Data-Efficient Generalization of Robot Skills with Contextual Policy Search

Andras Gabor Kupcsik
Department of Electrical and Computer Engineering, National University of Singapore
kupcsik@nus.edu.sg

Marc Peter Deisenroth and **Jan Peters** and **Gerhard Neumann**
Intelligent Autonomous Systems Lab, Technische Unversität Darmstadt
marc@ias.tu-darmstadt.de, peters@ias.tu-darmstadt.de, neumann@ias.tu-darmstadt.de

Abstract

In robotics, controllers make the robot solve a task within a specific context. The context can describe the objectives of the robot or physical properties of the environment and is always specified before task execution. To generalize the controller to multiple contexts, we follow a hierarchical approach for policy learning: A lower-level policy controls the robot for a given context and an upper-level policy generalizes among contexts. Current approaches for learning such upper-level policies are based on model-free policy search, which require an excessive number of interactions of the robot with its environment. More data-efficient policy search approaches are model based but, thus far, without the capability of learning hierarchical policies. We propose a new model-based policy search approach that can also learn contextual upper-level policies. Our approach is based on learning probabilistic forward models for long-term predictions. Using these predictions, we use information-theoretic insights to improve the upper-level policy. Our method achieves a substantial improvement in learning speed compared to existing methods on simulated and real robotic tasks.

Introduction

Policy search seeks a parameter vector $\boldsymbol{\omega}$ of a parametrized policy π that yields high expected long-term reward $\mathcal{R}_{\boldsymbol{\omega}}$ and has been successfully applied to learning movement skills (Peters and Schaal 2008; Theodorou et al. 2010; Kohl and Stone 2003; Kormushev et al. 2010). In this paper, we consider the *contextual policy search* scenario where policies are generalized to multiple contexts. The context specifies task variables that are specified before task execution but are relevant for determining the controller for the task. For example, the context might contain the objectives of the robot, such as a target location of the end-effector, or properties of the environment such as a mass to lift. In our hierarchical approach to contextual policy search, a lower-level control policy $\pi(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega})$ computes the control signals $\boldsymbol{u}$ for the robot given the state $\boldsymbol{x}$ of the robot. The lower-level policy with parameters $\boldsymbol{\omega}$ is applied in a specific context $\boldsymbol{s}$, e.g., throwing a ball to a *particular* location. To generalize the movement to *multiple* contexts $\boldsymbol{s}_i$, e.g., throwing a ball to different locations, we learn an upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s}_i)$ that selects the parameter vector $\boldsymbol{\omega}$ of the lower-level policy for a given context $\boldsymbol{s}_i$. A graphical model of contextual policy search is given in Fig. 1.

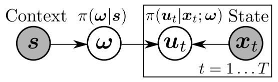
Context
$\pi(\boldsymbol{\omega}|\boldsymbol{s})$
$\pi(\boldsymbol{u}_t|\boldsymbol{x}_t;\boldsymbol{\omega})$
State
$\boldsymbol{s}$
$\boldsymbol{\omega}$
$\boldsymbol{u}_t$
$\boldsymbol{x}_t$
$t = 1 \ldots T$

Figure 1: Hierarchical graphical model for contextual policy search. The upper-level policy uses the context $\boldsymbol{s}$ to select the parameters of the lower-level policy $\pi(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega})$, which itself is used to generate the controls $\boldsymbol{u}_t$ for the robot.

Policy search methods are divided into model-free (Williams 1992; Peters and Schaal 2008; Kober and Peters 2010) and model-based (Bagnell and Schneider 2001; Abbeel et al. 2006; Deisenroth and Rasmussen 2011) approaches. Model-free policy search methods update the policy by executing roll-outs on the real system and collecting the corresponding rewards. Model-free policy search does not need an elaborate model of the robot's dynamics and they do not suffer from optimization bias as much as model-based methods. Model-free methods have been applied to contextual policy search (Kober et al. 2010; Neumann 2011). However, model-free policy search requires an excessive number interactions with the robot, which is often impracticable without informative prior knowledge. In contrast, model-based approaches learn a forward model of the robot and its environment (Deisenroth et al. 2011; Bagnell and Schneider 2001; Schneider 1997; Atkeson and Santamaría 1997) and use the model to predict the long term rewards. The policy is updated solely on these predictions, which makes them sensitive to model errors, potentially leading to poor performance of the learned policies (Atkeson and Santamaría 1997). One solution to alleviate this problem is to represent the uncertainty of the learned model itself by using probabilistic models (Schneider 1997; Deisenroth et al. 2011). Model-based approaches use data efficiently but they are restricted in the usable policy class. They can be used for learning a single parameter vector $\boldsymbol{\omega}$ of the lower-level policy, but they cannot learn an upper-level policy to generalize to multiple contexts.

In this paper, we introduce Gaussian Process Relative Entropy Policy Search (GPREPS), a new model-based contextual policy search method that can learn upper-level policies $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with a small number of robot interactions. GPREPS learns Gaussian Process (GP) forward models and uses sampling for long-term predictions. An information-theoretic policy update step is used to improve the upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$. The policy updates are based on the REPS algorithm (Peters et al. 2010). We applied GPREPS to learning a simulated throwing task with a 4-link robot arm and to a simulated and real-robot hockey task with a 7-link robot arm. Both tasks were learned using two orders of magnitude fewer interactions compared to previous model-free REPS, while yielding policies of higher quality.

Episodic Contextual Policy Learning

Contextual policy search (Kober and Peters 2010; Neumann 2011) aims to adapt the parameters $\boldsymbol{\omega}$ of a lower-level policy to the current context $\boldsymbol{s}$. An upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ chooses parameters $\boldsymbol{\omega}$ that maximize the expected reward

$$J = \mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega}}[\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}] = \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} \mu(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}},$$

where $\mu(\boldsymbol{s})$ is the distribution over the contexts and

$$\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} = \mathbb{E}_{\boldsymbol{\tau}}[r(\boldsymbol{\tau},\boldsymbol{s})|\boldsymbol{s},\boldsymbol{\omega}] = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})r(\boldsymbol{\tau},\boldsymbol{s}), \quad (1)$$

is the expected reward of the whole episode when using parameters $\boldsymbol{\omega}$ in context $\boldsymbol{s}$. In (1), $\boldsymbol{\tau}$ denotes a trajectory, $p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$ a trajectory distribution, and $r(\boldsymbol{\tau},\boldsymbol{s})$ a user-specified reward function. The trajectory distribution might still depend on the context $\boldsymbol{s}$, which can also describe environmental variables. After choosing the parameter vector $\boldsymbol{\omega}$ at the beginning of an episode, the lower-level policy $\pi(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega})$ determines the controls $\boldsymbol{u}$ during the episode based on the state $\boldsymbol{x}$ of the robot.

In the following, we briefly introduce the model-free episodic REPS algorithm for contextual policy search.

Contextual Episodic REPS

The key insight of REPS (Peters et al. 2010) is to bound the difference between two subsequent trajectory distributions in the policy update step to avoid overly aggressive policy update steps. In REPS, the policy update step directly results from bounding the relative entropy between the new and the previous trajectory distribution.

In the episodic contextual setting, the context $\boldsymbol{s}$ and the parameter $\boldsymbol{\omega}$ uniquely determine the trajectory distribution (Daniel et al. 2012). For this reason, the trajectory distribution can be abstracted as joint distribution over the parameter vector $\boldsymbol{\omega}$ and the context $\boldsymbol{s}$, i.e., $p(\boldsymbol{s},\boldsymbol{\omega}) = p(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})$. REPS optimizes over this joint distribution $p(\boldsymbol{s},\boldsymbol{\omega})$ of contexts and parameters $\boldsymbol{\omega}$ to maximize the expected reward

$$J = \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}, \quad (2)$$

with respect to p while bounding the relative entropy to the previously observed distribution $q(\boldsymbol{s},\boldsymbol{\omega})$, i.e.,

$$\epsilon \geq \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega})}{q(\boldsymbol{s},\boldsymbol{\omega})}. \quad (3)$$

As the context distribution $p(\boldsymbol{s}) = \sum_{\boldsymbol{\omega}} p(\boldsymbol{s},\boldsymbol{\omega})$ is specified by the learning problem and given by $\mu(\boldsymbol{s})$, the constraints

$$\forall \boldsymbol{s} : p(\boldsymbol{s}) = \mu(\boldsymbol{s})$$

need to be added to the REPS optimization problem.

To fulfill these constraints also for continuous or high-dimensional discrete state spaces, REPS matches feature averages instead of single probability values, i.e.,

$$\sum\nolimits_{\boldsymbol{s}} p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}) = \hat{\boldsymbol{\phi}}, \quad (4)$$

where $\boldsymbol{\phi}(\boldsymbol{s})$ is a feature vector describing the context and $\hat{\boldsymbol{\phi}}$ is the feature vector averaged over all observed contexts. The REPS optimization problem for the episodic contextual policy search is defined as maximizing (2) with respect to p under the constraints (3) and (4). It can be solved by the method of Lagrange multipliers and minimizing the dual function g of the optimization problem (Daniel et al. 2012). REPS yields the closed-form solution

$$p(\boldsymbol{s},\boldsymbol{\omega}) \propto q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\tfrac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - V(\boldsymbol{s})}{\eta}\right) \quad (5)$$

to (2) where $V(\boldsymbol{s}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$ is a value function (Peters et al. 2010). The parameters η and $\boldsymbol{\theta}$ are Lagrangian multipliers used for the constraints (3) and (4), respectively. These parameters are found by optimizing the dual function $g(\eta,\boldsymbol{\theta})$. The function $V(\boldsymbol{s})$ emerges from the feature constraint in (4) and depends linearly on the features $\boldsymbol{\phi}(\boldsymbol{s})$ of the context. It serves as context dependent baseline which is subtracted from the reward signal.

Learning Upper-Level Policies

The optimization defined by the REPS algorithm is only performed on a discrete set of samples $\mathcal{D} = \{\boldsymbol{s}^{[i]}, \boldsymbol{\omega}^{[i]}, \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]}\}$, $i = 1,\ldots,N$. The resulting probabilities $p^{[i]} \propto \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[i]} - V(\boldsymbol{s}^{[i]})}{\eta}\right)$ of these samples are used to weight the samples. Note that the distribution $q(\boldsymbol{s},\boldsymbol{\omega})$ can be skipped from the weighting as we already sampled from $q(\boldsymbol{s},\boldsymbol{\omega})$. The upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ is subsequently learned by performing a weighted maximum-likelihood estimate for the parameters $\boldsymbol{\omega}$ of the policy. In our experiments, we use a linear-Gaussian model to represent the upper-level policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, i.e.,

$$\pi(\boldsymbol{\omega}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{a} + \boldsymbol{A}\boldsymbol{s}, \boldsymbol{\Sigma}). \quad (6)$$

In this example, the parameters of the upper-level policy are given as $\{\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}\}$.

The sample-based implementation of REPS can be problematic if the variance of the reward is high as the expected reward $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}$ is approximated by a single roll-out. The subsequent exponential weighting of the reward can lead to risk-seeking policies.

Exploiting Reward Models as Prior Knowledge

The reward function $r(\boldsymbol{\tau},\boldsymbol{s})$ is typically specified by the user and is assumed known. If the outcome $\boldsymbol{\tau}$ of an experiment is independent of the context variables, the reward model can easily generate additional samples for the policy update. In this case, we can use the outcome $\boldsymbol{\tau}^{[i]}$ to evaluate the reward $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^{[j]} = r(\boldsymbol{\tau}^{[i]}, \boldsymbol{s}^{[j]})$ for multiple contexts $\boldsymbol{s}^{[j]}$ instead of just for a single context $\boldsymbol{s}^{[i]}$. 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.

```
GPREPS Algorithm
Input: relative entropy bound $\epsilon$, initial policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$,
number of iterations $K$.
for $k = 1, \ldots, K$
    Collect Data:
        Observe context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s}), i = 1, \ldots, N$
        Execute policy with $\boldsymbol{\omega}^{[i]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[i]})$
    Train forward models, estimate $\hat{\mu}(\boldsymbol{s})$
    for $j = 1, \ldots, M$
        Predict Rewards:
            Draw context $\boldsymbol{s}^{[j]} \sim \hat{\mu}(\boldsymbol{s})$
            Draw lower-level parameters $\boldsymbol{\omega}^{[j]} \sim \pi(\boldsymbol{\omega}|\boldsymbol{s}^{[j]})$
            Predict L trajectories $\boldsymbol{\tau}_j^{[l]}|\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]}$
            Compute $\mathcal{R}_{\boldsymbol{s\omega}}^{[j]} = \sum_l r(\boldsymbol{\tau}_j^{[l]}, \boldsymbol{s}^{[j]})/L$
    end
    Update Policy:
        Optimize dual function:
            $[\eta, \boldsymbol{\theta}] = \text{argmin}_{\eta', \boldsymbol{\theta}'} g(\eta', \boldsymbol{\theta}')$
        Calculate sample weighting:
            $p^{[j]} \propto \exp\left(\frac{\mathcal{R}_{\boldsymbol{s\omega}}^{[j]} - V(\boldsymbol{s}^{[j]})}{\eta}\right), j = 1, \ldots, M$
        Update policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with weighted ML
end
Output: policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$
```

Table 1: In each iteration, we generate N trajectories on the real system and update our data set for learning forward models. Based on the updated forward model, we create M artificial samples and predict their corresponding rewards. These artificial samples are subsequently used for optimizing the dual function $g(\eta, \boldsymbol{\theta})$ updating the upper-level policy by weighted maximum likelihood estimates.

Gaussian Process REPS

In our approach, we enhance the model-free REPS algorithm with learned probabilistic forward models to increase the data efficiency and the learning speed. The GPREPS algorithm is summarized in Tab. 1. With data from the real robot, we learn the reward function and forward models of the robot. Moreover, we estimate a distribution $\hat{\mu}(\boldsymbol{s})$ over the contexts, such that we can easily generate new context samples $\boldsymbol{s}^{[j]}$. Subsequently, we sample a parameter vector $\boldsymbol{\omega}^{[j]}$ and use the learned probabilistic forward models to sample L trajectories $\boldsymbol{\tau}_j^{[l]}$ for the given context-parameter pair. Finally, we obtain additional artificial samples $(\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]}, \mathcal{R}_{\boldsymbol{s\omega}}^{[j]})$, where $R_{\boldsymbol{s\omega}}^{[j]} = \sum_l r(\boldsymbol{\tau}_j^{[l]}, \boldsymbol{s}^{[j]})/L$, that are used for updating the policy. The learned forward models are implemented as GPs. The most relevant parameter GPREPS is the relative-entropy bound ϵ that scales the exploration of the algorithm. Small values ϵ encourage continuing exploration and allow for collecting more data to improve the forward models. Large ϵ causes increasingly greedy policy updates.

We use the observed trajectories from the real system solely to update the learned forward models but not for evaluating expected rewards. Hence, GPREPS avoids the typical problem of the model-free REPS approach of high variance in the rewards. GPREPS is well suited for our sample-based prediction of expected rewards as several artificial samples $(\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]})$ can be evaluated in parallel.

Learning GP Forward Models

We learn forward models to predict the 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 $\boldsymbol{y}_{t+1} = \boldsymbol{f}(\boldsymbol{y}_t, \boldsymbol{u}_t) + \epsilon$, where $\boldsymbol{y} = [\boldsymbol{x}, \boldsymbol{b}]$ 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 ϵ 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 its environment. For example, for throwing a ball, we use the prior knowledge that the initial state $\boldsymbol{b}$ of the ball is a function of the state $\boldsymbol{x}$ of the robot at the time point t_r when releasing the ball. We use a separate forward model to predict the initial state $\boldsymbol{b}$ of the ball from the robot state $\boldsymbol{x}$ at time point t_r and a forward model for predicting the free dynamics $h(\boldsymbol{b}_t)$ of the ball which is independent of the robot's state $\boldsymbol{x}_t$, $t > t_r$.

Our learned models are implemented as probabilistic non-parametric Gaussian processes, which are trained using the standard approach of evidence maximization (Rasmussen and Williams 2006). As training data, we use data collected from interacting with the real robot. As a GP captures uncertainty about the learned model itself, averaging out this uncertainty reduces the effect of model errors and results in robust policy updates (Deisenroth and Rasmussen 2011). To reduce the computational demands of learning the GP models, we use sparse GP models with pseudo inputs (Snelson and Ghahramani 2006).

Trajectory and Reward Predictions

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

$$\mathcal{R}_{\boldsymbol{s\omega}} = \int_{\boldsymbol{\tau}} \hat{p}(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s}) r(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau} \tag{7}$$

for a given parameter vector $\boldsymbol{\omega}$ executed in context $\boldsymbol{s}$. The probability $\hat{p}(\boldsymbol{\tau}|\boldsymbol{\omega}, \boldsymbol{s})$ of a trajectory is now estimated using learned forward models. Our approach to predicting trajectories is based on sampling. To predict a trajectory, we iterate the following procedure: First, we compute the GP predictive distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{[l]}, \boldsymbol{u}_t^{[l]})$ for a given state-action pair $\boldsymbol{x}_t^{[l]}, \boldsymbol{u}_t^{[l]}$. The next state $\boldsymbol{x}_{t+1}^{[l]}$ in trajectory $\boldsymbol{\tau}^{[l]}$ is subsequently drawn from this distribution. To estimate the expected reward $\mathcal{R}_{\boldsymbol{s\omega}}{}^{[j]}$ in (7) for a given context-parameter pair $(\boldsymbol{s}^{[j]}, \boldsymbol{\omega}^{[j]})$, we use several trajectories $\boldsymbol{\tau}^{[j,l]}$. For each sample trajectory $\boldsymbol{\tau}^{[j,l]}$, a reward $r(\boldsymbol{\tau}^{[j,l]}, \boldsymbol{s}^{[j]})$ is obtained. Averaging over these rewards yields an estimate of $\mathcal{R}_{\boldsymbol{s\omega}}{}^{[j]}$. Reducing the variance of this estimate requires many samples. However, sampling can be easily parallelized. In the limit of infinitely many samples, this estimate is unbiased.

As alternative to sampling-based predictions, deterministic inference methods, such as moment matching, can be used. For instance, the PILCO policy search framework (Deisenroth and Rasmussen 2011) approximates the expectation given in (7) analytically by moment matching (Deisenroth et al. 2012). In particular, PILCO approximates the state distributions $p(\boldsymbol{x}_t)$ at each time step by Gaussians. Due to this approximation, the predictions of the expected trajectory reward $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}$ can be biased. While moment matching allows for closed-form predictions, the policy class and the reward models are restricted due to the computational constraints.

In our experiments, we could obtain 300 sample trajectories within the same computation time of a single prediction of the deterministic inference algorithm. The accuracy of the deterministic inference is usually met after obtaining around 100 sample trajectories. Using a massive parallel hardware (e.g., a GPU), we obtained 7000 trajectories with the same amount of computation time.

Results

We evaluated our data-efficient contextual policy search method on a context-free comparison task and two contextual motor skill learning tasks. As the lower-level policy needs to scale to anthropomorphic robotics, we implement them using the Dynamic Movement Primitive (Ijspeert and Schaal 2003) approach, which we will now briefly review.

Dynamic Movement Primitives

To parametrize the lower-level policy we use an extension (Kober et al. 2010) to Dynamic Movement Primitives (DMPs). A DMP is a spring-damper system that is modulated by a forcing function $f(z; \boldsymbol{v}) = \boldsymbol{\Phi}(z)^T \boldsymbol{v}$. The variable z is the phase of the movement. The parameters $\boldsymbol{v}$ specify the shape of the movement and can be learned by imitation. The final position and velocity of the DMP can be adapted by changing additional hyper-parameters $\boldsymbol{y}_f$ and $\dot{\boldsymbol{y}}_f$ of the DMP. The execution speed of the movement is determined by the time constant τ. For a more detailed description of the DMP framework we refer to (Kober et al. 2010).

After obtaining a desired trajectory through the DMPs, the trajectory is followed by a feedback controller, i.e., the lower-level control policy.

4-Link Pendulum Balancing Task

In the following, we consider the task of balancing a 4-link planar pendulum. The objective was to learn a linear control policy that was move a 4-link pendulum to the desired upright position from randomly distributed starting angles around the upright position. The 4-link pendulum had a total length of 2m and a mass of 70kg. We used a quadratic reward function $r(\boldsymbol{x}) = -\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x}$ for each time step.

We compared the performance of GPREPS to the PILCO algorithm (Deisenroth et al. 2011), the state-of-the-art model-based policy search method. As PILCO cannot be applied to the contextual policy search setting, we used only a single context. We initialized both PILCO and GPREPS with 6s of real experience using a *random* policy. With GREPS we use 400 artificial samples, each with 10 sample trajectories. We also learned policies with the model-free version of REPS (Peters et al. 2010). Tab. 2 shows the required amount of experience for achieving several reward limits. The model-based methods quickly found stable controllers, while REPS required two orders of magnitude more trials until convergence.

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

Ball-Throwing Task

In the ball-throwing task, we consider the problem of contextual policy search. The goal of the 4-DOF planar robot was to hit a randomly placed target at position $\boldsymbol{s}$ with a ball. Here $\boldsymbol{s}$ is a two-dimensional coordinate, which also defines the context of the task. The x and y-dimensional coordinates vary between $5\,\mathrm{m}$–$15\,\mathrm{m}$ and between $0\,\mathrm{m}$–$4\,\mathrm{m}$ from the robot base. Our 4-DOF robot coarsely modeled a human and had an ankle, knee, hip and shoulder joint. The lengths of the links were given by $\boldsymbol{l} = [0.5, 0.5, 1.0, 1.0]\,\mathrm{m}$ and the masses by $\boldsymbol{m} = [17.5, 17.5, 26.5, 8.5]\,\mathrm{kg}$.

In this experiment, we used GPREPS to find DMP parameters for throwing a ball to multiple targets while maintaining balance. The reward function was defined as

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

The first term punishes minimum distance of the ball to the target $\boldsymbol{s}$. The second term describes a punishment term to prevent the robot from falling over. The last term favors energy-efficient movements.

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 $\boldsymbol{\omega}$.

GPREPS learned a forward model consisting of three components: the dynamics model of the robot, the free dynamics of the ball, and the model of how the robot's state influences the initial position and velocity of the ball at the given release time t_r. These models were used to predict ball trajectories for a given parameter $\boldsymbol{\omega}$.

The policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ was initialized such that it was expected to throw the ball approximately $5\,\mathrm{m}$ without maintaining balance, which led to high penalties. Fig. 2 shows the learned motion sequence for two different targets. GPREPS learned to accurately hit the target for a wide range of different targets. The displacement for targets above $\boldsymbol{s} = [13, 3]\,\mathrm{m}$ could raise up to $0.5\,\mathrm{m}$, otherwise the maximal error was smaller than $10\,\mathrm{cm}$. The policy chose different DMP

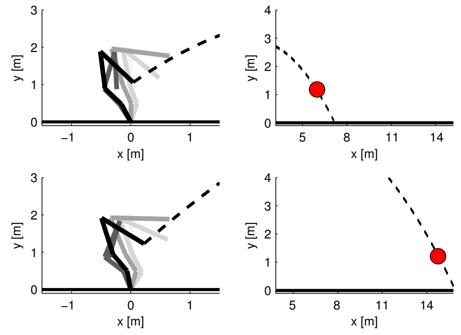

Figure 2: Throwing motion sequence. The robot releases the ball after the specified release time and hits different targets with high accuracy.

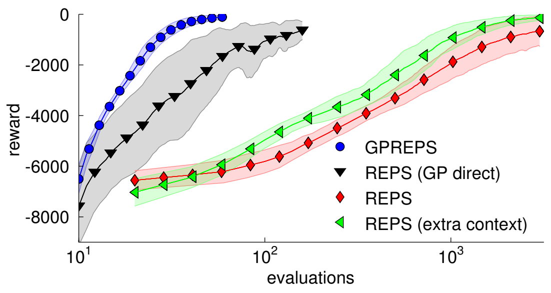

Figure 3: Learning curves for the ball-throwing problem. The shaded regions represent the standard deviation of the rewards over 20 independent trials. GPREPS converged after 30–40 interactions with the environment, while REPS required ≥ 5000 interactions. Using the reward model to generate additional samples for REPS led to better final policies, but could not compete with GPREPS in terms of learning speed. Learning the direct reward model $\mathcal{R}_{s\omega} = f(s, \omega)$ yielded faster learning than model-free REPS, but the quality of the final policy is limited.

parametrizations and release times for different target positions. To illustrate this effect we show two target positions $s_1 = [6, 1.1]\,\mathrm{m}$, $s_2 = [14.5, 1.2]\,\mathrm{m}$ in Fig. 2. For the target 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. 3. In addition, we evaluated REPS using the known reward model $r(\tau, s)$ to generate additional samples with randomly sampled contexts $s^{[i]}$ as described previously. We denote these experiments as *extra context*. We also evaluated REPS when learning the expected reward model directly $\mathcal{R}_{s\omega} = f(s, \omega)$ as a function of the context and parameters ω with a GP (*GP direct*). Using the learned reward model we evaluate artificial samples $(s^{[i]}, \omega^{[i]})$. Fig. 3 shows that REPS converged to a good solution after 5000 episodes in most cases. In a few instances, however, we observed premature convergence resulting in 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 additional samples and 20 sample trajectories per sample for each policy update. GPREPS converged to a good solution in all cases after 30–40 real evaluations. In contrast, while directly learning $\mathcal{R}_{s\omega}$ also resulted in an improved learning speed, we observed a highly varying, on average lower quality in the resulting policies (*GP direct*). This result confirms our intuition that decomposing the forward model into multiple components simplifies the learning task.

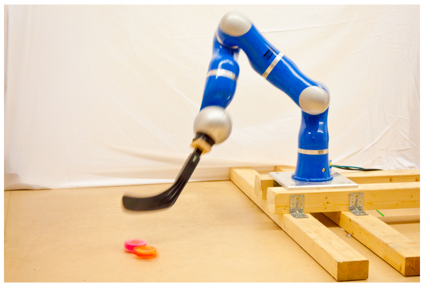

Figure 4: KUKA lightweight arm shooting hockey pucks.

Robot Hockey with Simulated Environment

In this task, the KUKA lightweight robot arm was equipped with a racket, see Fig. 4, and had to learn shooting hockey pucks. The objective was to make a *target puck* move a specified distance. Since the target puck was out of reach, it could only be moved indirectly by shooting a second hockey puck at the target puck. The initial location $[b_x, b_y]^T$ of the target puck was varied in both dimensions as well as the distance d^* the target puck had to be shoot. Consequently, the context was given by $s = [b_x, b_y, d^*]^T$. The simulated robot task is depicted in Fig. 5.

We obtained the shape parameters v of the DMP by imitation learning and kept them fixed during learning. The learning algorithms had to adapt the final positions y_f, velocities $\dot{y}_f$ and the time scaling parameter τ of the DMPs, resulting in a 15-dimensional lower-level policy parameter ω. The reward function was the negative minimum squared distance of the robot's puck to the target puck during a play. In addition, we penalized the squared difference between the desired travel distance and the distance the target puck actually moved. We chose the position of the target puck to be within 0.5m from its initial position, i.e., $[2, 1]$m from the robot's base. The desired distance to move the target puck ranged between $0\,\mathrm{m}$ and $1\,\mathrm{m}$.

GPREPS first learned a forward model to predict the initial position and velocity of the first puck after contact with the racket and a travel distance of $20\,\mathrm{cm}$. Subsequently, GPREPS learned the free dynamics model of both pucks and the contact model of the pucks. Based on the position and velocities of the pucks before contact, the velocities of both

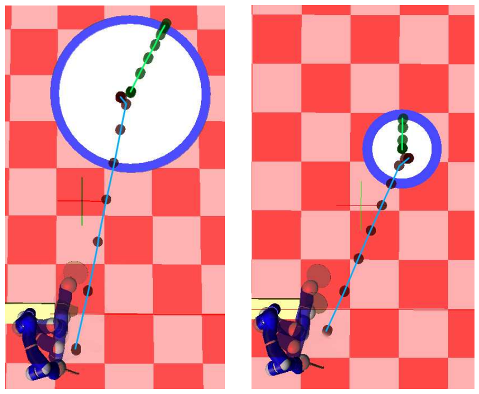

Figure 5: Robot hockey task. The robot shot a puck at the target puck to make the target puck move for a specified distance. Both, the initial location of the target puck $[b_x, b_y]^T$ and the desired distance d^* to move the puck were varied. The context was $\boldsymbol{s} = [b_x, b_y, d^*]$. The learned skill for two different contexts $\boldsymbol{s}$ is shown, where the robot learned to (indirectly) place the target puck at the desired distance.

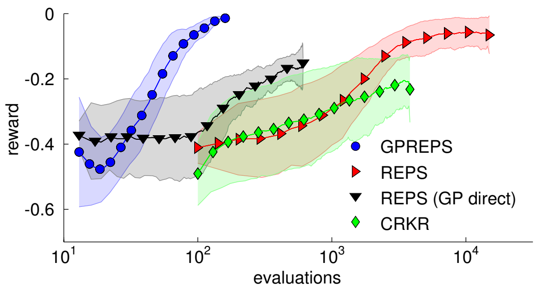

Figure 6: Learning curves on the robot hockey task. GPREPS was able to learn the task within 120 interactions with the environment, while the model-free version of REPS was not able to find high-quality solutions.

pucks were predicted after contact with the learned model. With GPREPS we used 500 parameter samples, each with 20 sample trajectories.

We compared GPREPS to directly predicting the reward $\mathcal{R}_{s\omega}$, model-free REPS and CrKR (Kober et al. 2010), a state-of-the-art model-free contextual policy search method. The resulting learning curves are shown in Fig. 6. GPREPS learns to accurately hit the second puck within 50 interactions with the environment, while to learn the whole task around 120 evaluations were needed. The model-free version of REPS needed approximately 10000 interactions. Directly predicting the rewards resulted in faster convergence but the resulting policies still showed a poor performance (*GP direct*). CrKR used 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.

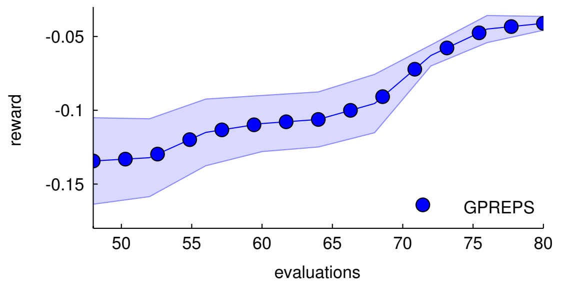

Figure 7: GPREPS learning curve on the real robot arm.

The learned movement is shown in Fig. 5 for two different contexts. After 120 evaluations, GPREPS placed the target puck accurately at the desired distance with a displacement error $\leq 5\,\mathrm{cm}$.

Robot Hockey with Real Environment

Finally, we evaluated the performance of GPREPS on the hockey task using a real KUKA lightweight arm, see Fig. 4. 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 and subsequently learned the GP models. The desired distance to move the target puck ranged between 0 m and 0.6 m.

The learning curve of GPREPS for the real robot is shown in Fig. 7, which shows that the robot clearly improved its initial policy within a small number of real robot experiments.

Conclusion

We presented GPREPS, a novel model-based contextual policy search algorithm. GPREPS learns hierarchical policies and is based on information-theoretic policy updates. Moreover, GPREPS exploits learned probabilistic forward models of the robot and its environment to predict expected rewards. 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 easily parallelizable and does not limit the policy class or reward model.

We demonstrated that for learning high-quality contextual policies, GPREPS exploits the learned forward models to significantly reduce the required amount of experience from the real robot compared to state-of-the-art model-free contextual policy search approaches. 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 opens the door to many new applications of model-based policy search.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 270327.

References

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

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

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

Daniel, C.; Neumann, G.; and Peters, J. 2012. Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics*.

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

Deisenroth, M. P.; Turner, R.; Huber, M.; Hanebeck, U. D.; and Rasmussen, C. E. 2012. Robust Filtering and Smoothing with Gaussian Processes. *IEEE Transactions on Automatic Control* 57(7):1865–1871.

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

Ijspeert, A. J., and Schaal, S. 2003. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems*.

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

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

Kober, J.; Oztop, E.; and Peters, J. 2010. Reinforcement Learning to adjust Robot Movements to New Situations. In *Robotics: Science & Systems*.

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

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

Neumann, G. 2011. Variational Inference for Policy Search in Changing Situations. In *Proceedings of the International Conference on Machine Learning*.

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

Peters, J.; Mülling, K.; and Altun, Y. 2010. Relative Entropy Policy Search. In *Proceedings of the National Conference on Artificial Intelligence*.

Rasmussen, C. E. and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. MIT Press.

Schaal, S.; Peters, J.; Nakanishi, J.; and Ijspeert, A. J. 2003. Learning Movement Primitives. In *International Symposium on Robotics Research*.

Schneider, J. G. 1997. Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *Advances in Neural Information Processing Systems*.

Snelson, E., and Ghahramani, Z. 2006. Sparse Gaussian Processes using Pseudo-Inputs. In *Advances in Neural Information Processing Systems*.

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

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