2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

# Policy Search For Learning Robot Control Using Sparse Data

B. Bischoff[1], D. Nguyen-Tuong[1], H. van Hoof[2], A. McHutchon[3],
C. E. Rasmussen[3], A. Knoll[4], J. Peters[2,5], M. P. Deisenroth[6,2]

***Abstract*— In many complex robot applications, such as grasping and manipulation, it is difficult to program desired task solutions beforehand, as robots are within an uncertain and dynamic environment. In such cases, learning tasks from experience can be a useful alternative. To obtain a sound learning and generalization performance, machine learning, especially, reinforcement learning, usually requires sufficient data. However, in cases where only little data is available for learning, due to system constraints and practical issues, reinforcement learning can act suboptimally. In this paper, we investigate how model-based reinforcement learning, in particular the probabilistic inference for learning control method (PILCO), can be tailored to cope with the case of sparse data to speed up learning. The basic idea is to include further prior knowledge into the learning process. As PILCO is built on the probabilistic Gaussian processes framework, additional system knowledge can be incorporated by defining appropriate prior distributions, e.g. a linear mean Gaussian prior. The resulting PILCO formulation remains in closed form and analytically tractable. The proposed approach is evaluated in simulation as well as on a physical robot, the Festo Robotino XT. For the robot evaluation, we employ the approach for learning an object pick-up task. The results show that by including prior knowledge, policy learning can be sped up in presence of sparse data.**

## I. INTRODUCTION

In recent years, robots have increasingly become a natural part of daily life. As service robots, they are introduced to the customer market fulfilling tasks, such as lawn mowing and vacuum cleaning. However, most of today's robot applications are rather simple, where the tasks are pre-programmed. More complex applications, such as grasping and manipulation, are difficult to hard-code beforehand. The main reason is that robots and other autonomous systems are within uncertain, stochastic and dynamic environments, which can not be analytically and explicitly described, making pre-programming impossible in many cases. Here, machine learning offers a powerful alternative.

Especially for robot control manual development of control strategies can be challenging. It is often hard to accurately model the uncertain and dynamic environment while inferring appropriate controllers for solving desired tasks. In such cases, reinforcement learning (RL) can help to efficiently *learn* a control policy from sampled data instead of manual design using expert knowledge [1], [4], [5]. State-of-the-art RL approaches, e.g. policy search techniques [3], usually require sufficiently many data samples for learning an accurate dynamics model, based on which the control policy can be inferred. Thus, in the presence of sparse data RL approaches can be suboptimal, especially, for high-dimensional learning problems. In this paper, we investigate how model-based policy search, in particular the probabilistic inference for learning control method (PILCO), can be tailored to cope with sparse training data to speed up the learning process.

The policy search RL algorithm PILCO [2] employs Gaussian processes (GP) to model the system dynamics. PILCO has been shown to be effective for learning control in an uncertain and dynamic environment [12], [13]. To further speed up learning and to make PILCO appropriate for learning with sparse data in high-dimensional problems, e.g. an 18 dimensional dynamics model, we propose to include further prior knowledge into the learning process. As non-parametric GP regression is employed as the core technique in PILCO, prior knowledge can be straightforwardly incorporated by defining appropriate prior distributions. It has been shown in the past that incorporating prior knowledge into GP learning can significantly improve the learning performance in the presence of sparse data [11]. In this study, we choose a linear mean Gaussian distribution as prior for modeling the system dynamics. This prior is especially appropriate for systems with underlying linear or close to linear behavior. The evaluations show that this choice of prior can help to improve the learning performance while the resulting RL formulation remains in closed form and analytically tractable.

The remainder of the paper will be organized as follows. First, we give a brief review on the basic concepts of RL and PILCO. In Section II, we introduce the extension of PILCO where a linear mean Gaussian prior is employed. In Section III, we provide experiments in simulation as well as on a physical robot. Here, we aim to learn an object pick-up task with the Festo Robotino XT shown in Figure 1, using the proposed approach. A conclusion and summary can be found in Section IV.

### A. Background: Reinforcement Learning

RL considers a learning agent and its interactions with the environment [3], [7]. In each *state* $s \in S$ the agent can apply an *action* $a \in A$ and, subsequently, moves to a new state $s'$. The system *dynamics* define the next state probability $p(s'|s,a)$. In every state, the agent determines the action to

[1]Cognitive Systems, Bosch Corporate Research, Germany
[2]Intelligent Autonomous Systems, TU Darmstadt, Germany
[3]Computational and Biological Learning Lab, Univ. of Cambridge, UK
[4]Robotics and Embedded Systems, TU München, Germany
[5]Max Planck Institute for Intelligent Systems, Tübingen, Germany
[6]Department of Computing, Imperial College London, UK

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement #270327 and the Department of Computing, Imperial College London.

Fig. 1: The Festo Robotino XT, a mobile service robot with a pneumatic arm, used for evaluation.

be used according to its *controller*, $\pi : S \rightarrow A$. Application of the controller for $T$ timesteps results in a state-action trajectory $\{s_0, a_0\}, \{s_1, a_1\}, \ldots, \{s_{T-1}, a_{T-1}\}, s_T$ denoted as *rollout*. Multiple rollouts will not be identical in case of uncertainty and noise. Thus, probability distributions need to be employed to describe the controller rollout. Each state $s_i$ is rated by the environment with a cost function, $c : S \rightarrow \mathbb{R}$. It is the goal of the learning agent to find a controller that minimizes the expected long-term cost $J(\pi)$ given by

$$J(\pi) = \sum_{t=0}^{T} \mathrm{E}_{s_t}\left[c(s_t)\right] \tag{1}$$

where $s_t$ is the resulting state distribution when the controller $\pi$ is applied for $t$ timesteps. The cost function $c$ encodes the learning goal and must be set accordingly. The learning algorithm uses samples $s_i, a_i, s_{i+1}$ to optimize the controller with respect to the expected long-term cost. The way how this experience is used to learn a new, improved controller $\pi$ is what makes the difference between various RL approaches. For example, RL techniques can be classified as *model-free* and *model-based*. While model-free RL directly optimizes the controller $\pi$, model-based approaches explicitly model the dynamics, i.e. $p(s'|s, a)$, and optimize the controller using this model. Another characterization of RL methods is *policy search* versus *value-function* approaches. In policy search, the learning algorithm directly operates on the parameters $\theta$ of a controller $\pi_\theta$ to minimize the expected long-term cost. On the other hand, value-function approaches learn a long-term cost estimate for each state. Using this estimate, a controller can be determined.

### B. PILCO: A Fast Policy Search Approach

To find control parameters $\theta^*$, which minimize Eq. (1), we employ the PILCO policy search framework [2] to learn low-level controllers. PILCO's key component is a probabilistic model of the robot's forward dynamics $p(s'|s, a)$, implemented as a Gaussian process (GP) [6].

Policy learning consists of two steps: policy evaluation and policy improvement. For policy evaluation, we use the GP dynamics model to iteratively compute Gaussian approximations to the long-term predictions $p(s_0|\theta), \ldots, p(s_T|\theta)$ for a given policy parametrization $\theta$. Since PILCO explicitly accounts for model uncertainty in this process, it reduces the effect of model bias [2]. With the predicted state distributions $p(s_t|\theta)$, $t = 1, \ldots, T$, an approximation to the expected long-term cost $J(\pi)$ in Eq. (1) and the gradients $dJ(\theta)/d\theta$ can be computed analytically. For policy improvement, we use this gradient information and apply Quasi-Newton methods for non-convex optimization to find the desired set of policy parameters $\theta^*$. Note that policy learning does not involve any interactions with the robot, but is solely based on the learned GP dynamics model.

When the policy parameters have been learned, the corresponding policy is applied to the robot. The data from this experiment is collected and used to update the learned dynamics model. Then, this new model is used to update the policy. The process of model learning, policy learning, and application of the policy to the robot is repeated until a good policy is found. In this framework, it is essential to obtain a sufficiently accurate dynamics model. If only little data is available, e.g. 200 data points for an 18 dim. model, the model learning performance can be improved when using additional prior system knowledge [11]. As our considered problems appear to have a linear underlying trend, we choose a linear mean prior for the GP employed for learning the forward dynamics. By doing so, the resulting PILCO formulation remains in closed form as shown in the following section.

## II. PILCO WITH LINEAR MEAN GAUSSIAN PRIOR

At the heart of the PILCO algorithm lies the Gaussian process dynamics model. GPs provide a principled and powerful method for modelling observed data by specifying a distribution over possible function values $h(\boldsymbol{x})$, i.e.

$$h(\boldsymbol{x}) \;\sim\; \mathcal{GP}(m(\boldsymbol{x}),\, k(\boldsymbol{x},\, \boldsymbol{x}')) \tag{2}$$

where $m(\boldsymbol{x})$ is the mean function and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the covariance function. For a given data set $\{X, \boldsymbol{y}\}$, a GP prediction at a test point $\boldsymbol{x}_*$ is normally distributed with

$$h_* \;\sim\; \mathcal{N}(\mu_*,\, \Sigma_*)\,, \tag{3}$$

$$\mu_* \;=\; k(\boldsymbol{x}_*,\, X)[K + \sigma_n^2 I]^{-1}\,(\boldsymbol{y} - m(X)) + m(\boldsymbol{x}_*)\,, \tag{4}$$

$$\triangleq\; k(\boldsymbol{x}_*,\, X)\boldsymbol{\beta} + m(\boldsymbol{x}_*)\,, \tag{5}$$

$$\Sigma_* \;=\; k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, X)[K + \sigma_n^2 I]^{-1}\, k(X, \boldsymbol{x}_*) \tag{6}$$

where $K = k(X, X)$, $\boldsymbol{\beta} = [K + \sigma_n^2 I]^{-1}\,(\boldsymbol{y} - m(X))$, $\sigma_n^2$ is a noise variance parameter. Within the GP framework, the GP hyperparameters including the parameters of the mean function $m(\boldsymbol{x})$ can be estimated from data [6]. In case of multiple output dimensions, a separate GP can be trained for each output dimension.

In previous applications of PILCO, the default prior mean function $m(\boldsymbol{x})$ has been set to either zero, $m(\boldsymbol{x}) = 0$, or the identity function $m(\boldsymbol{x}) = \boldsymbol{x}$ [2]. Many systems of interest, however, contain a number of underlying linear, or close to linear, relationships. This is especially true for state spaces containing both variables and their time derivatives

(e.g. positions and velocities) for which a linear relationship (Euler integration) can be a good starting point. By using a more informative linear mean prior, the GP can model the complex nonlinearities better and has improved extrapolation performance. Using a linear mean prior with the GP dynamics model therefore promises performance improvements over previous work. The PILCO algorithm makes predictions using its dynamics model at uncertain test points. In this section, we derive the necessary predictive equations for a GP with a affine mean prior, $m(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$, when the test points are uncertain, i.e. $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. For computational tractability, we choose the squared-exponential covariance function, $k(\boldsymbol{x}, \boldsymbol{x}') = \alpha^2 \exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')^T\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}'))$, with hyperparameters $\alpha^2$ (signal variance) and $\Lambda$ (squared length scales), which are learned within the GP framework.

For an uncertain test input $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, the predictive mean $\mu_*$ is given by

$$\begin{aligned}\mu_* &= \mathrm{E}_{\boldsymbol{x}_*,h}[h(\boldsymbol{x}_*)] = \mathrm{E}_{\boldsymbol{x}_*}[\mathrm{E}_h[h(\boldsymbol{x}_*)]] \\ &= \mathrm{E}_{\boldsymbol{x}_*}[k(\boldsymbol{x}_*, X)\boldsymbol{\beta} + m(\boldsymbol{x}_*)] = \boldsymbol{\beta}^T\boldsymbol{q} + A\boldsymbol{\mu} + b\,, \quad (7)\end{aligned}$$

where $q_i = \eta \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T(\Sigma + \Lambda)^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right)$ with $\eta = \alpha^2 \left|\Sigma\Lambda^{-1} + I\right|^{-1/2}$. The predictive variance $\mathrm{var}_{\boldsymbol{x}_*,h}[h(\boldsymbol{x}_*)]$ is given as

$$\begin{aligned}\mathrm{var}_{\boldsymbol{x}_*,h}[h(\boldsymbol{x}_*)] = \mathrm{E}_{\boldsymbol{x}_*}\left[\mathrm{var}_h[h(\boldsymbol{x}_*)]\right] + \mathrm{E}_{x_*}\left[\mathrm{E}_h[h(\boldsymbol{x}_*)]^2\right] \\ - \mathrm{E}_{x_*}\left[\mathrm{E}_h[h(\boldsymbol{x}_*)]\right]^2 . \quad (8)\end{aligned}$$

Since $\mathrm{var}_h[h(\boldsymbol{x}_*)]$ does not depend on the prior mean function, we use the derivation from [9]. Thus,

$$\mathrm{E}_{\boldsymbol{x}_*}\left[\mathrm{var}_h[h(\boldsymbol{x}_*)]\right] = \alpha^2 - \mathrm{tr}\left((K + \sigma_\epsilon^2 I)^{-1}\tilde{Q}\right) , \quad (9)$$

$$\tilde{Q} = \mathrm{E}_{x_*}\left[k(X, \boldsymbol{x}_*)k(\boldsymbol{x}_*, X)\right] , \quad (10)$$

$$\begin{aligned}\tilde{Q}_{i,j} = &\left|2\Sigma\Lambda^{-1} + I\right|^{-1/2} k(\boldsymbol{x}_i, \boldsymbol{\mu})k(\boldsymbol{x}_j, \boldsymbol{\mu}) \\ &\exp\left((z_{i,j} - \boldsymbol{\mu})^T\left(\Sigma + \tfrac{1}{2}\Lambda\right)^{-1}\Sigma\Lambda^{-1}(z_{i,j} - \boldsymbol{\mu})\right) ,\end{aligned}$$

and $z_{i,j} = \frac{\boldsymbol{x}_i + \boldsymbol{x}_j}{2}$. Using Eq. (7) we get

$$\mathrm{E}_{\boldsymbol{x}_*}\left[\mathrm{E}_h[h(\boldsymbol{x}_*)]\right]^2 = (A\boldsymbol{\mu} + b + \boldsymbol{\beta}^T\boldsymbol{q})^2 \quad (11)$$

which leaves

$$\begin{aligned}\mathrm{E}_{\boldsymbol{x}_*}\left[\mathrm{E}_h[h(\boldsymbol{x}_*)]^2\right] = \mathrm{E}_{\boldsymbol{x}_*}[(k(\boldsymbol{x}_*, X)\boldsymbol{\beta})^2 + m(\boldsymbol{x}_*)^2 \\ + 2m(\boldsymbol{x}_*)k(\boldsymbol{x}_*, X)\boldsymbol{\beta}]. \quad (12)\end{aligned}$$

We derive the expected values on the right hand side of Eq. (12) in a form generalized to multiple output dimensions, since we will reuse the equations for calculating the covariance between output dimensions. Here, the superscripts $u, v$ denote the respective output dimensions. From Eq. (10), we obtain

$$\mathrm{E}_{\boldsymbol{x}_*}[k(\boldsymbol{x}_*, X)\boldsymbol{\beta}^u k(\boldsymbol{x}_*, X)\boldsymbol{\beta}^v] = (\boldsymbol{\beta}^u)^T\tilde{Q}\,\boldsymbol{\beta}^v\,, \quad (13)$$

$$\begin{aligned}\mathrm{E}_{\boldsymbol{x}_*}[m^u(\boldsymbol{x}_*)m^v(\boldsymbol{x}_*)] &= A^u\,\mathrm{E}_{\boldsymbol{x}_*}[\boldsymbol{x}_*\boldsymbol{x}_*^T](A^v)^T \\ &\quad + b^u A^v\boldsymbol{\mu} + b^v A^u\boldsymbol{\mu} + b^u b^v \\ &= A^u\left(\Sigma + \boldsymbol{\mu}\boldsymbol{\mu}^T\right)(A^v)^T + \\ &\quad b^u A^v\boldsymbol{\mu} + b^v A^u\boldsymbol{\mu} + b^u b^v\,. \quad (14)\end{aligned}$$

The final expectation is

$$\begin{aligned}&\mathrm{E}_{\boldsymbol{x}_*}[m^u(\boldsymbol{x}_*)k^v(\boldsymbol{x}_i, \boldsymbol{x}_*)] \\ &\quad = A^u\,\mathrm{E}_{\boldsymbol{x}_*}[\boldsymbol{x}_* k^v(\boldsymbol{x}_i, \boldsymbol{x}_*)] + b^u\,\mathrm{E}_{\boldsymbol{x}_*}[k^v(\boldsymbol{x}_i, \boldsymbol{x}_*)] \\ &\quad = A^k\alpha_v^2(2\pi)^{D/2}|\Lambda_v|^{1/2} \\ &\qquad \cdot \int \boldsymbol{x}_*\mathcal{N}(\boldsymbol{x}_*|\boldsymbol{x}_i, \Lambda_v)\mathcal{N}(\boldsymbol{x}_*|\boldsymbol{\mu}, \Sigma)\mathrm{d}\boldsymbol{x}_* + b^u q_i^v \\ &\quad = A^u q_i^v c_i^v + b^u q_i^v \quad (15)\end{aligned}$$

with

$$\boldsymbol{c}_i = \Lambda(\Sigma + \Lambda)^{-1}\boldsymbol{\mu} + \Sigma(\Sigma + \Lambda)^{-1}\boldsymbol{x}_i\,.$$

Here, we exploited the fact that the squared-exponential kernel can be written as an unnormalized Gaussian distribution. Using the relationship given in Eq. (8), we can now combine equations (9)–(15), for a single output dimension to get

$$\begin{aligned}\mathrm{var}_{\boldsymbol{x}_*,h}[h(\boldsymbol{x}_*)] = \alpha^2 - \mathrm{tr}\left((K + \sigma_\epsilon^2 I)^{-1}\tilde{Q}\right) + \boldsymbol{\beta}\tilde{Q}\boldsymbol{\beta}^T \\ - (\boldsymbol{\beta}^T q)^2 + A\Sigma A^T - 2A\boldsymbol{\mu}\boldsymbol{\beta}^T q \\ + 2A\sum_{i=1}^{n}\beta_i q_i \boldsymbol{c_i}\end{aligned}$$

for the predictive variance. For each output dimension $h^u(\boldsymbol{x})$, we independently train a GP and calculate the mean and variance of the predictive Gaussian distribution as derived above. Next, we consider the predictive covariance between output dimensions $u \neq v$. As a first step, we can write

$$\begin{aligned}\mathrm{cov}_{\boldsymbol{x}_*,h}[h^u(\boldsymbol{x}_*), h^v(\boldsymbol{x}_*)] = \mathrm{E}_{\boldsymbol{x}_*}\left[\mathrm{cov}_h[h^u(\boldsymbol{x}_*), h^v(\boldsymbol{x}_*)]\right] \\ + \mathrm{cov}_{\boldsymbol{x}_*}\left[\mathrm{E}_h[h^u(\boldsymbol{x}_*)], \mathrm{E}_h[h^v(\boldsymbol{x}_*)]\right] . \quad (16)\end{aligned}$$

Due to the conditional independence assumption, i.e. $h^u(\boldsymbol{x}_*) \perp\!\!\!\perp h^v(\boldsymbol{x}_*) \mid \boldsymbol{x}_*$, the first term in Eq. (16) is zero, which leaves the covariance of the means. This can be solved analogously to eq. (12) and (13) to get

$$\begin{aligned}&\mathrm{cov}_{h,\boldsymbol{x}_*}[h^u(\boldsymbol{x}_*), h^v(\boldsymbol{x}_*)] \\ &\quad = (\boldsymbol{\beta}^u)^T Q\boldsymbol{\beta}^v - (\boldsymbol{\beta}^u)^T\boldsymbol{q}^u(\boldsymbol{\beta}^v)^T\boldsymbol{q}^v + A^u\Sigma(A^v)^T \\ &\qquad + A^u\sum_{i=1}^{n}\beta_i^v q_i^v \boldsymbol{c}_i^v + A^v\sum_{i=1}^{n}\beta_i^u q_i^u \boldsymbol{c}_i^u \\ &\qquad - A^u\boldsymbol{\mu}(\boldsymbol{\beta}^v)^T\boldsymbol{q}^v - A^v\boldsymbol{\mu}(\boldsymbol{\beta}^u)^T\boldsymbol{q}^u\end{aligned}$$

for the covariance between output dimensions. Finally, we proceed to derive the input-output covariance between the input $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and the estimated function output $h(\boldsymbol{x}_*) \sim \mathcal{N}(\mu_*, \Sigma_*)$. We need this covariance $\Sigma_{\boldsymbol{x}_*,h_*}$ to derive the joint distribution

$$p(\boldsymbol{x}_*, h(\boldsymbol{x}_*)) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu} \\ \mu_*\end{bmatrix}, \begin{bmatrix}\Sigma & \Sigma_{x_*,h_*} \\ \Sigma_{x_*,h_*}^T & \Sigma_*\end{bmatrix}\right)$$

of the function input $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and output $h(\boldsymbol{x}_*)$. It is

$$\Sigma_{x_*,h_*} = \mathrm{E}_{x_*,h}\left[\boldsymbol{x}_* h(\boldsymbol{x}_*)^T\right] - \mathrm{E}_{x_*}\left[\boldsymbol{x}_*\right]\mathrm{E}_{x_*,h}\left[h(\boldsymbol{x}_*)\right]^T ,$$

where the second term is directly given by Eq. (7). To compute $\mathrm{E}_{x_*,h}\left[\boldsymbol{x}_* h(\boldsymbol{x}_*)^T\right]$, we first rewrite the expression

$$\begin{aligned}\mathrm{E}_{x_*,h}\left[\boldsymbol{x}_* h(\boldsymbol{x}_*)^T\right] &= \mathrm{E}_{x_*}\left[\boldsymbol{x}_* m(\boldsymbol{x}_*) + \boldsymbol{x}_* k(\boldsymbol{x}_*, X)\boldsymbol{\beta}\right] \\ &= \mathrm{E}_{x_*}\left[\boldsymbol{x}_*\boldsymbol{x}_*^T\right]A^T + b\boldsymbol{\mu} + \mathrm{E}_{x_*}\left[\boldsymbol{x}_* k(\boldsymbol{x}_*, X)\boldsymbol{\beta}\right] .\end{aligned}$$

Now, we use again the definition of the variance, Eq. (15) and the simplification steps described in [14, eq. (2.68)–(2.70)] to obtain

$$\Sigma_{x_*,h_*} = \sum_{i=1}^{n} \beta_i q_i \Sigma(\Sigma + \Lambda)^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}) + \Sigma A^T$$

for the input-output covariance. This concludes the derivations for the GP prediction with a linear prior mean function at uncertain test inputs.

Note that all terms for the prediction with linear mean prior can be split in the terms we have in the zero mean case, plus some additional expressions. Using PILCO with linear mean prior, the complexity to compute the additional terms for the mean and input-output covariance is $\mathcal{O}(DE)$ resp. $\mathcal{O}\left(D^2E\right)$ and, hence, independent of the number of training samples $n$. Here, $D$ denotes the input dimensionality (i.e. state plus action dimensionality) while $E$ corresponds to the number of output dimensions (i.e. state dimensionality). Assuming $n > D, E$, variance and covariance prediction can be performed in $\mathcal{O}\left(DE^2 + nD^2\right)$, where $n$ is the number of training samples. Compared to $\mathcal{O}\left(DE^2n^2\right)$ for the zero prior mean prediction (see [14]), the additional terms for the linear mean prior do not increase the complexity class.

## III. Experiments & Evaluations

In this section, we evaluate the proposed approach for learning control policies in the sparse data setting. First, we apply our extended PILCO algorithm on a control task in simulation, namely position control of a throttle valve. Subsequently, we learn an object grasping policy for a pick-up task with the Festo Robotino XT, a mobile service robot with pneumatic arm shown in Figure 1. In both cases, sparse data is employed for learning the system dynamics, e.g. 75 sampled data points for learning throttle valve control and 200 data points for learning pick-up task.

### A. Comparative Evaluation on a Throttle Valve Simulation

The throttle valve shown in Figure 3 is an important technical device that allows flow regulation of gas and fluids. It is widely used in various industrial applications, such as cooling systems for power plants and pressure control in gasoline combustion engines. The valve system basically consists of a DC-motor, a spring and a valve with position sensors. The dynamics of the throttle valve system can be analytically approximated by the model

Fig. 3: Throttle valve system to regulate a gas or fluid flow.

$$\begin{bmatrix}\dot{\alpha}(t)\\ \dot{\omega}(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ -K_s & -K_d\end{bmatrix}\begin{bmatrix}\alpha(t)\\ \omega(t)\end{bmatrix} + \begin{bmatrix}0\\ L(t)\end{bmatrix} + \begin{bmatrix}0\\ T(t)\end{bmatrix}, \quad (17)$$

where $\alpha$ and $\omega$ are the valve angle and corresponding angular velocity $T$ is the actuator input and $L(t) = C_s - K_f \operatorname{sgn}(\omega(t))$ [8]. The parameters $K_s$, $K_d$, $K_f$ and $C_s$ are dynamics parameters and need to be identified for a given system. Here, we use $K_s = 1.5$, $K_d = 3$, $K_f = 6$, $C_s = 20$. The input $T$ is determined as $T(t) = u(t)K_j/(R)$, where $u(t) \in [-20, 20]$ is the input voltage with $K_j = 50$ and $R = 4$.

In the RL setting, we describe the system state as valve angle $\alpha$ and velocity $\omega$, the voltage $u(t)$ corresponds to the action space. The learning goal is to move the valve to a desired angle $g$. Hence, the cost function can be defined as saturated immediate cost $c(s) = 1 - \exp(-(s-g)^T L(s-g))$ with diagonal width matrix $L$ [2]. We set the start state to be $\alpha_0 = 10°$, the desired angle is $g = 90°$. In each controller rollout, 15 dynamics samples $s, a, s'$ are collected. This results in a sparse data set of 75 samples after initial random movements and 4 learning episodes. The controller structure employed for policy search is a radial basis function (RBF) network, which will be described in Section III-B in more detail. Figure 2 shows the learning results of PILCO with and without usage of a linear mean Gaussian prior, as introduced in last section. As can be seen from the results, the proposed approach using a linear mean prior converges faster and more robustly in this control task.

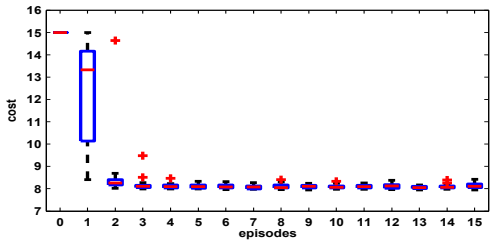
### B. Learning a Pick-Up Task with a Festo Robotino XT

In this section, we aim to solve an object pick-up task using the proposed method. Here, a mobile robot, namely a Festo Robotino XT shown in Figure 1, is controlled to approach the target object, e.g. a mug, to grasp this object and, subsequently, deliver it to the user. In the next sections, we describe the Festo Robotino XT and the pick-up task first. In Section III-B.2, learning results of the task are provided and discussed in detail.

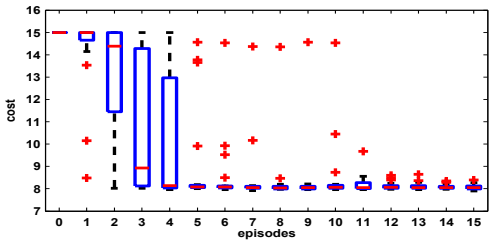
*1) Object Grasping with Robotino:* The Festo Robotino XT shown in Figure 1, is a mobile service robot with an omni-directional drive and an attached pneumatic arm. The design of the arm is inspired biologically by the trunk of an elephant. The arm itself consists of two segments with three bellows each. The pressure in each bellow can be regulated in a range of 0.0 to 1.5 bar to move the arm. As material, a polyamide structure is employed resulting in a low arm weight. A soft gripper is mounted as end-effector implementing the so-called Fin-Ray-effect. Under pressure at a surface point, the gripper does not deviate but bends towards that point and, thus, allows a form-closed and stable grasp.

The robot is also well-suited to operate in human environments, as the low pressure pneumatics allows a compliant behavior. However, kinematics and dynamics of the pneumatic arm are quite complex. A further constraint of the Robotino is the limited work space of the arm. Due to this limited workspace, we need to control the robot base as well as the arm movement while grasping an object. Thus, the forward dynamics for the grasping control results in an 18 dimensional model. These dimensions correspond to the robot state, i.e. position and orientation of the base and end-effector, and robot action space, i.e. base movements and change in arm pressures.

The application we consider for the Robotino are pick-up tasks. This task can be partitioned into three phases: the robot approaches the object, grasps the object and, subsequently,

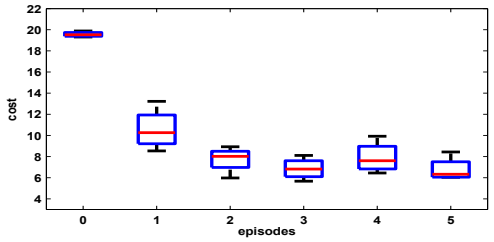



(a) PILCO with linear mean prior

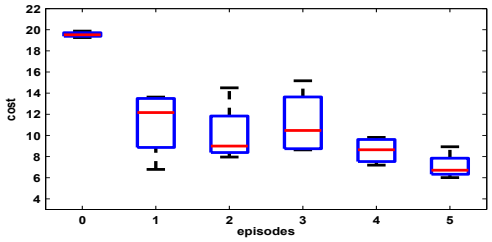



(b) PILCO with zero mean prior

Fig. 2: We performed 20 independent learning runs of PILCO with and without linear mean prior for the task of position control on a simulated throttle valve system. As can be seen, our proposed method with linear mean prior in (a) converges faster and more robust compared to PILCO with zero mean prior (b). Here, the boxes indicate the respective mean and percentiles, while red crosses represent runs interpreted as outliers.




(a) PILCO with linear mean prior




(b) PILCO with zero mean prior

Fig. 4: The figure shows learning results for the task of object grasping. We performed 4 independent learning runs, the plots show the mean and percentiles of the cost over episodes. Our approach with linear mean prior is shown in (a), while (b) shows the learning results of Pilco with zero mean prior.

delivers the object to the user. While standard navigation approaches can be employed to approach the object and to deliver the object (see e.g. [10]), it is challenging to grasp an object. While grasping an object, such as a mug on the floor, the aim is to learn a control policy bringing the robot, i.e. base and end-effector, to a proper posture where the grasp can be conducted. Here, an external optical tracking system is used to track the mobile robot and the object, for which markers are appropriately attached as shown in Figure 1. At the surface of the arm, six strings are attached. The robot's end-effector pose is estimated with a learned kinematics model using the extension of these strings as input.

*2) Learning Object Grasping:* To learn object grasping in the policy search setting, we need to define states, actions, cost, and a control structure. For the Robotino XT, the state consists of the position $x, y$ and orientation $\theta$ of the base, as well as 6D pose of the end-effector, i.e. position $x, y, z$ and orientation roll, yaw and pitch. The action space is also 9 dimensional, $A \subset \mathbb{R}^9$, with 3 dimensions for the base movement and 6 dimensions to modify the bellow pressure in a range of $-0.3$ bar to 0.3 bar. The dynamics $f$, $f : S \times A \rightarrow S$, is hence an 18 to 9 dimensional mapping. The robot is tracked at a rate of 2 Hz resulting in 120 dynamics samples per minute. The training data, e.g. 200 samples in an 18 dimensional space after 4 episodes, is consequently sparse.

Next, we need to find an appropriate cost function for object grasping. Similar to the task of throttle valve control in Section III-A, we use the saturated immediate cost function with width $L$, $c(s) = 1 - \exp(-(s-g)^T L(s-g))$, that rewards small distances of the end-effector to the desired grasping pose $g$. Here, we use a sum of two saturated cost functions—a wide cost to achieve robust learning and a peaked cost for high grasping accuracy.

Finally, application of policy search requires the definition of a parametrized control structure. As for learning throttle valve control, we will again employ a radial basis function network (RBF) with Gaussian shaped basis function. The control structure can be written as $\pi(s) = \sum_{i=0}^{N} w_i \phi_i(s)$, with $\phi_i(s) = \sigma_f^2 \exp(-\frac{1}{2}(s-s_i)^T \Lambda (s-s_i))$, weight vector $w$, and $S = \{s_0, \ldots, s_N\}$ the set of support points. The weight vector, the support points as well as $\Lambda = \mathrm{diag}(l_1^2, \ldots, l_D^2)^{-1}$ and $\sigma_f$ represent the open control parameters. The RBF control structure is flexible and, thus, applicable to a wide range of systems.

Now, we can employ PILCO as introduced in Section I-B. The initial robot pose has a displacement of approximately 50 cm to the mug. We start with application of random actions to generate initial dynamics data. Here, we collect four random trajectories of 15 seconds starting from the initial pose. This results in an initial dynamics data set of 120 samples. Now, we iterate the steps: *learn dynamics model*, *improve controller*, *apply controller* to collect additional data. In each rollout, the controller is applied for 10 seconds resulting in 20 additional dynamics samples. We repeat the learning process four times, the learning results with our proposed method and the previous PILCO approach are shown in Figure 4. It can be seen that learning with the proposed method converges after only 3 episodes, corresponding to 90 seconds system interaction or 180 dynamics samples, and outperforms the previous approach on this task.

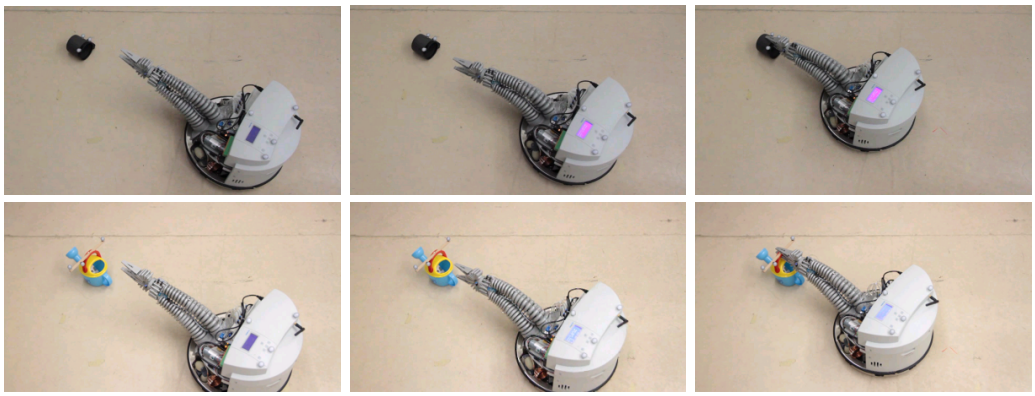


Fig. 5: The top pictures show the application of the final controller obtained after 5 episodes. The agent successfully grasps the mug. The learning result can be generalized to a new object, e.g. a watering can, with only 2 additional episodes. The rollout to grasp the handle of the can is shown in the pictures at the second row.

Fig. 6: The Figure shows our approach to solve pick-up tasks using the Robotino XT. First, the robot approaches the object it wants to grasp—in this case, a mug on the floor. Second, the learned policy for object grasping is employed. Subsequently, the robot delivers the object to the user, as shown in the most right picture.

A successful grasp of a learned controller is shown in the upper part of Figure 5. Starting from a controller learned for mug grasping, we can generalize the learning result for grasping of new objects. Here, we reuse the previously learned dynamics model and policy. After two additional learning episodes—using an adapted cost function with new desired grasping pose $g$—grasping a watering can is performed successfully as shown at the second row of Figure 5.

Finally, we use the learned controller to solve pick-up tasks. Figure 6 shows an exemplary solution of a pick-up task. The Robotino approaches the target object, i.e. a mug on the floor, grasps it using the learned policy, and, finally, delivers it to the user. A video showing the learning process with PILCO as well as an exemplary pick-up task is attached as supplementary material.

## IV. CONCLUSIONS

In this paper, we propose an extension of PILCO to policy learning in presence of sparse data. As PILCO is built on GP, the basic idea is to incorporate additional system knowledge into the learning process as a mean function of the GP prior. Here, we choose a linear mean Gaussian prior for modeling the system dynamics. This choice is especially appropriate when the system is linear or having an underlying linear trend. By doing so, the resulting formulation of PILCO remains analytically tractable and can be given in closed-form. Experimental results in simulation and on a real robot show that the proposed approach can help to speed up learning in the presence of sparse data. We successfully employ the approach to learn a control policy to solve an object pick-up task for the mobile robot platform Robotino XT.

## REFERENCES


[1] J. Kober and J. A. Bagnell and J. Peters. *Reinforcement learning in robotics: A survey*. International Journal Of Robotics Research, 2013.

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

[3] M. P. Deisenroth, G. Neumann and J. Peters. *A Survey on Policy Search for Robotics*. Vol. 2 of Frontiers and Trends in Robotics, 2013.

[4] B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber. *Quasi-online Reinforcement Learning for Robots*. International Conference on Robotics and Automation, 2006.

[5] A. Y. Ng and H. J. Kim and M. I. Jordan and S. Sastry. *Inverted autonomous helicopter flight via reinforcement learning*. International Symposium on Experimental Robotics, 2004.

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

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA, 1998.

[8] P. G. Griffiths. *Embedded Software Control Design for an Electronic Throttle Body*. Master's Thesis, Berkeley, California, 2000.

[9] M. P. Deisenroth, D. Fox, C. E. Rasmussen. *Gaussian Processes for Data-Efficient Learning in Robotics and Control*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2014.

[10] S. Thrun, W. Burgard and D. Fox. *Probabilistic Robotics: Intelligent Robotics and Autonomous Agents*. The MIT Press, 2005.

[11] D. Nguyen-Tuong and J. Peters. *Using Model Knowledge for Learning Inverse Dynamics*. International Conference on Robotics and Automation, 2010.

[12] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert and A. Knoll. *Learning Throttle Valve Control Using Policy Search*. 23rd European Conference on Machine Learning, 2013.

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

[14] M. P. Deisenroth *Efficient reinforcement learning using Gaussian processes*. PhD thesis, Karlsruhe Institute of Technology, 2010.