Gaussian Processes for Data-Efficient Learning in Robotics and Control

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen

Abstract—Autonomous learning has been a promising direction in control and robotics for more than a decade since data-driven learning allows to reduce the amount of engineering knowledge, which is otherwise required. However, autonomous reinforcement learning (RL) approaches typically require many interactions with the system to learn controllers, which is a practical limitation in real systems, such as robots, where many interactions can be impractical and time consuming. To address this problem, current learning approaches typically require task-specific knowledge in form of expert demonstrations, realistic simulators, pre-shaped policies, or specific knowledge about the underlying dynamics. In this paper, we follow a different approach and speed up learning by extracting more information from data. In particular, we learn a probabilistic, non-parametric Gaussian process transition model of the system. By explicitly incorporating model uncertainty into long-term planning and controller learning our approach reduces the effects of model errors, a key problem in model-based learning. Compared to state-of-the art RL our model-based policy search method achieves an unprecedented speed of learning. We demonstrate its applicability to autonomous learning in real robot and control tasks.

Index Terms—Policy search, robotics, control, Gaussian processes, Bayesian inference, reinforcement learning

1 Introduction

One of the main limitations of many current reinforcement learning (RL) algorithms is that learning is prohibitively slow, i.e., the required number of interactions with the environment is impractically high. For example, many RL approaches in problems with low-dimensional state spaces and fairly benign dynamics require thousands of trials to learn. This *data inefficiency* makes learning in real control/robotic systems impractical and prohibits RL approaches in more challenging scenarios.

Increasing the data efficiency in RL requires either task-specific prior knowledge or extraction of more information from available data. In this paper, we assume that expert knowledge (e.g., in terms of expert demonstrations [48], realistic simulators, or explicit differential equations for the dynamics) is unavailable. Instead, we carefully model the observed dynamics using a general flexible nonparametric approach.

Generally, model-based methods, i.e., methods which learn an explicit dynamics model of the environment, are more promising to efficiently extract valuable information from available data [5] than model-free methods, such as Q-learning [55] or TD-learning [52]. The main reason why model-based methods are not widely used in RL is that they can suffer severely from *model errors*, i.e., they inherently assume that the learned model resembles the real environment sufficiently accurately [5], [48], [49]. Model errors are especially an issue when only a few samples and no informative prior knowledge about the task are available. Fig. 1 illustrates how model errors can affect learning. Given a small data set of observed transitions (left), multiple transition functions plausibly could have generated them (center). Choosing a single deterministic model has severe consequences: Long-term predictions often leave the range of the training data in which case the predictions become essentially arbitrary. However, the deterministic model claims them with full confidence! By contrast, a probabilistic model places a posterior distribution on plausible transition functions (right) and expresses the level of uncertainty about the model itself.

When learning models, considerable model uncertainty is present, especially early on in learning. Thus, we require *probabilistic* models to express this uncertainty. Moreover, model uncertainty needs to be incorporated into planning and policy evaluation. Based on these ideas, we propose Probabilistic Inference for Learning Control (PILCO), a model-based policy search method [15], [16]. As a probabilistic model we use nonparametric Gaussian processes (GPs) [47]. PILCO uses computationally efficient deterministic approximate inference for long-term predictions and policy evaluation. Policy improvement is based on *analytic* policy gradients. Due to probabilistic modeling and inference PILCO achieves unprecedented learning efficiency in continuous state-action domains and, hence, is directly applicable to complex mechanical systems, such as robots.

In this paper, we provide a detailed overview of the key ingredients of the PILCO learning framework. In particular, we assess the quality of two different approximate inference methods in the context of policy search. Moreover, we give

- *M.P. Deisenroth is with the Department of Computing, Imperial College London, 180 Queen's Gate, London SW72AZ, United Kingdom, and the Department of Computer Science, TU Darmstadt, Germany.*
- *D. Fox is with the Department of Computer Science & Engineering, University of Washington, Box 352350, Seattle, WA 98195-2350.*
- *C.E. Rasmussen is with the Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB21PZ, United Kingdom.*

Manuscript received 15 Sept. 2012; revised 6 May 2013; accepted 20 Oct. 2013. Date of publication 3 Nov. 2013; date of current version 14 Jan. 2015. Recommended for acceptance by R.P. Adams, E. Fox, E. Sudderth, and Y. W. Teh.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2013.218

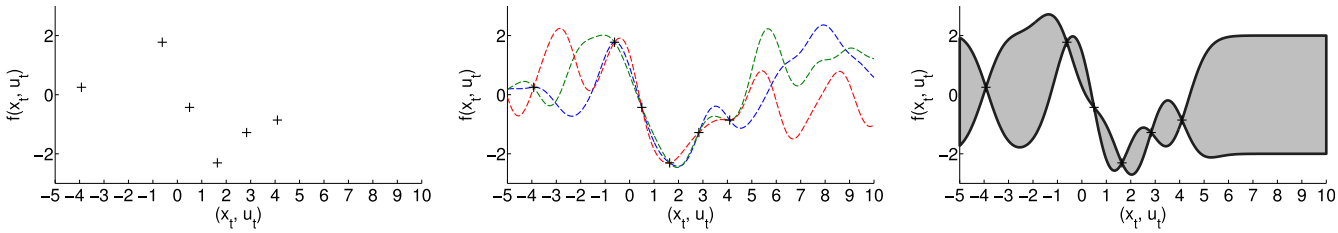


Fig. 1. Effect of model errors. Left: Small data set of observed transitions from an idealized one-dimensional representations of states and actions (x_t, u_t) to the next state $x_{t+1} = f(x_t, u_t)$. Center: Multiple plausible deterministic models. Right: Probabilistic model. The probabilistic model describes the uncertainty about the latent function by a probability distribution on the set of all plausible transition functions. Predictions with deterministic models are claimed with full confidence, while the probabilistic model expresses its predictive uncertainty by a probability distribution.

a concrete example of the importance of Bayesian modeling and inference for fast learning from scratch. We demonstrate that PILCO's unprecedented learning speed makes it directly applicable to realistic control and robotic hardware platforms.

This paper is organized as follows: After discussing related work in Section 2, we describe the key ideas of the PILCO learning framework in Section 3, i.e., the dynamics model, policy evaluation, and gradient-based policy improvement. In Section 4, we detail two approaches for long-term predictions for policy evaluation. In Section 5, we describe how the policy is represented and practically implemented. A particular cost function and its natural exploration/exploitation trade-off are discussed in Section 6. Experimental results are provided in Section 7. In Section 8, we discuss key properties, limitations, and extensions of the PILCO framework before concluding in Section 9.

2 Related Work

Controlling systems under parameter uncertainty has been investigated for decades in robust and adaptive control [4], [35]. Typically, a certainty equivalence principle is applied, which treats estimates of the model parameters as if they were the true values [58]. Approaches to designing adaptive controllers that explicitly take uncertainty about the model parameters into account are stochastic adaptive control [4] and dual control [23]. Dual control aims to reduce parameter uncertainty by explicit probing, which is closely related to the exploration problem in RL. Robust, adaptive, and dual control are most often applied to linear systems [58]; nonlinear extensions exist in special cases [22].

The specification of parametric models for a particular control problem is often challenging and requires intricate knowledge about the system. Sometimes, a rough model estimate with uncertain parameters is sufficient to solve challenging control problems. For instance, in [3], this approach was applied together with locally optimal controllers and temporal bias terms for handling model errors. The key idea was to ground policy evaluations using real-life trials, but not the approximate model.

All above-mentioned approaches to finding controllers require more or less accurate *parametric* models. These models are problem specific and have to be manually specified, i.e., they are not suited for learning models for a broad range of tasks. Nonparametric regression methods, however, are promising to automatically extract the important features of the latent dynamics from data. In [7], [49] locally weighted Bayesian regression was used as a nonparametric method for learning these models. To deal with model uncertainty, in [7] model parameters were sampled from the parameter posterior, which accounts for temporal correlation. In [49], model uncertainty was treated as noise. The approach to controller learning was based on stochastic dynamic programming in discretized spaces, where the model errors at each time step were assumed independent.

PILCO builds upon the idea of treating model uncertainty as noise [49]. However, unlike [49], PILCO is a policy search method and does not require state space discretization. Instead closed-form Bayesian averaging over infinitely many plausible dynamics models is possible by using nonparametric GPs.

Nonparametric GP dynamics models in RL were previously proposed in [17], [30], [46], where the GP training data were obtained from "motor babbling". Unlike PILCO, these approaches model global value functions to derive policies, requiring accurate value function models. To reduce the effect of model errors in the value functions, many data points are necessary as value functions are often discontinuous, rendering value-function based methods in high-dimensional state spaces often statistically and computationally impractical. Therefore, [17], [19], [46], [57] propose to learn GP value function models to address the issue of model errors in the value function. However, these methods can usually only be applied to low-dimensional RL problems. As a policy search method, PILCO does not require an explicit global value function model but rather searches directly in policy space. However, unlike value-function based methods, PILCO is currently limited to episodic set-ups.

3 Model-Based Policy Search

In this paper, we consider dynamical systems

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}, \quad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_w), \tag{1}$$

with continuous-valued states $\boldsymbol{x} \in \mathbb{R}^D$ and controls $\boldsymbol{u} \in \mathbb{R}^F$, i.i.d. Gaussian system noise $\boldsymbol{w}$, and unknown transition dynamics f. The policy search objective is to find a *policy/controller* $\pi : \boldsymbol{x} \mapsto \pi(\boldsymbol{x}, \theta) = \boldsymbol{u}$, which minimizes the *expected long-term cost*

$$J^\pi(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)], \quad \boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{2}$$

of following π for T steps, where $c(\boldsymbol{x}_t)$ is the cost of being in state $\boldsymbol{x}$ at time t. We assume that π is a function parametrized by $\boldsymbol{\theta}$.[1]

To find a policy π^*, which minimizes (2), PILCO builds upon three components: 1) a probabilistic GP dynamics model (Section 3.1), 2) deterministic approximate inference for long-term predictions and policy evaluation (Section 3.2), 3) analytic computation of the policy gradients $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$ for policy improvement (Section 3.3). The GP model internally represents the dynamics in (1) and is subsequently employed for long-term predictions $p(\boldsymbol{x}_1|\pi), \ldots, p(\boldsymbol{x}_T|\pi)$, given a policy π. These predictions are obtained through approximate inference and used to evaluate the expected long-term cost $J^\pi(\boldsymbol{\theta})$ in (2). The policy π is improved based on gradient information $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$. Algorithm 1 summarizes the PILCO learning framework.

Algorithm 1 PILCO

1: **init:** Sample controller parameters $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Apply random control signals and record data.
2: **repeat**
3: Learn probabilistic (GP) dynamics model, see Sec. 3.1, using all data
4: **repeat**
5: Approximate inference for policy evaluation, see Sec. 3.2: get $J^\pi(\boldsymbol{\theta})$, Eq. (9)–(11)
6: Gradient-based policy improvement, see Sec. 3.3: get $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$, Eq. (12)–(16)
7: Update parameters $\boldsymbol{\theta}$ (e.g., CG or L-BFGS).
8: **until** convergence; **return** $\boldsymbol{\theta}^*$
9: Set $\pi^* \leftarrow \pi(\boldsymbol{\theta}^*)$
10: Apply π^* to system and record data
11: **until** task learned

3.1 Model Learning

PILCO's probabilistic dynamics model is implemented as a GP, where we use tuples $(\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathbb{R}^{D+F}$ as training inputs and differences $\boldsymbol{\Delta}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t \in \mathbb{R}^D$ as training targets.[2] A GP is completely specified by a mean function $m(\cdot)$ and a positive semidefinite covariance function/kernel $k(\cdot, \cdot)$. In this paper, we consider a prior mean function $m \equiv 0$ and the covariance function

$$k(\tilde{\boldsymbol{x}}_p, \tilde{\boldsymbol{x}}_q) = \sigma_f^2 \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q)^\top \boldsymbol{\Lambda}^{-1}(\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q)\right) + \delta_{pq}\sigma_w^2 \quad (3)$$

with $\tilde{\boldsymbol{x}} := [\boldsymbol{x}^\top \boldsymbol{u}^\top]^\top$. We defined $\boldsymbol{\Lambda} := \mathrm{diag}([\ell_1^2, \ldots, \ell_{D+F}^2])$ in (3), which depends on the characteristic length-scales ℓ_i, and σ_f^2 is the variance of the latent transition function f. Given n training inputs $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n]$ and corresponding training targets $\boldsymbol{y} = [\Delta_1, \ldots, \Delta_n]^\top$, the posterior GP hyper-parameters (length-scales ℓ_i, signal variance σ_f^2, and noise variance σ_w^2) are learned by evidence maximization [34], [47].

1. In our experiments in Section 7, we use a) nonlinear parametrizations by means of RBF networks, where the parameters $\boldsymbol{\theta}$ are the weights and the features, or b) linear-affine parametrizations, where the parameters $\boldsymbol{\theta}$ are the weight matrix and a bias term.

2. Using differences as training targets encodes an implicit prior mean function $m(\boldsymbol{x}) = \boldsymbol{x}$. This means that when leaving the training data, the GP predictions do not fall back to 0 but they remain constant.

The posterior GP is a one-step prediction model, and the predicted successor state $\boldsymbol{x}_{t+1}$ is Gaussian distributed

$$p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1} \mid \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}), \quad (4)$$

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{x}_t + \mathbb{E}_f[\boldsymbol{\Delta}_t], \quad \boldsymbol{\Sigma}_{t+1} = \mathrm{var}_f[\boldsymbol{\Delta}_t], \quad (5)$$

where the mean and variance of the GP prediction are

$$\mathbb{E}_f[\boldsymbol{\Delta}_t] = m_f(\tilde{\boldsymbol{x}}_t) = \boldsymbol{k}_*^\top \left(\boldsymbol{K} + \sigma_w^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} = \boldsymbol{k}_*^\top \boldsymbol{\beta}, \quad (6)$$

$$\mathrm{var}_f[\boldsymbol{\Delta}_t] = k_{**} - \boldsymbol{k}_*^\top \left(\boldsymbol{K} + \sigma_w^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_*, \quad (7)$$

respectively, with $\boldsymbol{k}_* := k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{x}}_t)$, $k_{**} := k(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t)$, and $\boldsymbol{\beta} := (\boldsymbol{K} + \sigma_w^2 \boldsymbol{I})^{-1} \boldsymbol{y}$, where $\boldsymbol{K}$ is the kernel matrix with entries $K_{ij} = k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$.

For multivariate targets, we train conditionally independent GPs for each target dimension, i.e., the GPs are independent for given test inputs. For uncertain inputs, the target dimensions covary [44], see also Section 4.

3.2 Policy Evaluation

To evaluate and minimize J^π in (2) PILCO uses long-term predictions of the state evolution. In particular, we determine the marginal t-step-ahead predictive distributions $p(\boldsymbol{x}_1 \mid \pi), \ldots, p(\boldsymbol{x}_T \mid \pi)$ from the initial state distribution $p(\boldsymbol{x}_0)$, $t = 1, \ldots, T$. To obtain these long-term predictions, we cascade one-step predictions, see (4)-(5), which requires mapping uncertain test inputs through the GP dynamics model. In the following, we assume that these test inputs are Gaussian distributed. For notational convenience, we omit the explicit conditioning on the policy π in the following and assume that episodes start from $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0 \mid \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$.

For predicting $\boldsymbol{x}_{t+1}$ from $p(\boldsymbol{x}_t)$, we require a joint distribution $p(\tilde{\boldsymbol{x}}_t) = p(\boldsymbol{x}_t, \boldsymbol{u}_t)$, see (1). The control $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \boldsymbol{\theta})$ is a function of the state, and we approximate the desired joint distribution $p(\tilde{\boldsymbol{x}}_t) = p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ by a Gaussian. Details are provided in Section 5.5.

From now on, we assume a joint Gaussian distribution distribution $p(\tilde{\boldsymbol{x}}_t) = \mathcal{N}(\tilde{\boldsymbol{x}}_t | \tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ at time t. To compute

$$p(\boldsymbol{\Delta}_t) = \iint p(f(\tilde{\boldsymbol{x}}_t) \mid \tilde{\boldsymbol{x}}_t) p(\tilde{\boldsymbol{x}}_t) \mathrm{d}f \mathrm{d}\tilde{\boldsymbol{x}}_t, \quad (8)$$

we integrate out both the random variable $\tilde{\boldsymbol{x}}_t$ and the random function f, the latter one according to the posterior GP distribution. Computing the exact predictive distribution in (8) is analytically intractable as illustrated in Fig. 2. Hence, we approximate $p(\boldsymbol{\Delta}_t)$ by a Gaussian.

Assume the mean $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ and the covariance $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ of the predictive distribution $p(\boldsymbol{\Delta}_t)$ are known.[3] Then, a Gaussian approximation to the desired predictive distribution $p(\boldsymbol{x}_{t+1})$ is given as $\mathcal{N}(\boldsymbol{x}_{t+1} | \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$ with

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_{\boldsymbol{\Delta}}, \quad (9)$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{\boldsymbol{\Delta}} + \mathrm{cov}[\boldsymbol{x}_t, \boldsymbol{\Delta}_t] + \mathrm{cov}[\boldsymbol{\Delta}_t, \boldsymbol{x}_t]. \quad (10)$$

3. We will detail their computations in Sections 4.1 and 4.2.

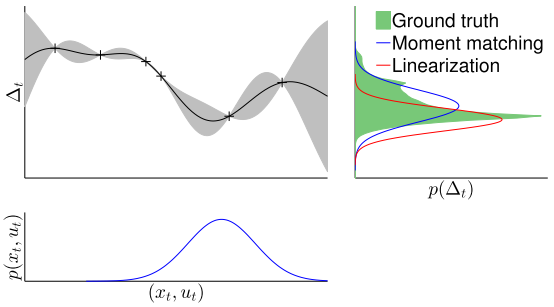


Fig. 2. GP prediction at an uncertain input. The input distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is assumed Gaussian (lower left panel). When propagating it through the GP model (upper left panel), we obtain the shaded distribution $p(\boldsymbol{\Delta}_t)$, upper right panel. We approximate $p(\boldsymbol{\Delta}_t)$ by a Gaussian (upper right panel), which is computed by means of either moment matching (blue) or linearization of the posterior GP mean (red). Using linearization for approximate inference can lead to predictive distributions that are too tight.

Note that both $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ are functions of the mean $\boldsymbol{\mu}_u$ and the covariance $\boldsymbol{\Sigma}_u$ of the control signal.

To evaluate the expected long-term cost J^π in (2), it remains to compute the expected values

$$\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] = \int c(\boldsymbol{x}_t)\mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\, \mathrm{d}\boldsymbol{x}_t, \tag{11}$$

$t = 1, \ldots, T$, of the cost c with respect to the predictive state distributions. We choose the cost c such that the integral in (11) and, thus, J^π in (2) can computed analytically. Examples of such cost functions include polynomials and mixtures of Gaussians.

3.3 Analytic Gradients for Policy Improvement

To find policy parameters $\boldsymbol{\theta}$, which minimize $J^\pi(\boldsymbol{\theta})$ in (2), we use gradient information $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$. We require that the expected cost in (11) is differentiable with respect to the moments of the state distribution. Moreover, we assume that the moments of the control distribution $\boldsymbol{\mu}_u$ and $\boldsymbol{\Sigma}_u$ can be computed analytically and are differentiable with respect to the policy parameters $\boldsymbol{\theta}$.

In the following, we describe how to analytically compute these gradients for a gradient-based policy search. We obtain the gradient $\mathrm{d}J^\pi/\mathrm{d}\boldsymbol{\theta}$ by repeated application of the chain-rule: First, we move the gradient into the sum in (2), and with $\mathcal{E}_t := \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$ we obtain

$$\begin{aligned}\frac{\mathrm{d}J^\pi(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} &= \sum_{t=1}^{T} \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}},\\ \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(\boldsymbol{x}_t)}\frac{\mathrm{d}p(\boldsymbol{x}_t)}{\mathrm{d}\boldsymbol{\theta}} := \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\mu}_t}\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\Sigma}_t}\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}},\end{aligned} \tag{12}$$

where we used the shorthand notation $\mathrm{d}\mathcal{E}_t/\mathrm{d}p(\boldsymbol{x}_t) = \{\mathrm{d}\mathcal{E}_t/\mathrm{d}\boldsymbol{\mu}_t, \mathrm{d}\mathcal{E}_t/\mathrm{d}\boldsymbol{\Sigma}_t\}$ for taking the derivative of $\mathcal{E}_t$ with respect to both the mean and covariance of $p(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Second, as we will show in Section 4, the predicted mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$ depend on the moments of $p(\boldsymbol{x}_{t-1})$ and the controller parameters $\boldsymbol{\theta}$. By applying the chain-rule to (12), we obtain then

$$\frac{\mathrm{d}p(\boldsymbol{x}_t)}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial p(\boldsymbol{x}_t)}{\partial p(\boldsymbol{x}_{t-1})}\frac{\mathrm{d}p(\boldsymbol{x}_{t-1})}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial p(\boldsymbol{x}_t)}{\partial \boldsymbol{\theta}}, \tag{13}$$

$$\frac{\partial p(\boldsymbol{x}_t)}{\partial p(\boldsymbol{x}_{t-1})} = \left\{\frac{\partial \boldsymbol{\mu}_t}{\partial p(\boldsymbol{x}_{t-1})}, \frac{\partial \boldsymbol{\Sigma}_t}{\partial p(\boldsymbol{x}_{t-1})}\right\}. \tag{14}$$

From here onward, we focus on $\mathrm{d}\boldsymbol{\mu}_t/\mathrm{d}\boldsymbol{\theta}$, see (12), but computing $\mathrm{d}\boldsymbol{\Sigma}_t/\mathrm{d}\boldsymbol{\theta}$ in (12) is similar. For $\mathrm{d}\boldsymbol{\mu}_t/\mathrm{d}\boldsymbol{\theta}$, we compute the derivative

$$\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\theta}}. \tag{15}$$

Since $\mathrm{d}p(\boldsymbol{x}_{t-1})/\mathrm{d}\boldsymbol{\theta}$ in (13) is known from time step $t-1$ and $\partial\boldsymbol{\mu}_t/\partial p(\boldsymbol{x}_{t-1})$ is computed by applying the chain-rule to (17)-(20), we conclude with

$$\frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial p(\boldsymbol{u}_{t-1})}\frac{\partial p(\boldsymbol{u}_{t-1})}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial \boldsymbol{\mu}_u}\frac{\partial \boldsymbol{\mu}_u}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial \boldsymbol{\Sigma}_u}\frac{\partial \boldsymbol{\Sigma}_u}{\partial \boldsymbol{\theta}}. \tag{16}$$

The partial derivatives of $\boldsymbol{\mu}_u$ and $\boldsymbol{\Sigma}_u$, i.e., the mean and covariance of $p(\boldsymbol{u}_t)$, used in (16) depend on the policy representation. The individual partial derivatives in (12)-(16) depend on the approximate inference method used for propagating state distributions through time. For example, with moment matching (MM) or linearization of the posterior GP (see Section 4 for details) the desired gradients can be computed analytically by repeated application of the chain-rule. The Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2013.218, derives the gradients for the moment-matching approximation.

A gradient-based optimization method using *estimates* of the gradient of $J^\pi(\boldsymbol{\theta})$ such as finite differences or more efficient sampling-based methods (see [43] for an overview) requires many function evaluations, which can be computationally expensive. However, since in our case policy evaluation can be performed analytically, we profit from analytic expressions for the gradients, which allows for standard gradient-based non-convex optimization methods, such as CG or BFGS, to determine optimized policy parameters $\boldsymbol{\theta}^*$.

4 Long-Term Predictions

Long-term predictions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$ for a given policy parametrization are essential for policy evaluation and improvement as described in Sections 3.2 and 3.3, respectively. These long-term predictions are computed iteratively: At each time step, PILCO approximates the predictive state distribution $p(\boldsymbol{x}_{t+1})$ by a Gaussian, see (9)-(10). For this approximation, we need to predict with GPs when the input is given by a probability distribution $p(\tilde{\boldsymbol{x}}_t)$, see (8). In this section, we detail the computations of the mean $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ and covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ of the GP predictive distribution, see (8), as well as the cross-covariances $\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t] = \mathrm{cov}\big[[\boldsymbol{x}_t^\top, \boldsymbol{u}_t^\top]^\top, \boldsymbol{\Delta}_t\big]$, which are required in (9)-(10). We present two approximations to predicting with GPs at uncertain inputs: Moment matching [15], [44] and linearization of the posterior GP mean function [28]. While moment matching computes the

first two moments of the predictive distribution exactly, their approximation by explicit linearization of the posterior GP is computationally advantageous.

4.1 Moment Matching

Following the law of iterated expectations, for target dimensions $a = 1, \ldots, D$, we obtain the *predictive mean*

$$\begin{aligned}\mu_{\boldsymbol{\Delta}}^a &= \mathbb{E}_{\tilde{\boldsymbol{x}}_t}[\mathbb{E}_{f_a}[f_a(\tilde{\boldsymbol{x}}_t)|\tilde{\boldsymbol{x}}_t]] = \mathbb{E}_{\tilde{\boldsymbol{x}}_t}[m_{f_a}(\tilde{\boldsymbol{x}}_t)] \\ &= \int m_{f_a}(\tilde{\boldsymbol{x}}_t)\mathcal{N}(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)\, \mathrm{d}\tilde{\boldsymbol{x}}_t = \boldsymbol{\beta}_a^\top \boldsymbol{q}_a,\end{aligned} \tag{17}$$

$$\boldsymbol{\beta}_a = \left(\boldsymbol{K}_a + \sigma_{w_a}^2\right)^{-1}\boldsymbol{y}_a, \tag{18}$$

with $\boldsymbol{q}_a = [q_{a_1}, \ldots, q_{a_n}]^\top$. The entries of $\boldsymbol{q}_a \in \mathbb{R}^n$ are computed using standard results from multiplying and integrating over Gaussians and are given by

$$\begin{aligned}q_{a_i} &= \int k_a(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}})\mathcal{N}(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)\, \mathrm{d}\tilde{\boldsymbol{x}}_t \\ &= \sigma_{f_a}^2 \left|\tilde{\boldsymbol{\Sigma}}_t\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{I}\right|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\boldsymbol{\nu}_i^\top(\tilde{\boldsymbol{\Sigma}}_t + \boldsymbol{\Lambda}_a)^{-1}\boldsymbol{\nu}_i\right),\end{aligned} \tag{19}$$

where we define

$$\boldsymbol{\nu}_i := (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t) \tag{20}$$

as the difference between the training input $\tilde{\boldsymbol{x}}_i$ and the mean of the test input distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

Computing the *predictive covariance matrix* $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}} \in \mathbb{R}^{D\times D}$ requires us to distinguish between diagonal elements σ_{aa}^2 and off-diagonal elements σ_{ab}^2, $a \neq b$: Using the law of total (co-)variance, we obtain for target dimensions $a, b = 1, \ldots, D$

$$\sigma_{aa}^2 = \mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathrm{var}_f[\Delta_a|\tilde{\boldsymbol{x}}_t]\right] + \mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a^2\right] - \left(\boldsymbol{\mu}_{\boldsymbol{\Delta}}^a\right)^2, \tag{21}$$

$$\sigma_{ab}^2 = \mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}[\Delta_a\Delta_b] - \boldsymbol{\mu}_{\boldsymbol{\Delta}}^a\boldsymbol{\mu}_{\boldsymbol{\Delta}}^b, \quad a \neq b, \tag{22}$$

respectively, where $\mu_{\boldsymbol{\Delta}}^a$ is known from (17). The off-diagonal terms σ_{ab}^2 do not contain the additional term $\mathbb{E}_{\tilde{\boldsymbol{x}}_t}[\mathrm{cov}_f[\Delta_a, \Delta_b|\tilde{\boldsymbol{x}}_t]]$ because of the conditional independence assumption of the GP models: Different target dimensions do not covary for given $\tilde{\boldsymbol{x}}_t$.

We start the computation of the covariance matrix with the terms that are common to both the diagonal and the off-diagonal entries: With $p(\tilde{\boldsymbol{x}}_t) = \mathcal{N}(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ and the law of iterated expectations, we obtain

$$\begin{aligned}\mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}[\Delta_a\Delta_b] &= \mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathbb{E}_f[\Delta_a|\tilde{\boldsymbol{x}}_t]\mathbb{E}_f[\Delta_b|\tilde{\boldsymbol{x}}_t]\right] \\ &\overset{(6)}{=} \int m_f^a(\tilde{\boldsymbol{x}}_t)m_f^b(\tilde{\boldsymbol{x}}_t)p(\tilde{\boldsymbol{x}}_t)\mathrm{d}\tilde{\boldsymbol{x}}_t\end{aligned} \tag{23}$$

because of the conditional independence of Δ_a and Δ_b given $\tilde{\boldsymbol{x}}_t$. Using the definition of the GP mean function in (6), we obtain

$$\mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}[\Delta_a\Delta_b] = \boldsymbol{\beta}_a^\top\boldsymbol{Q}\boldsymbol{\beta}_b, \tag{24}$$

$$\boldsymbol{Q} := \int k_a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{X}})^\top k_b(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{X}})p(\tilde{\boldsymbol{x}}_t)\mathrm{d}\tilde{\boldsymbol{x}}_t. \tag{25}$$

Using standard results from Gaussian multiplications and integration, we obtain the entries Q_{ij} of $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$

$$Q_{ij} = |\boldsymbol{R}|^{-\frac{1}{2}} k_a(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{\mu}}_t)k_b(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{\mu}}_t)\exp\left(\frac{1}{2}\boldsymbol{z}_{ij}^\top\boldsymbol{T}^{-1}\boldsymbol{z}_{ij}\right), \tag{26}$$

where we define

$$\begin{aligned}&\boldsymbol{R} := \tilde{\boldsymbol{\Sigma}}_t\left(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}\right) + \boldsymbol{I}, \quad \boldsymbol{T} := \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \tilde{\boldsymbol{\Sigma}}_t^{-1}, \\ &\boldsymbol{z}_{ij} := \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\nu}_i + \boldsymbol{\Lambda}_b^{-1}\boldsymbol{\nu}_j,\end{aligned}$$

with $\boldsymbol{\nu}_i$ defined in (20). Hence, the off-diagonal entries of $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ are fully determined by (17)-(20), (22), and (24)-(26).

From (21), we see that the diagonal entries contain the additional term

$$\mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathrm{var}_f[\Delta_a|\tilde{\boldsymbol{x}}_t]\right] = \sigma_{f_a}^2 - \mathrm{tr}\left(\left(\boldsymbol{K}_a + \sigma_{w_a}^2\boldsymbol{I}\right)^{-1}\boldsymbol{Q}\right) + \sigma_{w_a}^2 \tag{27}$$

with $\boldsymbol{Q}$ given in (26) and $\sigma_{w_a}^2$ being the system noise variance of the ath target dimension. This term is the expected variance of the function, see (7), under the distribution $p(\tilde{\boldsymbol{x}}_t)$.

To obtain the *cross-covariances* $\mathrm{cov}[\boldsymbol{x}_t, \boldsymbol{\Delta}_t]$ in (10), we compute the cross-covariance $\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t]$ between an uncertain state-action pair $\tilde{\boldsymbol{x}}_t \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ and the corresponding predicted state difference $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t = \boldsymbol{\Delta}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\Delta}}, \boldsymbol{\Sigma}_{\boldsymbol{\Delta}})$. This cross-covariance is given by

$$\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t] = \mathbb{E}_{\tilde{\boldsymbol{x}}_t, f}\left[\tilde{\boldsymbol{x}}_t\boldsymbol{\Delta}_t^\top\right] - \tilde{\boldsymbol{\mu}}_t\boldsymbol{\mu}_{\boldsymbol{\Delta}}^\top, \tag{28}$$

where the components of $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ are given in (17), and $\tilde{\boldsymbol{\mu}}_t$ is the known mean of the input distribution of the state-action pair at time step t.

Using the law of iterated expectation, for each state dimension $a = 1, \ldots, D$, we compute $\mathbb{E}_{\tilde{\boldsymbol{x}}_t, f}[\tilde{\boldsymbol{x}}_t\Delta_t^a]$ as

$$\begin{aligned}\mathbb{E}_{\tilde{\boldsymbol{x}}_t, f}[\tilde{\boldsymbol{x}}_t\Delta_t^a] &= \mathbb{E}_{\tilde{\boldsymbol{x}}_t}[\tilde{\boldsymbol{x}}_t\mathbb{E}_f[\Delta_t^a|\tilde{\boldsymbol{x}}_t]] = \int \tilde{\boldsymbol{x}}_t m_f^a(\tilde{\boldsymbol{x}}_t)p(\tilde{\boldsymbol{x}}_t)\, \mathrm{d}\tilde{\boldsymbol{x}}_t \\ &\overset{(6)}{=} \int \tilde{\boldsymbol{x}}_t\left(\sum_{i=1}^n \beta_{a_i}k_f^a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i)\right)p(\tilde{\boldsymbol{x}}_t)\, \mathrm{d}\tilde{\boldsymbol{x}}_t,\end{aligned} \tag{29}$$

where the (posterior) GP mean function $m_f(\tilde{\boldsymbol{x}}_t)$ was represented as a finite kernel expansion. Note that $\tilde{\boldsymbol{x}}_i$ are the state-action pairs, which were used to train the dynamics GP model. By pulling the constant β_{a_i} out of the integral and changing the order of summation and integration, we obtain

$$\mathbb{E}_{\tilde{\boldsymbol{x}}_t, f}\left[\tilde{\boldsymbol{x}}_t\Delta_t^a\right] = \sum_{i=1}^n \beta_{a_i}\int \tilde{\boldsymbol{x}}_t \underbrace{c_1\mathcal{N}(\tilde{\boldsymbol{x}}_t \mid \tilde{\boldsymbol{x}}_i, \boldsymbol{\Lambda}_a)}_{=k_f^a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i)} \underbrace{\mathcal{N}(\tilde{\boldsymbol{x}}_t \mid \tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)}_{p(\tilde{\boldsymbol{x}}_t)} \mathrm{d}\tilde{\boldsymbol{x}}_t, \tag{30}$$

where we define $c_1 := \sigma_{f_a}^2(2\pi)^{\frac{D+F}{2}}|\boldsymbol{\Lambda}_a|^{\frac{1}{2}}$ with $\tilde{\boldsymbol{x}} \in \mathbb{R}^{D+F}$, such that $k_f^a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i) = c_1\mathcal{N}(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{x}}_i, \boldsymbol{\Lambda}_a)$ is an unnormalized Gaussian probability distribution in $\tilde{\boldsymbol{x}}_t$, where $\tilde{\boldsymbol{x}}_i$, $i = 1, \ldots, n$, are the GP training inputs. The product of the two Gaussians in (30) yields a new (unnormalized) Gaussian $c_2^{-1}\mathcal{N}(\tilde{\boldsymbol{x}}_t \mid \boldsymbol{\psi}_i, \boldsymbol{\Psi})$ with

$$\begin{aligned}c_2^{-1} &= (2\pi)^{-\frac{D+F}{2}}|\boldsymbol{\Lambda}_a + \tilde{\boldsymbol{\Sigma}}_t|^{-\frac{1}{2}} \\ &\quad \times \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t)^\top(\boldsymbol{\Lambda}_a + \tilde{\boldsymbol{\Sigma}}_t)^{-1}(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t)\right), \\ \boldsymbol{\Psi} &= \left(\boldsymbol{\Lambda}_a^{-1} + \tilde{\boldsymbol{\Sigma}}_t^{-1}\right)^{-1}, \quad \boldsymbol{\psi}_i = \boldsymbol{\Psi}\left(\boldsymbol{\Lambda}_a^{-1}\tilde{\boldsymbol{x}}_i + \tilde{\boldsymbol{\Sigma}}_t^{-1}\tilde{\boldsymbol{\mu}}_t\right).\end{aligned}$$

By pulling all remaining variables, which are independent of $\tilde{\boldsymbol{x}}_t$, out of the integral in (30), the integral determines the expected value of the product of the two Gaussians, $\boldsymbol{\psi}_i$. Hence, we obtain

$$\begin{aligned}\mathbb{E}_{\tilde{\boldsymbol{x}}_t,f}\left[\tilde{\boldsymbol{x}}_t\Delta_t^a\right] &= \sum_{i=1}^{n} c_1 c_2^{-1}\beta_{a_i}\boldsymbol{\psi}_i, a=1,\ldots,D,\\ \operatorname{cov}_{\tilde{\boldsymbol{x}}_t,f}\left[\tilde{\boldsymbol{x}}_t,\Delta_t^a\right] &= \sum_{i=1}^{n} c_1 c_2^{-1}\beta_{a_i}\boldsymbol{\psi}_i - \tilde{\boldsymbol{\mu}}_t\mu_{\boldsymbol{\Delta}}^a,\end{aligned} \tag{31}$$

for all predictive dimensions $a=1,\ldots,E$. With $c_1c_2^{-1}=q_{a_i}$, see (19), and $\boldsymbol{\psi}_i=\tilde{\boldsymbol{\Sigma}}_t(\tilde{\boldsymbol{\Sigma}}_t+\boldsymbol{\Lambda}_a)^{-1}\tilde{\boldsymbol{x}}_i+\boldsymbol{\Lambda}(\tilde{\boldsymbol{\Sigma}}_t+\boldsymbol{\Lambda}_a)^{-1}\tilde{\boldsymbol{\mu}}_t$ we simplify (31) and obtain

$$\operatorname{cov}_{\tilde{\boldsymbol{x}}_t,f}\left[\tilde{\boldsymbol{x}}_t,\Delta_t^a\right] = \sum_{i=1}^{n}\beta_{a_i}q_{a_i}\tilde{\boldsymbol{\Sigma}}_t(\tilde{\boldsymbol{\Sigma}}_t+\boldsymbol{\Lambda}_a)^{-1}(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{\mu}}_t), \tag{32}$$

$a=1,\ldots,E$. The desired covariance $\operatorname{cov}[\boldsymbol{x}_t,\Delta_t]$ is a $D\times E$ submatrix of the $(D+F)\times E$ cross-covariance computed into (32).

A visualization of the approximation of the predictive distribution by means of exact moment matching is given in Fig. 2.

4.2 Linearization of the Posterior GP Mean Function

An alternative way of approximating the predictive distribution $p(\boldsymbol{\Delta}_t)$ by a Gaussian for $\tilde{\boldsymbol{x}}_t\sim\mathcal{N}(\tilde{\boldsymbol{x}}_t\mid\tilde{\boldsymbol{\mu}}_t,\tilde{\boldsymbol{\Sigma}}_t)$ is to linearize the posterior GP mean function. Fig. 2 visualizes the approximation by means of linearizing the posterior GP mean function.

The *predicted mean* is obtained by evaluating the posterior GP mean in (5) at the mean $\tilde{\boldsymbol{\mu}}_t$ of the input distribution, i.e.,

$$\boldsymbol{\mu}_{\boldsymbol{\Delta}}^a = \mathbb{E}_f[f_a(\tilde{\boldsymbol{\mu}}_t)] = m_{f_a}(\tilde{\boldsymbol{\mu}}_t) = \boldsymbol{\beta}_a^\top k_a(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{\mu}}_t), \tag{33}$$

$a=1,\ldots,E$, where $\boldsymbol{\beta}_a$ is given in (18).

To compute the GP *predictive covariance matrix* $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$, we explicitly linearize the posterior GP mean function around $\tilde{\boldsymbol{\mu}}_t$. By applying standard results for mapping Gaussian distributions through linear models, the predictive covariance is given by

$$\boldsymbol{\Sigma}_{\boldsymbol{\Delta}} = \boldsymbol{V}\tilde{\boldsymbol{\Sigma}}_t\boldsymbol{V}^\top+\boldsymbol{\Sigma}_w, \tag{34}$$

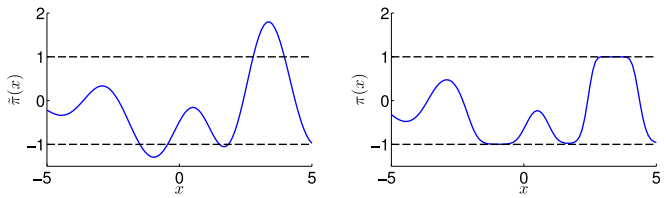
$$\boldsymbol{V} = \frac{\partial\boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial\tilde{\boldsymbol{\mu}}_t} = \boldsymbol{\beta}^\top\frac{\partial k(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{\mu}}_t)}{\partial\tilde{\boldsymbol{\mu}}_t}. \tag{35}$$

In (34), $\boldsymbol{\Sigma}_w$ is a diagonal matrix whose entries are the noise variances $\sigma_{w_a}^2$ plus the model uncertainties $\operatorname{var}_f[\Delta_t^a|\tilde{\boldsymbol{\mu}}_t]$ evaluated at $\tilde{\boldsymbol{\mu}}_t$, see (7). This means, model uncertainty no longer depends on the density of the data points. Instead it is assumed to be constant. Note that the moments computed in (33)-(34) are not exact.

The *cross-covariance* $\operatorname{cov}[\tilde{\boldsymbol{x}}_t,\boldsymbol{\Delta}_t]$ is given by $\tilde{\boldsymbol{\Sigma}}_t\boldsymbol{V}$, where $\boldsymbol{V}$ is defined in (35).

5 Policy

In the following, we describe the desired properties of the policy within the PILCO learning framework. First, to compute the long-term predictions $p(\boldsymbol{x}_1),\ldots,p(\boldsymbol{x}_T)$ for policy evaluation, the policy must allow us to compute a distribution over controls $p(\boldsymbol{u})=p(\pi(\boldsymbol{x}))$ for a given (Gaussian) state distribution $p(\boldsymbol{x})$. Second, in a realistic real-world application, the amplitudes of the control signals are bounded. Ideally, the learning system takes these constraints explicitly into account. In the following, we detail how PILCO implements these desiderata.



(a) Preliminary policy $\tilde{\pi}$ as a function of the state.

(b) Policy $\pi=\sigma(\tilde{\pi}(x))$ as a function of the state.

Fig. 3. Constraining the control signal. Panel (a) shows an example of an unconstrained preliminary policy $\tilde{\pi}$ as a function of the state x. Panel (b) shows the constrained policy $\pi(x)=\sigma(\tilde{\pi}(x))$ as a function of the state x.

5.1 Predictive Distribution over Controls

During the long-term predictions, the states are given by a probability distribution $p(\boldsymbol{x}_t)$, $t=0,\ldots,T$. The probability distribution of the state $\boldsymbol{x}_t$ induces a predictive distribution $p(\boldsymbol{u}_t)=p(\pi(\boldsymbol{x}_t))$ over controls, even when the policy is deterministic. We approximate the distribution over controls using moment matching, which is in many interesting cases analytically tractable.

5.2 Constrained Control Signals

In practical applications, force or torque limits are present and must be accounted for during planning. Suppose the control limits are such that $\boldsymbol{u}\in[-\boldsymbol{u}_{\max},\boldsymbol{u}_{\max}]$. Let us consider a *preliminary policy* $\tilde{\pi}$ with an unconstrained amplitude. To account for the control limits coherently during simulation, we squash the preliminary policy $\tilde{\pi}$ through a bounded and differentiable squashing function, which limits the amplitude of the final policy π. As a squashing function, we use

$$\sigma(x) = \frac{9}{8}\sin(x)+\frac{1}{8}\sin(3x) \quad\in[-1,1], \tag{36}$$

which is the third-order Fourier series expansion of a trapezoidal wave, normalized to the interval $[-1,1]$. The squashing function in (36) is computationally convenient as we can analytically compute predictive moments for Gaussian distributed states. Subsequently, we multiply the squashed policy by $\boldsymbol{u}_{\max}$ and obtain the final policy

$$\pi(\boldsymbol{x}) = \boldsymbol{u}_{\max}\sigma(\tilde{\pi}(\boldsymbol{x}))\in[-\boldsymbol{u}_{\max},\boldsymbol{u}_{\max}], \tag{37}$$

an illustration of which is shown in Fig. 3. Although the squashing function in (36) is periodic, it is almost always used within a half wave if the preliminary policy $\tilde{\pi}$ is initialized to produce function values that do not exceed the domain of a single period. Therefore, the periodicity does not matter in practice.

To compute a distribution over constrained control signals, we execute the following steps:

$$p(\boldsymbol{x}_t)\mapsto p(\tilde{\pi}(\boldsymbol{x}_t))\mapsto p(\boldsymbol{u}_{\max}\sigma(\tilde{\pi}(\boldsymbol{x}_t)))=p(\boldsymbol{u}_t). \tag{38}$$

First, we map the Gaussian state distribution $p(\boldsymbol{x}_t)$ through the preliminary (unconstrained) policy $\tilde{\pi}$. Thus, we require a preliminary policy $\tilde{\pi}$ that allows for closed-form computation of the moments of the distribution over controls $p(\tilde{\pi}(\boldsymbol{x}_t))$. Second, we squash the approximate Gaussian distribution $p(\tilde{\pi}(\boldsymbol{x}))$ according to (37) and compute exactly the mean and variance of $p(\tilde{\pi}(\boldsymbol{x}))$. Details are given in the Appendix, available in the online supplemental material. We approximate $p(\tilde{\pi}(\boldsymbol{x}))$ by a Gaussian with these moments, yielding the distribution $p(\boldsymbol{u})$ over controls in (38).

5.3 Representations of the Preliminary Policy

In the following, we present two representations of the preliminary policy $\tilde{\pi}$, which allow for closed-form computations of the mean and covariance of $p(\tilde{\pi}(\boldsymbol{x}))$ when the state $\boldsymbol{x}$ is Gaussian distributed. We consider both a linear and a nonlinear representations of $\tilde{\pi}$.

5.3.1 Linear Policy

The linear preliminary policy is given by

$$\tilde{\pi}(\boldsymbol{x}_*) = \boldsymbol{A}\boldsymbol{x}_* + \boldsymbol{b}, \tag{39}$$

where $\boldsymbol{A}$ is a parameter matrix of weights and $\boldsymbol{b}$ is an offset vector. In each control dimension d, the policy in (39) is a linear combination of the states (the weights are given by the dth row in $\boldsymbol{A}$) plus an offset b_d.

The predictive distribution $p(\tilde{\pi}(\boldsymbol{x}_*))$ for a state distribution $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ is an exact Gaussian with mean and covariance

$$\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}(\boldsymbol{x}_*)] = \boldsymbol{A}\boldsymbol{\mu}_* + \boldsymbol{b}, \quad \operatorname{cov}_{\boldsymbol{x}_*}[\tilde{\pi}(\boldsymbol{x}_*)] = \boldsymbol{A}\boldsymbol{\Sigma}_*\boldsymbol{A}^\top, \tag{40}$$

respectively. A drawback of the linear policy is that it is not flexible. However, a linear controller can often be used for stabilization around an equilibrium.

5.3.2 Nonlinear Policy: Deterministic Gaussian Process

In the nonlinear case, we represent the preliminary policy $\tilde{\pi}$ by

$$\tilde{\pi}(\boldsymbol{x}_*) = \sum_{i=1}^{N} k(\boldsymbol{m}_i, \boldsymbol{x}_*)\big(\boldsymbol{K} + \sigma_\pi^2 \boldsymbol{I}\big)^{-1}\boldsymbol{t} = k(\boldsymbol{M}, \boldsymbol{x}_*)^\top \boldsymbol{\alpha}, \tag{41}$$

where $\boldsymbol{x}_*$ is a test input, $\boldsymbol{\alpha} = (\boldsymbol{K} + 0.01\boldsymbol{I})^{-1}\boldsymbol{t}$, where $\boldsymbol{t}$ plays the role of a GP's training targets. In (41), $\boldsymbol{M} = [\boldsymbol{m}_1, \ldots, \boldsymbol{m}_N]$ are the centers of the (axis-aligned) Gaussian basis functions

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \exp\left(-\frac{1}{2}(\boldsymbol{x}_p - \boldsymbol{x}_q)^\top \boldsymbol{\Lambda}^{-1}(\boldsymbol{x}_p - \boldsymbol{x}_q)\right). \tag{42}$$

We call the policy representation in (41) a *deterministic GP* with a fixed number of N basis functions. Here, "deterministic" means that there is no uncertainty about the underlying function, that is, $\operatorname{var}_{\tilde{\pi}}[\tilde{\pi}(\boldsymbol{x})] = 0$. Therefore, the deterministic GP is a degenerate model, which is functionally equivalent to a regularized RBF network. The deterministic GP is functionally equivalent to the posterior GP mean function in (6), where we set the signal variance to 1, see (42), and the noise variance to 0.01. As the preliminary policy will be squashed through σ in (36) whose relevant support is the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, a signal variance of 1 is about right. Setting additionally the noise standard deviation to 0.1 corresponds to fixing the signal-to-noise ratio of the policy to 10 and, hence, the regularization.

For a Gaussian distributed state $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$, the *predictive mean* of $\tilde{\pi}(\boldsymbol{x}_*)$ as defined in (41) is given as

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}(\boldsymbol{x}_*)] &= \boldsymbol{\alpha}_a^\top \mathbb{E}_{\boldsymbol{x}_*}[k(\boldsymbol{M}, \boldsymbol{x}_*)] \\ &= \boldsymbol{\alpha}_a^\top \int k(\boldsymbol{M}, \boldsymbol{x}_*) p(\boldsymbol{x}_*)\, \mathrm{d}\boldsymbol{x}_* = \boldsymbol{\alpha}_a^\top \boldsymbol{r}_a,\end{aligned} \tag{43}$$

where for $i = 1, \ldots, N$ and all policy dimensions $a = 1, \ldots, F$

$$\begin{aligned} r_{a_i} = {} & |\boldsymbol{\Sigma}_* \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{I}|^{-\frac{1}{2}} \\ & \times \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_* - \boldsymbol{m}_i)^\top (\boldsymbol{\Sigma}_* + \boldsymbol{\Lambda}_a)^{-1}(\boldsymbol{\mu}_* - \boldsymbol{m}_i)\right). \end{aligned}$$

The diagonal matrix $\boldsymbol{\Lambda}_a$ contains the squared length-scales ℓ_i, $i = 1, \ldots, D$. The predicted mean in (43) is equivalent to the standard predicted GP mean in (17).

For $a, b = 1, \ldots, F$, the entries of the *predictive covariance matrix* are computed according to

$$\begin{aligned} & \operatorname{cov}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*), \tilde{\pi}_b(\boldsymbol{x}_*)] \\ & \quad = \mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\tilde{\pi}_b(\boldsymbol{x}_*)] - \mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)]\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_b(\boldsymbol{x}_*)], \end{aligned}$$

where $\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_{\{a,b\}}(\boldsymbol{x}_*)]$ is given in (43). Hence, we focus on the term $\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\tilde{\pi}_b(\boldsymbol{x}_*)]$, which for $a, b = 1, \ldots, F$ is given by

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\tilde{\pi}_b(\boldsymbol{x}_*)] &= \boldsymbol{\alpha}_a^\top \mathbb{E}_{\boldsymbol{x}_*}[k_a(\boldsymbol{M}, \boldsymbol{x}_*) k_b(\boldsymbol{M}, \boldsymbol{x}_*)^\top]\boldsymbol{\alpha}_b \\ &= \boldsymbol{\alpha}_a^\top \boldsymbol{Q} \boldsymbol{\alpha}_b.\end{aligned}$$

For $i, j = 1, \ldots, N$, we compute the entries of $\boldsymbol{Q}$ as

$$\begin{aligned} Q_{ij} &= \int k_a(\boldsymbol{m}_i, \boldsymbol{x}_*) k_b(\boldsymbol{m}_j, \boldsymbol{x}_*) p(\boldsymbol{x}_*) \mathrm{d}\boldsymbol{x}_* \\ &= k_a(\boldsymbol{m}_i, \boldsymbol{x}_*) k_b(\boldsymbol{m}_j, \boldsymbol{x}_*) |\boldsymbol{R}|^{-\frac{1}{2}} \exp\big(\boldsymbol{z}_{ij}^\top \boldsymbol{T}^{-1} \boldsymbol{z}_{ij}\big), \\ \boldsymbol{R} &= \boldsymbol{\Sigma}_* \big(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}\big) + \boldsymbol{I}, \quad \boldsymbol{T} = \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \boldsymbol{\Sigma}_*^{-1}, \\ \boldsymbol{z}_{ij} &= \boldsymbol{\Lambda}_a^{-1}(\boldsymbol{\mu}_* - \boldsymbol{m}_i) + \boldsymbol{\Lambda}_b^{-1}(\boldsymbol{\mu}_* - \boldsymbol{m}_j). \end{aligned}$$

Combining this result with (43) fully determines the predictive covariance matrix of the preliminary policy.

Unlike the predictive covariance of a probabilistic GP, see (21)-(22), the predictive covariance matrix of the deterministic GP does not comprise any model uncertainty in its diagonal entries.

5.4 Policy Parameters

In the following, we describe the policy parameters for both the linear and the nonlinear policy.[4]

5.4.1 Linear Policy

The linear policy in (39) possesses $D + 1$ parameters per control dimension: For control dimension d there are D weights in the dth row of the matrix $\boldsymbol{A}$. One additional parameter originates from the offset parameter b_d.

4. For notational convenience, with a (non)linear policy we mean the (non)linear preliminary policy $\tilde{\pi}$ mapped through the squashing function σ and subsequently multiplied by $\boldsymbol{u}_{\max}$.

5.4.2 Nonlinear Policy

The parameters of the deterministic GP in (41) are the locations $\boldsymbol{M}$ of the centers (DN parameters), the (shared) length-scales of the Gaussian basis functions (D length-scale parameters per target dimension), and the N targets $\boldsymbol{t}$ per target dimension. In the case of multivariate controls, the basis function centers $\boldsymbol{M}$ are shared.

5.5 Computing the Successor State Distribution

Algorithm 2 summarizes the computational steps required to compute the successor state distribution $p(\boldsymbol{x}_{t+1})$ from $p(\boldsymbol{x}_t)$.

Algorithm 2 Computing the Successor State Distribution

1: **init:** $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$
2: Control distribution $p(\boldsymbol{u}_t) = p(\boldsymbol{u}_{\max}\sigma(\tilde{\pi}(\boldsymbol{x}_t, \boldsymbol{\theta})))$
3: Joint state-control distribution $p(\tilde{\boldsymbol{x}}_t) = p(\boldsymbol{x}_t, \boldsymbol{u}_t)$
4: Predictive GP distribution of change in state $p(\boldsymbol{\Delta}_t)$
5: Distribution of successor state $p(\boldsymbol{x}_{t+1})$

The computation of a distribution over controls $p(\boldsymbol{u}_t)$ from the state distribution $p(\boldsymbol{x}_t)$ requires two steps: First, for a Gaussian state distribution $p(\boldsymbol{x}_t)$ at time t a Gaussian approximation of the distribution $p(\tilde{\pi}(\boldsymbol{x}_t))$ of the preliminary policy is computed analytically. Second, the preliminary policy is squashed through σ and an approximate Gaussian distribution of $p(\boldsymbol{u}_{\max}\sigma(\tilde{\pi}(\boldsymbol{x}_t)))$ is computed analytically in (38) using results from the Appendix, available in the online supplemental material. Third, we analytically compute a Gaussian approximation to the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = p(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))$. For this, we compute (a) a Gaussian approximation to the joint distribution $p(\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x}_t))$, which is exact if $\tilde{\pi}$ is linear, and (b) an approximate fully joint Gaussian distribution $p(\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x}_t), \boldsymbol{u}_t)$. We obtain cross-covariance information between the state $\boldsymbol{x}_t$ and the control signal $\boldsymbol{u}_t = \boldsymbol{u}_{\max}\sigma(\tilde{\pi}(\boldsymbol{x}_t))$ via

$$\operatorname{cov}[\boldsymbol{x}_t, \boldsymbol{u}_t] = \operatorname{cov}[\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x}_t)]\operatorname{cov}[\tilde{\pi}(\boldsymbol{x}_t), \tilde{\pi}(\boldsymbol{x}_t)]^{-1}\operatorname{cov}[\tilde{\pi}(\boldsymbol{x}_t), \boldsymbol{u}_t],$$

where we exploit the conditional independence of $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ given $\tilde{\pi}(\boldsymbol{x}_t)$. Then, we integrate $\tilde{\pi}(\boldsymbol{x}_t)$ out to obtain the desired joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$. This leads to an approximate Gaussian joint probability distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = p(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) = p(\tilde{\boldsymbol{x}}_t)$. Fourth, with the approximate Gaussian input distribution $p(\tilde{\boldsymbol{x}}_t)$, the distribution $p(\boldsymbol{\Delta}_t)$ of the change in state is computed using the results from Section 4. Finally, the mean and covariance of a Gaussian approximation of the successor state distribution $p(\boldsymbol{x}_{t+1})$ are given by (9) and (10), respectively.

All required computations can be performed analytically because of the choice of the Gaussian covariance function for the GP dynamics model, see (3), the representations of the preliminary policy $\tilde{\pi}$, see Section 5.3, and the choice of the squashing function, see (36).

6 Cost Function

In our learning set-up, we use a cost function that solely penalizes the euclidean distance d of the current state to the target state. Using only distance penalties is often sufficient to solve a task: Reaching a target $\boldsymbol{x}_{\text{target}}$ with high speed naturally leads to overshooting and, thus, to high long-term costs. In particular, we use the generalized binary saturating cost

$$c(\boldsymbol{x}) = 1 - \exp\left(-\frac{1}{2\sigma_c^2} d(\boldsymbol{x}, \boldsymbol{x}_{\text{target}})^2\right) \in [0, 1], \tag{44}$$

which is locally quadratic but saturates at unity for large deviations d from the desired target $\boldsymbol{x}_{\text{target}}$. In (44), the geometric distance from the state $\boldsymbol{x}$ to the target state is denoted by d, and the parameter σ_c controls the width of the cost function.[5]

In classical control, typically a quadratic cost is assumed. However, a quadratic cost tends to focus attention on the worst deviation from the target state along a predicted trajectory. In the early stages of learning the predictive uncertainty is large and, therefore, the policy gradients, which are described in Section 3.3 become less useful. Therefore, we use the saturating cost in (44) as a default within the PILCO learning framework.

The immediate cost in (44) is an unnormalized Gaussian with mean $\boldsymbol{x}_{\text{target}}$ and variance σ_c^2, subtracted from unity. Therefore, the expected immediate cost can be computed analytically according to

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] &= \int c(\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\\ &= 1 - \int \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_{\text{target}})^\top \boldsymbol{T}^{-1}(\boldsymbol{x} - \boldsymbol{x}_{\text{target}})\right)p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},\end{aligned} \tag{45}$$

where $\boldsymbol{T}^{-1}$ is the precision matrix of the unnormalized Gaussian in (45). If the state $\boldsymbol{x}$ has the same representation as the target vector, $\boldsymbol{T}^{-1}$ is a diagonal matrix with entries either unity or zero, scaled by $1/\sigma_c^2$. Hence, for $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we obtain the expected immediate cost

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] = 1 &- |\boldsymbol{I} + \boldsymbol{\Sigma}\boldsymbol{T}^{-1}|^{-1/2}\\ &\times \exp\left(-\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{x}_{\text{target}})^\top \tilde{\boldsymbol{S}}_1(\boldsymbol{\mu} - \boldsymbol{x}_{\text{target}})\right),\end{aligned} \tag{46}$$

$$\tilde{\boldsymbol{S}}_1 := \boldsymbol{T}^{-1}(\boldsymbol{I} + \boldsymbol{\Sigma}\boldsymbol{T}^{-1})^{-1}. \tag{47}$$

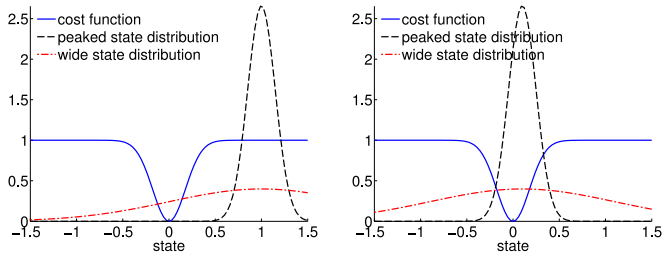
The partial derivatives $\frac{\partial}{\partial \boldsymbol{\mu}_t}\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)], \frac{\partial}{\partial \boldsymbol{\Sigma}_t}\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$ of the immediate cost with respect to the mean and the covariance of the state distribution $p(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, which are required to compute the policy gradients analytically, are given by

$$\frac{\partial \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]}{\partial \boldsymbol{\mu}_t} = -\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)](\boldsymbol{\mu}_t - \boldsymbol{x}_{\text{target}})^\top \tilde{\boldsymbol{S}}_1, \tag{48}$$

$$\begin{aligned}\frac{\partial \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]}{\partial \boldsymbol{\Sigma}_t} &= \frac{1}{2}\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]\\ &\times \left(\tilde{\boldsymbol{S}}_1(\boldsymbol{\mu}_t - \boldsymbol{x}_{\text{target}})(\boldsymbol{\mu}_t - \boldsymbol{x}_{\text{target}})^\top - \boldsymbol{I}\right)\tilde{\boldsymbol{S}}_1,\end{aligned} \tag{49}$$

respectively, where $\tilde{\boldsymbol{S}}_1$ is given in (47).

5. In the context of sensorimotor control, the saturating cost function in (44) resembles the cost function in human reasoning as experimentally validated by Körding and Wolpert [31].



(a) When the mean of the state is far away from the target, uncertain states (red, dashed-dotted) are preferred to more certain states with a more peaked distribution (black, dashed). This leads to initial exploration.

(b) When the mean of the state is close to the target, peaked state distributions (black, dashed) cause less expected cost and, thus, are preferable to more uncertain states (red, dashed-dotted), leading to exploitation close to the target.

Fig. 4. Automatic exploration and exploitation with the saturating cost function (blue, solid). The x-axes describe the state space. The target state is the origin.

6.1 Exploration and Exploitation

The saturating cost function in (44) allows for a natural exploration when the policy aims to minimize the expected long-term cost in (2). This property is illustrated in Fig. 4 for a single time step where we assume a Gaussian state distribution $p(\boldsymbol{x}_t)$. If the mean of $p(\boldsymbol{x}_t)$ is *far away from the target* $\boldsymbol{x}_{\text{target}}$, a wide state distribution is more likely to have substantial tails in some low-cost region than a more peaked distribution as shown in Fig. 4a. In the early stages of learning, the predictive state uncertainty is largely due to propagating model uncertainties forward. If we predict a state distribution in a high-cost region, the saturating cost then leads to automatic *exploration* by favoring uncertain states, i.e., states in regions far from the target with a poor dynamics model. When visiting these regions during interaction with the physical system, subsequent model learning reduces the model uncertainty locally. In the subsequent policy evaluation, PILCO will predict a tighter state distribution in the situations described in Fig. 4.

If the mean of the state distribution is *close to the target* as in Fig. 4b, wide distributions are likely to have substantial tails in high-cost regions. By contrast, the mass of a peaked distribution is more concentrated in low-cost regions. In this case, the policy prefers peaked distributions close to the target, leading to *exploitation*.

To summarize, combining a probabilistic dynamics model, Bayesian inference, and a saturating cost leads to automatic exploration as long as the predictions are far from the target—even for a policy, which greedily minimizes the expected cost. Once close to the target, the policy does not substantially deviate from a confident trajectory that leads the system close to the target.[6]

7 Experimental Results

In this section, we assess PILCO's key properties and show that PILCO scales to high-dimensional control problems. Moreover, we demonstrate the hardware applicability of our learning framework on two real systems. In all cases, PILCO followed the steps outlined in Algorithm 1. To reduce the computational burden, we used the sparse GP method of [50] after 300 collected data points.

6. Code is available at http://mloss.org/software/view/508/.

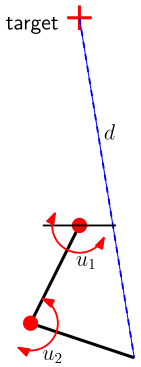


Fig. 5. Double pendulum with two actuators applying torques u_1 and u_2. The cost function penalizes the distance d to the target.

7.1 Evaluation of Key Properties

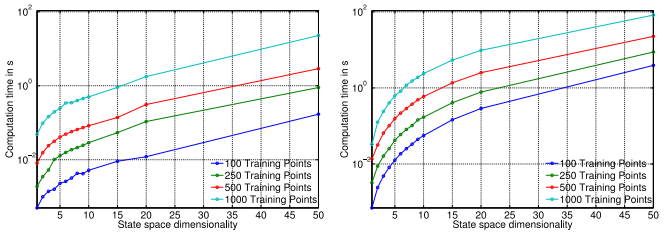
In the following, we assess the quality of the approximate inference method used for long-term predictions in terms of computational demand and learning speed. Moreover, we shed some light on the quality of the Gaussian approximations of the predictive state distributions and the importance of Bayesian averaging. For these assessments, we applied PILCO to two nonlinear control tasks, which are introduced in the following.

7.1.1 Task Descriptions

We considered two simulated tasks (double-pendulum swing-up, cart-pole swing-up) to evaluate important properties of the PILCO policy search framework: learning speed, quality of approximate inference, importance of Bayesian averaging, and hardware applicability. In the following we briefly introduce the experimental set-ups.

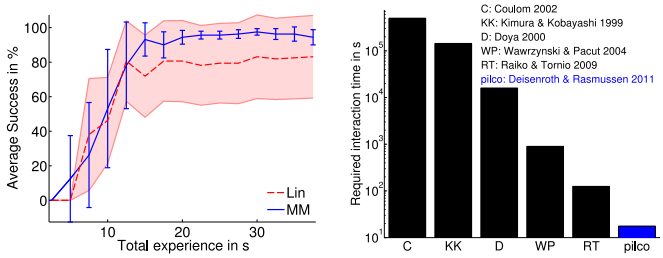
Double-pendulum swing-up with two actuators. The double pendulum system is a two-link robot arm with two actuators, see Fig. 5. The state $\boldsymbol{x}$ is given by the angles θ_1, θ_2 and the corresponding angular velocities $\dot{\theta}_1, \dot{\theta}_2$ of the inner and outer link, respectively, measured from being upright. Each link was of length 1m and mass 0.5 kg. Both torques u_1 and u_2 were constrained to $[-3, 3]\text{Nm}$. The control signal could be changed every $100\,\text{ms}$. In the meantime it was constant (zero-order-hold control). The objective was to learn a controller that swings the double pendulum up from an initial distribution $p(\boldsymbol{x}_0)$ around $\boldsymbol{\mu}_0 = [\pi, \pi, 0, 0]^\top$ and balances it in the inverted position with $\theta_1 = 0 = \theta_2$. The prediction horizon was $2.5\,\text{s}$.

The task is challenging since its solution requires the interplay of two correlated control signals. The challenge is to automatically learn this interplay from experience. To solve the double pendulum swing-up task, a nonlinear policy is required. Thus, we parametrized the preliminary policy as a deterministic GP, see (41), with 100 basis functions resulting in 812 policy parameters. We chose the saturating immediate cost in (44), where the Euclidean distance between the upright position and the tip of the outer link was penalized. We chose the cost width $\sigma_c = 0.5$, which means that the tip of the outer pendulum had to cross horizontal to achieve an immediate cost smaller than unity.



(a) Linearizing the mean function. (b) Moment matching.

Fig. 6. Empirical computational demand for approximate inference and derivative computation with GPs for a single time step, shown on a log scale. (a): Linearization of the posterior GP mean. (b): Exact moment matching.



(a) Average learning curves with 95% standard errors: moment matching (MM) and posterior GP linearization (Lin).

(b) Required interaction time of different RL algorithms for learning the cart-pole swing-up from scratch, shown on a log scale.

Fig. 7. Results for the cart-pole swing-up task. (a) Learning curves for moment matching and linearization (simulation task), (b) required interaction time for solving the cart-pole swing-up task compared with other algorithms.

Cart-Pole swing-up. The cart-pole system consists of a cart running on a track and a freely swinging pendulum attached to the cart. The state of the system is the position x of the cart, the velocity $\dot{x}$ of the cart, the angle θ of the pendulum measured from hanging downward, and the angular velocity $\dot{\theta}$. A horizontal force $u \in [-10, 10]\text{N}$ could be applied to the cart. The objective was to learn a controller to swing the pendulum up from around $\boldsymbol{\mu}_0 = [x_0, \dot{x}_0, \theta_0, \dot{\theta}_0]^\top = [0, 0, 0, 0]^\top$ and to balance it in the inverted position in the middle of the track, i.e., around $\boldsymbol{x}_{\text{target}} = [0, *, \pi, *]^\top$. Since a linear controller is not capable of solving the task [45], PILCO learned a nonlinear state-feedback controller based on a deterministic GP with 50 basis functions (see Section 5.3.2), resulting in 305 policy parameters to be learned. We chose the saturating cost in (44), where the Euclidean distance between the target position (pendulum upright in the middle of the track) and the tip of the pendulum was penalized.

In our simulation, we set the masses of the cart and the pendulum to $0.5\,\text{kg}$ each, the length of the pendulum to $0.5\,\text{m}$, and the coefficient of friction between cart and ground to $0.1\,\text{Ns/m}$. The prediction horizon was set to $2.5\,\text{s}$. The control signal could be changed every $100\,\text{ms}$. In the meantime, it was constant (zero-order-hold control). The only knowledge employed about the system was the length of the pendulum to find appropriate orders of magnitude to set the sampling frequency ($10\,\text{Hz}$) and the standard deviation of the cost function ($\sigma_c = 0.25\,\text{m}$), requiring the tip of the pendulum to move above horizontal not to incur full cost.

7.1.2 Approximate Inference Assessment

In the following, we evaluate the quality of the presented approximate inference methods for policy evaluation (moment matching as described in Section 4.1) and linearization of the posterior GP mean as described in Section 4.2) with respect to computational demand (Section 7.1.2) and learning speed (Section 7.1.2).

Computational demand For a single time step, the computational complexity of *moment matching* is $\mathcal{O}(n^2E^2D)$, where n is the number of GP training points, D is the input dimensionality, and E the dimension of the prediction. The most expensive computations are the entries of $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, which are given in (26). Each entry Q_{ij} requires evaluating a kernel, which is essentially a D-dimensional scalar product. The values $\boldsymbol{z}_{ij}$ are cheap to compute and $\boldsymbol{R}$ needs to be computed only once. We end up with $\mathcal{O}(n^2E^2D)$ since $\boldsymbol{Q}$ needs to be computed for all entries of the $E \times E$ predictive covariance matrix.

For a single time step, the computational complexity of *linearizing the posterior GP mean function* is $\mathcal{O}(n^2DE)$. The most expensive operation is the determination of $\boldsymbol{\Sigma}_w$ in (34), i.e., the model uncertainty at the mean of the input distribution, which scales in $\mathcal{O}(n^2D)$. This computation is performed for all E predictive dimensions, resulting in a computational complexity of $\mathcal{O}(n^2DE)$.

Fig. 6 illustrates the empirical computational effort for both linearization of the posterior GP mean and exact moment matching. We randomly generated GP models in $D = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 50$ dimensions and GP training set sizes of $n = 100, 250, 500, 1{,}000$ data points. We set the predictive dimension $E = D$. The CPU time (single core) for computing a predictive state distribution and the required derivatives are shown as a function of the dimensionality of the state. Four graphs are shown for setups with 100, 250, 500, and 1,000 GP training points, respectively. Fig. 6a shows the graphs for approximate inference based on linearization of the posterior GP mean, and Fig. 6b shows the corresponding graphs for exact moment matching on a logarithmic scale. Computations based on linearization were consistently faster by a factor of 5-10.

Learning Speed. For eight different random initial trajectories and controller initializations, PILCO followed Algorithm 1 to learn policies. In the cart-pole swing-up task, PILCO learned for 15 episodes, which corresponds to a total of $37.5\,\text{s}$ of data. In the double-pendulum swing-up task, PILCO learned for 30 episodes, corresponding to a total of $75\,\text{s}$ of data. To evaluate the learning progress, we applied the learned controllers after each policy search (see line 10 in Algorihm 1) 20 times for $2.5\,\text{s}$, starting from 20 different initial states $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$. The learned controller was considered successful when the tip of the pendulum was close to the target location from $2\,\text{s}$ to $2.5\,\text{s}$, i.e., at the end of the rollout.

- *Cart-Pole swing-up.* Fig. 7a shows PILCO's average learning success for the cart-pole swing-up task as a function of the total experience. We evaluated both approximate inference methods for policy evaluation, moment matching and linearization of the posterior GP mean function. Fig. 7a shows that learning using the computationally more demanding moment matching is more reliable than using the computationally more advantageous linearization.

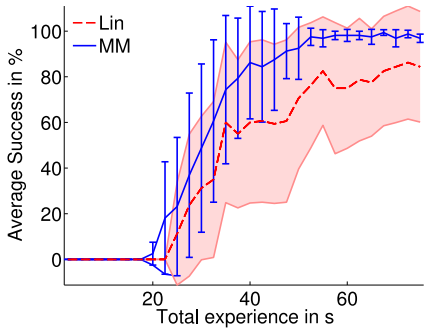


Fig. 8. Average success as a function of the total data used for learning (double pendulum swing-up). The blue error bars show the 95 percent confidence bounds of the standard error for the moment matching approximation, the red area represents the corresponding confidence bounds of success when using approximate inference by means of linearizing the posterior GP mean (Lin).

On average, after 15–20 s of experience, PILCO reliably, i.e., in ≈95 percent of the test runs, solved the cart-pole swing-up task, whereas the linearization resulted in a success rate of about 83 percent.

Fig. 7b relates PILCO's learning speed (blue bar) to other RL methods (black bars), which solved the cart-pole swing-up task from scratch, i.e., without human demonstrations or known dynamics models [11], [18], [27], [45], [56]. Dynamics models were only learned in [18], [45], using RBF networks and multilayered perceptrons, respectively. In all cases without state-space discretization, cost functions similar to ours (see (44)) were used. Fig. 7b stresses PILCO's data efficiency: PILCO outperforms any other currently existing RL algorithm by at least one order of magnitude.

- *Double-pendulum swing-up with two actuators.* Fig. 8 shows the learning curves for the double-pendulum swing-up task when using either moment matching or mean function linearization for approximate inference during policy evaluation. Fig. 8 shows that PILCO learns faster (learning already kicks in after 20 s of data) and overall more successfully with moment matching. Policy evaluation based on linearization of the posterior GP mean function achieved about 80 percent success on average, whereas moment matching on average solved the task reliably after about 50 s of data with a success rate ≈95 percent.

Summary. We have seen that both approximate inference methods have pros and cons: Moment matching requires more computational resources than linearization, but learns faster and more reliably. The reason why linearization did not reliably succeed in learning the tasks is that it gets relatively easily stuck in local minima, which is largely a result of underestimating predictive variances, an example of which is given in Fig. 2. Propagating too confident predictions over a longer horizon often worsens the problem. Hence, in the following, we focus solely on the moment matching approximation.

7.1.3 Quality of the Gaussian Approximation

PILCO strongly relies on the quality of approximate inference, which is used for long-term predictions and policy evaluation, see Section 4. We already saw differences between linearization and moment matching; however, both methods approximate predictive distributions by a Gaussian. Although we ultimately cannot answer whether this approximation is good under all circumstances, we will shed some light on this issue.

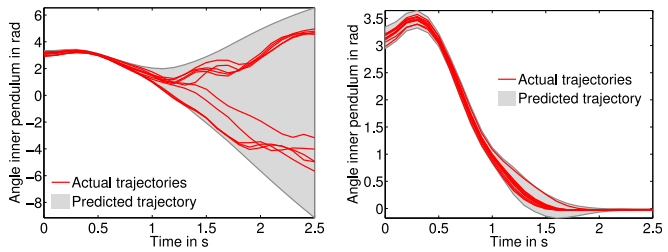


Fig. 9. Long-term predictive (Gaussian) distributions during planning (shaded) and sample rollouts (red). (a) In the early stages of learning, the Gaussian approximation is a suboptimal choice. (b) PILCO learned a controller such that the Gaussian approximations of the predictive states are good. Note the different scales in (a) and (b).

Fig. 9 shows a typical example of the angle of the inner pendulum of the double pendulum system where, in the early stages of learning, the Gaussian approximation to the multi-step ahead predictive distribution is not ideal. The trajectory distribution of a set of rollouts (red) is multimodal. PILCO deals with this inappropriate modeling by learning a controller that forces the actual trajectories into a unimodal distribution such that a Gaussian approximation is appropriate, Fig. 9b.

We explain this behavior as follows: Assuming that PILCO found different paths that lead to a target, a wide Gaussian distribution is required to capture the variability of the bimodal distribution. However, when computing the expected cost using a quadratic or saturating cost, for example, uncertainty in the predicted state leads to higher expected cost, assuming that the mean is close to the target. Therefore, PILCO uses its ability to choose control policies to push the marginally multimodal trajectory distribution into a single mode—from the perspective of minimizing expected cost with limited expressive power, this approach is desirable. Effectively, learning good controllers and models goes hand in hand with good Gaussian approximations.

7.1.4 Importance of Bayesian Averaging

Model-based RL greatly profits from the flexibility of nonparametric models as motivated in Section 2. In the following, we have a closer look at whether Bayesian models are strictly necessary as well. In particular, we evaluated whether Bayesian averaging is necessary for successfully learning from scratch. To do so, we considered the cart-pole swing-up task with two different dynamics models: first, the standard nonparametric Bayesian GP model, second, a nonparametric deterministic GP model, i.e., a GP where we considered only the posterior mean, but discarded the posterior model uncertainty when doing long-term predictions. We already described a similar kind of function representation to learn a deterministic policy, see Section 5.3.2. The difference to the policy is that in this section the deterministic

TABLE 1
Average Learning Success with Learned Nonparametric (NP) Transition Models (Cart-Pole Swing-Up)

	Bayesian NP model	Deterministic NP model
Learning success	**94.52%**	0%

GP is still nonparametric (new basis functions are added if we get more data), whereas the number of basis functions in the policy is fixed. However, the deterministic GP is no longer probabilistic because of the loss of model uncertainty, which also results in a degenerate model. Note that we still propagate uncertainties resulting from the initial state distribution $p(\boldsymbol{x}_0)$ forward.

Table 1 shows the average learning success of swinging the pendulum up and balancing it in the inverted position in the middle of the track. We used moment matching for approximate inference, see Section 4. Table 1 shows that learning is only successful when model uncertainties are taken into account during long-term planning and control learning, which strongly suggests Bayesian nonparametric models in model-based RL.

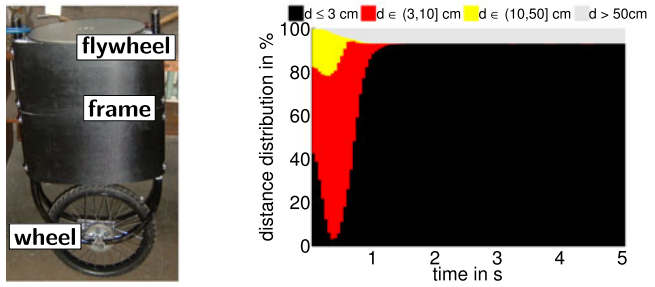
The reason why model uncertainties must be appropriately taken into account is the following: In the early stages of learning, the learned dynamics model is based on a relatively small data set. States close to the target are unlikely to be observed when applying random controls. Therefore, the model must extrapolate from the current set of observed states. This requires to predict function values in regions with large posterior model uncertainty. Depending on the choice of the deterministic function (we chose the MAP estimate), the predictions (point estimates) are very different. Iteratively predicting state distributions ends up in predicting trajectories, which are essentially arbitrary and not close to the target state either, resulting in vanishing policy gradients.

7.2 Scaling to Higher Dimensions: Unicycling

We applied PILCO to learning to ride a five-DoF unicycle with $\boldsymbol{x} \in \mathbb{R}^{12}$ and $\boldsymbol{u} \in \mathbb{R}^2$ in a realistic simulation of the one shown in Fig. 10a. The unicycle was $0.76\,\mathrm{m}$ high and consisted of a $1\,\mathrm{kg}$ wheel, a $23.5\,\mathrm{kg}$ frame, and a $10\,\mathrm{kg}$ flywheel mounted perpendicularly to the frame. Two torques could be applied to the unicycle: The first torque $|u_w| \leq 10\,\mathrm{Nm}$ was applied directly on the wheel to mimic a human rider using pedals. The torque produced longitudinal and tilt accelerations. Lateral stability of the wheel could be maintained by steering the wheel toward the falling direction of the unicycle and by applying a torque $|u_t| \leq 50\,\mathrm{Nm}$ to the flywheel. The dynamics of the robotic unicycle were described by 12 coupled first-order ODEs, see [24].

The objective was to learn a controller for riding the unicycle, i.e., to prevent it from falling. To solve the balancing task, we used the linear preliminary policy $\tilde{\pi}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ with $\boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\} \in \mathbb{R}^{28}$. The covariance $\boldsymbol{\Sigma}_0$ of the initial state was $0.25^2\boldsymbol{I}$ allowing each angle to be off by about 30 degree (twice the standard deviation).

PILCO differs from conventional controllers in that it learns a single controller for all control dimensions *jointly*. Thus, PILCO takes the correlation of all control and state dimensions into account during planning and control.



(a) Robotic unicycle.

(b) Histogram (after 1,000 test runs) of the distances of the flywheel from being upright.

Fig. 10. Robotic unicycle system and simulation results. The state space is $\mathbb{R}^{12}$, the control space $\mathbb{R}^2$.

Learning separate controllers for each control variable is often unsuccessful [37].

PILCO required about 20 trials, corresponding to an overall experience of about $30\,\mathrm{s}$, to learn a dynamics model and a controller that keeps the unicycle upright. A trial was aborted when the turntable hit the ground, which happened quickly during the five random trials used for initialization. Fig. 10b shows empirical results after 1,000 test runs with the learned policy: Differently-colored bars show the distance of the flywheel from a fully upright position. Depending on the initial configuration of the angles, the unicycle had a transient phase of about a second. After $1.2\,\mathrm{s}$, either the unicycle had fallen or the learned controller had managed to balance it very closely to the desired upright position. The success rate was approximately 93 percent; bringing the unicycle upright from extreme initial configurations was sometimes impossible due to the torque constraints.

7.3 Hardware Tasks

In the following, we present results from [15], [16], where we successfully applied the PILCO policy search framework to challenging control and robotics tasks, respectively. It is important to mention that no task-specific modifications were necessary, besides choosing a controller representation and defining an immediate cost function. In particular, we used the same standard GP priors for learning the forward dynamics models.

7.3.1 *Cart-Pole Swing-Up*

As described in [15], PILCO was applied to learning to control the *real* cart-pole system, see Fig. 11, developed by [26]. The masses of the cart and pendulum were $0.7\,\mathrm{kg}$ and $0.325\,\mathrm{kg}$, respectively. A horizontal force $u \in [-10, 10]\mathrm{N}$ could be applied to the cart.

PILCO successfully learned a sufficiently good dynamics model and a good controller fully automatically in only a handful of trials and a total experience of $17.5\,\mathrm{s}$, which also confirms the learning speed of the simulated cart-pole system in Fig. 7b despite the fact that the parameters of the system dynamics (masses, pendulum length, friction, delays, stiction, etc.) are different. Snapshots of a $20\,\mathrm{s}$ test trajectory are shown in Fig. 11; a video of the entire learning process is available at http://www.youtube.com/user/PilcoLearner.

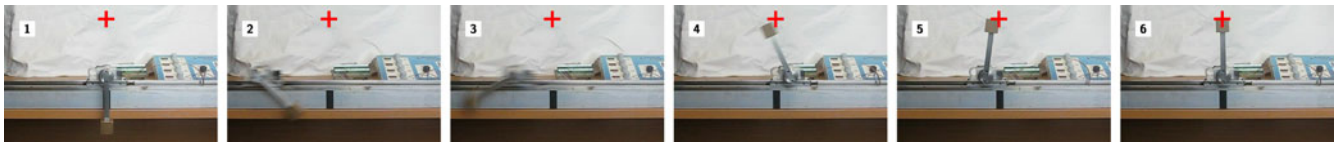


Fig. 11. Real cart-pole system [15]. Snapshots of a controlled trajectory of $20\,\mathrm{s}$ length after having learned the task. To solve the swing-up plus balancing, PILCO required only $17.5\,\mathrm{s}$ of interaction with the physical system.

7.3.2 Controlling a Low-Cost Robotic Manipulator

We applied PILCO to make a low-precision robotic arm learn to stack a tower of foam blocks—fully autonomously [16]. For this purpose, we used the lightweight robotic manipulator by Lynxmotion [1] shown in Fig. 12. The arm costs approximately $370 and possesses six controllable degrees of freedom: base rotate, three joints, wrist rotate, and a gripper (open/close). The plastic arm was controllable by commanding both a desired configuration of the six servos via their pulse durations and the duration for executing the command. The arm was very noisy: Tapping on the base made the end effector swing in a radius of about $2\,\mathrm{cm}$. The system noise was particularly pronounced when moving the arm vertically (up/down). Additionally, the servo motors had some play.

Knowledge about the joint configuration of the robot was not available. We used a PrimeSense depth camera [2] as an external sensor for visual tracking the block in the gripper of the robot. The camera was identical to the Kinect sensor, providing a synchronized depth image and a 640×480 RGB image at $30\,\mathrm{Hz}$. Using structured infrared light, the camera delivered useful depth information of objects in a range of about 0.5-$5\,\mathrm{m}$. The depth resolution was approximately $1\,\mathrm{cm}$ at a distance of $2\,\mathrm{m}$ [2].

Every $500\,\mathrm{ms}$, the robot used the 3D center of the block in its gripper as the state $\boldsymbol{x} \in \mathbb{R}^3$ to compute a continuous-valued control signal $\boldsymbol{u} \in \mathbb{R}^4$, which comprised the commanded pulse widths for the first four servo motors. Wrist rotation and gripper opening/closing were not learned. For block tracking we used real-time ($50\,\mathrm{Hz}$) color-based region growing to estimate the extent and 3D center of the object, which was used as the state $\boldsymbol{x} \in \mathbb{R}^3$ by PILCO.

As an initial state distribution, we chose $p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0 \mid \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ with $\boldsymbol{\mu}_0$ being a single noisy measurement of the 3D camera coordinates of the block in the gripper when the robot was in its initial configuration. The initial covariance $\boldsymbol{\Sigma}_0$ was diagonal, where the 95 percent-confidence bounds were the edge length b of the block. Similarly, the target state was set based on a single noisy measurement using the PrimeSense camera. We used linear preliminary policies, i.e., $\tilde{\pi}(\boldsymbol{x}) = \boldsymbol{u} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$, and initialized the controller parameters $\boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\} \in \mathbb{R}^{16}$ to zero. The euclidean distance d of the end effector from the camera was approximately 0.7–$2.0\,\mathrm{m}$, depending on the robot's configuration. The cost function in (44) penalized the Euclidean distance of the block in the gripper from its desired target location on top of the current tower. Both the frequency at which the controls were changed and the time discretization were set to $2\,\mathrm{Hz}$; the planning horizon T was $5\,\mathrm{s}$. After $5\,\mathrm{s}$, the robot opened the gripper and released the block.

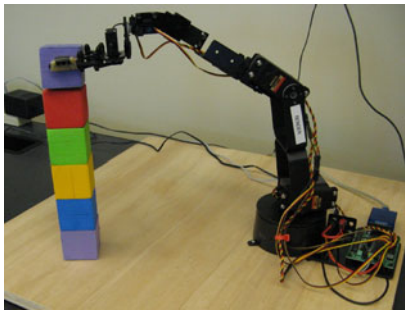


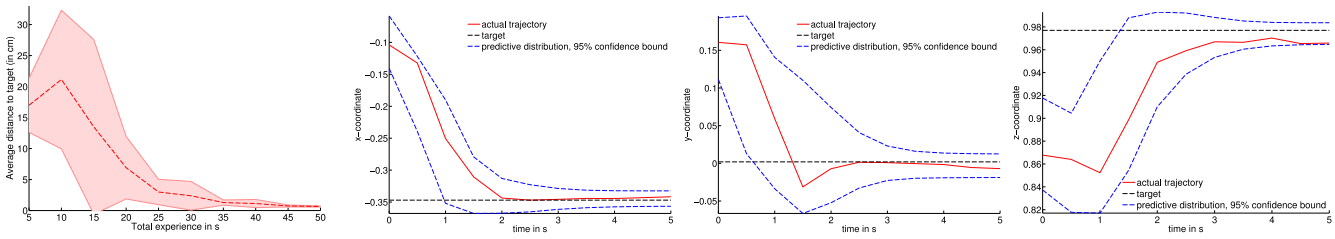
Fig. 12. Low-cost robotic arm by Lynxmotion [1]. The manipulator does not provide any pose feedback. Hence, PILCO learns a controller directly in the task space using visual feedback from a PrimeSense depth camera.

We split the task of building a tower into learning individual controllers for each target block B2-B6 (bottom to top), see Fig. 12, starting from a configuration, in which the robot arm was upright. All independently trained controllers shared the same initial trial.

The motion of the block in the end effector was modeled by GPs. The inferred system noise standard deviations, which comprised stochasticity of the robot arm, synchronization errors, delays, image processing errors, etc., ranged from 0.5 to $2.0\,\mathrm{cm}$. Here, the y-coordinate, which corresponded to the height, suffered from larger noise than the other coordinates. The reason for this is that the robot movement was particularly jerky in the up/down movements. These learned noise levels were in the right ballpark since they were slightly larger than the expected camera noise [2]. The signal-to-noise ratio in our experiments ranged from 2 to 6.

A total of ten learning-interacting iterations (including the random initial trial) generally sufficed to learn both good forward models and good controllers as shown in Fig. 13a, which displays the learning curve for a typical training session, averaged over ten test runs after each learning stage and all blocks B2–B6. The effects of learning became noticeable after about four learning iterations. After 10 learning iterations, the block in the gripper was expected to be very close (approximately at noise level) to the target. The required interaction time sums up to only $50\,\mathrm{s}$ per controller and $230\,\mathrm{s}$ in total (the initial random trial is counted only once). This speed of learning is difficult to achieve by other RL methods that learn from scratch as shown in Section 7.1.1.

Fig. 13b gives some insights into the quality of the learned forward model after 10 controlled trials. It shows the marginal predictive distributions and the actual trajectories of the block in the gripper. The robot learned to pay attention to stabilizing the y-coordinate quickly: Moving the arm up/down caused relatively large "system noise" as the arm was quite jerky in this direction: In the y-coordinate the predictive marginal distribution noticeably increases between $0\,\mathrm{s}$ and $2\,\mathrm{s}$. As soon as the y-coordinate was



(a) Average learning curve (block-stacking task). The horizontal axis shows the learning stage, the vertical axis the average distance to the target at the end of the episode.

(b) Marginal long-term predictive distributions and actually incurred trajectories. The red lines show the trajectories of the block in the end effector, the two dashed blue lines represent the 95% confidence intervals of the corresponding multi-step ahead predictions using moment matching. The target state is marked by the straight lines. All coordinates are measured in cm.

Fig. 13. Robot block stacking task: (a) Average learning curve with 95 percent standard error, (b) Long-term predictions.

stabilized, the predictive uncertainty in all three coordinates collapsed. Videos of the block-stacking robot are available at http://www.youtube.com/user/PilcoLearner.

8 Discussion

We have shed some light on essential ingredients for successful and efficient policy learning: (1) a probabilistic forward model with a faithful representation of model uncertainty and (2) Bayesian inference. We focused on very basic representations: GPs for the probabilistic forward model and Gaussian distributions for the state and control distributions. More expressive representations and Bayesian inference methods are conceivable to account for multi-modality, for instance. However, even with our current set-up, PILCO can already learn complex control and robotics tasks. In [8], our framework was used in an industrial application for throttle valve control in a combustion engine.

PILCO is a model-based policy search method, which uses the GP forward model to predict state sequences given the current policy. These predictions are based on deterministic approximate inference, e.g., moment matching. Unlike all model-free policy search methods, which are inherently based on sampling trajectories [14], PILCO exploits the learned GP model to compute analytic gradients of an approximation to the expected long-term cost J^π for policy search. Finite differences or more efficient sampling-based approximations of the gradients require many function evaluations, which limits the effective number of policy parameters [14], [42]. Instead, PILCO computes the gradients analytically and, therefore, can learn thousands of policy parameters [15].

It is possible to exploit the learned GP model for sampling trajectories using the PEGASUS algorithm [39], for instance. Sampling with GPs can be straightforwardly parallelized, and was exploited in [32] for learning meta controllers. However, even with high parallelization, policy search methods based on trajectory sampling do usually not rely on gradients [7], [30], [32], [40] and are practically limited by a relatively small number of a few tens of policy parameters they can manage [38].[7]

7. "Typically, PEGASUS policy search algorithms have been using [...] maybe on the order of ten parameters or tens of parameters; so, 30, 40 parameters, but not thousands of parameters [...]", Ng [38].

In Section 6.1, we discussed PILCO's natural exploration property as a result of Bayesian averaging. It is, however, also possible to explicitly encourage additional exploration in a UCB (upper confidence bounds) sense [6]: Instead of summing up expected immediate costs, see (2), we would add the sum of cost standard deviations, weighted by a factor $\kappa \in \mathbb{R}$. Then, $J^\pi(\boldsymbol{\theta}) = \sum_t \big(\mathbb{E}[c(\boldsymbol{x}_t)] + \kappa\sigma[c(\boldsymbol{x}_t)]\big)$. This type of utility function is also often used in experimental design [10] and Bayesian optimization [9], [33], [41], [51] to avoid getting stuck in local minima. Since PILCO's approximate state distributions $p(\boldsymbol{x}_t)$ are Gaussian, the cost standard deviations $\sigma[c(\boldsymbol{x}_t)]$ can often be computed analytically. For further details, we refer the reader to [12].

One of PILCO's key benefits is the reduction of model errors by explicitly incorporating model uncertainty into planning and control. PILCO, however, does not take temporal correlation into account. Instead, model uncertainty is treated as noise, which can result in an under-estimation of model uncertainty [49]. On the other hand, the moment-matching approximation used for approximate inference is typically a conservative approximation.

In this paper, we focused on learning controllers in MDPs with transition dynamics that suffer from *system noise*, see (1). The case of *measurement noise* is more challenging: Learning the GP models is a real challenge since we no longer have direct access to the state. However, approaches for training GPs with noise on both the training inputs and training targets yield initial promising results [36]. For a more general POMDP set-up, Gaussian Process Dynamical Models (GPDMs) [29], [54] could be used for learning both a transition mapping and the observation mapping. However, GPDMs typically need a good initialization [53] since the learning problem is very high dimensional.

In [25], the PILCO framework was extended to allow for learning reference tracking controllers instead of solely controlling the system to a fixed target location. In [16], we used PILCO for planning and control in *constrained environments*, i.e., environments with obstacles. This learning set-up is important for practical robot applications. By discouraging obstacle collisions in the cost function, PILCO was able to find paths around obstacles without ever colliding with them, not even during training. Initially, when the model was uncertain, the policy was conservative to stay away from obstacles. The PILCO framework has been applied in the context of model-based imitation learning

to learn controllers that minimize the Kullback-Leibler divergence between a distribution of demonstrated trajectories and the predictive distribution of robot trajectories [20], [21]. Recently, PILCO has also been extended to a multi-task set-up [13].

9 Conclusion

We have introduced PILCO, a practical model-based policy search method using analytic gradients for policy learning. PILCO advances state-of-the-art RL methods for continuous state and control spaces in terms of learning speed by at least an order of magnitude. Key to PILCO's success is a principled way of reducing the effect of model errors in model learning, long-term planning, and policy learning. PILCO is one of the few RL methods that has been directly applied to robotics without human demonstrations or other kinds of informative initializations or prior knowledge.

The PILCO learning framework has demonstrated that Bayesian inference and nonparametric models for learning controllers is not only possible but also practicable. Hence, nonparametric Bayesian models can play a fundamental role in classical control set-ups, while avoiding the typically excessive reliance on explicit models.

Acknowledgments

The research leading to these results has received funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement #270327, ONR MURI grant N00014-09-1-1052, Intel Labs, and the Department of Computing, Imperial College London.

References

[1] http://www.lynxmotion.com, 2014.
[2] http://www.primesense.com, 2014.
[3] P. Abbeel, M. Quigley, and A.Y. Ng, "Using Inaccurate Models in Reinforcement Learning," *Proc. 23rd Int'l Conf. Machine Learning*, 2006.
[4] K.J. Aström and B. Wittenmark, *Adaptive Control*. Dover Publications, 2008.
[5] C.G. Atkeson and J.C. Santamaría, "A Comparison of Direct and Model-Based Reinforcement Learning," *Proc. IEEE Int'l Conf. Robotics and Automation*, 1997.
[6] P. Auer, "Using Confidence Bounds for Exploitation-Exploration Trade-Offs," *J. Machine Learning Research*, vol. 3, pp. 397-422, 2002.
[7] J.A. Bagnell and J.G. Schneider, "Autonomous Helicopter Control Using Reinforcement Learning Policy Search Methods," *Proc. Int'l Conf. Robotics and Automation*, 2001.
[8] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll, "Learning Throttle Valve Control Using Policy Search," *Proc. European Conf. Machine Learning and Knowledge Discovery in Databases*, 2013.
[9] E. Brochu, V.M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," Technical Report TR-2009-023, Dept. of Computer Science, Univ. of British Columbia, 2009.
[10] K. Chaloner and I. Verdinelli, "Bayesian Experimental Design: A Review," *Statistical Science*, vol. 10, pp. 273-304, 1995.
[11] R. Coulom, *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*, PhD thesis, Inst. Nat'l Polytechnique de Grenoble, 2002.
[12] M.P. Deisenroth, *Efficient Reinforcement Learning Using Gaussian Processes*. KIT Scientific Publishing, 2010.
[13] M.P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-Task Policy Search," http://arxiv.org/abs/1307.0813, July 2013.
[14] M.P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, NOW Publishers, 2013.
[15] M.P. Deisenroth and C.E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," *Proc. Int'l Conf. Machine Learning*, 2011.
[16] M.P. Deisenroth, C.E. Rasmussen, and D. Fox, "Learning to Control a Low-Cost Manipulator Using Data-Efficient Reinforcement Learning," *Proc. Robotics: Science and Systems*, 2011.
[17] M.P. Deisenroth, C.E. Rasmussen, and J. Peters, "Gaussian Process Dynamic Programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508-1524, 2009.
[18] K. Doya, "Reinforcement Learning in Continuous Time and Space," *Neural Computation*, vol. 12, no. 1, pp. 219-245, 2000.
[19] Y. Engel, S. Mannor, and R. Meir, "Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning," *Proc. Int'l Conf. Machine Learning*, 2003.
[20] P. Englert, A. Paraschos, J. Peters, and M.P. Deisenroth, "Model-Based Imitation Learning by Probabilistic Trajectory Matching," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2013.
[21] P. Englert, A. Paraschos, J. Peters, and M.P. Deisenroth, "Probabilistic Model-Based Imitation Learning," *Adaptive Behavior*, vol. 21, pp. 388-403, 2013.
[22] S. Fabri and V. Kadirkamanathan, "Dual Adaptive Control of Nonlinear Stochastic Systems Using Neural Networks," *Automatica*, vol. 34, no. 2, pp. 245-253, 1998.
[23] A.A. Fel'dbaum, "Dual Control Theory, Parts I and II," *Automation and Remote Control*, vol. 21, no. 11, pp. 874-880, 1961.
[24] D. Forster, "Robotic Unicycle," report, Dept. of Eng., Univ. of Cambridge, United Kingdom, 2009.
[25] J. Hall, C.E. Rasmussen, and J. Maciejowski, "Reinforcement Learning with Reference Tracking Control in Continuous State Spaces," *Proc. IEEE Int'l Conf. Decision and Control*, 2011.
[26] T.T. Jervis and F. Fallside, "Pole Balancing on a Real Rig Using a Reinforcement Learning Controller," Technical Report CUED/F-INFENG/TR 115, Univ. of Cambridge, Dec. 1992.
[27] H. Kimura and S. Kobayashi, "Efficient Non-Linear Control by Combining Q-learning with Local Linear Controllers," *Proc. 16th Int'l Conf. Machine Learning*, 1999.
[28] J. Ko and D. Fox, "GP-BayesFilters: Bayesian Filtering Using Gaussian Process Prediction and Observation Models," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, 2008.
[29] J. Ko and D. Fox, "Learning GP-BayesFilters via Gaussian Process Latent Variable Models," *Proc. Robotics: Science and Systems*, 2009.
[30] J. Ko, D.J. Klein, D. Fox, and D. Haehnel, "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2007.
[31] K.P. Körding and D.M. Wolpert, "The Loss Function of Sensorimotor Learning," *Proc. Nat'l Academy of Sciences of USA*, vol. 101, pp. 9839-9842, 2004.
[32] A. Kupcsik, M.P. Deisenroth, J. Peters, and G. Neumann, "Data-Efficient Generalization of Robot Skills with Contextual Policy Search," *Proc. AAAI Conf. Artificial Intelligence*, 2013.
[33] D. Lizotte, "Practical Bayesian Optimization," PhD thesis, Univ. of Alberta, Edmonton, Alberta, 2008.
[34] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge Univ. Press, 2003.
[35] D.C. McFarlane and K. Glover, *Robust Controller Design Using Normalised Coprime Factor Plant Descriptions*, vol. 138, Lecture Notes in Control and Information Sciences. Springer-Verlag, 1989.
[36] A. McHutchon and C.E. Rasmussen, "Gaussian Process Training with Input Noise," *Proc. Advances in Neural Information Processing Systems*, 2011.
[37] Y. Naveh, P.Z. Bar-Yoseph, and Y. Halevi, "Nonlinear Modeling and Control of a Unicycle," *J. Dynamics and Control*, vol. 9, no. 4, pp. 279-296, Oct. 1999.
[38] A.Y. Ng, Stanford Engineering Everywhere CS229—Machine Learning, Lecture 20, http://see.stanford.edu/materials/aimlcs229/transcripts/MachineLearning-Lecture20.html, 2008.
[39] A.Y. Ng and M. Jordan, "PEGASUS: A Policy Search Method for Large Mdps and Pomdps," *Proc. Conf. Uncertainty in Artificial Intelligence*, 2000.
[40] A.Y. Ng, H.J. Kim, M.I. Jordan, and S. Sastry, "Autonomous Helicopter Flight via Reinforcement Learning," *Proc. Advances in Neural Information Processing Systems*, 2004.

[41] M.A. Osborne, R. Garnett, and S.J. Roberts, "Gaussian Processes for Global Optimization," *Proc. Int'l Conf. Learning and Intelligent Optimization*, 2009.
[42] J. Peters and S. Schaal, "Policy Gradient Methods for Robotics," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robotics Systems*, 2006.
[43] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, pp. 682-697, 2008.
[44] J. Quiñonero-Candela, A. Girard, J. Larsen, and C.E. Rasmussen, "Propagation of Uncertainty in Bayesian Kernel Models—Application to Multiple-Step Ahead Forecasting," *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing*, 2003.
[45] T. Raiko and M. Tornio, "Variational Bayesian Learning of Nonlinear Hidden State-Space Models for Model Predictive Control," *Neurocomputing*, vol. 72, no. 16-18, pp. 3702-3712, 2009.
[46] C.E. Rasmussen and M. Kuss, "Gaussian Processes in Reinforcement Learning," *Proc. Advances in Neural Information Processing Systems*, 2004.
[47] C.E. Rasmussen and C.K.I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.
[48] S. Schaal, "Learning From Demonstration," *Proc. Advances in Neural Information Processing Systems*, 1997.
[49] J.G. Schneider, "Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning," *Proc. Advances in Neural Information Processing Systems*, 1997.
[50] E. Snelson and Z. Ghahramani, "Sparse Gaussian Processes Using Pseudo-Inputs," *Proc. Advances in Neural Information Processing Systems*, 2006.
[51] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design," *Proc. Int'l Conf. Machine Learning*, 2010.
[52] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[53] R. Turner, M.P. Deisenroth, and C.E. Rasmussen, "State-Space Inference and Learning with Gaussian Processes," *Proc. Int'l Conf. Artificial Intelligence and Statistics*, 2010.
[54] J.M. Wang, D.J. Fleet, and A. Hertzmann, "Gaussian Process Dynamical Models for Human Motion," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 283-298, Feb. 2008.
[55] C.J.C.H. Watkins, "Learning from Delayed Rewards," PhD thesis, Univ. of Cambridge, Cambridge, United Kingdom, 1989.
[56] P. Wawrzynski and A. Pacut, "Model-Free Off-Policy Reinforcement Learning in Continuous Environment," *Proc. Int'l Joint Conf. Neural Networks*, 2004.
[57] A. Wilson, A. Fern, and P. Tadepalli, "Incorporating Domain Models into Bayesian Optimization for RL," *Proc. European Conf. Machine Learning and Knowledge Discovery in Databases*, 2010.
[58] B. Wittenmark, "Adaptive Dual Control Methods: An Overview," *Proc. IFAC Symp. Adaptive Systems in Control and Signal Processing*, 1995.

Marc Peter Deisenroth conducted his PhD research at the Max Planck Institute for Biological Cybernetics (2006-2007) and at the University of Cambridge (2007-2009) and received the PhD degree in 2009. He is a research fellow in the Department of Computing at Imperial College London. He is also an adjunct researcher in the Computer Science Department at TU Darmstadt, where he has been a group leader and a senior researcher from December 2011 to August 2013. From February 2010 to December 2011, he has been a research associate at the University of Washington. His research interests center around modern Bayesian machine learning and its application to autonomous control and robotic systems.

Dieter Fox received the PhD degree from the University of Bonn, Germany. He is a professor in the Department of Computer Science & Engineering at the University of Washington, where he heads the UW Robotics and State Estimation Lab. From 2009 to 2011, he was also a director of the Intel Research Labs Seattle. Before going to UW, he spent two years as a postdoctoral researcher at the CMU Robot Learning Lab. His research is in artificial intelligence, with a focus on state estimation applied to robotics and activity recognition. He has published over 150 technical papers and is a coauthor of the text book *Probabilistic Robotics*. He is an editor of the *IEEE Transactions on Robotics*, was a program co-chair of the 2008 AAAI Conference on Artificial Intelligence, and served as the program chair of the 2013 Robotics Science and Systems conference. He is a fellow of the AAAI and a senior member of the IEEE.

Carl Edward Rasmussen is a reader in information engineering at the Department of Engineering at the University of Cambridge. He was a junior research group leader at the Max Planck Institute for Biological Cybernetics in Tübingen and a senior research fellow at the Gatsby Computational Neuroscience Unit at UCL. He has wide interests in probabilistic methods in machine learning, including nonparametric Bayesian inference, and has coauthored the text book *Gaussian Processes for Machine Learning*.