Reinforcement Learning for tactile-based finger gaiting

Selbstverstärkendes Lernen von Griffwechseln unter Nutzung von Tastinformationen

Bachelor-Thesis von Lena Mareike Plage aus Wiesbaden

Dezember 2016

TECHNISCHE UNIVERSITÄT DARMSTADT

Reinforcement Learning for tactile-based finger gaiting
Selbstverstärkendes Lernen von Griffwechseln unter Nutzung von Tastinformationen

Vorgelegte Bachelor-Thesis von Lena Mareike Plage aus Wiesbaden

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Daniel Tanneberg

Tag der Einreichung: 23.12.2016

Erklärung zur Bachelor-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, den 23. Dezember 2016

(Lena Mareike Plage)

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, December 23, 2016

(Lena Mareike Plage)

Abstract

Improving robotic in-hand manipulation based on tactile feedback will open new possibilities to use robots. One basic ability needed in in-hand manipulation is finger gaiting. Finger gaiting is the change of a grasp while the object is kept in its orientation and position. In this thesis, the simulated robot will learn to change from a two finger grasp to a three finger grasp by using tactile feedback.
To learn the task the Parameter-exploring policy gradients (PGPE) algorithm is used. This reinforcement learning algorithm is based on policy gradient methods. It samples N rollouts with different parameters for the controller in every episode, before the policy is improved. Tactile feedback and joint positions are used for the reward function and the state representation.
The features will be generated as Random Fourier Features, which avoids hand-crafting them. Random Fourier Features use a kernel approximation where the state is projected to a one dimensional space and D of these random projections are concatenated for a higher accuracy of the kernel approximation. In this thesis, two methods to use this features in a linear controller will be compared. In one method the state representation is projected and multiplied with the parameters. In the other method, parameters will be projected as well and then be multiplied with the Random Features. It is shown, that direct parameter learning is problematic due to a reward race to the bottom caused by parameter oscillation and choosing too high actions. Projecting parameters vanishes the problem of too high actions because the projection limits the range of the actions. It will be shown, that using a high D and therefore a more accurate kernel approximation leads to better rewards. Additionally, the influence of the state representation and the number of histories used per episode in PGPE will be analyzed. Another advantage of projecting the parameters makes it possible to use a high value for D while only a comparable small number of parameters must be learned.

Zusammenfassung

Neue Möglichkeiten für den Einsatz von Robotern werden sich öffnen, wenn sich die Fähigkeit von Robotern, Dinge in der Hand neu auszurichten, verbessert. Eine Grundfähigkeit für das Manipulieren von Gegenständen in der Hand sind Griffwechsel. Bei Griffwechseln wird der Griff geändert, ohne die Orientierung oder Position des Objekts zu verändern. In diese Thesis wird ein Simulierter Roboter mit Hilfe von Tastinformationen lernen, von einem Griff mit Zeigefinger und Daumen zu einem Griff mit Zeigefinger, Mittelfinger und Daumen zu wechseln.
Um diese Aufgabe zu erlenen wird der “Parameter-expolring policy gradients“-Algorithmus (PGPE) verwendet. Dieser selbstverstärkender Lernalgorithmus basiert auf Policy-Gradient-Methoden. In jeder Episode werden in N Versuchen unterschiedliche Parametern für die Steuerungseinheit ausprobiert, bevor die Entscheidungsregeln verbessert werden. Tastinformationen und die Auslenkung der Gelenke werden für die Bewertungsfunktion und die Zustandsrepräsentation verwendet.
Die Eigenschaften (Features) werden mit als Random Fourier Features generiert, womit man die manuelle Definition von Eigenschaften umgeht. Random Fourier Features basieren auf einer Kernel-Approximation, bei der der Zustand in einen eindimensionalen Raum projiziert wird und D solcher Projektionen zu einem Eigenschaftsvektor verkettet werden, um eine höhere Genauigkeit für die Approximation zu erhalten. In dieser Thesis werden zwei Methoden verglichen, wie man diese Eigenschaftsvektoren in einer linearen Steuerungseinheit nutzen kann. In der einen Methode wird die Zustandsrepräsentation projiziert und mit den Parametern multipliziert. In der anderen Methode werden die Parameter ebenfalls projiziert und dann mit den Random Featurs multipliziert. Es wird gezeigt, dass das direkte Lernen von Parametern anfällig für einen selbstverstärkenden Rückgang der Bewertung ist, der durch Parameteroszillationen und zu große Aktionen verursacht wird. Das Problem zu großer Aktionen wird durch die Projektion der Parameter behoben, da sie in ein beschränktes Intervall projiziert werden. Es wird dargestellt, dass die Wahl eines großen D, und somit einer genaueren Kernelapproximation, zu einer besseren Bewertung führt. Weiterhin wird der Einfluss der Zustandsrepräsentation und der Anzahl der Versuche je Episode in PGPE auf die Bewertung untersucht. Ein weitere Vorteil der Parameterprojektionsmethode ist, dass man mit vielen Features arbeiten kann, während man lediglich eine verhältnismäßig geringe Anzahl von Parametern lernen muss.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
D	Number of concatenated projections used for the Random Fourier Features. This is equal to the number of used features.
DOF	Degree of freedom
DP	Dynamic Programming
i.i.d.	independently and identically distributed
MDP	Markov Decision Process
N	Number of histories sampled per episode. In the case of the symmetric PGPE the number of history pairs.
PGPE	Policy gradients with parameter-based exploration

List of Symbols

Notation	Description
$\boldsymbol{\theta}$	parameter vector of the policy

List of Operators

Notation	Description	Operator
$\langle \bullet, \bullet \rangle$	inner product	$\langle \bullet, \bullet \rangle$
ln	the natural logarithm	$\ln(\bullet)$

1 Introduction

The human hand is a complex and multi-functional manipulator. Fingers can be placed and replaced on an item with an accuracy of less than an millimeter in tasks like playing a violin and they can be used for object manipulations such as folding origami or repairing a watch. On the other side tasks requiring high forces can be done as well. The hand can hold a climbing human or give a save grasp for a tool like a hammer. With human tactile senses blind people can read and many tasks of daily life can be done without visual supervision.
Robots with increasing fine manipulation abilities and an increasing number of joints for their hands were build. This agility allows to fulfill in-hand manipulation tasks with them. The drawback of the increasing number of degrees of freedom is the more complex control. Learning skills is therefore suggested as a good solution to face this complexity [2].
While for humans tactile sensing is natural, robots did not have this possibility for a long time. They were equipped with force or proximity sensors for a while, but compared to human sensing many information is missing. Feeling the temperature, the slippage, the surface structure and the shape cannot be compared to the information a force vector provides. Providing robots with these skills means giving them the possibility to know where they hold an item between their fingers. For example holding a screw, they would know how the screw is orientated between their fingers, whether the hand needs to be turned to place it or if the screw is already in the desired orientation. Additionally, it would notice which part of the screw is hold between the fingers . If tactile information is combined with a fine manipulation task, the robot will become able to turn the screw between its fingers into the right position to drive it in. Manipulating fragile objects will be possible with tactile sensing and in-hand manipulation. A strawberry can be picked with an enclosing three finger grasp with adding exactly the amount of force that is necessary to pick it without damaging it and change to a two finger grasp to place it in a basket without damaging surrounding fruits.

1.1 Motivation

Although the research on robotic in-hand manipulation increased during the last years, it remains a challenging task. In robotic in-hand manipulation a robot holds an object between different fingers and changes the object's orientation or position relative to the hand. In some situation, for example when rolling an item between the fingers, joints of the finger will reach their limit and need to be replaced to continue the task. This finger replacing is called finger gaiting. In some cases it can be necessary to use an additional finger to hold the object while a finger is replaced. This problem is very general and occurs in different tasks with different objects. Many existing methods for finger gaiting often rely on models of the object in the hand and give object specific finger gaiting motions. In interactions with unknown objects this is not possible. Learning methods were successfully and frequently implemented for robot grasping as well as for tactile-based object classification tasks. In this thesis a robot will learn finger gaiting with reinforcement learning which only uses a demonstration of the desired new grip for the reward function and will learn to change the grip based on tactile feedback provided by fingertip sensors.

Improving the in-hand manipulation skills of robots will enable them to fulfill filigree tasks and open new possibilities to use them. For example grasping a screw and autonomously reorienting it between the finger tips to place it at its destination. Visual feedback will not be useful in this task if the screw is small and most of it is covered by the fingers. Using reinforcement learning to teach a robot means that the robot learns a policy. This policy is a rule which action to use in which situation. Therefore it gives them the possibility to fulfill a task in a changing environment, if they were trained for it. For example they can be taught to locate an object and move towards it. They will then find a way to the object independent from their relative position to it, as long as they can locate it.

Using tactile feedback is a possibility to learn tasks which cannot be supervised with cameras and for which classical force torque sensor do not provide enough information. For example the orientation of a small screw hold between two fingers can be measured using the pressure distribution in the tactile sensor, but not with the total force between these fingers. Depending on the sensor type, tactile sensors provide additional information such as the heat flow, vibrations and the pressure distribution. From this information, object properties like friction, material, weight and shape can be computed. The high dimensionality challenges the reinforcement learning methods, because they depend on the state definition, which is very complex on high dimensional data.

1.2 Related Work

Firstly, it is necessary to explain what finger gaiting means in the context of this thesis because different definitions exist. The term 'Finger Gaiting' will be used as a movement that changes a grasp with breaking or adding a finger contact to an object hold in the hand without replacing it to the ground or allowing to change it's orientation or position during the gaiting [3]. While other definitions like in [3] and [4] define it as a periodic process where fingers are replaced multiple times until the goal is reached, in this thesis only the non-periodic change from one finger configuration to another is meant.
The task of finger gaiting as regrasping an object while keeping it in the hand is often needed in in-hand manipulation. In [5] finger gaiting is used in the context of object stabilization and an optimal new finger position is computed to which the finger is slided using tactile feedback. In [6] stratified motion planning methods are proposed to control finger gaiting. Computing a new grasp with a genetic algorithm and reaching it with finger gaiting [7] is another proposal. Finger gaiting in the context of joints reaching their limits during task execution, while a movement needs to be continued to fulfill the task is analyzed in [4], [6] and [7]. In this case, a finger needs to be replaced to continue the manipulation. To replace a finger, it can be necessary to place an additional finger at the object to stabilize it, while the finger that reached a joint limit is removed and newly placed. Another reason for finger gaiting can be that an object was grasped in a position where only a two finger grasp was applicable and a more stable three finger grasp can be used after the object was moved. The work above concentrates on finding new grasps and deciding when to regrasp. Focus in this thesis is to learn the movement from one grasp to another. Furthermore, learning is done without a model of the object, which is usually required for computing finger gaits. Nevertheless, work for finger gaiting without an object model exist. For example [8] uses a model of the kinematics of the manipulation, but not of the object.

Generating good features is crucial for learning. A method of increasing popularity in classification with support vector machines is to use Random Features. This method was developed for kernel machines to reduce the dimensionality of the feature space [1]. In the context of robot control it was successfully applied in robot learning as features for a non-parametric policy [9] and for learning robot dynamics [10]. Other dimensionality-reduction methods for tactile data proofed good results in tactile material classification [11] and shows the necessity of dimensionality reduction. Using Random Features is an alternative for hand-crafted features.

Learning in-hand manipulation tasks with tactile feedback without an object model were applied to underactuated hands [12] as well as for fully actuated hands in combination with other feedback [13]. Tactile feedback contains information of slip [14] which can be used to increase object stability. It allows object stabilization and force optimization [5]. Furthermore it can be used to recognize materials [11]. The contact information generated from tactile feedback can be used to classify grasp qualities [15]. With the tactile image of an item its location between the sensors can be allocated [16].

1.3 Outlook

In this thesis, first some foundations of reinforcement learning will be explained in Chapter 2. Afterwards, the used methods will be explained in detail in Chapter 3. These contain the used learning algorithm and the Random Fourier Feature approach from which the features will be sampled. Additionally, the simulation set-up and the used assumptions will be shown. This includes the simulation of the hardware and the used software. In the Chapter 4 the different experiment set-ups are described and the results are presented. Finally, the approach will be discussed in Chapter 5 and an outlook for future work is given.

2 Foundations

This chapter gives an overview of reinforcement learning. All explanations and equations in this chapter are based on [17]. Basic principles and central terms of reinforcement learning will be explained. These contain Markov decision processes, value functions and reward functions. Furthermore, the difference of episode-based and step-based reinforcement learning algorithms is explained. Three important groups of reinforcement learning will be presented: Dynamic Programming, Temporal Difference Learning and Monte Carlo Methods. Finally, Policy Gradients methods, to which the learning algorithm used in this thesis belongs, are introduced and the concept of policy hierarchies is explained.

2.1 Reinforcement learning

Reinforcement learning is a machine learning method. The agent, e.g. the machine that tries to solve a task, should learn the best way to do it by improving its behavior according to the observation of rewards that followed its actions.

A certain type of reinforcement learning problems are called partially observable Markov decision processes. In these processes the agent does not know all relevant properties of its environment. For example in the finger gaiting task it has no information about the objects position without finger contacts.

In reinforcement learning, the agent interacts with its environment without labeled data, which is used in supervised learning. In every time step, the agent makes a decision, what it wants to do. This decisions is not made randomly but according to a policy. The policy is a rule or a set of rules which action shall be taken in which state. It can be deterministic or stochastic. With deterministic policies for a state s always the same action a is executed. For stochastic policies it is probabilistic which action is taken. Thr policy is periodically updated which will improve the agents behavior. This policy update is the actual learning, because it will improve the decisions. The state notated as s is a representation of the environment. The robot's joint positions are considered to be part of the environment because the agent can only decide which torques it applies to the joint motors but not the current position.
After a action a is executed it will be rated. The return value of this rating is the reward. The rating usually includes the action and the new state. The agent tries to improve its policy to increase its rewards over all time steps of a rollout. In formula, the reward for the complete rollout is usually notated as R and the reward for a single action in timestep t as r_t.

Markov decision process

Most reinforcement learning approaches base on **Markov decision processes (MDP)**. In MDPs, the state representation shall fulfill the Markov property

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} = Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}.$$

That means the state must contain all relevant information of the environment known by the agent and the agent must be able to decide from this state what to do without knowing former states. The probability of choosing action a in state s must be independent from time and former states. MDPs for a finite number of timesteps are called finite MDP. They are defined by:

$$\begin{aligned} P^a_{ss'} &= Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad and \\ R^a_{ss'} &= E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}. \end{aligned}$$

This means that the probability for getting into state s' in timestep $t+1$ depends on the state and the action in t. The expected reward $R^a_{ss'}$ for the next state s' is the expectation for the next reward r_{t+1} when action a is choosen in state s in timestep t and the new state in $t+1$ is s'.

Value functions and their optimality

For reinforcement learning, the use of a state value function, a reward function and a policy is characteristic. In the following, these two functions will be described.
The **state-value function** V^π for a policy π is given by

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\}, \quad with \quad R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

with state s, s_t the state in time step t and R_t the future reward for s. The value function contains information about how good a state is under policy π.
A second value function, the **state-action value function** $Q^\pi(s,a)$ denotes how good it is to take action a in state s for all states and action. This function is given by:

$$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\}.$$

The **reward function** is very important because it evaluates the action and rates it. The rewards are used to improve the policy. Therefore, it is only possible to find a policy which is optimal with regard to the reward function. That means, that a better policy may exist for a task, which will not be found because it is not optimal for the given reward function. This dependence shows why the reward function is crucial for learning. The design of the reward function is task dependent.

2.1.1 Reinforcement learning algorithms

There are three classical groups of reinforcement algorithms: Dynamic Programming, Monte Carlo Methods and Temporal Difference Learning. Some of them are episode-based and some are step-based. Episode-based algorithms sample a complete rollout before the value function is updated.

For these three groups the Bellman equations are fundamental. The Bellmann equations give the optimal value function $V^*(s)$ and the optimal state-action function $Q^*(s,a)$ as

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \quad and \tag{2.1}$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\} \tag{2.2}$$

where γ is a discount factor and max_a denotes taking the action a for which the following expression has the highest value.
The main differences between the methods below is whether they require a complete model of the environment and whether they are episode-based or step-based. In episode-based methods a complete rollout is sampled. That means the agent does one trial to archieve its task. After the episode is sampled an evaluation is done and the policy will be improved, while step-based methods do updates during the rollout.

Dynamic Programming

Dynamic Programming (DP) requires a perfect model of the world and is expensive in computing. In general, it will not find optimal solutions in continuous state and action spaces. It is characterized by policy evaluation and policy improvement, which can be done with methods like value iteration or policy iteration.
For policy evaluation the value function V^Π is computed. This update is based on the Bellmann equation:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')].$$

This means that all possible future rewards and values weighted with their probabilities are used to update the value function. The policy $\pi(s,a)$ gives the probability of choosing action a when the agent is in state s and γ is the weighting for the value function entry of the next state. The update is done for all states and repeated until the function converges. The evaluation is repeated until convergence is reached. Afterwards, policy improvement is done, where π is improved. For this an action-value function is used, which contains information how good an action is for the state. The policy is updated by increasing the probability to take the action with the best expectations for the future reward, e.g.

$$\pi(s) = argmax_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')].$$

With the improved policy, the value function is evaluated again. For an updated policy π' the new state-value function will be $V^{\pi'}$ and then the policy π' will be updated with the results for $V^{\pi'}$, so the the policy and the state-value function are improving each other. Another view on it is, that the state-value function is an evaluation of the policy. The update process called policy iteration is repeated until convergence is reached.
For value iteration the policy evaluation includes policy improvement. The state function is updated with

$$V(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')].$$

This is done for all states and repeated until the function converges. After convergence, the optimal, deterministic policy is

$$\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')].$$

Monte Carlo Methods

Monte Carlo Methods are another classical category of reinforcement learning algorithms. Opposite to DP they do not require a complete model of the environment. They use states, actions and rewards from complete rollouts, what makes it episode-based. Besides this, it uses the same principals as Dynamic Programming.

Temporal Difference Learning

Temporal Difference Learning does not need an accurate model of the world. The agent learns from experiences instead, like it does in Monte Carlo methods as well. Opposite to Monte Carlo methods it is step-based. For updating the value function only the reward of one timestep is used, not a sum of rewards. For TD(0), a simple Temporal Difference method, the state-value function update is

$$\mathbf{V}(s_t) = \mathbf{V}(s_t) + \alpha[r_{t+1} + \gamma \mathbf{V}(s_{t+1}) - V(s_t)].$$

It uses bootstrapping like DP methods. Bootstrapping is updating the function with the value of the next state. The value of that state will be again updated by the value of the following state and this process is continued until the state is a terminal state.

2.1.2 Policy Gradient Methods

The three method categories shown in Section 2.1.1 require a finite number of states and actions. In order to deal with continuous state and action spaces, an approximate of the value and action functions is needed, as it is impossible to save infinite states in a table.
The idea behind Policy Gradient methods is to improve the policy into the direction which promises the best rewards. To define this direction the gradient of the expected rewards is used. To approximate the value function, a differentiable function f with parameters $\boldsymbol{\theta}$ is used. Value function updating is done by updating the parameters $\boldsymbol{\theta}$. For this purpose, the gradient of $\mathbf{V_t}(s_t)$ with respect to $\boldsymbol{\theta}$ is used.
In liner methods the policy vector $\boldsymbol{\theta}$ is multiplied with the feature vector element wise. The state-value function then is given by $\mathbf{V_t}(s) = \sum_{i=1}^{n} \boldsymbol{\theta}(i) * \boldsymbol{\varphi_s}(i)$ where $\boldsymbol{\varphi_s}$ is the feature vector for state s and the gradient for this function is the feature vector.
There is another common notation, $\mathbf{J}_{\boldsymbol{\theta}}$ for the expected rewards. According to [18] the policy update and the reward gradient are

$$\begin{aligned} \boldsymbol{\theta}_{\mathbf{k+1}} &= \boldsymbol{\theta}_{\mathbf{k}} + \alpha \nabla_{\boldsymbol{\theta}} \mathbf{J}_{\boldsymbol{\theta}} \quad and \\ \nabla_{\boldsymbol{\theta}} \mathbf{J}_{\boldsymbol{\theta}} &= \int_{\tau} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}} R(\boldsymbol{\tau}) d\boldsymbol{\tau} \end{aligned}$$

where τ is the frequence of states and actions of rollout, in robotics this is the trajectory. The parameter α is the learning rate.

2.1.3 Upper-Level Policy and Lower-Level Policy

Several reinforcement learning algorithms use the concept of upper-level policies from which a lower-level policy is sampled [18]. In this concept the agent learns an upper-level policy $\pi_{\omega}(\boldsymbol{\theta})$. From this upper-level policy the parameters $\boldsymbol{\theta}$ of the lower-level policy are sampled. These lower-level policy is usually deterministic while the upper-level policy is a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$ in common [18]. The parameters $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ of the upper-level policy will be improved by learning while the lower-level policy will not be updated. The quality of the lower-level policy will increase with the improvement of the upper-level policy.

3 Set-Up And Implementation

In this chapter the used algorithm and the feature generation method are explained. Afterwards, a detailed task description is given. Then, the simulation set-up is explained. The simulation set-up includes the used software and the simulated object, sensors and the robot hand. Following, the implementation of the reward function and action calculation are shown.

3.1 Policy gradients with parameter-based exploration

In this chapter Policy gradients with parameter-based exploration (PGPE) is presented like Sehnke et al. [19] introduced it.
Policy gradients with parameter-based exploration is a model free reinforcement learning method for partially observable Markov Decision problems. Opposite from other reinforcement learning with policy gradients, this method is episode-based. Episode-based means that a complete rollout is sampled with the same parameters. In the context of PGPE these rollouts are usually referred to as histories. Furthermore it does not directly learn a policy but learns hyper parameters a policy is sampled from. This means, it uses an upper-level policy for learning and a lower-level policy for control. This policy hierarchy is described in 2.1.3. The hyper parameters are the parameters of the upper-level policy. The parameters for the controller, the lower-level policy, are sampled from the upper-level policy once per rollout. While the lower-level policy is deterministic the upper-level policy is stochastic. Other policy gradient algorithms like REINFORCE [20] improve the policy which determines the actions directly and therefore do not need an upper-level policy. Using one deterministic parameter set per history reduces the variance in the gradient estimator, compared to the variance when parameters are drawn from a stochastic policy in every time step of a rollout. Additionally, PGPE does not require that the controller policy is differentiable.
The aim of the algorithm is to maximize the reward of the agent by optimizing the parameters $\boldsymbol{\theta}$ by learning the upper-level policy. This optimization is done with the gradient $\nabla J(\boldsymbol{\theta}) = \int_H p(h \mid \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(h \mid \boldsymbol{\theta}) r(h) dh$ where h is a history and $r(h)$ is the reward of this history. The mean $\boldsymbol{\mu}$ and the standard derivation $\boldsymbol{\sigma}$ of the parameters are set to an initial value. Then, in every episode, N policies $\boldsymbol{\theta}^1$, $\boldsymbol{\theta}^2$, ...$\boldsymbol{\theta}^n$ are drawn from the current distribution of parameters $\mathcal{N}(\boldsymbol{\mu}, \mathbf{I}\boldsymbol{\sigma}^2)$. With every parameter set a history is sampled and the reward saved.

At the end of an episode, mean and standard derivation of the upper level policy are updated according to the update policy shown in Equation (3.1) with learning rate α.

Update of upper level policy (3.1)

$$
\begin{aligned}
\mathbf{T} &= [t_{ij}]_{ij} \quad with \quad t_{ij} := (\theta_i^j - \mu_i) \\
\mathbf{S} &= [s_{ij}]_{ij} \quad with \quad s_{ij} := \frac{t_{ij}^2 - \sigma_i^2}{\sigma_i} \\
\mathbf{r} &= [(r^1 - b), ..., (r^N - b)]^T \\
\\
\boldsymbol{\mu} &= \boldsymbol{\mu} + \alpha \mathbf{T}\mathbf{r} \\
\boldsymbol{\sigma} &= \boldsymbol{\sigma} + \alpha \mathbf{S}\mathbf{r}
\end{aligned}
$$

For faster convergence a baseline b for the rewards can be used. The original paper [19] uses a moving average. For further variance reduction an optimal baseline[21] can be used. The baseline is updated after mean and standard derivation are updated.

3.1.1 PGPE with symmetric sampling

The authors of [19] also present an extension of PGPE where symmetric sampling is used. This version vanishes the drawback of misleading baselines and improves gradients for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Instead of sampling $\boldsymbol{\theta}$ from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{I}\boldsymbol{\sigma}^2)$, now an $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}\boldsymbol{\sigma}^2)$ is sampled, then added and subtracted from the mean. This leads to the two policies $\boldsymbol{\theta}^{+,n} = \boldsymbol{\theta} + \boldsymbol{\epsilon}$ and $\boldsymbol{\theta}^{-,n} = \boldsymbol{\theta} - \boldsymbol{\epsilon}$. For both policies a history is sampled and their rewards will further be referred to as r^+ and r^-,

respectively. The difference of $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ is exactly $\boldsymbol{\epsilon}$, therefore $t_{ij} = \epsilon_i^j$ where i denotes the entry of the parameter vector and j is the history. To make the policy update indifferent towards the reward's range, the change of the mean is normalized with $\frac{1}{m-r^+-r^-}$ and the change of the standard derivation with $\frac{1}{m-b}$ where m is the maximal reward.

Pseudocode of Symmetric PGPE

```
// initialization
```
$\boldsymbol{\mu} = \boldsymbol{\mu}_{\text{init}}$
$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{\text{init}}$

while *TRUE* **do**

  for n *= 1 to* N **do**

    draw $\boldsymbol{\epsilon}^{\mathbf{n}} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{I}\boldsymbol{\sigma}^{\mathbf{2}})$

    $\boldsymbol{\theta}+, \mathbf{n} = \boldsymbol{\mu} + \boldsymbol{\epsilon}^{\mathbf{n}}$

    $\boldsymbol{\theta}-, \mathbf{n} = \boldsymbol{\mu}\boldsymbol{\epsilon}^{\mathbf{n}}$

    sample historeies

    $r^{+,n} = r(h(\boldsymbol{\theta}^{+,\mathbf{n}}))$

    $r^{-,n} = r(h(\boldsymbol{\theta}^{-,\mathbf{n}}))$

  $\mathbf{T} = [t_{ij}]_{ij}\ with\ t_{ij} := \epsilon_i^j$

  $\mathbf{S} = [S_{ij}]_{ij}\ with\ s_{ij} := \frac{(\epsilon_i^j)^2 - \sigma_i^2}{\sigma_i}$

  $\mathbf{r_T} = [\frac{(r^{+,1}-r^{-,1})}{m-r^{+,1}-r^{-,1}}, ..., \frac{(r^{+,n}-r^{-,n})}{m-r^{+,n}-r^{-,n}}]^T$

  $\mathbf{r_S} = \frac{1}{m-b} * [(\frac{r^{+,1}+r^{-,1}}{2} - b), ..., (\frac{r^{+,n}+r^{-,n}}{2} - b)]^T$

  ```
  // update policy
  ```
  $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha_\mu \mathbf{S}\mathbf{r}_\mathbf{T}$

  $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha_\sigma \mathbf{S}\mathbf{r}_\mathbf{S}$

  update baseline b

Algorithm 1: Symmetric PGPE with reward normalization

3.2 Random Fourier Features

There are different approaches how to generate features from continuous, high-dimensional state spaces. One of them is to use random features which do a projection from a high dimensional state space to a one-dimensional space and then combining D of this projections to a feature vector. Instead of factorizing the kernel matrix the kernel function itself is factorized. This method presented in [1] is explained in this section. The kernel evaluation is approximated as

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x) * z(y)$$

The authors propose two different kinds of features, Random Fourier Features and Random Binning Features. In the following we will have a closer look at the Random Fourier Features. In general, Random Fourier Features are referred to as Random Features in this thesis.

The method works for shift invariant kernels like the Gaussian kernel. To compute the features, a feature map $z(x) : \mathscr{R}^d \to \mathscr{R}^D$, where d is the dimensionality of the state $\mathbf{x}$ and D is the number of concatenated projections. The concatenation is done to reduce the variance in the features. A suggestion for choosing D is

$$D = \mathcal{O}(d\epsilon^{-2} \log \frac{1}{\epsilon^2}) \tag{3.2}$$

where ϵ is the accuracy of the kernel approximation. One possible definition for z is

$$\mathbf{z}_\omega(\mathbf{x}) = \sqrt{\frac{2}{D}} [\cos(\omega_1' \mathbf{x} + b_1),\ ...,\ \cos(\omega_D' \mathbf{x} + b_D)]'.$$

The normalization with D is not necessary but reduces the variance. The vectors $\omega_i \in \mathbb{R}^d$ are drawn independently and identically distributed from the Fourier transform of the kernel and $b \in \mathbb{R}^D$ is sampled independently and identically distributed from a uniform distribution over $[0, 2\pi]$. For a Gaussian kernel the Fourier transform is

$$p(\omega) = (2\pi)^{-D/2} e^{-\|\omega\|_2^2/2}.$$

For support vector machines this features are useful because they reduce dimensionality. Using them in robot control may avoid the use of hand-crafted features. Kernel methods are often used to learn models of dynamics in robotics [22], but in a different manner.

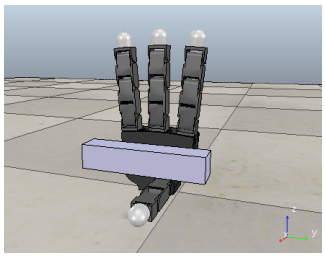

Figure 3.1: Simulation in home pose

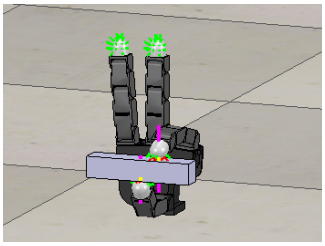

Figure 3.2: Index finger grip

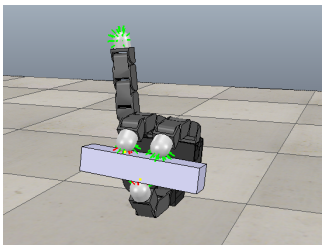

Figure 3.3: Three finger grip

All rollouts begin in home position. At the beginning of a rollout, the robot moves to the index finger grip.
The three finger grip is the desired position. The figure shows the demonstration that is used for the reward calculation.
In Figure 3.2 and 3.3 the contact normal (purple) and the sensor simulation (green) are active

3.3 Task Description

The task is to learn finger gaiting. That means to learn policies that change between different fingertip grips with the symmetric PGPE (Algorithm 1). There are two given grips: a thumb-index finger grip shown in Figure 3.2 and a grip with three fingers shown in Figure 3.3.

The joint positions of these grips are predefined and used as start positions. The reward function uses the difference of the joint positions and sensor data from the fingertip contacts to the demonstration of these. How the state is defined will be described later, actions are defined by a linear, deterministic controller which uses random features as described in Chapter 2 and the parameters for the controller are calculated with PGPE. The primary task is to learn to change between grips and the object stabilization needed for this is secondary. Therefore a limited time horizon of 15 timesteps is used. This time is long enough to change the grip and to see whether the desired grasp is hold stable without further movements and keeps the rollouts short, what decreases the computation time for the learning.

3.4 Simulation Set-Up

The simulation is done with V-Rep which is a robot simulation environment by Coppelia Robotics [23]. The robot control is done as an external API via ROS. ROS is a pseudo operating system developed for robotic applications which coordinates communication between different programs and machines running in ROS.

3.4.1 Communication

Communication between ROS nodes and V-Rep simulation
To control the simulation in V-Rep from a ROS node, a communication interface is needed. The V-Rep exteranl API is a general method to control a simulation in V-Rep from an extern application. It provides all commands which are available in V-Rep and is therefore a very powerful way to control a simulation [24]. Other possibilities to use V-Rep in ROS are the vrep_ros_bridge, the RosInterface and the RosPlugin [25]. The vrep_ros_bridge [26] is a plugin for V-Rep. It uses ROS messages and services to control the simulation. While the other three interfaces belong to V-Rep this one belongs to ROS. It is a comparable new interface and uses different object handlers for robot control. It is based on the v_repExtPluginSkeleton, a V-Rep plugin template. The RosInterface is a fast and powerful interface which makes it possible to run a ROS node within V-Rep and control the robot with this node. The disadvantage of it is, that catkin_tools is required to customize the interface which may be necessary for additional messages and services [27]. In cases V-Rep should be integrated to bigger projects in ROS which were not build with catkin_tools this is inconvenient. The RosPlugin is a higher abstraction of the ROS methods. It can only deal with few standard ROS messages and has a low flexibility [28].

Communication flow
The control of the robot and the learning are done in three ROS nodes. Each ROS node is an independent running program, nevertheless they all need to cooperate to do their purpose.
The **wrapper** node is responsible for controlling the simulation. It transfers information from V-Rep into ROS topics and publishes them in order to make this information available for other nodes. Furthermore, it controls the simulation using information provided by the controller node. It starts and stops the simulation, can change the dynamic properties of the bar which is the manipulation object and sets the desired positions which were calculated by the controller. Therefore,

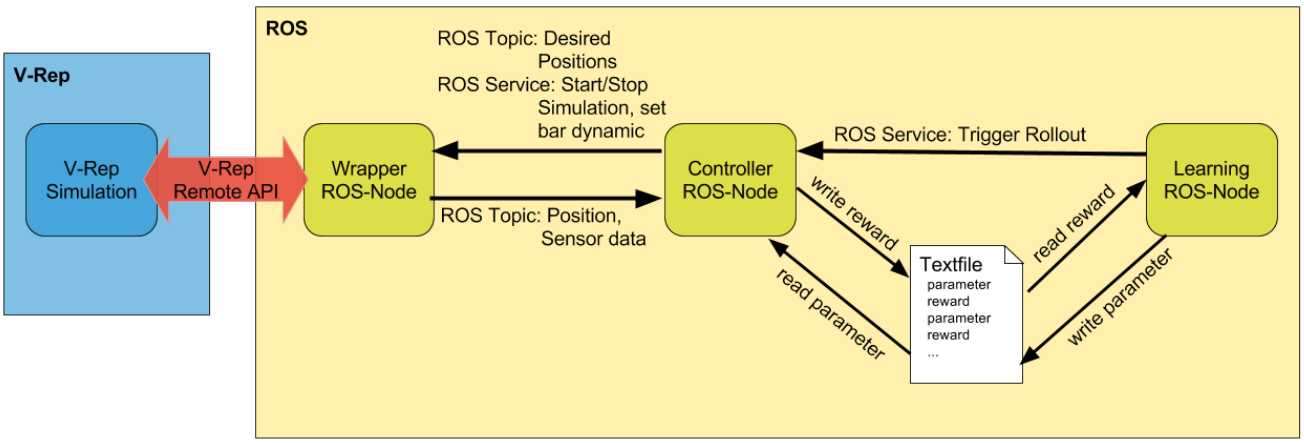

Figure 3.4: Communication set-up.

it subscribes a topic in which the desired positions are sent and sets them via the remote API as desired positions in the simulation. At the end of a simulation time step it reads out the current positions and sensor data, reprocesses them using a filter and publishes them as topic. Furthermore it offers a service to start and stop the simulation. When this service is called it starts, respectively stops the simulation using the remote API.
The **controller node** offers the service of rollout triggering. When the service is called the controller reads in the policy parameters from a text file, calculates the robot's state using the position and sensor information the wrapper node provides and publishes the new desired positions according to the policy. These published positions will be set in the simulation by the wrapper. Furthermore it calculates the reward of the chosen actions. The reward for the complete rollout is written to the file at the end of the service call.
In the **learning** node PGPE is implemented. In this node the upper level policy from which the control policy is sampled, is learned. The sampled policy is written to the file and the rollout service from the controller node is called. When the rollout is done, the reward is read in from the file for further use in the learning algorithm.
The information flow is illustrated in Figure 3.4.

The difference between ROS topics and services

In ROS multiple nodes are processed in a timestep. The communication between them is organized in topics. Each node can subscribe and publish messages to this topics. That means, they can send a message to this topic to all other nodes that subscribed it and for subscribing nodes it means they receive the messages to these topics and process them in a callback. The order of processing callbacks and executing additional code is predefined. During a timestep, all nodes are processed and during the processing callbacks for messages the processed node subscribed are evaluated. The callbacks are evaluated before their published messages are delivered in the next timestep. Therefore it is not possible to send a message and receive an answer in the same timestep.
While ROS topics are published and processed when the subscribing node has its turn in a defined order of callback handling and processing of nodes a services is called immediately and the calling node waits for the result before the next line of code is processed. Therefore, it is not necessary to wait for the next timestep until a result is received as it would be the case with messages. This ensures, that a complete rollout is sampled before the result is read in.
When a node sends a topic as an answer, this causes a big time delay because the sending node will publish the topic, then all nodes will have their turn and process arrived messages. When the receiving node has its turn it will compute an answer and send it. Then it will do a spin so all other nodes including the first sender have their turn and the sender can process the answer. With services only the calling and the answering node are concerned. Nevertheless, like in this approach it is still possible that the service providing node gives other nodes their turns.

3.4.2 General Simulation Settings

The model's dynamics are computed with Bullet 2.38 Engine. The simulation will run with 10 ms timesteps. The new positions will be send at every tenth timestep to give the controller enough time to reach the desired postion and to generate a higher change of the state.

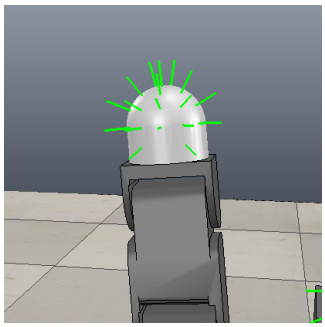

Figure 3.6: Simulated sensor points

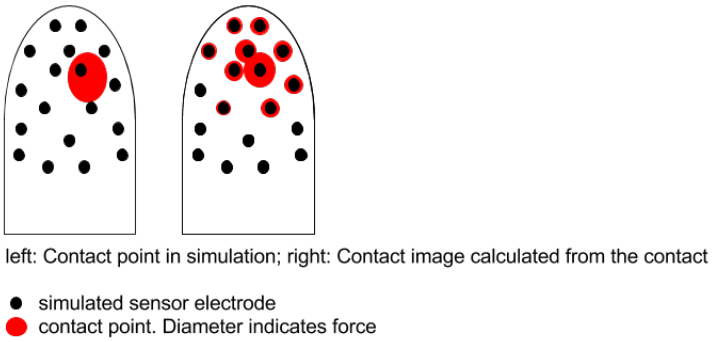

Figure 3.7: Map contact point to sensor image

3.4.3 Simulation of the robot hand, sensors and grasping object

The hand model

The robot hand used is an Allegro hand with four fingers. The model of the hand in the simulation has the same size and joint properties as the real robot.

As simplification, only eight of the sixteen joints of the allegro hand are used. The little finger is not needed for the task and therefore not used. The lowest Degree of freedom (DOF) of index and middle finger and the two lowest DOFs of the thumb are fixed as well. They are orthogonal to the upper joints which open and close the fingers to a palm. Therefore the workspaces of the fingers is limited to a two dimensional movement and fingers cannot cross. This reduces the dimensionality and vanishes the necessity of collision detection and avoidance. Figure 3.5 illustrates which joints are used.

While real human fingers are deformable and the BioTac sensors which could be used for the real robot experiment are soft as well in simulation the fingertips where modeled as half spheres. The reason for this is that it is far easier to compute the dynamics with convex, simple shapes and that V-Rep does not provide deformable materials. The shape of the fingertip makes finding stable grips more complicated.

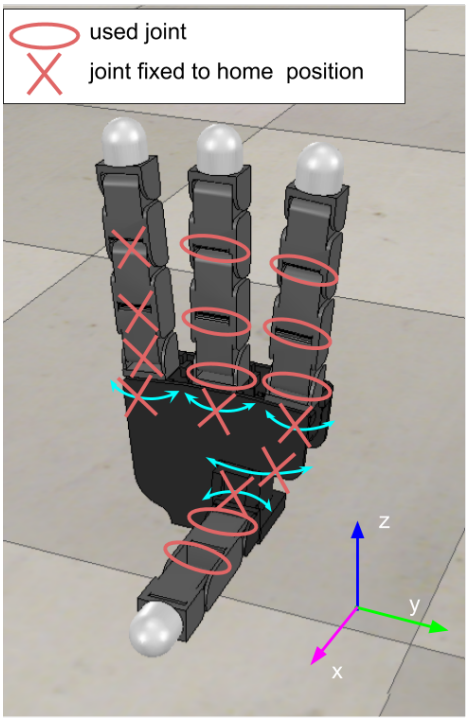

Figure 3.5: Illustration of used joints. The little finger is not needed in the task. The joints which move the finger around the joint's z-axis are fixed as well.

Sensor Approximation

While V-Rep provides proximity and torque force sensors, tactile sensors are not provided. To simulate a tactile image the contact point and force available in V-Rep is used to calculate it. Therefore, the contact force is distributed over a net of sensor points shown in Figure 3.6 to generate the tactile image which is comparable to the image a tactile sensor would generate. This mapping is illustrated in Figure 3.7. The force from the contact point is distributed over the surrounding sensor points with decreasing force values for an increasing radius.

The Grasping Object

The bar is placed in front of the hand and dynamically disabled at the beginning of a rollout which means that it is not movable and does not fall down. This makes it possible to move the robot to the initial grasp position. After grasping the object the dynamics of the bar is enabled and it can be moved and is accelerated by gravity. From that point on, the desired joint position will be computed using the learned parameters. Neither in the reward function nor in the state representation is a model of the bar used.

3.5 Implementation

In this section the used versions of PGPE, the different reward functions which will be compared and the state representations will be explained.

3.5.1 Metaparameters in PGPE

All experiments were done with the symmetric version of PGPE using the reward normalization as shown in Algorithm 1. The maximal reward m uses the so far best reward. As learning rate α_σ 0.2 is used and for α_μ 0.1 is used, as these values were also used for all experiments in [19] and returned good results in this set-up as well. The mean $\boldsymbol{\mu}$ is initialized to one and $\boldsymbol{\sigma}$ is initialized to two, as proposed in [19].

3.5.2 Reward Function

The reward function is one of the most critical parts. If the reward has to high changes or if little changes in policy cause a big change in the reward, this is problematic for the learning algorithm. This is because the reward will have a huge impact on the hyper parameters update and can cause oscillation or divergence in policy which will slow down learning or make it impossible. On the other hand to low rewards will cause no change to the upper level policy and therefore no learning progress can be seen. In addition the scaling of the reward influences the policy update because the average reward minus the baseline is the multiplier for the standard derivation update and the reward difference is used for the update of the mean. Whether all rewards are scaled between one and zero or one and hundred should not influence the learning. The problem is addressed in the normalization extension of PGPE, nevertheless it is useful to have this problem in mind while designing the reward function.
When all fingers loose contacts to the bar, the simulation will be aborted and a very high negative reward for the time step is given. This prevents the algorithm to stop exploration. Otherwise dropping the bar at once to avoid future punishments would be the optimal behavior. Furthermore dropping the bar would violate the Markov property because no matter witch action the robot takes it cannot fulfill its task. This is the case because in this thesis the sensor data is not part of all used state representations and therefore the robot does not know in all scenarios whether the bar is between his fingers or not.

Classical Reward function

A common way to define a reward function is

$$r(s,a) = -(\mathbf{q}-\mathbf{q_d})^T\mathbf{W_q}(\mathbf{q}-\mathbf{q_d}) - \mathbf{a}^T\mathbf{W_a}\mathbf{a}$$

where $\mathbf{q}$ are the joint positions, $\mathbf{q_d}$ are the desired joint positions, $\mathbf{a}$ are the actions taken and $\mathbf{W}$ are the corresponding weight matrices.
Adding the punishment p for dropping this becomes

$$r(s,a) = -(\mathbf{q}-\mathbf{q_d})^T\mathbf{W_q}(\mathbf{q}-\mathbf{q_d}) - \mathbf{a}^T\mathbf{W_a}\mathbf{a} - \delta_{dropped} \tag{3.3}$$
$$with \quad \delta_{dropped} = \begin{cases} p, & \text{if no contact to object} \\ 0, & \text{else} \end{cases}$$

The reward of an episode is the sum of the discounted rewards from all T timesteps of the episode:

$$R = 0.1\sum_{t=0}^{T} r_t * \gamma^t \tag{3.4}$$

For all experiments a discount factor γ of 0.95 will be used. The constant 0.1 was added to reduce the reward range for direct parameter learning and kept for the version with projected parameters. The idea behind it was to reduce the reward variance within an episode.

Variations used

In this section the compared reward functions are listed.
The basic reward function with joint positions $\mathbf{q}$, sensor image $\mathbf{c}$, contact points $\mathbf{CP}$ and punishment p for dropping the bar is:

$$r(s,a) = -\frac{\alpha_1}{\mathbf{c_d}^T\mathbf{c_d}} * (\mathbf{c_d}-\mathbf{c})^T(\mathbf{c_d}-\mathbf{c}) - \frac{\alpha_2}{\mathbf{q_d}^T\mathbf{q_d}}(\mathbf{q_d}-\mathbf{q})^T(\mathbf{q_d}-\mathbf{q}) - \alpha_3(\mathbf{a}^T\mathbf{a}) - \alpha_4 \sum_{fingers} || CP_{finger} - CP_{finger,d} ||_2^2 - \delta_{dropped} \tag{3.5}$$

$$\text{with } \delta_{dropped} = \begin{cases} p, & \text{if no contact to object} \\ 0, & \text{else} \end{cases} \tag{3.6}$$

The weights used in the two used reward functions are shown in Table 3.1. In the first row are weights designed to cause very small reward changes and is used for the experiments with direct parameter learning. The increased stability of the upper level policy in learning with projected parameters made it possible to improved weights for the reward components. These are shown in the second row of the table.

Function used for:	α_1	α_2	α_3	α_4	p
Direct Parameter Learning	1	30	$10/(\mathbf{sensorrange}^T \mathbf{sensorrange})$	1	1000
Learning With Projected Parameters	0.1	6	$1/(average\,sensor\,range)$	6	400

Table 3.1: Weights used in the reward function

3.5.3 State Representation and Feature generation

For the state representation, data from the sensors and the joint positions are used. How the sensors are computed is explained in Section 3.4.3. The contact points are computed as center of force using the relative coordinates of the sensor array and the force at these points. For the joint positions a simplification is used. While the Allegro hand has 16 degrees of freedom (DOFs), only eight are controlled with the learned policy. Which joints are used is described in Section 3.4.3. The lower two degrees of freedom of the thumb, the little finger and the lowest DOFs of the index and middle finger fixed in the way that their desired position is always set to the same value.
Joint angles and sensor data are real valued. For this reason infinitely many states exist. For eight used DOFs this is already an eight dimensional vector. For using sensor data compressed to the contact point which give the relative coordinates of the point of the fingertip that has contact to the bar, 9 further dimensions are added. The whole sensory image is 54 dimensional.
To face this problem of dimensionality Random Features from [1] will be used.
The amount of used features depends on the experiment. Feature sets of 100, 300, 1500, 5400 and 8000 features will be used. Using the suggestion to choose the number of features D as $D = \mathcal{O}(d\epsilon^{-2}\log\frac{1}{\epsilon^2})$ from [1] the necessary number of features D for an kernel approximation with accuracy ϵ is shown in Table 3.2:

State base	Dimensions	D for $\epsilon = 0.1$	D for $\epsilon = 0.2$	D for $\epsilon = 0.3$
Joint Positions	8	1600	280	90
Joint Positions & Contact Points	17	3400	560	200
Sensory Image	54	10800	1890	630
Sensory Image & Joint Positions	63	12600	2200	730

Table 3.2: Recommended number of features for different state representations based on Equation (3.2) from [1]

3.5.4 Action calculation

For each joint a parameter vector $\boldsymbol{\theta}_\mathbf{i}$ is provided by PGPE. The Random Fourier Features $z_\omega(\mathbf{x})$ that represent the state s will be used for all joints. It is the feature map from the Random Fourier Features which were developed by [1] and were explained in Section 3.2. The input data $\mathbf{x}$ are the joint positions, the contact points, the sensor image or a combination of these respectively. For each joint an action a_i will be computed as scalar product of the parameters and features:

$$a_i = \boldsymbol{\theta}_\mathbf{i}^T z_\omega(\mathbf{x}). \tag{3.7}$$

This version of directly learning the parameters for the D features will be compared to the version using projections of state and parameters. In this second version, the actions are computed as:

$$a_i = z_\omega(\boldsymbol{\theta}_\mathbf{i})^T z_\omega(\mathbf{x}). \tag{3.8}$$

The parameter vector $\boldsymbol{\theta}_\mathbf{i}$ has the same dimensionality as the state representation. This is necessary, because the same mapping function $z_\omega(\mathbf{x})$ is used for the parameters and the state mapping. Using the same mapping causes that the

number of required parameters is equal to the number of elements in the state representation $\mathbf{x}$.
While Equation (3.7) requires as many parameters as the dimensionality of the mapping D is, when the parameters are projected as well, the dimensionality becomes independent of D. As shown above, for better results a higher D is necessary. Therefore several thousand parameters must be learned for an action calculation as shown in (3.7).
In general, the action calculation is the evaluation of the approximation of the used Gaussian kernel.

4 Experiments

In the first section of this chapter experiments with direct parameter learning are described. That means, for action calculation Equation (3.7) is used. The problems of this approach will be pointed out and illustrated in an example. In the second section a projection of the parameters is done. The actions are calculated as kernel approximation of state base and parameters. It will be shown that the policy converges earlier and to a higher reward with features generated from a low dimensional state representation than with features based on a more complex state representation.

4.1 Directly learning parameters

For the following experiments parameters for all features were learned directly. Therefore, the features were computed as Random Fourier Features and the corresponding parameters were learned with PGPE. The actions were calculated as scalar product of features and parameters as shown in Equation (3.7). For the reward function the coefficients shown in row one of Table 3.1 are used. The features are 100-dimensional because this was the maximal possible number of features for which the algorithm was solid enough.

4.1.1 Evaluation of learning convergence

To implement the version with direct parameter learning several reward functions were used. Different reward functions were tested as well as different initial values for the upper level policy. Most implementations led to values outside the range of a double in mean and standard derivation of the upper level policy and a reward of minus infinity. Lower initial values for mean and standard derivation of the upper level policy did not improve the result. The use of reward function (3.6) led to a result that was solid long enough to see a learning effect. Figure 4.1 illustrates this in an example. The shown example was sampled with 50 histories per episode. The sampling contained 150 episodes which were cut in the graphic, because the decrease of reward is continued until the range of the used data type is reached and therefore the scaling covered the learning progress in the first 100 episodes. The trial was done with 100 parameters per joint and the state representation is based on joint positions and contact points. Using only 100 features is actually insufficient for a state calculated with Random Fourier Features based on contact points and joints positions, but pretests showed, that it is the highest possible number of features to realize any reward improvement before the race to the bottom begins.

In the first 50 episodes the average reward of the episode is nearly constant, while the variance of the reward within an episode increases slowly. With the increasing improvement of the reward the variance in the rewards of an episodes increases as well. When the reward reaches its maximum and has small changes again, also the variance of the episode's rewards decreases. Nevertheless, the changes in the rewards cause increasing updates of the upper level policy, which are not compensated by the normalization. The increase of the absolute values of the mean and standard derivation in the upper level policy leads to higher parameters of the controller policy, that lead to the choice of bigger actions, which are beyond the joint ranges by factor 100 after a few episodes. This decreases the reward further, because taken actions are punished and the desired position is not reached as well. The changes in the upper level policy and the overcorrection followed by rapidly decreasing rewards lead to an parameter oscillation, which enhance the problem. Pointing out the too high actions as a problem it would be reasonable to constrain the actions and limit the action punishment. Several runs with this constraints showed, that the minimal or maximal joint position was chosen for all actions after less than ten episodes and no recovery occurred within the next 100 episodes. This was the case as well, when only a maximal action reward or the constraint of the action to the limits of the joint ranges was used.

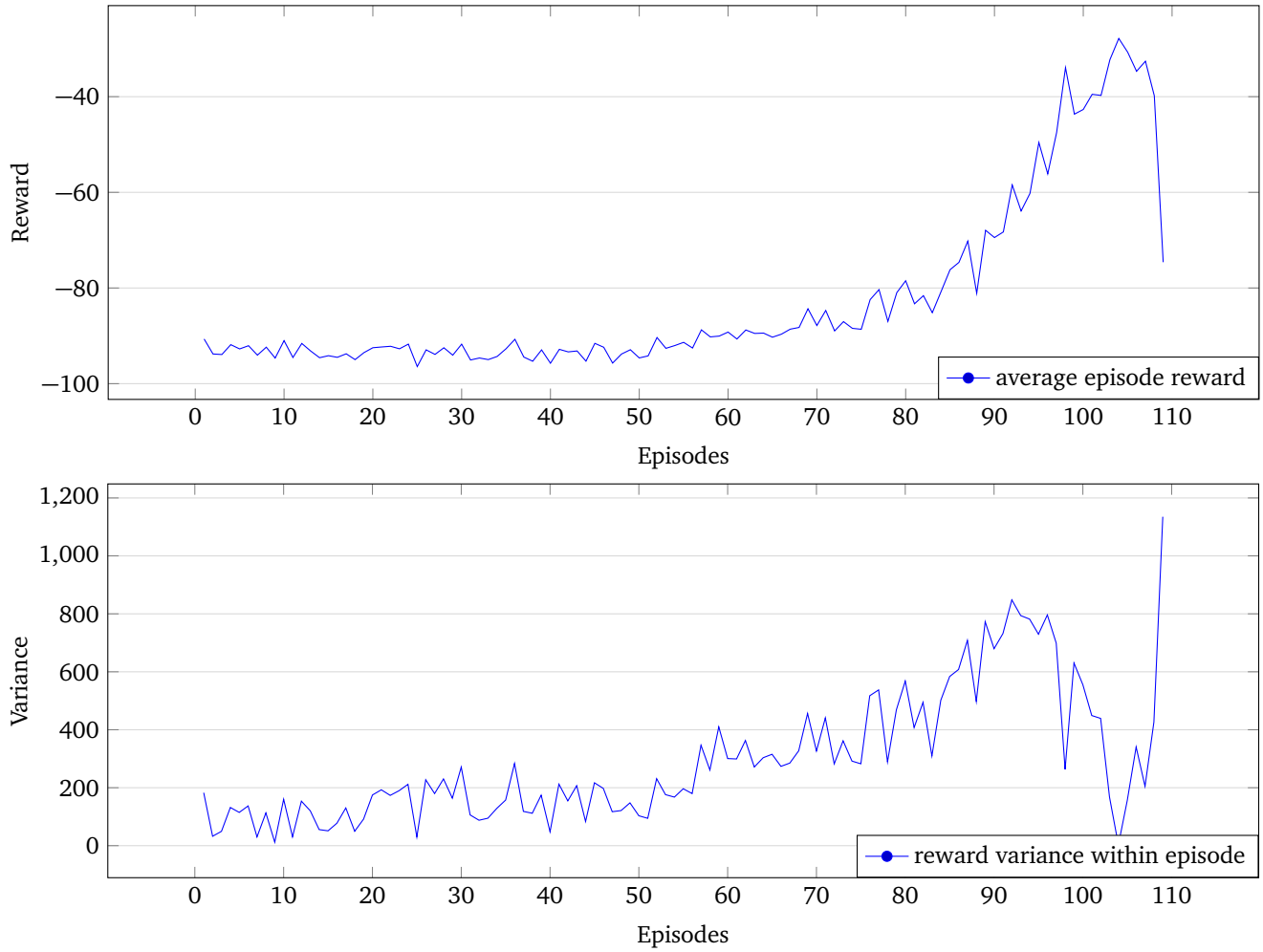

Figure 4.1: Example for late learning success and reward race to the bottom in direct parameter learning.

4.1.2 Impact of number of histories per episode

An approach of comparing the impact of the number of histories per episode was done as well, but were only compared for a total number of 2000 histories. As shown above, the learning progress in the first episodes is very small. Therefore, it is not surprising, that the results indicate, that the reward compared to the total number of histories is constant. A closer investigation of this remains as future work.

4.1.3 Summary of results

This method is prone to a reward race to the bottom. Furthermore, it has tendencies to parameter oscillation and choosing too high actions. The policy improvement begins comparable late and is aborted, before the task is learned. The reward function must be tuned to cause small reward changes. Another drawback is the high number of parameters that must be learned. For 100 features 800 parameters must be learned in the set-up with eight used joints. To estimate the gradient for these is difficult. Furthermore, the accuracy of the kernel approximation is low when only a small number of features can be used.

4.2 Learning with parameter projection

As shown above, directly learning the parameters does not lead to the desired result. Especially the problem of choosing actions out of range, the race to the bottom of the rewards and the necessity to learn hundreds or thousands of parameters makes it rather impractical. The high number of parameters is caused by the necessity to learn D parameters per joint, where D needs to be high for a good kernel approximation as shown in Table 3.2.

Therefore, the results from above will be compared with results for projected parameters. The most obvious advantage of this approach is that the number of parameters to learn is only the number of joints to control times the dimensionality of the state representation. The actions are calculated as shown in Equation (3.8). The increased stability of the upper level policy allowed reward function tuning. Therefore, all experiments in this section were done with the improved reward function shown in Equation (3.6) with parameters from row two of Table 3.1.

4.2.1 Evaluation of learning convergence

Reward convergence

The convergence of the reward depends on the used state representation base. For smaller state representation, e.g. only joint positions, the convergence is reached earlier. Therefore plots of trials with states based on the joint position will have view episodes. The influence of the state base is shown in Figure 4.4 where the convergence of the rewards can be seen as well. Moreover, the convergence depends on the number of used features and the number of histories per episode. The influence of the number of features and histories per episode will be examined below.

Convergence of upper level policy

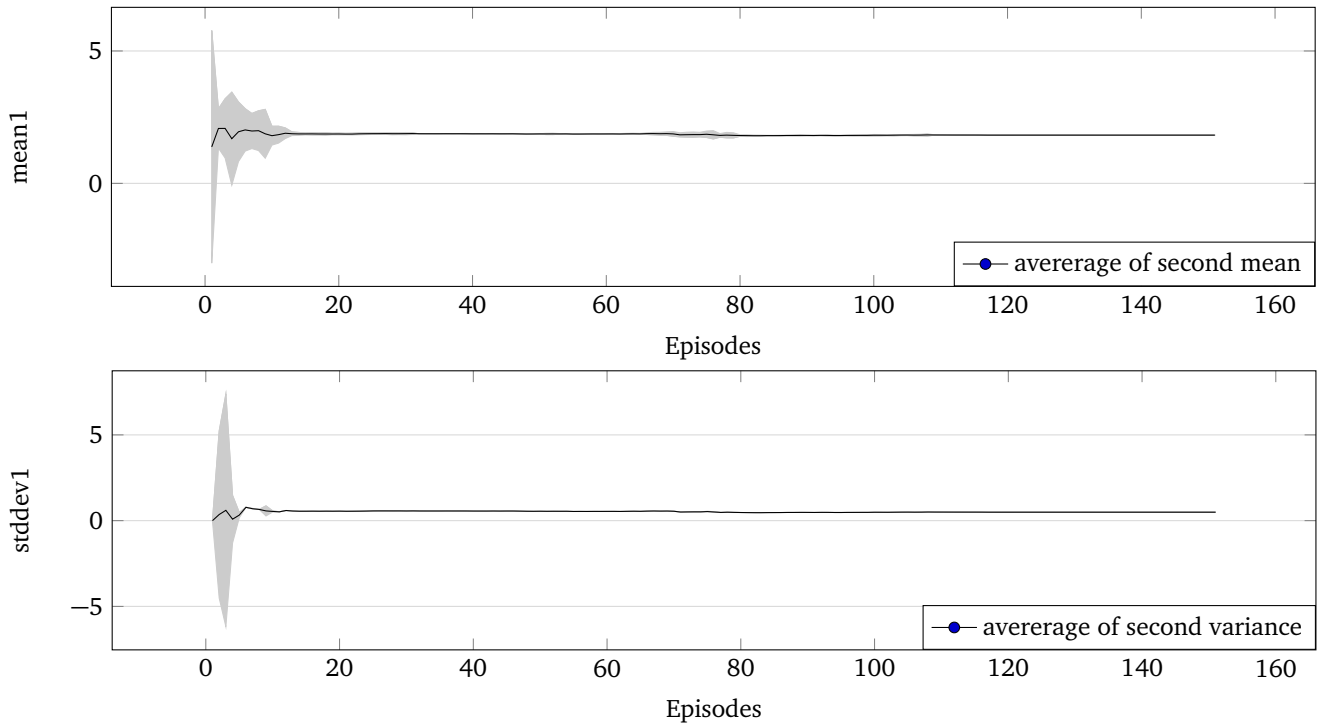

Figure 4.2: Top: average value for second upper level parameter mean. Gray shadow is the variance of mean in three independent trials
Bottom: average variance value of second upper level parameter. The variance of the upper level variance parameter two is shaded in gray.

The convergence of the upper level policy is illustrated in Figure 4.2. Three independent learning trials with a state based on contact points and joint positions were sampled with 1500 features. To reduce the dimensionality of the parameter vector of the upper level policy, only the second mean and standard derivation entries are plotted. The black line are the average values for the parameters in the three trials. The gray shadow illustrates the variance of the parameters over the three trials. Although the mean is different for different trials and the variance of the mean over the three trials is therefore high, the standard derivation converges for all trials to zero and therefore, the mean converges. In the figure, the convergence is reached early. Other parameters change after episode 40, but converge as well.

4.2.2 Impact of number of features

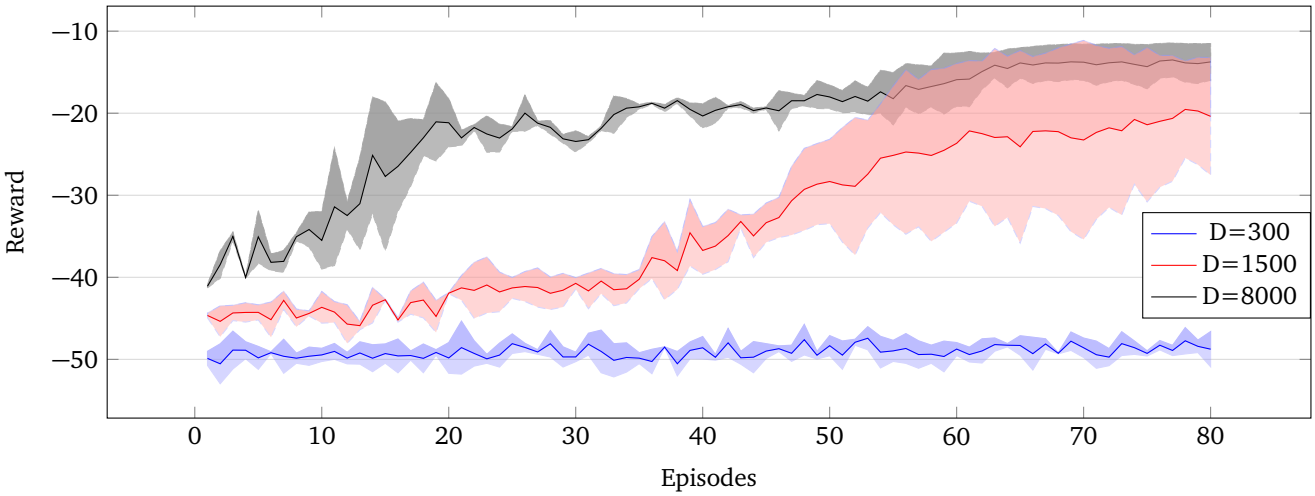

Figure 4.3: Illustratian of the impact of the number of features. For D=300, D=1500 and D=8000 three independent trials were sampled. As state based the joint positions were used. The standard derivation of the samples is shown as shadow.

For the analysis of the influence of D on the reward, three independent trials with $D = 300$, $D = 1500$ and $D = 8000$ were done. For all trials states based on joint positions were used. For higher D, the reward increases faster and converges to a better value. Additionaly, for $D = 8000$ the standard derivation is lower than for $D = 1500$. The rewards for $D = 300$ show no learning effect in the investigated time of 150 episodes. The reason for this may be, that the used kernel is only approximated to an accuracy of 80%, while $D = 8000$ is an estimated accuracy of 95%. The approximation accuracies for different state bases are shown in Table 3.2. This results given an good explanation why the learning success with other state representations is smaller. Like the trials for $D = 300$ and $D = 1500$, other state representations were compared with accuracies between 0.1 and 0.2. As shown in Figure 4.3, for theses feature sets lower rewards and slower learning can be expected. That the result improves with higher D was also shown in the results of [10] and [1]. Their result, that learning is possible with smaller feature sets can be confirmed with this results too, but only for

4.2.3 Comparison of different Feature sets

Although Random Features are used, the base of the features is hand-crafted. To compare which information has best learning results, independent trials with states based on joint positions, joints positions and contact points and sensor image instead of contact points were sampled. For better comparability, 1500 features were used for joint positions and joint positions with contact points. Using the complete sensor image with 54 entries requires a higher number of features to generate an equivalent approximation accuracy of the kernel. Therefore, for the state based on joint position and sensor image $D = 5400$ was used.

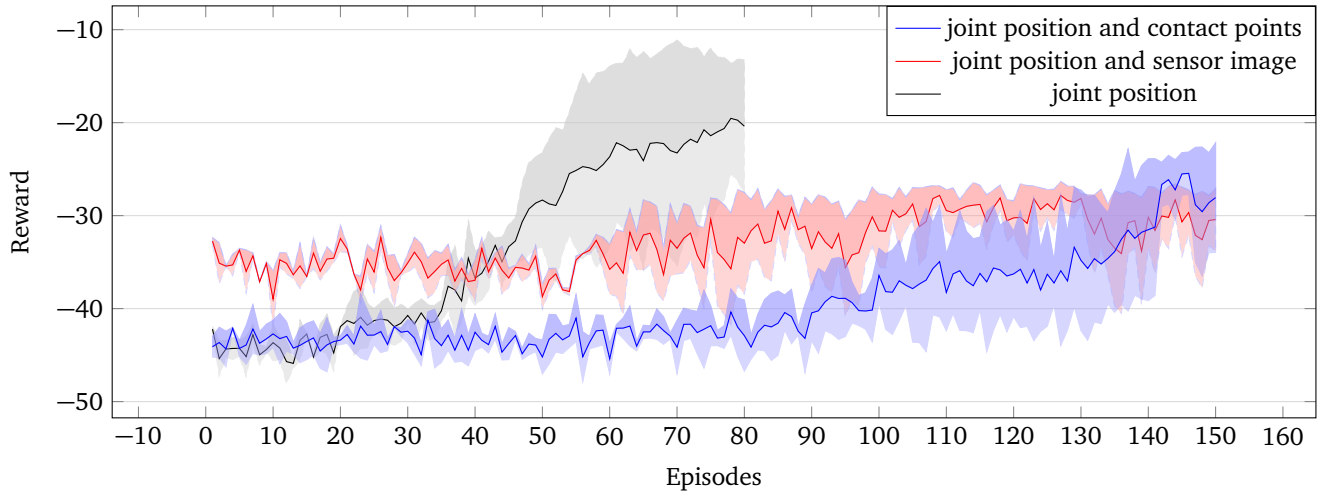

Figure 4.4: Comparison of different state bases. Three independent trials for each state base were sampled and the average is plotted as line. The standard derivation of the trials is plotted as shadowed.

The results show, that the state based on joint positions, which has the lowest dimensionality, has the fastest learning progress and receives the highest reward. The time to convergence is the shortest as well. Therefore, only 80 episodes are sampled for this state base. The most complex data, the joint positions with sensor data, have the smallest learning progress within 150 episodes.

4.2.4 The influence of the number of histories per episode

To compare the influence of the number of histories per episode N is compared for three independent trials of $N = 15$ and three independent trials for $N = 50$. For all rollouts a state representation based on joint positions and contact points was chosen and 1500 features were generated.

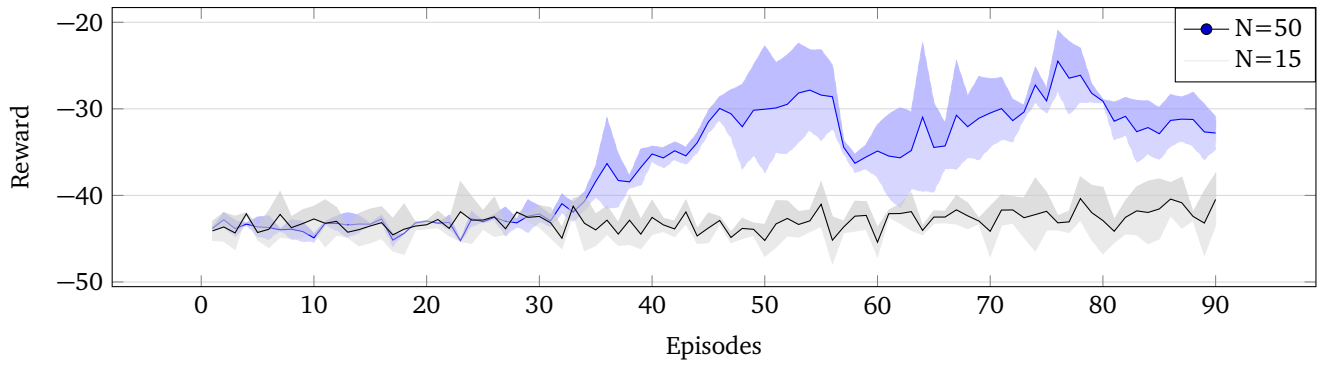

Figure 4.5: Comparison of the average reward of three independent trials with 50 histories per episode and 15 histories per episode. The shaded Area is the standard derivation of the trials.

In the trials with the higher number of histories, the reward is only gently higher than in the trials with few histories. This is surprising because a higher number of histories includes more parameter variations and therefore the gradient estimate should be more accurate. In practice, the first significant reward increase is for the higher N after 40 episodes, but after a short interval the reward drops. For the lower number of histories, the reward increases after 90 episodes to a similar value, which is shown in Figure 4.6. The variance of this version is lower and no drastic reward drops are visible. Opposite to the action calculation from directly learned parameters, for projected parameters the reward recovers after a couple of episodes.

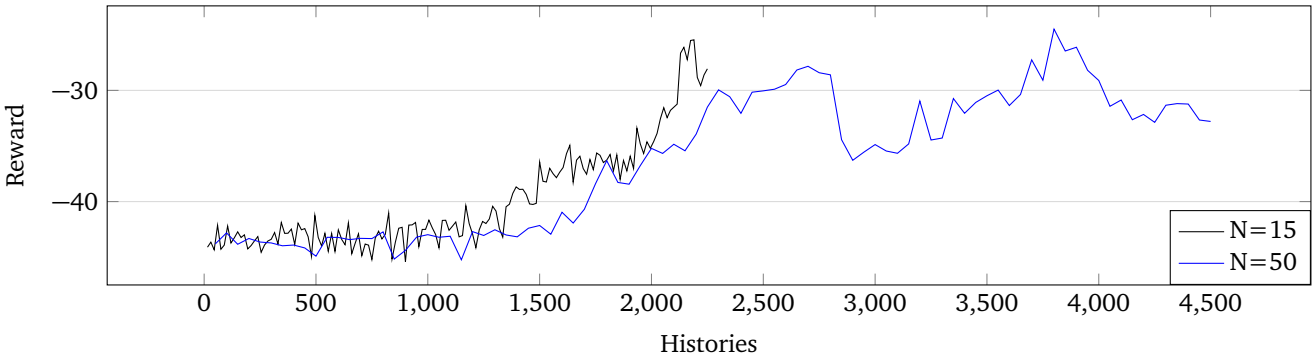

Figure 4.6: Learning progress mapped to computation time. The reward compared to the total number of sampled histories is shown for three independent trials with 50 and 15 histories per episode.

Figure 4.6 illustrates the reward per computation time. The computation time is determined by the number of histories that were sampled. The results indicate, that the number of histories have only a small impact on the reward. It seems, that the policy improvement depends primary on the total number of samples and that smaller numbers of histories and therefore a more frequently update of the upper level policy causes a faster improvement of the behavior. Additionally, the results indicate, that a small number of histories per episode leads faster to better results. Another advantage of small numbers of histories is that the reward development is smoother and the reward does not drop rapidly as it does for $N = 50$.

4.2.5 Summary of results

While the version of direct parameter learning caused parameter divergence, in the version of projected parameters the upper level policy converges to an optimal policy for the reward function and the given number of features. The number of features D is important for learning progress and the maximum reward, that can be reached with the method. Using smaller numbers of histories per episode leads to more efficient learning because a higher reward can be reached with less computational effort.
The comparison of different feature sets is difficult because for different dimensionality of state base the number of features D must be adapted.

4.3 Comparison of the results

It is difficult to compare the results directly, because sampling for the direct learning method with the same set-up as for projected parameter learning method is impossible. The results for projected parameter learning show, that only a small learning progress can be seen for a feature set based on joint positions and contact points, 50 histories per rollout and 1500 features. Comparing the graphs for direct learning in Figure 4.1 and for projected learning in Figure 4.5, which were sampled with 50 histories per episode and a state based on joint positions and contact points the learning process is similar, although trial 4.1 was sampled with 100 features while 1500 features were used in the parameter projection version. For direct learning the phase without visible learning progress is 70 episodes long, while for projected learning the reward increase begins in episode 30. In both cases a strong reward improvement is followed by a reward drop, from which the projected learning method recovers while the direct learning method reinforces the negative trend.

5 Discussion

In this chapter the results of the experiments will be discussed. A critical view on the used methods will show possibilities for future improvements. Finally, an outlook to future work is given.
While the task was learned successfully with projected parameters, with direct parameter learning the robot was not able to change the grasp.

Is this task really Markovian?
Markov decision processes depend on time independent state representations. In this approach, the state representation depends on joint positions and sensor data. The position of the bar is not directly included in the state. In cases the state base includes sensor data or contact points, which are computed from sensor data, information about the position of the bar are available. Without tactile information the a state in which the robot holds the bar in a stable grasp put in a different position, for example two centimeter higher would be the same state as being in this pose when the bar was dropped. This case is avoided by aborting the simulation when the fingers lose contact with the bar. The uncertainty about the position of the bar makes this task a partially observable Markov decision process.

Is the method suitable for the task?
The experiments show that it is generally possible to learn in-hand manipulation from tactile sensor data and joint positions without a model of the grasped item. Nevertheless it is a very complicated matter. It is useful to choose reinforcement learning, if slightly different tasks can be learned with the method. The method of directly learning parameters in its present version is not suitable for the task. To find a reward function, with which learning was possible was difficult, especially because most rollouts needed to be aborted due to the to high actions and parameters. Even with the present solution, the task was not learned. For real robot applications the increasing actions are unacceptable, especially because action restriction led to a situation where always the highest possible action was chosen. Highest action means the amount of the action, not its direction.
Using the parameter projection the task was learned. In the set-up with the state based on joint positions, 8000 used features and 15 histories per episode finger gaiting was learned in less than 90 episodes. Therefore, this version is suitable to learn the task. Other set-ups learned slower or reached a lower reward. They might receive better rewards if more features were used. In the experiments, the number of features for the other state representations was lower with regard to the number of features. A comparison of the states with an accuracy of the kernel approximation within 5% remains as feature work.

Possible improvements for direct parameter learning
After the task was learned successfully, the knowledge gained from these experiments can be applied to the direct parameter learning method. For future approaches, it will be useful to use a lower number of episodes per history because the results from the method with parameter projection indicate that few histories per episode cause smaller changes of the reward. Considering that for direct parameter learning the number of parameters to learn is defined as $number\ of\ joints\ to\ control * number\ of\ features$. Therefore it is useful to use a small state representation which requires the lowest number of features. Additionally, the state representation based on joint position has the fastest learning progress and the best rewards compared to other state representations as illustrated in Figure **??**.
Additionally, a restriction of actions or, if possible, of parameters is necessary.

5.1 Conclusion

The results of this thesis show that using Random Features in combination with directly learned parameters cannot be recommended due to the instability of the learning, the too high actions and the missing convergence of the upper-level policy parameters.
A method to successfully learn with Random Features is presented. This method uses a projection for state and parameters and needs only a small amount of parameters compared to directly learning parameters. The upper level parameters converge and the robot learns the task. It was shown that a state representation based on joint positions is sufficient for the task. Therefore, tactile feedback is only used for the reward function. Including tactile feedback in the state representation decreases the received reward but may be useful in other tasks or for trials with more episodes.

5.2 Future Work

It will be interesting to see how learning finger gaiting can be adapted for different items. A generalization of the policy to deal with different items would increase the practical use of finger gaiting policies.

After the problem of too high actions was solved by using the parameter projection method with a restriction of the actions, it will be very interesting to see how this approach works for real robots. A real robot has better friction properties than the simulated fingertip hemispheres and might with this better stability reduce the probability of dropping the bar when only a small change in the pose is done. In simulation the parameters of the upper-level policy converge after 50 episodes in the best set-up. Considering that each episode samples 15 histories which consist of two rollouts for symmetric PGPE, 1500 rollouts are necessary to learn the task. Therefore it may be useful to combine a real robot experiment with a grasping task, because resetting the hand and the object to the initial position will be time intensive.

Furthermore, it would be interesting to see whether direct parameter learning with a parameter restriction, for example using the *cos* of the parameters would lead to the same result as parameter projection.

As a next step, learning to change from an index finger grip to a middle finger grip should be learned. Therefore different grasp changes must be learned and an upper level control policy can then control the finger gaiting. This extension will increase the application area because only changing from one grasp to another without a context has a low practical use.

Bibliography

[1] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, pp. 1177–1184, 2007.

[2] O. Sigaud, C. Salaün, and V. Padois, "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1115–1129, 2011.

[3] J. Hong, G. Lafferriere, B. Mishra, and X. Tan, "Fine manipulation with multifinger hands," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 1568–1573 vol.3, May 1990.

[4] L. Han and J. C. Trinkle, "Object reorientation with finger gaiting," 1998.

[5] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. T. Pokorny, A. Billard, and D. Kragic, "Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation," *IEEE Transactions on robotics*, 2016.

[6] B. Goodwine and Y. Wei, "Theoretical and experimental investigation of stratified robotic finger gaiting and manipulation," in *Proceedings of the Thirty-Eight Annual Allerton Conference on Communications, Control and Computing*, vol. 2, pp. 906–915, Citeseer.

[7] A. Fernandez, J. Gazeau, S. Zeghloul, and S. Lahouar, "Regrasping objects during manipulation tasks by combining genetic algorithms and finger gaiting," *Meccanica*, vol. 47, no. 4, pp. 939–950, 2012.

[8] H. Maekawa, K. Tanie, and K. Komoriya, "Tactile sensor based manipulation of an unknown object by a multifingered hand with rolling contact," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 743–750 vol.1, May 1995.

[9] H. van Hoof, J. Peters, and G. Neumann, "Learning of non-parametric control policies with high-dimensional state features," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[10] A. Gijsberts and G. Metta, "Incremental learning of robot dynamics using random features," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 951–956, IEEE, 2011.

[11] J. Hoelscher, J. Peters, and T. Hermans, "Evaluation of interactive object recognition with tactile sensing," in *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2015.

[12] H. van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2015.

[13] V. Kumar, A. Gupta, E. Todorov, and S. Levine, "Learning dexterous manipulation policies from experience and imitation," *CoRR*, vol. abs/1611.05095, 2016.

[14] F. Veiga, H. van Hoof, J. Peters, and T. Hermans, "Stabilizing novel objects by learning to predict tactile slip," in *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2015.

[15] H. Dang, J. Weisz, and P. K. Allen, "Blind grasping: Stable robotic grasping using tactile feedback and hand kinematics," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5917–5922, IEEE, 2011.

[16] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," in *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2014.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[18] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics.," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[19] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," 2010.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[21] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama, "Analysis and improvement of policy gradient estimation," in *Advances in Neural Information Processing Systems*, pp. 262–270, 2011.

[22] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[23] Coppelia Robotics, "V-rep." `http://www.coppeliarobotics.com/`. Accessed: 2016-12-22.

[24] Coppelia Robotics, "Remote api." `http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm`. Accessed: 2016-12-10.

[25] Coppelia Robotics, "Ros interfaces." `http://www.coppeliarobotics.com/helpFiles/en/rosInterfaces.htm`. Accessed: 2016-12-10.

[26] "Ros wiki." `http://wiki.ros.org/vrep_ros_bridge`. Accessed: 2016-12-10.

[27] Coppelia Robotics, "Rosinterface." `http://www.coppeliarobotics.com/helpFiles/en/rosInterf.htm`. Accessed: 2016-12-10.

[28] Coppelia Robotics, "Rosplugin." `http://www.coppeliarobotics.com/helpFiles/en/rosInterfaceOverview.htm`. Accessed: 2016-12-10.