressure of both fingers. The relative distance Δz is not considered in the state representation, as it is already represented indirectly by Δx and $\tilde{p}$. Adding it would limit the generality of the policy to a certain size of the object, which is not wanted here. Changes of the relative finger tip positions Δx and Δz are defined as actions of the two dimensional task. Given this state representation, the suggested reward function is made up as follows:

$$r(\boldsymbol{s}, \boldsymbol{a}) = -\begin{pmatrix} \Delta x - \Delta x_d \\ \tilde{p} \end{pmatrix}^{\intercal} \boldsymbol{W_s} \begin{pmatrix} \Delta x - \Delta x_d \\ \tilde{p} \end{pmatrix} - \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^{\intercal} \boldsymbol{W_a} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \tag{4.2}$$

4.3 Collecting Samples and Exception Handling

To train this task on the robot gripper, some practical problems need to be solved. Most importantly, the tactile sensors have to be protected from damage due to high pressures. Another problem is that during the sampling process the object can fall out of the hand. In this particular case, the agent collects wrong samples, as there is no object any more between the fingertips and thus no tactile feedback. For these reasons, an emergency stop has to be

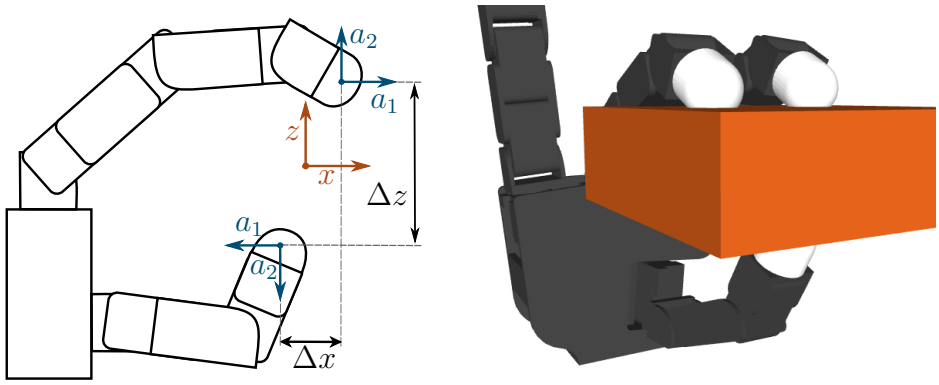


Figure 4.2: The actions change the distance of the finger tips in x- and z-direction. For a better grip, the middle finger mimics the index finger.

implemented for both cases, that is executed if the pressure exceeds a defined threshold.

The sampling is done in several episodes, starting from different initial states. This is necessary to obtain samples for all regions of the state-action space. After a defined number of steps the episode ends. To start a new episode, the gripper loosens the grip and moves the finger tips to new arbitrary positions in the workspace. If the fingers exit the workspace, an episode should stop earlier. In the case of leaving the workspace or emergency stop, the last sample should be considered with high costs. Otherwise, this case can be interpreted as too promising, because it has no following transitions and thus it does not regard additional future costs.

As already mentioned the state variables have to be transformed. This step is important mainly to scale roughly the state variables to the same range. Highly diverging magnitudes in the values lead to problems in the policy iteration, particular in regression and regularization. To avoid these problems, the range of the state variables is rescaled to the range between -1 and 1. Another reason for the preprocessing is that not all regions of the samples are equally important. For instance, a pressure signal of 200 is very undesirable, if the defined desired pressure is 100. A pressure signal of 250 would be even worse, but it is not so important to distinguish between both cases, because both cases are highly undesirable and thus are considered with a low reward accordingly. Figure 4.3 shows the utilized preprocessing of the average pressure signal of both involved finger tips. Having this preprocessed state variable $\tilde{p}$, the equally spaced features are concentrated in the region

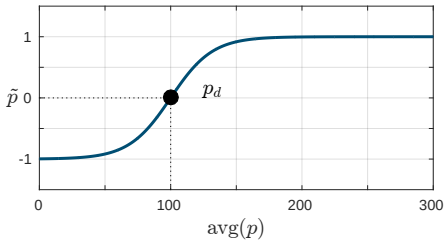


Figure 4.3: The average pressure signal of both finger tips avg(p) is transformed to the state variable $\tilde{p} = \tanh\left(\frac{\pi}{p_d}(\text{avg}(p) - p_d)\right)$.

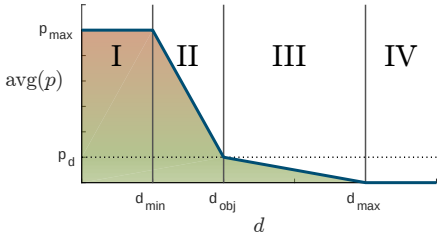


Figure 4.4: The simulated pressure signal avg(p) is related to the distance d between the tips of thumb and index finger.

around the defined desired pressure p_d.

4.4 Learning the Policy by Simulation

Learning on the real hand without any initial policy is tricky and tedious. With no proper policy the agent takes random actions, without any strategy. Because of this, the agents would often violate the pressure limits and get consistently the object out of hand. In this case, the object has to be handed back manually, which is a tedious procedure. It would be possible that another robot helps during the learning and places the dropped object precisely back. But to avoid that effort another method is tried out: An initial policy is learned in a simulation and then in a second step, this policy is improved in the real situation.

The software framework of the robot hand can be used, including the inverse kinematic controller, to simulate a simple training scenario without the need of the real gripper. But to learn the initial policy, the tactile feedback has also to be simulated in some way. The goal of the simulation is, that the resulting policy is better than picking random actions. In other words, the gripper should drop the object or cancel the episode for other reasons less often. For this reason, the simulation has not to model the reality absolutely exactly. It is sufficient to achieve a better policy.

In this experiment several assumptions were made to construct a simple pressure signal. A particular object width is simulated, but the resulting policy fits various widths. It is assumed, that the pressure signal is directly related to the distance of the finger tips. Figure 4.4 shows a possible function of the distance, that is used for this experiment. This proposed function

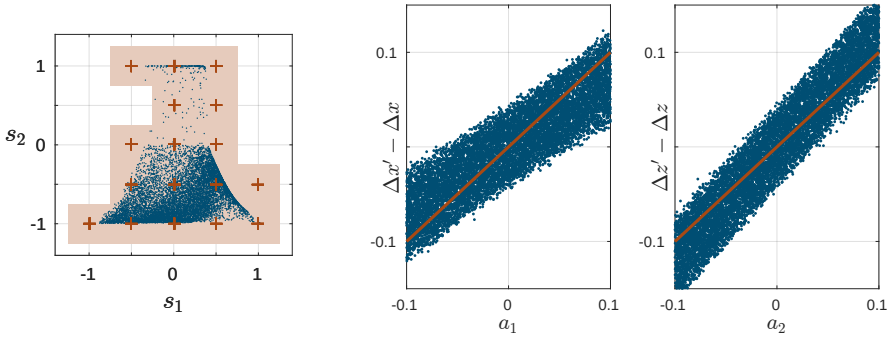


Figure 4.5: On the right side a reduced grid is shown which has only feature centers where data actually exists. The sampled data is indicated as blue dots and the feature centers as orange crosses. The figure on the left shows the state transitions that are noisy and biased.

relates only on the distance and is linear in four segments. In segment I the distance is a certain amount smaller than the object width and therefore the pressure signal is at its maximum. In this case the limits are exceeded and the episode is canceled. Such a region should never be reached with a proper policy. The optimal pressure is located between segment II and III. In segment IV the fingers lose contact to the object and the episode is canceled.

A survey of the samples shows that if all features are spaced equally in a grid, it is not always possible to get all of them covered by data. This case happens on the $s_1 s_2$ plane, as it is shown in Figure 4.5. A simple workaround is reducing the grid by taking out features where no data exists. Figure 4.5 also shows that the state transitions in the samples are noisy and biased.

4.5 Learning Progress and the Results

The result of the initial learning step is shown in Figure 4.6. It turns out largely as expected. High actions are taken to achieve a grasp with the desired pressure on the tips. The taken actions also aim to reach the relative desired distance that is set by the context variable. If the gripper has grasped the object, the actual states are altering only in a narrow region around the desired pressure. In this region, the actions point to the goal position, but it could be observed that they are decreasing as they approach the goal position. This behavior is caused by the quadratic formulation of the reward function. One possible solution might be adding an extra penalty on the difference $\|\Delta x - \Delta x_d\|$, that increases over time.

In a second step the policy is inspected by running it in the kinematic

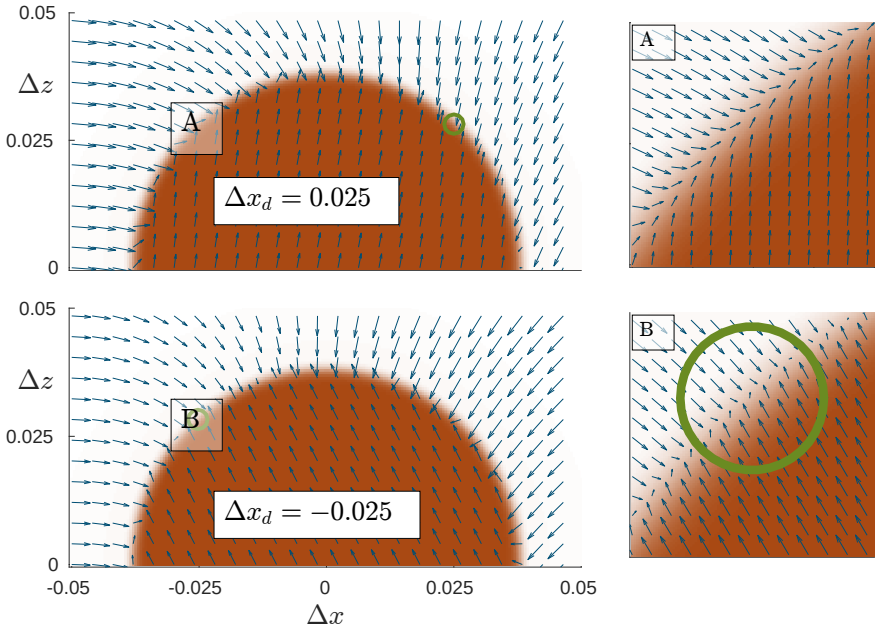
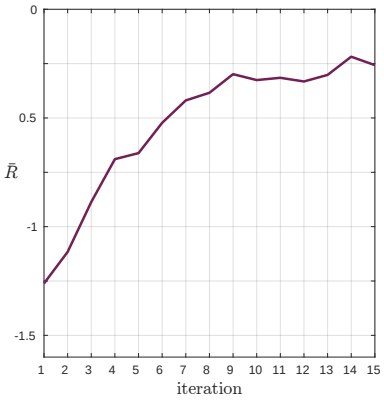


Figure 4.6: The visualization of the policy shows a red region of high pressure. The policy takes actions that aim for a desired pressure and a desired relative distance Δx_d.



Experiment Setup	
state feature grid	$5 \times 5 \times 3$
action feature grid	3×3
relative bandwidth b	3.0
discount factor γ	0.7
KL divergence bound ϵ (REPS)	0.1
steps per episode	100
training set size $\lvert\mathcal{D}^{\text{train}}\rvert$	2000
state penalty $\boldsymbol{W}_s$	1
action penalty $\boldsymbol{W}_a$	100
range of actions $[\boldsymbol{a}^{\min}, \boldsymbol{a}^{\max}]$	$[-0.1, 0.1]$
range of states $[\boldsymbol{s}^{\min}, \boldsymbol{s}^{\max}]$	$[0, 1]^3$

Figure 4.7: The learning curve shows that there is progress in the policy improvement with respect to the average reward.

simulation and see how actually the policy execution looks like. It shows that the gripper approaches the desired pressure relatively fast, but reaching the goal position proceeds rather sluggish, while the fingers are moving wobbly. This behavior cannot be observed in the policy itself and is probably caused by the one-step training. However, the learned policy takes by tendency the correct actions to reach the given goal and hence it is vastly better than the random policy.

In contrast, the situation on the real hand is more difficult. It was not possible to apply the policy on the real system. The fingers consistently showed an unstable control behavior and began oscillating. A possible reason might be the small actions, that trigger self-excited oscillations, due to the stick-slip phenomenon [34] and high controller gains. Although some attempts were made to solve these problems, no proper solution could be found. For instance, a gravity compensation was combined with the PD-controller. Another approach was the use of linear interpolated trajectories as actions instead of single steps. The implementation of a robust task-space position controller goes beyond the scope of this thesis and therefore no further investigations were made on this issue.

For the reason that the experiments could not be applied on the real gripper, the policy iteration was continued, not as intended, with the kinematic simulation. The results of the learning progress is shown in Figure 4.7. Collecting one sample in the used simulation framework needs approximately one second. For this reason, the sampling is done with a fixed $\Delta x_d = 0$, reducing the amount of needed samples drastically and thus making the iteration possible in appropriate time. The results show that the policy iteration converges after approximately 10 iterations which corresponds to the learning from 20000 samples.

Chapter 5

Discussion and Outlook

This thesis presents a method, based on reinforcement learning, with which dexterous manipulation primitives can be learned. More precisely, in an iterative learning process a policy is learned to fulfill the task, previously specified in the reward function. In this thesis, also individual components of the approach are inspected in preliminary studies. A stabilization task is presented in a realistic experiment and the applied method is evaluated in a simulated robotic setup. The results of the preliminary studies and the experiment show that the presented method is feasible and suitable for the proposed manipulation task. Most of the concepts in this approach are intuitive and readily understandable. The computed results look decent: The learned policy takes proper actions to reach the intended objective.

However, in detail, the presented approach needs much fine tuning. A lot of parameters have to be chosen that affect the results substantially. The presented approach can certainly be improved in many respects. For instance, the hand-tuned weighting of the reward function plays a huge role in how the final policy looks like. An unbalanced weighting leads to poor results and finding a satisfying policy is rather a trial-and-error method. One possible solution might be an automatic rating of the policy under particular metrics and quality measures. Then, the weights could be adjusted until a policy is found that meets the requirements. However such requirements are actually another formulation of reward and demand again human intervention. As future work, preference-based reinforcement learning could be investigated. This approach utilizes comparisons of trajectories instead of a reward function.

Another aspect is the use of feedback in the experiment: The presented experiment considers only few data of the environment, such as the tip position and the average pressure signal. Generally, it is desirably to obtain more information, whenever it is useful for the task. As already mentioned, the

tactile sensors can provide much more information than the plain pressure signal. These capabilities were not exploited, to preserve the feasibility in the scope of this thesis and a clear survey of the proposed method. However, to achieve a reactive and robust behavior in manipulation, exploiting the full capabilities of the tactile sensors is valuable and preferable.

If additional data expand the state representation, the proposed method leads to problems. With the number of states the number of features grows exponentially in the equally spaced feature grid and with these also the computational expense. Likewise, it is not guaranteed, that the features fit actual data. Therefore, the presented function approximation is widely impractical for larger state representations. More advanced methods for function approximation should be taken into consideration. For example, non-parametric methods, such as Gaussian processes could be used.

To sum up all in a conclusion, it can be stated that reinforcement learning is definitely a key technology to new control systems, especially for complex systems and unknown models. These qualities unlock a potential for material improvements – also in the challenging domain of dexterous manipulation. So one may be curious what the future brings.

References

[1] L. Han and J. C. Trinkle, "Dextrous manipulation by rolling and finger gaiting," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, pp. 730–735 vol.1, May 1998.

[2] Z. Doulgeri and L. Droukas, "On rolling contact motion by robotic fingers via prescribed performance control," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 3976–3981, May 2013.

[3] M. Cherif and K. K. Gupta, "Planning quasi-static fingertip manipulations for reconfiguring objects," *IEEE Transactions on Robotics and Automation*, vol. 15, pp. 837–848, Oct 1999.

[4] R. R. Ma and A. M. Dollar, "On dexterity and dexterous manipulation," in *Advanced Robotics (ICAR), 2011 15th International Conference on*, pp. 1–7, June 2011.

[5] A. Bicchi, "Hands for dexterous manipulation and robust grasping: a difficult road toward simplicity," *IEEE Transactions on Robotics and Automation*, vol. 16, pp. 652–662, Dec 2000.

[6] A. M. Okamura, N. Smaby, and M. R. Cutkosky, "An overview of dexterous manipulation," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 255–262 vol.1, 2000.

[7] P. Wright, J. Demmel, and M. Nagurka, "The dexterity of manufacturing hands," *Robotics Research, DSC*, vol. 14, pp. 157–163, 1989.

[8] T. H. Speeter, "Primitive based control of the Utah/MIT dextrous hand," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 866–877 vol.1, Apr 1991.

[9] Y.-J. Cho, J.-M. Park, J. Park, S.-R. Oh, and C. W. Lee, "A control architecture to achieve manipulation task goals for a humanoid robot," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, pp. 206–212 vol.1, May 1998.

[10] U. Thomas, B. Finkemeyer, T. Kroger, and F. M. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, pp. 3069–3075 vol.3, Sept 2003.

[11] Y. Kobayashi, H. Fujii, and S. Hosoe, "Reinforcement learning for manipulation using constraint between object and robot," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 871–876 Vol. 1, Oct 2005.

[12] F. Stulp, E. Theodorou, M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning motion primitive goals for robust manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 325–331, Sept 2011.

[13] B. Bischoff, D. Nguyen-Tuong, H. van Hoof, A. McHutchon, C. E. Rasmussen, A. Knoll, J. Peters, and M. P. Deisenroth, "Policy search for learning robot control using sparse data," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3882–3887, May 2014.

[14] S. Hangl, E. Ugur, S. Szedmak, J. Piater, and A. Ude, "Reactive, task-specific object manipulation by metric reinforcement learning," in *Advanced Robotics (ICAR), 2015 International Conference on*, pp. 557–564, July 2015.

[15] Y. Bekiroglu, J. Laaksonen, J. A. Jorgensen, V. Kyrki, and D. Kragic, "Assessing grasp stability based on learning and haptic data," *IEEE Transactions on Robotics*, vol. 27, pp. 616–629, June 2011.

[16] M. Li, Y. Bekiroglu, D. Kragic, and A. Billard, "Learning of grasp adaptation through experience and tactile sensing," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3339–3346, Sept 2014.

[17] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[18] M. Buhmann, *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2003.

[19] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[20] S. J. Bradtke and A. G. Barto, *Linear Least-Squares Algorithms for Temporal Difference Learning*, pp. 33–57. Boston, MA: Springer US, 1996.

[21] A. Nedić and D. P. Bertsekas, "Least squares policy evaluation algorithms with linear function approximation," *Discrete Event Dynamic Systems*, vol. 13, no. 1, pp. 79–110, 2003.

[22] A. Geramifard, M. Bowling, and R. S. Sutton, "Incremental least-squares temporal difference learning," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, p. 356, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[23] C.-G. Li, M. Wang, and S.-H. Yang, "Incremental least squares policy iteration in reinforcement learning for control," in *2008 International Conference on Machine Learning and Cybernetics*, vol. 4, pp. 2010–2014, July 2008.

[24] C. Dann, G. Neumann, and J. Peters, "Policy evaluation with temporal differences: A survey and comparison," no. March, pp. 809–883, 2014.

[25] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[26] M. W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos, *Regularized Least Squares Temporal Difference Learning with Nested ℓ_2 and ℓ_1 Penalization*, pp. 102–114. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[27] J. Z. Kolter and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[28] C. Wirth, J. Fürnkranz, and N. G., "Model-free preference-based reinforcement learning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

[29] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 1291–1307, Nov 2012.

[30] J. Peters, K. Muelling, and Y. Altun, "Relative entropy policy search," in *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, 2010.

[31] M. G. Lagoudakis, R. Parr, and M. L. Littman, *Least-Squares Methods in Reinforcement Learning for Control*, pp. 249–260. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[32] Wonik Robotics, "Allegro Hand – product description and specifications." [`http://www.wonikrobotics.com/Allegro-Hand.htm`; accessed July 11, 2016].

[33] SynTouch, "BioTac – product description and manual." [`http://www.syntouchllc.com/Products/BioTac/`; accessed July 11, 2016].

[34] G. Capone, V. D'agostino, S. d. Valle, and D. Guida, "Stick-slip instability analysis," *Meccanica*, vol. 27, no. 2, pp. 111–118, 1992.