Model-based Contextual Policy Search for Data-Efficient Generalization of Robot Skills

Andras Kupcsik[a,b], Marc Peter Deisenroth[e], Jan Peters[c,d], Loh Ai Poh[a], Prahlad Vadakkepat[a], Gerhard Neumann[c]

[a]National University of Singapore, Department of Electrical and Computer Engineering, 4 Engineering Drive 3, Singapore 118571
[b]National University of Singapore, School of Computing, 13 Computing Drive, Singapore 117417
[c]Technische Universität Darmstadt, Fachbereich Informatik, Fachgebiet Intelligente Autonome Systeme, Hochschulstr. 10, D-64289 Darmstadt
[d] Max-Planck Institute for Intelligent Systems, Spemannstrasse 38, 72076 Tübingen
[e]Imperial College London, Department of Computing, 180 Queen's Gate, London SW7 2AZ

Abstract

In robotics, lower-level controllers are typically used to make the robot solve a specific task in a fixed context. For example, the lower-level controller can encode a hitting movement while the context defines the target coordinates to hit. However, in many learning problems the context may change between task executions. To adapt the policy to a new context, we utilize a hierarchical approach by learning an upper-level policy that generalizes the lower-level controllers to new contexts. A common approach to learn such upper-level policies is to use policy search. However, the majority of current contextual policy search approaches are model-free and require a high number of interactions with the robot and its environment. Model-based approaches are known to significantly reduce the amount of robot experiments, however, current model-based techniques cannot be applied straightforwardly to the problem of learning contextual upper-level policies. They rely on specific parametrizations of the policy and the reward function, which are often unrealistic in the contextual policy search formulation. In this paper, we propose a novel model-based contextual policy search algorithm that is able to generalize lower-level controllers, and is data-efficient. Our approach is based on learned probabilistic forward models and information theoretic policy search. Unlike current algorithms, our method does not require any assumption on the parametrization of the policy or the reward function. We show on complex simulated robotic tasks and in a real robot experiment that the proposed learning framework speeds up the learning process by up to two orders of magnitude in comparison to existing methods, while learning high quality policies.

1. Introduction

Learning is a successful alternative to hand-designing robot controllers to solve complex tasks in robotics. Algorithms that learn such controllers need to take several important challenges into consideration. First, robots typically operate in high-dimensional continuous state-action spaces. Thus, the learning algorithm has to scale well to higher dimensional robot tasks. Second, running experiments with real robots typically has a high cost. An experiment rollout is time consuming, usually requires expert supervision and it might lead to robot damage. Thus, the learning algorithm is required to operate with a limited number of evaluations. Furthermore, when learning with real robots, safety becomes an important factor. To avoid robot and environmental damage, the learning algorithm has to provide robot controllers that generate robot trajectories close to the already explored and, therefore, safe trajectory space. Lastly, robot skills have to be able to adapt to changing environmental conditions. For example, if the task is defined as throwing a ball at varying target positions, the controller has to be adapted to the current target position. In the following, we will refer to such task variables as context $\boldsymbol{s}$. In the throwing example, the context is represented as the target position to throw to. In this paper, we introduce a new model-based policy search method to generalize

Email addresses: kupcsik@comp.nus.edu.sg (Andras Kupcsik), m.deisenroth@imperial.ac.uk (Marc Peter Deisenroth), peters@ias.tu-darmstadt.de (Jan Peters), elelohap@nus.edu.sg (Loh Ai Poh), prahlad@nus.edu.sg (Prahlad Vadakkepat), neumann@ias.tu-darmstadt.de (Gerhard Neumann)

a learned skill to a new context. For example, if we have learned to throw a ball to a specific location, we want to generalize this skill such that we can throw the ball to multiple locations.

Policy Search (PS) methods are one of the most successful Reinforcement Learning (RL) algorithms for learning complex movement tasks in robotics [31, 38, 22, 23, 21, 8, 30, 20, 27, 13]. PS algorithms typically optimize the parameters $\boldsymbol{\omega}$ of a parametrized control policy, which generates the control commands for the robot, such that the policy obtains maximum reward. A common approach to parametrize the policy is to use a compact representation of a movement with a moderate amount of parameters, such as movement primitives [17, 21]. In many approaches to movement primitives, the parameters $\boldsymbol{\omega}$ specify the shape of a desired trajectory. The policy is then defined as trajectory tracking controller that follows this desired trajectory. Such a desired trajectory, represented by a single parameter vector $\boldsymbol{\omega}$, can be used to solve one specific task, characterized by the context vector $\boldsymbol{s}$. The goal in contextual policy search is to learn how to choose the parameter vector $\boldsymbol{\omega}$ of the control policy as a function of the context $\boldsymbol{s}$. To do so, it is convenient to define two different levels of policies that are used in policy search. At the lower level, the control policy specifies the controls of the robot as a function of its state. The lower level policy is parametrized by the parameter vector $\boldsymbol{\omega}$. The lower-level policy can, for example, be implemented as movement primitive [17]. On the upper-level, a policy that chooses the parameters $\boldsymbol{\omega}$ of the lower-level policy is used. We will denote this policy as upper-level policy. Given the current task description $\boldsymbol{s}$, the upper-level policy chooses the parameters $\boldsymbol{\omega}$ of the lower-level policy. The lower level policy is subsequently executed with the given parameters $\boldsymbol{\omega}$ for the whole episode. Although PS algorithms can be applied to learn a large variety of robot skills, in this paper we focus on learning stroke-based movements, such as throwing, hitting, etc.

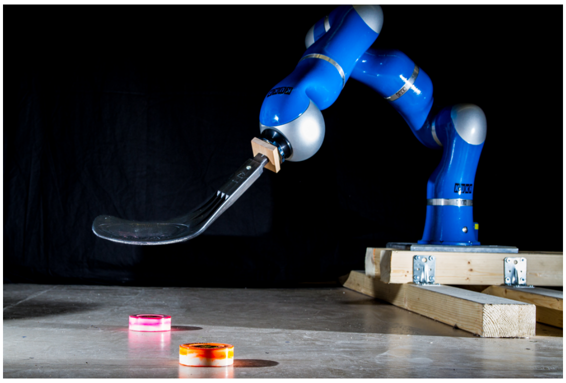


Figure 1: KUKA lightweight arm shooting hockey pucks.

Most of the existing contextual policy search methods are model-free [20, 27], i.e., they try to optimize the policy without estimating a model of the robot and the environment. Model-free PS algorithms execute rollouts on the real robot to evaluate parameter vectors $\boldsymbol{\omega}$. These evaluations are finally used to improve the policy. Most model-free PS algorithms require hundreds if not thousands of real robot interactions until converging to a high quality policy. For many robot learning problems, such data inefficiency is impractical, as executing real robot experiments is time consuming, requires expert supervision and it might lead to robot wear, or even robot damage. It has been shown that the data-efficiency of policy search methods can be considerably improved by learning forward models of the robot and its environment. These models are used to predict the experiment outcome, which allows more robust and efficient policy updates. We refer to such algorithms as model-based policy search algorithms [10, 1, 4, 34, 2, 18, 29]. However, current model-based policy search methods such as PILCO [9, 12] suffer from severe limitations that make it hard to apply these methods for learning generalized robot skills. PILCO uses computationally demanding deterministic approximate inference techniques that assume a specific structure of the reward function as well as of the used lower-level policy. These assumptions do not hold for many applications that occur in contextual policy search and have hindered the use of model-based policy search for learning contextual upper level policies. Moreover, the deterministic approximate inference method adds a bias in the prediction of the experiment outcome. Recently, the PILCO algorithm has been extended for learning generalized lower-level controllers [11]. Promising results have been demonstrated for learning robot controllers for hitting and box stacking tasks. Still, PILCO suffers from the restrictions on the structure of the used reward function and lower-level controllers.

In this paper we introduce a new model-based policy search method that relaxes these assumptions from current model-based approaches and can therefore be used to efficiently generalize lower level-robot control policies to new contexts $\boldsymbol{s}$. We rely on the contextual extension to Relative Entropy Policy Search (REPS), an information-theoretic PS algorithm [30]. REPS maximizes the expected reward of the upper-level policy, while staying close to the observed data, given by the parameter samples of the old upper-level policy. Our approach is to extend REPS to be a model-based method. Due to the closeness-bound, REPS is well suited for this extension as REPS will not explore areas of the parameter space where it has not seen data and the learned models are of poor quality. We learn probabilistic forward models of the robot and its environment and exploit expert knowledge about the task setup to decompose the model of the environment into several simpler dynamic and contact models. We use the learned forward models to generate artificial experiment outcomes, which we subsequently use for the policy updates. The benefit of using models is two-fold. First, by using artificial samples for policy updates, we significantly improve the data-efficiency of the learning framework. Second, we use models to compute the expected return and thus, avoiding the risk sensitive bias in the original REPS algorithm. We call the resulting algorithm Gaussian Process Relative Entropy Policy Search (GPREPS). We show in three complex simulated robotic tasks and in one real experiment that GPREPS reduces real robot interactions with two orders of magnitude, while learning policies of higher quality.

This paper is an extension to the results published in [24], with the novel contributions as follows. First, we present a more detailed description of the technical background and thoroughly discuss related work. Second, we present the full derivation of the contextual REPS algorithm. Third, we present a qualitative comparison of moment-matching and sampling using GP models in terms of prediction accuracy and computation times. Fourth, we compare the efficiency of the full and the sparse GP model for learning robot dynamics. Finally, we present a novel evaluation of GPREPS in a table tennis learning scenario with a simulated Biorob robot arm.

In the following, we present the problem formulation in Sec. 2. In Sec. 3, we discuss related work. In Sec. 4, we introduce the contextual extension to REPS and show how we can learn upper-level policies. In Sec. 5 we introduce the GPREPS algorithm and explain how model learning and trajectory prediction is integrated in the policy updates of contextual REPS. In Sec. 6, we show experimental results, while Sec. 7 concludes our work.

2. Problem Formulation

In this paper, we denote the state of the robot and its environment as $\boldsymbol{x}$. Typically, this state is composed of the joint angles $\boldsymbol{q} \in \mathbb{R}^d$ and joint velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^d$, where d is the number of degrees of freedom[1], but it can also contain external variables such as the position of a ball. The vector of control torques $\boldsymbol{u} \in \mathbb{R}^d$ is computed by the lower-level policy $\boldsymbol{u} = \pi(\boldsymbol{x}; \boldsymbol{\omega})$[2] parametrized by $\boldsymbol{\omega} \in \boldsymbol{\Omega}$. The space of lower-level policy parameters is denoted by $\boldsymbol{\Omega}$. A typical approach in policy search is to use simple parametrizations of the lower-level policy with a small number of parameters. Depending on the task, we can use linear feedback controllers, movement primitives [17] or torque profiles [28]. Such parametrizations are easier to learn as, for example, neural network controllers, however, they are usually limited to solve a single task. In order to generalize the lower-level controllers to different contexts $\boldsymbol{s}$, we need to choose different parameters $\boldsymbol{\omega}$ in each context.

The trajectory of the robot is defined as the set of state-action pairs at each time step of the episode of length T, $\boldsymbol{\tau} = \{\boldsymbol{x}_1, \boldsymbol{u}_1, \dots, \boldsymbol{x}_T, \boldsymbol{u}_T\}$. In our formulation, the upper-level policy is represented by a search distribution over $\boldsymbol{\Omega}$, $\pi(\boldsymbol{\omega})$. It is typically defined as a Gaussian $\pi(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_{\omega}, \boldsymbol{\Sigma}_{\omega})$.We consider the episode-based policy search framework [20, 27, 33] where search for the optimal upper-level policy by solving

$$\pi^*(\boldsymbol{\omega}) = \arg\max_{\pi} \int_{\boldsymbol{\Omega}} \pi(\boldsymbol{\omega}) \mathcal{R}_{\omega} d\boldsymbol{\omega}, \tag{1}$$

where the expected episode reward is denoted by $\mathcal{R}_{\omega} = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\omega}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$, $R(\boldsymbol{\tau})$ is the trajectory reward and $p(\boldsymbol{\tau}|\boldsymbol{\omega})$ is the probability of the trajectory given the parameters $\boldsymbol{\omega}$ of the low-level controller. In many cases, the reward function might be defined as the sum of immediate rewards $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$, $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{u}_t)$. However, some objectives can only be defined as the function of the whole trajectory.

As an example, consider a throwing task in Figure 2(a), where the robot has to throw a ball at a specific target position while maintaining balance. The robot executes the throwing motion using controller $\pi(\boldsymbol{x}; \boldsymbol{\omega})$ and releases the ball at a specific time point t_r. Then, we record the ball trajectory $\boldsymbol{\tau}_b$, which contains the ball position at each time

[1] In this paper we only assume rotational robot joints.

[2] In this paper, a deterministic lower-level controller is used. Alternatively, we can use a stochastic controller $\boldsymbol{u} \sim \pi(\boldsymbol{u}|\boldsymbol{x}; \boldsymbol{\omega})$, which is, however, typically not necessary as exploration is already implemented by the upper-level policy.

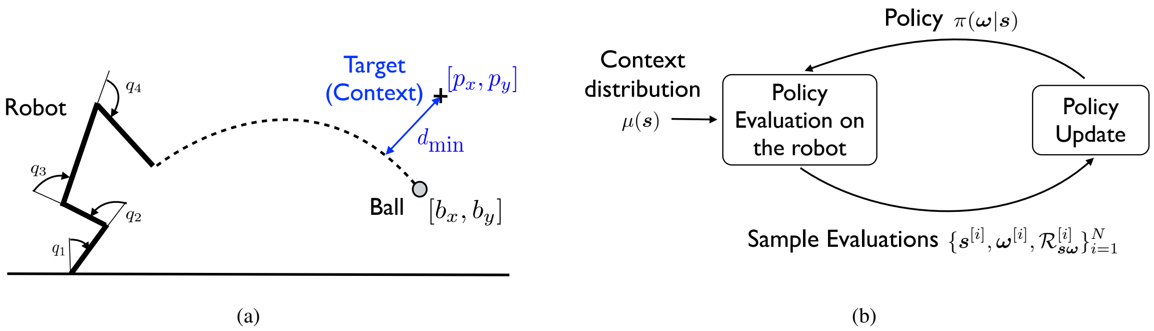

Robot
q_4
q_3
q_2
q_1
Target (Context)
$[p_x, p_y]$
$d_{\min}$
Ball
$[b_x, b_y]$
(a)
Policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$
Context distribution
$\mu(\boldsymbol{s})$
Policy Evaluation on the robot
Policy Update
Sample Evaluations $\{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, \mathcal{R}^{[i]}_{\boldsymbol{s\omega}}\}_{i=1}^{N}$
(b)

Figure 2: **(a)** The ball throwing task. The 4-link robot has to throw a ball at a target position $[p_x, p_y]$, while maintaining balance. **(b)** The learning framework of model-free contextual PS algorithms, such as the contextual REPS algorithm introduced in Section 4. We sample the lower-level policy parameter using the upper-level policy $\boldsymbol{\omega} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s})$ using the observed context $\boldsymbol{s}$. Subsequently, we evaluate $\boldsymbol{\omega}$ on the robot to obtain the reward. For each policy update we obtain N rollouts.

step $\boldsymbol{b}_t = [b_{x,t}, b_{y,t}]^T$. For the throwing task the reward function can be defined as the minimum distance of the ball trajectory to a target position $\boldsymbol{p}$, $R(\boldsymbol{\tau}) = -\min_t ||\boldsymbol{p} - \boldsymbol{b}_t||_2$. Additionally, we can include other objectives in the reward function, such as torque penalty $-\sum_{t=1}^{T} \boldsymbol{u}_t^T \boldsymbol{u}_t$, or deviation from a target configuration $-\sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{x}_g)^T (\boldsymbol{x}_t - \boldsymbol{x}_g)$, where $\boldsymbol{x}_g$ is the target state.

In contextual policy search [21, 27], our goal is to generalize the lower-level control policy $\boldsymbol{u} = \pi(\boldsymbol{x}; \boldsymbol{\omega})$ to multiple contexts. The context vector $\boldsymbol{s}$ for a learning problem is defined as the set of variables that fully specify the task. It typically contains the objectives of the agent, e.g., for the throwing task it might refer to the target position of the target, $\boldsymbol{s} = [p_x, p_y]^T$, but it can also contain properties of the environment, e.g., the weight of a mass that needs to be lifted. We assume that the continuous context variable is drawn from an unknown distribution $\boldsymbol{s} \sim \mu(\boldsymbol{s})$ in the beginning of an episode, and that the context variable is fully observable. The context distribution $\mu(\boldsymbol{s})$ is defined by the learning problem. Solving a contextual problem with the standard episode-based PS approach would require us to learn $\pi_{\boldsymbol{s}}(\boldsymbol{\omega})$ for each context $\boldsymbol{s}$, which is clearly a tedious and data inefficient approach.

Instead, we follow a hierarchical approach, where we learn an upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, which provides the lower-level controller parametrization $\boldsymbol{\omega}$ given the context $\boldsymbol{s}$. Our goal is to find the optimal policy $\pi^*(\boldsymbol{\omega}|\boldsymbol{s})$, such that it maximizes the expected reward

$$\pi^*(\boldsymbol{\omega}|\boldsymbol{s}) = \arg\max_{\pi} \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\omega}} \pi(\boldsymbol{\omega}|\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s\omega}} d\boldsymbol{\omega} d\boldsymbol{s}, \tag{2}$$

where $\mathcal{R}_{\boldsymbol{s\omega}}$ 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 its environment are stochastic, the reward $\mathcal{R}_{\boldsymbol{s\omega}}$ is given by the expected reward over all trajectories

$$\mathcal{R}_{\boldsymbol{s\omega}} = \mathbb{E}_{\boldsymbol{\tau}}[R(\boldsymbol{\tau}, \boldsymbol{s})|\boldsymbol{s}, \boldsymbol{\omega}] = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\omega}) R(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau}. \tag{3}$$

The probability of a trajectory $p(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\omega})$ now also depends on the context and the reward function $R(\boldsymbol{\tau}, \boldsymbol{s})$ now also depend on the context $\boldsymbol{s}$. Using our motivating example, for the throwing task, we might define the reward function as the minimal distance to the target that is now defined by the context, i.e., $R(\boldsymbol{\tau}, \boldsymbol{s}) = -\min_t ||\boldsymbol{s} - \boldsymbol{b}_t||_2$, where now the context defines the target position to throw to, $\boldsymbol{s} = [p_x, p_y]^T$. We also illustrate the basic concept of (model-free) contextual policy search in Figure 2(b). In the beginning of the episode, the lower-level policy parameter is drawn from $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ given the observed context variable $\boldsymbol{s} \sim \mu(\boldsymbol{s})$. Subsequently, the policy parameter is evaluated on the robot and the reward is obtained. We collect N sample rollouts, which we use then to update the policy.

3. Related Work

While policy search became highly successful in recent years, there are multiple approaches to acquire robot skills. Standard reinforcement learning tools that rely on the Markov Decision Process (MDP) framework share the same goal as policy search, that is, maximize the reward by interacting with the environment. However, one major disadvantage of the MDP formulation is that the number of states, and thus, the number of (action-)value functions grow exponentially with the state dimensionality. As robots typically operate in high dimensional continuous state-action spaces, standard RL techniques are difficult to apply with success.

The goal and methods of RL closely relate to that of optimal control. However, 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. While we can learn the models from data in the same way done in this paper, the optimal control solution can only be obtained for linear systems with Gaussian noise. For all other systems, optimal control has to rely on approximations that might lead to a poor quality of the optimal policy. Policy search avoids these approximations as we directly search in the policy parameter space and is therefore likely to produce policies of higher quality.

In the following we give a brief overview of related work in the fields of robot skill generalization, contextual model-free policy search methods and model-based policy search for robot learning.

Robot Skill Generalization. Robot skill generalization has been investigated by many researchers in recent years [40, 15, 7, 26, 20, 27]. In [15, 40], robot skills are represented by Dynamic Movement Primitives [17] or DMPs. 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. To account for skill improvement, policy search methods have been applied [20, 27]. For example, Kober et al. [20] proposed the Cost regularized Kernel Regression (CRKR) algorithm to learn DMP parameters for throwing a dart at different targets, and for robot table tennis. However, CRKR does not scale well to higher dimensional learning problems due to its uncorrelated exploration strategy. Another PS algorithm is proposed in [27] for robot skill generalization. The algorithm uses a probabilistic approach based in 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 is computationally very costly to obtain. Recently, the Mixture of Movement Primitives (MoMP) algorithm has been introduced [26] 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 depending how successful the movement primitives are in the given context. Promising results in real robot table tennis have been presented [26]. However, the used 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. Finally, a general robot skill learning framework has been proposed by da Silva et al. [7]. The proposed approach separates generalization and policy learning to a classification and a regression problem. The resulting policy is a mixture of finite number 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.

The contextual REPS algorithm presented in this paper is highly related to the hierarchical (Hi-)REPS algorithm presented in [8]. Hi-REPS is applied to learn multiple options to solve a given task. It has been used for learning versatile robot skills for the tetherball game. While the contextual REPS algorithm presented in this paper can be seen as special case of Hi-REPS with only one option and without state transitions, the Hi-REPS algorithm used in [8] is model-free and therefore needs many evaluations on the real robot system.

Model-free Policy Search Algorithms. The key concept of model-free policy search algorithms is to update the policy solely based on parameter-reward samples obtained using the real system, without any assumption, or knowledge about the system dynamics. Model-free policy search methods can be coarsely categorized according to the exploration and the policy update strategy. Exploration can be either implemented at the level of the actions, i.e., we add exploration noise to the controls at each time step, or, at the level of the parameters $\boldsymbol{\omega}$ of the control policy.

Some of the earliest successful policy search methods were policy gradient (PG) algorithms [42, 5, 37]. PG algorithms use the likelihood-ratio trick to compute the expected performance gradient. The policy parameters are updated using the gradient and a user defined learning rate. Some of the most important extensions to PG algorithm was the introduction of the natural policy gradient [31, 3]. The natural policy gradient bounds the relative entropy or Kullback-Leibler (KL) divergence of two subsequent policies by using a second order approximation of the KL divergence. Due to this bound, the algorithm achieves uniform convergence in the whole parameter space which typically results in an increased learning speed in comparison to standard PG algorithms. Natural policy gradient algorithms are currently one of the most commonly used PG algorithms. While the early PG algorithms were developed for action-based exploration, the ideas have been adopted to parameter-based exploration algorithms [43, 35, 33] as well. One significant challenge when using PG methods is the choice of an appropriate learning rate, which is crucial for good performance, but usually difficult to find.

An alternative approach to learn upper-level policies is to treat the policy search problem as a latent variable estimation problem and use Expectation Maximization to solve it [27, 20, 41, 21]. In episodic Monte-Carlo EM approaches [20, 41, 21], the new upper-level policy parameters are found by using the Moment-projection of the reward weighted old policy, which is essentially using a weighted maximum likelihood estimation for the new policy parameters. Alternatively, we can perform the Information-projection of the reward weighted old policy [27]. By doing so, we avoid the typical problem of MC-EM algorithms, that is, averaging over multiple modes of the reward weighted parameter distribution. However, when using I-projection, the new policy parameters cannot be computed in closed form for most policies [27]. The idea of avoiding failed and low quality experiments during exploration has also been investigated by Grollman and Billard [16]. They assume that the expert's demonstration of the task is unsuccessful, but gives us an idea how the solution should look like. Thus, the robot should try skills that that are not exactly the same as the failed demonstrations, but similar. The distribution of exploratory trajectories is encoded in a Donut Mixture Model (DMM), which can be regarded as the pseudo-inverse of a Gaussian Mixture Model [16] representing the failed trajectory distribution.

A significant advantage of EM-based PS methods compared to PG algorithms is that no user defined learning rate is required and that they can be applied to learn contextual upper-level policies [20]. To combine the uniform convergence property of the natural gradient approaches [31] and the closed form policy update of EM-based PS approaches, the information theoretic Relative Entropy Policy Search (REPS) algorithm has been proposed in [30]. The REPS algorithm limits the information loss between subsequent policies, while maximizing the expected reward, resulting in smooth convergence. The only required parameter for REPS is the upper bound on the information loss, which is significantly easier to choose than a learning rate with PG algorithms. We will investigate in details the contextual extension to REPS in Sec. 3.

Model-based Policy Search Methods. To improve the data-efficiency of model free methods, several model-based approaches have been proposed in the literature [9, 12, 34, 4]. Model-based methods learn the forward model of the controllable system and its environment, which is subsequently used in simulations to predict the experiment outcome. While in [1], a time dependent forward model is used, we typically learn the dynamic model of the real hardware [34, 4, 29, 10, 18, 14]. Time-dependent models fail to generalize to unseen situations, and only provide accurate models along the observed trajectories [1], however, they might be able to predict the system dynamics more accurately especially if the state of the system includes unobserved variables. Forward models are typically used to provide long term trajectory predictions. 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 $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ is i.i.d. Gaussian noise.

A common method to learn the non-linear models of the system dynamics is to use the Locally Weighted Bayesian Regression (LWBR) algorithm [4, 34, 29]. 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 every query point $\boldsymbol{y}_* = [\boldsymbol{x}_*^T, \boldsymbol{u}_*^T]^T$. LWBR has been applied to learn the forward model of a helicopter [4, 29] as well as the inverted pendulum [34].

An alternative successful method is to use Gaussian Process (GP) models that have proven to be efficient in learning a stochastic model of the dynamics [9, 10, 18, 14]. With GP models [32], we can compute the posterior 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 GP models integrate out the model uncertainty, the model becomes less biased. GP models also provide us with a variance of the prediction. The variance/uncertainty of the prediction typically decreases with the number of data points in the neighbourhood. Consequently, we can avoid overly confident predictions in unexplored state spaces. The state of the art model-based policy search algorithm is the Probabilistic Inference for Learning Control (PILCO) algorithm [9, 12], 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 the future rewards following the current control policy. However, computing the successor state distribution with GP models given a non-deterministic query input is not straightforward. PILCO solves this problem by matching the first and second moment of the predictive distribution. Using the approximated trajectory distribution and reward distribution, PILCO computes the gradient of the long term rewards w.r.t. the controller parameters in closed form. This process is repeated until the optimal policy is found using the current GP model. Subsequently, the policy is executed on the real robot to obtain new measurement data to improve the learned forward models. PILCO has been successfully applied for learning the controllers for a low-cost robot arm [10] and a robotic unicycle [9] with unprecedented data-efficiency. Recently, it also has been applied for imitation learning [14]. However, as PILCO directly optimizes lower-level controller parameters, it cannot be straightforwardly applied to learn upper-level policies. Moreover, the class of representable low-level controllers is restricted to functions through which a Gaussian distribution can be mapped in closed form.

4. Contextual Episode-based REPS

The intuition of REPS [30] is to maximize the expected reward, while staying close to the observed data to balance out exploration and experience loss. The constraint of staying close to the data is implemented by bounding the relative entropy, also called Kullback-Leibler (KL) Divergence, between the old trajectory distribution and the trajectory distribution induces by the new policy that we want to estimate. The use of the KL-bound provides an intuitive way to define the exploration-exploitation tradeoff. With a very small KL-bound, we favor exploration and we will continue to explore with the old exploration policy. Hence, we obtain a slower learning speed, but we are likely to find good solutions. With a high ϵ, we favor exploitation. The resulting policy will be more greedy and reduce exploration. We will get a fast learning speed, but the quality of the found solution will be worse on average due to premature convergence of the algorithm [8] In the episodic learning setting, the context $\boldsymbol{s}$ and the parameter $\boldsymbol{\omega}$ uniquely determine the trajectory distribution [8]. 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, REPS uses the constraint

$$\int_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s}, \boldsymbol{\omega}) \log \frac{p(\boldsymbol{s}, \boldsymbol{\omega})}{q(\boldsymbol{s}, \boldsymbol{\omega})} d\boldsymbol{s} d\boldsymbol{\omega} \leq \epsilon, \tag{4}$$

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

$$\int_{\boldsymbol{s}} p(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \tag{5}$$

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

$$\begin{aligned}
\max_{p} \quad & \iint_{s,\omega} p(s,\omega)\mathcal{R}_{s\omega} ds d\omega, && (6)\\
\text{s.t.:} \quad & \iint_{s,\omega} p(s,\omega) \log \frac{p(s,\omega)}{q(s,\omega)} ds d\omega \leq \epsilon, \\
& \iint_{s,\omega} p(s,\omega)\phi(s) ds d\omega = \hat{\phi}, \\
& \iint_{s,\omega} p(s,\omega) ds d\omega = 1.
\end{aligned}$$

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

$$p(s,\omega) \propto q(s,\omega) \exp\left(\frac{\mathcal{R}_{s\omega} - V(s)}{\eta}\right). \tag{7}$$

Here, $V(s) = \theta^T \phi(s)$ is a context dependent baseline, while η and θ are Lagrangian parameters. Subtracting the baseline from the reward is corrected from its context dependent part and it allows is to evaluate the parameter ω independently from the context s. The temperature parameter η 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

$$g(\eta,\theta) = \eta \log\left(\iint_{s,\omega} q(s,\omega) \exp\left(\frac{\mathcal{R}_{s\omega} - V(s)}{\eta}\right) ds d\omega\right) + \eta\epsilon + \theta^T \hat{\phi}. \tag{8}$$

The dual function is convex in θ and η, and the corresponding gradients can be obtained in closed form. In difference to recent EM-based policy search methods [21, 27], the exponential weighting emerges from the relative entropy bound and does not require additional assumptions. For a details of the derivation we refer to Appendix A.

4.1. Sample-based REPS

As the relationship between the context-policy parameter pair $\{s, \omega\}$ and the corresponding expected reward $\mathcal{R}_{s\omega}$ is not known, sample evaluations are used to approximate the integral given in the dual function [8, 24]. We denote these evaluations as rollouts. To execute the i^{th} rollout, we first observe the context $s^{[i]} \sim \mu(s)$. Subsequently, we sample the lower-level controller parameter using the upper-level policy $\omega^{[i]} \sim \pi(\omega|s^{[i]})$. Finally, we execute the lower-level policy with parametrization $\omega^{[i]}$ in context $s^{[i]}$ to obtain $\mathcal{R}^{[i]}_{s\omega}$. This process is repeated for N rollouts $\{s^{[i]}, \omega^{[i]}, \mathcal{R}^{[i]}_{s\omega}\}$, $i = 1, \ldots, N$ such that we can approximate the integral given in the dual function with samples. The sample-based approximation of the dual function is given in the Appendix in Equation (A.8).

As we sampled the controller parameters with the old policy, the samples have been generated from $q(s, \omega)$. Using the optimized Lagrangian parameters θ and η, we can compute the probabilities of the updated context-parameter distribution for our finite set of samples using

$$p^{[i]} \propto \exp\left(\frac{\mathcal{R}^{[i]}_{s\omega} - V(s^{[i]})}{\eta}\right). \tag{9}$$

However, in order to generate new samples, we need a parametric model to estimate $\pi(\omega|s)$. Thus, 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 $\pi(\omega|s) = \mathcal{N}(\omega|a + As, \Sigma)$, we can compute the parameters $\{a, A, \Sigma\}$ with weighted maximum likelihood estimation, that is,

$$\begin{aligned}
\begin{bmatrix} a^T \\ A^T \end{bmatrix} &= (S^T P S)^{-1} S^T P B, && (10)\\
\Sigma &= \frac{\sum_{i=1}^{N} p^{[i]} (\omega^{[i]} - \mu^{[i]})(\omega^{[i]} - \mu^{[i]})^T}{\sum_{i=1}^{N} p^{[i]}}, && (11)\\
\mu^{[i]} &= a + A s^{[i]}, && (12)
\end{aligned}$$

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 restricted only to use the last N samples. To improve the accuracy of policy updates, we can define $q(s, a)$ to be a mixture of the last H policies, and thus, reuse old samples without the need of importance weighting. The model-free contextual REPS algorithm is summarized in Table 1.

However, model-free REPS produces biased policy updates as the expected reward $\mathcal{R}_{s\omega}$ is evaluated using a single rollout. This bias can be seen by looking at the exponential sample weighting of REPS given in Eq. (9). For example, if we only have two actions a_1 and a_2, and the expected reward of a_1 is lower than the expected reward of a_2. However, if the variance of the reward for a_1 is higher, such that there will be samples from a_1 with higher reward then all samples from a_2, REPS will prefer a_1. Thus, the resulting policy is risk-seeking, a behavior that we want in general to avoid. The same bias is inherent to all other PS methods that are based on weighting samples with an exponential function, for example PoWER [21], CrKR [20] and PI^2[3] [38].

```
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 $H$.
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 $\mathcal{R}_{s\omega}^{[i]} = R(\boldsymbol{\tau}^{[i]}, \boldsymbol{s}^{[i]})$
    New dataset: $\mathcal{D}_k = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, R_{s\omega}^{[i]}\}_{i=1\ldots N}$
    Reuse old datasets: $\mathcal{D} = \{\mathcal{D}_h\}_{h=\max(1,k-H)\ldots k}$
    Update policy:
        Optimize dual function using $\mathcal{D}$, Eq. (A.8)
            $[\eta, \boldsymbol{\theta}] = \operatorname{argmin}_{\eta',\boldsymbol{\theta}'} g(\eta', \boldsymbol{\theta}'; \mathcal{D})$
        Compute sample weighting:
            $p^{[i]} \propto \exp\left(\frac{\mathcal{R}_{s\omega}^{[i]} - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right)$, for each sample $i$ in $\mathcal{D}$
        Update policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with weighted ML
            using $\mathcal{D}$ and $p^{[i]}$, Eqs. (10-11)
    end
Output: policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$
```

Table 1: In each iteration of the contextual REPS algorithm, we collect a dataset $\mathcal{D}_k = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, \mathcal{R}_{s\omega}^{[i]}\}_{i=1\ldots N}$ by performing rollouts on the real system. For the REPS algorithm, we reuse the last H 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 $\pi(\boldsymbol{\omega}|\boldsymbol{s})$.

5. Model-based Contextual Policy Search

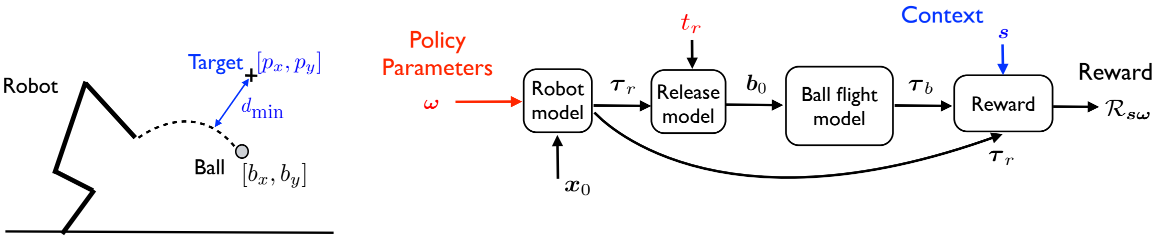
Our main motivation to use models with contextual policy search is two-fold. First, we want to improve the data-efficiency of the model-free REPS using artificial rollouts. Second, we want to obtain an accurate estimate of the expected reward $\mathcal{R}_{s\omega}$ for a given context-policy parameter pair to avoid the bias in the sample-based REPS formulation. The expected reward $\mathcal{R}_{s\omega} = \mathbb{E}_{\boldsymbol{\tau}}[R(\boldsymbol{\tau}, \boldsymbol{s})|\boldsymbol{s}, \boldsymbol{\omega}]$ can be estimated by multiple samples from the trajectory distribution $p(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\omega})$, that is,

$$\mathcal{R}_{s\omega} = \frac{1}{L} \sum_{l=1}^{L} R(\boldsymbol{\tau}^{[l]}, \boldsymbol{s}), \tag{13}$$

where the trajectories are now generated using the learned forward models in computer simulation and we will assume that the trajectory-dependent reward function $R(\boldsymbol{\tau}, \boldsymbol{s})$ is known. We generate M artificial context-parameter samples using an estimate of the context distribution and the upper level policy. Using the learned models, we evaluate these artificial samples in computer simulation.

To gain the most benefit out of the learned forward models, we use structural knowledge about the task to decompose the monolithic forward model of the system in smaller forward models that are easier to learn. For example, when throwing a ball with a robot, we can learn individual forward models to predict the movement of the robot, the

[3] The PI^2 is claimed to be unbiased [38]. However, this is only the case if all the noise in the system can be controlled, which is often an unrealistic assumption.



(a)

(b)

Figure 3: **(a)** The illustration of the throwing task. **(b)** The decomposition of the experiment into individual forward models. First, the learned robot dynamics model is used to obtain the robot trajectory $\boldsymbol{\tau}_r$ following the lower-level policy $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\omega})$ starting at state $\boldsymbol{x}_0$. Subsequently, the learned release model predicts the initial state of the ball $\boldsymbol{b}_0$ using the joint state at time t_r, $\boldsymbol{x}_{t_r}$. Afterwards, the learned ball flight model provides the ball trajectory $\boldsymbol{\tau}_b$. Finally, the reward is computed using the context $\boldsymbol{s}$ (target position), the ball trajectory $\boldsymbol{\tau}_b$ and the torques predicted in $\boldsymbol{\tau}_r$.

initial configuration of the ball when it is released from the robot and the flight of the ball. We show such a decomposition of the forward models for the ball throwing example in Figure 3(b), the illustration of the experiment is shown in Figure 3(a).

In Figure 4, we show the learning framework of the GPREPS algorithm. One of the key difference compared to the model-free case is that now we evaluate the policy on the real-robot only to obtain measurement data. Using our ball throwing example, such measurement data might consist of the joint trajectory $\boldsymbol{\tau}_r$, the trajectory of the thrown ball $\boldsymbol{\tau}_b$, etc. Using the obtained measurement data, we learn models of the dynamics and discrete events, such as releasing the ball.

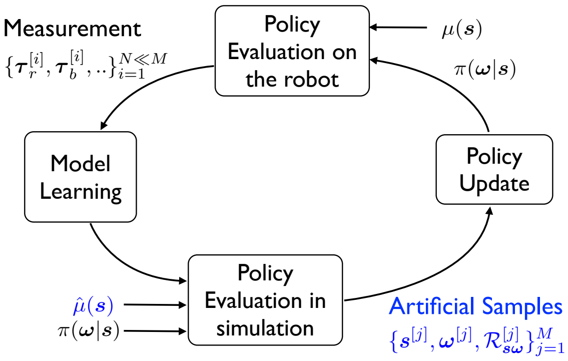

Measurement
$\{\boldsymbol{\tau}_r^{[i]}, \boldsymbol{\tau}_b^{[i]}, ..\}_{i=1}^{N \ll M}$
Policy Evaluation on the robot
$\mu(\boldsymbol{s})$
$\pi(\boldsymbol{\omega}|\boldsymbol{s})$
Model Learning
Policy Update
$\hat{\mu}(\boldsymbol{s})$
$\pi(\boldsymbol{\omega}|\boldsymbol{s})$
Policy Evaluation in simulation
Artificial Samples
$\{\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]}, \mathcal{R}_{s\omega}^{[j]}\}_{j=1}^{M}$

Figure 4: The learning framework of model-based contextual PS, such as GPREPS. We use real robot evaluations to obtain the measurement data, which is used to learn the problem specific models. Finally, we evaluate artificial samples in computer simulation using the learned models. These artificial samples are now used for the policy update.

Note that in the model-based formulation we use solely the artificial samples to update the policy. Thus, in case of unbiased models, we can avoid the possibly noisy reward samples evaluated on the real system and, hence, eliminate the risk sensitive bias inherent to the REPS algorithm. Additionally, we can increase the number of artificial samples significantly $N \ll M$ to further improve the accuracy of policy updates. Due to the recent success of using Gaussian Process models to reduce the model bias when learning complex system dynamics [9], 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).

5.1. *Gaussian Process REPS*

In each iteration, first we collect measurement data from the robot and its environment. For data collection, we observe the context $\boldsymbol{s}^{[i]}$ and sample the parameters $\boldsymbol{\omega}^{[i]}$ using the upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$. Subsequently, we use the lower-level control policy $\pi(\boldsymbol{x}; \boldsymbol{\omega}^{[i]})$ to obtain the trajectory sample $\boldsymbol{\tau}^{[i]}$. It is important to evaluate sufficiently many samples to obtain enough measurement data, such that the GP models produce accurate predictions over the relevant part of the state space. Therefore, we repeat the data collection step N times. To favor data-efficiency, we want 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 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. We retrain the GP models using the so far obtained measurement data. The GPREPS algorithm is summarized in Tab. 2.

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. Given the observed context variables $\{\boldsymbol{s}_i\}_{i=1}^N$, we fit the distribution parameters with maximum likelihood estimation. After updating the estimated context distribution $\hat{\mu}(s)$, 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 $\boldsymbol{\tau}^{[j,l]}$, $l = 1, \dots, L$ 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 $g(\eta, \boldsymbol{\theta})$ (Eq. A.8) using the artificially generated samples and compute the new weight $p^{[j]}$ (Eq. 9) 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 as to use the expected reward for the REPS algorithm instead of a single sample estimate of the reward. Consequently, if the models produce unbiased rewards $\mathcal{R}_{s\omega}$, the final policy will also be unbiased. Finally, we update the policy by the weighted maximum likelihood estimate using Eqs. (10-11). In the following, we will explain in more detail how to learn the models and how to sample the trajectories.

```
GPREPS Algorithm
Input: relative entropy bound $\epsilon$, initial policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, number of policy updates $K$.
for $k = 1, \dots, K$
    Collect Data:
        Observe context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s}), i = 1, \dots, N$
        Execute policy with $\boldsymbol{\omega}^{[i]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$
    Train forward models with all data, estimate $\hat{\mu}(s)$
    for $j = 1, \dots, M$
        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 $\mathcal{R}_{s\omega}^{[j]} = \sum_l R(\boldsymbol{\tau}_j^{[l]}, \boldsymbol{s}^{[j]})/L$
    Construct artificial dataset: $\tilde{\mathcal{D}} = \{\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]}, R_{s\omega}^{[j]}\}_{j=1\dots M}$
    Update Policy:
        Optimize dual function using $\tilde{\mathcal{D}}$, Eq. (A.8)):
            $[\eta, \boldsymbol{\theta}] = \text{argmin}_{\eta', \boldsymbol{\theta}'} g(\eta', \boldsymbol{\theta}'; \tilde{\mathcal{D}})$
        Compute sample weighting:
            $p^{[j]} \propto \exp\left(\frac{\mathcal{R}_{s\omega}^{[j]} - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}^{[j]})}{\eta}\right), j = 1, \dots, M$
        Update policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with weighted ML
            using $\tilde{\mathcal{D}}$ and $p^{[j]}$, Eqs. (10-11)
end
Output: policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$
```

Table 2: In each iteration of the GPREPS algorithm, we collect data from the environment by observing the context and executing the policy. Using the observed data we update the models, which we subsequently use to generate M artificial samples. We obtain the expected reward for each sample by sampling L trajectories and averaging over the trajectory rewards. Finally, we update the policy by optimizing the dual function and computing the sample weights.

5.2. Learning GP Forward Models

We use forward models to simulate a trajectory $\boldsymbol{\tau}$ 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. For example, when learning to play the table tennis game, we use the prior knowledge that the trajectory of the ball is described by its free dynamics and the contacts with the table and the racket.

Gaussian Processes are efficient non-parametric Bayesian regression tools [32] that explicitly represent model uncertainty. For learning the model, the set of training data is given by $\mathcal{D} = \{\boldsymbol{v}_i, \boldsymbol{w}_i\}_{i=1}^N$, with $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ being the training input and target values. We use individual GP models for each output dimension and, therefore, we will assume scalar target values w in the subsequent discussion. For a new test input $\boldsymbol{v}_*$, the predictive distribution $p(w_*|\boldsymbol{v}_*, \mathcal{D})$ of the posterior Gaussian process is a Gaussian $\mathcal{N}(w_*|\mu_*, \sigma_*^2)$ with mean and variance

$$\mu_* = \boldsymbol{k}^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{w}, \tag{14}$$

$$\sigma_*^2 = \sigma_\epsilon^2 + k(\boldsymbol{v}_*, \boldsymbol{v}_*) - \boldsymbol{k}^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{k}, \tag{15}$$

respectively, where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{K}$ is the $N \times N$ kernel matrix with elements $\boldsymbol{K}_{ij} = k(\boldsymbol{v}_i, \boldsymbol{v}_j)$ and $\boldsymbol{k}$ denotes the kernel vector for the query input with $k_i = k(\boldsymbol{v}_i, \boldsymbol{v}_*)$. The parameter σ_ϵ^2 represents the variance of the system noise

ϵ. The covariance function $k(\cdot,\cdot)$ defines a similarity measure between the input data. We use the squared exponential function as covariance function

$$k(\boldsymbol{v},\boldsymbol{v}') = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{v}-\boldsymbol{v}')^T \boldsymbol{L}^{-1}(\boldsymbol{v}-\boldsymbol{v}')}{2}\right),$$

where $\boldsymbol{L} = \mathrm{diag}(\boldsymbol{l}^2)$ is the diagonal matrix containing the bandwidth parameters of the squared exponential kernel. The parameter σ_f^2 represents the variance of the function. We refer to parameters $\{\boldsymbol{l}, \sigma_f, \sigma_\epsilon\}$ as hyper-parameters of the GP. To obtain accurate predictions, the hyper-parameters have to be set properly. To do so, we optimize the hyper-parameters of the GP by maximizing the marginal log-likelihood using gradient-based optimizers [32]. The computational complexity of optimizing the hyper-parameters is dominated by the $O(N^3)$ matrix inversion in Eqs. (14-15). To reduce the computational demands, we use sparse GP models [36, 39].

GPREPS can be regarded as a combination of information theoretic PS, contextual PS and model-based RL to address the important problem of learning real world robot skill efficiently. Current model-based policy search methods rely on deterministic approximate inference methods. While such approximate inference is efficient for computing the policy gradient, it also suffers from severe limitations. The structure of the used policy representations and the reward functions is limited to a specific type of functions and the approximation may cause a severe bias in the policy update, GPREPS uses sampling to evaluate a new context-parameter pair. By the use of sampling, we do not rely on any assumption on the policy representation as well as on the reward function except that we can sample from the policy and evaluate the reward function for a given state action pair. Due to the increased generality, efficient model-based policy search can now be used in a much wider range of applications, including the contextual policy search application introduced in this paper. However, other types of structured policies are also possible. For example, we could learn mixture of experts models similar as the one introduced in the HiREPS method [8].

5.3. Trajectory and Reward Predictions

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

$$\mathcal{R}_{s\omega} = \int_{\tau} \hat{p}(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau} \tag{16}$$

for a given parameter vector $\boldsymbol{\omega}$ executed in context $\boldsymbol{s}$. The expectation over the trajectories is now estimated using the learned forward models.

To obtain the trajectory distribution in closed form, at each time step we have to compute the GP predictive distribution

$$p(\boldsymbol{x}_{t+1}) = \iint_{\boldsymbol{x}_t,\boldsymbol{u}_t} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t) p(\boldsymbol{x}_t,\boldsymbol{u}_t) d\boldsymbol{x}_t d\boldsymbol{u}_t. \tag{17}$$

However, if the query input 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 non-linear, 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 $p(\boldsymbol{x}_{t+1})$. To overcome this problem, we use samples to solve the integral. Using the sample state $\boldsymbol{x}_t$, we first compute the control $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\omega})$ 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.

An alternative approach to solve the integral in Equation (17) is to use moment matching [9, 10]. Moment matching computes the first and second moment of the predictive distribution $p(\boldsymbol{x}_{t+1})$, i.e., it approximates it by a Gaussian. Moment matching is a deterministic approximate inference technique that provides a closed form solution. However, the Gaussian approximation of the predictive distribution might result in a biased trajectory distribution $p(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s})$, and therefore, in a biased estimate of the expected reward $\mathcal{R}_{s\omega}$. Furthermore, when using moment matching, the class of the lower-level controllers is restricted to all functions through which we can map a Gaussian analytically, i.e., linear controllers, squared exponentials and trigonometric functions. Thus, many policies are infeasible, for example policies with a hard limit on the controls. Such hard limit needs to be approximated by the use of trigonometric functions. Nevertheless, obtaining the trajectory (and reward) distribution with moment matching is the method of choice, in particular, when using gradient based policy search [9] where accurate analytical estimates of the policy gradient are required.

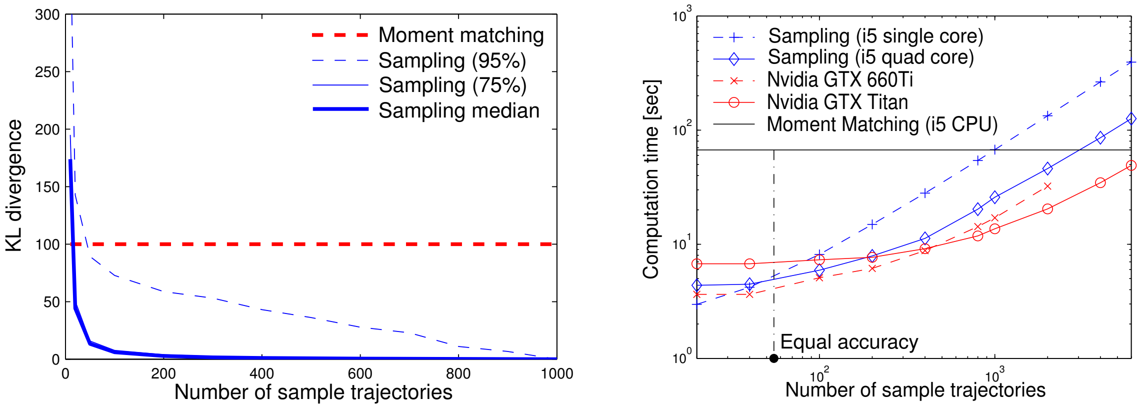
When using sampling, the lower-level policy class is not restricted and torque limits can be applied with ease. On the other hand, to accurately approximate the trajectory distribution, the number of sample trajectories L must be relatively high and typically needs to increase with the dimensionality of the system. As the moment matching is also computationally highly demanding, the question arises which method can be implemented more efficiently. 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. Thus, 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.

5.3.1. Quantitative Comparison of Sampling and Moment Matching

To compare moment matching and sampling, we evaluated both approaches on predicting the joint trajectory of a 4-link simulated non-linear pendulum. The lower-level policy was given by a linear trajectory-tracking PD controller. We used 1500 observed data points and 300 pseudo-inputs to train the sparse GP models. The prediction horizon was set to 100 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 $\mathrm{KL}\left(p(\boldsymbol{x}_t)||\tilde{p}(\boldsymbol{x}_t)\right)$ 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., $\sum_{t=1}^{100} \mathrm{KL}\left(p(\boldsymbol{x}_t)||p_{MM}(\boldsymbol{x}_t)\right) = 100$, where $p_{MM}(\boldsymbol{x}_t)$ is the state distribution predicted by moment matching. The result is shown in Fig. 5(a). 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.



(a)

(b)

Figure 5: (a) 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. (b) 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.

We also evaluated the computation time of both approaches. For the sampling approach, we evaluated the computation time for different types of parallelization. As can be observed from Figure 5(b), 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 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. Thus, using sampled trajectories results in a moderate speed-up already for main stream computers and can further be improved when using a GPU implementation. In addition to the improved computation speed, sampling avoids the approximations involved in the moment matching approach, for example, using trigonometric functions to approximate torque limits [9]. Sampling produces unbiased estimates of the expected reward, and thus, we can improve the accuracy of the prediction by increasing the number of samples. Nevertheless, moment matching is still favorable for algorithms that require the computation of the gradient of the state distribution w.r.t. the policy-parameters, e.g., for PILCO [9].

5.3.2. 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. [36, 39]. In general, sparse GP methods maintain a set of $M < N$ highly representative training samples, where N is the total number of training samples. When using sparse methods, the training time scales only $\mathcal{O}(M^2N)$, while the posterior mean and variance prediction time scales $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ respectively.

When learning GP forward models, numerical problems may emerge. In order to learn accurate models, we often need thousands of training samples $\{\boldsymbol{v}, \boldsymbol{w}\}$. When using this 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 result in numerical problems which can easily lead to an inaccurate trajectory prediction. To avoid this problem, a common strategy is to add noise to the training target data $w_i = w_i + \epsilon_{add}, i = 1, \ldots, N$. This will naturally increase the value of σ_ϵ, 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 $\sigma_{add} = 10^{-2}\text{std}(\boldsymbol{w})$. 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 [36] and the standard GP approach [32]. We also tested the sparse method presented in [39], but ran into numerical problems with this method. The control problem is to balance a simulated planar 4-link pendulum 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 [36], with the same measurement data. Subsequently, we use the GP models to predict the reward of 50 context-parameter pairs for 20 time steps. We use 20 trajectories per context-parameter pair. The corresponding 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 $\boldsymbol{x}_r$ represents 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.

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 α is a scaler. 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 this 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, $M = N/4$.

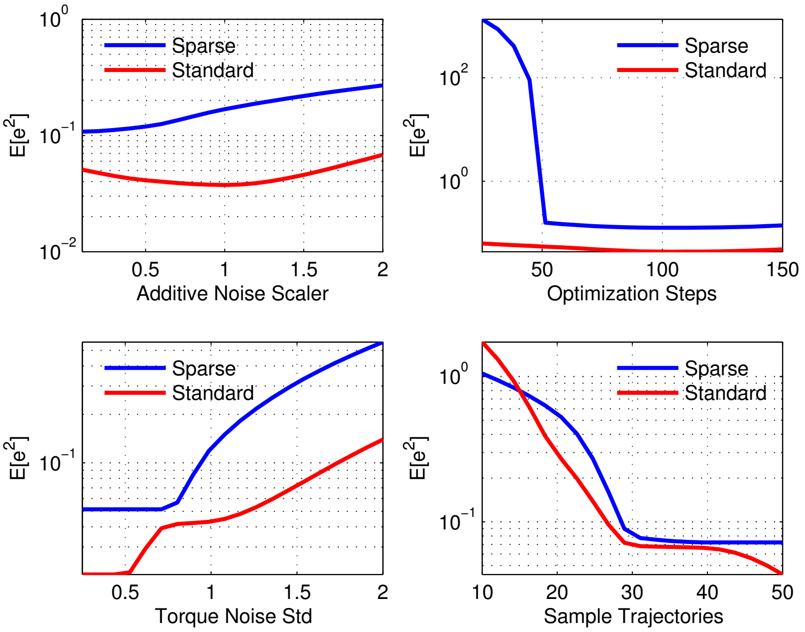


Figure 6: Comparison of the standard and the sparse GP approach. **Top left:** an increased additive noise offers higher numerical stability with lower prediction accuracy. **Top right:** we can already obtain a good model after 50 hyper-parameter optimization steps. **Bottom left:** the sparse method has difficulties in capturing the stochasticity with increasing control noise. **Bottom right:** increasing the amount of training data clearly has a positive effect on the prediction performance.

The results of the model comparison tasks can be seen in Fig. 6. 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 optimization 50 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.

6. Results

We evaluated our data-efficient contextual policy search method on a context-free comparison task and three contextual motor skill learning tasks. In our context-free task, we compare the GPREPS approach with the competing PILCO method on a simulated 4-link pendulum balancing task. In the contextual learning tasks, we learn how to throw a ball to distinct targets with a simulated 4-link robot. In the second task, a 7-DoF robot arm has to learn how to move a target puck in a hockey game. Here we present simulated results as well as real robot results. We also compare our approach with CrKR [20] and to contextual model-free REPS. In the third experiment we use GPREPS to learn to play table tennis with a simulated robot arm. As the lower-level controllers needs to scale to anthropomorphic robotics, we implement them using the Dynamic Movement Primitive [17] approach, which we will now briefly review.

6.1. Dynamic Movement Primitives

To parametrize the lower-level policy we use an extension [19] to the Dynamic Movement Primitives (DMPs) introduced in [17]. A dynamic movement primitive is defined as second order dynamical system that acts like a spring-damper system which is activated by a non-linear forcing function f

$$\ddot{x}_t = \tau^2\alpha_x(\beta_x(g - x_t) - \dot{x}_t) + \tau^2 f(z_t; \boldsymbol{v}), \quad (18)$$

$$\dot{z}_t = -\tau\alpha_z z_t. \quad (19)$$

with constant parameters α_x,β_x and α_z. 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 and synchronizes the joints. 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 drives the non-linear forcing function f. The parameter g is the unique point attractor of the system. The spring-damper system is modulated by the function $f(z_t; \boldsymbol{v}) = \boldsymbol{\phi}(z_t)^T\boldsymbol{v}$ that 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 demonstrations. The basis functions $\boldsymbol{\phi}_i(z_t), i = 1, \dots, K$ 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 center of the basis center and σ_i the bandwidth. The squared exponential basis functions are multiplied by z_t such that f vanishes for $t \to \infty$. Thus, 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

$$\boldsymbol{v} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{Y}, \quad \boldsymbol{Y} = \frac{1}{\tau^2}\ddot{\boldsymbol{x}}_{obs} - \alpha_x(\beta_x(g - \boldsymbol{x}_{obs}) - \dot{\boldsymbol{x}}_{obs}), \quad (20)$$

where $\boldsymbol{\Phi}_{t,\cdot} = \boldsymbol{\phi}^T(z_t)$ is the matrix of basis vectors at time step t. In a robot skill learning task, we can adapt the weight parameters $\boldsymbol{v}$, the goal attractor g and the time scaling factor τ to optimize the trajectory. Additionally, we can also adapt the final desired velocity $\dot{g}$ of the movement with the extension given in [19].

To reduce the dimensionality of the learning problem we usually learn only a subset of the DMP hyper-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 maximize the reward. After obtaining a desired trajectory by the DMP, the trajectory is followed by a feedback controller which is part of the lower-level control policy. 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 [19].

6.2. Exploiting Reward Models as Prior Knowledge

A simple approach to improve the data-efficiency of the model-free contextual REPS algorithm is to use the known reward model $R(\boldsymbol{\tau}, \boldsymbol{s})$ evaluate a single outcome trajectory in multiple contexts $\boldsymbol{s}$. Such a strategy is possible if the evaluated trajectories $\boldsymbol{\tau}^{[i]}$ 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 targets $\boldsymbol{s}^{[i]}$.

6.3. 4-Link Pendulum Balancing Task

In this task, the goal is to find a PD controller that balances a simulated 4-link planar pendulum around the upright position. The pendulum has a total length of 2m and a total mass of 70kg. 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. We compare GPREPS to the model-free REPS [30] and PILCO [9, 12], a state of the art model-based policy search algorithm. As PILCO cannot learn contextual policies, we learn context-free upper-level policies with REPS and GPREPS to have a fair comparison. Thus, 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 $\boldsymbol{G}_{4\times8}$ is obtained by reshaping the parameter vector

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

$\omega_{32\times1}$. 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 [36] for PILCO and GPREPS.

In Tab. 3 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 and GPREPS significantly outperformed the model-free REPS in terms of data-efficiency, the model-based algorithms typically require a higher amount of computational time for policy update. In Figure 7, we show the learning progress against computational time required by the algorithms. In this evaluation we only consider the amount of time taken by the algorithms, but we omit the time taken for policy evaluation on the robot. In this regard REPS is superior compared to the model-based algorithms, as the samples are already evaluated on the robot. Note that this comparison is heavily biased towards favoring the model-free method. Nevertheless, the experiment gives an intuition of the scale of real world computational times when working with real robots.

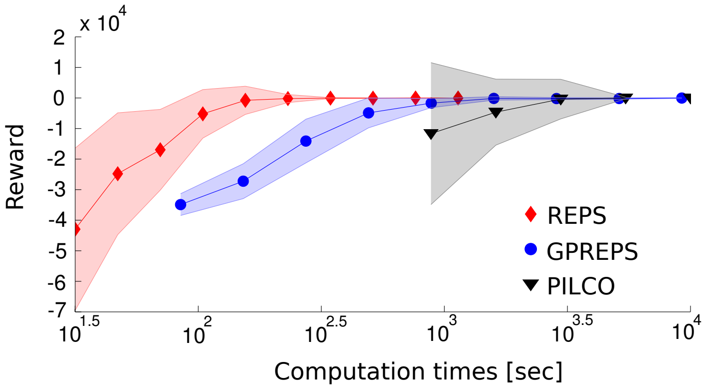


Figure 7: The learning progress of the algorithms against the computational times. Note that the time taken for sample evaluation on the robot is not included in the computational time. Thus, model-free REPS outperforms both model-based approaches. However, for many real-world learning problems sample evaluation on the robot is time consuming and could lead to robot wear, and thus, model-based methods are preferred. For the evaluation we use a standard desktop PC (with a quad-core Intel i5 CPU) and a high-end GPU (Nvidia GTX Titan) with high double-precision computational power.

For both GPREPS and PILCO we use GP models, for which model training takes a significant amount of time. Note that in our learning scenario 1 second of real experience corresponds to 50 data points, with 12 dimensional training input and 8 dimensional training target. Other than model training, the model-based methods require additional computational time for policy update. For GPREPS, we use the sampling approach to evaluate 500 artificial policy parameter samples, where each predicted trajectory consists of 100 steps and we use 20 trajectories for a single parametrization. In other words, GPREPS requires the prediction of $10,000$ trajectories per policy update, which can be evaluated efficiently on GPUs (Fig. 5(b)). For PILCO on the other hand, we cannot use parallelization straightforwardly and the computations are more involved due to the moment-matching approach and gradient computations. Additionally, PILCO repeatedly recomputes the policy gradient until it converges to a local minimum. Thus, computational times may vary significantly for PILCO. As we can see in the figure, GPREPS requires roughly an order of

magnitude higher computational time compared to the model-free REPS, while for PILCO this number varies between 1.5 to 2 orders of magnitude.
Although computational times are generally higher with model-based algorithms, especially when using GP models, in many real-world scenario model-based approaches might still learn the task faster compared to model-free methods. Some real robot experiments might require minutes to evaluate, which would make learning times impractically large with model-free algorithms. Furthermore, the aforementioned concerns with robot experiments, such as robot wear and the requirement for expert supervision, are not directly addressed with model-free methods.

6.4. Ball-Throwing Task

In this task, a 4-link robot has to learn to throw a ball at a target position. The target position $\boldsymbol{s} = [x, y]$ is uniformly distributed in a specified range, and we learn a contextual upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$. The context is varied from $x \in [5, 15]$m and $y \in [0, 3]$m. The lengths and the masses of the links were set to $\boldsymbol{l} = [0.5, 0.5, 1.0, 1.0]$ m and $\boldsymbol{m} = [17.5, 17.5, 26.5, 8.5]$ kg respectively. The robot coarsely models a human with the joints representing the ankle, knee, hip and shoulder.

In this experiment, we used GPREPS to find DMP shape parameters $\boldsymbol{v}$ for throwing a ball to multiple targets while maintaining balance. The reward function was defined as

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

The first term punishes minimum distance of the ball trajectory $\boldsymbol{b}$ to the target $\boldsymbol{s}$. 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 penalty term is 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 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 joint angle and angular velocity limits are defined as

$$\begin{aligned} \boldsymbol{q}_l &= [-0.8, -2.5, -0.1, -\pi]^T \text{rad}, & \dot{\boldsymbol{q}}_l &= [-50, -50, -50, -50]^T \text{rad/s}, \\ \boldsymbol{q}_u &= [0.8, 0.05, 2, \pi]^T \text{rad}, & \dot{\boldsymbol{q}}_u &= [50, 50, 50, 50]^T \text{rad/s}, \end{aligned}$$

and finally $\boldsymbol{x}_l = [\boldsymbol{q}_l^T, \dot{\boldsymbol{q}}_l^T]^T$ and $\boldsymbol{x}_u = [\boldsymbol{q}_u^T, \dot{\boldsymbol{q}}_u^T]^T$. The last term of the reward function favors energy-efficient movement. In our experiment we set the reward weighting factors to $c_1 = 10^2$, $c_2 = 10^3$ and $c_3 = 10^{-8}$.

As lower-level controllers, we used DMPs with 10 basis functions per joint. We modified the shape parameters, but fixed the final position and velocity of the DMP to be the upright position 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 $\boldsymbol{u}_t$.

To produce trajectory rollouts with the model-based GPREPS, we learn three distinct models of the environment. The first model represents the dynamics of the robot. We use the observed state transitions as training samples for this model. The second model is used to predict the initial position and velocity of the ball at the release time t_r. 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.

The policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ was initialized such that the robot is expected to throw the ball approximately 5 m without maintaining balance, which led to high penalties. We found this policy by applying the context-free REPS. GPREPS learned to accurately hit the target for the given range of targets. Fig. 8 shows the learned motion sequence for two different targets. The displacement for targets above $\boldsymbol{s} = [13, 3]^T$ m could raise up to 0.5 m, otherwise the maximal

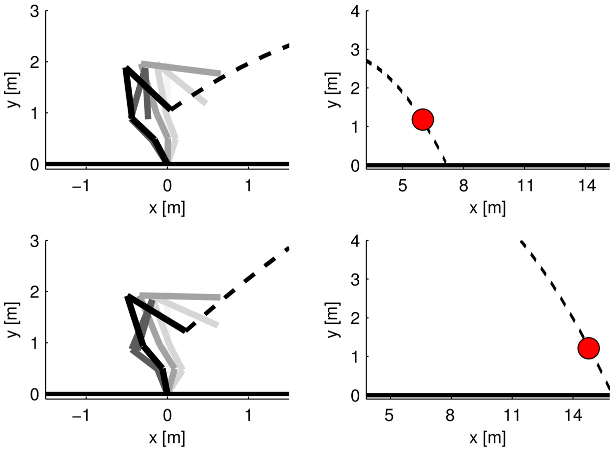


Figure 8: Throwing motion sequence. The robot releases the ball after the specified release time and hits different targets with high accuracy.

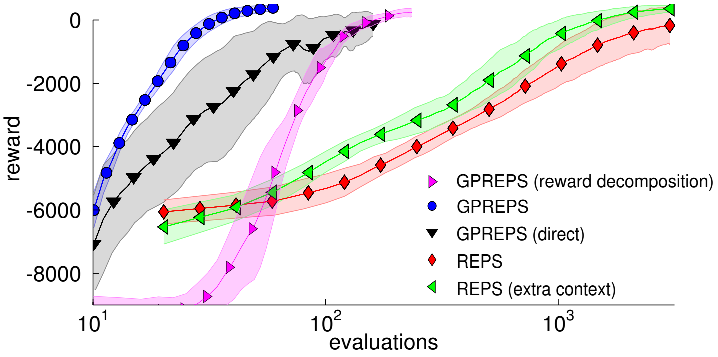


Figure 9: Learning curves for the ball-throwing problem. The shaded regions represent the standard deviation of the rewards over 20 independent trials. GPREPS converged after 30–40 interactions with the environment, while REPS required ≥ 5000 interactions. Using the reward model to generate additional samples for REPS led to better final policies, but could not compete with GPREPS in terms of learning speed. Learning the direct reward model $\mathcal{R}_{s\omega} = f(\boldsymbol{s}, \boldsymbol{\omega})$ yielded faster learning than model-free REPS, but the quality of the final policy is limited.

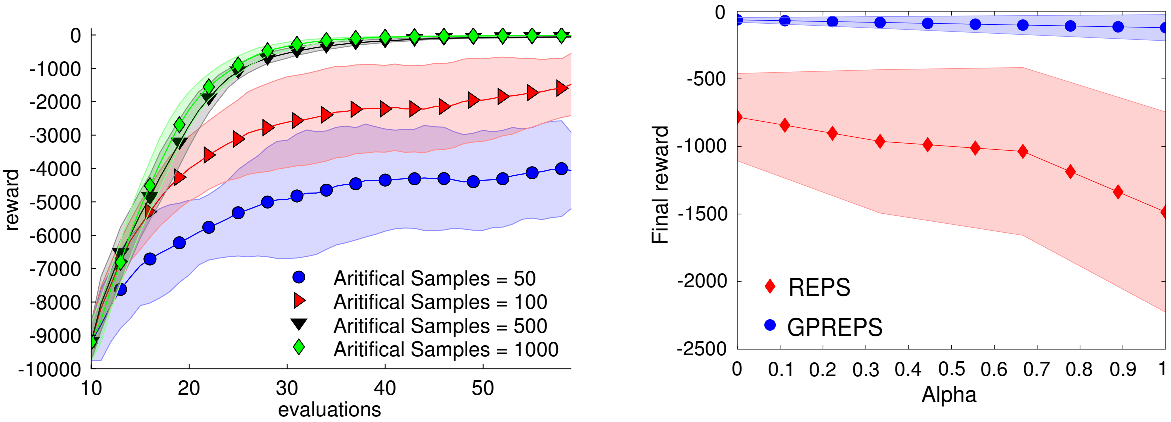
error was smaller than 10 cm. The policy chose different DMP parametrizations and release times for different target positions. To illustrate this effect we show two target positions $\boldsymbol{s}_1 = [6, 1]$ m, $\boldsymbol{s}_2 = [12, 3]$ m in Fig. 8. When the target was farther away the robot showed a more distinctive throwing movement and released the ball slightly later.

The learning curves for REPS and GPREPS are shown in Fig. 9. In addition, we evaluated REPS using the known reward model $R(\boldsymbol{\tau}, \boldsymbol{s})$ to generate additional samples with randomly sampled contexts $\boldsymbol{s}^{[i]}$. We denote these experiments as *extra context*. We also evaluated GPREPS when learning the expected reward model directly $\mathcal{R}_{s\omega} = f(\boldsymbol{s}, \boldsymbol{\omega})$ as a function of the context and parameters $\boldsymbol{\omega}$ with a GP model (denoted by *direct*). Additionally, we investigated a learning scenario where we observe the individual terms of the decomposed reward function and learn models that predict the individual terms from policy parameters $\boldsymbol{\omega}$ and context $\boldsymbol{s}$. The terms we observe are the distance penalty $R_{dist} = -c_1 \min_t ||\boldsymbol{b}_t - \boldsymbol{s}||_2$, the state constraint penalty term $R_{\boldsymbol{x}} = -c_2 \sum_t f_c(\boldsymbol{x}_t)$ and the torque penalty term $R_{\boldsymbol{u}} = -c_3 \sum_t \boldsymbol{u}_t^T \boldsymbol{u}_t$. This experiment represents a transition between the standard GPREPS and the GPREPS direct approach where we only use a limited amount of expert knowledge about the task. Thus, this method is easy to implement on any new test setting. In the following we refer to this approach as *GPREPS (reward decomposition)*

Fig. 9 shows that REPS converged to a good solution after 5000 episodes 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. For GPREPS, we sampled ten trajectories initially to obtain a confident GP model. We evaluated only one sample after each policy update and, subsequently, updated the learned forward models. We used 500 artificial samples and 20 sample trajectories per sample to obtain the expectation. GPREPS converged to a good solution in all cases after 30–40 real evaluations. Directly learning $\mathcal{R}_{s\omega}$ also resulted in an improved learning speed, but we observed a highly varying, on average lower quality in the resulting policies (*GPREPS (direct)*). However, we observe that the final results with the *GPREPS (reward decomposition)* approach is better than GPREPS direct, but worse than GPREPS. Interestingly, the final solution is consistently better than REPS. Thus, we can conclude that even a limited amount of expert knowledge about the task can provide better results compared to the model-free REPS. Note that this approach is easy to implement on any novel test scenario. This results confirms our intuition that decomposing the forward model into multiple components simplifies the learning task.

6.4.1. Influence of the Number of Artificial Samples

In the following, we investigate the influence of the number of artificial samples used to update the policy on the learning performance. To obtain an accurate estimation of the distribution $p(\boldsymbol{s}, \boldsymbol{\omega})$, we are interested in 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 it as low as possible, without affecting the learning performance.



(a)

(b)

Figure 10: **(a)** The learning curves of GPREPS with different amount of artificial samples used. An increasing amount of artificial samples clearly improves performance. However, we do not observe much difference in learning efficiency above 500 samples. Using lower number of artificial samples resulted in lower quality final policies. **(b)** The reward of the learned final policy against the level of dynamics stochasticity. By using the GP models, GPREPS is able to capture the stochasticity, even with a higher amount of noise. Thus, for policy update it able to approximate the expected reward well. On the other hand, REPS tends to produce worse final policies as it only uses reward samples from the real distribution and not the expected reward.

In Fig. 10(a) 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.

6.4.2. Learning with Stochastic Dynamics

For learning problems where the dynamics of the robot and its environment are stochastic, the variance of the resulting trajectory $\boldsymbol{\tau}$, and thus, the variance of the reward $R(\boldsymbol{\tau}, \boldsymbol{s})$ might be considerably large. As discussed earlier, with the model-free REPS algorithm we typically require a single rollout evaluation for a given context-parameter pair, which we then assume to be the expected outcome. While this is a reasonable assumption for deterministic problems, most real-world robot tasks contain some level of stochasticity. GPREPS avoids the problem of learning with stochastic outcomes by averaging over multiple samples of the same context-parameter pair evaluation, as in Eq. (13).

In the following we demonstrate the effect of noise in the dynamics on the learning performance of the robot throwing task. We implement the stochasticity of the dynamics by adding Gaussian noise to the initial state of the ball at the release time. In a real robot experiment, this stochasticity might emerge from the unknown dynamics of ball release with a robot hand. The standard deviation of the initial position of the ball is set to α7.5cm, while for the initial velocity it is set to α30cm/s, where α is a scaler. We investigate the effect of increasing amount noise on the learning performance, by varying $\alpha \in [0, 1]$.

In Fig. 10(b), we can see the quality of the converged policy of REPS and GPREPS with an increasing level of stochasticity in the dynamics. As the figure shows, GPREPS already outperforms REPS in the deterministic case. However, with an increasing level of system stochasticity, the quality of the final policy learned by REPS gets significantly worse. The reason for this phenomena is that REPS uses noisy samples of the reward instead of the expected reward. In contrast, the forward models of GPREPS learns the stochasticity of the system, and thus, it is able to approximate the expected reward well. Even with a significant amount of noise on the system ($\alpha = 1$), GPREPS is able to avoid converging to local optima, and provides good quality policies with relatively low variance.

6.5. Robot Hockey with Simulated Environment

In this task we learn a robot hockey game using the KUKA lightweight robot arm in Fig. 1. The goal of the robot is to shoot a hockey puck using the attached hockey stick to move a target puck, which is located at a certain distance. The robot can move the target puck by hitting it with another puck, that we denote as control puck. The initial 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 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 provided puck in the direction of the target puck, but also with the appropriate force. The simulated hockey task is depicted in Fig. 11. The context variable is defined as $\boldsymbol{s} = [b_x, b_y, d^*]^T$.

We first evaluated our method in simulation. 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 τ 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. 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

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

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.

In the robot hockey task, modeling the contact of the stick and the control puck is challenging due to the curved shape of the hockey stick. When shooting the puck, the stick 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 stick 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 pucks, we assume that we know the radius of the pucks, and thus, we can always predict when a contact is happening. For modeling the effect of the contact, we also learn a separate GP model that predicts the state of the pucks after the contact given the state of the pucks before the contact.

We compared GPREPS to model-free REPS and CrKR [20], a state-of-the-art model-free contextual policy search method. Furthermore, we evaluated GPREPS without decomposing the experiment, and directly predict the reward $\mathcal{R}_{s\omega}$ with a GP model using the policy parameter $\boldsymbol{\omega}$ and the context $\boldsymbol{s}$ as input (*GPREPS (direct)*). The resulting learning curves are shown in Fig. 12(a). 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 learned by model-free REPS were of lower quality. The *GPREPS (direct)* algorithm resulted in faster convergence than the

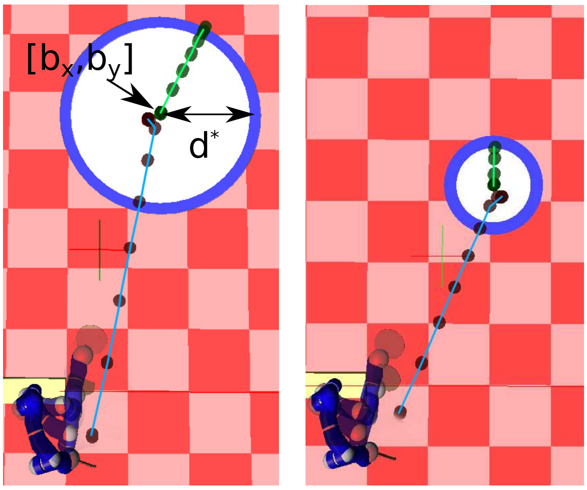


Figure 11: Robot hockey task. The robot shoots the control puck at the target puck to make the target puck move for a specified distance. Both, the initial location of the target puck $[b_x, b_y]^T$ and the desired distance d^* to move the puck were varied. The context was given by $\boldsymbol{s} = [b_x, b_y, d^*]$. The learned skill for two different contexts $\boldsymbol{s}$ is shown, where the robot learned to place the target puck at the desired distance.

model-free REPS version, but the resulting policies had lower quality. 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 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 learned movement is shown in Fig. 11 for two different contexts. After 100 evaluations, GPREPS placed the target puck accurately at the desired distance with a displacement error ≤ 5 cm.

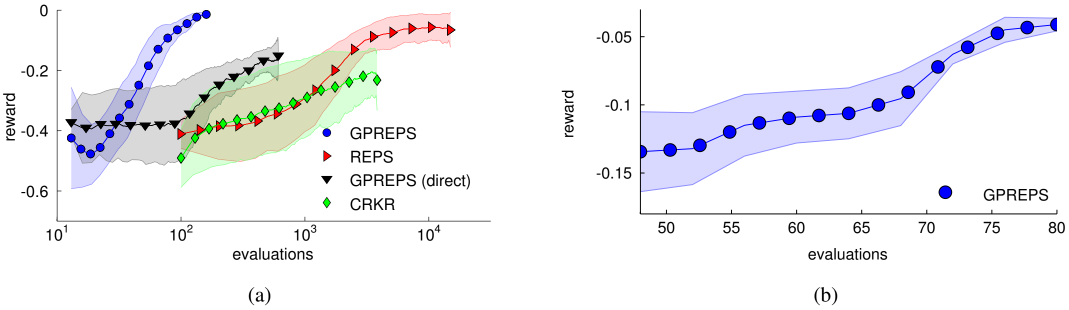
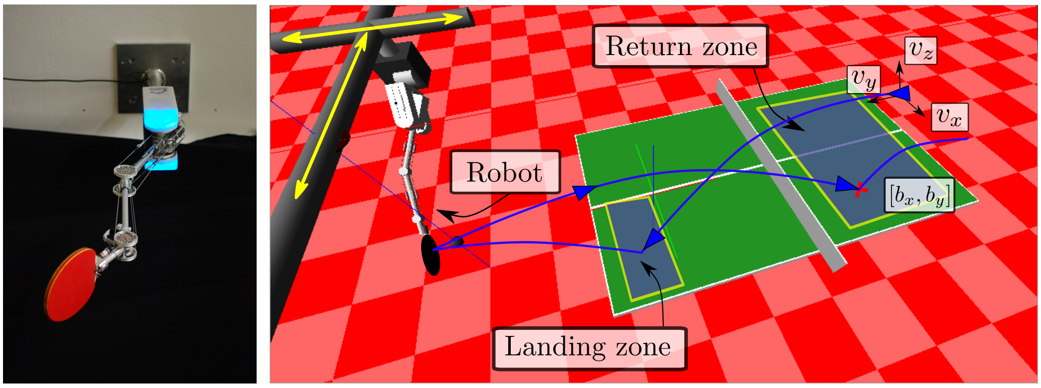


Figure 12: (a) Learning curves on the robot hockey task. GPREPS was able to learn the task within 120 interactions with the environment, while the model-free version of REPS was not able to find high-quality solutions. (b) The GPREPS learning curve on the real robot arm. The error bars represent the standard deviation using 5 independent evaluations.

We also evaluated the performance of GPREPS on the hockey task using a real KUKA lightweight arm, see Fig. 1. 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 unchanged, but we slightly altered the modeling of the environment. Due to the low sampling frequency of the Kinect sensor, we did not receive enough information about the exact contact model of the pucks. 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. 12(b). As we can see, the robot adapts the lower-level policy parameters towards the optimum within a small low number of interactions with the real environment. The final reward is slightly different compared to the simulated environment due to the altered modeling and slightly distorted



(a)

(b)

Figure 13: (a) The BioRob Robot (b) 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 $\boldsymbol{s} = [\boldsymbol{v}^T, \boldsymbol{b}^T]^T$.

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.

6.6. Robot Table Tennis

In this task, we learn hitting strokes in a table tennis game with a simulated Biorob [25] arm (Fig. 13(a)). 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. The Biorob is a lightweight tendon-driven robot arm that, due to its small weight, can perform highly dynamic movements. The simulated robot can be seen in Figure 13(b). The construction of the real robot platform is ongoing work. In simulation, we simulated the ball with a standard ballistic flight model with 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. Thus, the context is defined as $\boldsymbol{s} = [v_x, v_y, v_z, b_x, b_y]^T$. For an illustration of the task see Fig. 13. We chose the range of the initial velocities such that the incoming ball bounces only once on the forehand side of the table. To learn the task, we use DMPs where we initialize the DMP weight parameters by kinesthetic teach-in, such that the movement resembles a forehand hitting motion. We only learn the final positions and final velocities of the DMP trajectories, furthermore the τ time-scaling parameter and the starting time point of the movement, altogether 18 parameters. The initialized policy is able to execute only the demonstrated movement without adapting to the incoming ball.

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. Such a model is sufficient for our modeling as we want to learn to return only 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 predicts 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. The task of the GP is now 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 ν. Finally, the fifth model predicts the landing position of the returned ball in case of a contact, which is detected by an 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 point 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.

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

$$R(\boldsymbol{\tau}, \boldsymbol{s}) = -c_1 \min ||\boldsymbol{\tau}_b - \boldsymbol{\tau}_r||_2 - c_2||\boldsymbol{b} - \boldsymbol{p}||_2 \tag{21}$$

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.

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 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 $\boldsymbol{c} = [c_1, c_2]^T$, such that in the beginning of the learning c_2 is negligible compared to c_1, but we add an extra constant penalty term. By doing so, 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.

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. 14. 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 into the net and the displacement from the desired target return position remains below 30cm. An example for an complete experiment outcome prediction is depicted in Fig. 16.

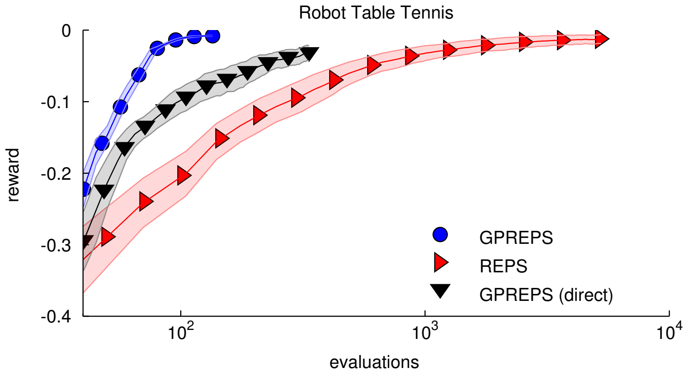


Figure 14: The learning curves of the table tennis experiment. REPS is able to learn a good policy after 4000 evaluations, but it sometimes learns a sub-optimal policy that hits the ball in the net. When using GPREPS, which directly predicts the reward from context-policy parameter pairs *(GPREPS (direct))*, the resulting policies showed highly varying performance, often getting stuck in local optima. However, GPREPS which can exploit the given structure of the experiment 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.

When using the *GPREPS (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. An animation of two different strikes learned with GPREPS is shown in Figure 15.

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 [8], 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 that allows balls with varying initial positions and we will evaluate GPREPS on the real robot platform.

6.7. Initialization and Limitations 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 $\pi(\boldsymbol{\omega}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{a} + \boldsymbol{s}\boldsymbol{A}, \boldsymbol{\Sigma})$, with parameters $\{\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}\}$. Initially, we set the linear model $\boldsymbol{A} = \boldsymbol{0}$ to zero and obtain the offset

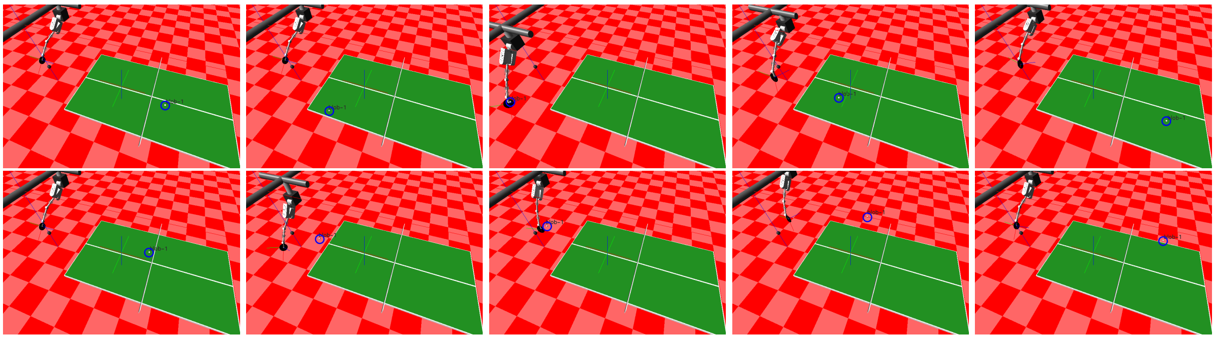


Figure 15: Animation of two shots to different targets and different serving positions of the ball learned with GPREPS.

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 is still safe for the robot. As 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.

Model based solutions with gradient-based policy updates, such as PILCO, can scale to higher dimensions as they neglect the exploration problem. With REPS we usually learn policies with not more than hundred parameters as the computation of the covariance matrix gets intractable. Thus, currently GPREPS cannot be applied for complex robot learning tasks with more than 100 parameters.

6.8. *Learning with Real Robots*

For all the evaluations presented in this paper we used robot simulators except for one task where we used a KUKA lightweight arm. For most cases it is convenient to use a robot simulator to demonstrate the learning efficiency, but ultimately our goal is to provide good real world results. As simulators typically use hand tuned models of the robot and its environment, skills learned with simulators might perform poorly on the real robot. This is mostly due to the inaccuracies of the hand crafted models, limited expert knowledge and the lack of model adaptation.

In our approach, we propose to learn the models by interacting with the environment. The resulting data driven modeling approach addresses the inaccuracies resulting from imperfectly tuned models and the lack of accurate expert knowledge. Moreover, it is able to adapt the learned models by continuously updating the model parameters using the most recent measurement data. We chose Gaussian Process regression as modeling approach, which can significantly reduce the bias coming from the limited representational power of mathematical models.

When working on real robot experiments, obtaining the measurement data for model learning is one of the most important challenge. With real world experiments the measured quantities are typically corrupted by noise, which need to be filtered to have a more reliable estimate. Furthermore, devices with lower sampling frequency provide less measurement data, which ultimately increases the complexity of the learned model. Interpolation might help to build more reliable models, but it can introduce a higher model bias. To avoid using poor quality models, it is advisable to validate the learned models before using them to simulate experiment rollouts.

In some tasks, it might not even be necessary to thoroughly model the whole experiment in order to infer the reward of a rollout, but it is sufficient to predict the relevant quantities for computing the reward from the policy and context parameters. We illustrated this idea for the ball throwing task, where we decomposed the reward function and solved several regression problems to obtain the models. This approach typically has a lower accuracy but a significantly faster computational time assuming long-term trajectory prediction with GP models is not necessary. On the other hand, due to the presumably lower quality models, the number of required evaluations might slightly increase. Nevertheless, for a new experiment it is advisable to begin with the simpler modeling approach to reduce the modeling effort and the amount of expert knowledge required.

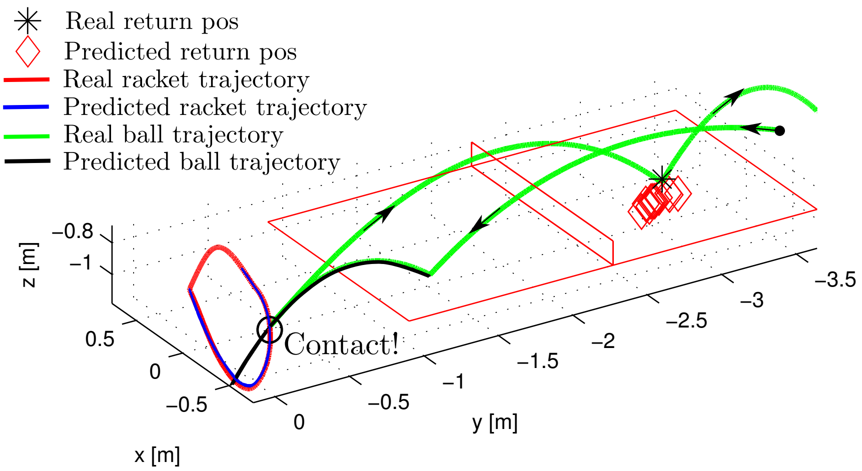


Figure 16: An example of prediction outcome with the table tennis experiment. The first and second model predicts the initial position and velocity of the incoming ball and its trajectory after bouncing back (**black** lines). The third and fourth model predicts the position (**blue** lines) and the 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.

7. Conclusion

We presented GPREPS, a novel model-based contextual policy search algorithm based on GP models and information-theoretic policy search. We learn an upper level policy that efficiently generalizes the lower level policy parameters ω over multiple contexts. GPREPS is based on REPS, an information-theoretic policy search algorithm. It 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 incorporate the 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.

Acknowledgement

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/20072013) under grant agreement #270327 (CompLACS). Marc Peter Deisenroth was supported by an Imperial College Junior Research Fellowship.

Appendix A. Derivation of Contextual Episode-based REPS

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

$$\begin{aligned} \max_{p} \quad & \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega})\mathcal{R}_{s\omega} d\boldsymbol{s}d\boldsymbol{\omega}, \\ \text{s.t.:} \quad & \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})} d\boldsymbol{s}d\boldsymbol{\omega} \le \epsilon, \\ & \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega})\boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{s}d\boldsymbol{\omega} = \hat{\boldsymbol{\phi}}, \\ & \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega}) d\boldsymbol{s}d\boldsymbol{\omega} = 1. \end{aligned} \tag{A.1}$$

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

$$\mathcal{L}(p,\eta,\boldsymbol{\theta}) = \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega})\mathcal{R}_{s\omega} d\boldsymbol{s}d\boldsymbol{\omega} + \eta\left(\epsilon - \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})} d\boldsymbol{s}d\boldsymbol{\omega}\right) \tag{A.2}$$

$$+\boldsymbol{\theta}^T\left(\hat{\boldsymbol{\phi}} - \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega})\boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{s}d\boldsymbol{\omega}\right) + \lambda\left(1 - \iint_{s,\omega} p(\boldsymbol{s},\boldsymbol{\omega}) d\boldsymbol{s}d\boldsymbol{\omega}\right). \tag{A.3}$$

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

$$p(\boldsymbol{s},\boldsymbol{\omega}) = q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{s\omega} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) \exp\left(-\frac{\eta+\lambda}{\eta}\right), \tag{A.4}$$

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

$$\exp\left(-\frac{\eta+\lambda}{\eta}\right) = \left[\iint_{s,\omega} q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{s\omega} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) d\boldsymbol{s}d\boldsymbol{\omega}\right]^{-1}. \tag{A.5}$$

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

$$g(\eta,\boldsymbol{\theta},\lambda) = \eta + \lambda + \eta\epsilon + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}} = \eta \log \exp\left(\frac{\eta+\lambda}{\eta}\right) + \eta\epsilon + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}} \tag{A.6}$$

Setting Equation (A.5) into the dual we can eliminate the λ parameter and obtain the dual function

$$g(\eta,\boldsymbol{\theta}) = \eta \log\left(\iint_{s,\omega} q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{s\omega} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right) d\boldsymbol{s}d\boldsymbol{\omega}\right) + \eta\epsilon + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}. \tag{A.7}$$

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

$$g(\eta,\boldsymbol{\theta};\mathcal{D}) = \eta \log\left(\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{\mathcal{R}_{s\omega}^{[i]} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right)\right) + \eta\epsilon + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}. \tag{A.8}$$

The dual function is convex in η and $\boldsymbol{\theta}$ [30]. To solve the original optimization problem, we need to minimize $g(\eta,\boldsymbol{\theta};\mathcal{D})$ such that $\eta > 0$ [6], 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

$$\frac{\partial g(\eta,\boldsymbol{\theta})}{\partial \eta} = \epsilon + \log \frac{1}{N}\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]}) - \frac{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]})(\mathcal{R}_{s\omega}^{[i]} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}^{[i]}))}{\eta\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]})}, \tag{A.9}$$

$$\frac{\partial g(\eta,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \hat{\boldsymbol{\phi}} - \frac{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]})\boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\sum_{i=1}^{N} Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]})}, \text{ with } Z(\boldsymbol{s}^{[i]},\boldsymbol{\omega}^{[i]}) = \exp\left(\frac{\mathcal{R}_{s\omega}^{[i]} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}^{[i]})}{\eta}\right). \tag{A.10}$$

References

[1] P. Abbeel, M. Quigley, and A. Y. Ng. Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, 2006.

[2] C. G. Atkeson and J. C. Santamaría. A Comparison of Direct and Model-Based Reinforcement Learning. In *Proceedings of the International Conference on Robotics and Automation*, 1997.

[3] J. A. Bagnell and J. C. Schneider. Covariant Policy Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[4] J. A. Bagnell and J. G. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceedings of the International Conference on Robotics and Automation*, 2001.

[5] J. Baxter and P. Bartlett. Reinforcement Learning in POMDP's via Direct Gradient Ascent. In *Proceedings of the 17th Intl. Conference on Machine Learning (ICML)*, pages 41–48, 2000.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] B.C. da Silva, G.D. Konidaris, and A.G. Barto. Learning Parameterized Skills. In *Proceedings of the International Conference on Machine Learning (ICML)*, June 2012.

[8] C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[9] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the International Conference on Machine Learning*, 2011.

[10] M. P. Deisenroth, C. E. Rasmussen, and D. Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Robotics: Science & Systems*, 2011.

[11] M.P. Deisenroth, P. Englert, J. Peters, and D. Fox. Multi-task policy search for robotics. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[12] M.P. Deisenroth, D. Fox, and C.E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[13] M.P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[14] P. Englert, A. Paraschos, J. Peters, and M. P. Deisenroth. Model-based Imitation Learning by Probabilistic Trajectory Matching. In *Proceedings of 2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[15] A. Gams and A. Ude. Generalization of Example Movements with Dynamic Systems. In *International Conference on Humanoid Robots (Humanoids)*, pages 28–33. IEEE, 2009.

[16] D.H Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *In Proceedings of IEEE International Conference on Robotics and Automation*, 2011.

[17] A. J. Ijspeert and S. Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems (NIPS)*. 2003.

[18] J. Ko and D. Klein. 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.

[19] J. Kober, K. Mülling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement Templates for Learning of Hitting and Batting. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[20] J. Kober, E. Oztop, and J. Peters. Reinforcement Learning to adjust Robot Movements to New Situations. In *Robotics: Science & Systems (RSS)*, 2010.

[21] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, pages 1–33, 2010.

[22] N. Kohl and P. Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the International Conference on Robotics and Automation*, 2003.

[23] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[24] 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.

[25] T. Lens. *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.

[26] K. Muelling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research (IJRR)*, (3):263–279, 2013.

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

[28] G. Neumann, W. Maass, and J. Peters. Learning Complex Motions by Sequencing Simpler Motion Templates. In *International Conference on Machine Learning (ICML)*, (ICML 2009), 2009.

[29] A. Y. Ng, H. Jin Kim, M. I. Jordan, and S. Sastry. Inverted Autonomous Helicopter Flight via Reinforcement Learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

[30] J. Peters, K. Mülling, and Y. Altun. Relative Entropy Policy Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2010.

[31] J. Peters and S. Schaal. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, (4):682–97, 2008.

[32] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[33] T. Rückstieß, F. Sehnke, Schaul T., D. Wierstra, S. Yi, and Schmidhuber J. Exploring Parameter Space in Reinforcement Learning. *PALADYN Journal of Behavioral Robotics*, 1(1):14 – 24, 2010.

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

[35] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmiedhuber. Parameter-Exploring Policy Gradients. *Neural Networks*, 23:551–559, 2010.

[36] E. Snelson and Z. Ghahramani. Sparse Gaussian Processes using Pseudo-Inputs. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

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

[38] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement Learning of Motor Skills in High Dimensions: a Path Integral Approach. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010.

[39] M. Titsias. Variational learning of inducing variables in sparse gaussian processes. *Journal of Machine Learning Research - Proceedings Track*, 5:567–574, 2009.

[40] A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transaction on Robotics*, 26(5):800–815, 2010.

[41] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Fitness Expectation Maximization. In *Lectue Notes in Computer Science, Parallel Problem Solving from Nature, PPSN X*, pages 337–346. Springer-Verlag, 2008.

[42] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

[43] S. Yi, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic Search using the Natural Gradient. In *Proceedings of the 26th International Conference On Machine Learning (ICML)*, pages 1161 – 1168, 2009.