Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Marc Peter Deisenroth
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

Carl Edward Rasmussen
Dept. of Engineering
University of Cambridge
Cambridge, UK

Dieter Fox
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

***Abstract*—Over the last years, there has been substantial progress in robust manipulation in unstructured environments. The long-term goal of our work is to get away from precise, but very expensive robotic systems and to develop affordable, potentially imprecise, self-adaptive manipulator systems that can interactively perform tasks such as playing with children. In this paper, we demonstrate how a low-cost off-the-shelf robotic system can learn closed-loop policies for a stacking task in only a handful of trials—from scratch. Our manipulator is inaccurate and provides no pose feedback. For learning a controller in the work space of a Kinect-style depth camera, we use a model-based reinforcement learning technique. Our learning method is data efficient, reduces model bias, and deals with several noise sources in a principled way during long-term planning. We present a way of incorporating state-space constraints into the learning process and analyze the learning gain by exploiting the sequential structure of the stacking task.**

I. Introduction

Over the last years, there has been substantial progress in robust manipulation in unstructured environments. While existing techniques have the potential to solve various household manipulation tasks, they typically rely on extremely expensive robot hardware [12]. The long-term goal of our work is to develop affordable, light-weight manipulator systems that can interactively play with children. A key problem of cheap manipulators, however, is their inaccuracy and the limited sensor feedback, if any. In this paper, we show how to use a cheap, off-the-shelf robotic manipulator ($370) and a Kinect-style (http://www.xbox.com/kinect) depth camera (<$120) to learn a block stacking task [2, 1] under state-space constraints. We use data-efficient reinforcement learning (RL) to train a controller directly in the work space of the depth camera.

Fully autonomous RL methods typically require many trials to successfully solve a task (e.g., Q-learning), a good initialization (e.g., by imitation [3]), or a deep understanding of the system. If this knowledge is unavailable, due to the lack of understanding of complicated dynamics or because a solution is simply not known, data-intensive learning methods are required. In a robotic system, however, many physical interactions with the environment are often infeasible and lead to worn-out robots. The more fragile a robotic system the more important data-efficient learning methods are.

To sidestep these problems, we build on PILCO (probabilistic inference for learning control), a data-efficient model-based (indirect) policy search method [7] that reduces model bias, a typical problem of model-based methods: PILCO employs a flexible probabilistic non-parametric Gaussian process (GP) dynamics model and takes model uncertainty consistently into account during long-term planning. PILCO learns good controllers from scratch, i.e., with random initializations; no deep understanding of the system is required. In this paper, we show how obstacle information provided by the depth camera can be incorporated into PILCO's planning and learning to avoid collisions even during training, and how knowledge can be efficiently transferred across related tasks.

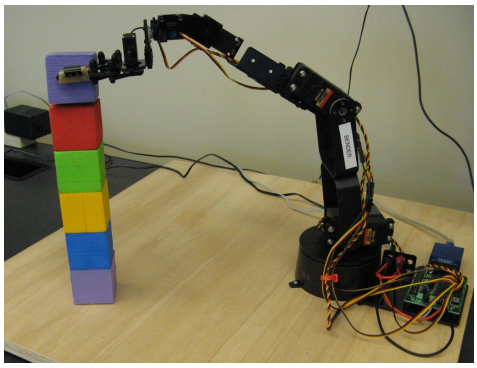


Fig. 1. Low-cost robotic arm by Lynxmotion [1] performing a block stacking task. Since the manipulator does not provide any pose feedback, our system learns a controller directly in the task space using visual feedback from a Kinect-style depth camera.

The paper is structured as follows. After discussing related work, we describe the task to be solved, the low-cost hardware used, and a basic tracking algorithm in Sec. III. Sec. IV summarizes the PILCO framework and details how we incorporate collision avoidance into long-term planning under uncertainty. Sec. V presents the experimental results. Sec. VI concludes the paper with a discussion.

II. Related Work

In [11], a model-free policy learning method is presented, which relies on rollouts sampled from the system. Even in a simple task (mountain-car) with only two policy parameters, 80 rollouts are required. For more complicated tasks, the number of required rollouts quickly goes into the thousands.

[5] propose a consistent learning-planning method in partially observable domains. A compact model of a discrete latent space is learned and used for control learning by means of point-based value iteration [16]. The approach in [5] for

learning a latent-space dynamics model requires thousands of trajectories. Furthermore, it does not naturally deal with continuous (latent) domains or model uncertainty.

In recent years, GP dynamics models were more often used for learning robot dynamics [9, 10, 14]. However, they are usually not used for long-term planning and policy learning, but rather for myopic control and trajectory following. Typically, the training data for the GP dynamics models are obtained either by motor babbling [9] or by demonstrations [14]. For the purpose of data-efficient fully autonomous learning, these approaches are not suitable: Motor babbling is data-inefficient and does not guarantee good models along a good trajectory; demonstrations would contradict fully autonomous learning.

Other algorithms that use GP dynamics models in an RL setup were proposed in [20, 8]. In [20, 8], value function models have to be maintained, which becomes difficult in higher-dimensional state spaces. Although the approaches in [20, 8] do long-term planning for finding a policy, they cannot directly deal with constraints in the state space (e.g., obstacles).

Model-based RL methods are typically better suited for data-efficient learning than model-free methods. However, they often employ a certainty equivalence assumption [22, 23, 4] by assuming that the learned model is a good approximation of the latent system dynamics. This assumption leads to "model bias", which often makes learning from scratch "daunting" [22], especially when only a few samples from the system are available. Reducing model bias requires accounting for model uncertainty during planning [23].

Unlike most other model-based approaches, PILCO [6, 7] does *not* make a certainty equivalence assumption on the learned model or simply take the maximum likelihood model. Instead, it learns a *probabilistic* dynamics model and explicitly incorporates model uncertainty into long-term planning [7]. Unlike [23, 4, 20, 8], PILCO, however, neither requires sampling methods for planning, nor needs to maintain an explicit value function model.

In [18], the authors also aim at developing low-cost manipulators. However, while their focus is on building novel manipulation *hardware* equipped with sufficient sensing, our goal is to develop reasoning algorithms to be used with cheap off-the-shelf systems.

III. PRELIMINARIES

In this paper, we describe how a low-precision robotic arm can learn to stack a tower of foam blocks—fully autonomously. We employ the following assumptions: First, since grasping is not the focus of this work, we assume that the block is placed in the robot's gripper. Second, the arm's joint angles and velocities are not measured internally. However, the location of the center of the block in the robot's gripper can be determined using the depth camera. Third, no desired path/trajectory is a priori known. This also excludes human demonstrations. Fourth, we assume that the initial location and the target location of the block in the gripper are fixed.

Trajectory-following methods such as Jacobian-transpose control [13] are not suitable in our case: A desired trajectory is not known in advance. Simply following a straight path between the initial and the target state might not succeed due to obstacles (e.g., partial stack). We furthermore have to cope with multiple sources of uncertainty: camera noise, time synchronization (camera/controller), idealized assumptions (e.g., constant duration between measurements), delays, image processing noise, and robot arm noise. The camera noise and the robot arm noise are the major noise sources, see Sec. III-A for details. For long-term planning and controller learning, all these uncertainties have to be taken into account.

A. Hardware Description

We use a lightweight robotic arm by Lynxmotion [1], see also Fig. 1. The arm costs approximately \$370 and has six controllable degrees of freedom: base rotate, three joints, wrist rotate, and a gripper (open/close). The plastic arm can be controlled by commanding both a desired configuration of the six servos (via their pulse durations, which range from $0.75\,\mathrm{ms}$–$2.25\,\mathrm{ms}$) and the duration for executing the command. The arm is very noisy: Tapping on the base makes the end effector swing in a radius of about $2\,\mathrm{cm}$. The system noise is especially pronounced when moving the arm vertically (up/down). The robotic arm is shipped without any sensors. Thus, neither the joint angles nor the configuration of the servos can be obtained directly. Instead of equipping the robot with further sensors and/or markers, we demonstrate that good controllers can be learned without additional information.

We use a PrimeSense depth camera [2] for visual tracking. The camera is identical to the Kinect sensor, providing a synchronized depth image and a 640×480 color (RGB) image at $30\,\mathrm{Hz}$. Using structured light, the camera delivers useful depth information of objects in a range of about $0.5\,\mathrm{m}$–$5\,\mathrm{m}$. The depth resolution is approximately $1\,\mathrm{cm}$ at $2\,\mathrm{m}$ distance [2]. The total cost of the robot and the camera is about \$500. ROS [19] handles the communication with the hardware.

B. Block Tracking

At every time step, the robot uses the center of the block in its gripper to compute a continuous-valued control signal $\mathbf{u} \in \mathbb{R}^4$, which comprises the commanded pulse widths for the first four servo motors. Wrist rotation and gripper opening/closing are not learned. For tracking the block in the gripper of the robot arm, we use a simple but fast blob tracking algorithm. At the beginning of an experiment, the user marks the object in the gripper of the robot by clicking on it in a display. Assuming that the object has a uniform color, we use color-based region growing starting at the clicked pixel to estimate the extent and 3D center of the object, which is used as the state $\mathbf{x} \in \mathbb{R}^3$ by the RL algorithm. Finding the 3D center of the block requires less than $0.02\,\mathrm{s}$ per frame.

IV. POLICY LEARNING WITH STATE-SPACE CONSTRAINTS

In the following, we summarize the PILCO-framework [6, 7] for learning a good closed-loop policy (state-feedback controller) $\pi : \mathbb{R}^3 \to \mathbb{R}^4, \mathbf{x} \mapsto \mathbf{u}$. Here, $\mathbf{x}$ is called the *state* defined as the coordinates of the center (x_c, y_c, z_c) of the block

Algorithm 1 PILCO

init: Set controller parameters $\boldsymbol{\psi}$ to random.
Apply random control signals and record data.
repeat
  Learn probabilistic GP dynamics model using all data
  repeat ▷ Model-based policy search
    Approx. inference for policy evaluation: get $J^\pi(\boldsymbol{\psi})$
    Gradients $\mathrm{d}J^\pi(\boldsymbol{\psi})/\,\mathrm{d}\boldsymbol{\psi}$ for policy improvement
    Update parameters $\boldsymbol{\psi}$ (e.g., CG or L-BFGS).
  until convergence; **return** $\boldsymbol{\psi}^*$
  Set $\pi^* \leftarrow \pi(\boldsymbol{\psi}^*)$.
  Apply π^* to robot (single trial/episode); record data.
until task learned

in the gripper. We attempt to learn this policy from scratch, i.e., with only very general prior knowledge about the task and the solution itself. Moreover, we want to find π in only a few trials, i.e., we require a data-efficient learning method.

As a criterion to judge the performance of a controller π, we use the long-term expected return

$$J^\pi = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]\,, \tag{1}$$

of a trajectory $(\mathbf{x}_0, \ldots, \mathbf{x}_T)$ when applying π. In Eq. (1), T is the prediction horizon and $c(\mathbf{x}_t)$ is the instantaneous cost of being in state $\mathbf{x}$ at time t. If not stated otherwise, throughout this paper, we use a saturating cost function $c = -\exp(-d^2/\sigma_c^2)$ that penalizes Euclidean distances d of the block in the end effector from the target location $\mathbf{x}_{\text{target}}$. We assume the policy π is parametrized by $\boldsymbol{\psi}$. PILCO learns a good parametrized policy by following Alg. 1 [7].

A. Probabilistic Dynamics Model

To avoid certainty equivalence assumptions on the learned model, PILCO takes model uncertainties into account during planning. Hence, a (posterior) distribution over plausible dynamics models is required. We use GPs [21] to infer this posterior distribution from currently available experience.

Following [21], we briefly introduce the notation and standard prediction models for GPs, which are used to infer a distribution on a latent function f from noisy observations $y_i = f(\mathbf{x}_i) + \varepsilon$, where in this paper, we consider $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ i.i.d. system noise. A GP is completely specified by a mean function $m(\,\cdot\,)$ and a positive semidefinite covariance function $k(\,\cdot\,,\,\cdot\,)$, also called a *kernel*. Throughout this paper, we consider a prior mean function $m \equiv 0$ and the squared exponential (SE) kernel with automatic relevance determination defined as

$$k(\mathbf{x}, \mathbf{x}') = \alpha^2 \exp\big(-\tfrac{1}{2}(\mathbf{x}-\mathbf{x}')^\top \boldsymbol{\Lambda}^{-1}(\mathbf{x}-\mathbf{x}')\big)\,. \tag{2}$$

Here, $\boldsymbol{\Lambda} \coloneqq \mathrm{diag}([\ell_1^2, \ldots, \ell_D^2])$ depends on the characteristic length-scales ℓ_i, and α^2 is the variance of the latent function f. Given n training inputs $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and corresponding training targets $\mathbf{y} = [y_1, \ldots, y_n]^\top$, the GP hyper-parameters (length-scales ℓ_i, signal variance α^2, noise variance σ_ε^2) are learned using evidence maximization [21].

The posterior predictive distribution $p(f_*|\mathbf{x}_*)$ of the function value $f_* = f(\mathbf{x}_*)$ for an arbitrary, but known, test input $\mathbf{x}_*$ is Gaussian with mean and variance

$$m_f(\mathbf{x}_*) = \mathbb{E}_f[f_*] = \mathbf{k}_*^\top (\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y} = \mathbf{k}_*^\top \boldsymbol{\beta}\,, \tag{3}$$

$$\sigma_f^2(\mathbf{x}_*) = \mathrm{var}_f[f_*] = k_{**} - \mathbf{k}_*^\top (\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{k}_* + \sigma_\varepsilon^2\,, \tag{4}$$

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

In our robotic system, see Sec. III, the GP models the function $f : \mathbb{R}^7 \to \mathbb{R}^3$, $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \mapsto \Delta_t \coloneqq \mathbf{x}_t - \mathbf{x}_{t-1} + \boldsymbol{\varepsilon}_t$, where $\boldsymbol{\varepsilon}_t \in \mathbb{R}^3$ is i.i.d. Gaussian system noise. The training inputs and targets to the GP model are tuples $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, and the corresponding differences Δ_t, respectively.

B. Long-Term Planning through Approximate Inference

Minimizing and evaluating J^π in Eq. (1) requires long-term predictions of the state evolution. To obtain the state distributions $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_T)$, we cascade one-step predictions. This requires mapping uncertain (test) inputs through a GP model. In the following, we assume that these test inputs are Gaussian distributed and extend the results from [17, 6, 7] to long-term planning in stochastic systems with control inputs.

For predicting $\mathbf{x}_t$ from $p(\mathbf{x}_{t-1})$, we require a joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$. To compute this distribution, we use that $\mathbf{u}_{t-1} = \pi(\mathbf{x}_{t-1})$, i.e., the control is a function of the state: We first compute the predictive control signal $p(\mathbf{u}_{t-1})$ and subsequently the cross-covariance $\mathrm{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]$. Finally, we approximate $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ by a Gaussian distribution with the correct mean and covariance. The computation depends on the policy parametrization $\boldsymbol{\psi}$ of the policy π. In this paper, we assume $\mathbf{u}_{t-1} = \pi(\mathbf{x}_{t-1}) = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{b}$, with $\boldsymbol{\psi} = \{\mathbf{A}, \mathbf{b}\}$. With $p(\mathbf{x}_{t-1}) = \mathcal{N}\big(\mathbf{x}_{t-1} \,|\, \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}\big)$, we obtain

$$p(\mathbf{u}_{t-1}) = \mathcal{N}\big(\mathbf{u}_{t-1} \,|\, \boldsymbol{\mu}_u, \boldsymbol{\Sigma}_u\big)\,,$$
$$\boldsymbol{\mu}_u = \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{b}\,, \quad \boldsymbol{\Sigma}_u = \mathbf{A}\boldsymbol{\Sigma}_{t-1}\mathbf{A}^\top\,,$$

by applying standard results from linear-Gaussian models. In this example, π is a linear function of $\mathbf{x}_{t-1}$ and, thus, the desired joint distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is exactly Gaussian and given by

$$\mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_{t-1}\\ \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{b}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{t-1} & \boldsymbol{\Sigma}_{t-1}\mathbf{A}^\top\\ \mathbf{A}\boldsymbol{\Sigma}_{t-1} & \mathbf{A}\boldsymbol{\Sigma}_{t-1}\mathbf{A}^\top\end{bmatrix}\right) \tag{5}$$

with the cross-covariance $\mathrm{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}] = \boldsymbol{\Sigma}_{t-1}\mathbf{A}^\top$. For many other interesting controller parametrizations, the mean and covariance can be computed analytically [6], although $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ may no longer be exactly Gaussian.

From now on, we assume a joint Gaussian distribution $p(\tilde{\mathbf{x}}_{t-1}) = \mathcal{N}\big(\tilde{\mathbf{x}}_{t-1} \,|\, \tilde{\boldsymbol{\mu}}_{t-1}, \tilde{\boldsymbol{\Sigma}}_{t-1}\big)$ at time $t-1$, where we define $\tilde{\mathbf{x}} \coloneqq [\mathbf{x}^\top \mathbf{u}^\top]^\top$ and $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$ are the respective mean and covariance of this augmented variable. When predicting

$$p(\Delta_t) = \int p(f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1}) p(\tilde{\mathbf{x}}_{t-1})\,\mathrm{d}\tilde{\mathbf{x}}_{t-1}\,, \tag{6}$$

we integrate out the random variable $\tilde{\mathbf{x}}_{t-1}$. The transition probability $p(f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1})$ is obtained from the posterior GP.

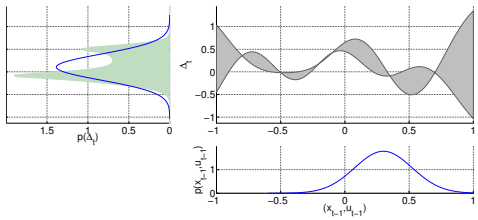


Fig. 2. GP prediction at an uncertain input. The input distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is assumed Gaussian (lower right). Propagating it through the GP model (upper right) yields the shaded distribution $p(\Delta_t)$ in the upper left, which is approximated by a Gaussian with the exact mean and variance.

Computing the exact predictive distribution in Eq. (6) is analytically intractable. Thus, we approximate $p(\Delta_t)$ by a Gaussian with the exact mean and variance (moment matching). Fig. 2 illustrates the scenario. Note that for computing the mean μ_Δ and the variance σ_Δ^2 of the predictive distribution, the standard GP predictive distribution (see Eqs. (3) and (4), respectively) does not suffice because $\tilde{\mathbf{x}}_{t-1}$ is not given deterministically.

Assume the mean $\boldsymbol{\mu}_\Delta$ and the covariance $\boldsymbol{\Sigma}_\Delta$ of the predictive distribution $p(\Delta_t)$ are known. Then, a Gaussian approximation $\mathcal{N}\big(\mathbf{x}_t \,|\, \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$ to the desired state distribution $p(\mathbf{x}_t)$ has mean and covariance

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{\mu}_\Delta \tag{7}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} + \boldsymbol{\Sigma}_\Delta + \operatorname{cov}[\mathbf{x}_{t-1}, \Delta_t] + \operatorname{cov}[\Delta_t, \mathbf{x}_{t-1}]\,, \tag{8}$$

$$\operatorname{cov}[\mathbf{x}_{t-1}, \Delta_t] = \operatorname{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]\boldsymbol{\Sigma}_u^{-1}\operatorname{cov}[\mathbf{u}_{t-1}, \Delta_t]\,, \tag{9}$$

respectively. The computation of the required cross-covariances in Eq. (9) depends on the policy parametrization, but can often be computed analytically.

In the following, we compute the mean μ_Δ and the variance σ_Δ^2 of the predictive distribution $p(\Delta_t)$, see Eq. (6). We focus on the univariate case and refer to [6] for the multivariate case.

1) Mean: Following the law of iterated expectations,

$$\begin{aligned}\mu_\Delta &= \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[\mathbb{E}_f[f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1}]] = \mathbb{E}_{\mathbf{x}_*}[m_f(\tilde{\mathbf{x}}_{t-1})] \\ &= \int m_f(\tilde{\mathbf{x}}_{t-1})\mathcal{N}\big(\tilde{\mathbf{x}}_{t-1} \,|\, \tilde{\boldsymbol{\mu}}_{t-1}, \tilde{\boldsymbol{\Sigma}}_{t-1}\big)\,\mathrm{d}\tilde{\mathbf{x}}_{t-1} = \boldsymbol{\beta}^\top \mathbf{q}\end{aligned} \tag{10}$$

with $\mathbf{q} = [q_1, \ldots, q_n]^\top$ and $\boldsymbol{\beta} = (\mathbf{K} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}$. The entries of $\mathbf{q} \in \mathbb{R}^n$ are given as

$$\begin{aligned}q_i &= \int k(\mathbf{x}_i, \mathbf{x}_*)\mathcal{N}\big(\tilde{\mathbf{x}}_{t-1} \,|\, \tilde{\boldsymbol{\mu}}_{t-1}, \tilde{\boldsymbol{\Sigma}}_{t-1}\big)\,\mathrm{d}\tilde{\mathbf{x}}_{t-1} \\ &= \frac{\alpha^2 \exp\big(-\frac{1}{2}(\mathbf{x}_i - \tilde{\boldsymbol{\mu}}_{t-1})^\top(\tilde{\boldsymbol{\Sigma}}_{t-1} + \boldsymbol{\Lambda})^{-1}(\mathbf{x}_i - \tilde{\boldsymbol{\mu}}_{t-1})\big)}{\sqrt{|\tilde{\boldsymbol{\Sigma}}_{t-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|}}\,.\end{aligned}$$

2) Variance: Using the law of total variance, we obtain

$$\begin{aligned}\sigma_\Delta^2 &= \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[m_f(\tilde{\mathbf{x}}_{t-1})^2] + \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[\sigma_f^2(\tilde{\mathbf{x}}_{t-1}) \\ &\quad - \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[m_f(\tilde{\mathbf{x}}_{t-1})]^2 \\ &= \boldsymbol{\beta}^\top\mathbf{Q}\boldsymbol{\beta} + \alpha^2 - \operatorname{tr}\big((\mathbf{K} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{Q}\big) - \mu_\Delta^2 + \sigma_\varepsilon^2\,,\end{aligned} \tag{11}$$

where $\operatorname{tr}(\,\cdot\,)$ is the trace. The entries of $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are

$$\begin{aligned}Q_{ij} = {}& k(\mathbf{x}_i, \tilde{\boldsymbol{\mu}}_{t-1})k(\mathbf{x}_j, \tilde{\boldsymbol{\mu}}_{t-1})|2\tilde{\boldsymbol{\Sigma}}_{t-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{-\frac{1}{2}} \\ &\times \exp\big(\tfrac{1}{2}\mathbf{z}_{ij}^\top(2\tilde{\boldsymbol{\Sigma}}_{t-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I})^{-1}\tilde{\boldsymbol{\Sigma}}_{t-1}\mathbf{z}_{ij}\big)\end{aligned}$$

with $\boldsymbol{\zeta}_i := (\mathbf{x}_i - \tilde{\boldsymbol{\mu}}_{t-1})$ and $\mathbf{z}_{ij} := \boldsymbol{\Lambda}^{-1}(\boldsymbol{\zeta}_i + \boldsymbol{\zeta}_j)$.

Note that both μ_Δ and σ_Δ^2 are functionally dependent on the mean $\boldsymbol{\mu}_u$ and the covariance $\boldsymbol{\Sigma}_u$ of the control signal through $\tilde{\boldsymbol{\mu}}_{t-1}$ and $\tilde{\boldsymbol{\Sigma}}_{t-1}$, respectively, see Eqs. (10) and (11). We can see from Eqs. (10) and (11) that the uncertainty about the latent function f (according to the GP posterior) is integrated out, which explicitly accounts for model uncertainty.

C. Controller Learning through Indirect Policy Search

From Sec. IV-B we know how to cascade one-step predictions to obtain Gaussian approximations to the predictive distributions $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_T)$. To evaluate the expected return J^π in Eq. (1), it remains to compute the expected values

$$\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \int c(\mathbf{x}_t)\mathcal{N}\big(\mathbf{x}_t \,|\, \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)\,\mathrm{d}\mathbf{x}_t \tag{12}$$

of the instantaneous cost c with respect to the predictive state distributions. We assume that the cost function c is chosen such that this integral can be solved analytically.

To apply a gradient-based policy search to find controller parameters $\boldsymbol{\psi}$ that minimize J^π, see Eq. (1), we first swap the order of differentiation and summation in Eq. (1). With $\mathcal{E}_t := \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$ we obtain

$$\frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\mu}_t}\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\Sigma}_t}\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\psi}}\,. \tag{13}$$

The total derivatives of the mean $\boldsymbol{\mu}_t$ and the covariance $\boldsymbol{\Sigma}_t$ of $p(\mathbf{x}_t)$ with respect to the policy parameters $\boldsymbol{\psi}$ can be computed analytically by repeated application of the chain-rule to Eqs. (7), (8), (9), (10), (11). This also involves computing the partial derivatives of $\partial\boldsymbol{\mu}_u/\partial\boldsymbol{\psi}$ and $\partial\boldsymbol{\Sigma}_u/\partial\boldsymbol{\psi}$. We omit further lengthy details here, but point out that these derivatives are computed analytically [6, 7]. This allows for standard gradient-based non-convex optimization methods, e.g., CG or L-BFGS, which return an optimized parameter vector $\boldsymbol{\psi}^*$.

D. Planning with State-Space Constraints

In a classical RL setup, it is assumed that the learner is not aware of any constraints in the state space, but has to discover walls etc. by running into them and gaining a high penalty. In a robotic setup, this general, but not necessary, assumption is less desirable because the robot can be damaged.

If constraints (e.g., obstacles) in the state space are known a priori, we would like to incorporate this prior knowledge directly into planning and policy learning. We propose to define obstacles as "undesirable" regions, i.e., regions the robot is supposed to avoid. We define "undesirability" as a penalty in the instantaneous cost function c, which we re-define as

$$c(\mathbf{x}) = -\sum\nolimits_{k=1}^{K} c_k^+(\mathbf{x}) + \sum\nolimits_{j=1}^{J} \iota_j c_j^-(\mathbf{x})\,, \tag{14}$$

where c_k^+ are desirable states (e.g., the target state) and c_j^- are undesirable states (e.g., obstacles), weighted by $\iota_j \geq 0$. Bigger values for ι_j make the policy more averse to a particular undesirable state. In this paper, we always set $\iota_j = 1$. For c_k^+ and c_j^-, we choose squared-exponentials, which trade off exploration with exploitation when averaging according to the

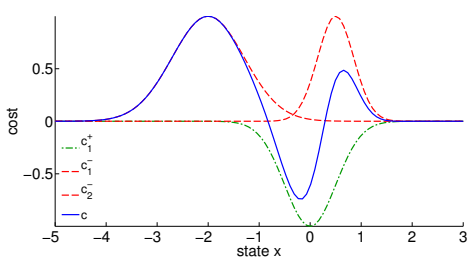


Fig. 3. Cost function that takes constraints (e.g., obstacles) into account by making them "undesirable". The non-solid curves are the individual components c_j^- and c_k^+, see Eq. (14), the solid curve is their sum c.

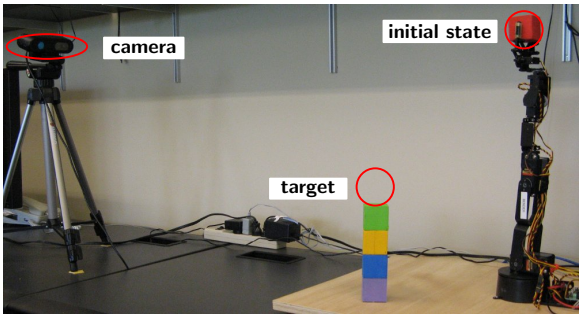


Fig. 4. Learning setup 1: The initial position is *above* the tower's top.

state distribution [6]. The squared exponentials are unnormalized with potentially different widths $\mathbf{\Sigma}_k^+$. The widths of the individual constraints c_j^- define how "soft" the constraints are. Hard constraints would be described by very peaked squared exponentials c_j^- with $\iota_j \to \infty$. The idea is related to [24], where planning is performed with fully known dynamics and a piecewise linear controller.

Fig. 3 illustrates Eq. (14) with two penalties c_j^- and one reward c_k^+. The figure shows that if an undesirable state and a desirable state are close, the total cost c somewhat trades off between both objectives. Furthermore, the optimal state $x_* \in \arg\min_x c(x)$ no longer corresponds to $x_*^+ \in \arg\min_x c^+(x)$: Moving a little bit away from the target state (away from the undesirable state) is optimal.

The expectations of the cost in Eq. (14) and the derivatives with respect to the mean $\boldsymbol{\mu}_t$ and the covariance $\mathbf{\Sigma}_t$ of the state distribution $p(\mathbf{x}_t)$ can be computed for each individual c_k^+ and c_j^- and summed up. Then, we apply the chain-rule according to Eq. (13) for the gradient-based policy search.

Phrasing constraints in terms of undesirability in the cost function in Eq. (14) still allows for fully probabilistic long-term planning and for a guidance of the robot through the state space *without* "experiencing" obstacles by running into them.

Collisions within a Bayesian inference framework can be discouraged, but not strictly excluded in expectation. This does not mean that averaging out uncertainties is wrong—it rather tells us that it is not expected to violate constraints with a certain *confidence*. A faithful description of predictive uncertainty is often more worth than claiming full confidence and occasionally violating constraints unexpectedly.

V. Experimental Validation

In the following, we analyze PILCO's performance on the task of learning to stack a tower of six foam blocks B1–B6 (bottom to top), see Fig. 1. The tower's bottom block B1 was given. To apply PILCO, we need to specify the initial state distribution, the target state, the cost function, the controller parametrization, and optionally obstacles.

As an initial state distribution, we chose $p(\mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_0 \,|\, \boldsymbol{\mu}_0, \mathbf{\Sigma}_0\big)$ with $\boldsymbol{\mu}_0$ being a single (noisy) measurement of the initial block location using the tracking method from Sec. III-B. The initial covariance $\mathbf{\Sigma}_0$ was diagonal with the 95%-confidence bounds being the edge length b of the block. The target state was set to a single noisy measurement using the tracking method from Sec. III-B.

The first term of the immediate cost in Eq. (14) that describes favorable states was set to $-\frac{1}{4}\sum_{k=1}^4 \exp(-\frac{1}{2}d^2/\sigma_k^2)$, where $d := \|\mathbf{x}_t - \mathbf{x}_{\text{target}}\|$ and $\sigma_k = \{\frac{1}{4}b, \frac{1}{2}b, b, 2b\}$, $k = 1, \ldots, 4$, and b being the edge length of the foam block. The scale mixture of squared exponentials makes the choice of a single σ_k less important and yields non-zero policy gradients $\mathrm{d}J/\,\mathrm{d}\boldsymbol{\psi}$ even relatively far away from the target $\mathbf{x}_{\text{target}}$.

We used linear controllers, i.e., $\pi(\mathbf{x}) = \mathbf{u} = \mathbf{A}\mathbf{x} + \mathbf{b}$, and initialized the controller parameters $\boldsymbol{\psi} = \{\mathbf{A}, \mathbf{b}\} \in \mathbb{R}^{16}$ to $\mathbf{0}$.

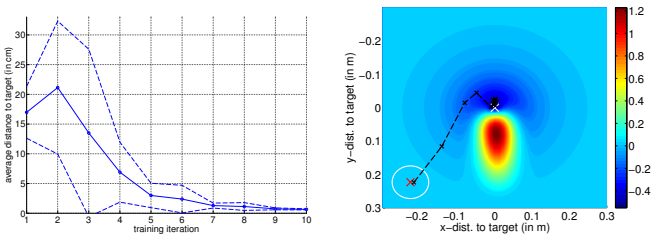
The Euclidean distance d of the end effector from the camera was approximately $0.7\,\mathrm{m}$–$2.0\,\mathrm{m}$, depending on the robot configuration. Both the control sampling frequency and the time discretization Δ_t were set to rather slow $2\,\mathrm{Hz}$; the planning/episode length T was $5\,\mathrm{s}$. After $5\,\mathrm{s}$, the robot opened the gripper and freed the block.

The motion of the block grasped by the end effector was modeled by GPs as described in Sec. IV-A. The inferred system noise standard deviations, which comprise stochasticity of the robotic arm, synchronization errors, delays, and image processing errors, ranged from $0.5\,\mathrm{cm}$ to $2.0\,\mathrm{cm}$. These learned noise levels were in the right ballpark: They were slightly larger than the expected camera noise [2]. The signal-to-noise ratio in our experiments ranged from 2 to 6.

In Sec. V-A, we evaluate the applicability of the PILCO framework to autonomous block stacking when starting from a fully upright robot configuration. For each block, an independent controller is learned. In Sec. V-B, we analyze PILCO's ability to exploit useful prior information by transferring knowledge from one learned controller to another one. In Sec. V-C, the robot learned building a tower, where the initial position was below the topmost block. For this task, state-space constraints such as obstacles were taken into account during planning, see Sec. IV-D. Videos can be found at http://www.cs.uw.edu/ai/Mobile_Robotics/projects/robot-rl.

A. Independent Controllers for Building a Tower

We split the task of building a tower into learning individual controllers for each target block B2–B6 (bottom to top) starting from the same initial configuration, in which the robot arm was upright, see Fig. 4.



(a) Typical learning curve as a function of training iterations.

(b) Two-dimensional slice through the cost function with obstacles encoded.

Fig. 5. (a) Typical learning curve. The horizontal axis shows the learning stage, the vertical axis shows the average distance to the target at time T (with 95% standard error). (b) Two-dimensional slice through the cost function defined in task space with observed end effector trajectory. The z-coordinate is set to be at the target. Red and blue colors indicate high and low costs, respectively.

All independently trained controllers shared the same initial trial. A total of ten learning-interacting iterations (including the random initial trial) generally sufficed to learn both good dynamics models and good controllers. Fig. 5(a) shows a learning curve for a typical training session (averaged over ten test runs after each learning stage and all blocks B2–B6). Learning noticeably kicked in after about four 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 s per controller and 230 s in total (the initial random trial is counted only once). This speed of learning is very difficult to achieve by other RL methods that learn from scratch [7].

A standard myopic task-space control method such as Jacobian-transpose control [13] (using the GP dynamics model) could solve the problem, too, without any planning. However, this approach benefits from a good dynamics model along the desired trajectory in task space. Obtaining this model through motor babbling can be data inefficient.

B. Sequential Transfer Learning

We now evaluate how much we can speed up learning by transferring knowledge. To do so, we exploited the sequential nature of the block-stacking task. In Sec. V-A, we trained five independent controllers for the five different blocks B2–B6. In the following, we report results for training the first controller for the bottom block B2 as earlier. Subsequently, however, we reused both the dynamics model and the controller parameters when learning the controller for the next block. This initialization of the learning process was more informed than a random one and gave the learner a head-start: Learning to stack a new block on the topmost one requires a sufficiently good dynamics model in similar parts of the state space.

Tab. I summarizes the gains through this kind of transfer learning. Learning to stack a block of six blocks (the base B1 is given) required only 90 s of experience when PILCO exploited the sequential nature of this task, compared to 230 s when five controllers were learned independently from scratch, see Sec. V-A. In other words, the amount of data required for learning independent controllers for B2–B3 was sufficient to learn stacking the entire tower of six blocks when knowledge was transferred. Sequential controller learning required only two additional trials per block to achieve a performance similar or better to a corresponding controller learned independent of all other controllers, see Tab. II.

TABLE I
TRANSFER LEARNING GAINS (SETUP 1).

	B2	B2–B3	B2–B4	B2–B5	B2–B6
trials (seconds) independent controllers	10 (50)	19 (95)	28 (140)	37 (185)	46 (230)
trials (seconds) sequential controllers	10 (50)	12 (60)	14 (70)	16 (80)	**18 (90)**
speedup (independent/sequential)	1	1.58	2	2.31	2.56

TABLE II
AVERAGE BLOCK DEPOSIT SUCCESS IN 10 TEST TRIALS AND FOUR DIFFERENT (RANDOM) LEARNING INITIALIZATIONS (SETUP 1).

	B2	B3	B4	B5	B6
independent controllers	92.5%	80%	42.5%	96%	100%
sequential controllers	92.5%	87.5%	82.5%	95%	95%

Tab. II reports the rates for successfully depositing the block on the top of the current foam tower in 10 test trials and four different learning initializations. Failures were largely caused by the foam block bumping off the topmost block. The table indicates that sequentially trained controllers perform at least as well as the independently learned controllers—despite the fact that they use substantially fewer training iterations, see Tab. I. In one of the four learning setups, 10 learning iterations did not suffice to learn a good (independent) controller for B4, which is the reason for the corresponding poor average deposit success in Tab. II. The corresponding sequential controller exploited the informative initialization of the dynamics model from B3 and did not suffer from this kind of failure. Although deposit failure feedback was not available to the learner, the deposit success is good across 10 test trials and four different training setups. Note that high-precision control with the Lynx arm is difficult because the arm can be very jerky.

Knowledge transfer through the dynamics model is very valuable. Additionally transferring the controller parameters ψ in the sequential setup is not decisive: Resetting the controller parameters to zero and retraining leads to very similar results. An "informative" controller initialization without the dynamics model would not help learning because the controller parameters are learned in the light of the current dynamics model.

C. Collision Avoidance

If prior knowledge about the environment is known, it is helpful to incorporate this into planning; at least in order to extend the robot's life time. Thus far, we assumed that the learning system is fully uninformed about the environment. This means, when there was an obstacle, say, a table, the robot learned about the table as follows: When the robot arm banged on the table, the predictive trajectory of the block in the gripper and the observed trajectory did not match well. In subsequent trials, when the GP dynamics model accounted for this experience, the robot discovered a better trajectory that did not get stuck on the surface of the table.

In the following, we consider a modification of the exper-

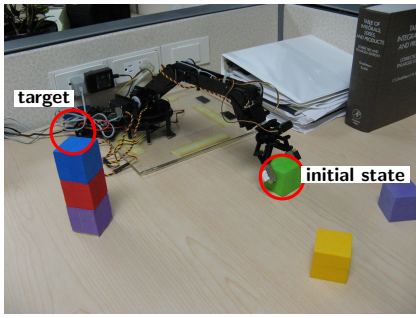


Fig. 6. Learning setup 2: The initial position is *below* the tower's top.

imental setup: Obstacles in the environment were explicitly incorporated into planning, see Sec. IV-D when the robot was supposed to learn building a block tower, where the initial state was *below* the target state, see Fig. 6.

Since a desired *trajectory* was not known in advance, Jacobian-transpose control would result in a collision between the block in the end effector and the tower's top-most block. Since the control dimension $\mathbb{R}^4$ was larger than the task space dimension $\mathbb{R}^3$, a linear policy $\pi : \mathbb{R}^3 \to \mathbb{R}^4$ was still sufficient to solve this problem, which is nonlinear in task space.

In the experimental setup considered, we modeled the tower as a set of blocks. Following Sec. IV-D, we added a Gaussian-shaped penalty for each block: The mean $\boldsymbol{\mu}_j^-$ of the penalty was the center of the block, and the covariance was set to $\boldsymbol{\Sigma}_j^- = (\frac{3}{4})^2\mathbf{I}$. Defining obstacles in task space can be automated using 3D object detection algorithms and mixture-of-Gaussians clustering. Fig. 5(b) shows the most interesting two-dimensional slice of the cost function in task space around the target state. The third coordinate is assumed to be in the target. The penalty due to the tower obstacle is defined through the high-cost regions. Note that the lowest cost does not occur exactly in the target but slightly above the tower. In Fig. 5(b), the ellipse is a two-dimensional projection of the initial state distribution $p(\mathbf{x}_0)$, the dashed line is an observed trajectory when applying the controller. It can be seen how the robot arm avoids the tower to deposit the block on top of the stack.

To evaluate the effectiveness of our approach to collision-avoidance, PILCO learned five independent controllers for building a tower of foam blocks based on planning either with or without constraints. All controllers shared the same single random trial and 10 (controlled) training rollouts. This corresponds to a total experience of 55 s per controller. Tab. III summarizes the results for these setups (averaged over four different random learning initializations) under the following aspects: effectiveness of collision avoidance, block deposit success rate, and controller quality.

First, we investigated the effectiveness of collision avoidance. We defined a "collision" to occur when the robot arm collided with the tower of foam blocks. Tab. III indicates that planning with state-space constraints led to fewer collisions with the obstacles than agnostic training. Note that the numbers in Tab. III are the collisions during *training*, not during testing. This means that even in the early stages of learning (when the dynamics model was very uncertain), PILCO learned a "cautious" controller to avoid collisions.

Second, Tab. III reports the block-deposit success rates for 10 test runs (and four different training initializations) after 10 training iterations. Here, we see that planning with state-space constraints led to a substantially higher success rate in depositing blocks. Planning without state-space constraints often led to a controller that slightly struck the topmost block of the tower, i.e., it caused a collision.

Finally, Tab. III reports the distances of the block in the gripper at time T, averaged over 10 test runs (after the corresponding controllers have been trained) and four different training setups. At time T, the gripper opened and dropped the block. The distances were measured independent of a collision. In both constrained and unconstrained planning the learned controller brought the block in the gripper close to the target location. Note that the distances in Tab. III approximately equal the noise level (image capture, image processing, robot arm). The results here do not suggest that any training setup leads to better "drop-off locations" on average. However, learning without state-space constraints started showing improvements one or two stages earlier than learning based on planning with collision avoidance (not reported in Tab. III).

VI. Discussion

PILCO is not optimal control because it merely finds *a* solution for the task. There are no guarantees of global optimality: Since the optimization problem for learning the policy parameters is not convex, the discovered solution is invariably only a local optimum. It is also conditional on the experience the learner was exposed to.

PILCO exploits analytic gradients of an approximation to the expected return J^π for an indirect policy search. Thus, PILCO does not need to maintain an explicit value function model, which does not scale well to high dimensions. Sampling for estimating the policy gradients [15] is unnecessary.

Computing a plan for a *given* policy required about one second of computation time. Learning the policy requires iterative probabilistic planning and updating the policy parameters. The exact duration depends on size of the GP training set. In this paper's experiments, PILCO required between one and three minutes to learn a policy for a given dynamics model. Thus, data efficiency comes with the price of more computational overhead. Nevertheless, *applying the policy (testing)* is real-time capable as it requires a simple function evaluation $\mathbf{u}_t = \pi(\mathbf{x}_t)$, which often is a matrix-vector multiplication.

In principle, there is nothing that prevents PILCO from scaling to higher-dimensional problems, see [7] for some examples. Policy evaluation and gradient computation scale cubically in the state dimension [6]. Although policy search scales only quadratically in the size n of the GP training set [6], this is PILCO's practical bottleneck. Hence, we use sparse GP approximations for $n \geq 400$. This is quickly exceeded if the underlying dynamics are complicated and/or high sampling frequencies are used.

TABLE III
EXPERIMENTAL RESULTS FOR PLANNING WITH AND WITHOUT COLLISION AVOIDANCE (SETUP 2).

without collision avoidance	B2	B3	B4	B5	B6
collisions during training	12/40 (30%)	11/40 (27.5%)	13/40 (32.5%)	18/40 (45%)	21/40 (52.5%)
block deposit success rate	50%	43%	37%	47%	33%
distance (in cm) to target at time T	1.39 ± 0.81	0.73 ± 0.36	0.65 ± 0.35	0.71 ± 0.46	0.59 ± 0.34
with collision avoidance	B2	B3	B4	B5	B6
collisions during training	0/40 (0%)	2/40 (5%)	1/40 (2.5%)	3/40 (7.5%)	1/40 (2.5%)
block deposit success rate	90%	97%	90%	70%	97%
distance (in cm) to target at time T	0.89 ± 0.80	0.65 ± 0.33	0.67 ± 0.46	0.80 ± 0.37	1.34 ± 0.56

VII. CONCLUSION

We presented a data-efficient and fully autonomous approach for learning robot control even when the robotic system is very imprecise. Our model-based policy search method profits from closed-form approximate inference for policy evaluation and analytic gradients for policy learning. To avoid collisions, we presented a way of taking knowledge about obstacles in the environment into account during planning and controlling under uncertainty. Furthermore, we evaluated the gains of reusing dynamics models in a sequential task. With only very general prior knowledge about the robot and the task to be learned, we demonstrated that good controllers for a low-cost robotic system consisting of a cheap manipulator and depth camera could be learned in only a few trials.

Despite the limitations of our current system, we believe that the overall framework can be readily adapted to handle more complex tasks. In future work, we aim to learn more general controllers that can deal with arbitrary start locations of the gripper and the target stack. Grasping objects with such a cheap manipulator is also a promising research direction.

ACKNOWLEDGEMENTS

M.P. Deisenroth and D. Fox have been supported by ONR MURI grant N00014-09-1-1052 and by Intel Labs.

REFERENCES

[1] http://www.lynxmotion.com.
[2] http://www.primesense.com.
[3] P. Abbeel and A. Y. Ng. Exploration and Apprenticeship Learning in Reinforcement Learning. In *ICML*, 2005.
[4] J. A. Bagnell and J. G. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *ICRA*, pp. 1615–1620, 2001.
[5] B. Boots, S. M. Siddiqi, and G. J. Gordon. Closing the Learning-Planning Loop with Predictive State Representations. In *R:SS*, 2010.
[6] M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. KIT Scientific Publishing, 2010. ISBN 978-3-86644-569-7.
[7] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *ICML*, 2011.
[8] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7–9):1508–1524, 2009.
[9] J. Ko, D. J. Klein, D. Fox, and D. Haehnel. Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. In *ICRA*, 2007.
[10] J. Ko and D. Fox. Learning GP-BayesFilters via Gaussian Process Latent Variable Models. *Autonomous Robots*, 30(1), 2011.
[11] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, 2011.
[12] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth Grasp Point Detection based on Multiple-View Geometric Cues with Application to Robotic Towel Folding. In *ICRA*, 2010.
[13] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational Space Control: A Theoretical and Empirical Comparison. *IJRR*, 27(737), June 2008.
[14] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics*, 23(15):2015–2034, 2009.
[15] J. Peters and S. Schaal. Policy Gradient Methods for Robotics. In *IROS*, pp. 2219–2225, 2006.
[16] J. Pineau, G. Gordon, and S. Thrun. Point-based Value Iteration: An Anytime Algorithm for POMDPs. In *IJCAI*, pp. 1025–1030, 2003.
[17] J. Quiñonero-Candela, A. Girard, J. Larsen, and C. E. Rasmussen. Propagation of Uncertainty in Bayesian Kernel Models—Application to Multiple-Step Ahead Forecasting. In *ICASSP*, pp. 701–704, 2003.
[18] M. Quigley, R. Brewer, S. P. Soundararaj, V. Pradeep, Q. Le, and A. Y. Ng. Low-cost Accelerometers for Robotic Manipulator Perception. In *IROS*, 2010.
[19] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. ROS: An Open-Source Robot Operating System. In *ICRA Open-Source Software Workshop*, 2009.
[20] C. E. Rasmussen and M. Kuss. Gaussian Processes in Reinforcement Learning. In *NIPS*, pp. 751–759, 2004.
[21] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
[22] S. Schaal. Learning From Demonstration. In *NIPS*, pp. 1040–1046, 1997.
[23] J. G. Schneider. Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *NIPS*, 1997.
[24] M. Toussaint and C. Goerick. *From Motor Learning to Interaction Learning in Robots*, chapter A Bayesian View on Motor Control and Planning. Springer-Verlag, 2010.