# Non-parametric Models and Contextual Policy Search for More Efficient Robot Skill Generalization

Andras Gabor Kupcsik
*Dipl.-Ing., TU Budapest*

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHYLOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2014

# Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Signed: Andras Gabor Kupcsik

Date: Tuesday, 14 January, 2014

*“We have to continually be jumping off cliffs and developing our wings on the way down."*

Kurt Vonnegut

# *Acknowledgements*

First and foremost, I thank my supervisors, Dr. Loh Ai Poh and Dr. Prahlad Vadakkepat for their support and excellent guidance during my years at NUS. They taught me how to do high quality research and to always strive for perfection in my work. They have been invaluable both at an academic and a personal level, for which I am extremely grateful.

The research for this thesis would not have been possible without financial support from the National University of Singapore (NUS) and the Singapore International Graduate Award (SINGA). I'm greatly thankful for the generous and unique offer that allowed me to study at this prestigious university.

A very special thanks goes to Dr. Jan Peters for offering me the opportunity to join his lab at the TU Darmstadt, Germany for a semester. Besides Jan Peters, I had the opportunity to work with Marc Deisenroth and Gerhard Neumann, from whom I learned a lot in doing high quality research in the fields of machine learning and robotics. I am most thankful for their excellent scientific and career advices. Additionally, I thank the whole IAS group for their kindness and hospitality during my stay in Germany.

I would like to express my sincere gratitude to my Diploma thesis advisor Dr. Balint Kiss and my former supervisor Dr. Bela Lantos from the TU Budapest. They were the two persons who motivated and encouraged me in the first place to pursue Ph.D. studies, which eventually led to this thesis. I will be always grateful for their guidance and immense help during my "early" days in research.

I owe my whole family a huge thanks for their support throughout my entire life and for always supporting my choice, such as joining NUS as a Ph.D. student. While it was sometimes difficult without them, I always felt their continuous support and care, which gave me strength and helped me through the tough days. I cannot thank enough my parents who helped me with their excellent suggestions and continuous support throughout my studies. I thank my brother for his support and for being not only a caring brother, but a great friend as well. I also owe a special thanks to my grandparents, without whom my childhood would not have been as wonderful as it was. A very special thanks goes to my godparents for their excellent motivational talks. Last, but not least, I thank my lovely girlfriend for her understanding and love during the past few years.

# Contents

# *Abstract*

Endowing robots with human-like skills has a significant impact in industrial applications, medicine, elderly care, handling emergency situations and in many applications for human-robot interaction. Designing robot skills is, however, inherently difficult due to the high dimensional continuous state-action spaces of high degree of freedom anthropomorphic robots. Most design approaches rely on accurate robot modeling and model-based optimal control methods. However, good models are often difficult to obtain due to unmodeled nonlinearities and stochasticity. Furthermore, adapting robot skills to unseen situations remains cumbersome. Thus, manually designing controllers that encode robot skills becomes a difficult, time consuming task.

As opposed to optimal control algorithms, Reinforcement Learning (RL) offers a trial-and-error approach to learn the required robot skill from experience. In recent years, Policy Search (PS) algorithms, a subclass of RL methods, have become particularly successful in learning complex skills for robot tasks. As opposed to standard RL algorithms, PS methods scale well for robot learning tasks with high-dimensional continuous state-action spaces. Furthermore, expert demonstration can be exploited to initialize the learning process. However, standard model-free PS algorithms are inherently data-inefficient. Even for simpler tasks, model-free approaches require an overly large amount of interactions with the real robot to learn the task. Furthermore, most algorithms are designed for learning a fixed task, without taking changing situations into consideration.

The main goal of this thesis is to propose new approaches and algorithms to bring robot skill learning closer to real applications by improving the learning efficiency. The thesis mainly focuses on learning generalized robot skills using contextual policy search and learning nonlinear stochastic models of robot dynamics for application in model-based RL.

We first present the contextual Relative Entropy Policy Search (REPS), a PS algorithm based on information theoretic insights. The contextual extension of REPS constrains the experience loss between policy updates, and thus, the algorithm provides smooth learning and safe exploration, which is essential for real robot applications. We show in several complex robot learning problems that the resulting policy search framework is able to robustly learn the robot skill, while outperforming other approaches.

Despite the good learning performance of contextual REPS, the data inefficiency of the algorithm prevents efficient application for real robot tasks. Thus, we propose the Gaussian Process Relative Entropy Policy Search (GPREPS), a model-based extension of contextual REPS using Gaussian Process (GP) models. GP models use non-parametric Bayesian regression to capture model nonlinearity and stochasticity. The resulting extension not only improves data-efficiency, but integrates out model uncertainty, resulting in higher quality learned skills. We show in simulations and in a real robot experiment that GPREPS is able to improve data-efficiency in complex learning tasks up to two orders of magnitude. Furthermore, we propose a novel probabilistic model learning framework, which directly minimizes the divergence between the observed and the predicted trajectory distributions. We discuss possible application with Gaussian Process models.

Trajectory prediction with probabilistic models often becomes challenging for high dimensional state-action spaces, due to assumptions on state transition models. We propose a novel probabilistic non-parametric modeling method for learning the robot dynamics and to perform trajectory predictions. We use kernel embedding of conditional distributions to evaluate the prediction in feature spaces associated with kernel functions. Manipulation of distributions in the feature space becomes linear algebraic operations with Gram matrices. Thus, the resulting algorithm avoids modeling assumptions and computational approximations, making it simple and straightforward to implement. We show in simulated results that the proposed method provides better trajectory prediction accuracy and it is computationally more efficient, compared to Gaussian Process models.

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| **DMP** | **D**ynamic **M**ovement **P**rimitive |
| **DoF** | **D**egree **o**f **F**reedom |
| **EM** | **E**xpectation **M**aximization |
| **GP** | **G**aussian **P**rocess |
| **KL** | **K**ullback–**L**eibler |
| **MDP** | **M**arkov **D**ecison **P**rocess |
| **PD** | **P**roportional **D**erivative |
| **PG** | **P**olicy **G**radient |
| **PS** | **P**olicy **S**earch |
| **REPS** | **R**elative **E**ntropy **P**olicy **S**earch |
| **RL** | **R**einforcement **L**earning |
| **SVM** | **S**upport **V**ector **M**achine |
| **w.r.t.** | **w**ith **r**espect **t**o |
| **i.i.d.** | **i**ndependent and **i**dentically **d**istributed |

# Symbols and Notations

| | |
|---|---|
| $t$ | time |
| $\boldsymbol{x}$ | state |
| $\boldsymbol{u}$ | control |
| $\boldsymbol{s}$ | context |
| $\boldsymbol{\omega}$ | control policy parameter |
| $\boldsymbol{\tau}$ | trajectory |
| $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ | angular position, velocity and acceleration of robot joints |
| $\boldsymbol{\theta}$ | hyper-parameters of a function |
| $\pi_{\boldsymbol{\omega}}(\boldsymbol{u}\|\boldsymbol{x})$ | control policy with parameters $\boldsymbol{\omega}$ |
| $\pi_{\boldsymbol{\theta}}(\boldsymbol{\omega})$ | upper-level policy with parameters $\boldsymbol{\theta}$ |
| $\boldsymbol{\phi}(\boldsymbol{x})$ | feature function |
| $k(\boldsymbol{x}, \boldsymbol{x}')$ | kernel function |
| $\boldsymbol{\mu}_{\boldsymbol{x}}$ | mean embedding of distribution $p(\boldsymbol{x})$ |
| $\mathcal{C}_{\boldsymbol{y}\|\boldsymbol{x}}$ | conditional covariance operator |
| $r(\boldsymbol{x}, \boldsymbol{u})$ | immediate reward |
| $R(\boldsymbol{\tau})$ | trajectory (episode) reward |
| $\mathcal{R}_{\boldsymbol{\omega}}$ | expected trajectory (episode) reward |
| $J(\boldsymbol{\theta})$ | objective function with parameter $\boldsymbol{\theta}$ |
| $\lambda$ | ridge factor |
| $KL(p(\boldsymbol{x})\|\|q(\boldsymbol{x}))$ | KL-divergence |
| | |
| $\{\boldsymbol{x}_i\}_{i=1}^N$ | set of elements $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ |
| $\boldsymbol{x} = [x_1, \ldots, x_N]^T$ | vector |

| | |
|---|---|
| $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ | matrix |
| $\boldsymbol{X}^{-1}$ | matrix inverse |
| $\boldsymbol{X}^T$ | matrix transpose |
| $\mathbb{R}$ | real numbers |
| $p(\boldsymbol{x})$ | probability density of $\boldsymbol{x}$ |
| $\mathbb{E}[\boldsymbol{x}]$ | expectation of $\boldsymbol{x}$ |
| $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ | derivative of $J(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ |
| $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ | partial derivative |

*I dedicate this thesis to my beloved family.*

# Chapter 1

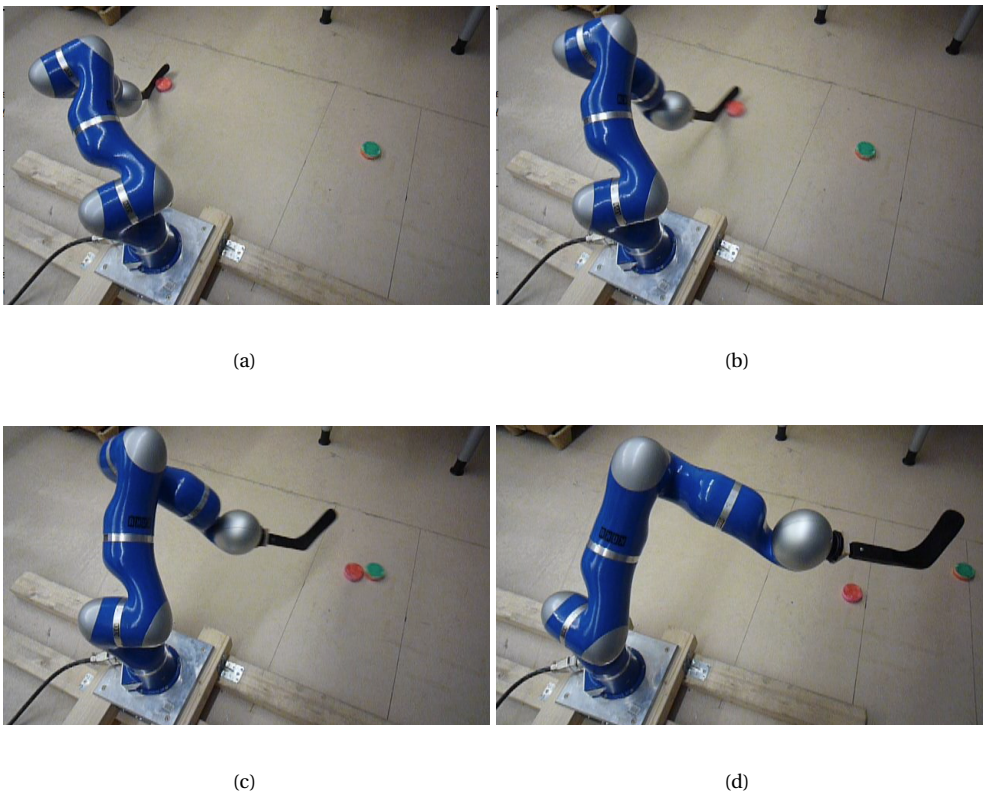
# Introduction

It has been a long standing vision of robotics research to endow robots with dexterous skills to solve complex tasks in human environments. Controllers that encode such skills are predominantly set by experienced engineers. This approach usually requires years of training and typically assumes perfect knowledge of the robot and its environment. Hand-tuning robot skills becomes particularly challenging in the face of uncertainties and with changing environmental conditions. For robot skill programming tasks, one major challenge is to predict the influence of the control actions over time (torques, desired accelerations) on the objective function we wish to maximize. As we typically use a high level objective, which is difficult to capture solely by the states and actions of the robot, we need to accurately predict the arguments of the objective function. For example, in a ball throwing task we might define the objective as the minimal distance of the ball to the target position. In general, the prediction of such quantities requires the highly accurate modeling of the dynamics

involved in the task, which is difficult due to nonlinear effects and stochasticity. Additionally, even with a very good prediction model, we still have solve a complex sequential decision making problem, which provides the optimal control actions over several time steps. Building models and solving the emerging sequential decision making problem are both highly challenging and time consuming for an expert engineer, even for simpler tasks. Furthermore, to account for changing situations we would have to re-program the robot, which is clearly impractical.

We illustrate these challenges on a real problem. Consider the robot hockey task depicted in Figure 1.1. The goal is to acquire a hitting skill with a 6-DoF robot in a hockey game such that the target puck (in green color) is moved to a certain distance after a collision with the robot's puck (in pink color, Fig. 1.1(a)). Thus, the robot not only has to shoot in the right direction, but with the right force. The robot is equipped with a hockey racket and the pink puck is placed just before the racket (Fig. 1.1(a)). The robot executes the hitting motion (Fig. 1.1(b)) and the two pucks collide (Fig. 1.1(c)). After a blink of an eye the final position of the robot and the pucks are reached (Fig. 1.1(d)).

In order to program this skill to the robot, that is, to generate the required torque commands over time, we have to have a good understanding of the dynamics models involved in the experiment. However, building these models becomes particularly challenging due to the high level of complexity in the dynamics. First, we need to be able to predict how the hockey racket motion influences the puck's trajectory while pushing it. Second, we have to have an accurate model for the sliding motion of the puck and the collision model. Moreover, the dynamics might change over time, or



(a) (b)

(c) (d)

FIGURE 1.1: **(a)** The robot's pink puck is placed before the hockey racket. The green target puck is further away, such that the robot cannot reach it. **(b)** The robot executes the hitting motion. **(c)** The moment when the two pucks collide. **(d)** The final positions of the pucks and the robot after the experiment.

might show stochastic behavior. Additionally, in many cases the task might vary between skill executions. For example, the position of the target puck might vary in a certain range. Manually programing the skill for each possible position is clearly impractical. Thus, hand-tuning robot trajectories already for this simpler task becomes highly challenging and time consuming even for experienced engineers.

In order for robots to adapt to new situations and improve upon their acquired skills without profound knowledge of the environment, the machine learning community has offered novel methods to allow robots to learn from experience. The key idea

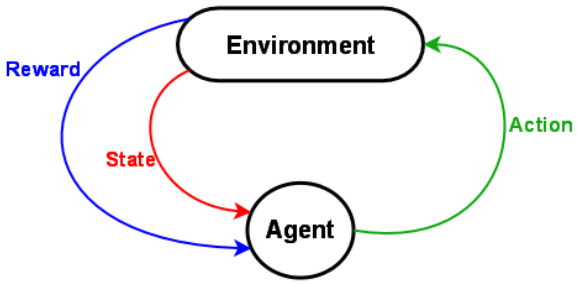



FIGURE 1.2: The reinforcement learning setup. The agent observes the state of the environment and uses its policy to choose an action. Given the action, the environment alters its state and provides a reward for the agent.

behind these approaches is to enable robots to learn the desired skill by interacting with the environment and evaluating their success in the given task. The resulting trial-and-error approach has yielded promising results with different robotic systems, such as arms, bipeds and mobile robots. Despite the success with simpler robotic tasks, it is currently difficult to achieve learning of complex and dexterous robots skills, such as humanoid locomotion, grasping, or playing games, such as table tennis and soccer.

One of the earliest class of algorithms that has proven to be successful in robot learning tasks is Reinforcement Learning (RL). In RL [80, 81], an agent interacts with the environment by observing the state of the environment and choosing an appropriate action using its control policy (Fig. 1.2). However, the agent does not know which is the optimal action to choose, but receives a reward signal about how successful the chosen action is. By repeatedly interacting with the environment, the agent builds up a history of observed states, actions and rewards. Using the accumulated experience, the agent updates its control policy to improve the expected future reward. In robot control tasks, the agent corresponds to a controller, which provides the control signal, such as desired acceleration or torque. The environment represents the

dynamics of the robot and its surroundings. The states are typically defined as joint angles, angle velocities and accelerations and other task relevant information, such as object position, etc. The reward signal is typically defined by the user and it encodes the desired goal for the learning task. Such objectives may include low energy consumption, distance to goal states, or safety.

While the goal and methods of RL closely relates to that of optimal control [8], there is a significant difference in the assumed prior knowledge. In optimal control, the exact model of the controllable system, or at least the structure of the model is usually assumed to be known. In most RL algorithms however, the agent solely learns from its experience of observed states, rewards and actions, without any prior knowledge about the model of the environment or the structure of the reward signal. The resulting approach to solve sequential decision making problems makes RL applicable in many other disciplines, such as finance, games, etc. On the other hand, due to the generality of RL, the underlying problem often becomes difficult to solve. For example, the agent repeatedly faces the problem of choosing an action using limited information. On one hand the agent is interested in choosing a greedy action, which maximizes the expected reward based on its accumulated experience. On the other hand, the agent is motivated to try out novel actions that possibly lead to higher (or lower) rewards. Choosing an action in the presence of the two contradictory objectives is referred to as the exploration-exploitation dilemma.

While early reinforcement learning algorithms were successful in solving problems with low-dimensional, discrete state-action spaces, direct application for more complex robotic tasks becomes cumbersome. One of the most significant obstacle when

applying RL algorithms for robot learning tasks is the high dimensionality of the state space[1]. Standard RL algorithms typically learn the value function of states, which measures how beneficial it is to reach the given state for maximizing the expected reward. However, the number of states grows exponentially with the state dimensionality, and thus, solving higher dimensional robot tasks becomes impractical. While function approximation helps to mitigate the problem, standard RL algorithms are not the preferred class of methods to solve complex robot skill learning tasks. Other obstacles that prevent the straightforward application of RL in robot learning tasks are the partial observability of the state space, noisy measurements and the difficulty to define an appropriate reward function that reflects our expectations of the learning outcome.

Starting from the early 1990's, with the pioneering work of Williams [88], Policy Search (PS) algorithms became one of the most successful subclass of RL algorithms, primarily due to their success in higher dimensional robot learning problems [16]. In PS, the agent explicitly represents its control strategy as a parametrized policy, and its goal is to find the optimal parameters of the policy that maximizes the expected reward. By omitting the explicit use of value functions, PS algorithms scale well to higher dimensional, continuous state-action spaces, which is typical for robot skill learning tasks. Successful applications of PS algorithms enabled robots to play table tennis, tether ball, to drum, throw darts, play hockey and many more [33].

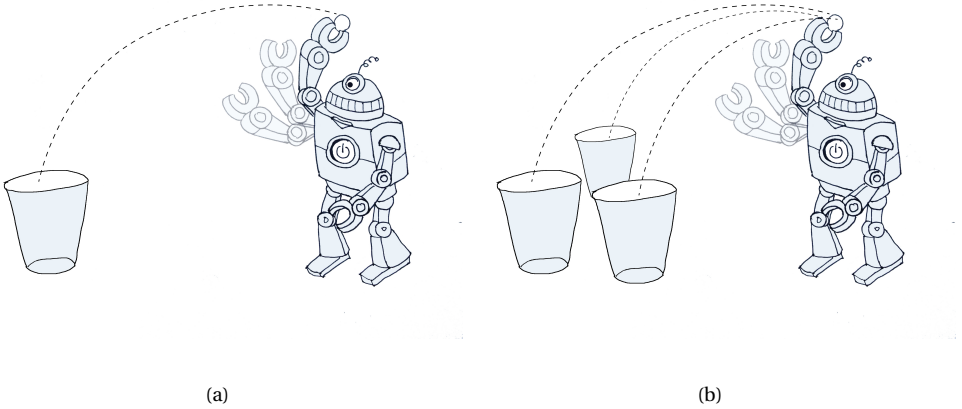
Like standard RL algorithms, PS methods exhibit a high level of generality to solve

[1] For robot learning tasks, a state dimensionality of 10 - 30 is considered high. If we define the state as the collection of angular positions and velocities at each joint, a 5 to 15 degree of freedom robot is considered complex.

learning problems. For many learning problems however, we might sacrifice generality and introduce prior knowledge about the task to improve learning efficiency. Such assumptions might manifest in the form of a structured reward function or the model of the robot and its environment. However, learning algorithms notoriously exploit imperfections of the prior knowledge, e.g., using a biased model could lead to a poor performance in the real task. As learning a good model of the robot and its environment is substantially more difficult than learning the policy itself, model-free RL and PS algorithms are often preferred over model-based methods. While model-free algorithms learn unbiased policies, they typically require thousands of robot experiment evaluations to achieve the learning of a high quality skill. This becomes a significant disadvantage when working with real robots, as experiments are typically costly, time consuming, require expert supervision and might lead to premature robot wear and damage. Model-based RL and PS algorithms address the data inefficiency by learning a model of the robot and its environment and use it as a simulator to replace the real robot experiments. Using models not only decreases the required amount of interactions with the real robot, but also helps to generalize in unexplored parts of the state-action space. Despite their favorable properties, model-based approaches are less frequently applied, as learning models of high dimensional nonlinear robot dynamics is challenging, especially in the face of uncertainty. If unbiased models of the robot dynamics are available, the optimized robot skill will perform well in the real task.

To learn unbiased models of nonlinear robot dynamics, we have to take several requirements into consideration. Firstly, we need probabilistic models, which can account for model uncertainty. Secondly, the model has to be able to capture the nonlinearity in the dynamics. Most modeling methods typically involve fixing the model class and finding the optimal parameters of the model that best describes the observed data. However, this approach may still be unsatisfactory if the model is not perfect. In recent years, non-parametric models have proven to be highly efficient in learning nonlinear robot dynamics. Non-parametric methods avoid specifying the model class, instead, the model structure is implicitly captured in the observed training data and their features. Despite the good generalization property, the computational demand might be impractically large [62]. However, promising results have been shown with Gaussian Process models for learning, e.g., the dynamics of a 7-DoF tendon-driven robot arm [21].

While both model-free and model-based PS algorithms have been successfully applied for many robot skill learning tasks, most experiments consider only a fixed environmental setup. Generalizing robot skills for multiple task setups allows robots to become more flexible and adaptable to changing situations. For example, consider the learning task where the robot has to learn a throwing skill in order to land a ball in a basket (Fig. 1.3(a)). While we can solve this task with standard policy search methods, generalization of the task for changing basket positions (Fig. 1.3(b)) becomes a challenging problem. Re-learning the task for every possible position clearly leads to an impractical learning approach. However, due to the structured nature of



(a) (b)

FIGURE 1.3: **(a)** The robot has to learn a throwing skill, where the objective is to land the ball in the basket located at a *fixed* position. **(b)** The robot has to learn a generalized throwing skill, where the basket position might *change* in a certain range between experiments. Here we show three example basket positions.

motor skills, we can exploit the high similarity of optimal throwing strokes for different basket positions, and learn a generalized skill, which provides accurate throwing skills for a wide range of basked positions. Currently there are only a few algorithms that exploit such a structure for robot skill generalization. Most of these methods are model-free and require an overly large number of interactions with the robot to learn the task.

In order to learn high quality robot skill for complex robots, such as humanoids, we need new tools and learning methodologies that are able to scale to higher dimensional tasks. This will require the exploitation of prior knowledge in the form of models and pre-structured policies, and efficient generalization by learning algorithms.

## 1.1 Contributions

Motivated by recent progress in model-free policy search [55] and promising results of non-parametric model-based policy search [17], we propose novel techniques and algorithms in this thesis for more efficient robot skill learning. We believe that the proposed algorithms provide significant contributions for advancing robot skill learning and narrowing the gap between current applications and the vision of future robots with super-human skills. The main contributions of the thesis are as follows.

- **Generalized robot skill learning.** Robot skill generalization and learning are often treated as separate problems. One of the most common approach to generalize robot skills is to record tasks demonstrated by a human expert and interpolating over these demonstrations to obtain the skill in a new, yet unseen situation. This method typically achieves good performance in the demonstrated region, but the generalization accuracy inherently depends on the quality and the quantity of demonstrated skills. To account for suboptimal demonstrations, learning algorithms are often proposed as an extension. However, the two problems are rarely addressed in a single framework, resulting in slow learning and less effective generalization property. To address this problem, we evaluate a novel contextual PS algorithm, the Contextual Relative Entropy Policy Search (REPS), which treats learning and generalization in a single framework. The algorithm naturally incorporates skill initialization by demonstration and self improvement by learning. The method is tested in complex tasks, such as simulated robot hockey, throwing and table tennis.

- **Model-based framework for robot skill generalization.** So far, data-efficiency has not been addressed in the context of learning generalized robot skills. We propose a novel, model-based contextual policy search framework that unifies probabilistic, non-parametric modeling with information theoretic contextual policy search. We will discuss how to use Gaussian Process dynamics models in this framework to account for the shortcomings of the model-free policy search setup. We show in complex simulated and real robot experiments that the proposed model-based contextual learning algorithm is able to reduce real robot experiments up to two orders of magnitude compared to the model-free variant, while learning higher quality policies.

- **Probabilistic model learning for trajectory prediction.** A key component of many model-based robot learning algorithms is the probabilistic modeling of the robot dynamics. Time independent dynamics models are typically used to predict the expected experiment outcome. Such models are often trained to maximize the likelihood of the observed state transitions, given the observed data. While this leads to a tractable training approach, which is easy to implement, it does not necessarily maximize the likelihood of the observed trajectories. This might lead to the problem of inconsistent trajectory prediction. To address this problem, we propose a general probabilistic model learning framework based on information theoretic insights. Our method directly maximizes the likelihood over the trajectories while the computations remain tractable. We will discuss possible applications for Gaussian Process models.

- **Kernel embedding of trajectory distributions:** Kernel methods have become

popular in the machine learning community in the last decade. Recently, works on embedding conditional probability distributions into feature spaces has opened the door for applications of probabilistic inference. As features might be high, or even infinite dimensional, computations in the feature spaces become involved and they are likely to be intractable. However, due to the virtue of the "kernel-trick", we do not have to work with features, but with their inner product defined by a kernel function. When embedding distributions into feature spaces, the product rule, the sum rule and the Bayes' rule become linear algebraic operations with Gram matrices of the training data. Besides having tractable computations, in feature spaces we do not restrict the predictive distribution to a parametric class. Instead, the distribution structure is implicitly represented by the training data and the kernel function. As predicting trajectories using probabilistic models is merely the repeated application of the sum rule, obtaining the expected trajectory in feature space becomes surprisingly simple as opposed to standard prediction techniques, where we often use assumptions and approximations. We present a novel trajectory prediction algorithm and discuss model training. We show that by using kernel embedding of trajectory distributions, we obtain better generalization properties compared to conventional prediction methods.

## 1.2 Thesis Outline

The thesis structure is shown in Fig. 1.4.

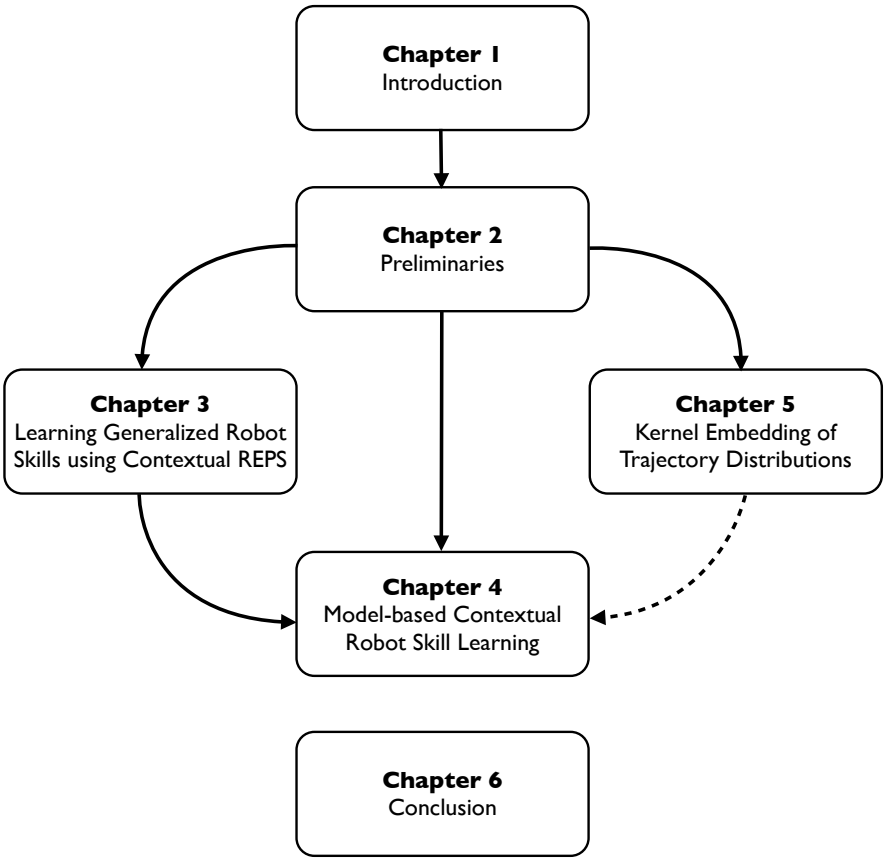



FIGURE 1.4: The thesis structure. The solid lines express dependency between chapters, while the dashed lines refer to related, but not directly dependent chapters.

**Chapter 2: Preliminaries.** First, we explain how parametrized skills can be improved by reinforcement learning. We will review the basics of model-free and model-based policy search algorithms and we show successful examples of robot skill learning. Subsequently, we will discuss robot skill representation techniques with a special focus on motor primitives. Finally, we will review the theoretical background of Gaussian Process regression and kernel embedding of conditional distributions.

**Chapter 3: Learning Generalized Robot Skills using Contextual REPS.** In this chapter we evaluate Contextual REPS, a contextual policy search algorithm for learning and generalizing robot skills. We give an overview of related methods and point out their shortcomings and advantages. We show in complex skill learning tasks that the algorithm achieves superior performance over existing methods. The research lead to these results were published in [41, 42, 48].

**Chapter 4: Model-based Contextual Robot Skill Learning.** We propose a model-based learning architecture, which builds on the generalization algorithm discussed in Chapter 3. We show how non-parametric models can be used in this learning framework. We present a computationally efficient and easy to implement algorithm to sample from trajectory distributions. We test the learning framework with the model-free variant and show simulation and real-world experiment results. In this chapter we will also discuss a general probabilistic model learning framework for accurate trajectory prediction, which is essential for model-based RL algorithms. We show that our approach provides tractable solutions and we discuss possible applications for Gaussian Process models. The research lead to these results were partly published in [41, 42, 48].

**Chapter 5: Kernel Embedding of Trajectory Distributions.** In this chapter we show how we can obtain the expected trajectory in the feature space and how we can train the model that generates the embedding. We show in simulated results that the proposed algorithm provides better generalization properties compared to, e.g., Gaussian Process models, while the computations remain simple and tractable. The research lead to these results will be submitted to a machine learning conference in the

close future.

**Chapter 6: Conclusion.** This chapter concludes the work, and discusses future work.

# Chapter 2

# Preliminaries

In this Chapter, we will review the essential theoretical background of the thesis. First, we give a detailed overview of Policy Search methods. We begin with model-free policy search for a fixed environmental setup and present successful applications for robot skill learning. Subsequently, we review several contextual PS algorithms that have been successfully applied for robot skill generalization. We then turn our attention to model-based policy search methods to discuss the applied techniques, results and challenges. After reviewing the essentials of skill learning with PS algorithms, we will discuss skill representation techniques, which can be improved by learning algorithms. We will focus our overview on representational ability and adaptability, followed by the theoretical backgrounds of non-parametric modeling techniques, which will be essential for improving the efficiency of skill learning algorithms.

## 2.1 Policy Search

Policy Search algorithms are one of the most successful subclass of RL algorithms for learning complex robot skills [16]. In the following, we use two different factorization of PS algorithms. First, we distinguish between model-free and model-based methods. While model model-free methods can be considered as general stochastic optimization algorithms, model-based approaches exploit prior knowledge about the task, and thus, they can only be applied for a limited class of problems. PS algorithms may also be differentiated between context-free and context-based approaches. Context-free methods learn a policy for a fixed task, while context-based policies condition on a task specific context variable, e.g., the basket position in the ball throwing task (Fig. 1.3). We assume that the context variable is observable and it cannot be influenced by the policy. Thus, context-based algorithms aim to learn a generalized skill, that is able to provide good policy parametrization for a larger variety of context variables. We first give an overview of model- and context-free PS methods, which form the basis for context-based and model-based PS algorithms.

### 2.1.1 Model-Free Policy Search

The key concept behind model-free policy search algorithms is to update the policy solely based on action-reward samples evaluated on the real system, without any assumption, or knowledge about the system dynamics. The learning loop of model-free PS methods can be divided into three distinct steps. First, we draw an explorative action using the current policy in the hope of obtaining higher rewards. Subsequently, the chosen action is evaluated on the real system to obtain the reward. Finally, the

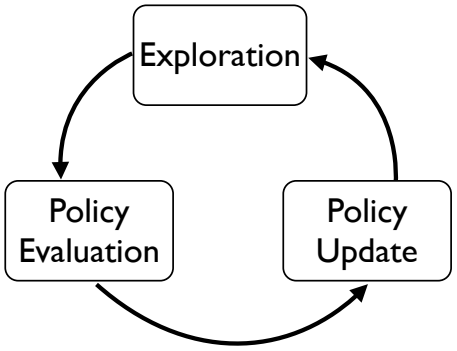



FIGURE 2.1: The learning loop of model-free policy search algorithms.

policy parameters are updated according to the acquired knowledge. In Fig. 2.1 we show the learning loop of model-free policy search.

For skill learning, we define the skill of the robot as the trajectory consisting of states and actions $\boldsymbol{\tau} = \{\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{x}_T, \boldsymbol{u}_T\}, T < \infty$ with joint angles $\boldsymbol{q}$ and angular velocities $\dot{\boldsymbol{q}}$ as states $\boldsymbol{x} = [\boldsymbol{q}^T, \dot{\boldsymbol{q}}^T]^T$ and desired motor torques or accelerations as controls $\boldsymbol{u}$. When learning the skill, we keep $T$ fixed for each trajectory evaluation. In the following, we refer to evaluating a trajectory $\boldsymbol{\tau}$ of length $T$ as an *episode*, while the transition from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ using control $\boldsymbol{u}_t$ is one *step*. The control signal is generated using either a deterministic $\boldsymbol{u} = \pi_{\boldsymbol{\omega}}(\boldsymbol{x})$, or a stochastic control policy $\boldsymbol{u} \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x})$ with parametrization $\boldsymbol{\omega}$. The objective of standard PS algorithms is to obtain the optimal policy parametrization $\boldsymbol{\omega}^*$ that yields the highest expected reward

$$J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{\tau}}[R(\boldsymbol{\tau})] = \int p(\boldsymbol{\tau};\boldsymbol{\omega}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}, \tag{2.1}$$

where $R(\boldsymbol{\tau})$ is the reward for executing trajectory $\boldsymbol{\tau}$ and $p(\boldsymbol{\tau};\boldsymbol{\omega})$ is the trajectory distribution given parametrization $\boldsymbol{\omega}$. In order to find policy parameters that possibly lead

to higher rewards, we have to incorporate exploration into the PS learning framework. In the earliest PS algorithms [7, 79, 88], and also in many recent approaches [36, 57, 78], the exploration is introduced by a stochastic control policy. For each time step $t$, the control signal $\boldsymbol{u}_t$ is drawn from distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x}_t)$ after observing state $\boldsymbol{x}_t$. As this control strategy takes an explorative action at each time step, we refer to this exploration technique as *step-based*. To evaluate the explorative controls, we use an immediate reward function $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and approximate the trajectory reward as the sum over the immediate rewards $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

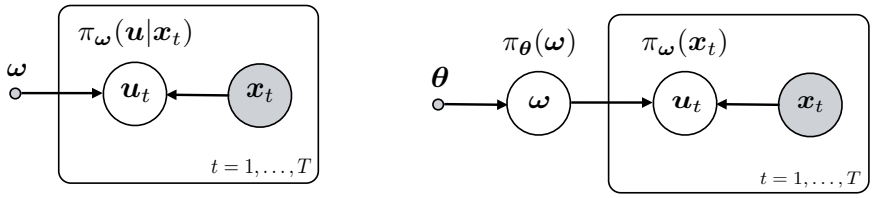
While exploiting the model of the immediate rewards is beneficial to reduce the variance of policy parameter update, this approach is typically limited in the class of control policies, i.e., most approaches require linear control policies [36, 58]. Furthermore, when using step-based exploration techniques, we often face the credit assignment problem: it is not clear what is the influence of control $\boldsymbol{u}_t$ on future rewards $r(\boldsymbol{x}_i, \boldsymbol{u}_i),\ i > t$, as future states and rewards will depend on past controls as well. Additionally, due to the low-pass filter nature of robot dynamics, the effect of high frequency exploration might be suppressed. Finally, the reward for many stroke-based tasks, such as throwing and hitting, cannot be expressed solely by immediate rewards, but as a function of the whole robot trajectory $\boldsymbol{\tau}$. For these problems, *episode-based* PS algorithms [14, 30, 35, 37, 39, 47] have been proposed.

Instead of directly optimizing the control policy parameters $\boldsymbol{\omega}$, episode-based PS algorithms learn the parameters $\boldsymbol{\theta}$ of an upper-level policy $\boldsymbol{\omega} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{\omega})$. The upper-level policy provides the parametrization for the typically deterministic lower-level control policy $\boldsymbol{u} = \pi_{\boldsymbol{\omega}}(\boldsymbol{x})$. This controller is then used to obtain the trajectory distribution

$p(\boldsymbol{\tau};\boldsymbol{\omega})$. The lower-level policy parameter is fixed at the beginning of the episode and it is kept fixed throughout the episode. The learning objective for episode-based PS algorithms is to maximize the expected reward over policy parameters

$$\begin{aligned} J(\boldsymbol{\theta}) &= \int \pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \int p(\boldsymbol{\tau};\boldsymbol{\omega}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} d\boldsymbol{\theta} \\ &= \int \pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \mathcal{R}_{\boldsymbol{\omega}} d\boldsymbol{\omega}, \end{aligned} \tag{2.2}$$

where $\mathcal{R}_{\boldsymbol{\omega}} = \int p(\boldsymbol{\tau};\boldsymbol{\omega}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$ is the expected reward for executing the lower-level policy with parameterization $\boldsymbol{\omega}$. By directly optimizing in parameter space, any pre-structured lower-level control policy is applicable and more sophisticated exploration strategies can be implemented. On the other hand, by assigning a single reward value $\mathcal{R}_{\boldsymbol{\omega}}$ for the whole episode, we fail to exploit the structure of possible immediate rewards, and thus, we might end up with higher parameter update variance. The upper-level policy is typically represented as a Gaussian $\pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_{\boldsymbol{\omega}}, \boldsymbol{\Sigma}_{\boldsymbol{\omega}})$, with parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_{\boldsymbol{\omega}}, \boldsymbol{\Sigma}_{\boldsymbol{\omega}}\}$. Following [16], in Fig. 2.2 we show the graphical models of the two exploration strategies. While step-based exploration (Fig. 2.2(a)) uses




(a) (b)

FIGURE 2.2: **(a)** The graphical model of the step-based exploration strategy. The lower-level control policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x}_t)$ is stochastic. **(b)** The graphical model of the episode-based exploration. The exploration happens in parameter space, the lower-level policy is usually deterministic.

a stochastic lower-level control policy, algorithms with episode-based exploration (Fig. 2.2(b)) avoid the credit assignment problem by exploring in parameter space, using a deterministic control policy. Up to this point we compared the learning objectives, exploration and evaluation strategies of step-based and episode-based PS algorithms. We now turn our attention to the policy update techniques of the two approaches.

**Gradient-based techniques.** One of the earliest successful PS learning algorithms were Policy Gradient (PG) techniques [7, 46, 57, 63, 79, 82, 88]. The basic idea behind PG methods is to obtain the gradient of the objective function

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \int \nabla_{\boldsymbol{\omega}} p(\boldsymbol{\tau};\boldsymbol{\omega}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{2.3}$$

using it to update the parameters

$$\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \alpha \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}), \tag{2.4}$$

where $\alpha$ is a user specified learning rate. Assuming the robot model is Markovian, we can factorize the trajectory distribution

$$p(\boldsymbol{\tau};\boldsymbol{\omega}) = p(\boldsymbol{x}_0) \prod_{t=1}^{T-1} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) \pi_{\boldsymbol{\omega}}(\boldsymbol{u}_t|\boldsymbol{x}_t). \tag{2.5}$$

By using the identity $\nabla_{\boldsymbol{\omega}} p(\boldsymbol{\tau};\boldsymbol{\omega}) = p(\boldsymbol{\tau};\boldsymbol{\omega})\nabla_{\boldsymbol{\omega}} \log p(\boldsymbol{\tau};\boldsymbol{\omega})$, the policy gradient can be computed *without* the knowledge of the transition dynamics $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{\tau}} \left[ \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\omega}} \log \pi_{\boldsymbol{\omega}}(\boldsymbol{u}_t|\boldsymbol{x}_t) R(\boldsymbol{\tau}) \right] \tag{2.6}$$

An important extension to standard PG algorithms is the use of a baseline $b \in \mathbb{R}$

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{\tau}} \left[ \sum_{t=1}^{T-1} \nabla_{\boldsymbol{\omega}} \log \pi_{\boldsymbol{\omega}}(\boldsymbol{u}_t|\boldsymbol{x}_t)(R(\boldsymbol{\tau}) - b) \right]. \tag{2.7}$$

The baseline does not change the value of the expected gradient and it can be chosen such that it minimizes the variance of the gradient [88]. Baxter et al. [7] and Sutton et al. [79] showed that the gradient variance can further be reduced by exploiting the immediate rewards and introducing a time dependent bias term. In the late 2000's Peters et al. [57, 58] suggested the use of the natural gradient to account for smooth learning. By replacing the standard PG with natural gradients, the information loss between subsequent policies can be upper bounded, resulting in safe exploration and smoother learning performance. By Tang et al. in [82] it was shown that the policy gradient computation in (2.6) is a special case of an importance sampled gradient computation approach, where we only use the current policy outcome to compute the gradient. As it was shown in [82], building on the idea of importance sampling, we can efficiently reuse previous policy evaluations to compute the current gradient. Moreover, the formula for a generalized unbiased baseline is also given.

The policy gradient technique can straightforwardly be applied to episode-based PS algorithms [64, 65, 71, 90]. For episode-based algorithms the policy gradient, similar

to Eq. (2.6), is computed as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\omega}} \left[ \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}) \mathcal{R}_{\boldsymbol{\omega}} \right]. \tag{2.8}$$

While gradient variance reduction can be achieved using baselines [64, 90], the immediate rewards cannot be used to improve the gradient. Nevertheless, promising results have been shown using natural gradients [90]. However, one significant disadvantage of gradient-based techniques is the hand tuned learning rate, which is often difficult to choose, but crucial for good learning performance. A good learning rate can often be found after repeated restarts of the learning process, which is clearly impractical for real robot learning tasks.

Episode-based model-free policy search algorithms can often be considered as general black-box optimization methods. Thus, for episode-based model-free robot skill learning, we can use other black-box optimizers, such as evolutionary computation techniques, e.g., the CMA-ES algorithm [29]. However, these techniques tend to explore aggressively, which we generally try to avoid with robot skill learning due to safety reasons.

**Expectation Maximization Based Algorithms.** As it was shown in [36] and in [16], we can write up the optimization problem using the EM approach as maximizing the log-likelihood of the reward event $p(R = 1)$, or simply $p(R)$. We introduce the variational trajectory distribution $q(\boldsymbol{\tau})$ to decompose the log-likelihood

$$\log p(R;\boldsymbol{\omega}) = \int q(\boldsymbol{\tau}) \log \frac{p(\boldsymbol{\tau}, R;\boldsymbol{\omega})}{q(\boldsymbol{\tau})} d\boldsymbol{\tau} + \int q(\boldsymbol{\tau}) \log \frac{q(\boldsymbol{\tau})}{p(\boldsymbol{\tau}|R;\boldsymbol{\omega})} d\boldsymbol{\tau}. \tag{2.9}$$

The first term represents a lower-bound for the likelihood function, as the second term (the divergence $KL(q(\boldsymbol{\tau})||p(\boldsymbol{\tau}|R;\boldsymbol{\omega}))$[1]) is non-negative. To compute the second term, we use $p(\boldsymbol{\tau}|R;\boldsymbol{\omega}) \propto p(R|\boldsymbol{\tau})p(\boldsymbol{\tau};\boldsymbol{\omega})$ and we typically assume that $p(R|\boldsymbol{\tau}) \propto \exp(R(\boldsymbol{\tau})/\eta)$, where $\eta$ is a hand-tuned constant parameter. Thus, in the E-step of the $i^{th}$ policy update we choose $q(\boldsymbol{\tau}) = p(R|\boldsymbol{\tau})p(\boldsymbol{\tau};\boldsymbol{\omega}_i)$ to set the KL divergence to 0, which will provide a tight lower bound for the log-likelihood.

In the M-step our goal is to maximize the expected complete data log-likelihood w.r.t. the policy parameters $\boldsymbol{\omega}_{i+1}$, that is,

$$\boldsymbol{\omega}_{i+1} = \arg\max_{\boldsymbol{\omega}} \int q(\boldsymbol{\tau}) \log \frac{p(R|\boldsymbol{\tau})p(\boldsymbol{\tau};\boldsymbol{\omega})}{q(\boldsymbol{\tau})} d\boldsymbol{\tau} + \int q(\boldsymbol{\tau}) \log q(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{2.10}$$

$$= -KL(\underbrace{p(R|\boldsymbol{\tau})p(\boldsymbol{\tau};\boldsymbol{\omega}_i)}_{q(\boldsymbol{\tau})}||p(\boldsymbol{\tau};\boldsymbol{\omega})) + \underbrace{\int q(\boldsymbol{\tau}) \log p(R|\boldsymbol{\tau})q(\boldsymbol{\tau})d\boldsymbol{\tau}}_{\text{const}}. \tag{2.11}$$

Thus, the optimal $\boldsymbol{\omega}_{i+1}$ minimizes $KL(p(R|\boldsymbol{\tau})p(\boldsymbol{\tau};\boldsymbol{\omega}_i))||p(\boldsymbol{\tau};\boldsymbol{\omega}_{i+1}))$.

For EM-based PS algorithms the resulting policy update algorithm will depend on what kind of lower-level control policy is applied to obtain the trajectory distribution $p(\boldsymbol{\tau};\boldsymbol{\omega}_i)$. For example, the Reward Weighted Regression algorithm [36, 56] uses a linear controller to learn, e.g., the inverse dynamics model of the robot [56]. Instead of using step-based exploration, the PoWER algorithm [36] applies a state dependent exploration technique, which often results in better policy updates. Promising results have been shown in underactuated swing-up and in the ball in a cup game [36]. The application of EM-based policy search method straightforwardly extends to episode-based algorithms [87].

[1] The KL-divergence is a non-negative asymmetric distance measure between distributions, which is defined as $KL(p(x)||q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx$.

For standard EM-based policy search methods, the new policy parameter is found by Moment-projection of the reward weighted old policy to the new policy. In practice, this results in performing a weighted maximum likelihood estimation of the new policy parameters, where the weights are typically defined as the exponentially weighted rewards $\exp(\beta R(\boldsymbol{\tau})$ with scalar $\beta$. Alternatively, we can perform the Information-projection of the reward weighted old policy [47], where the optimal solution is found by minimizing $KL(p(\boldsymbol{\tau};\boldsymbol{\omega}_{i+1})||p(\boldsymbol{\tau};\boldsymbol{\omega}_i)R(\boldsymbol{\tau}))$. By doing so, we avoid the typical problem of standard EM algorithms, that is, averaging over multiple modes of the reward weighted parameter distribution. However, when using Information-projection, the new policy parameters cannot be computed in closed form for most policies [47].

A significant advantage of EM-based PS methods compared to PG algorithms is that no user defined learning rate is required anymore and they can be applied to learn contextual upper-level policies (see 2.1.3 for more details).

**Information Theoretic Approaches.** While EM-based PS algorithms provide a principled approach to obtain the new policy parameters in closed form, due to policy update heuristics, the policy update might become overly aggressive or passive. While passive updates only lead to slower convergence, aggressive updates could result in trajectories that significantly differ from previously explored motions, which might cause premature convergence, unsafe situations and robot damage. To limit the experience loss between policy updates, natural gradient approaches [57, 58] enforce an upper bound of the KL-divergence between subsequent policies. However, natural PG algorithms only use an approximate KL-divergence and require a user defined learning rate for good performance.

To combine the smooth and safe learning properties when using natural gradients and the closed form policy update of EM-based PS algorithms, the information theoretic Relative Entropy Policy Search (REPS) algorithm has been proposed by Peters et al. [55]. The REPS algorithm limits the information loss between subsequent policies, while maximizing the expected reward, resulting in smooth convergence to the optimal solution. The episode-based REPS is a constrained optimization problem in the form of

$$\max_{\pi} \int \pi(\boldsymbol{\omega}) \mathcal{R}_{\boldsymbol{\omega}} d\boldsymbol{\omega}, \tag{2.12}$$

$$s.t. \quad \int \pi(\boldsymbol{\omega}) \log \frac{\pi(\boldsymbol{\omega})}{q(\boldsymbol{\omega})} d\boldsymbol{\omega} \leq \epsilon, \tag{2.13}$$

$$\int \pi(\boldsymbol{\omega}) d\boldsymbol{\omega} = 1, \tag{2.14}$$

where the first constraint represents the KL bound between the new $\pi(\boldsymbol{\omega})$, and the previously used upper-level policy $q(\boldsymbol{\omega})$. The upper bound $\epsilon \in \mathbb{R}^+$ is the only open parameter of REPS, which is usually easy to choose for a given learning problem. The second constraint ensures that the new policy $\pi(\boldsymbol{\omega})$ is a proper distribution.

The solution of the emerging constrained optimization problem exists in closed form

$$\pi(\boldsymbol{\omega}) \propto q(\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{\omega}}}{\eta}\right), \tag{2.15}$$

where $\eta$ is a Lagrange multiplier, which is found by optimizing the resulting convex dual function $g(\eta)$ [16]. The temperature parameter $\eta$ will scale the reward $\mathcal{R}_{\boldsymbol{\omega}}$, such that after the policy update the KL bound is satisfied. The significant advantage of REPS over other PS methods is that the temperature parameter is the result of the

constrained optimization problem and no extra heuristics is needed to compute it, as opposed to other methods [47, 87]. For more details of the derivation, we refer to [16].

### 2.1.2 Model-based Policy Search

After giving a compact overview of model-free PS methods, we turn our attention to model-based algorithms. While model-free PS algorithms provide unbiased policy updates by using the real robot for policy evaluation, the resulting approaches are inherently data inefficient. Even for simple learning tasks, model-free methods require hundreds if not thousands of policy evaluations before converging to a high quality solution. For many robot learning problems, such data inefficiency is impractical, as executing real robot experiments is time consuming, requires expert supervision and it result in robot wear, or even robot damage.

As opposed to model-free methods, model-based approaches exploit hand-tuned or learned models of the robot and its environment to predict the experiment outcome in computer simulation [3, 4, 6, 17, 18, 21, 32, 49, 68]. This approach can significantly increase the data-efficiency of the learning method, however, the success of the learning algorithm depends on the quality of the models. As hand-tuned models often fail to capture certain nonlinearities and stochasticity in the dynamics, data driven model learning is typically used to augment, or even replace the hand-tuned models. While rather good models can be learned in case of smooth dynamics, modeling abrupt changes and discrete events (e.g., contacts) is generally harder and it

requires more prior knowledge. Even if a good model is obtained using the measurement data, generalization for unseen situations remains a significant challenge. Overly confident prediction in the absence of data might lead to a biased policy [6]. For these reasons, model-based methods are not as widely used for robot skill learning as model-free approaches.

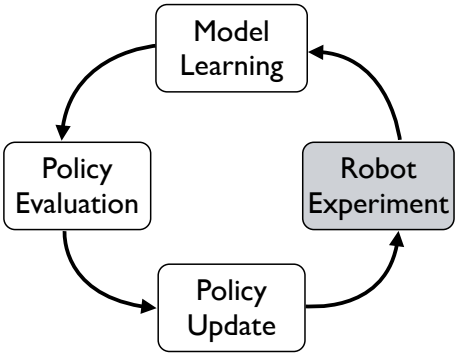



FIGURE 2.3: The learning loop of model-based policy search algorithms.

The general learning loop of model-based PS methods is shown in Fig. 2.3. The loop is similar to that of model-free PS (Fig. 2.1), however, the learning process is now augmented with model learning. To collect measurement data for learning a good model of the robot and its environment, we evaluate the policy on the real robot. The role of real robot experiment is solely to obtain new measurement data and to evaluate the learning progress. Additionally, we can introduce prior knowledge about the robot dynamics in the form of a mathematical model. In this case, learning should focus only on capturing the difference between our observations and the prior model prediction. In the next step, we use the learned model to predict the experiment outcome, *without* the use of the real hardware. Finally, we update the policy using

the simulated experiment outcome. In the following, we will give an overview of the most important model-based policy search methods. We will focus our discussion on modeling techniques, policy evaluation methods and policy updates.

**Modeling techniques.** While Abbeel et al. [3] suggested the use of a time dependent forward model, we typically learn the forward dynamics of the real hardware [6, 18, 21, 32, 49, 68]. Time-dependent models fail to generalize for unseen situations, and only provide accurate predictions along the observed trajectories [3]. However, they might be able to predict the system dynamics more accurately, especially if the state of the system includes unobserved variables.

For most model-based algorithms however, forward models are used to provide long term trajectory prediction. The most common approach is to learn the discrete-time stochastic state transition model $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}$ using measurement data, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ is i.i.d. Gaussian noise.

To learn such general nonlinear probabilistic models of the system dynamics, we can use the Locally Weighted Bayesian Regression (LWBR) algorithm [6, 49, 68]. LWBR learns local linear models $\boldsymbol{x}_{t+1} = [1, \boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T \boldsymbol{\beta} + \boldsymbol{\epsilon}$ of the state transition, where the parameter vector $\boldsymbol{\beta}$ is re-estimated locally for query point $\boldsymbol{y}_* = [\boldsymbol{x}_*^T, \boldsymbol{u}_*^T]^T$ using the observed data. Given a Gaussian distribution over $\boldsymbol{\beta}$, we can compute the successor state mean and variance in closed form. LWBR has been applied to learn the forward model of a helicopter [6, 49] as well as an inverted pendulum [68].

In recent years, non-parametric models have been proposed to learn the forward dynamics of robots [17, 18, 21, 32]. Instead of finding the optimal parameters of a parametric model given some observations, non-parametric models implicitly represent the model structure using the measurement data. In case the parametric model cannot capture the real dynamics perfectly, the model with the fitted parameters will be a more or less biased approximation of the real dynamics, which could lead to biased policies [6]. Non-parametric models circumvent this problem by avoiding the use of a specific parametric model and compute the predictive distribution solely using the observed data. Thus, prediction with non-parametric models becomes less biased.

One of the most successful non-parametric modeling technique for nonlinear stochastic dynamics is Gaussian Process (GP) regression [62]. GP models have proven to be efficient in learning the stochastic dynamics of a robot unicycle [17], a simple robot manipulator [18], a blimp [32], and a tendon driven robot arm [21]. With GP models [62], we can compute the predictive distribution of the successor state $\boldsymbol{x}_{t+1}$ in closed form given the query input $[\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T$ and the measurement data $\{\boldsymbol{x}_{i+1}, \boldsymbol{x}_i, \boldsymbol{u}_i\}_{i=1}^K$. As the posterior distribution is modeled as a Gaussian, the variance will reflect the confidence of the prediction. Consequently, we can avoid overly confident predictions in the absence of data in unexplored parts of the state space.

While probabilistic model learning offers a principled way to address model nonlinearity and prediction uncertainty in unexplored parts of the state space, prior knowledge of the dynamics in the form of a mathematical model can be exploited to improve the generalization accuracy. Once a crude mathematical model of the dynamics is available, probabilistic modeling can efficiently learn the difference between

the observations and the mathematical model prediction. This approach has proven to be efficient to learn the dynamics of a blimp using GP regression [32].

**Policy Evaluation.** As a robot skill trajectory is represented as a set of states and actions, for policy evaluation we need to execute long-term trajectory predictions using the stochastic dynamics models. The resulting trajectory distribution can then be used to compute the expected reward and the policy gradient along the trajectory. Long-term trajectory prediction using probabilistic models is computed either using the stochastic, or the deterministic approach [16].

When using the stochastic approach [6, 32, 49], we repeatedly sample the successor state $\boldsymbol{x}_{t+1} \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ from the predictive distribution at each time step. The implementation of stochastic trajectory prediction is straightforward and the computations are simple for most modeling techniques. Furthermore, control and state constraints are easy to incorporate into the prediction procedure, as the successor state can be considered deterministic after sampling from the predictive distribution. In Fig. 2.4 we show the prediction procedure for stochastic trajectory evaluation. For

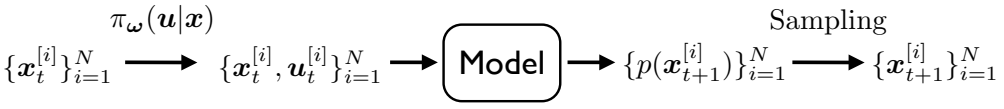



FIGURE 2.4: The prediction procedure using stochastic trajectory evaluation.

each trajectory sample, we first compute the control signal $\boldsymbol{u}_t \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x}_t)$. Using the probabilistic model, we compute the predictive distribution $p(\boldsymbol{x}_{t+1})$. Finally, we draw a sample from this distribution to obtain the successor state $\boldsymbol{x}_{t+1}$. To improve

the accuracy of the prediction, we typically sample $N$ trajectories for a single controller parametrization $\boldsymbol{\omega}$. Using the stochastic evaluation technique, the trajectory distribution will be unbiased in the limit.

When using the deterministic approach [17, 18, 60], we wish to compute the predictive distribution in closed form. For this purpose, we have to propagate the joint distribution over states and controls $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ through the model, as shown in Figure 2.5. In this case, we have to take several important challenges into consideration.

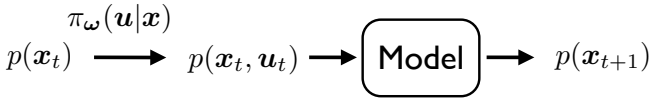



FIGURE 2.5: The prediction procedure using deterministic trajectory evaluation.

First, we need to obtain the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in closed form given the control policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x})$ and $p(\boldsymbol{x}_t)$. This requirement limits the class of applicable control policies, such that $p(\boldsymbol{x}, \boldsymbol{u})$ belongs to a parametric distribution, e.g., Gaussian. Furthermore, handling control constraints is not straightforward in this situation. The second important challenge when using deterministic trajectory prediction is that we need to propagate the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ through the model. However, most probabilistic modeling method can only provide the predictive distribution $p(\boldsymbol{x}_{t+1})$ given a deterministic input $[\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T$. Although in special cases we can compute the predictive distribution in closed form given $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$. For the general case however, we have to rely on approximations techniques.

One solution to the problem is to use model linearization to obtain an approximation of the predictive distribution, where we assume that both the input and output distributions are Gaussian. We visualize this approach in Fig. 2.6. While this method is

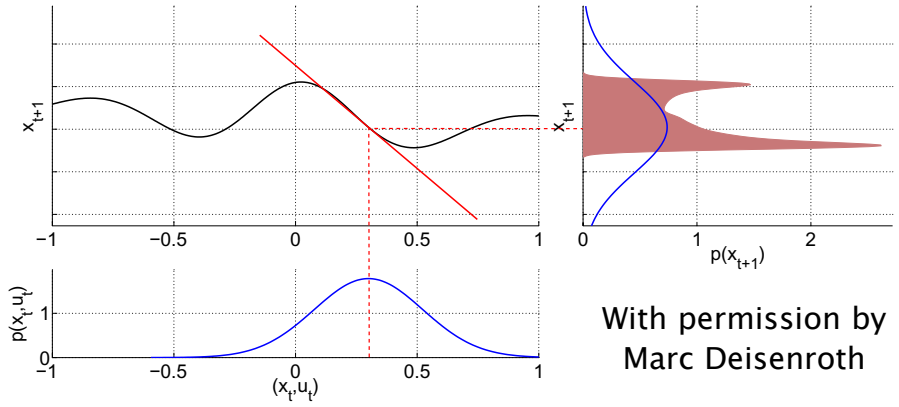



FIGURE 2.6: **Bottom left:** the illustration of the Gaussian joint distribution over states and controls. **Top left:** the dynamics model and the linearization at the mean of the query distribution. **Top right:** in red the real successor state distribution, while in blue the Gaussian approximation of the successor state distribution.

relatively easy to implement, the accuracy of the predictive distribution will depend on the nonlinearity of the dynamics in the region of the query distribution. In our example, the linearization gave a good approximation of the model only in a narrow region, and thus, we could only obtain a poor approximation of the real successor state distribution.

Another, more involved approximation technique is moment matching [17, 18, 60]. With moment matching, our aim is to obtain the first and second moment of the predictive distribution in closed form and approximate the distribution with a Gaussian. We illustrate this prediction technique in Fig. 2.7. Although the moment matching approach gives a more accurate prediction compared to linearization, computing the first and second moment are computationally involved.

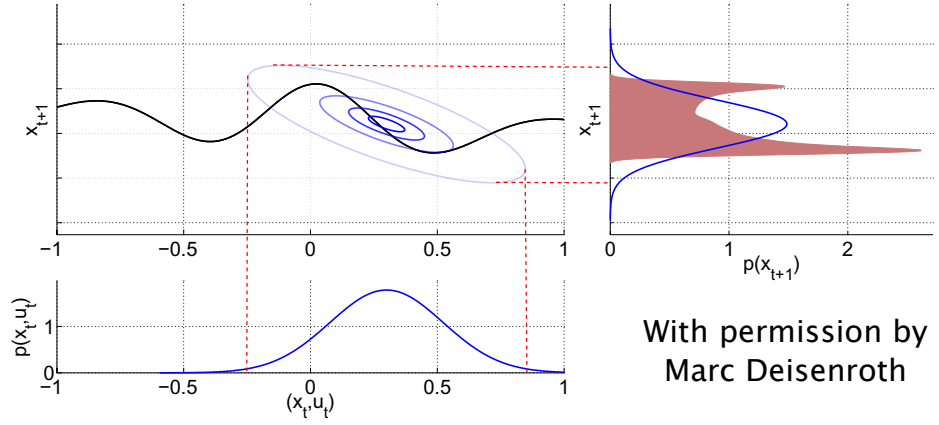



FIGURE 2.7: **Bottom left:** the illustration of the Gaussian joint distribution over states and controls. **Top left:** the dynamics model. **Top right:** in red the real successor state distribution, while in blue the Gaussian approximation of the successor state distribution.

While both stochastic and deterministic approaches are applicable to obtain the episode reward $\mathcal{R}_{\boldsymbol{\omega}}$ and the policy gradient along the trajectory, it is worth discussing their advantages and disadvantages when performing the predictions. When using the stochastic trajectory prediction approach, we avoid any approximation of the successor state distribution. Furthermore, the class of control policies is not restricted and control constraints can be satisfied. On the other hand, to obtain unbiased predictions, we need a significant amount of samples, which is computationally demanding. When approximating the policy gradient using the stochastic prediction approach, its variance might become overly large, resulting in inaccurate gradients. However, this problem can be circumvented when using the deterministic prediction method. Thus, in general, stochastic trajectory prediction is favorable for approximating the trajectory reward [6, 32, 49], while the deterministic prediction is preferred for policy gradient approaches [17, 18].

**Policy Updates.** The earliest model-based algorithms were the combination of model-free PS methods with learned forward models and stochastic policy evaluation. Promising results have been shown for learning autonomous helicopter flight [6, 49], flight controller of a blimp [32], and cart-pole balancing task [68].

The state of the art model-based policy search algorithm for learning lower-level controllers is the Probabilistic Inference for Learning Control (PILCO) algorithm [17], which uses GP models. PILCO first learns a GP model of the dynamics of the robot. Subsequently, it predicts the expected trajectory, its variance and the distribution of immediate rewards following the current control policy. For policy update, PILCO computes the policy gradient in closed form along the predicted trajectory using on the moment matching approach. However, this requires that the immediate reward is differentiable w.r.t. states and controls. The policy is updated using computer simulations until the optimal parametrization is found given the current model. The policy is then executed on the real robot to test the control performance and to collect new measurement data. The learning loop repeats until convergence to the optimal policy parameters is attained. However, as PILCO directly optimizes lower-level controller parameters, it cannot be straightforwardly applied to learn upper-level policies. Moreover, the class of representable lower-level controllers is restricted to functions through which a Gaussian distribution can be mapped in closed form. Nevertheless, PILCO has been successfully applied to learn the pendulum swing-up task [17] and box stacking with a simple robot manipulator [18] with unprecedented data efficiency.

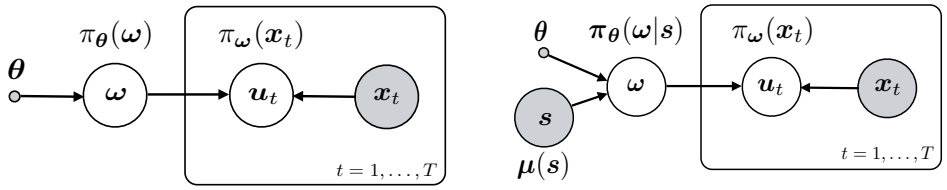
With appropriate assumptions on the control policy and the reward function, policy

gradient model-based algorithms, such as PILCO, are able to learn controllers with thousands of parameters by exploiting the gradient information. This is a significant advantage over EM-based [35, 47] and information theoretic approaches [14, 55], where a policy with only 50 parameters is difficult to learn.

### 2.1.3 Contextual Policy Search

Many robot tasks require the adaptation to a new environmental situation. For example, the robot ball throwing task (Fig. 1.3) requires the throwing stroke to be adapted to changing basket positions. Contextual policy search algorithms aim to learn an upper-level policy, which is able to generalize the robot skill for multiple situations. In the following, we denote task relevant context variables as $\boldsymbol{s}$. For the ball throwing task, the context might be defined as $\boldsymbol{s} = [p_x, p_y]$, where $p_x$ and $p_y$ are the $x$ and $y$ coordinates of the basket. We model the context as a random variable $\boldsymbol{s} \sim \boldsymbol{\mu}(\boldsymbol{s})$. During our experiments, we will assume that the context variable is observable, and that it is constant for a given episode. The context distribution $\boldsymbol{\mu}(\boldsymbol{s})$ is often known for a given task, but it also can be approximated using observations.

In contextual policy search [36, 47], our goal is to generalize the lower-level control policy, without having to learn an individual policy $\pi_{\boldsymbol{\theta}}^{\boldsymbol{s}}(\boldsymbol{\omega})$ for each context, which would be tedious and data inefficient. Instead, we follow a hierarchical approach, where we learn an upper-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}|\boldsymbol{s})$, which provides the lower-level controller parametrization $\boldsymbol{\omega}$ given the context $\boldsymbol{s}$. For comparison, we show the graphical model of context-free and contextual policy search in Fig. 2.8. In Fig. 2.8(a) we show




(a) (b)

FIGURE 2.8: **(a)** Episode-based and context-free policy search. The environmental setup, and thus the context $\boldsymbol{s}$ is fixed. **(b)** The contextual policy search setup. The context changes in the beginning of each episode according to $\boldsymbol{\mu}(\boldsymbol{s})$. The upper-level policy conditions on the context to provide the lower-level policy parametrization $\boldsymbol{\omega}$.

the episode-based context-free policy search setup. The context is fixed in this situation. However, in contextual policy search (Fig. 2.8(b)), the context changes between experiments according to distribution $\boldsymbol{\mu}(\boldsymbol{s})$. Thus, we use a conditional upper-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}|\boldsymbol{s})$ to choose the parametrization for the lower-level control policy. We can consider context-free policy search methods as a special case of contextual PS algorithms, where the context is a fixed deterministic variable.

In contextual PS, our goal is to find the optimal policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})^*$, such that it maximizes the expected reward

$$\pi(\boldsymbol{\omega}|\boldsymbol{s})^* = \arg\max_{\pi} \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\omega}} \pi(\boldsymbol{\omega}|\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} d\boldsymbol{\omega} d\boldsymbol{s}, \tag{2.16}$$

where the context distribution $\mu(\boldsymbol{s})$ is defined by the learning problem and $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}$ denotes the expected reward when executing the lower-level policy with parameter $\boldsymbol{\omega}$ in context $\boldsymbol{s}$. As the dynamics of the robot and the environment are stochastic, the

reward $\mathcal{R}_{s\omega}$ is given by the expected reward over all trajectories

$$\mathcal{R}_{s\omega} = \mathbb{E}_{\tau}[R(\tau, s)|s, \omega] = \int_{\tau} p(\tau|s, \omega) R(\tau, s) d\tau. \tag{2.17}$$

The trajectory distribution in context $s$ using the lower-level policy parameters $\omega$ is denoted by $p(\tau|s, \omega)$. The function $R(\tau, s)$ specifies the reward for executing trajectory $\tau$ in context $s$. Typically, $R(\tau, s)$ is defined as the sum of immediate rewards, but the above formulation also allows an arbitrary function of the trajectory and the context. Thus, we can incorporate terms in the reward function that cannot be computed using state-control pairs $[x_t^T, u_t^T]^T$, but with the complete trajectory $\tau$, e.g., minimum distances. In the following, we give an overview of the relatively few contextual policy search algorithms that solve the optimization problem in Eq. (2.16).

Reward Weighted Regression (RWR) approximates the upper-level policy $\pi_\theta(\omega|s)$ with a Gaussian $\mathcal{N}(\omega|A\phi(s), \Sigma_\omega)$ [56]. The mean of the Gaussian is linear in the context feature $\phi(x)$, and the weight parameters are found by performing a weighed least squares estimation. The weights are given as exponentially weighted rewards $w^{[i]} = \exp(\beta \mathcal{R}_{s\omega}{}^{[i]}), i = 1, \ldots, N$, where $N$ is the number of sample evaluations and $\beta$ is a weighting parameter. The upper-level policy parameter $A$ is then found by solving the weighted maximum likelihood problem $\max_A \sum_{i=1}^N w^{[i]} \log \pi_A(\omega^{[i]}|s^{[i]})$,

$$A = (\Phi(s)^T W \Phi(s))^{-1} \Phi(s)^T W \Omega, \tag{2.18}$$

where $W = \mathrm{diag}(w)$, $\Phi(s) = \left[\phi(s^{[1]}), \ldots, \phi(s^{[N]})\right]$ is the feature matrix and the matrix of policy parameters is $\Omega = \left[\omega^{[1]}, \ldots, \omega^{[N]}\right]$. The covariance matrix of the Gaussian

can easily be computed by weighted maximum likelihood estimation using the policy parameter $\boldsymbol{A}$, the features $\boldsymbol{\Phi}(\boldsymbol{s})$ and the lower-level policy parameters $\boldsymbol{\Omega}$ [16, 56]. However, RWR suffers from the heuristics involved when choosing the weighting parameter $\beta$.

The Cost Regularized Kernel Regression (CrKR) algorithm [35] is the kernelized version of RWR. CrKR is a non-parametric approach, where the policy parameters are implicitly represented using the parameter-context-reward samples. The policy is approximated with a Gaussian distribution $\mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}(\boldsymbol{s}_*), \text{diag}(\boldsymbol{\sigma}^2(\boldsymbol{s}_*)))$ for query context $\boldsymbol{s}_*$, where the mean and variance are given by

$$\boldsymbol{\mu}(\boldsymbol{s}_*) = \boldsymbol{k}(\boldsymbol{s}_*)(\boldsymbol{K}+\lambda\boldsymbol{C})^{-1}\boldsymbol{\Omega}^T, \tag{2.19}$$

$$\boldsymbol{\sigma}^2(\boldsymbol{s}_*) = k(\boldsymbol{s}_*,\boldsymbol{s}_*)+\lambda-\boldsymbol{k}(\boldsymbol{s}_*)^T(\boldsymbol{K}+\lambda\boldsymbol{C})^{-1}\boldsymbol{k}(\boldsymbol{s}_*), \tag{2.20}$$

with $\boldsymbol{C} = \boldsymbol{W}^{-1}$ as the cost matrix. While we are still working with features, we do not actually have to define $\boldsymbol{\phi}(\boldsymbol{s})$, but a kernel function $k(\boldsymbol{s},\boldsymbol{s}') = \boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{\phi}(\boldsymbol{s}')$. The resulting kernel matrix is defined as $\boldsymbol{K} = \boldsymbol{\Phi}(\boldsymbol{s})^T\boldsymbol{\Phi}(\boldsymbol{x})$ and $\boldsymbol{k}(\boldsymbol{s}_*) = \boldsymbol{\Phi}(\boldsymbol{s})^T\boldsymbol{\phi}(\boldsymbol{s}_*)$. The resulting regression problem closely relates to Gaussian Processes [62]. The CrKR algorithm has been successfully applied to learn robot table tennis and darts [35]. The disadvantage of using CrKR for representing the policy is that the policy parameters become uncorrelated with the diagonal covariance matrix.

Daniel et. al [14] proposed a hierarchical learning architecture to learn multiple skills for a single problem. The algorithm first observers the context and uses a gating policy $\pi(o|\boldsymbol{s})$ to choose an option for executing the skill. Such an option might be a

back- or forehand stroke in the table tennis game. The gating policy activates one of the upper-level policies $\pi(\boldsymbol{\omega}|\boldsymbol{s}, o)$, which provide the parametrization to the control policy. Successful application for the robot tether ball game is presented in [14].

Lastly, the Variational Inference for Policy Search (VIP) algorithm is an EM-based policy search method [47]. However, VIP uses the Information Projection of the reward weighted policy and it can be extended by using contextual policies. Successful application for a robot balancing task is discussed in [47].

## 2.2 Robot Skill Representations

When using policy search for robot skill learning, we benefit from the option of choosing a task-appropriate representation of the skill in the form of a parametrized policy. Furthermore, we can exploit prior knowledge about the task to initialize the policy parameters using expert demonstration. Subsequently, the policy parameters are optimized using PS methods to obtain high quality skills. By using a parametrized policy, PS algorithms can be straightforwardly applied for robot learning tasks with high dimensional continuous state-action spaces.

A lower-level policy can be represented as an open-loop controller that directly encodes controls over time, such as torque or joint acceleration. In this case, the policy parameters encode the control for each joint at each time step. To reduce the number of parameters, we might store the data only at specific time points and use interpolation techniques to obtain the missing information. One of the most commonly used time indexed via-point skill representation is cubic splines. Splines have been

applied to learn energy efficient and stable impact recovery policies for a wheeled robot [40]. Despite the compact trajectory representation of cubic splines, the time dependency limits the generalization efficiency of the skill and the open loop control could lead to poor result in stochastic environments.

Alternatively, the skill can be represented as a state dependent policy, which directly maps states to controls. As the mapping is a general regression problem, we can apply many well understood techniques for model learning and prediction. For example, neural networks has been applied for online learning of the walking gait for biped locomotion [25]. We can alternatively learn local controllers and use radial basis functions for weighting the local controls to produce the desired torque or acceleration. Such local policies were applied for learning the pendulum swing-up task [17] and for learning block stacking with a simple robot manipulator [18]. While these policies encode general, time independent skills, the amount of policy parameters might become overly large, which could be difficult to learn with PS algorithms.

In recent years, movement primitives became one of the most successful class of policies for skill learning [31, 34, 53, 67], due to their compact representation of high dimensional robot movement skills. Movement primitives encode discrete, or periodic movements, such as a throwing stroke or a walking gait. By combining primitives sequentially or simultaneously, movement primitives can be regarded as building blocks for constructing more complex skills. Primitives typically encode the reference angular position and velocity for each joint of the robot, which is then tracked using, e.g., by a feedback or an inverse dynamics controller. Dynamic Movement Primitives (DMPs) generate the trajectory using a second order dynamical system,

where the trajectory shape can be modified using policy parameters [31, 34]. By modifying the parameters of the dynamical system, the movement can be scaled both spatially and temporarily. A probabilistic representation of movement primitives allows for blending, simultaneously activating skills and computing the optimal tracking controller parameters as well [53]. During our experiments we will often use DMPs to represent the robot skill, and a detailed description of DMPs is given in the following section.

### 2.2.1 Dynamic Movement Primitives

In the following, we will only consider learning stroke based skills. Thus our review will only focus on representing discrete movements. For representing periodic skills, we refer to [31] and [34]. A dynamic movement primitive [31] is defined as a second order dynamical system that acts like a spring-damper system which is activated by a non-linear forcing function $f$. The system is described by

$$\ddot{x}_t = \tau^2 \alpha_x(\beta_x(g - x_t) - \dot{x}_t) + \tau^2 f(z_t; \boldsymbol{v}), \tag{2.21}$$

$$\dot{z}_t = -\tau \alpha_z z_t. \tag{2.22}$$

with constant parameters $\alpha_x, \beta_x$, $\alpha_z$ and time scaling parameter $\tau$. Typically, a separate DMP is used for each joint of the robot. The phase variable $z_t$ acts as internal clock of the movement. It is shared between all joints to synchronize their movement. It is simulated by a separate first order dynamical system and is initialized as

$z_0 = 1$. It converges to 0 as $t \to \infty$ and it influences the shape of the trajectory by driving the non-linear forcing function $f$. The parameter $g$ is the unique point attractor of the system, which can be set by the user or initialized using demonstration. The spring-damper system is modulated by the function $f(z_t; \boldsymbol{v}) = \boldsymbol{\phi}(z_t)^T \boldsymbol{v}$, which is linear in its weights $\boldsymbol{v}$, but non-linear in the phase $z_t$. The weights $\boldsymbol{v}$ specify the shape of the movement and can be initialized with expert demonstration as well. The basis functions $\boldsymbol{\phi}_i(z_t), i = 1, \ldots, K$ activate the weights as the trajectory evolves. The basis functions are defined as

$$\boldsymbol{\phi}_i(z_t) = \frac{\exp(-(z_t - c_i)^2/(2\sigma_i^2))z_t}{\sum_{j=1}^{K} \exp(-(z_t - c_j)^2/(2\sigma_j^2))},$$

where $c_i$ is the basis center and $\sigma_i$ it the bandwidth parameter. The squared exponential basis functions are multiplied by $z_t$ such that $f$ vanishes for $t \to \infty$. Hence, for $t \to \infty$, the DMP will behave as linear, stable system with point attractor $g$. The speed of the trajectory execution can be regulated by the time scaling factor $\tau \in \mathbb{R}^+$. The weight parameters $\boldsymbol{v}$ of the DMP can be initialized from observed trajectories $\{\boldsymbol{x}_{obs}, \dot{\boldsymbol{x}}_{obs}, \ddot{\boldsymbol{x}}_{obs}\}$ by solving

$$\boldsymbol{v} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{Y}, \quad \boldsymbol{Y} = \frac{1}{\tau^2} \ddot{\boldsymbol{x}}_{obs} - \alpha_x(\beta_x(g - \boldsymbol{x}_{obs}) - \dot{\boldsymbol{x}}_{obs}), \tag{2.23}$$

where $\boldsymbol{\Phi}_{t,\cdot} = \boldsymbol{\phi}^T(z_t)$ is the matrix of basis vectors at time step $t$ and $g$ is the final element of $\boldsymbol{x}_{obs}$. In a robot skill learning task, we can adapt the weight parameters $\boldsymbol{v}$, the goal attractor $g$ and the time scaling factor $\tau$ to optimize the trajectory. Additionally, we can also adapt the final desired velocity $\dot{g}$ of the movement with the extension

given in [34].

To reduce the dimensionality of the learning problem, we usually learn only a subset of the DMP parameters. For example, when learning to return balls in table tennis, we initialize and fix the weights $\boldsymbol{v}$ from expert demonstration. Subsequently, we adapt the goal attractor $g$ and the final velocity $\dot{g}$ of the DMPs to improve the hitting stroke. After obtaining a desired trajectory encoded by the DMP, the trajectory is followed by a feedback or an inverse dynamics controller. In the presented tasks, the motor primitive is always executed for a predefined amount of time. For a more detailed description of the DMP framework we refer to [34].

**Example.** We give an example of initializing the DMP parameters for a 1 DoF robot arm from demonstration. Assume that we have observed a demonstration of the required joint trajectories (Fig. 2.9). Subsequently, we train the weight parameters

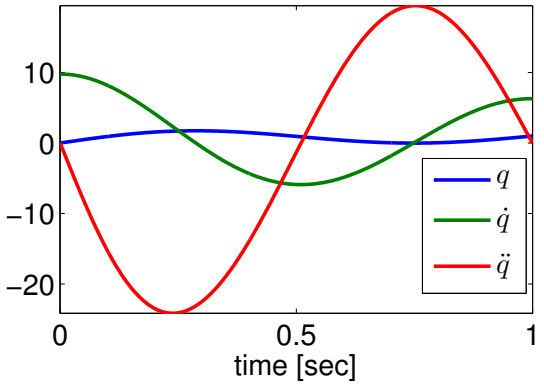
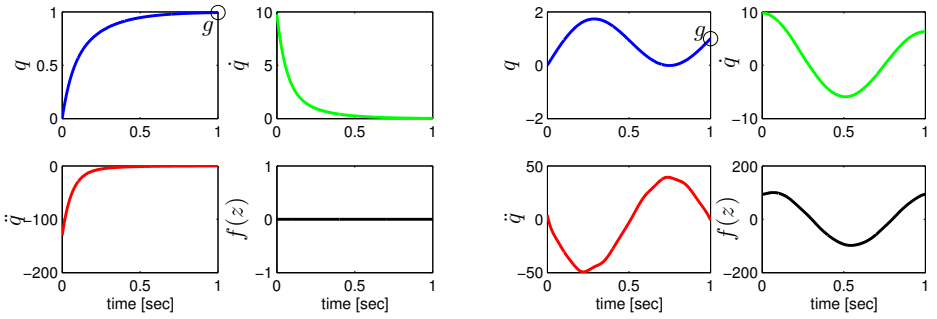



FIGURE 2.9: The demonstrated angular position $q$, velocity $\dot{q}$ and acceleration $\ddot{q}$ for a 1 DoF robot arm.

$\boldsymbol{v}$ using Eq. (2.23) for fixed basis function parameters $\{c_i, \sigma_i\}_{i=1}^{10}$. In Fig. 2.10(a) we show the DMP trajectories *without* the forcing function, that is $\boldsymbol{v} = \mathbf{0}$. In Fig. 2.10(b) we show the DMP generated trajectories using the trained weight parameters $\boldsymbol{v}$. As we can clearly see, the forcing function changes the shape of the trajectory, but the

final angular position $g$ remains unaltered. From Fig. 2.11 we can see that the DMP




(a) (b)

FIGURE 2.10: **(a)** The trajectories generated by the DMP without the forcing function ($\boldsymbol{\nu} = \mathbf{0}$). **(b)** The DMP trajectories generated by the trained forcing function. The robot arm reaches the same angular position $g$, but the shape of the trajectory is altered so that it matches the demonstration.

accurately captured the demonstrated trajectories using only 10 parameters.

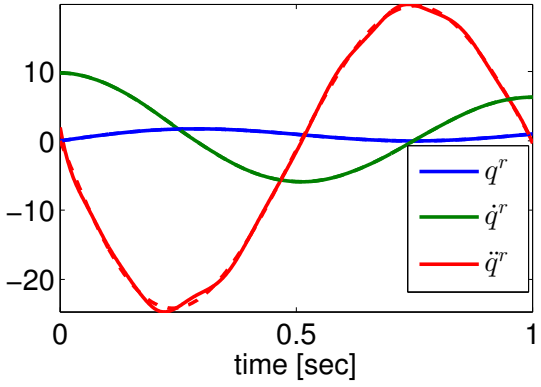



FIGURE 2.11: The demonstrated (**dashed**) and the DMP encoded trajectories (**solid**). The DMP accurately captured the demonstration using only 10 parameters.

## 2.3 Gaussian Process Regression

Gaussian Processes are efficient non-parametric Bayesian regression tools [62] that explicitly represent model uncertainties. GP models can be applied for general regression problems, such as modeling the dynamics of a robot [17], time series analysis [62], or signal processing [15, 19]. GP models are non-parametric, the model prediction is explicitly represented with the observed training data and a properly chosen kernel function, without the need of specifying the model class. The prediction is expressed in terms of a Gaussian distribution, and thus, GP models provide a confidence measure of the prediction. However, the computational demand for model training and prediction might be overly large. In the following, we give a brief overview of Gaussian Process Regression. For further details we refer to [62].

For a general regression problem, given the training data $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$, $\boldsymbol{x} \in \mathbb{R}^d$, $y \in \mathbb{R}$, we wish to approximate the target values $y$ as a function of the input $\boldsymbol{x}$, $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is zero mean i.i.d. Gaussian noise. With GPR our goal is to predict the distribution over $y_*$ for test input $\boldsymbol{x}_*$ in a conditional Gaussian model $p(y_*|\boldsymbol{x}_*, \mathcal{D})$. The joint prior over training and test targets can be expressed as

$$\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \boldsymbol{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I} & \boldsymbol{k} \\ \boldsymbol{k}^T & k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma_\epsilon^2 \end{bmatrix} \right), \tag{2.24}$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{K}$ is the $N \times N$ kernel matrix with elements $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{k}$ denotes the kernel vector for the query input with $\boldsymbol{k}_i = k(\boldsymbol{x}_i, \boldsymbol{x}_*)$, $i =$

$1,\ldots,N$. The covariance or kernel function $k(\cdot,\cdot)$ defines a similarity measure between the input data. A common choice for kernel function is the squared exponential

$$k(\boldsymbol{x},\boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{x}')^T \boldsymbol{W}^{-1}(\boldsymbol{x}-\boldsymbol{x}')}{2}\right), \tag{2.25}$$

which expresses the closeness of $\boldsymbol{x}$ an $\boldsymbol{x}'$ according to the diagonal weighting matrix $\boldsymbol{W} = \text{diag}(\boldsymbol{w}^2)$. The parameter $\sigma_f^2$ represents the variance of the function. We refer to parameters $\boldsymbol{\theta} = \{\boldsymbol{w}, \sigma_f, \sigma_\epsilon\}$ as hyper-parameters of the GP model. For the prior in Eq. (2.24), we used zero mean, but knowledge about the regression problem might suggest more informative prior mean functions.

For a new test input $\boldsymbol{x}_*$, the predictive distribution $p(y_*|\boldsymbol{x}_*, \mathcal{D})$ of the posterior Gaussian process is a Gaussian $\mathcal{N}(y_*|\mu_*, \sigma_*^2)$ with mean and variance

$$\mu_* = \boldsymbol{k}^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{y}, \tag{2.26}$$

$$\sigma_*^2 = \sigma_\epsilon^2 + k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{k}. \tag{2.27}$$

We show an illustration of GP regression in Fig. 2.12. For simplicity, we use one dimensional inputs and targets. The shaded area represents the 95% confidence interval of the prediction. In the figure, we also depict samples from the target function $p(f_*|\boldsymbol{x}_*, \mathcal{D})$ against $\boldsymbol{x}_*$. As we can see, without observing any samples (represented with **black** dots), the prediction variance is high. However, after observing an increasing amount of samples, the variance rapidly reduces and the sample functions converge to the real underlying function.

Although GP models are non-parametric, we still have to solve the model selection

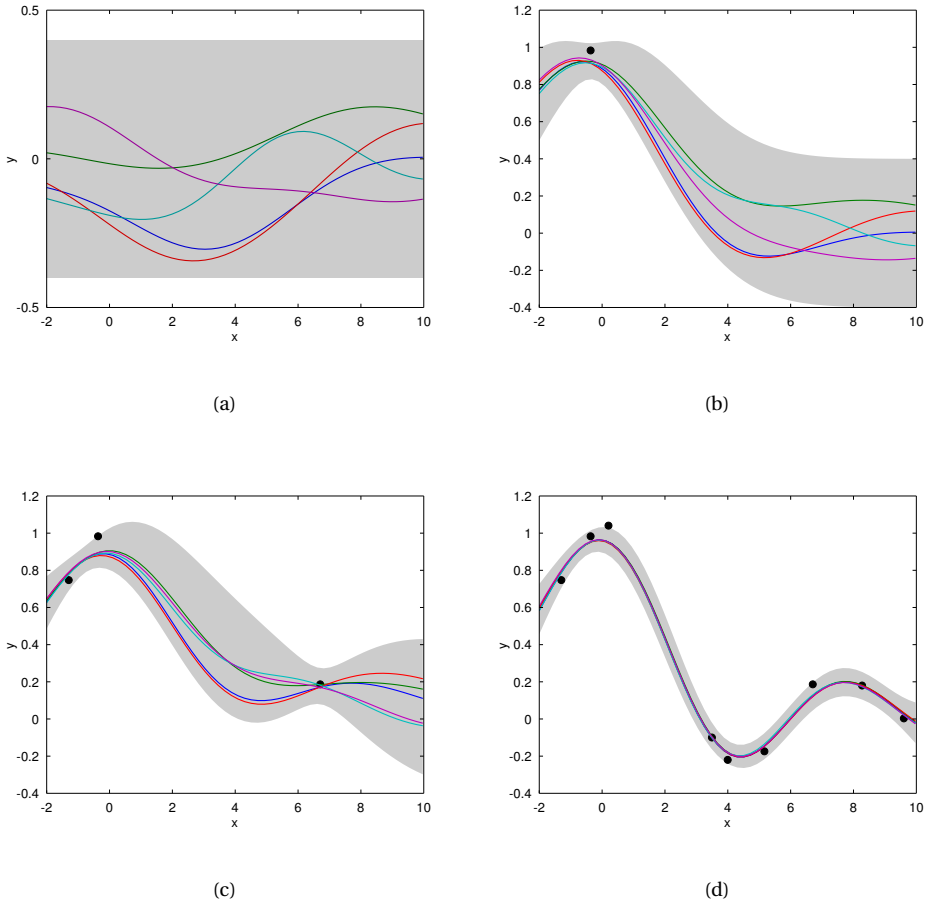



FIGURE 2.12: Prediction with Gaussian Process models for an increasing number of samples (**black** dots). The shaded area represents the 95% prediction confidence interval. The curves represent samples from $p(f_*|x_*, \mathcal{D})$.

problem. This involves finding the kernel function and its hyper-parameters that best describes the training data. While the regression problem usually gives a good intuition for choosing the kernel function, finding the optimal hyper-parameters is a more difficult problem. In order to find the optimal hyper-parameters, we often maximize the marginal log-likelihood [62]

$$\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T(\boldsymbol{K}+\sigma_\epsilon^2\boldsymbol{I})^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}+\sigma_\epsilon^2\boldsymbol{I}| - \frac{n}{2}\log 2\pi, \tag{2.28}$$

where $\boldsymbol{\theta}$ represents the hyper-parameters. Other optimization methods, such as cross validation is also applicable [62].

One important disadvantage of kernel-based methods is the high computational demand. When executing a prediction using Eqs. (2.26-2.27), we first compute the kernel vector $\boldsymbol{k}$, then an inner product of two vectors for the predictive mean, which has $\mathcal{O}(N)$ computational complexity given $(\boldsymbol{K}+\sigma_\epsilon \boldsymbol{I})^{-1}$, where $N$ is the number of observed samples. For the predictive variance, the computational complexity is $\mathcal{O}(N^2)$. However, during training the hyper-parameters, we repeatedly have to invert $(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})$ with the latest hyper-parameters, which has $\mathcal{O}(N^3)$ computational complexity. While we can use Cholesky decomposition to reduce the computation time of the inverse [62, 70], training and prediction still remains expensive. This is a significant problem when learning robot dynamics, as the speed of the control loop is typically around $500 - 1000$Hz, and thus, computations become impractically slow after only a few seconds of recorded data. While undersampling could reduce the data size, the resulting regression problem might be more difficult to solve.

An alternative way to reduce the computational complexity is the use of sparse Gaussian Process models [11, 59, 72, 85]. Sparse GPs maintain only a low number $M \ll N$ of highly representative samples. The resulting computational complexity for mean prediction typically becomes $\mathcal{O}(M)$, for variance prediction it is $\mathcal{O}(M^2)$, and for model training it is typically $\mathcal{O}(NM^2)$. However, sparse GP models tend to over- or underfit the data compared to the standard GP approach, which could lead to an overly confident or timid prediction. Recently, local GP models have been proposed [50, 51, 52] to decompose the global regression problem into local ones. The local regression

models use only a low amount of samples, and thus, the prediction can be performed relatively fast. Promising results have been shown for learning the inverse dynamics model of the robot online [52].

## 2.4 Kernel Embedding of Conditional Distributions

In recent years, kernel methods became popular in the machine learning community [69]. The key idea of kernel methods is to execute computations in possibly infinite dimensional feature spaces where the underlying problem is easier to solve. However, due to the virtue of the "kernel-trick", we can avoid the explicit definition of the feature function. Instead, we only need to specify the inner product of features in the form of a kernel function. Thus, computations become tractable and they are typically easy to implement in a data-driven fashion. Successful applications are, for example, Support Vector Machines [10] for classification, Gaussian Processes [62] for both classification and regression and many more [69].

In recent years, there has been an increasing focus on applying kernel methods for probabilistic inference applications [74] in graphical models [38]. The key idea is to embed conditional distributions into feature spaces, where the manipulation of distributions can be performed by linear algebraic operations. The kernel embedding approach has many advantages compared to direct manipulation of probability distributions. First, probabilistic operations, such as sum rule, chain rule and Bayes' rule become simple linear algebraic operations in feature spaces for arbitrary distributions [23]. Thus, we can avoid solving complex integrals and using approximations

for the predictive distribution. Second, as kernel embedding is a non-parametric method, we can avoid specifying the class of a parametric distribution. Thus, we can capture rich statistical features of the distribution using the observed data. Finally, by avoiding the explicit use of parametric distributions, kernel embedding is suitable for working with discrete variables, strings, etc., as long as a positive definite kernel is defined [74]. Successful applications of the kernel embedding approach are, for example, belief propagation [74, 75] and filtering with Hidden Markov Models [73, 76]. In the following, we give a brief overview on the kernel embedding of conditional distributions. For more details we refer to the review paper of Song et al. [74].

We begin by defining the positive definite kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$ for arbitrary points $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$. We define the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{F}_{\mathcal{X}}$ on $\mathcal{X}$ with kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ as a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ with inner product $\langle \cdot, \cdot \rangle$. Each element $\boldsymbol{k}(\boldsymbol{x}, \cdot)$ of $\mathcal{F}$ satisfies the reproducing property

$$f(\boldsymbol{x}) = \left\langle \sum_i \alpha_i \boldsymbol{k}(\boldsymbol{x}_i, \cdot), \boldsymbol{k}(\boldsymbol{x}, \cdot) \right\rangle \tag{2.29}$$

$$= \sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}), \tag{2.30}$$

where the function is $f(\cdot) = \sum_i \alpha_i \boldsymbol{k}(\boldsymbol{x}_i, \cdot)$ and the $\alpha_i$ values are constants. Thus, evaluating function $f(\cdot)$ at point $\boldsymbol{x}$ can be represented as an inner product. The kernel function is often referred to as a feature $\boldsymbol{k}(\boldsymbol{x}, \cdot) = \boldsymbol{\phi}(\boldsymbol{x})$. However, for function evaluation, we only have to compute $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$ without the explicit representation of the possibly infinite dimensional feature $\boldsymbol{\phi}(\boldsymbol{x})$. Typical kernel functions are the squared exponential and the polynomial kernels.

The kernel embedding of a distribution is represented as the expected feature map, which is a point in the RKHS

$$\boldsymbol{\mu}_{\boldsymbol{x}} = \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{\phi}(\boldsymbol{x})] = \int_{\mathcal{X}} p(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x}. \tag{2.31}$$

The embedding can be generalized for joint distributions over an arbitrary amount of random variables. The joint distribution can be embedded to a tensor product feature space. For example, for joint distribution $p(\boldsymbol{x},\boldsymbol{y})$, $\boldsymbol{x}\in\mathcal{X}$, $\boldsymbol{y}\in\mathcal{X}$ the tensor product feature space is $\mathcal{F}\otimes\mathcal{F}$. Thus, the embedding of $p(\boldsymbol{x},\boldsymbol{y})$ can be expressed as

$$\mathcal{C}_{\boldsymbol{xy}} = \mathbb{E}_{\boldsymbol{xy}}[\boldsymbol{\phi}(\boldsymbol{x})\otimes\boldsymbol{\phi}(\boldsymbol{y})] = \iint_{\mathcal{X}\times\mathcal{X}} p(\boldsymbol{x},\boldsymbol{y})\boldsymbol{\phi}(\boldsymbol{x})\otimes\boldsymbol{\phi}(\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y}. \tag{2.32}$$

For most practical examples, we only have access to samples from distribution $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$, $\{\boldsymbol{x}_i,\boldsymbol{y}_i\}_{i=1}^N$. In this case we can use an empirical estimation for the mean embedding and the covariance operator

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_i), \tag{2.33}$$

$$\hat{\mathcal{C}}_{\boldsymbol{xy}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_i)\otimes\boldsymbol{\phi}(\boldsymbol{y}_i). \tag{2.34}$$

We will see later that computation with the empirical estimates of the mean embedding and covariance operator will lead to linear algebraic operations with Gram matrices.

We now turn our attention to the manipulation of distributions in feature spaces. First, we discuss the embedding of conditional distribution $p(\boldsymbol{y}|\boldsymbol{x}_*)$, where $\boldsymbol{x}_*$ is the

deterministic query point. The mean embedding of the conditional is defined as

$$\boldsymbol{\mu}_{\boldsymbol{y}|\boldsymbol{x}_*} = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}_*}[\boldsymbol{\phi}(\boldsymbol{y})] = \int_{\mathcal{X}} p(\boldsymbol{y}|\boldsymbol{x}_*)\boldsymbol{\phi}(\boldsymbol{y})d\boldsymbol{y}. \tag{2.35}$$

The conditional embedding can also be defined using the conditional covariance operator

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{y}|\boldsymbol{x}_*} &= \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\boldsymbol{\phi}(\boldsymbol{x}_*), & (2.36)\\ &= \mathcal{C}_{\boldsymbol{y}\boldsymbol{x}}\mathcal{C}_{\boldsymbol{x}\boldsymbol{x}}^{-1}\boldsymbol{\phi}(\boldsymbol{x}_*), & (2.37) \end{aligned}$$

where we use the definition $\mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}} = \mathcal{C}_{\boldsymbol{y}\boldsymbol{x}}\mathcal{C}_{\boldsymbol{x}\boldsymbol{x}}^{-1}$ [74]. The empirical estimation of the covariance operator $\mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}$ when observing $N$ samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ is

$$\hat{\mathcal{C}}_{\boldsymbol{y}|\boldsymbol{x}} = \boldsymbol{\Phi}_{\boldsymbol{y}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}_{\boldsymbol{x}}^T, \tag{2.38}$$

where $\lambda \in \mathbb{R}^+$ is a small constant to avoid numerical problems during matrix inversion, the feature matrices of observations $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ are $\boldsymbol{\Phi}_{\boldsymbol{x}} = [\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N)]$ and $\boldsymbol{\Phi}_{\boldsymbol{y}} = [\boldsymbol{\phi}(\boldsymbol{y}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{y}_N)]$, and the kernel matrix is $\boldsymbol{K}_{ij} = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j)\rangle = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The empirical estimation of the conditional mean embedding given $\hat{\mathcal{C}}_{\boldsymbol{y}|\boldsymbol{x}}$ and query $\boldsymbol{x}_*$ is

$$\begin{aligned} \hat{\boldsymbol{\mu}}_{\boldsymbol{y}|\boldsymbol{x}_*} &= \boldsymbol{\Phi}_{\boldsymbol{y}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\overbrace{\boldsymbol{\Phi}_{\boldsymbol{x}}^T\boldsymbol{\phi}(\boldsymbol{x}_*)}^{\boldsymbol{k}(\boldsymbol{x}_*)}, & (2.39)\\ &= \boldsymbol{\Phi}_{\boldsymbol{y}}\boldsymbol{\beta}, & (2.40) \end{aligned}$$

with $\boldsymbol{\beta} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{k}(\boldsymbol{x}_*)$ and $\boldsymbol{k}_i(\boldsymbol{x}_*) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_*)\rangle = k(\boldsymbol{x}_i, \boldsymbol{x}_*)$. However, in many cases the query might be represented as a distribution over $\boldsymbol{x}$, e.g., $q(\boldsymbol{x})$, instead of a

deterministic value $\boldsymbol{x}_*$. In this case, we have to marginalize over $\boldsymbol{x}$ to obtain $p(\boldsymbol{y})$ and the embedding $\boldsymbol{\mu}_{\boldsymbol{y}}^q$, where we emphasize that $\boldsymbol{x}$ comes from distribution $q(\boldsymbol{x})$. Using the covariance operator $\mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}$ we can write up the mean embedding $\boldsymbol{\mu}_{\boldsymbol{y}}^q$ as

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{y}}^q &= \mathbb{E}_{\boldsymbol{x}}\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\boldsymbol{\phi}(\boldsymbol{y})] \\ &= \mathbb{E}_{\boldsymbol{x}}[\mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\boldsymbol{\phi}(\boldsymbol{x})] \\ &= \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x})] \\ &= \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\boldsymbol{\mu}_{\boldsymbol{x}}^q. \end{aligned} \tag{2.41}$$

In practice, we assume that the mean embedding of the prior is given by $\boldsymbol{\mu}_{\boldsymbol{x}}^q = \boldsymbol{\Phi}_{\tilde{\boldsymbol{x}}}\boldsymbol{\alpha}$ with samples $\{\tilde{\boldsymbol{x}}_i\}_{i=1}^M$, $\tilde{\boldsymbol{x}} \in \mathcal{X}$, and thus, the sample based approximation of the marginal embedding is

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{y}}^q = \boldsymbol{\Phi}_{\boldsymbol{y}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\tilde{\boldsymbol{K}}\boldsymbol{\alpha}, \tag{2.42}$$

where $\tilde{\boldsymbol{K}} = \boldsymbol{\Phi}_{\boldsymbol{x}}^T\boldsymbol{\Phi}_{\tilde{\boldsymbol{x}}}$.

In some cases we wish to compute the expected value of a function $f \in \mathcal{F}_{\mathcal{X}}$ given the embedding of the prior distribution $\boldsymbol{\mu}_{\boldsymbol{x}}^q = \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\beta}$. Assume that the function is given by $f = \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\alpha}$. To compute the expected value, we use the reproducing property

$$f(\boldsymbol{x}) = \langle f, \boldsymbol{\mu}_{\boldsymbol{x}}^q \rangle = \boldsymbol{\beta}^T\boldsymbol{K}\boldsymbol{\alpha}. \tag{2.43}$$

In other cases, we wish to decode the most likely value $\boldsymbol{x}_*$ for embedding $\boldsymbol{\mu}_{\boldsymbol{x}} = \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\beta}$. We can find this value by solving

$$\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}} ||\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\beta}||_2 \tag{2.44}$$

$$= \arg\min_{\boldsymbol{x}} k(\boldsymbol{x},\boldsymbol{x}) - 2\boldsymbol{\beta}^T \boldsymbol{k}(\boldsymbol{x}) + \text{const.} \tag{2.45}$$

This optimization problem is typically easy to solve. For example, for the squared exponential kernel, the optimal $\boldsymbol{x}_*$ can be found by fixed point iteration [23]. For derivation of the kernel sum rule and the kernel Bayes' rule, we refer to [74].

While kernel embedding offers an elegant solution for manipulating distributions, model selection becomes a challenging problem. Model selection involves choosing an appropriate kernel function and its hyper-parameters. As with Gaussian Processes, prior knowledge about task often gives a good intuition about the kernel function. However, the hyper-parameters have to be optimized for good performance. When using Gaussian Processes, we typically maximize the log-likelihood of the predictions. However, when using kernel embedding, we do not have access to a parametric likelihood function and we cannot directly maximize the log-likelihood. On the other hand, other approaches, such as cross validation are applicable for tuning the hyper-parameters.

# Chapter 3

# Learning Generalized Robot Skills using Contextual REPS

In order to let robots operate outside laboratory environments, we need novel learning algorithms that allow robots to adapt their skills to new situations based on their perception. We showed such a problem in Fig. 1.3. The robot has to learn a throwing stroke, but the target position to throw to, which is defined by the context variable $\boldsymbol{s}$, changes between task executions. One way to solve the problem is to demonstrate the throwing stroke for a wide variety of target positions, and interpolate between the demonstrations for a new, yet unseen context. While this approach offers good performance for the observed contexts, the generalization efficiency of the skill will inherently depend on the demonstrations, which are often suboptimal in terms of consumed energy and accuracy. Furthermore, interpolation between demonstrated

skills is not straightforward. As the lower-level policy usually encodes joint trajectories over time, averaging over these trajectories might lead to an infeasible solution.

To account for suboptimal demonstrations and adaptation to new situations, the generalization is often augmented with learning. Typically, the two problems are solved separately, that is, first improve upon the acquired skills, then use a generalization approach to provide the skill for a new context. However, by decomposing the objectives to two distinct goals, the problem might be more difficult to solve. Thus, to efficiently solve the generalization and learning problem, we need an algorithm that treats the two problem in a single framework. In recent years, contextual policy search algorithms have been proposed to tackle this problem [14, 35, 47]. Contextual PS algorithms (Sec. 2.1.3) learn an upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, which provides the lower-level policy parameters $\boldsymbol{\omega}$ by conditioning on the observed context $\boldsymbol{s}$. As the objective of contextual PS algorithms is to maximize the expected reward over the context and lower-level policy parameter distribution $p(\boldsymbol{s},\boldsymbol{\omega})$, generalization and learning are directly addressed.

When learning robot skills, the upper-level policy provides explorative skill parameters to search for better solutions of the learning task. However, explorative actions might lead to infeasible, or even dangerous skills, which might cause damage to the robot and the environment. To account for smooth learning and avoiding aggressive exploration, the Relative Entropy Policy Search [55] algorithm offers an elegant solution to upper bound the experience loss between policy updates. The updated policy will provide robot trajectories, which are close to the already observed and safe trajectory distribution. For more details on REPS, we refer to Sec. 2.1.1.

In the following, we describe the contextual extension to episode-based REPS for learning generalized robot skills. Contextual REPS uses the upper bound over the joint context-policy parameter space, and thus, it aims to provide a high quality policy over all possible tasks. Choosing a good upper bound parameter is less problem dependent and it is typically easy to choose for a learning problem. Other contextual policy search methods do not use the information loss bound and require heuristics to set the temperature parameter [35, 47], which will ultimately influence the greediness of policy updates. The temperature parameter is typically more difficult to choose for good learning performance than the upper-bound on information loss.

In the following, we give an overview on related work in robot skill generalization and learning. Then, we formulate the contextual skill learning framework based on the ideas of episode-based context-free REPS. We show that the solution to the problem exists in closed form, and we provide a sample-based version of the algorithm, which is useful for robot skill learning problems. Subsequently, we show simulation results on complex robot learning tasks, such as robot throwing, robot hockey and robot table tennis.

Contextual REPS has first appeared in [13], but thorough evaluation with complex robot learning tasks has also been published by Kupcsik et al. [41, 42] and by Neumann et al. [48].

## 3.1 Related Work

Robot skill generalization has been investigated by many researchers in recent years. In [24, 86], robot skills are represented by Dynamic Movement Primitives [31]. The DMPs are generalized using demonstrated trajectories. First, the DMP parameters $\boldsymbol{\omega}$ are extracted from a human expert's demonstration in a specific context $\boldsymbol{s}$. Using multiple demonstrations, a library of context-parameter pairs $\{\boldsymbol{s},\boldsymbol{\omega}\}_{i=1}^{N}$ is built up. Subsequently, the DMP parameters $\boldsymbol{\omega}_*$ for a query context $\boldsymbol{s}_*$ is chosen using regression techniques. While generalization has proven to be accurate in humanoid reaching, grasping and drumming, the parameters are not improved by reinforcement learning. Thus, the quality of the reproduced skill inherently depends on the quality and quantity of the expert demonstration [44]. In [22], Gaussian Process regression has been applied to generalize demonstrated skills to new situations. The proposed method can also be applied in an online fashion, where the context changes over time. Promising results have been shown for robot grasping task, where the target object was in motion, while the robot was executing the reaching skill. Stulp et al. [77] proposes a context-based regression technique to generalize over DMP parameters from demonstrations. Promising results have been shown for object transportation, where the context was represented by the height of an obstacle between start and goal positions.

To account for skill improvement, policy search methods have been applied [35, 47]. Kober et al. [35] proposed the Cost-regularized Kernel Regression algorithm (see Sec. 2.1.3) to learn DMP parameters for throwing a dart at different targets and for robot

table tennis. However, CrKR cannot scale well to higher dimensional learning problems due to its uncorrelated exploration strategy. Another PS algorithm is proposed in [47] by Neumann for robot skill generalization. The algorithm uses a probabilistic approach based on variational inference to solve the underlying RL problem. The proposed method was able to generalize a lower-level controller that balances a 4 DoF planar robot around the upright position after a random initial push. However, the solution for the proposed PS algorithm cannot be computed in closed form for most upper-level policies and it is computationally very costly to obtain. Recently, the Mixture of Movement Primitives (MoMP) algorithm has been introduced [45] for robot table tennis. The MoMP approach first initializes a library of movement primitives and contexts using human demonstrations. Then, given a query context, a gating network is used to combine the demonstrated DMPs into a single one, which is then executed on the robot. The gating network parameters can be adapted based on how successful the movement primitives are in the given context. Promising results in real robot table tennis have been presented [45]. However, the formulation of the learning problem required a lot of prior knowledge and it is unclear how the algorithm scales to domains where this prior knowledge is not applicable. In [14], a hierarchical version of REPS is applied to learn multiple options to execute a given task. Daniel et. al [13] proposed a time indexed version of REPS to combine skills sequentially in a robot hockey game. The algorithm learns an upper-level policy with the aim of achieving good long-term performance, instead of independently maximizing the reward for each stage of the task. Finally, a general robot skill learning framework has been proposed by daSilva et al. [12]. The approach separates generalization and policy learning to a classification and a regression problem. The resulting

policy is a mixture of finite amount of local policies. However, choosing the amount of local policies is not straightforward and separating the generalization and policy improvement step into two distinct algorithms seem data inefficient and counter intuitive.

## 3.2 Contextual Episode-based REPS

After reviewing relevant works in the literature, we turn our attention to the contextual extension for the Relative Entropy Policy Search algorithm. The intuition of REPS [55] is to maximize the expected reward, while staying close to the observed data to balance out exploration and experience loss. REPS uses an information theoretic approach, where the relative entropy between consecutive trajectory distributions is bounded. By upper bounding the experience loss, we can avoid wild exploration, which could lead to robot damage. The relative entropy bound results in smooth learning, which avoids greedy updates and prevents premature convergence [14, 42]. In the episodic learning setting, the context $\boldsymbol{s}$ and the parameter $\boldsymbol{\omega}$ uniquely determine the trajectory distribution [14]. For this reason, the trajectory distribution can be abstracted as the joint distribution over the parameter vector $\boldsymbol{\omega}$ and the context $\boldsymbol{s}$, i.e., $p(\boldsymbol{s},\boldsymbol{\omega}) = \mu(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})$. To bound the relative entropy between consecutive trajectory distributions, contextual episode-based REPS uses the constraint

$$\int_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})} d\boldsymbol{s} d\boldsymbol{\omega} \leq \epsilon, \tag{3.1}$$

where $p(\boldsymbol{s},\boldsymbol{\omega})$ represent the updated and $q(\boldsymbol{s},\boldsymbol{\omega})$ is the previously used context-parameter distribution. The parameter $\epsilon \in \mathbb{R}^+$ is the upper bound of the relative entropy. A smaller value of $\epsilon$ results in more conservative policy updates, while a higher $\epsilon$ leads to faster converging policies.

As the context distribution $\mu(\boldsymbol{s})$ is defined by the learning problem and cannot be chosen by the learning algorithm, the constraints $\int_{\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) = \mu(\boldsymbol{s}),\ \forall \boldsymbol{s}$ must also be satisfied. However, in the case of continuous context variables, we would have infinite number of instances of this constraint. To keep the optimization problem tractable, we require only to match feature averages instead of single probability values, i.e.,

$$\int_{\boldsymbol{s}} p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \tag{3.2}$$

where $p(\boldsymbol{s}) = \int_{\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})d\boldsymbol{\omega}$. The feature vector is denoted as $\boldsymbol{\phi}(\boldsymbol{s})$, while $\hat{\boldsymbol{\phi}}$ denotes the observed average feature vector. For example, if the feature vector contains all linear and quadratic terms of the context, the above constraint translates to matching the mean and the variance of the distributions $p(\boldsymbol{s})$ and $\mu(\boldsymbol{s})$. The contextual episode-based REPS learning problem is now given by

$$\max_{p} \quad \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}d\boldsymbol{s}d\boldsymbol{\omega}, \tag{3.3}$$

$$\text{s.t.:} \quad \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\log\frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})}d\boldsymbol{s}d\boldsymbol{\omega} \leq \epsilon, \tag{3.4}$$

$$\iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\boldsymbol{\phi}(\boldsymbol{s})d\boldsymbol{s}d\boldsymbol{\omega} = \hat{\boldsymbol{\phi}}, \tag{3.5}$$

$$\iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})d\boldsymbol{s}d\boldsymbol{\omega} = 1. \tag{3.6}$$

The optimization objective is to maximize the expected reward $\mathcal{R}_{s\omega}$ over contexts $s$ and lower-level policy parameters $\omega$ using the joint distribution $p(s,\omega)$ (Eq. (3.3)). For the sake of clarity, we restate the definition of the expected reward

$$\mathcal{R}_{s\omega} = \int p(\tau; s, \omega) R(\tau, s) d\tau,$$

where $R(\tau, s)$ is the reward for executing trajectory $\tau$ in context $s$ and $p(\tau; s, \omega)$ is the corresponding trajectory distribution. The constraint in Eq. (3.4) guarantees that the non-negative KL-divergence between consecutive policies is upper bounded. The constraint in Eq. (3.5) ensures that the expected and the observed feature average match each other, while with the constraint in Eq. (3.6) we enforce that the updated policy is a proper distribution.

The emerging constrained optimization problem can be solved by the Lagrange multiplier method. The closed form solution for the new distribution is given by

$$p(s,\omega) \propto q(s,\omega) \exp\left(\frac{\mathcal{R}_{s\omega} - V(s)}{\eta}\right). \tag{3.7}$$

Here, $V(s) = \gamma^T \phi(s)$ is a context dependent baseline, while $\eta$ and $\gamma$ are Lagrangian parameters. Subtracting the baseline from the reward turns the term $\mathcal{R}_{s\omega} - V(s)$ into an advantage function in context $s$. The temperature parameter $\eta$ scales the advantage term such that the relative entropy bound is met after the policy update. The Lagrangian parameters are found by optimizing the dual function

$$g(\eta, \gamma) = \eta \log\left(\iint_{s,\omega} q(s,\omega) \exp\left(\frac{\mathcal{R}_{s\omega} - V(s)}{\eta}\right) ds d\omega\right) + \eta\epsilon + \gamma^T \hat{\phi}. \tag{3.8}$$

The dual function is convex in $\boldsymbol{\gamma}$ and $\eta$, and the corresponding gradients can be obtained in closed form. In contrast to recent EM-based policy search methods [36, 47], the exponential weighting emerges from the relative entropy bound and does not require additional assumptions. For details of the derivation, we refer to Appendix A.

## 3.3 Sample-based Contextual REPS

As the relationship between the context-policy parameter pair $\{\boldsymbol{s}, \boldsymbol{\omega}\}$ and the corresponding expected reward $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}$ is not known, sample evaluations are used to approximate the integral given in the dual function [14, 42]. We denote these evaluations as rollouts. To execute the $i^{th}$ rollout, we first observe the context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$. Subsequently, we sample the lower-level controller parameter using the upper-level policy $\boldsymbol{\omega}^{[i]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$. Finally, we execute the lower-level policy with parametrization $\boldsymbol{\omega}^{[i]}$ in context $\boldsymbol{s}^{[i]}$ to obtain $\mathcal{R}^{[i]}_{\boldsymbol{s}\boldsymbol{\omega}}$. This process is repeated for $N$ rollouts $\mathcal{D} = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, \mathcal{R}^{[i]}_{\boldsymbol{s}\boldsymbol{\omega}}\}$, $i = 1, \ldots, N$, such that we can accurately approximate the integral given in the dual function with samples. The sample-based approximation of the dual function is given by

$$g(\eta, \boldsymbol{\gamma}; \mathcal{D}) = \eta \log\left(\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{\mathcal{R}^{[i]}_{\boldsymbol{s}\boldsymbol{\omega}} - \boldsymbol{\gamma}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right)\right) + \eta\epsilon + \boldsymbol{\gamma}^T \hat{\boldsymbol{\phi}}, \tag{3.9}$$

where we replaced the integral with a sum. For more details of the derivation, see Appendix A.

`Contextual REPS Algorithm`

**Input:** relative entropy bound $\epsilon$, initial policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, number of policy updates $K$, number of old datasets for reusing data $L$.

**for** $k = 1,\ldots,K$

  **for** $i = 1,\ldots,N$

    *Observe context* $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$, $i = 1,\ldots,N$

    *Execute policy with* $\boldsymbol{\omega}^{[i]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$, *observe trajectory* $\boldsymbol{\tau}^{[i]}$

    *Compute rewards* $\mathcal{R}_{\boldsymbol{s\omega}}^{[i]} = R(\boldsymbol{\tau}^{[i]}, \boldsymbol{s}^{[i]})$

  **New dataset:** $\mathcal{D}_k = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, R_{\boldsymbol{s\omega}}^{[i]}\}_{i=1\ldots N}$

  **Reuse old datasets:** $\mathcal{D} = \{\mathcal{D}_l\}_{l=\max(1,k-L)\ldots k}$

  **Update policy:**

    *Optimize dual function using* $\mathcal{D}$, *Eq. (3.9)*

      $[\eta, \boldsymbol{\gamma}] = \text{argmin}_{\eta',\boldsymbol{\gamma}'} g(\eta', \boldsymbol{\gamma}'; \mathcal{D})$

    *with gradients in Eqs. (A.11) and (A.12)*

    *Compute sample weighting:*

      $p^{[i]} \propto \exp\left(\frac{\mathcal{R}_{\boldsymbol{s\omega}}^{[i]} - \boldsymbol{\gamma}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right)$, *for each sample* $i$ *in* $\mathcal{D}$

    *Update policy* $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ *with weighted ML*

      *using* $\mathcal{D}$ *and* $p^{[i]}$, *Eqs. (3.11-3.12)*

  **end**

**Output:** policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$

TABLE 3.1: In each iteration of the contextual REPS algorithm, we collect a dataset $\mathcal{D}_k = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, R_{\boldsymbol{s\omega}}^{[i]}\}_{i=1\ldots N}$ by performing rollouts on the real system. For the REPS algorithm, we reuse the last $L$ datasets in combine them in the dataset $\mathcal{D}$. Finally, we update the policy by optimizing the dual function on dataset $\mathcal{D}$, computing the sample weights and performing a weighted maximum likelihood (ML) estimate to obtain a new parametric policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$.

Using the optimized Lagrangian parameters $\boldsymbol{\gamma}$ and $\eta$, we can compute the probabilities of the updated context-parameter distribution for our finite set of samples using

$$p^{[i]} \propto \exp\left(\frac{\mathcal{R}_{\boldsymbol{s\omega}}^{[i]} - V(\boldsymbol{s}^{[i]})}{\eta}\right). \tag{3.10}$$

However, in order to generate new samples, we need a parametric model to estimate $\pi(\boldsymbol{\omega}|\boldsymbol{s})$. Hence, we estimate the parameters of this model given the samples and using $p^{[i]}$ as weight for these samples. For example, for a linear Gaussian model $\pi(\boldsymbol{\omega}|\boldsymbol{s}) =$

$\mathcal{N}(\boldsymbol{\omega}|\boldsymbol{a}+\boldsymbol{A}\boldsymbol{s},\boldsymbol{\Sigma})$, we can compute the parameters $\boldsymbol{\theta}=\{\boldsymbol{a},\boldsymbol{A},\boldsymbol{\Sigma}\}$ with a weighted maximum likelihood estimation

$$\begin{bmatrix} \boldsymbol{a}^T \\ \boldsymbol{A}^T \end{bmatrix} = (\boldsymbol{S}^T\boldsymbol{P}\boldsymbol{S})^{-1}\boldsymbol{S}^T\boldsymbol{P}\boldsymbol{B}, \tag{3.11}$$

$$\boldsymbol{\Sigma} = \frac{\sum_{i=1}^{N} p^{[i]}(\boldsymbol{\omega}^{[i]}-\boldsymbol{\mu}^{[i]})(\boldsymbol{\omega}^{[i]}-\boldsymbol{\mu}^{[i]})^T}{\sum_{i=1}^{N} p^{[i]}}, \tag{3.12}$$

$$\boldsymbol{\mu}^{[i]} = \boldsymbol{a}+\boldsymbol{A}\boldsymbol{s}^{[i]}, \tag{3.13}$$

where $\boldsymbol{S}=[\hat{\boldsymbol{s}}^{[1]},\ldots,\hat{\boldsymbol{s}}^{[N]}]^T$ is the context matrix with $\hat{\boldsymbol{s}}^{[i]}=[1,\boldsymbol{s}^{[i]^T}]^T$, $\boldsymbol{B}=[\boldsymbol{\omega}^{[1]},\ldots,\boldsymbol{\omega}^{[N]}]^T$ is the parameter matrix and $\boldsymbol{P}_{ii}=p^{[i]}$ is the diagonal weighting matrix. When updating the policy parameters, we are not solely restricted to use the last $N$ samples. To improve the accuracy of policy updates, we can define $q(\boldsymbol{s},\boldsymbol{\omega})$ to be a mixture of the last $L$ policies, and hence reuse old samples without the need of importance weighting [28]. The model-free contextual REPS algorithm is summarized in Table 3.1.

## 3.4 Results

In this section we will present results for learning contextual skills for complex robot tasks. We will evaluate contextual REPS on three challenging problems. First, we begin with a ball throwing task for a 4-DoF planar robot. The goal of the robot is to throw a ball at a target position, which is changing between experiments. In the second task we learn hockey skills for a 7-DoF lightweight robot arm. The robot has to shoot at a target puck whose position is changing between task executions. Finally, we test contextual REPS in a challenging robot table tennis task using a 8-DoF tendon

driven robotic system. The robot not only has to hit incoming balls with the provided racket, but has to return at specified target positions on the opponent's side of the table. All the results presented in this section are evaluated in simulations using the SL software package [66]. For each learning task we use Dynamic Movement Primitives as the lower-level policy.

**Exploiting Reward Models as Prior Knowledge.** Throughout the experiments, we will evaluate another version of contextual REPS with improved data efficiency as well. We will use the known reward model $R(\boldsymbol{\tau}, \boldsymbol{s})$ to evaluate a single trajectory outcome $\boldsymbol{\tau}$ in multiple contexts $\boldsymbol{s}^{[i]}, i = 1, \ldots, N$. As a result we will obtain multiple rewards $\mathcal{R}_{\boldsymbol{s\omega}}{}^{[i]}, i = 1, \ldots, N$ for a single trajectory in different contexts. Such strategy is possible if the evaluated trajectories $\boldsymbol{\tau}$ do not depend on the context variables $\boldsymbol{s}$. For example, if the context specifies a desired target for throwing a ball, we can use the ball trajectory to evaluate the reward for multiple target positions $\boldsymbol{s}^{[i]}, i = 1, \ldots, N$.

### 3.4.1 Ball Throwing Task

In the ball throwing task we use a 4-DoF planar robot to throw a ball at a target position. However, the target position is changing between task executions. The target coordinates $[x, y]$ are distributed uniformly in intervals $x \in [5, 15]$m and $y \in [1, 3]$m from the robot's base. The robot is roughly modeled as a human with ankle, knee hip and shoulder joints (see Fig. 3.1 for more details).

The reward function of the task is defined as the sum of three terms. The first term penalizes the minimum squared distance of the target position to the ball trajectory.

To make the problem more challenging, the second term of the reward function penalizes joint angles, which would be unrealistic for a human executing the task. Such penalties are, for example, bending the knee backwards. Finally, the third term of the reward penalizes high energy consumption. For a complete description of the task we refer to Appendix C.1.

As lower-level control policy, we used DMPs with 10 basis functions per joint. We learned the shape parameters, but fixed the final position and velocity of the DMP to be the upright position $\boldsymbol{q} = [0,0,0,\pi]$, and zero velocity. In addition, the lower-level policy also contained the release time $t_r$ of the ball as a free parameter, resulting in a 41-dimensional parameter vector $\boldsymbol{\omega}$. After generating the reference trajectory with the DMPs, we use a PD trajectory tracker controller to generate the control inputs $\boldsymbol{u}$.

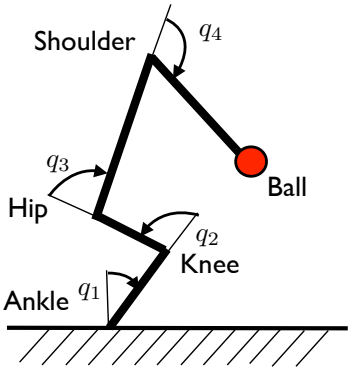



FIGURE 3.1: The 4-DoF planar robot.

The policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ was initialized such that the robot is expected to throw the ball approximately 5 m away from the robot's base without maintaining balance, which led to high penalties. We found this policy by applying the context-free REPS up to a few policy updates. The contextual REPS algorithm has learned to accurately hit the target distributed according to $\mu(\boldsymbol{s})$. Fig. 3.2 shows the learnt motion sequence for two different targets. The absolute displacement error for targets further away from the

robot's base (with context $\boldsymbol{s} \leq [13,3]^T$ m) could raise up to 0.5 m, otherwise the maximal error was smaller than 10 cm. The policy chose different DMP parametrizations and release times for different target positions. To illustrate this effect, we show two target positions $\boldsymbol{s}_1 = [6,1]$ m, $\boldsymbol{s}_2 = [12,1]$ m in Fig. 3.2. For the target farther away the robot showed a more distinctive throwing movement and released the ball slightly later.

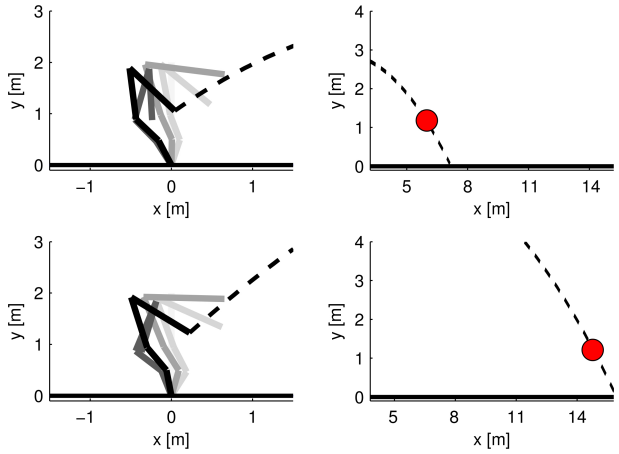



FIGURE 3.2: Throwing motion sequence. The robot releases the ball after the specified release time and hits different targets with high accuracy.

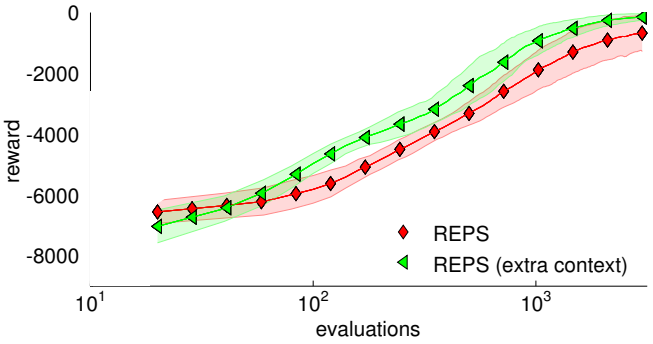



FIGURE 3.3: Learning curves for the ball-throwing problem. The shaded regions represent the standard deviation of the rewards over 20 independent trials.

The learning curves of contextual REPS are shown in Fig. 3.3. In addition, we evaluated REPS using the known reward model $R(\boldsymbol{\tau}, \boldsymbol{s})$ to generate additional samples with randomly sampled contexts $\{\boldsymbol{s}^{[i]}\}_{i=1}^{20}$. We denote these experiments as *extra context*.

Fig. 3.3 shows that REPS converged to a good solution after 5000 skill evaluations in most cases. In a few instances, however, we observed premature convergence resulting is suboptimal performance. The performance of REPS could be improved by using extra samples generated with the known reward model (*extra context*). In this case, REPS always converged to good solutions.

Throughout the experiments, we evaluated 25 rollouts between each policy update. However, we reused the last 200 samples to improve data-efficiency and avoid premature convergence. We set the upper bound on experience loss $\epsilon$ to 0.6, which provided relatively fast convergence, without compromising the smoothness of the learning curves. However, we were not able to decrease the amount of required rollouts anymore without risking premature convergence.

### 3.4.2 Robot Hockey Task

In the robot hockey task we use a 7-DoF lightweight robot arm to shoot a puck at a target puck. The task involves two goals: shooting the provided puck in the direction of the target puck and shoot it with the right force, such that the target puck's displacement meets a specified value. The position of the target puck $[b_x, b_y]$ and its required displacement $d^*$ are changing between task executions (Fig. 3.4). Thus, the context variable is defined as $\boldsymbol{s} = [b_x, b_y, d^*]$. The reward for the task penalizes missing the target puck and inaccurate displacement of the target puck. For a complete description of the task we refer to Appendix C.2.

We encoded the hitting motion into a DMP. The weight parameters were set by imitation learning. We learn only the final position $g$, final velocity $\dot{g}$ and the time scaling

parameter $\tau$ of the DMP. As the robot has seven degrees of freedom, we get a 15-dimensional parameter vector $\boldsymbol{\omega}$ of the lower-level policy. For trajectory tracking, we used an inverse dynamics controller.

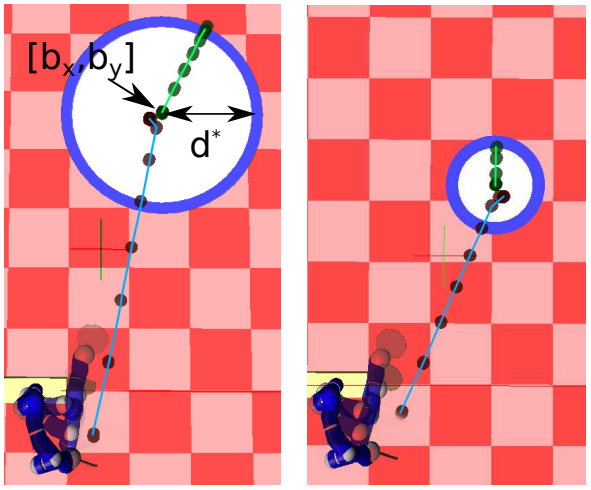



FIGURE 3.4: Robot hockey task. The robot shoots the control puck at the target puck to make the target puck move for a specified distance. Both, the initial location of the target puck $[b_x, b_y]^T$ and the desired distance $d^*$ to move the puck were varied. The context was given by $\boldsymbol{s} = [b_x, b_y, d^*]$. The learnt skill for two different contexts $\boldsymbol{s}$ is shown, where the robot learned to place the target puck at the desired distance.

We compared contextual REPS to CrKR [35], a state-of-the-art model-free contextual policy search method. The resulting learning curves are shown in Fig. 3.4. CrKR uses a kernel-based representation of the policy. For a fair comparison, we used a linear kernel for CrKR. The results show that CrKR could not compete with model-free contextual REPS. We believe the reason for the worse performance of CrKR lies in its uncorrelated exploration strategy. The resulting policy of CrKR is a Gaussian with a diagonal covariance matrix, while REPS estimates a full covariance matrix. Moreover, CrKR does not use an information-theoretic bound to determine the weightings of the samples. The learnt movement is shown in Fig. 3.4 for two different contexts.

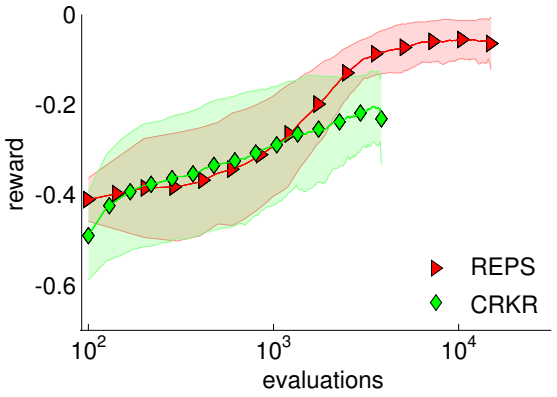



FIGURE 3.5: The learning curves for the robot hockey task with contextual REPS and CrKR.

By the end of the learning process, contextual REPS always provided policies, which were able to hit the target puck for 95% of the experiments and the displacement error remained below 20cm. However, CrKR provided policies that could hit the target puck only with a lower probability and displacement errors are considerably higher compared to REPS. Based on our observations, CrKR could not improve the policy after 5000 rollouts. However, with $\epsilon = 0.5$, contextual REPS could gradually improve the expected reward up to 8000 evaluations.

### 3.4.3 Robot Table Tennis

In this task, the goal is to learn hitting strokes for the table tennis game with a tendon driven Biorob robot arm. The robot is attached to a moving platform, which can move in the horizontal plane. The resulting robotic system has 8 degrees of freedom. The goal of the robot is to return incoming balls to specified target locations on the opponent's side of the table. However, the incoming ball has varying initial velocities $[v_x, v_y, v_z]$, which results in varying landing position on the robot's side of the table.

Thus, the robot has to generalize the hitting skill, such that it successfully returns the ball for a wide variety of incoming ball directions and velocities, as in a real table tennis game. See Figure 3.6 for an illustration.

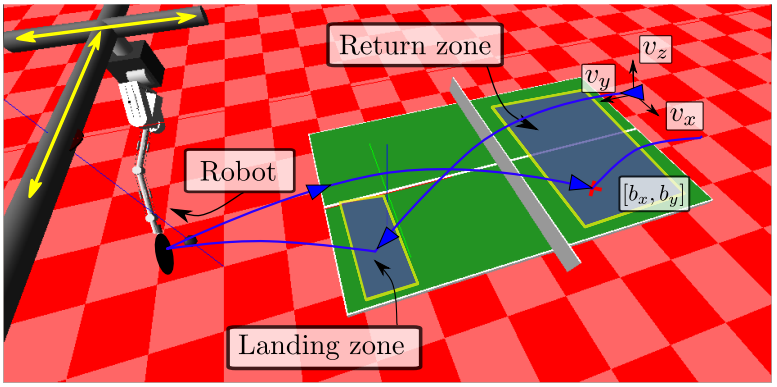



FIGURE 3.6: The table tennis learning setup. The incoming ball has a fix initial position and a random initial velocity of $\boldsymbol{v} = [v_x, v_y, v_z]^T$. The velocity distribution is defined such that the incoming ball lands inside the *Landing zone*. The goal of the robot is to hit the incoming ball back to the return position $\boldsymbol{b} = [b_x, b_y]^T$, which is distributed uniformly inside the *Return zone*. The context variable contains the initial velocity of the ball and the target return position $\boldsymbol{s} = [\boldsymbol{v}^T, \boldsymbol{b}^T]^T$.

However, the goal of the task is not only to hit the incoming ball, but to return it to a specific target location $[b_x, b_y]$ on the opponent side of the table. Thus, the upper-level policy will choose a parametrization based on context $\boldsymbol{s} = [v_x, v_y, v_z, b_x, b_y]$. The reward function for the task consists of two terms

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -c_1 \min ||\boldsymbol{\tau}_b - \boldsymbol{\tau}_r||_2 - c_2 ||\boldsymbol{b} - \boldsymbol{p}||_2.$$

The first term penalizes the minimal distance of the racket trajectory $\boldsymbol{\tau}_r$ to the incoming ball trajectory $\boldsymbol{\tau}_b$. The second term penalizes the displacement of the returned ball landing coordinate $\boldsymbol{p}$ to the desired landing position $\boldsymbol{b}$. We use equal weighting for the two terms $c_1 = c_2$. For a more detailed description of the task, we refer to

Appendix C.3.

For this task we represent the skill with a DMP. We set the values of the shape parameters of the DMP by kinesthetic teach-in, and learn only the final positions, velocities and the time scaling parameter of the DMP, altogether 17 parameters. We set the upper bound parameter of the contextual REPS algorithm to $\epsilon = 0.6$, which usually provided smooth learning without the danger of getting stuck in local optima. The learning curves of contextual REPS is shown in Figure 3.7. A reward level of $-0.1$ rep-

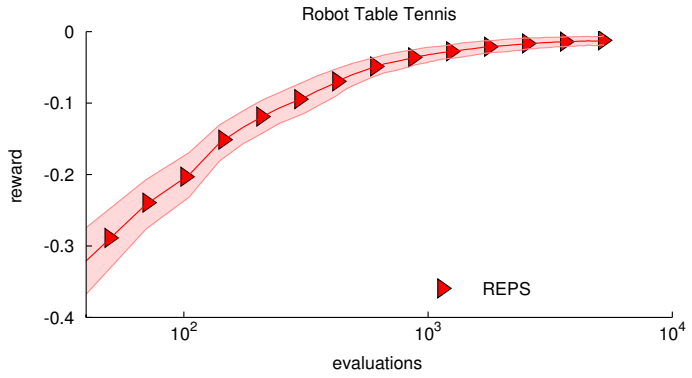



FIGURE 3.7: The learning curves of the table tennis experiment. REPS is able to learn a good policy after 4000 evaluations, but it sometimes learns a sub-optimal policy that hits the ball in the net.

resents a policy, which provides a skill that is able to hit the incoming ball, but it fails to return the ball to the opponent's side of the table. While contextual REPS consistently provided good learning performance, we sometimes observed policies, which produced hitting skills that landed the ball in the net. This is a suboptimal solution with a reward in interval $[-0.05, -0.02]$. However, for most of the evaluations high quality policies were found after observing 4000 rollouts, which landed the ball in $20-30$cm radius of the target location with a reward above $-0.02$. We illustrate the

experiment for two different contexts in Figure 3.8. After a fast and accurate forehand stroke, the robot returns the ball near the desired target position.

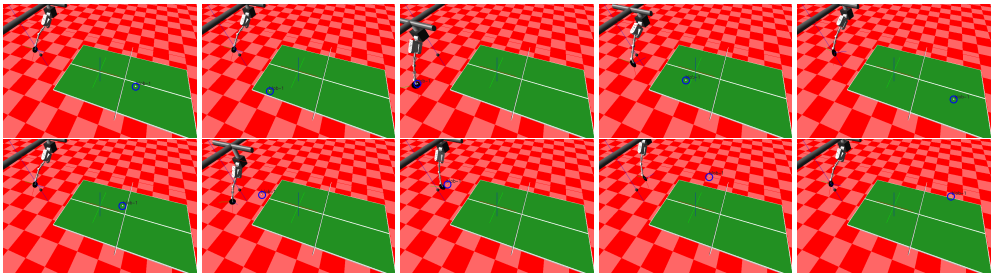


FIGURE 3.8: Animation of two shots to different targets and different serving positions of the ball learned with REPS.

## 3.5 Discussion

A seemingly simpler approach to optimize the upper-level policy would be to solve the optimization problem

$$\max_{\pi} \quad \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\omega}} \pi(\boldsymbol{\omega}|\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} d\boldsymbol{\omega} d\boldsymbol{s} \tag{3.14}$$

$$\text{s.t.:} \quad \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\omega}} \pi(\boldsymbol{\omega}|\boldsymbol{s}) \log \frac{\pi(\boldsymbol{\omega}|\boldsymbol{s})}{q(\boldsymbol{\omega}|\boldsymbol{s})} d\boldsymbol{\omega} d\boldsymbol{s} \leq \epsilon, \tag{3.15}$$

$$\int_{\boldsymbol{\omega}} \pi(\boldsymbol{\omega}|\boldsymbol{s}) d\boldsymbol{\omega} = 1, \forall \boldsymbol{s}. \tag{3.16}$$

However, for this optimization problem, the sample-based dual function to optimize takes the form

$$g(\eta, \lambda) = \eta \sum_{j=1}^{M} \mu(\boldsymbol{s}_j) \log \frac{1}{N} \sum_{i=1}^{N} \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}{}^{[i,j]}}{\eta}\right) + \eta\epsilon. \tag{3.17}$$

As we can see, in this case we need multiple samples of $\boldsymbol{\omega}_{1:N}$ for a single context $\boldsymbol{s}_j$ to evaluate the dual function, which would significantly slow down the learning progress. In the contextual REPS formulation used in Eqs. (3.3) - (3.6), we circumvent this problem by directly optimizing over the joint space $p(\boldsymbol{s},\boldsymbol{\omega})$, with an additional constraint to match feature averages in order to be consistent with the real context distribution $\mu(\boldsymbol{s})$.

In our experiments, contextual REPS has proven to be highly efficient in learning upper-level policies $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, which condition on the observed context $\boldsymbol{s}$ to choose parametrization $\boldsymbol{\omega}$. The only parameter that needs to be chosen is the upper bound on experience loss $\epsilon$. This parameter is less dependent on the learning problem as opposed to a learning rate for PG algorithms, and it is easy to choose for most learning problems. As we could see from the experiments, a range of $\epsilon \in [0.5, 1]$ can provide relatively fast and smooth learning performances. However, the amount of sample evaluations required by contextual REPS tends to be high, even by reusing previous sample evaluations for policy update. Thus, the resulting learning algorithm works well in simulations, but solving real robot tasks remains challenging. This is one of the major disadvantage of contextual REPS and almost all other model-free PS method. Although using the reward function to evaluate the reward for a single parametrization $\boldsymbol{\omega}$ and multiple artificial contexts $\{\boldsymbol{s}^{[i]}\}_{i=1}^{N}$ improves the quality of the final policy, it fails to considerably improve data-efficiency.

When using the sample-based implementation of contextual REPS, we need to fit the parameters of the upper-level policy at each policy update step. For a linear Gaussian representation $\pi(\boldsymbol{\omega}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{a} + \boldsymbol{A}\boldsymbol{s}, \boldsymbol{\Sigma})$, we have to find the optimal parameters

$\{\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}\}$. For an $N$-dimensional parameter vector $\boldsymbol{\omega}$, $\boldsymbol{a}$ and $\boldsymbol{A}$ are $N$-dimensional, but to find $\boldsymbol{\Sigma}$, we need to set $N^2$ parameters, which leads to a less accurate approximation of the true policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ for $N > 50$. Thus, contextual REPS does not scale well to learning problems above 50 parameters. More sophisticated models for $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ might overfit and endanger the generalization property of the upper-level policy.

To ensure safety and efficiency during the learning process, we have to properly initialize the upper level policy parameters. In our experiments we used a linear Gaussian model for representing of the upper level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{a} + \boldsymbol{A}\boldsymbol{s}, \boldsymbol{\Sigma})$, with parameters $\{\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}\}$. Initially, we set the linear model $\boldsymbol{A} = \mathbf{0}$ and obtain the offset parameter $\boldsymbol{a}$ by imitation learning. We set the initial exploration covariance $\boldsymbol{\Sigma}$ as diagonal matrix such that we get enough initial exploration of the parameter space, but exploration still remains safe for the robot. Contextual REPS typically decreases the exploration variance at each policy update step until $\boldsymbol{\Sigma}$ collapses and the policy parameters converge to the final solution. Thus, the initial $\boldsymbol{\Sigma}$ has to be chosen carefully, such that the optimal solution is *within* the range of the initial exploration range.

With contextual REPS, and with many other model-free PS approaches, we usually evaluate only one trajectory to approximate the expected reward for a given controller parametrization, that is, $\mathcal{R}_{\boldsymbol{\omega}} \simeq R(\boldsymbol{\tau})$. However, for stochastic lower-level policies, or in the presence of stochastic robot dynamics, the resulting inaccurate approximation might introduce bias in the policy update. To obtain a more accurate approximation of the real expected reward, we can evaluate the given controller parametrization multiple times. While this approach provides an unbiased $\mathcal{R}_{\boldsymbol{\omega}}$ in the limit, the

already high amount of required policy evaluations grows even further, resulting in an increased data-inefficiency.

# Chapter 4

# Model-based Contextual Robot Skill Learning

In general, model-based PS algorithms (Sec. 2.1.2) use a model of the robot and its environment for learning the required control policy without extensive use of the real hardware. The resulting algorithms are more data-efficient compared to model-free approaches, but the final learned policy inherently depends on the quality of the learned or hand-tuned model. When working with an inaccurate model, the optimized policy might become biased, and thus, it could perform poorly on the real hardware [6]. While we can exploit prior knowledge about the dynamics to obtain a more or less accurate mathematical model, learning is often used to directly optimize the model from measurement data [32, 54]. Data driven learning is able to account for unmodeled nonlinearities and stochasticity in the dynamics, and thus,

the acquired model is often less biased. However, generalization remains a significant challenge for the learning approach.

Previously, we briefly discussed two approaches for learning robot dynamics: Locally Weighted Bayesian Regression (LWBR) and Gaussian Process (GP) regression. Both methods learn a probabilistic, time-independent model of the robot dynamics, which can be applied to evaluate the robot experiment in computer simulations. However, one significant difference between the LWBR and GP regression is that the former uses a parametric representation of the model, while the latter represents the dynamics in a non-parametric way. Non-parametric approaches produce less biased models of the dynamics, but the computational demand for training and prediction might be impractically large. Nevertheless, due to the promise of high quality learned policies with the use of unbiased models, it is often beneficial to rely on computationally more involved, but less biased policy updates by the use of non-parametric models. The state-of-the-art model-based policy search algorithm, PILCO [17], also proposes the use of GP models. PILCO is able to learn lower-level controllers, e.g., for simple robot manipulators with unprecedented data-efficiency[18]. Although the required interaction time with the real hardware is typically below a few minutes, computation of the optimal policy with main-stream desktop PCs can raise up to a couple of hours.

Despite their data-efficiency, model-based policy search algorithms have only been applied to learn lower-level controllers [3, 6, 17, 18, 32, 49, 68]. Extension of these algorithms to learn contextual upper-level policies is not straightforward, due to assumptions on the lower-level control policy class and the reward function. In order

to learn contextual robot skills efficiently, we need novel algorithms that combine the generality of model-free policy search methods and the data-efficiency of model-based algorithms. To the best of our knowledge, model-based algorithms have not yet been proposed to learn upper-level, or even contextual policies for robot skill optimization.

In Chapter 3 we have evaluated the efficiency of contextual REPS to learn complex robot skills. While the final learnt policy provided high quality generalized skills, due to the data inefficiency, contextual REPS required thousands of skill evaluations to achieve convergence. This inefficiency poses a significant challenge for learning complex robot skills for real-world tasks with model-free PS algorithms. Moreover, model-free REPS produces biased policy updates as the expected reward $\mathcal{R}_{s\omega}$ is typically evaluated using a single rollout $\mathcal{R}_{s\omega} \approx R(\boldsymbol{\tau}, \boldsymbol{s})$. This bias comes from the exponential sample weighting of REPS in Eq. (3.10). For example, if we only have two actions $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$, and the expected reward of $\boldsymbol{\omega}_1$ is lower than the expected reward of $\boldsymbol{\omega}_2$ but the variance of the reward for $\boldsymbol{\omega}_1$ is higher (such that there will be samples from $\boldsymbol{\omega}_1$ with higher reward than all samples from $\boldsymbol{\omega}_2$), REPS will prefer $\boldsymbol{\omega}_1$. Hence, the resulting policy is risk-seeking, a behavior that we want in general to avoid. The same bias is inherent in all other PS methods that are based on weighting samples with an exponential function, for example PoWER [36], CrKR [35] and $\text{PI}^2$ [84].

To account for data-inefficiency and biased policy updates, we propose a model-based version of the contextual REPS algorithm, the Gaussian Process Relative Entropy Policy Search (GPREPS). GPREPS [42, 48] uses Gaussian Process models to learn the dynamics model of the robot and its environment. These models are then used to

evaluate experiments using computer simulations. By using a probabilistic model-based approach, we avoid the above mentioned two disadvantages of model-free REPS, the data-inefficiency and the biased policy updates. Since we are applying GP models, we can integrate out the model uncertainty, and thus, we obtain less biased prediction of the experiment outcome and the expected reward $\mathcal{R}_{s\omega}$. To the best of our knowledge, GPREPS is the first algorithm that uses the combination of model-based PS and contextual model-free PS. Thus, GPREPS is more generally applicable than the state of the art model-based PS, such as PILCO [17], and it is more data efficient compared to model-free REPS and other model-free contextual PS algorithms [14, 35, 47]. We will evaluate GPREPS on the same learning tasks as with the contextual REPS. Additionally, we will present real robot results.

In the following, we present the GPREPS algorithm. Then, we discuss how to learn and use GP models to obtain an accurate evaluation of $\mathcal{R}_{s\omega}$. We will compare several GP modeling techniques based on their prediction efficiency and computation times. Subsequently, we discuss a probabilistic model training algorithm for more accurate trajectory prediction. Finally, we present the robot results.

## 4.1 Gaussian Process REPS

Our main motivation to use models with contextual policy search is two-fold. First, we want to improve the data-efficiency of the model-free REPS using artificial rollouts generated by computer simulation. Second, we want to obtain the accurate expected reward $\mathcal{R}_{s\omega}$ for a given context-policy parameter pair to avoid the bias in the

sample-based REPS formulation. The expected reward $\mathcal{R}_{s\omega} = \mathbb{E}_{\tau}[R(\tau, s)|s, \omega]$ can be estimated by multiple samples from the trajectory distribution $p(\tau|s, \omega)$, that is,

$$\mathcal{R}_{s\omega} = \frac{1}{L} \sum_{l=1}^{L} R(\tau^{[l]}, s), \tag{4.1}$$

where the trajectories are now generated using the learned forward models in computer simulation and we will assume that the trajectory-dependent reward function $R(\tau, s)$ is known. We use GP models to learn the forward models of the robot and its environment. Therefore, our method is called Gaussian Process Relative Entropy Policy Search (GPREPS).

In each iteration, first we collect measurement data from the robot and its environment. For data collection, we observe the context $s^{[i]}$ and sample the parameters $\omega^{[i]}$ using the upper-level policy $\pi(\omega|s^{[i]})$. Subsequently, we use the lower-level deterministic control policy $\pi_{\omega^{[i]}}(x)$ to obtain the trajectory sample $\tau^{[i]}$. It is important to evaluate sufficiently many samples to obtain enough measurement data, such that the GP models produce accurate predictions over the task relevant part of the state space. Therefore, we repeat the data collection step $N$ times. To favor data-efficiency, we wish to keep $N$ as low as possible, while learning high quality models. In experiments, we usually choose a higher $N$ in the first iteration, e.g. $N = 20$, to obtain an accurate initial GP model. After the first policy update, we decrease $N$ to significantly lower value, e.g., $N = 1$. This way, we keep updating the GP models with relevant measurement data without compromising the data-efficiency. After each data collection step, we update the GP model hyper-parameters. The GPREPS algorithm is summarized in Table 4.1.

GPREPS Algorithm

**Input:** relative entropy bound $\epsilon$, initial policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, number of policy updates $K$.

**for** $k = 1,\ldots,K$

  **Collect Data:**

    **for** $i = 1,\ldots,N$

    *Observe context* $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$, $i = 1,\ldots,N$

    *Execute policy with* $\boldsymbol{\omega}^{[i]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$

    **end**

  **Train forward models with all data, estimate** $\hat{\mu}(\boldsymbol{s})$

  **for** $j = 1,\ldots,M$

    **Predict Rewards:**

      *Draw context* $\boldsymbol{s}^{[j]} \sim \hat{\mu}(\boldsymbol{s})$

      *Draw lower-level parameters* $\boldsymbol{\omega}^{[j]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[j]})$

      *Predict L trajectories* $\boldsymbol{\tau}_j^{[l]}|\boldsymbol{s}^{[j]},\boldsymbol{\omega}^{[j]}$

      *Compute* $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[j]} = \sum_l R(\boldsymbol{\tau}_j^{[l]},\boldsymbol{s}^{[j]})/L$

  **Construct artificial dataset:** $\tilde{\mathcal{D}} = \{\boldsymbol{s}^{[j]},\boldsymbol{\omega}^{[j]},R_{\boldsymbol{s}\boldsymbol{\omega}}^{[j]}\}_{j=1\ldots M}$

  **Update Policy:**

    *Optimize dual function using* $\tilde{\mathcal{D}}$*, Eq. (3.9):*

      $[\eta,\boldsymbol{\gamma}] = \text{argmin}_{\eta',\boldsymbol{\gamma}'} g(\eta',\boldsymbol{\gamma}';\tilde{\mathcal{D}})$

    *with gradients in Eqs. (A.11) and (A.12)*

    *Compute sample weighting:*

      $p^{[j]} \propto \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[j]} - \boldsymbol{\gamma}^T\boldsymbol{\phi}(\boldsymbol{s}^{[j]})}{\eta}\right)$, $j = 1,\ldots,M$

    *Update policy* $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ *with weighted ML*

      *using* $\tilde{\mathcal{D}}$ *and* $p^{[j]}$*, Eqs. (3.11-3.12)*

  **end**

**Output:** policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$

TABLE 4.1: The GPREPS algorithm.

In the prediction step, we predict the rewards for $M$ randomly sampled context-policy parameter pairs. We refer to these samples as artificial samples. To generate artificial samples, we need to obtain an estimate of the context distribution $\hat{\mu}(\boldsymbol{s})$ using the observed data. Depending on the learning problem, we typically approximate the context distribution with a Gaussian or a uniform distribution, but prior knowledge about the task can also be exploited to approximate $\hat{\mu}(\boldsymbol{s})$. After updating the estimated context distribution $\hat{\mu}(\boldsymbol{s})$, we draw a context parameter $\boldsymbol{s}^{[j]} \sim \hat{\mu}(\boldsymbol{s})$ for each artificial sample. Subsequently, we sample from the upper-level policy $\boldsymbol{\omega}^{[j]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[j]})$

and produce $L$ sample trajectories $\boldsymbol{\tau}^{[j,l]},\ l = 1,\ldots,L$ and the corresponding rewards $R(\boldsymbol{\tau}^{[j,l]}, \boldsymbol{s}^{[j]})$ for this given context-parameter pair. To update the policy, we first minimize the dual function $g(\eta,\boldsymbol{\gamma})$ in Eq. (3.9) using the artificially generated samples and compute the new weight $p^{[j]}$ (Eq. 3.10) for each artificial sample. Note, that we use solely the artificial context-parameter samples $\{\boldsymbol{s}^{[j]},\boldsymbol{\omega}^{[j]}\}_{j=1}^{M}$ to update the policy. By doing so, we can avoid biased policy updates due to possibly noisy rollouts. Consequently, if the models produce unbiased rewards $\mathcal{R}_{\boldsymbol{s\omega}}$, the final policy will also be unbiased. Finally, we update the policy by the weighted maximum likelihood estimate using Eqs. (3.11-3.12). In the following, we will explain in more detail how to sample the trajectories and how to learn the models for accurate trajectory prediction.

## 4.2 Model Learning and Trajectory Prediction with GP Forward Models

In the previous section we presented GPREPS, a contextual and model-based policy search framework. We discussed how to generate artificial samples, how we can use the model prediction result $R(\boldsymbol{\tau},\boldsymbol{s})$ to obtain the expected reward $\mathcal{R}_{\boldsymbol{s\omega}}$ and how to update the policy using the artificial samples $\{\boldsymbol{s}^{[j]},\boldsymbol{\omega}^{[j]},\mathcal{R}_{\boldsymbol{s\omega}}^{[j]}\}_{j=1}^{M}$. In the following, we focus our attention on how GP models are learned and how we can use these models to obtain the reward $R(\boldsymbol{\tau},\boldsymbol{s})$.

We begin by considering the reward prediction problem as a function approximation task, where we wish to map $\boldsymbol{\omega}$ and $\boldsymbol{s}$ to the corresponding reward $R(\boldsymbol{\tau},\boldsymbol{s})$ (Fig. 4.1).

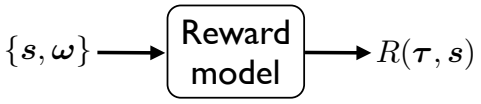



FIGURE 4.1: The reward prediction problem as a general function approximation task.

In this case, our goal is to find the function $h(\cdot)$, such that $R(\boldsymbol{\tau}, \boldsymbol{s}) = h(\boldsymbol{s}, \boldsymbol{\omega}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_R^2)$ is Gaussian noise. However, even for simple skill learning problems, $h(\cdot)$ is an unknown nonlinear function, without any knowledge about its structure and smoothness. While learning $h(\cdot)$ and evaluating predictions using the model can be executed relatively fast, due to the difficulty of the regression problem, the resulting predictions, and thus, the optimized policy might be biased.

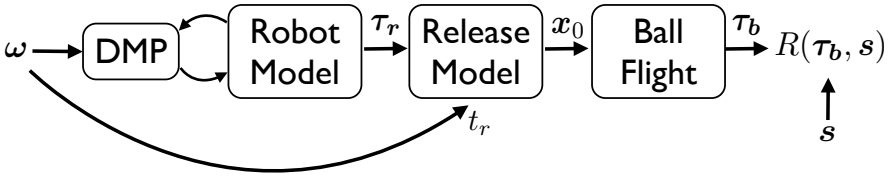



FIGURE 4.2: The reward prediction for the ball throwing task using decomposed models. First, we predict the robot trajectory $\boldsymbol{\tau}_r$ using the DMP and the robot model. Subsequently, we use the learned forward kinematics model to compute the angular position and velocity of the ball $\boldsymbol{x}_0$ at the release time $t_r$. Finally, we use the free dynamics model of the ball flight to provide the ball trajectory $\boldsymbol{\tau}_b$ to the reward function.

Instead, with GPREPS we use a computationally more involved, but intuitive approach. We exploit prior knowledge about the experiment and decompose $h(\cdot)$ into simpler models and their relationships. For example, in the ball throwing task, we know that the ball flight trajectory will depend on the robot joint trajectories, the forward kinematics and the release time of the ball. Furthermore, we know that the final reward will depend solely on the basket position and the ball trajectory (Fig. 4.2). Thus, we not only exploit the relationship between models, but the structure

of the known reward function as well. While learning dynamic models is relatively easy due to smoothness, models for discrete events, such as for contacts are typically more difficult to learn and requires more prior knowledge. Although the model decomposition approach requires the learning of multiple models, the learned models are typically more accurate and better at generalization compared to $h(\cdot)$. On the other hand, the prediction process is computationally more demanding and model learning might be corrupted due to missing measurement data, measurement error and noise.

Learning models for discrete events, such as the release model for the throwing task, is a one-step regression problem and predictions can be evaluated straightforwardly. However, performing long-term trajectory predictions, such as predicting the robot trajectory, is a more challenging task. Thus, in the following we will only focus on learning forward GP dynamic models and performing long-term predictions.

**The Model.** We use forward models to obtain the trajectory distribution $p(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s})$ given the context $\boldsymbol{s}$ and the lower-level policy parameters $\boldsymbol{\omega}$. We learn a forward model that is given by $\boldsymbol{y}_{t+1} = \boldsymbol{f}(\boldsymbol{y}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}$, where $\boldsymbol{y} = [\boldsymbol{x}^T, \boldsymbol{b}^T]^T$ is composed of the state of the robot and the state of the environment $\boldsymbol{b}$, for instance the position of a ball. The vector $\boldsymbol{\epsilon}$ denotes zero-mean Gaussian noise. In order to simplify the learning task, we decompose the forward model $\boldsymbol{f}$ into simpler models, which are easier to learn. To do so, we exploit prior structural knowledge of how the robot interacts with the environment.

### 4.2.1 Trajectory and Reward Prediction

Using the learned GP forward model, we need to predict the expected reward

$$\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau} \tag{4.2}$$

for a given parameter vector $\boldsymbol{\omega}$ evaluated in context $\boldsymbol{s}$. The expectation over the trajectories is now estimated using the learned forward models. For this purpose, we factorize the trajectory distribution as

$$p(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s}) = p(\boldsymbol{x}_0; \boldsymbol{s}) \prod_{t=0}^{T-1} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) \pi_{\boldsymbol{\omega}}(\boldsymbol{u}_t|\boldsymbol{x}_t), \tag{4.3}$$

where for some tasks $\boldsymbol{s}$ might influence the initial state distribution $p(\boldsymbol{x}_0; \boldsymbol{s})$.

To obtain the state distribution at each time step, we have to compute the GP predictive distribution

$$p(\boldsymbol{x}_{t+1}) = \iint_{\boldsymbol{x}_t, \boldsymbol{u}_t} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) p(\boldsymbol{x}_t, \boldsymbol{u}_t) d\boldsymbol{x}_t d\boldsymbol{u}_t, \tag{4.4}$$

with $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = p(\boldsymbol{x}_t)\pi_{\boldsymbol{\omega}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$. However, in case the current state $\boldsymbol{x}_t$ is Gaussian $\mathcal{N}([\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T|\boldsymbol{\mu}_{xu}, \boldsymbol{\Sigma}_{xu})$ and the model $f(\cdot)$ is nonlinear, the predictive distribution over the next state $p(\boldsymbol{x}_{t+1})$ becomes non-Gaussian. Thus, in general, we cannot obtain an analytic solution for $p(\boldsymbol{x}_{t+1})$. In Sec. 2.1.2 we discussed lower-level policy evaluation techniques to obtain the successor state distribution. In particular, we discussed deterministic evaluations, such as linearization and moment matching, and stochastic evaluation by sampling. We concluded that the deterministic

approach is beneficial for policy gradient algorithms, but for reward prediction we prefer the sampling approach.

As GPREPS uses information theoretic policy updates, we do not require the computation of policy gradients. Furthermore, we wish to avoid possibly biased trajectory and reward distributions when using approximations with, e.g., moment matching. Thus, we will rely on the stochastic evaluation of the integral in Equation (4.4), which provides unbiased estimates in the limit. When sampling trajectories, we first compute the control $\boldsymbol{u}_t = \pi_{\boldsymbol{\omega}}(\boldsymbol{x}_t)$ given $\boldsymbol{x}_t$, and subsequently, sample the next state from the predictive distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. We repeat this procedure until we obtain the complete trajectory (Sec. 2.1.2). When using sampling, the lower-level policy class is not restricted and torque limits can be applied with ease. On the other hand, to accurately approximate the trajectory and reward distribution, the number of sample trajectories $L$ must be relatively high and typically needs to increase with the dimensionality of the system. Note, that sampling multiple trajectories at the same time only consists of simple computations with large matrices, and thus, the computations can be executed in parallel. Hence, we can speed up computations significantly using high through-put processors, such as GPUs. This is particularly effective when evaluating multiple artificial samples with GPREPS. Such parallelization is not straightforward with the moment matching approach.

In the following, we evaluate the sampling approach for trajectory prediction and we compare it to the moment matching algorithm in terms of accuracy and computation time.

### 4.2.2 Quantitative Comparison of Sampling and Moment Matching

To compare moment matching and sampling, we evaluated both approaches on predicting trajectories with the 4-DoF planar robot in Fig. 3.1. The lower-level policy was given by a linear trajectory tracking PD controller. We used 12000 data points and 700 pseudo-inputs to train the sparse GP models [72]. The prediction horizon was set to 300 time steps.

We sampled 1000 trajectories and estimated a Gaussian state distribution $p(\boldsymbol{x}_t)$ for each time step from these samples. These distributions are used as ground truth. Subsequently, we compute the Kullback-Leibler divergence $\mathrm{KL}\big(p(\boldsymbol{x}_t)||\tilde{p}(\boldsymbol{x}_t)\big)$ of the approximations $\tilde{p}(\boldsymbol{x}_t)$ obtained by either using less samples or moment matching. This procedure was done for an increasing number of samples for the sampling approach. To improve the accuracy of the comparison, we evaluated the prediction for 100 independently chosen starting state with varying initial variance. To facilitate the comparison of the two prediction methods, we normalized the KL divergence values, such that the moment matching prediction accuracy remains constant, i.e., $\sum_{t=1}^{100} \mathrm{KL}\big(p(\boldsymbol{x}_t)||p_{MM}(\boldsymbol{x}_t)\big) = 100$, where $p_{MM}(\boldsymbol{x}_t)$ is the state distribution predicted by moment matching. The result is shown in Fig. 4.3. As the figure shows, the best 95% of the experiments required approximately 50 sample trajectories to reach the accuracy of the moment matching approach. However, in most cases (top 75%), it was enough to sample not more than 20 trajectories to reach the moment matching performance. The inaccuracies of the moment matching approach results from outliers and non-Gaussian state distributions which violate the Gaussian approximation assumption of moment matching.

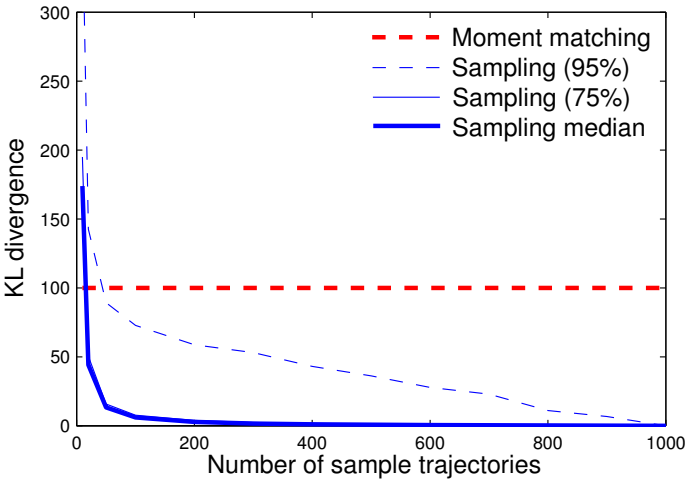



FIGURE 4.3: The sampling accuracy of sampling with an increasing number of samples. In most of our experiments (top 95%), the accuracy of moment matching is met by sampling only 50 samples per prediction. However, in most cases it was enough to sample 20 trajectories to reach the moment matching performance.

We also evaluated the computation time of both approaches. For the sampling approach, we evaluated the computation time for different types of hardwares and parallelization techniques. For CPU we chose Intel i5-3570k @3.4Ghz with both single and multi core setup. For GPUs we used Nvidia GTX 660Ti and Nvidia GTX Titan. As shown in Figure 4.4, already the single CPU core implementation outperforms moment matching and can produce approximately 1000 samples in the same computation time. This number can be increased to 7000 samples when using a high-end workstation graphics card.

We showed that only a few sample trajectories are needed to meet the accuracy of the moment matching approach while we are able to generate thousands of trajectories in the same computation time. Hence, using sampled trajectories results in a moderate speed-up for main-stream computers and can further be improved when using a GPU implementation. In addition to the improved computation speed, sampling

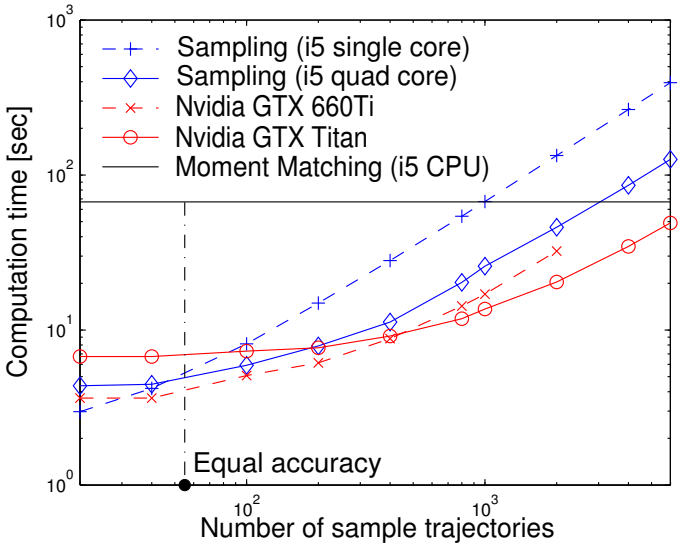



FIGURE 4.4: Comparison of the computation speed of moment matching and sampling-based long-term prediction. 50 sampled trajectories are needed to reach the accuracy of moment matching. Over 7000 samples can be created using a GPU implementation within the same computation time which is needed for the moment matching approach.

avoids the approximations involved in the moment matching approach, for example, using trigonometric functions to approximate torque limits [17]. Sampling produces unbiased estimates of the expected reward, and thus, we can improve the accuracy of the prediction by increasing the number of samples.

### 4.2.3 Comparison of Gaussian Process Models

When using GP models for trajectory prediction, we have to take the computation times into consideration. As discussed earlier, the training time of the hyper-parameters in the standard GP approach scales cubically with the number of training samples. The prediction time of the posterior mean scales linearly, while the computation of the posterior variance scales quadratically with the number of training samples. When learning dynamic models with high sampling rate, the number of training

samples can quickly increase to thousands, and thus, the computation time might become impractically large. To mitigate the computational demand while learning accurate models, sparse Gaussian process methods were proposed, e.g., by Snelson et al. [72] and by Titsias [85].

When learning GP forward models, numerical problems may emerge. In order to learn accurate models, we often need thousands of training samples. When using many training samples, the matrix $(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})$ used in the GP prediction and model training might have an overly high condition number. Thus, computing the inverse of this matrix might be inaccurate, which can easily lead to a biased trajectory prediction. To avoid this problem, a common strategy is to add noise to the training target data $w_i = w_i + \epsilon_{add}, i = 1, \ldots, N$, where $\epsilon_{add}$ is i.i.d. Gaussian noise. This will naturally increase the value of $\sigma_\epsilon$, and thus, decrease the condition number of $(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})$. In our experiments, we added i.i.d. Gaussian noise $\epsilon_{add} \sim \mathcal{N}(0, \sigma_{add}^2)$ to the training data with standard deviation $\sigma_{add} = 10^{-2}\text{std}(\boldsymbol{y})$, where $\text{std}(\boldsymbol{y})$ is the standard deviation of the target values. The amount of additive noise has proven to be efficient in balancing out the prediction accuracy and numerical instability.

In the following, we compare the accuracy of reward prediction of a control task when using the sparse GP method with pseudo inputs [72] and the standard GP approach [62]. We also tested the sparse method presented in [85], but ran into numerical problems with this method, which prevented a fair comparison with the other approaches. The control problem is to balance the planar 4-DoF robot to the upright position. The lower-level control policy is set such that the pendulum is robustly balanced to the optimal upright position from a random set of initial positions around

the upright position. We collect measurement data by executing a certain amount of experiment rollouts. Then, we train both GP models, the standard and the sparse model with the same measurement data. Subsequently, we use the GP models to predict the reward of 50 context-parameter pairs with trajectories of 20 time steps long. We use 20 trajectories per context-parameter pair. The reward of a single trajectory was given by $R(\hat{\boldsymbol{\tau}}) = -\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{x}_r)^T(\boldsymbol{x}_t - \boldsymbol{x}_r)$, where $\boldsymbol{x}_r$ is the upright position. Finally, we measure the accuracy of the GP models, by computing the average quadratic error of the mean reward prediction $\mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega}}[e^2] = \mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega}}[(\hat{R}(\boldsymbol{s},\boldsymbol{\omega}) - R(\boldsymbol{s},\boldsymbol{\omega}))^2]$, where $\hat{R}(\boldsymbol{s},\boldsymbol{\omega})$ denotes the mean predicted reward and $R(\boldsymbol{s},\boldsymbol{\omega})$ the real reward for that context parameter pair. We compare the models by changing several parameters of the training process. First, we investigate the influence of the amount of additive noise on the prediction performance. We set the additive noise to $\sigma_{add} = \alpha 10^{-2}\text{std}(\boldsymbol{w})$, where $\alpha$ is a scalar. Second, we investigate the amount of hyper-parameter optimization steps required to learn accurate models. In the third experiment, we evaluate how well the models can capture the stochasticity in the dynamics by adding noise to the control input. Finally, we investigate how well the models can generalize with only a limited amount of training data. For the first three experiments we use 50 sample trajectories to learn the model while we varied the number of sample trajectories in the fourth experiment.

In each experiment we only vary one parameter and keep the remaining parameters at their optimal value. We set the optimal values such that that the GP models provide the best prediction performance. In particular, we have chosen the standard deviation of the additive noise value as $\sigma_{add} = 10^{-2}\text{std}(\boldsymbol{w})$, that is, $\alpha = 1$. We optimized

the hyper-parameters of the models for 150 optimization steps and we assumed 0.5 Nm standard deviation for additive torque noise. We used 50 observed trajectories for training, that is, a total of 1000 training data points. For the sparse method, we used 25% of the observed data points as pseudo inputs, that is, $M = N/4$.

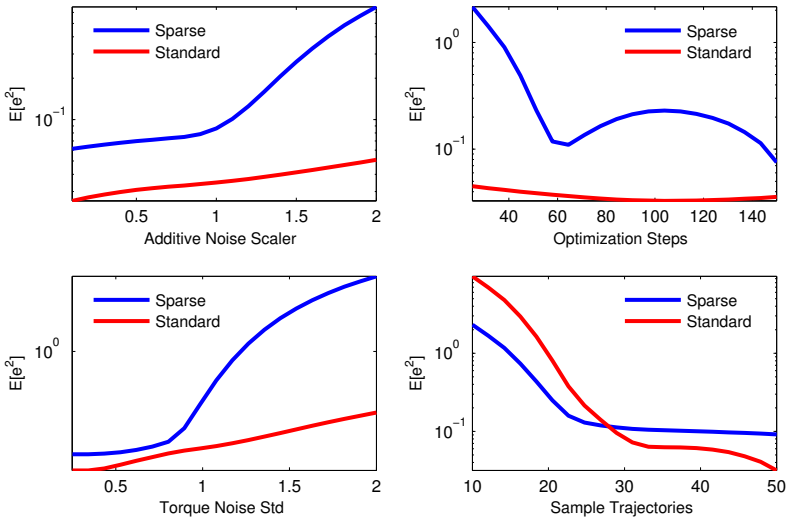



FIGURE 4.5: Comparison of the standard and the sparse GP approach.

The results of the model comparison tasks are shown in Fig. 4.5. With increasing additive noise factor we gain more numerical stability, but the accuracy of reward prediction decreases slightly. However, we observed that the sparse method often overfits the data, which results in the worse performance with higher additive noise factor. When comparing the amount of hyper-parameter optimization steps, we can conclude that after only 50 optimization steps we can already obtain accurate models. However, we also see a small overfitting effect for the sparse models as we continue the optimization. When we add additional control input noise to the system, the standard GP approach could capture the uncertainty well. However, just as with

the additive noise experiment, the sparse method tends to overfit the data and produces inaccurate predictions. Finally, an increasing number of sample trajectories clearly has a positive effect on the prediction accuracy. However, the training time steeply increases with a higher amount of training data.

### 4.2.4 Learning the Hyper-Parameters of GP Models

In Section 2.3 we discussed the basics of Gaussian Process regression and model selection, which includes choosing a kernel function and optimizing its hyper parameters based on the observed data. The typical approach is to set the hyper-parameters of the kernel is by maximizing the likelihood of training targets (Eq. (2.28)). For learning the forward dynamics of the robot, we use the GP model to map states and controls $[\boldsymbol{x}^T, \boldsymbol{u}^T]^T$ to subsequent states $\boldsymbol{x}'$. Thus, maximizing the likelihood of training targets will translate to maximizing the likelihood of observed state transitions. However, we wish to use the GP forward models to obtain the distribution over trajectories, that is, performing multi-step predictions. Thus, when we use the standard training technique to set the kernel hyper-parameters, we maximize the likelihood of state transitions, but we might fail to maximize the likelihood of observed trajectories, and thus, trajectory predictions can be biased.

Abbeel et al. [1] proposed an algorithm for learning the linear forward dynamics of vehicular systems, such as autonomous helicopters, to minimize the long term trajectory prediction error. The proposed algorithm directly minimizes the squared error between the model predicted and observed trajectories. While evaluating the prediction for deterministic models is straightforward, for probabilistic models we

often have to use approximations and assumptions to evaluate the prediction (see Sec. 2.1.2 for more details). Similar ideas were exploited in [2] to learn first order Markov models for more accurate transition probability approximation in an MDP framework.

In the following, we propose a novel model training algorithm for setting the parameters of probabilistic forward models to directly maximize the likelihood of observed trajectories. The proposed method is more general compared to previous work **??** as it is directly applicable to probabilistic models and it is applicable to continuous state spaces as opposed to [2]. Instead of evaluating rollouts with the GP model and minimizing the divergence to the observed trajectories, we will treat the marginal state distributions as parameters of the learning problem, and we introduce constraints that connects the time steps. While we introduce additional Lagrangian parameters into the learning problem, the resulting optimization problem becomes convex in these parameters. The benefit we gain from solving the dual problem instead of the primal is that we avoid long term predictions with probabilistic models.

We use probabilistic forward models to predict the trajectory distribution of the robot $p(\boldsymbol{\tau})$. The robot transits to state $\boldsymbol{x}'$ given the current state $\boldsymbol{x}$ and the control $\boldsymbol{u}$ according to $p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})$. The control signal $\boldsymbol{u}$ is computed according to the parametrized deterministic $\boldsymbol{u} = \pi(\boldsymbol{x})$, or stochastic control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$, where we omit highlighting the control policy parametrization $\boldsymbol{\omega}$. Using the Markovian assumption, we factorize the trajectory distribution as

$$p(\boldsymbol{\tau}) = p_1(\boldsymbol{x}) \prod_{t=2}^{T} p_t(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\pi(\boldsymbol{u}|\boldsymbol{x}).$$

Our goal is to find the model $p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta})$ with hyper-parameters $\boldsymbol{\theta}$, that minimizes the KL-divergence $KL(p(\boldsymbol{\tau};\boldsymbol{\theta})||q(\boldsymbol{\tau}))$ between the observed $q(\boldsymbol{\tau})$ and the predicted $p(\boldsymbol{\tau};\boldsymbol{\theta})$ trajectory distributions. Using the factorized model of the trajectory distribution, we can reformulate our objectives as minimizing the sum of KL divergences between the marginals

$$KL(p(\boldsymbol{\tau};\boldsymbol{\theta})||q(\boldsymbol{\tau})) = \sum_{t=1}^{T} KL(p_t(\boldsymbol{x};\boldsymbol{\theta})||q_t(\boldsymbol{x})). \tag{4.5}$$

Thus, the optimization problem can be written as

$$\min_{p_t,\boldsymbol{\theta}} \sum_{t=1}^{T} KL(p_t(\boldsymbol{x};\boldsymbol{\theta})||q_t(\boldsymbol{x})). \tag{4.6}$$

However, as we are optimizing over $p_t(\boldsymbol{x})$ as well, we have to ensure that the distribution is consistent with the system dynamics, that is, we have to add the constraint

$$\forall\, t,\ p_t(\boldsymbol{x}) = \iint_{\boldsymbol{xu}} p_{t-1}(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})d\boldsymbol{x}d\boldsymbol{u}. \tag{4.7}$$

On the other hand, in case of continuous state-control variables we have infinite constraints to satisfy. To keep the problem tractable, we require only to match feature averages instead of single probability values

$$\forall\, t,\ \int_{\boldsymbol{x}} p_t(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \iint_{\boldsymbol{xu}} p_{t-1}(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x}) \int_{\boldsymbol{x}'} p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\boldsymbol{\phi}(\boldsymbol{x}')d\boldsymbol{x}'d\boldsymbol{x}d\boldsymbol{u}, \tag{4.8}$$

where $\boldsymbol{\phi}(\boldsymbol{x})$ is the feature vector of the state $\boldsymbol{x}$. Furthermore, we require the initial expected state feature to match the observed initial average feature

$$\int_{\boldsymbol{x}} p_1(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \hat{\boldsymbol{\phi}}(\boldsymbol{x}). \tag{4.9}$$

The optimization problem is now

$$\min_{p_t,\boldsymbol{\theta}} \quad \sum_{t=1}^{T}\int_{\boldsymbol{x}} p_t(\boldsymbol{x})\log\frac{p_t(\boldsymbol{x})}{q_t(\boldsymbol{x})}d\boldsymbol{x}, \tag{4.10}$$

$$s.t.: \quad \int_{\boldsymbol{x}} p_1(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \hat{\boldsymbol{\phi}}(\boldsymbol{x}), \tag{4.11}$$

$$\int_{\boldsymbol{x}} p_t(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \int_{\boldsymbol{x}} p_{t-1}(\boldsymbol{x})\int_{\boldsymbol{u}} \pi(\boldsymbol{u}|\boldsymbol{x})\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]d\boldsymbol{u}d\boldsymbol{x},\ \forall t>1, \tag{4.12}$$

$$\int_{\boldsymbol{x}} p_t(\boldsymbol{x})d\boldsymbol{x} = 1,\ \forall t. \tag{4.13}$$

The constrained optimization problem can be solved with the Lagrange multiplier method. As the result of the optimization we obtain the state distribution at each time step in the form of

$$p_t(\boldsymbol{x}) \propto q_t(\boldsymbol{x})\exp\left(\boldsymbol{\gamma}_t^T\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\gamma}_{t+1}^T\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]\right), \tag{4.14}$$

where $\boldsymbol{\gamma}_t$ are Lagrange multipliers and $\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]$ is the expected feature of the subsequent state with model $p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta})$ and parametrization $\boldsymbol{\theta}$. Note that the exponential term will depend both on the current $\boldsymbol{\gamma}_t$ and the subsequent Lagrangian parameters $\boldsymbol{\gamma}_{t+1}$. Thus, the resulting algorithm will provide model parametrization that aims to reduce the long-term prediction error instead of greedily maximizing the likelihood of individual state transitions. To obtain the Lagrangian and the model parameters,

we need to minimize the dual function

$$g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta}) = -\sum_{t=1}^{T} \log \int_{\boldsymbol{x}} q_t(\boldsymbol{x}) \exp\left(\boldsymbol{\gamma}_t^T \boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\gamma}_{t+1}^T \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}'); \boldsymbol{\theta}]\right) d\boldsymbol{x} + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}). \tag{4.15}$$

When working with samples, the dual function can be approximated as

$$g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta}) = -\sum_{t=1}^{T} \log \frac{1}{M} \sum_{i=1}^{M} \exp\left(\boldsymbol{\gamma}_t^T \boldsymbol{\phi}(\boldsymbol{x}_i) - \boldsymbol{\gamma}_{t+1}^T \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}_i'); \boldsymbol{\theta}]\right) + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}), \tag{4.16}$$

where $M$ is the number of observed trajectories. The lengthy derivation of the gradients is presented is Appendix B.

The above parameter learning algorithm is applicable for a wide range of probabilistic models. One of the most important advantage of our approach is that we avoid long-term trajectory predictions, which might be inaccurate and time consuming. Instead, we treat the marginal distributions $p_t(\boldsymbol{x})$ as parameters of the learning problem and we connect the subsequent time steps by enforcing feature matching constraints. Thus, we maximize the likelihood of observed trajectories instead of state transitions. One major disadvantage of this approach however, is that the feature matching constraints must be satisfied for each time step, which might be infeasible for some modeling techniques. For example, for GP models these constraints are often difficult to satisfy due to the fixed set of training data and the kernel function. Due to these challenges, in our experiments we failed to apply the above training approach with success for GP models. However, we believe that the proposed method opens the door to novel GP methods, such as sparse GP models where the training

data could also be represented as model parameters. Future research will also investigate applying the learning algorithm presented in [1] with sample trajectories obtained using the GP model.

## 4.3 Results

In this section, we present the results of applying GPREPS for robot learning tasks. First, we compare GPREPS to PILCO in a context-free learning task. We show that while PILCO achieves better data-efficiency in learning the task, GPREPS provides almost equally good performance. Subsequently, we evaluate the same contextual robot learning tasks as with contextual model-free REPS. Additionally, we will present real robot results.

### 4.3.1 Robot Balancing Task

In this task, the goal is to find a PD controller that balances the 4-DoF planar robot (Fig. 3.1) around the upright position. The robot has a total length of 2m and a total mass of 70kg. For more details about the robot we refer to Appendix C.1. The reward for a sample trajectory is the sum of quadratic rewards along the trajectory $R(\boldsymbol{\tau}, \boldsymbol{s}) = -\sum_t^T \tilde{\boldsymbol{x}}_t^T \boldsymbol{Q} \tilde{\boldsymbol{x}}_t$, where $\tilde{\boldsymbol{x}}_t$ is the deviation from the upright position at time $t$. We chose the initial state distribution around the upright position to be Gaussian. As PILCO cannot learn contextual policies, we learn context-free upper-level policies with REPS and GPREPS to have a fair comparison. Hence, the upper-level policy is given by a Gaussian, $\pi(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The lower-level controller is represented as a

PD controller $\boldsymbol{u}_t = \boldsymbol{G}[\tilde{\boldsymbol{x}}_t^T, \dot{\tilde{\boldsymbol{x}}}_t^T]^T$. The gain matrix $\boldsymbol{G}_{4\times 8}$ is obtained by reshaping the parameter vector $\boldsymbol{\omega}_{32\times 1}$. We initialize the models of the model-based approaches with 6 seconds of real experience, which was collected by a random policy. We use sparse GP models [72] for PILCO and GPREPS.

| | Required Experience | | |
|---|---|---|---|
| Reward limit | PILCO | GPREPS | REPS |
| -100 | 10.18s | 10.68s | 1425s |
| -10 | 11.46s | 20.52s | 2300s |
| -1.5 | 12.18s | 38.50s | 4075s |

TABLE 4.2: Required experience to achieve reward limits for a 4-link balancing problem. Higher rewards require better policies.

In Tab. 4.2 we show the required amount of real experience to reach certain reward limits. As shown in the table, GPREPS requires two orders of magnitude fewer trials than REPS to converge to the optimal solution. PILCO achieved faster convergence as the gradient-based optimizer computes greedy updates while GPREPS continued to explore. The difference to PILCO might be decreased by updating the policy more than once between the data collections. However, the difference is negligible compared to the difference to the model-free method. Despite the fact that PILCO was the most data-efficient learning algorithm, it required at least twice as much computation time to achieve the same reward levels as GPREPS, which was highly parallelized. The data efficiency of GP model-based policy search algorithms comes with significantly higher computational demands compared to model-free approaches. The computation time is dominated by the GP model training and the trajectory prediction.

### 4.3.2 Ball Throwing Task

In the ball throwing task we use a 4-DoF freedom planar robot to throw a ball at a target, whose position is defined by the context variable (Appendix C.1). In Sec. 3.4.1 we evaluated the contextual model-free REPS on this task. The resulting final policy provides skills that throws the ball in the close vicinity of the target position for a wide range of contexts. However, due to data-inefficiency, model-free REPS requires up to 5000 evaluations to converge to a high quality solution.

With GPREPS our goal is to evaluate artificial policy parameter samples using the learned models to obtain the reward. To produce trajectory rollouts with the model-based GPREPS, we learn three distinct models of the task (Fig. 4.2). The first model represents the dynamics of the robot. We use the observed state transitions as training samples for this model. To reduce the computational demand, we use a sparse GP model based on pseudo inputs [72]. The second model is used to predict the initial position and velocity of the ball at the release time $t_r$. Although we could use the forward kinematic equations to approximate the initial state of the ball, in a real learning scenario this could lead to biased predictions. Instead, we use a standard GP model to predict the initial state of the ball based on our observations. The third GP model represents the free dynamics of the ball while in flight. It is used to predict the trajectory of the ball using its initial state predicted by the second GP model. While we could also use a mathematical model to predict the ball trajectory, to account for stochasticity and nonlinearities, such as drag, we use sparse GP models.

The learning curves of the ball throwing task are presented in Figure 4.6. For GPREPS

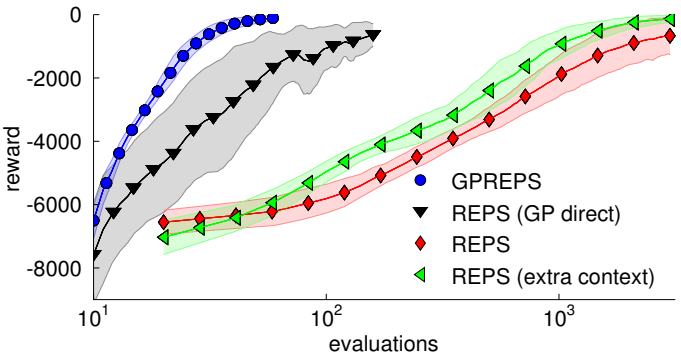



FIGURE 4.6: Learning curves for the ball throwing problem.

we initially evaluate 10 experiments to collect measurement data for the GP models. Subsequently, after each policy update we execute only one experiment to collect data using the new policy. For each policy update, we use 500 artificial sample evaluations and for each sample we evaluate 25 trajectories. As we optimize the dual function with significantly more samples, we obtain a better estimate of the Lagrangian parameters $\eta$ and $\boldsymbol{\theta}$. Thus, we can update the policy more confidently using $\epsilon = 1$. However, a higher $\epsilon$ can also be harmful if the predicted rewards, and thus, the parameters $\eta$ and $\boldsymbol{\theta}$ are biased. Based on our observations, our modeling technique provided reward predictions with absolute error below 10%.

We also evaluated a model-based version of contextual REPS, where GP models were used to map artificial contexts and policy parameters to expected rewards (denoted by *GP direct* in the figure). While we observed relatively good learning performance, the final result is of lower quality and has higher variance compared to REPS and GPREPS. On the other hand, GPREPS required only $30-40$ evaluations to converge to the highest quality solution. Furthermore, the variance of the final policy is negligible. Thus, this experiment confirms that incorporating prior knowledge about the

task and decomposing the reward model to multiple modules consistently provides higher quality final policies. GPREPS not only provides higher quality solutions compared to model-free REPS, but also requires two orders of magnitude less interactions with the robot.

#### 4.3.2.1 Influence of the Number of Artificial Samples

In the following, we investigate the influence of the number of artificial samples on the learning performance. To obtain an accurate estimate of the sample-based dual function (Eq. (3.9)), we are interested in evaluating a high number of artificial samples. However, the computation time between policy updates linearly increases with the number of artificial samples generated by the GP models. While a long policy update interval does not influence the data efficiency of GPREPS, from a practical point of view we prefer to keep the number of artificial samples as low as possible, without affecting the learning performance.

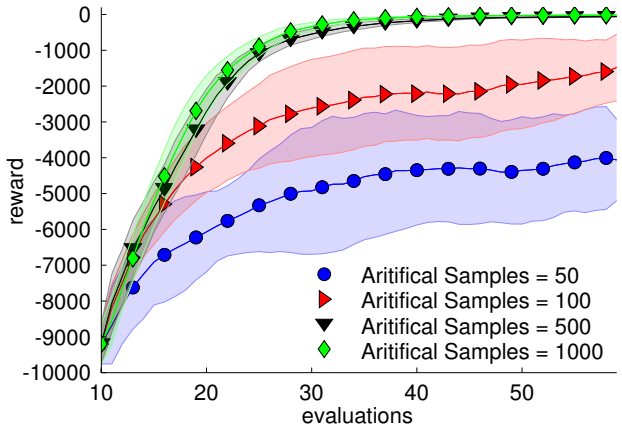



FIGURE 4.7: The learning curves of GPREPS with different amount of artificial samples used.

In Fig. 4.7 we show the learning curves with GPREPS when using different amount of artificial samples. As the figure shows, increasing the amount of artificial samples always has a positive influence on the learning performance. However, we do not see much difference above using 500 samples. Using a lower number of artificial samples resulted in lower quality final policies with highly varying performance. Based on our experience with the learning problems, the number of artificial samples should scale proportionally with the dimensions of the lower-level policy parameter for good performance.

#### 4.3.2.2 Learning with Stochastic Dynamics

For learning problems where the dynamics of the robot and its environment is stochastic, the variance of the resulting trajectory $\boldsymbol{\tau}$ and hence the reward $R(\boldsymbol{\tau}, \boldsymbol{s})$ might be considerably large. We show such an example in Fig. 4.8 with a single link pendulum. As already mentioned, the model-free REPS algorithm only evaluates a single rollout $R(\boldsymbol{\tau}, \boldsymbol{s})$ for a given context-parameter pair to obtain an estimate of the expected reward $\mathcal{R}_{\boldsymbol{s\omega}}$. Thus, the dual function in Eq. (3.9) and the optimized parameters $\eta$ and $\boldsymbol{\theta}$ might be biased. This could lead to premature convergence and biased final policies. GPREPS avoids this problem by averaging over multiple samples for the same context-parameter pair, as in Eq. (4.1).

To test the GP models, we implement the stochasticity by adding Gaussian noise to the initial state of the ball at the release time. The standard deviation of the initial positions is 7.5cm, while for the initial velocities it is 30cm/s. With the added noise,

the standard deviation of the reward $R(\boldsymbol{\tau}, \boldsymbol{s})$ will be roughly 10% of the absolute expected reward, that is $\text{Var}[R(\boldsymbol{\tau}, \boldsymbol{s})] \approx \mathcal{R}_{\boldsymbol{s\omega}}/10$. Thus, in the beginning of the learning the standard deviation is around 1000, while for the optimal policy it is below 10.

In Figs. 4.9(a) and 4.9(b), we can see the learning curves of REPS and GPREPS with

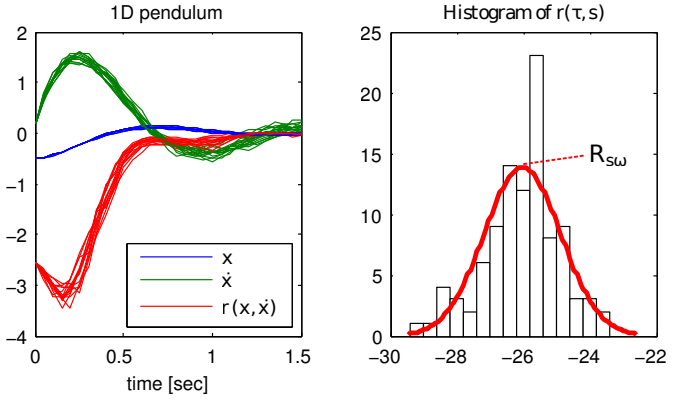
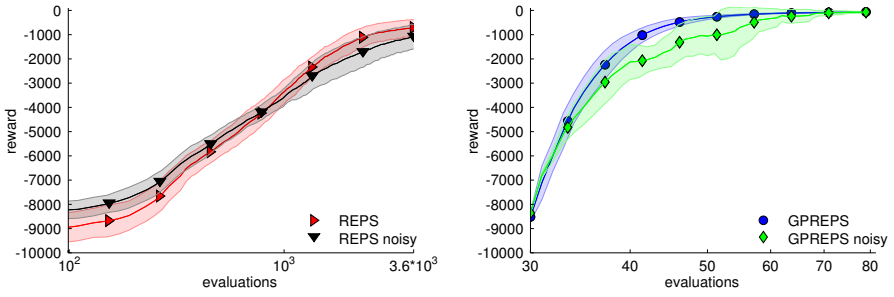



FIGURE 4.8: **(left)** Trajectory and reward samples of the single link pendulum. **(right)** The trajectory reward $r(\boldsymbol{\tau}, \boldsymbol{s}) = \sum_t r(x_t, \dot{x}_t)$ distribution and the expected reward $\mathcal{R}_{\boldsymbol{s\omega}} = \mathbb{E}[r(\boldsymbol{\tau}, \boldsymbol{s})]$.




(a) (b)

FIGURE 4.9: **(a)** The learning curves of REPS with the stochastic ball throwing task. At the end of the learning, the policy optimized with the noisy environment attains an expected reward of −1086, while the policy learned with the deterministic dynamics has a final reward of −715. **(b)** The learning curves of GPREPS with the stochastic ball throwing task. At the end of the learning, the policy optimized with the noisy environment attains an expected reward of −68, while the policy learned with the deterministic dynamics has a final reward of −64.

stochastic dynamics (denoted by *noisy* in the Figures). As a reference, we also displayed the learning curves with the deterministic dynamics. We can see that learning with stochastic dynamics resulted in a slower learning pace for both algorithms. However, we can see significant differences in the performance of the final policy. While GPREPS is able to learn a policy of similar quality with stochastic dynamics as with deterministic, the bias in model-free REPS prevents the algorithm from finding the final policy of similar quality. The performance of REPS could have been improved when using multiple rollouts for a given context-parameter pair. However, such strategy decreases the sampling efficiency significantly. This experiment confirms that by using the expected reward instead of a single noisy reward sample, we can learn high quality policies, even in the presence of stochasticity.

### 4.3.3 Robot Hockey Task

In the robot hockey task our goal is to shoot a target puck at a specified distance, which can be achieved by executing a hitting stroke with a 7-DoF robot arm (Appendix C.2). The learning problem becomes non-trivial, as the position and the required displacement of the target puck changes between task executions.

We have already evaluated the model-free contextual REPS and the CrKR algorithm on this learning problem in Sec. 3.4.2. While the final learned policy was of high quality, the amount of required task executions was in the thousands. To account for the data-inefficiency, we propose the use of GPREPS.

Using prior knowledge, we decompose the experiment into distinct models. Modeling the contact of the racket and the control puck is challenging due to the curved

shape of the hockey racket. When shooting the puck, the racket might push, or hit the puck multiple times. To avoid extensive modeling of these contacts, we use a GP model to directly predict the state of the puck at a constant distance of 0.2m in the $x$ direction from the robot's base, where contact between the racket and the puck is no longer possible. We use solely the DMP parameters $\boldsymbol{\omega}$ as the input to this model. We learn another model for the free dynamics of the sliding pucks, which are used for predicting the puck trajectories. For predicting the contact of the control and target pucks, we assume that we know the radius of the pucks, and thus, we can always detect contacts. For modeling the effect of the contact, we also learn a separate GP model that predicts the state of the pucks after a contact.

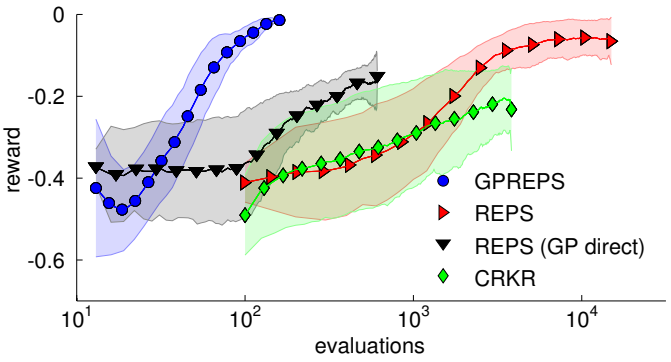



FIGURE 4.10: Learning curves on the robot hockey task. GPREPS was able to learn the task within 120 interactions with the environment.

The learning performance of GPREPS is shown in Figure 4.10. GPREPS learned the task already after 120 interactions with the environment, while the model-free version of REPS needed approximately 10,000 interactions. Moreover, the policies learnt by model-free REPS were of lower quality. After 100 evaluations, GPREPS placed the target puck accurately at the desired distance with a displacement error less than 5 cm.

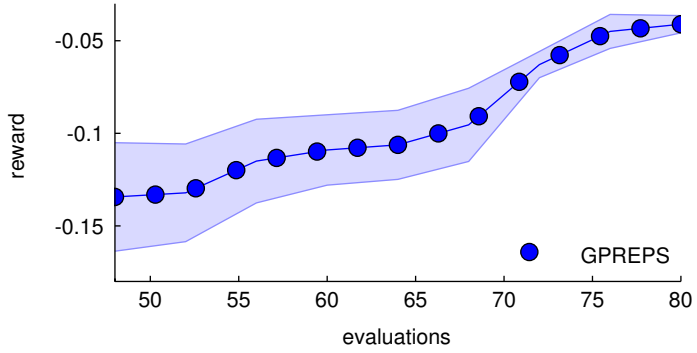



FIGURE 4.11: The GPREPS learning curve on the real robot arm. The shaded area represents the 95% confidence interval of the result after 5 independent evaluations.

We also evaluated the performance of GPREPS on the hockey task using a real KUKA lightweight arm. Due to environmental constraints, we use a slightly different test setup compared to the original task used for the model-free REPS. Details of the new problem setup is discussed in Appendix C.2. Due to the relatively low sampling frequency of the Kinect sensor, the contact between pucks might not be captured accurately. To avoid the errors coming from a crude model, we exchange the contact model by a model that directly predicts the displacement of the second puck using the incoming puck's relative position and velocity.

The resulting learning curve of GPREPS is shown in Fig. 4.11. As we can see, the robot adapts the lower-level policy parameters towards the optimum within a low number of interactions with the real environment. The final reward is slightly different compared to the simulated environment due to the altered context ranges and slightly distorted measurement data of the Kinect sensor. Despite these effects, the GP models could average over the uncertainty and produce accurate predictions of the expected rewards.

### 4.3.4 Robot Table Tennis

For the robot table tennis task (see Appendix C.3), we decompose the whole experiment into five distinct models. With the first model, we predict the landing position, landing velocity and the landing time of the incoming ball using the observed initial velocities $\boldsymbol{v}$ of the ball. This modeling approach is sufficient for our objectives, as we only want to learn returning incoming balls that land exactly once on the table. The second model predicts the trajectory of the ball given its position and velocity predicted by the first model. The third and fourth model predict the trajectory and orientation of the racket mounted at the endeffector. To avoid the complex modeling of the 8-DOF robot dynamics, we use time dependent GP models to directly predict the position and orientation (in the quaternion representation) from policy parameters. To create time dependent models, we fit a linear basis function model with 40 local basis functions $\boldsymbol{\phi}(t)$ per dimension to the trajectory of the racket, where the basis functions only depend on the execution time of the trajectory. We use the GP model to predict the weights of the basis functions given the policy parameters $\boldsymbol{\omega}$. The training input for each model is the lower-level policy parameter $\boldsymbol{\omega}$, the training target is the local model weight $v$. Finally, the fifth model predicts the landing position of the returned ball in case of a contact, which is detected by a Support Vector Machine (SVM) classifier with a linear kernel. The input to the classifier is the relative velocity of the ball and the racket, the position and orientation of the racket at the time when the absolute distance between the racket and the ball is minimal. For the contact model, we use the same training input data as for the classifier and the observed landing position $\boldsymbol{p} = [p_x, p_y]$ as the training target.

As learning a good contact model between the racket and the ball requires many samples, in the first few iterations we only have to focus on learning to hit the incoming ball. As soon as we learned a good hitting stroke and we have enough contact samples, we can use the learned contact model to provide confident predictions. Thus, we change the weighting parameters $\boldsymbol{c} = [c_1, c_2]^T$ in the reward function

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -c_1 \min ||\boldsymbol{\tau}_b - \boldsymbol{\tau}_r||_2 - c_2 ||\boldsymbol{b} - \boldsymbol{p}||_2 \tag{4.17}$$

such that at the beginning of the learning $c_2$ is negligible compared to $c_1$, but we add an extra constant penalty term. The size of the penalty term is the maximum possible penalty $\max ||\boldsymbol{b} - \boldsymbol{p}||_2$. Using this choice for the weight parameters, the algorithm focuses only on learning to hit the ball. After collecting enough samples to learn a good contact model, we set $c_1 = c_2$ and disable the constant penalty term. Now, the algorithm focuses both on hitting the ball and returning it close to the target position $\boldsymbol{b}$. Note, that we always use $c_1 = c_2$ for the model-free algorithms.

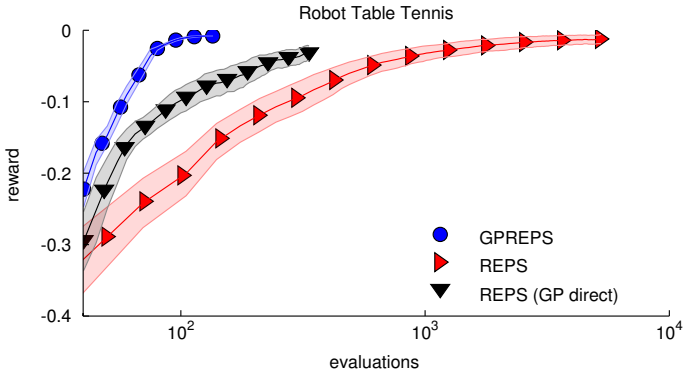



FIGURE 4.12: The learning curves of the table tennis experiment. GPREPS always provided a consistent performance, and the final policy was able to return the ball within 30cm of the target position, while avoiding hitting the ball into the net. This behavior could be learned within 150 evaluations.

We compared GPREPS with the model-free REPS and a model-based REPS, where we

directly predict the reward from context-policy parameter pairs. The learning curves are depicted in Fig. 4.12. We can clearly see that GPREPS outperforms the other two algorithms. GPREPS consistently provides high quality policies after only 150 evaluations. The final policy avoids hitting the ball into the net and the displacement from the desired target position remains below 30cm. In Figure 4.13 we show an experiment evaluation using the GP models.

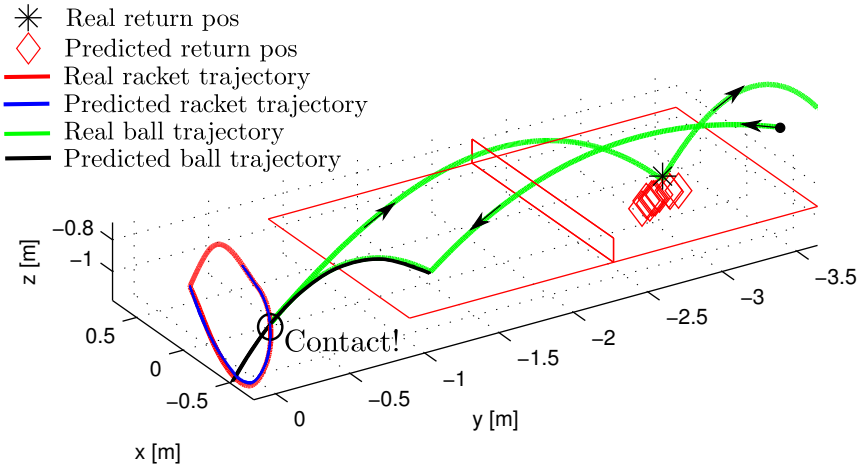



FIGURE 4.13: An example of prediction outcome with the table tennis experiment. The first and second model predicts the initial position and velocity of the incoming ball and its trajectory after bouncing back (**black** lines). The third and fourth model predicts the position (**blue** lines) and the orientation (not depicted here) of the racket. After detecting a contact, we predict the returned position of the ball (**red** diamonds). When we compare the predictions with the real experiment trajectories (**red** and **green** lines), we can see the high accuracy of the predictions. The models are learned from 100 experiment rollouts, we sample 10 trajectories to capture the stochasticity.

When using the *GP direct* approach that does not use the prior knowledge of the structure of the experiment, we obtain policies that often get stuck in a local optima. Typically we observe policies that hit the ball to the net, or even miss the ball. The model-free REPS approach results in a good performance in general, but with the

disadvantage of being data-inefficient. Model-free REPS requires at least 4000 evaluations on average to learn a policy that consistently returns the ball, with only a few instances of hitting the ball into the net.

This experiment concludes that GPREPS is applicable to learn complex robotic tasks, even in the presence of contact models that are, in general, more difficult to learn. In future work, we will investigate how we can use a GP model-based version HiREPS [14], to learn not only forehand, but backhand hitting motion as well. Furthermore, in order to learn to play a general table tennis game, we will extend the context with varying initial positions and we will evaluate GPREPS on the real robot platform.

## 4.4 Discussion

We have presented in four simulations and in one real robot experiment that GPREPS not only improves the data-efficiency, but the quality of the final policy. Due to the probabilistic modeling approach, GPREPS is able to learn high quality policies even in the presence of uncertainty. While GPREPS reduces the required amount of skill evaluations up to two orders of magnitude, the policy update times significantly increase due to computational demand for training and predicting with non-parametric GP models. While sparse GP models help to mitigate the problem, the expected time between policy updates can take up to half an hour. However, with clever implementation, e.g., sampling trajectories on GPUs, the policy update times can be reduced below a minute using main-stream PCs.

In order to make GPREPS work well on the real task, we have to introduce a substantial amount of prior knowledge about the experiment. It is often not clear which models we can learn accurately with GP regression. For example, in the table tennis task, instead of learning the robot dynamics and its forward kinematics to predict the trajectory of the endeffector, we used locally weighted GP regression to directly predict the position and orientation over time from DMP parameters. While the former approach is more intuitive, the models become high dimensional and prediction errors can be high. In the latter case, although the choice of model is not intuitive, predictions become highly accurate using GP models. Thus, finding the proper way of task decomposition might require longer experimentation times.

GP models are typically good in capturing the nonlinearity and the stochasticity in the dynamics at the cost of a relatively high computational demand. While rather good models can be learned in case of smooth dynamics, modeling abrupt changes and discrete events (e.g., contacts) is generally harder and it requires more prior knowledge.

Despite the above mentioned implementation issues, GPREPS offers a general contextual policy search framework for robot learning tasks with unprecedented data-efficiency. Furthermore, GPREPS can be considered as an example of a more general model-based contextual learning framework, where we used GP modeling and contextual REPS learning. As the purpose of modeling is solely to obtain unbiased rewards, the GP model can be exchanged to other models, such as LWBR. Similarly, other learning algorithms can also be augmented with GP modeling to obtain less biased reward prediction and better data efficiency. We believe that the ideas behind

GPREPS will open the door for novel algorithms for data-efficient robot skill generalization.

# Chapter 5

# Kernel Embedding of Trajectory Distributions

Model-based reinforcement learning of robot skills inherently depends on the hand-tuned or learned model of the robot dynamics. One of the most successful class of models for learning the complex dynamics of higher DoF robots is Gaussian Processes. The state of the art model-based controller learning algorithm PILCO [17], as well as the GPREPS algorithm [42, 48] uses GP models to learn the forward dynamics of the controllable system. While GP regression provides less biased models, as discussed in Sections 2.3 and 4, evaluating long-term predictions requires approximations and assumptions. When predicting trajectories, at each time step we wish to obtain the subsequent state distribution $p_{t+1}(\boldsymbol{x})$ given the current state distribution $p_t(\boldsymbol{x})$ and some control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$. To obtain the new state distribution, we have

to solve the integral

$$p_{t+1}(\boldsymbol{x}) = \iint_{\boldsymbol{x}\boldsymbol{u}} p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\pi(\boldsymbol{u}|\boldsymbol{x})p_t(\boldsymbol{x})d\boldsymbol{x}d\boldsymbol{u}. \tag{5.1}$$

To obtain $p_{t+1}(\boldsymbol{x})$ in closed form while keeping computations tractable, we typically make the assumption that $p_{t+1}(\boldsymbol{x})$ and the joint distribution $p_t(\boldsymbol{x},\boldsymbol{u}) = \pi(\boldsymbol{u}|\boldsymbol{x})p_t(\boldsymbol{x})$ are Gaussian. Furthermore, to solve the integral, we have to rely on the moment matching or the linearization approach (Sec. 2.3), as for nonlinear models the predictive distribution is not Gaussian. Alternatively, we can sample from distribution $p_t(\boldsymbol{x},\boldsymbol{u})$ and obtain a stochastic approximation of the posterior distribution, which is unbiased in the limit. On the other hand, this approach is computationally demanding.

For most model-based reinforcement learning algorithms, we use the predicted trajectory distribution to obtain the expected reward $\mathcal{R}(\boldsymbol{\tau}) = \int p(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau}$. For many robot learning tasks, the trajectory reward can be defined as the sum of immediate rewards $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} r(\boldsymbol{x}_t,\boldsymbol{u}_t)$. In this case, we can write up the expected trajectory reward $\mathcal{R}(\boldsymbol{\tau}) = \sum_{t=1}^{T} \mathbb{E}[r(\boldsymbol{x}_t,\boldsymbol{u}_t)]$, where we take the expectation w.r.t. the distribution $p_t(\boldsymbol{x},\boldsymbol{u})$ at each time step. However, to obtain the expected immediate reward in closed form given a Gaussian state-action distribution, the reward function has to come from a specific class for which we can compute the expectation in closed form. Such functions are, for example, quadratic, exponential, trigonometric, etc. On the other hand, for many other nonlinear functions and for non-Gaussian $p_t(\boldsymbol{x},\boldsymbol{u})$, we cannot obtain $\mathbb{E}[r(\boldsymbol{x}_t,\boldsymbol{u}_t)]$ in closed form.

In the following, we will present a novel trajectory and reward prediction algorithm for model-based reinforcement learning using kernel embedding of trajectory distributions. The key idea of our approach is to represent state-action distributions with mean embeddings in feature space, where manipulation of the embeddings become linear algebraic operations with Gram matrices. The first advantage of our approach compared to using GP models is that we avoid the parametric representation of the predictive distribution, and thus, we can capture rich statistical features of the state distribution. Second, we avoid approximations when computing the embedding of the successor state distribution while computations become substantially simpler compared to solving the integrals in Eq. (5.1). Finally, with kernel embedding, the computation of the expected reward $\mathbb{E}[r(\boldsymbol{x}_t, \boldsymbol{u}_t)]$ becomes straightforward without any computational approximation or limitation on the class of the reward function.

The kernel embedding approach has recently been applied to probabilistic inference problems, such as belief propagation [75] and filtering [23]. In reinforcement learning, kernel-based techniques are often used for value function approximation [5, 89]. Taylor et al. [83] give a unifying view of kernelized RL approaches and compare them to GP-based techniques, such as GPRL [61] and GP Temporal Difference [20]. Grünewälder et al. [26] use kernel embedding of the state transition dynamics to approximate the expected value of the successor state distribution in Markov Decision Processes (MDPs). To the best of our knowledge, the kernel embedding approach still has not been explored in the context of model-based policy search and learning dynamics models for long-term trajectory prediction.

Learning time-independent stochastic dynamics can also be represented as a regression problem, where we wish to map the state-action pair $[\boldsymbol{x}^T, \boldsymbol{u}^T]^T$ to the successor state $\boldsymbol{x}'$. Thus, in the following, we present important tools and methodologies for general regression problems, which will form the basis for long-term trajectory and reward prediction.

## 5.1 Regression using Kernel Embedding of Conditional Distributions

Assume the following problem: we wish to find the conditional model $p(\boldsymbol{y}|\boldsymbol{x})$, such that

$$\int_{\boldsymbol{y}} \boldsymbol{\phi}(\boldsymbol{y}) p(\boldsymbol{y}) d\boldsymbol{y} = \int_{\boldsymbol{y}} \boldsymbol{\phi}(\boldsymbol{y}) \int_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x}) q(\boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{y},$$

for some feature $\boldsymbol{\phi}(\boldsymbol{y})$, input $q(\boldsymbol{x})$ and target distribution $p(\boldsymbol{y})$. However, we do not know $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ exactly, but we have access to a finite amount of samples from the distributions $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$. We solve the regression problem with kernel embedding of the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$. Using the results presented in Section 2.4, we represent the mean embedding of the input and target distributions as

$$\boldsymbol{\mu}_{\boldsymbol{x}} = \int_{\boldsymbol{x}} q(\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}) d\boldsymbol{x}, \quad \boldsymbol{\mu}_{\boldsymbol{y}} = \int_{\boldsymbol{y}} p(\boldsymbol{y}) \boldsymbol{\phi}(\boldsymbol{y}) d\boldsymbol{y}.$$

However, as we do not know the distributions $q(\boldsymbol{x})$ and $p(\boldsymbol{y})$ in closed form, we use sample averages to obtain the approximate embeddings

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_i), \;\; \hat{\boldsymbol{\mu}}_{\boldsymbol{y}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\phi}(\boldsymbol{y}_i).$$

Note that in the general case $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{y} \in \mathcal{Y}$, and thus, the features $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x}, \cdot)$ and $\boldsymbol{\phi}(\boldsymbol{y}) = \boldsymbol{g}(\boldsymbol{y}, \cdot)$ are associated with RKHS $\mathcal{F}_{\mathcal{X}}$ and $\mathcal{F}_{\mathcal{Y}}$ respectively. To obtain the mean embedding $\boldsymbol{\mu}_{\boldsymbol{y}}^{q}$ with samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$, we use the conditional covariance operator $\mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}$

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{y}}^{q} &= \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\hat{\boldsymbol{\mu}}_{\boldsymbol{x}} && (5.2) \\ &= \boldsymbol{\Phi}_{\boldsymbol{y}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}_{\boldsymbol{x}}^{T}\hat{\boldsymbol{\mu}}_{\boldsymbol{x}}, && (5.3) \end{aligned}$$

where $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{\Phi}_{\boldsymbol{y}} = [\boldsymbol{\phi}(\boldsymbol{y}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{y}_N)]$, $\boldsymbol{\Phi}_{\boldsymbol{x}} = [\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N)]$ and $\lambda \in \mathbb{R}^{+}$ is a small number that helps to avoid numerical problems during matrix inversion. With notation $\boldsymbol{\mu}_{\boldsymbol{y}}^{q}$ we highlight that $\boldsymbol{x}$ is distributed according to $q(\boldsymbol{x})$. In order to be able to execute computations with the covariance operator, we need to express the mean embedding of $q(\boldsymbol{x})$ as a function of features $\hat{\boldsymbol{\mu}}_{\boldsymbol{x}} = \boldsymbol{\Phi}_{\tilde{\boldsymbol{x}}}\boldsymbol{\alpha}$, where $\boldsymbol{\Phi}_{\tilde{\boldsymbol{x}}} = [\boldsymbol{\phi}(\tilde{\boldsymbol{x}}_1), \ldots, \boldsymbol{\phi}(\tilde{\boldsymbol{x}}_M)]$. Although the values of $\{\tilde{\boldsymbol{x}}_i\}_{i=1}^{M}$ might be arbitrary, computations become simpler in case $\boldsymbol{\Phi}_{\tilde{\boldsymbol{x}}} = \boldsymbol{\Phi}_{\boldsymbol{x}}$. Thus, in the following, we will assume that $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i$, $\forall i$, and thus, $M = N$. For a given $\boldsymbol{\alpha}$, we can write up the mean embedding

$$\boldsymbol{\mu}_{\boldsymbol{y}}^{q} = \boldsymbol{\Phi}_{\boldsymbol{y}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\overbrace{\boldsymbol{\Phi}_{\boldsymbol{x}}^{T}\boldsymbol{\Phi}_{\boldsymbol{x}}}^{\boldsymbol{K}}\boldsymbol{\alpha}, \tag{5.4}$$

which gives $\boldsymbol{\mu}_{\boldsymbol{y}}^{q} = \boldsymbol{\Phi}_{\boldsymbol{y}}\boldsymbol{\alpha}$ for sufficiently small $\lambda$. The question arises, how do we compute $\boldsymbol{\alpha}$ for $q(\boldsymbol{x})$ represented by samples $\{\boldsymbol{x}_l^*\}_{l=1}^L$ from the distribution? One solution to the problem is to minimize the distance between $\boldsymbol{\phi}(\boldsymbol{x}_l^*),\ \forall l$ and the embedding $\boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\alpha}$. The optimal weight $\boldsymbol{\alpha}$ in this case can be found by solving

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_{l=1}^{L} ||\boldsymbol{\phi}(\boldsymbol{x}_l^*) - \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\alpha}||_2. \tag{5.5}$$

By taking the gradient of the objective function w.r.t. $\boldsymbol{\alpha}$ and setting it to 0, we obtain the optimal $\boldsymbol{\alpha}^*$ in closed form

$$\boldsymbol{\alpha}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \langle \boldsymbol{k}(\boldsymbol{x}_l^*) \rangle_{l=1:L}, \tag{5.6}$$

where $\boldsymbol{k}(\boldsymbol{x}_l^*)_j = k(\boldsymbol{x}_j, \boldsymbol{x}_l^*)$ and $\langle \cdot \rangle$ represents the average over the arguments. At this point, we are able to compute the mean embedding of the marginal distribution $\boldsymbol{\mu}_{\boldsymbol{y}}^{q}$ given observed training samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ and query samples $\{\boldsymbol{x}_l^*\}_{l=1}^L$ representing an arbitrary distribution $q(\boldsymbol{x})$,

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{y}}^{q} &= \boldsymbol{\Phi}_{\boldsymbol{y}} (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \langle \boldsymbol{k}(\boldsymbol{x}_l^*) \rangle_{l=1:L} && (5.7) \\ &= \boldsymbol{\Phi}_{\boldsymbol{y}} \boldsymbol{\beta}. && (5.8) \end{aligned}$$

After obtaining the mean embedding $\boldsymbol{\mu}_{\boldsymbol{y}}^{q}$, we might want to compute the most likely value $\boldsymbol{y}^*$ whose feature is the closest to the embedding. The corresponding optimization problem (Sec. 2.4, Eq. (2.45)) is easy to solve for, e.g., polinomial and squared exponential features [23].

### 5.1.1 Connection to Gaussian Process Regression

At this point, it is worth investigating the similarities and differences between Gaussian Process regression and the kernel embedding approach. One of the first important difference is that while GP provides a parametric predictive distribution with mean and variance information, with kernel embedding we obtain only the mean embedding without any information about the prediction variance. However, we find high similarity between the predictive mean of GPs (Eq. (2.26)) and the mean embedding in Eq. (5.7). In fact, with linear target feature $\boldsymbol{\phi}(\boldsymbol{y}) = \boldsymbol{y}$, and if $q(\boldsymbol{x})$ has density only around $\boldsymbol{x}$, that is, $\langle \boldsymbol{k}(\boldsymbol{x}_l^*) \rangle_{l=1:L} \approxeq \boldsymbol{k}(\boldsymbol{x})$, then the two predictive means are roughly the same. However, the kernel embedding approach is more general, as it allows for arbitrary target features $\boldsymbol{\phi}(\boldsymbol{y})$. On the other hand, as the sample weights $\boldsymbol{\beta}$ might have negative elements as well, it is not straightforward how to obtain information about the prediction variance with kernel embedding. Nevertheless, with mean embedding we can compute the predictive kernel embedding for arbitrary input distributions $q(\boldsymbol{x})$ without approximations or assumptions.

## 5.2 Trajectory Prediction with Kernel Embedding

For predicting multiple time steps ahead, we need to repeatedly compute the embedding of the successor state $\boldsymbol{\mu}_{\boldsymbol{x}_{t+1}}$ given the covariance operator $\mathcal{C}_{\boldsymbol{x}'|\boldsymbol{x}}$ and the embedding of the current state $\boldsymbol{\mu}_{\boldsymbol{x}_t}$. Assume that we are given the initial state distribution $p_1(\boldsymbol{x})$ represented by samples $\{\boldsymbol{x}_l^1\}_{l=1}^L$. In the first step, we wish to compute the corresponding embedding $\boldsymbol{\mu}_{\boldsymbol{x}_1} = \boldsymbol{\Phi}_{\boldsymbol{x}} \boldsymbol{\beta}_1$, where we use the features of the observed states

$\{\boldsymbol{x}_i\}_{i=1}^N$ to compute the embedding, that is, $\boldsymbol{\Phi}_{\boldsymbol{x}} = [\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_N)]$. To obtain the successor state embedding $\boldsymbol{\mu}_{\boldsymbol{x}_2}$, we apply the covariance operator $\mathcal{C}_{\boldsymbol{x}'|\boldsymbol{x}}$

$$\boldsymbol{\mu}_{\boldsymbol{x}_2} = \boldsymbol{\Phi}_{\boldsymbol{x}'}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{K}\boldsymbol{\beta}_1, \tag{5.9}$$

$$= \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_2, \tag{5.10}$$

where $\boldsymbol{\beta}_2 = \boldsymbol{\beta}_1$ for sufficiently small $\lambda$. As the successor state embedding will be given with the successor state features $\boldsymbol{\mu}_{\boldsymbol{x}_2} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_2$, the embedding $\boldsymbol{\mu}_{\boldsymbol{x}_3}$ will be computed as

$$\boldsymbol{\mu}_{\boldsymbol{x}_3} = \boldsymbol{\Phi}_{\boldsymbol{x}'}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\tilde{\boldsymbol{K}}\boldsymbol{\beta}_2, \tag{5.11}$$

$$= \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_3, \tag{5.12}$$

where $\tilde{\boldsymbol{K}}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}'_j)$ is the transition kernel matrix. Here we make the assumption that $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{x}' \in \mathcal{X}$ , and thus, we can compute the inner product $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$. Furthermore, we assume that the control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$ is kept fixed throughout the experiments, which will always provide the same trajectory distribution starting from an initial state distribution $p_1(\boldsymbol{x})$. We summarize the trajectory prediction procedure in Table 5.1.

At each time step, we have to compute the weight parameter $\boldsymbol{\beta}_{t+1}$ of the successor state embedding. This operation is a matrix-vector multiplication between matrix $\boldsymbol{C} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\tilde{\boldsymbol{K}}$ representing the dynamics model and the current state embedding weights $\boldsymbol{\beta}_t$. Subsequently, we have the option to either decode the most likely

`Kernel Trajectory Prediction`

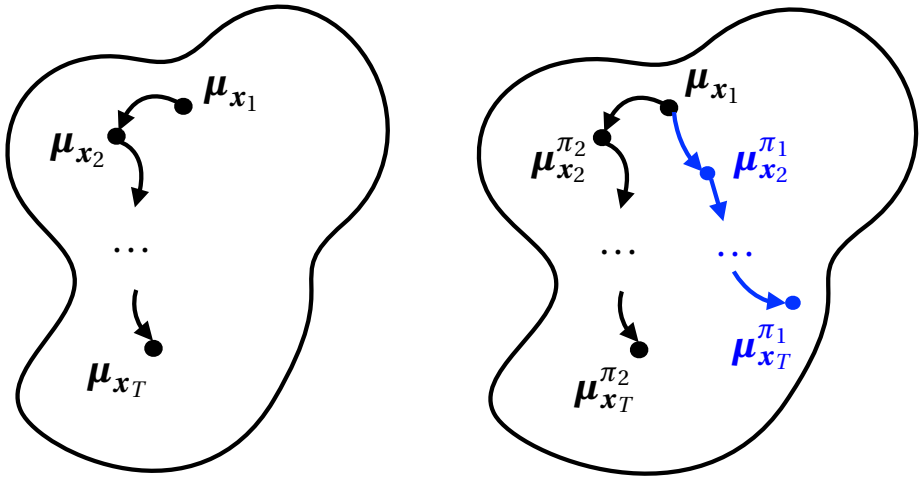
**Input:** observed measurement data $\{\boldsymbol{x}_i, \boldsymbol{x}'_i\}_{i=1}^N$, initial state distribution $\{\boldsymbol{x}_l\}_{l=1}^L$, kernel hyper-parameters $\boldsymbol{\theta}$
**Compute** $\boldsymbol{C} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\tilde{\boldsymbol{K}}$
**Compute** initial embedding $\boldsymbol{\mu}_{\boldsymbol{x}_1} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_1$ using Eq. (5.6)
**for** $t = 1, \ldots, T$
  $\boldsymbol{\beta}_{t+1} = \boldsymbol{C}\boldsymbol{\beta}_t$
  *Successor state embedding:* $\boldsymbol{\mu}_{\boldsymbol{x}_{t+1}} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_{t+1}$
  **Options:**
    *Compute the most likely* $\hat{\boldsymbol{x}}_{t+1}$ *by solving* (2.45)
    *Evaluate function* $\boldsymbol{f} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\delta}$ *using Eq.* (2.43)
    *Compute embedding* $\boldsymbol{\mu}_{\boldsymbol{y}_{t+1}} = \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\boldsymbol{\mu}_{\boldsymbol{x}_{t+1}}$ *for arbitrary* $\boldsymbol{y}$
**Output:** Trajectory embedding $\{\boldsymbol{\mu}_{\boldsymbol{x}_t}\}_{t=1}^T$

TABLE 5.1: The trajectory prediction procedure with kernel embedding using a fixed control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$.

value given the new embedding, evaluate an arbitrary function using the reproducing property, or compute the embedding of another, state dependent distribution, e.g., immediate reward.

Note that when using kernel embedding for trajectory prediction, we compute the embedding of the full state vector. With GP regression, we typically learn a distinct GP model for each output dimension. Thus, kernel embedding not only offers a simple solution to compute the successor state embedding, but requires significantly less hyper-parameters to tune. While the above algorithm serves well to compute the trajectory embedding when using a fixed control policy, in many cases we wish to use a different, possibly time dependent control policy. In this case, the trajectory embedding will not only depend on the initial state distribution, but the control policy as well.

In Figure 5.1 we show the difference of the two prediction procedure. In Fig. 5.1(a) the trajectory embedding will always be the same for a given initial state distribution with a fixed control policy. However, when the control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$ changes due to




(a) (b)

FIGURE 5.1: **(a)** The kernel embedding of the trajectory with a fixed control policy. **(b)** The kernel embedding of the trajectory distribution will depend on the control policy. In the Figure, with $\boldsymbol{\mu}_{\boldsymbol{x}}^{\pi}$ we denote that the embedding of the state will depend on the control policy $\pi(\boldsymbol{u}|\boldsymbol{x})$.

different parametrization, or altered reference trajectory, we obtain different trajectory embeddings due to the different controls associated with the new policy (Fig. 5.1(b)). In this case, we have to ensure that we have the correct embedding of the control distribution given the state. As the control signal only depends on the state variable and the reference trajectory in our formulation, we can write up the embedding of the joint distribution $p_t(\boldsymbol{x},\boldsymbol{u})$ given $p_t(\boldsymbol{x})$. For this purpose, we introduce the state-action variable $\boldsymbol{z}_t = [\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T$. The prediction now can be written as

$$\boldsymbol{\mu}_{\boldsymbol{x}_{t+1}} = \mathcal{C}_{\boldsymbol{x}'|\boldsymbol{z}} \mathcal{C}_{\boldsymbol{z}_t^{\pi}|\boldsymbol{x}_t} \boldsymbol{\mu}_{\boldsymbol{x}_t}, \tag{5.13}$$

where we highlight that $\boldsymbol{z}_t^{\pi}$ is the embedding of $p_t(\boldsymbol{x},\boldsymbol{u}) = p_t(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x}_t,t)$. Note that

the model $\mathcal{C}_{\boldsymbol{x}'|\boldsymbol{z}}$ only depends on the observed state transitions and it is independent of the control policy $\pi(\boldsymbol{u}|\boldsymbol{x},t)$ used for evaluating the prediction. To obtain the embedding of $p(\boldsymbol{z}_t^\pi)$, we have to compute the controls $\boldsymbol{u}_{i,t} \sim \pi(\boldsymbol{u}|\boldsymbol{x}_i,t)$ for each observed state $\boldsymbol{x}_i, i = 1,\ldots,N$ with the new control policy. Assume that the embedding of state $p_t(\boldsymbol{x})$ is given, $\boldsymbol{\mu}_{\boldsymbol{x}_t} = \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{\beta}_t$. In this case the embedding of $\boldsymbol{z}_t^\pi$ simply becomes $\boldsymbol{\mu}_{\boldsymbol{z}_t^\pi} = \boldsymbol{\Phi}_{\boldsymbol{z}_t^\pi}\boldsymbol{\beta}_t$. Thus, the embedding of the successor state distribution is

$$\boldsymbol{\mu}_{\boldsymbol{x}_{t+1}} = \boldsymbol{\Phi}_{\boldsymbol{x}'}(\boldsymbol{G}+\lambda\boldsymbol{I})^{-1}\overbrace{\boldsymbol{\Phi}_{\boldsymbol{z}}^T\boldsymbol{\Phi}_{\boldsymbol{z}_t^\pi}}^{\boldsymbol{G}_t^\pi}\boldsymbol{\beta}_t, \tag{5.14}$$

where $\boldsymbol{G} = \boldsymbol{\Phi}_{\boldsymbol{z}}^T\boldsymbol{\Phi}_{\boldsymbol{z}}$ is the kernel matrix based on the observed samples $\{\boldsymbol{z}_i\}_{i=1}^N$ and $\tilde{\boldsymbol{G}}_t^\pi = \boldsymbol{\Phi}_{\boldsymbol{z}}^T\boldsymbol{\Phi}_{\boldsymbol{z}_t^\pi}$ is the policy dependent kernel matrix. When predicting for multiple time steps ahead, we will obtain the embedding $\boldsymbol{\mu}_{\boldsymbol{x}_t} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_t$ with the observed successor state features $\boldsymbol{\Phi}_{\boldsymbol{x}'}$. Thus, when computing the successor state embedding, instead of $\boldsymbol{G}_t^\pi = \boldsymbol{\Phi}_{\boldsymbol{z}}^T\boldsymbol{\Phi}_{\boldsymbol{z}_t^\pi}$, we will use $\tilde{\boldsymbol{G}}_t^\pi = \boldsymbol{\Phi}_{\boldsymbol{z}}^T\boldsymbol{\Phi}_{\boldsymbol{z}_t'^\pi}$, which is the control policy dependent transition kernel matrix. Note, that in case we use the same time independent control policy for prediction and data collection, then $\tilde{\boldsymbol{G}}_t^\pi = \tilde{\boldsymbol{G}}^\pi, \forall\, t$ and $\tilde{\boldsymbol{G}}^\pi = \tilde{\boldsymbol{G}}$, and thus, we arrive to the prediction procedure presented in Table 5.1.

The prediction procedure for the general, time dependent case, is summarized in Table 5.2. In the first step, we compute $\boldsymbol{z}_{i,t}'^\pi$ for each observed state and time step, which we then use to compute the initial embedding $\boldsymbol{\mu}_{\boldsymbol{z}_1^\pi} = \boldsymbol{\Phi}_{\boldsymbol{z}_1'^\pi}\boldsymbol{\beta}_1$ and the policy dependent transition matrix $\boldsymbol{C}_t^\pi = (\boldsymbol{G}+\lambda\boldsymbol{I})^{-1}\tilde{\boldsymbol{G}}_t^\pi, \forall\, t$. Subsequently, for each time step we can obtain the embedding of the new state vector by a matrix-vector multiplication. Additionally, we can compute the most likely value for the current state embedding

`Kernel Trajectory Prediction with Arbitrary Control Policy`

**Input:** observed measurement data $\{\boldsymbol{x}_i, \boldsymbol{x}'_i\}_{i=1}^N$, control policy $\pi(\boldsymbol{u}|\boldsymbol{x}, t)$, initial state distribution $\{\boldsymbol{x}_l\}_{l=1}^L$, kernel hyper-parameters $\boldsymbol{\theta}$
**Compute** $\boldsymbol{z}'^{\pi}_{i,t} = [\boldsymbol{x}'^T_i, \boldsymbol{u}'^T_{i,t}]^T, i = 1,\ldots,N, t = 1,\ldots,T$
**Compute** $\boldsymbol{C}^{\pi}_t = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1}\tilde{\boldsymbol{G}}^{\pi}_t, t = 1,\ldots,T$
**Compute** initial embedding $\boldsymbol{\mu}_{\boldsymbol{z}^{\pi}_1} = \boldsymbol{\Phi}_{\boldsymbol{z}'^{\pi}_1}\boldsymbol{\beta}_1$ using Eq. (5.6)
**for** $t = 1,\ldots,T$
  $\boldsymbol{\beta}_{t+1} = \boldsymbol{C}^{\pi}_t\boldsymbol{\beta}_t$
  *Successor state embedding:* $\boldsymbol{\mu}_{\boldsymbol{x}^{\pi}_{t+1}} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\beta}_{t+1}$
  **Options:**
    *Compute the most likely* $\hat{\boldsymbol{x}}_{t+1}$ *by solving* (2.45)
    *Evaluate function* $\boldsymbol{f} = \boldsymbol{\Phi}_{\boldsymbol{x}'}\boldsymbol{\delta}$ *using Eq.* (2.43)
    *Compute embedding* $\boldsymbol{\mu}_{\boldsymbol{y}_{t+1}} = \mathcal{C}_{\boldsymbol{y}|\boldsymbol{x}}\boldsymbol{\mu}_{\boldsymbol{x}^{\pi}_{t+1}}$ *for arbitrary* $\boldsymbol{y}$
**Output:** Trajectory embedding $\{\boldsymbol{\mu}_{\boldsymbol{x}^{\pi}_t}\}_{t=1}^T$

TABLE 5.2: The trajectory prediction procedure with arbitrary, time dependent control policy using the kernel embedding approach.

with Eq. (2.45), compute another state dependent embedding $\boldsymbol{\mu}_{\boldsymbol{y}_{t+1}}$, or we might wish to evaluate a function with the reproducing property (Eq. (2.43)).

## 5.3 Model Selection

Although the kernel embedding approach is non-parametric, as with GP regression, we still have to solve the model selection problem, that is, choosing a kernel function and finding its optimal hyper-parameters. While the regression problem usually gives a good intuition for choosing the kernel function, finding the optimal hyper-parameters is a more challenging problem. For finding the hyper-parameters with GP regression, we typically maximize the likelihood of the predictive distribution, which offers a good solution in most cases. However, when using the kernel embedding approach, we do not have a parametric form of the predictive distribution, and thus, we do not have a likelihood function to optimize. Using cross-validation with

approximation of the likelihood function is not straightforward as we do not have any information about the prediction variance. An alternative approach for hyper-parameter optimization is to minimize the squared distance of the predictive mean embedding and the observed feature vector [27], that is,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} ||\boldsymbol{\phi}(\boldsymbol{z}'_i) - \boldsymbol{\mu}_{\boldsymbol{z}'_i}||_2 \tag{5.15}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} g(\boldsymbol{z}'_i, \boldsymbol{z}'_i) - 2\boldsymbol{g}'(\boldsymbol{z}'_i)\boldsymbol{\beta}_i + \boldsymbol{\beta}_i^T \boldsymbol{G}'\boldsymbol{\beta}_i, \tag{5.16}$$

where $\boldsymbol{\beta}_i = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1}\boldsymbol{g}(\boldsymbol{z}_i)$, the $j^{th}$ element of $\boldsymbol{g}'(\boldsymbol{z}'_i)$ is $g(\boldsymbol{z}'_i, \boldsymbol{z}'_j)$, $\boldsymbol{G}'_{ij} = g(\boldsymbol{z}'_i, \boldsymbol{z}'_j)$ and $\boldsymbol{\phi}(\boldsymbol{z}'_i)$ is the observed successor state-action feature. Here, $\boldsymbol{\theta}$ refers to the hyper-parameters of the kernel function $k$. The above optimization problem can be efficiently solved using cross-validation.

Note that for model-based RL tasks, instead of the state embedding error criterion in Eq. (5.15), we can directly minimize the reward prediction error by solving

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} ||\boldsymbol{\phi}(r'_i) - \boldsymbol{\mu}_{r'_i}||_2, \tag{5.17}$$

where $r'_i = r(\boldsymbol{x}'_i, \boldsymbol{u}'_i)$ is the observed immediate reward function of successor state-action pairs and $\boldsymbol{\mu}_{r'_i}$ is the model prediction. In case of linear reward kernel function, the optimization algorithm becomes

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{5.18}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (r'_i - \boldsymbol{r}'^T \boldsymbol{\beta}_i)^2, \tag{5.19}$$

where $\boldsymbol{r}'$ represents the vector of observed successor rewards and $\boldsymbol{\beta}_i = (\boldsymbol{G}+\lambda \boldsymbol{I})^{-1}\boldsymbol{g}(\boldsymbol{z}_i)$. The optimization problem can be solved by, e.g., the conjugate gradient algorithm, where the gradients are

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -2\sum_{i=1}^{N}(r'_i - \boldsymbol{r}'^T\boldsymbol{\beta}_i)\boldsymbol{r}'^T\frac{\partial \boldsymbol{\beta}_i}{\partial \boldsymbol{\theta}}, \tag{5.20}$$

$$\frac{\partial \boldsymbol{\beta}_i}{\partial \boldsymbol{\theta}} = (\boldsymbol{G}+\lambda \boldsymbol{I})^{-1}\left(\frac{\partial \boldsymbol{g}(\boldsymbol{z}_i)}{\partial \boldsymbol{\theta}} - \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{\theta}}(\boldsymbol{G}+\lambda \boldsymbol{I})^{-1}\boldsymbol{g}(\boldsymbol{z}_i)\right). \tag{5.21}$$

As it is discussed in [1], finding the parameters of dynamics models by minimizing the state prediction error *independently* for each time step might result in poor trajectory prediction performance. The prediction error is propagated forward in time, and thus, small errors in the first few prediction steps could lead to more biased trajectory predictions. Although [1] suggests an algorithm to learn linear models for accurate long-term trajectory prediction, direct application for parametric probabilistic models might lead to biased learned models, due to the approximations involved in the prediction process. In section 4.2.4 we proposed a novel probabilistic model learning framework, that does not require the solution of the integrals, but rather works with feature averages. On the other hand the algorithm requires the model to match these feature constraints, which is typically difficult for any probabilistic model. However, with the kernel embedding approach we do not require solving integrals or satisfying feature constraints, and thus, the optimization problem becomes simple while avoiding approximations. Given the observed trajectories $\{\boldsymbol{\tau}_i\}_{i=1}^{M}$, our

aim is to solve

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{5.22}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{M} \sum_{t=1}^{T} ||\boldsymbol{\phi}(r_t^i) - \boldsymbol{\Phi}_{r'} \boldsymbol{\beta}_t^i||_2, \tag{5.23}$$

where the model predicted reward embedding is $\boldsymbol{\mu}_{r_t^i} = \boldsymbol{\Phi}_{r'} \boldsymbol{\beta}_t^i$, with the weights $\boldsymbol{\beta}_t^i = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1} \tilde{\boldsymbol{G}}_{t-1}^{\pi_i} \boldsymbol{\beta}_{t-1}^i$ and $\boldsymbol{\beta}_1^i = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1} \tilde{\boldsymbol{g}}_1^{\pi_i}(\boldsymbol{z}_1^i)$. Note that the kernel function $g(\boldsymbol{z}, \boldsymbol{z}')$ is parametrized by $\boldsymbol{\theta}$. We will use the squared exponential kernel for state-action values and linear features for rewards. To avoid overfiting, we will use cross-validation. To solve the above optimization problem with gradient techniques, we need to compute the following gradients

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -2 \sum_{i=1}^{M} \sum_{t=1}^{T} (r_t^i - \boldsymbol{r}'^T \boldsymbol{\beta}_t^i) \boldsymbol{r}'^T \frac{\partial \boldsymbol{\beta}_t^i}{\partial \boldsymbol{\theta}}, \tag{5.24}$$

$$\frac{\partial \boldsymbol{\beta}_t^i}{\partial \boldsymbol{\theta}} = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1} \tilde{\boldsymbol{G}}_{t-1}^{\pi_i} \frac{\partial \boldsymbol{\beta}_{t-1}^i}{\partial \boldsymbol{\theta}}, \tag{5.25}$$

$$\frac{\partial \boldsymbol{\beta}_1^i}{\partial \boldsymbol{\theta}} = (\boldsymbol{G} + \lambda \boldsymbol{I})^{-1} \left( \frac{\partial \tilde{\boldsymbol{g}}_1^{\pi_i}(\boldsymbol{z}_1^i)}{\partial \boldsymbol{\theta}} - \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{\theta}} \boldsymbol{\beta}_1^i \right). \tag{5.26}$$

## 5.4 Results

After presenting the algorithms for obtaining the kernel embedding of trajectory distributions, we turn our attention to simulations. In our first experiment, we will compare GP modeling and the kernel embedding approach in a simple trajectory prediction problem. We wish to predict the trajectory of a single link pendulum, whose dynamics is described by

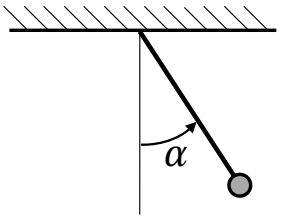


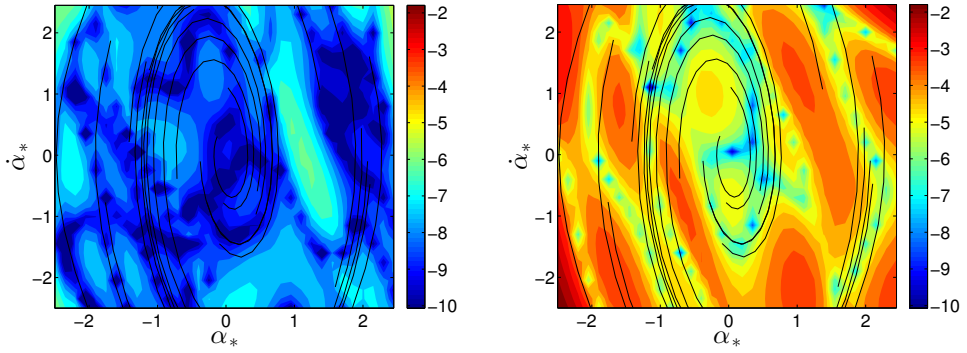

FIGURE 5.2: The pendulum.

$$\ddot{\alpha} = -9.81\sin(\alpha) + u,$$

where $\alpha$ is the vertical angle displacement (Fig. 5.2). We chose this simple system, such that we can easily visualize the results of trajectory prediction and generalization. We use a stochastic state feedback controller to control $\alpha$ to 0, $u \sim \mathcal{N}(u|\boldsymbol{K}^T\boldsymbol{x}, 0.05^2)$, where $K = [-3,-1]^T$ and $\boldsymbol{x} = [\alpha, \dot{\alpha}]^T$. Furthermore, we apply control constraints $|u| \leq 3$. Thus, the dynamics of the system is close to linear around $\alpha = 0$, but gets nonlinear as the absolute value of the angle and the angular velocity increases. To learn the models, first we collect measurement data from randomly initialized states. We evaluate 15 trajectories for 20 time steps, where the sampling time is $\Delta t = 0.1$ sec. To train the hyper-parameters of the GP model, we maximize the marginal log-likelihood in Eq. (2.28). For the kernel embedding approach, we minimize the squared distance between the prediction and the observed successor state

embedding for each sample, that is, we solve the optimization problem in Eq. (5.16) with 10-fold cross-validation.

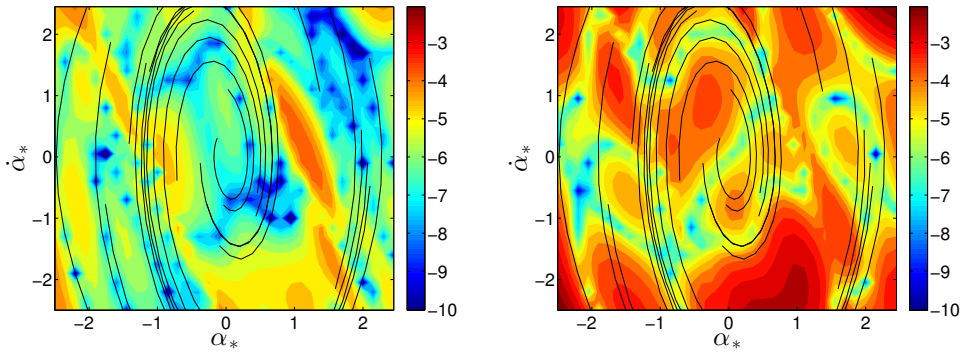
We first compare the two models in a single step prediction problem. For each query state $\boldsymbol{x}_* = [\alpha_*, \dot{\alpha}_*]^T$, we compute the predictive means $\mathbb{E}[\boldsymbol{x}'_*]$ and measure the prediction accuracy by the squared error measures $e_\alpha = (\hat{\alpha}'_* - \mathbb{E}[\alpha'])^2$ and $e_{\dot{\alpha}} = (\hat{\dot{\alpha}}'_* - \mathbb{E}[\dot{\alpha}'])^2$, $e_{\boldsymbol{x}} = (\hat{\boldsymbol{x}}'_* - \mathbb{E}[\boldsymbol{x}'])^T(\hat{\boldsymbol{x}}'_* - \mathbb{E}[\boldsymbol{x}'])$, where $\hat{\boldsymbol{x}}'_* = \mathbb{E}[\boldsymbol{x}'_*]$ is the real expected successor state.




(a) (b)

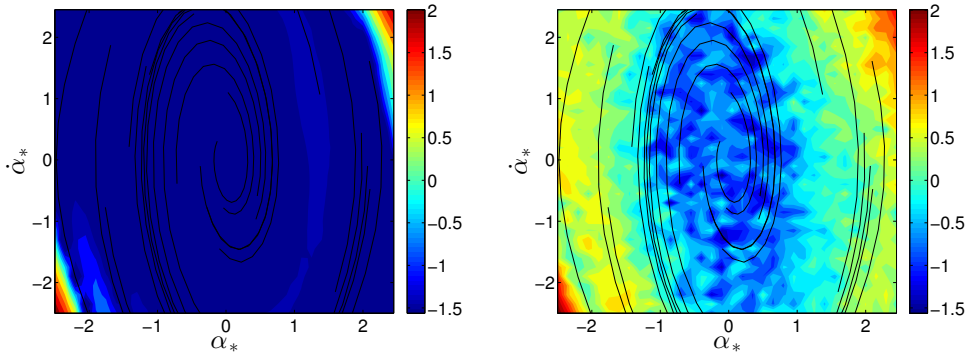
FIGURE 5.3: **(a)** Prediction error $\log_{10} e_\alpha$ with kernel embedding. **(b)** Prediction error $\log_{10} e_\alpha$ with GP regression. Note that the contour levels are the same for both Figures.

In Figures 5.3 and 5.4, we show the single step prediction errors with the kernel embedding and the GP modeling approach with the colored contours. The kernel embedding approach consistently provides more accurate predictions and it generalizes better in less explored parts of the state space. In the figures, we highlighted the observed trajectories with black curves. The models typically provide better predictions in the vicinity of the observed trajectories. In the trajectory prediction experiment, the kernel embedding approach significantly outperforms GP modeling (Fig. 5.5). Moreover, computations times are substantially lower compared to the GP approach,




(a) (b)

FIGURE 5.4: **(a)** Prediction error $\log_{10} e_{\dot{\alpha}}$ with kernel embedding. **(b)** Prediction error $\log_{10} e_{\dot{\alpha}}$ with GP regression. Note that the contour levels are the same for both figures.




(a) (b)

FIGURE 5.5: **(a)** Trajectory prediction error $\log_{10} \sum_{t=1}^{20} e_{\boldsymbol{x}_t}$ with kernel embedding. **(b)** Trajectory prediction error $\log_{10} \sum_{t=1}^{20} e_{\boldsymbol{x}_t}$ with GP regression. Note that the contour levels are the same for both figures.

where we used sampling to obtain an accurate approximation of the trajectory distribution.

In the next experiment, we compare the reward prediction accuracy when using kernel embedding and GP modeling. We solve the reward prediction problem of the balancing task presented in Sec. 4.3.1. We use a PD controller to balance the robot to

the upright position $\boldsymbol{q}_0 = [0,0,0,\pi]^T$ from a random starting state around the upright position. The uncorrelated random initial states are coming from a Gaussian distribution with state means $\boldsymbol{q}_0$, velocity means $\dot{\boldsymbol{q}} = \mathbf{0}$ and with standard deviation 0.3 in each dimension. Furthermore, we add i.i.d. Gaussian control torque noise with standard deviation 4 in each dimension, such that the dynamics becomes stochastic, but the controller is still able to robustly control the system. We define the immediate reward function to be quadratic $r(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u}) = -10\tilde{\boldsymbol{q}}^T\tilde{\boldsymbol{q}} - 10^{-1}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} - 10^{-5}\boldsymbol{u}^T\boldsymbol{u}$, where $\tilde{\boldsymbol{q}} = \boldsymbol{q} - \boldsymbol{q}_0$ is the deviation from the upright position. The reward for the whole trajectory is the sum of immediate rewards $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} r(\boldsymbol{q}_t,\dot{\boldsymbol{q}}_t,\boldsymbol{u}_t)$ with $T = 40$. When using GP models, we use the sampling approach to average over sample trajectory rewards to obtain the expected trajectory reward $\mathcal{R}(\boldsymbol{\tau}) = \sum_{i=1}^{M} R(\boldsymbol{\tau}_i)/M$ with $M = 100$. For the kernel embedding approach, we compute the embedding of immediate rewards $\boldsymbol{\mu}_{r_t} = \boldsymbol{\Phi}_{\boldsymbol{r}'}\boldsymbol{\beta}_t$ with the embedding $\boldsymbol{\mu}_{\boldsymbol{z}_t} = \boldsymbol{\Phi}_{\boldsymbol{z}'}\boldsymbol{\beta}_t$, where $\boldsymbol{r}'_i$ is the reward for observed samples $\boldsymbol{z}'_i = [\boldsymbol{q}'^T_i, \dot{\boldsymbol{q}}'^T_i, \boldsymbol{u}'^T_i]^T, i = 1,\ldots,N$. We use linear kernel for the reward, and thus, we get the expected immediate reward $\mathbb{E}[r_t] = \boldsymbol{r}'^T\boldsymbol{\beta}_t$. Finally, we compute the expected trajectory reward as $\mathcal{R}(\boldsymbol{\tau}) = \sum_{t=1}^{T}\mathbb{E}[r_t]$.

For GP modeling we optimize the hyper-parameters by maximizing the log-marginal likelihood in Eq. (2.28) for each state dimension separately. For the kernel embedding approach, we will consider two training methods presented in 5.3. First, we will train the hyper-parameters by minimizing the single step reward prediction error in feature space (Eq. (5.17)). We also evaluate the hyper-parameter training for accurate long-term prediction (Eq. (5.23)). To avoid overfiting, we use 10-fold cross-validation for both approaches.

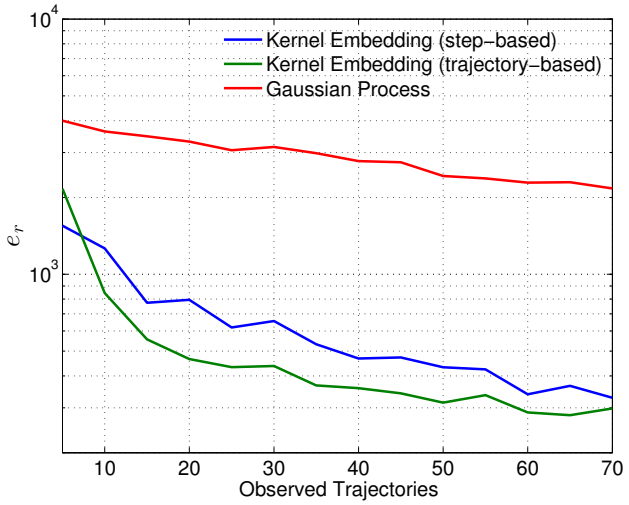



FIGURE 5.6: The expected trajectory reward prediction accuracy of the kernel embedding and the GP approach.

In Figure 5.6 we show the results of the expected trajectory reward prediction task for an increasing amount of observed trajectories. We define the reward error $e_r = \sum_{i=1}^{L}(\mathcal{R}(\boldsymbol{\tau}_i) - \hat{R}(\boldsymbol{\tau}_i))^2/L$, where $L = 200$ and $\hat{R}(\boldsymbol{\tau}_i)$ is the real expected trajectory reward for $i^{th}$ experiment, which we evaluated by sampling 100 trajectories using the real dynamics. As the figure shows, the kernel embedding approach provides an order of magnitude lower reward prediction error compared to the GP approach. For a lower amount of observed trajectories, the GP model provides less confident predictions, which leads to higher prediction variance, and thus, larger reward prediction error. The kernel embedding approach consistently provides accurate reward predictions, even for a lower amount of observed trajectories. The figure shows that directly maximizing the long term prediction error in Eq. (5.23) (denoted by *trajectory-based*) clearly has a positive effect on the predictive performance compared to the *step-based* approach (Eq. (5.17)). Such a training approach is not straightforward for GP and other probabilistic modeling methods, where computing the exact trajectory

distribution is difficult.

We also evaluated the reward prediction accuracy with an increasing amount of control torque noise on the system. We show the results in Figure 5.7. As we can see, the GP modeling tends to provide higher prediction error due to the less confident predictions. However, with the kernel embedding approach we barely see any difference in the prediction performance.

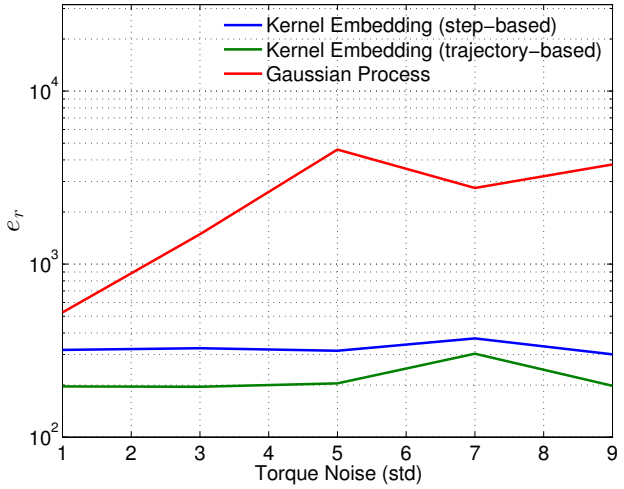



FIGURE 5.7: The expected trajectory reward prediction accuracy of the kernel embedding and the GP approach with an increasing amount of control torque noise.

This experiment concludes that the kernel embedding approach provides better generalization and prediction results, even with more complex dynamics, compared to GP models.

## 5.5 Discussion

Kernel embedding provides a principled approach for non-parametric probabilistic inference in a data driven fashion. The most important advantages compared to using GP regression are the superior generalization property, less hyper-parameters to tune, fast computations and the lack of assumptions and approximations.

However, a significant disadvantage of the kernel embedding approach is that we cannot obtain any information about the prediction confidence. This prevents the definition of a likelihood function, which might be useful for tuning the kernel function hyper-parameters. Additionally, evaluating functions $f(\boldsymbol{y})$, such as an immediate reward function, using the reproducing property is less intuitive than computing the expected function value in the target space $\mathcal{Y}$, $\boldsymbol{y} \in \mathcal{Y}$. Furthermore, approximating the function $f = \boldsymbol{\Phi}_{\boldsymbol{y}}\boldsymbol{\delta}$ using the observed data is challenging and it is prone to inaccuracies. Alternatively, we can use samples from the function evaluated at the training points to compute the embedding of the expected function value. For example, when computing the expected reward in the balancing task, we use the real reward function to compute the immediate reward only in the observed states and provide the predictive reward as the weighted sum of observed reward samples. While this approach provides a more or less accurate evaluation of the function, it is essential to extract prediction confidence with the kernel embedding approach for more principled function evaluation. Additionally, based on our observations, scaling to higher dimensional dynamics is challenging in the current setup. We typically require more samples compared to GP regression to achieve a better prediction performance.

As the kernel embedding approach is inherently sample based, we have to collect enough samples to be able to provide an accurate prediction. However, we can only guarantee good prediction performance for query inputs inside the convex hull of the observed input space. Outside the convex hull the prediction is less reliable and it is mostly dependent on the kernel function associated with the RKHS. Thus, in order to provide a good prediction performance we need a relatively high number of samples. However, with increasing input dimensionality we need exponentially more samples to cover a unit region of the space. Thus, scaling the model learning algorithm to higher degree of freedom robots becomes increasingly more difficult. Nevertheless, for most prediction tasks used for robot skill learning we are typically interested in only a smaller subspace of the full state-action space. Thus, obtaining only a few samples from the trajectory distribution might already provide a sufficient number of samples.

Furthermore, based on our observations, the hyper-parameter training method based on minimizing the squared distance between the observed and predicted features (Eq. (5.16)) often overfits the data. This could lead to overly confident predictions in unexplored parts of the state space. However, when modeling robot dynamics, we can exploit the ideas of training probabilistic models for accurate long-term trajectory prediction (Sec. 4.2.4). Using this approach, we can train the hyper-parameters to minimize the state-action feature difference between the observed and the predicted samples at each time step. By solving the primal problem, we do not require the model to satisfy the feature matching constraint at each time step. Thus,

the resulting algorithm will be straightforwardly applicable with any kernel function. We showed an example for this training approach for accurate long-term reward prediction with linear features (Eq. (5.23)). Altogether, we believe that the proposed kernel-based trajectory prediction algorithm already improves significantly the performance of existing model-based RL algorithms, such as GPREPS. Future work will investigate the application of the proposed method to higher dimensional state-action spaces.

# Chapter 6

# Conclusion

In this thesis, we presented novel algorithms for more efficient generalization of robot skills. The contributions focused on two major topics: model-based robot skill generalization and probabilistic robot dynamics learning with a special focus on non-parametric methods. We showed in simulations and in a real experiment that the novel tools provide unprecedented data-efficiency for learning, and superior trajectory and reward prediction accuracy compared to existing methods.

## 6.1 Summary of Contributions

**Evaluation of Contextual REPS.** First, we gave an overview of contextual REPS and we presented the derivation of the policy update equation and proposed a sample-based implementation of the algorithm. We showed in complex robot simulations, such as robot table tennis, hockey and ball throwing, that with contextual REPS we

are able to consistently learn high quality contextual upper-level policies, while promoting safe exploration. The only open parameter of the algorithm is the relative entropy bound, which is typically easy to choose for a given learning problem. Furthermore, the contextual REPS can be extended to learn hierarchical policies for choosing multiple options for performing the task [14].

**Gaussian Process REPS.** We presented GPREPS, a novel model-based contextual policy search algorithm based on GP models and on the contextual REPS algorithm. We learn an upper-level policy that efficiently generalizes the lower-level policy parameters over multiple contexts. GPREPS exploits learned probabilistic forward models of the robot and its environment to predict expected rewards of artificially generated data points. For evaluating the expected reward, GPREPS samples trajectories using the learned models. Unlike deterministic inference methods used in state-of-the art approaches for policy evaluation, trajectory sampling is easy to implement, easy to parallelize and does not limit the policy class or the used reward model.

With simulated and real robot experiments, we demonstrated that GPREPS significantly reduces the required amount of measurement data to learn high quality policies compared to state-of-the-art model-free contextual policy search approaches. Moreover, the GP models are able to account for model uncertainty and produce accurate trajectory distributions. Thus, with GPREPS we avoid the risk of learning from noisy reward samples that results in a bias in the model-free REPS formulation. The increased data efficiency makes GPREPS applicable to learning contextual policies in real-robot tasks. Since existing model-based policy search methods cannot be

applied to the contextual setup, GPREPS allows for many new applications of model-based policy search.

**Probabilistic model learning.** We presented a novel probabilistic model learning framework for accurate long-term trajectory prediction. Instead of solving the primal problem, which would require the computation of the whole trajectory distribution, we optimize the dual objective, which connects the time steps via the Lagrangian multipliers. While the Lagrangian parameters are convex in the dual function, we still have to compute the gradient w.r.t. the hyper-parameters of the probabilistic model, which is in general non-convex. Although the state transition constraints are difficult to satisfy for some models, we believe that the proposed algorithm will provide accurate long-term predictions for an appropriate class of models.

**Kernel embedding of trajectory distributions.** Using recent results in kernel embedding of conditional distributions, we proposed a novel trajectory prediction algorithm. The kernel-based algorithm avoids approximations and assumptions when computing the successor state distribution with probabilistic models. Furthermore, computations become substantially faster with linear algebraic operations with Gram matrices compared to solving integrals with, e.g., the moment matching approach. We showed simulation results for both trajectory and reward prediction accuracy, and concluded that the kernel embedding approach has better generalization and prediction accuracy compared to the GP modeling approach. Thus, we believe that the kernel embedding approach will be an important tool for future model-based RL algorithms.

## 6.2 Future Work

**Active learning of dynamics models.** A significant shortcoming of the GPREPS algorithm and other, GP model-based learning algorithms, such as PILCO, is the computational demand of model training and prediction. While sparse GP methods can mitigate the problem, for robot dynamics learning tasks, the amount of training samples might quickly become overly large, which could lead to impractically slow performance, even with sparse methods. In GPREPS, after each policy update, we collect new measurement data following the controller $\pi_{\boldsymbol{\omega}}(\boldsymbol{u}|\boldsymbol{x})$ with parametrization $\boldsymbol{\omega} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{\omega})$. However, the upper-level policy is set, such that it provides high reward rollouts in a given context, without any guarantees for collecting informative data points for model learning. This might result in an overly large number of total, but a surprisingly low number of informative training samples for model learning. Borrowing the ideas of dual control, future research will focus on learning control policies, which not only provide high reward rollouts, but highly informative training samples for model learning. While the two objectives might be contradictory around the optimal solution, the early stages of the learning greatly benefit from informative models. In later stages of the learning procedure, when the upper-level policy narrows down the region of possible optimal solutions, we can switch to model-free learning for a less biased final controller parametrization.

**Learning with partially observable context.** Contextual policy search algorithms, such as REPS and GPREPS, have proven to be highly efficient in robot skill generalization. In the experiments presented in this thesis, we assumed that the context is

a deterministic and observable variable. However, in many situations, such as lifting an unknown mass, the context might be a stochastic random variable. While we can maintain a belief over the context distribution, it is not straightforward how to generalize robot skills in this situation. Thus, future research will focus on robot skill generalization with stochastic and partially observable contexts. This will allow a more general learning setting.

**Contextual dynamics models.** In our experiments we assumed that the context variable does not influence the dynamics of the robot. However, in an object lifting task the mass of the object and thus, the dynamics of the robot might change between task executions. As the mass will influence the dynamics of the robot, the trajectory distribution will not only depend on the lower-level policy parametrization, but on the context variable as well. Thus, when using model-based techniques, such as PILCO and GPREPS, we have to condition on the context variable when predicting the successor state distribution. For GP models with deterministic and observable context variables, we can straightforwardly exploit the context information for model training and trajectory prediction. However, when we only have access to a distribution over the context variable, we have to marginalize over the context as well to obtain the successor state distribution. This would further complicate the already involved prediction procedure in the parametric case. However, the kernel embedding approach offers a principled solution to the problem. When computing the embedding of the successor state distribution, we can simply include the embedding of the context distribution in the prediction equation. Further evaluation of this approach will be the topic of future research.

**Real-time model learning.** The kernel embedding approach offers a novel solution for online learning the robot inverse dynamics model, similar to real-time local GPs [52]. Having an accurate inverse dynamics model allows for compliant robot control, which is important for robots working in human environments. While real-time local GPs learn the robot dynamics efficiently, adaptation to changing dynamics due to, e.g., grasping heavy objects, requires the forgetting of previous measurements. However, with the kernel embedding approach, we can exploit the kernel Bayes' rule to condition on the history of measurements to obtain the embedding of the forward torque. Thus, we can quickly adapt to new situations with altered dynamics without having to forget previous data. Thorough investigation of a local kernel embedding approach will be the topic of future research.

**Prediction variance with kernel embedding.** A major shortcoming of the kernel embedding approach is the lack of prediction variance information, which prevents accurate function evaluation and hyper-parameter training. Thus, future research will focus on how the conditional covariance operator can be represented in a probabilistic way, such that it provides prediction variance for the target mean embedding.

# Appendix A

# Derivation of Contextual Episode-based REPS

The constrained optimization problem of episode-based REPS for contextual policy search is given by

$$\max_{p} \quad \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} d\boldsymbol{s} d\boldsymbol{\omega}, \tag{A.1}$$

$$\text{s.t.:} \quad \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})} d\boldsymbol{s} d\boldsymbol{\omega} \leq \epsilon, \tag{A.2}$$

$$\iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{s} d\boldsymbol{\omega} = \hat{\boldsymbol{\phi}}, \tag{A.3}$$

$$\iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) d\boldsymbol{s} d\boldsymbol{\omega} = 1. \tag{A.4}$$

We can write up the Lagrangian of the corresponding constrained optimization problem in the form

$$\mathcal{L}(p,\eta,\boldsymbol{\gamma}) = \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}\, d\boldsymbol{s}d\boldsymbol{\omega} + \eta\left(\epsilon - \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})} d\boldsymbol{s}d\boldsymbol{\omega}\right)$$
$$+ \boldsymbol{\gamma}^T\left(\hat{\boldsymbol{\phi}} - \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{s}d\boldsymbol{\omega}\right) + \lambda\left(1 - \iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) d\boldsymbol{s}d\boldsymbol{\omega}\right). \tag{A.5}$$

By setting the gradient of $\mathcal{L}(p,\eta,\boldsymbol{\gamma})$ w.r.t. $p(\boldsymbol{s},\boldsymbol{\omega})$ to zero we obtain the solution

$$p(\boldsymbol{s},\boldsymbol{\omega}) = q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - \boldsymbol{\gamma}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) \exp\left(-\frac{\eta+\lambda}{\eta}\right), \tag{A.6}$$

with the base line $V(\boldsymbol{s}) = \boldsymbol{\gamma}^T\boldsymbol{\phi}(\boldsymbol{s})$. Due to the constraint $\iint_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) d\boldsymbol{s}d\boldsymbol{\omega} = 1$, we also have that

$$\exp\left(-\frac{\eta+\lambda}{\eta}\right) = \left[\iint_{\boldsymbol{s},\boldsymbol{\omega}} q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - \boldsymbol{\gamma}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) d\boldsymbol{s}d\boldsymbol{\omega}\right]^{-1}. \tag{A.7}$$

The dual function is obtained by setting the solution for $p(\boldsymbol{s},\boldsymbol{\omega})$ back into the Lagrangian. After rearranging terms, we obtain

$$g(\eta,\boldsymbol{\gamma},\lambda) = \eta + \lambda + \eta\epsilon + \boldsymbol{\gamma}^T\hat{\boldsymbol{\phi}} = \eta \log \exp\left(\frac{\eta+\lambda}{\eta}\right) + \eta\epsilon + \boldsymbol{\gamma}^T\hat{\boldsymbol{\phi}} \tag{A.8}$$

Setting Equation (A.7) into the dual we can eliminate the $\lambda$ parameter and obtain the dual function

$$g(\eta,\boldsymbol{\gamma}) = \eta \log\left(\iint_{\boldsymbol{s},\boldsymbol{\omega}} q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - \boldsymbol{\gamma}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) d\boldsymbol{s}d\boldsymbol{\omega}\right) + \eta\epsilon + \boldsymbol{\gamma}^T\hat{\boldsymbol{\phi}}. \tag{A.9}$$

Using a dataset $\mathcal{D} = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]}\}_{i=1\ldots N}$ where the context parameter pairs have been sampled from $q(\boldsymbol{s}, \boldsymbol{\omega})$, the integral in the dual function can be approximated as

$$g(\eta, \boldsymbol{\gamma}; \mathcal{D}) = \eta \log\left(\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]} - \boldsymbol{\gamma}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right)\right) + \eta\epsilon + \boldsymbol{\gamma}^T \hat{\boldsymbol{\phi}}. \tag{A.10}$$

The dual function is convex in $\eta$ and $\boldsymbol{\gamma}$ [55]. To solve the original optimization problem, we need to minimize $g(\eta, \boldsymbol{\gamma}; \mathcal{D})$ such that $\eta > 0$ [9], hence, we have to solve another constrained optimization problem, which is, however, much easier to solve. We can use any solver for such problems, e.g., the interior point algorithm. For an efficient optimization of the dual, also the corresponding gradients of the dual are required. They are given by

$$\begin{aligned}
\frac{\partial g(\eta, \boldsymbol{\gamma})}{\partial \eta} &= \epsilon + \log \frac{1}{N}\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}) \\
&\quad - \frac{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]})(\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]} - \boldsymbol{\gamma}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]}))}{\eta \sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]})}, \qquad \text{(A.11)} \\
\frac{\partial g(\eta, \boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}} &= \hat{\boldsymbol{\phi}} - \frac{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}) \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]})}, \qquad \text{(A.12)} \\
Z(\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}) &= \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]} - \boldsymbol{\gamma}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right).
\end{aligned}$$

# Appendix B

# Probabilistic Model Learning for Trajectory Prediction

The model learning problem can be formulated as

$$
\begin{aligned}
\min_{p_t,\boldsymbol{\theta}} \quad & \sum_{t=1}^{T} \int_{\boldsymbol{x}} p_t(\boldsymbol{x}) \log \frac{p_t(\boldsymbol{x})}{q_t(\boldsymbol{x})} d\boldsymbol{x}, \\
s.t.: \quad & \int_{\boldsymbol{x}} p_1(\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}) d\boldsymbol{x} = \hat{\boldsymbol{\phi}}(\boldsymbol{x}), \\
& \int_{\boldsymbol{x}} p_t(\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}) d\boldsymbol{x} = \int_{\boldsymbol{x}} p_{t-1}(\boldsymbol{x}) \int_{\boldsymbol{u}} \pi(\boldsymbol{u}|\boldsymbol{x}) \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}] d\boldsymbol{u} d\boldsymbol{x}, \ \forall t > 1, \\
& \int_{\boldsymbol{x}} p_t(\boldsymbol{x}) d\boldsymbol{x} = 1, \ \forall t,
\end{aligned}
$$

where $\boldsymbol{\theta}$ is the model parameter we wish to optimize, $\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]$ is the expected feature of the next state with model $p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta})$ and $q_t(\boldsymbol{x})$ is the observed state distribution at time step $t$. We write up the Lagrangian

$$\begin{aligned}\mathcal{L}(p_{1:T},\lambda_{1:T},\boldsymbol{\gamma}_{1:T},\boldsymbol{\theta}) &= \sum_{t=1}^{T}\int_{\boldsymbol{x}} p_t(\boldsymbol{x})\log\frac{p_t(\boldsymbol{x})}{q_t(\boldsymbol{x})}d\boldsymbol{x}+\boldsymbol{\gamma}_1^T\left(\hat{\boldsymbol{\phi}}(\boldsymbol{x})-\int_{\boldsymbol{x}} p_1(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x}\right)\\ &+\sum_{t=2}^{T}\boldsymbol{\gamma}_t^T\left(\int_{\boldsymbol{x}} p_{t-1}(\boldsymbol{x})\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]d\boldsymbol{x}-\int_{\boldsymbol{x}} p_t(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x}\right)\\ &+\sum_{t=1}^{T}\lambda_t\left(\int_{\boldsymbol{x}} p_t(\boldsymbol{x})d\boldsymbol{x}-1\right).\end{aligned}$$

We simplify and obtain

$$\begin{aligned}\mathcal{L}(p_{1:T},\lambda_{1:T},\boldsymbol{\gamma}_{1:T},\boldsymbol{\theta}) &= \sum_{t=1}^{T}\int_{\boldsymbol{x}} p_t(\boldsymbol{x})\Big(\log\frac{p_t(\boldsymbol{x})}{q_t(\boldsymbol{x})}+\lambda_t\\ &+\boldsymbol{\gamma}_{t+1}^T\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]-\boldsymbol{\gamma}_t^T\boldsymbol{\phi}(\boldsymbol{x})\Big)d\boldsymbol{x}-\lambda_t\\ &+\boldsymbol{\gamma}_1^T\hat{\boldsymbol{\phi}}(\boldsymbol{x})-\boldsymbol{\gamma}_{T+1}^T\int_{\boldsymbol{x}} p_T(\boldsymbol{x})\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}]d\boldsymbol{x}.\end{aligned}$$

We derive the gradients of the Lagrangian w.r.t. the distributions $p_t(\boldsymbol{x}),\ \forall t$ and set them to zero. This results in the following distributions

$$\begin{aligned}p_t(\boldsymbol{x}) &= q_t(\boldsymbol{x})\exp(\boldsymbol{\gamma}_t^T\boldsymbol{\phi}(\boldsymbol{x})-\boldsymbol{\gamma}_{t+1}^T\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}])\exp(-1-\lambda_t),\ t<T,\\ p_T(\boldsymbol{x}) &= q_T(\boldsymbol{x})\exp(\boldsymbol{\gamma}_T^T\boldsymbol{\phi}(\boldsymbol{x}))\exp(-1-\lambda_T).\end{aligned}$$

Note that the terms $\exp(-1-\lambda_t)$ can be obtained easily from the equation $\int_{\boldsymbol{x}} p_t(\boldsymbol{x})d\boldsymbol{x}=1$, that is,

$$\exp(-1-\lambda_t)=\left[\int_{\boldsymbol{x}} q_t(\boldsymbol{x})\exp(\boldsymbol{\gamma}_t^T\boldsymbol{\phi}(\boldsymbol{x})-\boldsymbol{\gamma}_{t+1}^T\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}');\boldsymbol{\theta}])d\boldsymbol{x}\right]^{-1},\ t<T.$$

Similarly we can compute $\exp(-1-\lambda_T)$ as well. The dual function is now given by

$$\begin{aligned} g(\lambda_{1:T}, \boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta}) &= \sum_{t=1}^{T} \frac{\int_{\boldsymbol{x}} Z_t(\boldsymbol{x})(-1)d\boldsymbol{x}}{\int_{\boldsymbol{x}} Z_t(\boldsymbol{x})d\boldsymbol{x}} - \lambda_t + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}), \\ Z_t(\boldsymbol{x}) &= q_t(\boldsymbol{x}) \exp(\boldsymbol{\gamma}_t^T \boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\gamma}_{t+1}^T \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}'); \boldsymbol{\theta}]), \ t < T, \qquad \text{(B.1)} \\ Z_T(\boldsymbol{x}) &= q_T(\boldsymbol{x}) \exp(\boldsymbol{\gamma}_T^T \boldsymbol{\phi}(\boldsymbol{x})). \qquad \text{(B.2)} \end{aligned}$$

After simplifying the equation, we obtain the solution

$$\begin{aligned} g(\lambda_{1:T}, \boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta}) &= \sum_{t=1}^{T} -1 - \lambda_t + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}) \\ &= \sum_{t=1}^{T} -\log\left[\exp(-1-\lambda_t)\right]^{-1} + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}). \end{aligned}$$

We now expand the dual function and we finally obtain

$$g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta}) = -\sum_{t=1}^{T} \log \int_{\boldsymbol{x}} Z_t(\boldsymbol{x})d\boldsymbol{x} + \boldsymbol{\gamma}_1^T \hat{\boldsymbol{\phi}}(\boldsymbol{x}). \qquad \text{(B.3)}$$

We compute the gradients of the dual function w.r.t. its parameters

$$\begin{aligned} \frac{\partial g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta})}{\partial \boldsymbol{\gamma}_1} &= \hat{\boldsymbol{\phi}}(\boldsymbol{x}) - \frac{\int_{\boldsymbol{x}} Z_1(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x}}{\int_{\boldsymbol{x}} Z_1(\boldsymbol{x})d\boldsymbol{x}}, \qquad \text{(B.4)} \\ \frac{\partial g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta})}{\partial \boldsymbol{\gamma}_t} &= \frac{\int_{\boldsymbol{x}} Z_{t-1}(\boldsymbol{x})\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}'); \boldsymbol{\theta}]d\boldsymbol{x}}{\int_{\boldsymbol{x}} Z_{t-1}(\boldsymbol{x})d\boldsymbol{x}} - \frac{\int_{\boldsymbol{x}} Z_t(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x}}{\int_{\boldsymbol{x}} Z_t(\boldsymbol{x})d\boldsymbol{x}}, \ t > 1, \qquad \text{(B.5)} \\ \frac{\partial g(\boldsymbol{\gamma}_{1:T}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \sum_{t=2}^{T} \frac{\int_{\boldsymbol{x}} Z_{t-1}(\boldsymbol{x})\boldsymbol{\gamma}_t^T \frac{\partial \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}'); \boldsymbol{\theta}]}{\partial \boldsymbol{\theta}} d\boldsymbol{x}}{\int_{\boldsymbol{x}} Z_{t-1}(\boldsymbol{x})d\boldsymbol{x}}, \qquad \text{(B.6)} \end{aligned}$$

In general, the model predicted feature gradient can be obtained by

$$\frac{\partial \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}'); \boldsymbol{\theta}]}{\partial \boldsymbol{\theta}} = \iint_{\boldsymbol{x}'\boldsymbol{u}} \frac{\partial p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \pi(\boldsymbol{u}|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}')d\boldsymbol{u}d\boldsymbol{x}'. \qquad \text{(B.7)}$$

In case the model predictive distribution is Gaussian, that is,

$$p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}'|\boldsymbol{\mu}(\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta}),\boldsymbol{\Sigma}(\boldsymbol{x},\boldsymbol{u};\boldsymbol{\theta})),$$

and if the feature vector contains linear and quadratic terms of the state, we can compute the gradients in closed form. The computation of the linear gradient terms is straightforward. For the quadratic terms we use the fact that $\mathbb{E}[X^2] = \mathbb{E}[X]^2 + \mathrm{Var}[X]$, where $X$ is a random variable. Therefore the gradient of the $i^{th}$ quadratic term is given by

$$\begin{aligned}\frac{\partial \mathbb{E}[\boldsymbol{x}_i'^2]}{\partial \boldsymbol{\theta}} &= \frac{\partial \boldsymbol{\mu}_i(\boldsymbol{\theta})^2}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\Sigma}_{ii}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= 2\boldsymbol{\mu}_i(\boldsymbol{\theta})\frac{\partial \boldsymbol{\mu}_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\Sigma}_{ii}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}.\end{aligned}$$

However, for more complex features, the gradient in Eq. (B.7) might be difficult to obtain in closed form. To apply the above parameter learning algorithm for a specific model, we have to compute the gradients of the predictive mean $\partial\mu(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ and variance $\partial\sigma^2(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ w.r.t. the model parameters $\boldsymbol{\theta}$.

## B.1 Gradients for Gaussian Process Models

In the following, we consider the Gaussian process model with the squared exponential kernel function Eq. (2.25). The predictive distribution for each output dimension can be obtained independently in closed form given the model parameters $\boldsymbol{\theta} = \{\boldsymbol{w}, \sigma_f, \sigma_\epsilon\}$ and the observed training data $\mathcal{D} = \{\boldsymbol{z}_i, \boldsymbol{x}_i'\}_{i=1}^N$, $\boldsymbol{z}_i = [\boldsymbol{x}_i^T, \boldsymbol{u}_i^T]^T$. For

more details we refer to Section 2.3. The gradients of the predictive mean and variance for query $\boldsymbol{z}_*$ are given by

$$\begin{aligned}
\frac{\partial \mu_*}{\partial \boldsymbol{\theta}} &= \frac{\partial \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)^T}{\partial \boldsymbol{\theta}} (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \boldsymbol{x}'_d \\
&\quad - \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)^T (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \frac{\partial (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})}{\partial \boldsymbol{\theta}} (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \boldsymbol{x}'_d, \qquad \text{(B.8)} \\
\frac{\partial \sigma_*^2}{\boldsymbol{\theta}} &= \frac{\partial k(\boldsymbol{z}_*, \boldsymbol{z}_*)}{\partial \boldsymbol{\theta}} - \frac{\partial \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)^T}{\partial \boldsymbol{\theta}} (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*) \\
&\quad + \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)^T (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \frac{\partial (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})}{\boldsymbol{\theta}} (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*) \\
&\quad - \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)^T (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \frac{\partial \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)}{\partial \boldsymbol{\theta}} + \frac{\partial \sigma_\epsilon^2}{\partial \boldsymbol{\theta}}. \qquad \text{(B.9)}
\end{aligned}$$

The gradients of the individual terms w.r.t. the hyper-parameters are given as

$$\begin{aligned}
\frac{\partial \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)}{\partial \sigma_\epsilon} &= \boldsymbol{0}, \\
\frac{\partial (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})}{\partial \sigma_\epsilon} &= 2\sigma_\epsilon \boldsymbol{I} \\
\frac{\partial \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*)}{\partial \sigma_f} &= \frac{2}{\sigma_f} \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}_*), \\
\frac{\partial (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})}{\partial \sigma_f} &= \frac{2}{\sigma_f} \boldsymbol{K}, \\
\frac{\partial k(\boldsymbol{z}^{[i]}, \boldsymbol{z}_*)}{\partial \boldsymbol{w}_d} &= k(\boldsymbol{z}^{[i]}, \boldsymbol{z}_*)(\boldsymbol{z}_d^{[i]} - \boldsymbol{z}_{*,d})^2 \boldsymbol{w}_d^{-3}, \\
\frac{\partial (\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{[i,j]}}{\partial \boldsymbol{w}_d} &= \boldsymbol{K}^{[i,j]} (\boldsymbol{z}_d^{[i]} - \boldsymbol{z}_d^{[j]})^2 \boldsymbol{w}_d^{-3}.
\end{aligned}$$

In practice, we often optimize the log-hyper-parameters $\log \boldsymbol{\theta}$ instead of the hyper-parameters $\boldsymbol{\theta}$ to avoid negative variances. The above computation still apply, but the

final gradients are computed now as

$$\begin{aligned}
\frac{\partial \mu(\boldsymbol{\theta})}{\partial \log \boldsymbol{\theta}} &= \frac{\partial \mu(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \log \boldsymbol{\theta}} = \frac{\partial \mu(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \mathrm{diag}(\boldsymbol{\theta}), \\
\frac{\partial \sigma^2(\boldsymbol{\theta})}{\partial \log \boldsymbol{\theta}} &= \frac{\partial \sigma^2(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \mathrm{diag}(\boldsymbol{\theta}).
\end{aligned}$$

# Appendix C

# Robot Learning Tasks

## C.1 The Robot Throwing Task

In this task, a 4-DoF robot has to learn to throw a ball at a target position, see Fig. C.1. The links of the planar 4-DoF robot have weights $\boldsymbol{m} = [17.5, 17.5, 26.5, 8.5]$ kg and lengths $\boldsymbol{l} = [0.5, 0.5, 1.0, 1.0]$ m respectively. Each joint of the robot is actuated. The goal of the robot is to throw a ball at a target position, which is defined by the context

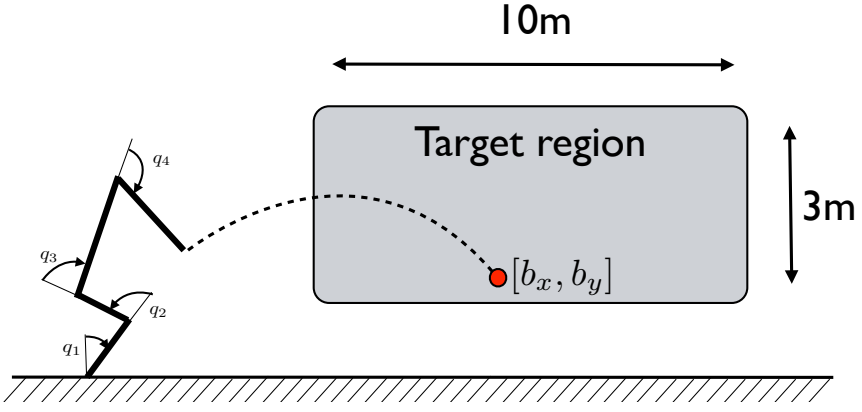



FIGURE C.1: The illustration of the ball throwing task.

$\boldsymbol{s} = [x, y]$. However, between each experiment the target position in varied uniformly in the $10 \times 3$m target region, $x \in [5, 15]$m and $y \in [0, 3]$m.

The reward function for the task is defined as

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -c_1 \min_t ||\boldsymbol{b}_t - \boldsymbol{s}||_2 - c_2 \sum_t f_c(\boldsymbol{x}_t) - c_3 \sum_t \boldsymbol{u}_t^T \boldsymbol{u}_t.$$

The first term punishes minimum distance of the ball trajectory $\boldsymbol{b} = [b_x, b_y]^T$ to the target position $\boldsymbol{s}$. We make the learning problem more challenging by penalizing joint angles that would be unrealistic for a human-like throwing motion. Thus, the second term describes a punishment term to force the robot to stay in given joint limits such that a human-like throwing motion is learned. The joint angle and angular velocity limits are defined as

$$\boldsymbol{q}_l = [-0.8, -2.5, -0.1, 0]^T \text{rad}, \quad \dot{\boldsymbol{q}}_l = [-50, -50, -50, -50]^T \text{rad/s},$$
$$\boldsymbol{q}_u = [0.8, 0.05, 2, 1.5\pi]^T \text{rad}, \quad \dot{\boldsymbol{q}}_u = [50, 50, 50, 50]^T \text{rad/s},$$

where with the lower indices $l$ and $u$, we refer to "lower" and "upper". We define the lower and upper state limit as $\boldsymbol{x}_l = [\boldsymbol{q}_l^T, \dot{\boldsymbol{q}}_l^T]^T$ and $\boldsymbol{x}_u = [\boldsymbol{q}_u^T, \dot{\boldsymbol{q}}_u^T]^T$. The penalty term can now be defined as

$$f_c(\boldsymbol{x}_t) = (\boldsymbol{x}_l - \boldsymbol{x}_t)^T \mathbb{I}_l (\boldsymbol{x}_l - \boldsymbol{x}_t) + (\boldsymbol{x}_u - \boldsymbol{x}_t)^T \mathbb{I}_u (\boldsymbol{x}_u - \boldsymbol{x}_t),$$

where $\mathbb{I}_l$ and $\mathbb{I}_u$ are the diagonal weighting matrices, where the diagonal elements take the value 0 if the joint limit constraints are not violated and 1 if the joint limits

are violated

$$\mathbb{I}^{l}_{i,i} = \begin{cases} 0, & \text{if } \boldsymbol{x}_{t,i} > \boldsymbol{x}_{l,i}, \\ 1, & \text{if } \boldsymbol{x}_{t,i} \leq \boldsymbol{x}_{l,i}, \end{cases} \quad \mathbb{I}^{u}_{i,i} = \begin{cases} 0, & \text{if } \boldsymbol{x}_{t,i} < \boldsymbol{x}_{u,i}, \\ 1, & \text{if } \boldsymbol{x}_{t,i} \geq \boldsymbol{x}_{u,i}. \end{cases}$$

The last term of the reward function penalizes high energy solutions. In our experiment we set the reward weighting factors to $c_1 = 10^2$, $c_2 = 10^3$ and $c_3 = 10^{-8}$.

This task was invented to test episode-based contextual policy search algorithms. As we are not able to define an immediate reward function due to the first term, the problem cannot be solved with step-based PS methods.

## C.2 The Robot Hockey Task

In this task we learn a robot hockey game using the KUKA lightweight robot arm in Fig. C.2. The goal of the robot is to shoot a hockey puck using the attached hockey racket to move a target puck, which is located at a certain distance. The robot can only move the target puck by hitting it with another puck, which we denote as control puck. The initial position of the control puck is fixed, but the position of the target puck $[b_x, b_y]^T$ is varied in both dimensions between experiments. As an additional goal, we require the displacement of the target puck

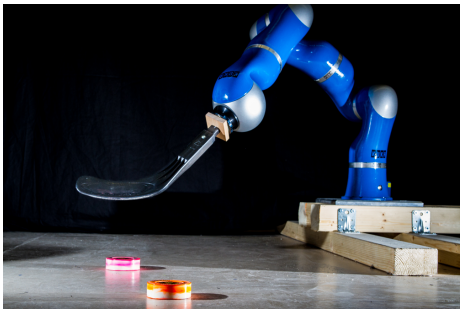


FIGURE C.2: The KUKA lightweight robot arm.

$d_t$ to be as close as possible to the desired distance $d^*$. We also vary the distance $d^*$ between experiments. Thus, the robot not only has to learn to shoot the target puck in the direction of the target puck, but also with the appropriate force. The simulated hockey task is depicted in Fig. 3.4. The context variable is defined as $\boldsymbol{s} = [b_x, b_y, d^*]^T$.

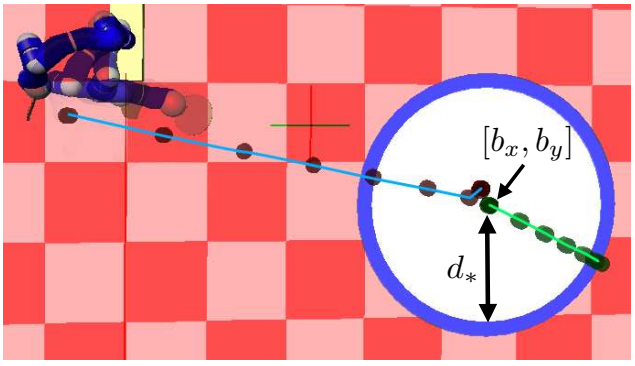



FIGURE C.3: The illustration of the robot hockey task.

We chose the initial position of the target puck to be uniformly distributed from the robot's base with displacements $b_x \in [1.5, 2.5]$m and $b_y \in [0.5, 1]$m. The desired displacement context parameter $d^*$ is also uniformly distributed $d^* \in [0, 1]$m. The reward function is defined as

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -\min_t ||\boldsymbol{x}_t - \boldsymbol{b}||_2 - ||d_T - d^*||_2,$$

which consist of two terms with equal weighting. The first term penalizes missing the target puck located at position $\boldsymbol{b} = [b_x, b_y]^T$, where the control puck trajectory is $\boldsymbol{x}_{1:T}$. The second term penalizes the error in the desired displacement of the target puck, where $d_T$ is the resulting displacement of the target puck after the shot.

We also evaluated the performance of learning algorithms on the hockey task using a real KUKA lightweight arm, see Fig. C.2. A Kinect sensor was used to track the position of the two pucks at a frame rate of 30Hz. We smoothed the trajectories in a pre-processing step with a Butterworth filter. We slightly changed the context variable ranges to meet the physical constraints of the test environment. We decreased the range of the position variables in both dimensions to $b_x \in [1.5, 2]$m and to $b_y \in [0.4, 0.8]$m from the robot's base. Furthermore, we decreased the desired distance range to $d^* \in [0, 0.6]$m. We kept the reward function unaltered.

The robot hockey task is designed to test contextual episode-based policy search algorithms. As the reward function requires the complete puck trajectories, we cannot define an immediate reward signal. Thus, step-based policy search algorithms cannot be applied for this task.

## C.3 The Robot Table Tennis Task

In this task, we learn hitting strokes in a table tennis game with a simulated Biorob [43] arm (Fig. C.4). The robot is mounted on two linear axis for moving in the horizontal plane. The robot itself has rotational joints, resulting in 8 actuated joints. A racket is mounted at the endeffector of the robot.

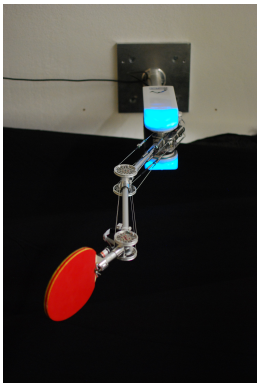


FIGURE C.4: The BioRob.

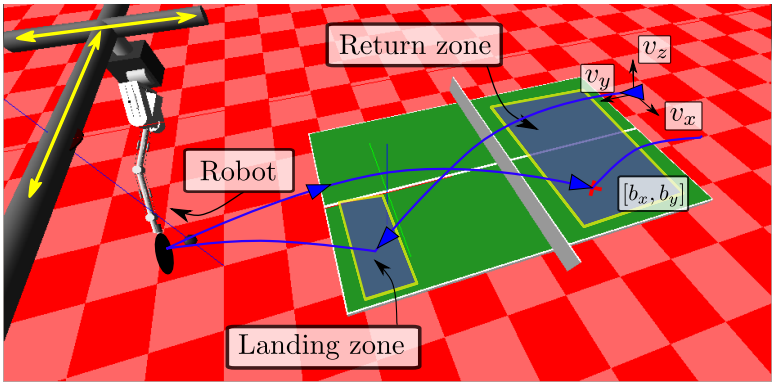



FIGURE C.5: The table tennis learning setup.

The Biorob is a lightweight tendon-driven robot arm that. Due to its small weight, it can perform highly dynamic movements. The simulated robot can be seen in Figure C.5. The construction of the real robot platform is ongoing work. In simulation, we modeled the ball with a standard ballistic flight model and air drag, but neglected simulating the spin or measurement noise. The goal of the robot is to return the incoming ball at a target position on the opponent's side of the table. However, the incoming ball has a changing initial velocity $\boldsymbol{v}$ and the return target position $\boldsymbol{b}$ is also varied uniformly on the opponent's side of the table. For an illustration of the task see Fig. C.5. We chose the range of the initial velocities such that the incoming ball bounces only once inside the *Landing zone*. We needed this simplification as the robot has a limited movement range in the $y$ direction (Fig. C.5). The goal of the robot is to hit the incoming ball back to the return position $\boldsymbol{b} = [b_x, b_y]^T$, which is distributed uniformly inside the *Return zone*. The context is defined as $\boldsymbol{s} = [v_x, v_y, v_z, b_x, b_y]^T$.

The reward function is defined by the sum of penalties for missing the ball and missing the target return position

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -c_1 \min ||\boldsymbol{\tau}_b - \boldsymbol{\tau}_r||_2 - c_2 ||\boldsymbol{b} - \boldsymbol{p}||_2 \tag{C.1}$$

where $\boldsymbol{c} = [c_1, c_2]^T$ are weighting parameters, $\boldsymbol{\tau}_b$ and $\boldsymbol{\tau}_r$ is the incoming ball and the racket trajectories, while $\boldsymbol{b}$ is the target and $\boldsymbol{p}$ is the returned ball landing position. The weighting parameters $c_1$ and $c_2$ are learning algorithm specific. We discuss their setting in the experimental sections.

This task is designed to test contextual episode-based PS algorithms. Due to the structure of the reward function, we require the whole experiment to be finished before we can evaluate it. We cannot define an immediate reward function that accurately captures our objectives for the learning task. Thus, step-based RL algorithms are not applicable for this task.

# Appendix D

# Publication List

[1] **A. Kupcsik**, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-Efficient Contextual Policy Search for Robot Movement Skills," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2013.

[2] G. Neumann, **A. Kupcsik**, M. Deisenroth, and J. Peters, "Information-theoretic motor skill learning," in *Proceedings of the AAAI 2013 Workshop on Intelligent Robotic Systems,* 2013.

[3] **A.G. Kupcsik**, M.P. Deisenroth, J. Peters, L. Ai Poh, P. Vadakkepat, G. Neumann, (conditionally accepted). *Model-based Contextual Policy Search for Data-Efficient Generalization of Robot Skills*, Artificial Intelligence

[4] **A. Kupcsik**, L. Ai Poh, P. Vadakkepat, J. Peters, G. Neumann (submitted). *Kernel Embedding of Trajectory Distributions for Model-based Policy Search*

# Bibliography


[1] ABBEEL, P., GANAPATHI, V., AND NG, A. Y. Learning vehicular dynamics, with application to modeling helicopters. In *Advances in Neural Information Processing Systems (NIPS)* (2005).

[2] ABBEEL, P., AND NG, A. Y. Learning first-order markov models for control. In *Advances in Neural Information Processing Systems (NIPS)* (2004).

[3] ABBEEL, P., QUIGLEY, M., AND NG, A. Y. Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)* (2006).

[4] ATKESON, C. G., AND SANTAMARÍA, J. C. A Comparison of Direct and Model-Based Reinforcement Learning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (1997).

[5] BAGNELL, A. D., AND SCHNEIDER, J. Policy search in reproducing kernel hilbert space. Tech. rep., Robotics Institute, Pittsburgh, PA, 2003.

[6] BAGNELL, J. A., AND SCHNEIDER, J. G. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2001).

[7] BAXTER, J., AND BARTLETT, P. Reinforcement Learning in POMDP's via Direct Gradient Ascent. In *Proceedings of the 17th Intl. Conference on Machine Learning (ICML)* (2000), pp. 41–48.

[8] BERTSEKAS, D. P. *Dynamic Programming and Optimal Control*, 2nd ed. Athena Scientific, 2000.

[9] BOYD, S., AND VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, 2004.

[10] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[11] CSATÓ, L., AND OPPER, M. Sparse online gaussian processes. *Neural Computation 14* (2002), 641–668.

[12] DA SILVA, B., KONIDARIS, G., AND BARTO, A. Learning Parameterized Skills. In *Proceedings of the International Conference on Machine Learning (ICML)* (June 2012).

[13] DANIEL, C., NEUMANN, G., KROEMER, O., AND PETERS, J. Learning sequential motor tasks. In *Proceedings of 2013 IEEE International Conference on Robotics andAutomation (ICRA)* (2013).

[14] DANIEL, C., NEUMANN, G., AND PETERS, J. Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2012).

[15] DEISENROTH, M. P., HUBER, M. F., AND HANEBECK, U. D. Analytic Moment-Based Gaussian Process Filtering. In *Proceedings of the International Conference on Machine Learning (ICML)* (2009).

[16] DEISENROTH, M. P., NEUMANN, G., AND PETERS, J. A survey on policy search for robotics. *Foundations and Trends in Robotics 2*, 1-2 (2013), 1–142.

[17] DEISENROTH, M. P., AND RASMUSSEN, C. E. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the International Conference on Machine Learning (ICML)* (2011).

[18] DEISENROTH, M. P., RASMUSSEN, C. E., AND FOX, D. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Robotics: Science & Systems (RSS)* (2011).

[19] DEISENROTH, M. P., TURNER, R., HUBER, M., HANEBECK, U. D., AND RASMUSSEN, C. E. Robust Filtering and Smoothing with Gaussian Processes. *IEEE Transactions on Automatic Control 57*, 7 (2012), 1865–1871.

[20] ENGEL, Y., MANNOR, S., AND MEIR, R. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (2005).

[21] ENGLERT, P., PARASCHOS, A., PETERS, J., AND DEISENROTH, M. P. Model-based Imitation Learning by Probabilistic Trajectory Matching. In *Proceedings of 2013 IEEE International Conference on Robotics and Automation (ICRA)* (2013).

[22] FORTE, D., GAMS, A., MORIMOTO, J., AND UDE, A. On-line motion synthesis and adaptation using a trajectorydatabase. *Robotics and Autonomous Systems 60*, 10 (2012), 1327–1339.

[23] FUKUMIZU, K., SONG, L., AND GRETTON, A. Kernel bayes' rule. In *Advances in Neural Information Processing Systems (NIPS)* (2011), pp. 1737–1745.

[24] GAMS, A., AND UDE, A. Generalization of Example Movements with Dynamic Systems. In *International Conference on Humanoid Robots (Humanoids)* (2009), IEEE, pp. 28–33.

[25] GENG, T., PORR, B., AND WÖRGÖTTER, F. Fast biped walking with a reflexive controller and real-time policy searching. In *Advances in Neural Information Processing Systems (NIPS)* (2005).

[26] GRÜNEWÄLDER, S., LEVER, G., BALDASSARRE, L., PONTIL, M., AND GRETTON, A. Modelling transition dynamics in mdps with rkhsembeddings. In *Proceedings of the International Conference on Machine Learning (ICML)* (2012).

[27] GRÜNEWÄLDER, S., LEVER, G., GRETTON, A., BALDASSARRE, L., PATTERSON, S., AND PONTIL, M. Conditional mean embeddings as regressors. In *Proceedings of the International Conference on Machine Learning (ICML)* (2012).

[28] HACHIYA, H., PETERS, J., AND SUGIYAMA, M. Reward-weighted regression with sample reuse for direct policy search in reinforcement learning. *Neural Computation 23*, 11 (2011), 2798–2832.

[29] HANSEN, N., MÜLLER, S. D., AND KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evol. Comput. 11*, 1 (Mar. 2003), 1–18.

[30] HEIDRICH-MEISNER, V., AND IGEL, C. Neuroevolution Strategies for Episodic Reinforcement Learning. *Journal of Algorithms 64*, 4 (oct 2009), 152–168.

[31] IJSPEERT, A. J., AND SCHAAL, S. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems (NIPS)*. 2003.

[32] KO, J., AND KLEIN, D. Gaussian processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2007).

[33] KOBER, J., BAGNELL, D., AND PETERS, J. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research* (2013).

[34] KOBER, J., MÜLLING, K., KROEMER, O., LAMPERT, C. H., SCHÖLKOPF, B., AND PETERS, J. Movement Templates for Learning of Hitting and Batting. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2010).

[35] KOBER, J., OZTOP, E., AND PETERS, J. Reinforcement Learning to adjust Robot Movements to New Situations. In *Robotics: Science & Systems (RSS)* (2010).

[36] KOBER, J., AND PETERS, J. Policy Search for Motor Primitives in Robotics. *Machine Learning* (2010), 1–33.

[37] KOHL, N., AND STONE, P. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2003).

[38] KOLLER, D., AND FRIEDMAN, N. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[39] KORMUSHEV, P., CALINON, S., AND CALDWELL, D. G. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)* (2010).

[40] KUINDERSMA, S., GRUPEN, R., AND BARTO, A. Learning dynamic arm motions for postural recovery. In *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots* (Bled, Slovenia, October 2011), pp. 7–12.

[41] KUPCSIK, A., DEISENROTH, M., PETERS, J., AI POH, L., VADAKKEPAT, V., AND NEUMANN, G. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence* (conditionally accepted).

[42] KUPCSIK, A., DEISENROTH, M. P., PETERS, J., AND NEUMANN, G. Data-Efficient Contextual Policy Search for Robot Movement Skills. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (2013).

[43] LENS, T. *Physical Human-Robot Interaction with a Lightweight, Elastic Tendon Driven Robotic Arm: Modeling, Control, and Safety Analysis*. PhD thesis, TU Darmstadt, Department of Computer Science, July 4 2012.

[44] MATSUBARA, T., HYON, S.-H., AND MORIMOTO, J. Learning parametric dynamic movement primitives from multiple demonstrations. In *Proceedings of*

*the 17th International Conference on Neural Information Processing: Theory and Algorithms - Volume Part I* (Berlin, Heidelberg, 2010), ICONIP'10, Springer-Verlag, pp. 347–354.

[45] MUELLING, K., KOBER, J., KROEMER, O., AND PETERS, J. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research (IJRR)*, 3 (2013), 263–279.

[46] MUNOS, R., AND LITTMAN, M. Policy gradient in continuous time. *Journal of Machine Learning Research 7* (2006), 771–791.

[47] NEUMANN, G. Variational Inference for Policy Search in Changing Situations. In *Proceedings of the International Conference on Machine Learning (ICML)* (2011).

[48] NEUMANN, G., KUPCSIK, A., DEISENROTH, M., AND PETERS, J. Information-theoretic motor skill learning. In *Proceedings of the AAAI 2013 Workshop on Intelligent Robotic Systems* (2013).

[49] NG, A. Y., KIM, H. J., JORDAN, M. I., AND SASTRY, S. Inverted Autonomous Helicopter Flight via Reinforcement Learning. In *In International Symposium on Experimental Robotics* (2004), MIT Press.

[50] NGUYEN-TUONG, D., AND PETERS, J. Incremental online sparsification for model learning in real-time robot control. *Neurocomputing 74*, 11 (2011), 1859–1867.

[51] NGUYEN-TUONG, D., AND PETERS, J. Model learning in robotics: a survey. *Cognitive Processing*, 4 (2011).

[52] NGUYEN-TUONG, D., SEEGER, M. W., AND PETERS, J. Real-time local gp model learning. In *From Motor Learning to Interaction Learning in Robots*. 2010, pp. 193–207.

[53] PARASCHOS, A., DANIEL, C., PETERS, J., AND NEUMANN, G. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS), Cambridge, MA: MIT Press.* (2013).

[54] PETELIN, D., AND KOCIJAN, J. Control systems with evolving gaussian process models. In *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)* (2011), pp. 178 – 184.

[55] PETERS, J., MÜLLING, K., AND ALTUN, Y. Relative Entropy Policy Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (2010).

[56] PETERS, J., AND SCHAAL, S. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine Learning (ICML)* (2007).

[57] PETERS, J., AND SCHAAL, S. Natural actor-critic. *Neurocomputing 71*, 7-9 (2008), 1180–1190.

[58] PETERS, J., AND SCHAAL, S. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 4 (2008), 682–97.

[59] QUIÑONERO CANDELA, J., AND RASMUSSEN, C. E. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res. 6* (Dec. 2005), 1939–1959.

[60] QUIÑONERO-CANDELA, J., GIRARD, A., LARSEN, J., AND RASMUSSEN, C. E. Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting. In *International Conference on Acoustics, Speech and Signal Processing* (2003), vol. 2, pp. 701–704.

[61] RASMUSSEN, C. E., AND KUSS, M. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)* (2004).

[62] RASMUSSEN, C. E., AND WILLIAMS, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[63] RIEDMILLER, M., PETERS, J., AND SCHAAL, S. Evaluation of policy gradient methods and variants on the cart-pole benchmark.

[64] RÜCKSTIESS, T., FELDER, M., AND SCHMIDHUBER, J. State-Dependent Exploration for policy gradient methods. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2008, Part II, LNAI 5212* (2008), pp. 234–249.

[65] RÜCKSTIESS, T., SEHNKE, F., T., S., WIERSTRA, D., YI, S., AND J., S. Exploring Parameter Space in Reinforcement Learning. *PALADYN Journal of Behavioral Robotics 1*, 1 (2010), 14 – 24.

[66] SCHAAL, S. The sl simulation and real-time control software package. Tech. rep., University of Southern California, 2009.

[67] SCHAAL, S., PETERS, J., NAKANISHI, J., AND IJSPEERT, A. J. Learning Movement Primitives. In *International Symposium on Robotics Research (ISRR)* (2003).

[68] SCHNEIDER, J. G. Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *Advances in Neural Information Processing Systems (NIPS)*. 1997.

[69] SCHÖLKOPF, B., AND SMOLA, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[70] SEEGER, M. Low Rank Updates for the Cholesky Decomposition. Tech. rep., 2007.

[71] SEHNKE, F., OSENDORFER, C., RÜCKSTIESS, T., GRAVES, A., PETERS, J., AND SCHMIEDHUBER, J. Parameter-Exploring Policy Gradients. *Neural Networks 23* (2010), 551–559.

[72] SNELSON, E., AND GHAHRAMANI, Z. Sparse Gaussian Processes using Pseudo-Inputs. In *Advances in Neural Information Processing Systems (NIPS)* (2006).

[73] SONG, L., BOOTS, B., SIDDIQI, S. M., GORDON, G. J., AND SMOLA, A. J. Hilbert space embeddings of hidden markov models. In *Proceedings of the International Conference on Machine Learning (ICML)* (2010), pp. 991–998.

[74] SONG, L., FUKUMIZU, K., AND GRETTON, A. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *IEEE Signal Process. Mag. 30*, 4 (2013), 98–111.

[75] SONG, L., GRETTON, A., BICKSON, D., LOW, Y., AND GUESTRIN, C. Kernel belief propagation. In *In Artificial Intelligence and Statistics (AISTATS)* (2011).

[76] SONG, L., HUANG, J., SMOLA, A. J., AND FUKUMIZU, K. Hilbert space embeddings of conditional distributionswith applications to dynamical systems. In *Proceedings of the International Conference on Machine Learning (ICML)* (2009), vol. 382 of *ACM International Conference Proceeding Series*, ACM, p. 121.

[77] STULP, F., RAIOLA, G., HOARAU, A., IVALDI, S., AND SIGAUD, O. Learning compact parameterized skills with a single regression. In *accepted for IEEE-RAS International Conference on Humanoid Robots* (2013).

[78] STULP, F., AND SIGAUD, O. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the International Conference on Machine Learning (ICML)* (2012).

[79] SUTTON, R., MCALLESTER, D., SINGH, S., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems (NIPS) 12* (2000), 1057–1063.

[80] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, Mar. 1998.

[81] SZEPESVÁRI, C. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[82] TANG, J., AND ABBEEL, P. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems (NIPS)* (2010), pp. 1000–1008.

[83] TAYLOR, G., AND PARR, R. Kernelized Value Function Approximation for Reinforcement Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)* (2009), pp. 1017–1024.

[84] THEODOROU, E., BUCHLI, J., AND SCHAAL, S. Reinforcement Learning of Motor Skills in High Dimensions: a Path Integral Approach. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2010).

[85] TITSIAS, M. Variational learning of inducing variables in sparse gaussian processes. *Journal of Machine Learning Research - Proceedings Track 5* (2009), 567–574.

[86] UDE, A., GAMS, A., ASFOUR, T., AND MORIMOTO, J. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transaction on Robotics 26*, 5 (2010), 800–815.

[87] WIERSTRA, D., SCHAUL, T., PETERS, J., AND SCHMIDHUBER, J. Fitness Expectation Maximization. In *Lectue Notes in Computer Science, Parallel Problem Solving from Nature, PPSN X* (2008), Springer-Verlag, pp. 337–346.

[88] WILLIAMS, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning 8* (1992), 229–256.

[89] XU, X., XIE, T., HU, D., AND LU, X. Kernel least-squares temporal difference learning. *Intenational Journal of Information Technology* (2005).

[90] YI, S., WIERSTRA, D., SCHAUL, T., AND SCHMIDHUBER, J. Stochastic Search using the Natural Gradient. In *Proceedings of the 26th International Conference On Machine Learning (ICML)* (2009), pp. 1161 – 1168.