An easy introduction to PILCO: A Review on Data-Efficient Learning for Physical Systems using Gaussian Processes

Preprint · May 2021

CITATIONS
0

READS
2,181

1 author:

Paul Brunzema
RWTH Aachen University
4 PUBLICATIONS **14** CITATIONS

SEE PROFILE

Review on Data-Efficient Learning for Physical Systems using Gaussian Processes

by Paul Brunzema
paul.brunzema@rwth-aachen.de
RWTH Aachen University
Aachen, Germany

***Abstract*—The use of reinforcement learning (RL) in robotics and control has been a promising area of research in recent years, enabling autonomous task learning. While RL algorithms show remarkable results in simulations, the number of iterations needed for learning tasks is not applicable to physical systems due to wear and tear. Therefore, algorithms which efficiently exploit each iteration of learning are needed to enable the use of RL for physical systems.**

As a part of the seminar *Learning-based Control* a contribution on data-efficient learning in robotics and control using a probabilistic approach with Gaussian processes (GPs) to model system dynamics in [4] is presented. This paper will cover its data-efficient learning algorithm PILCO, briefly talk about the required fundamentals such as GPs and RL, discuss developments on and modifications to PILCO, and state remaining challenges and open areas of research.

***Index Terms*—Gaussian processes, reinforcement learning, control, optimal control, robotics**

I. Introduction

RL in robotics and control shows promise for reducing the engineering effort required to adapt systems to new environments or tasks. Newly acquired data from the environment during a process could be used to steadily improve performance without human intervention. While RL can be divided into model-free and model-based methods, the number of iterations required to learn a task in model-free RL approaches such as Q-learning [14] makes it impractical to apply to physical systems, as it can be very time-consuming and thus expensive. Also the wear and tear of physical systems has to be taken into account. [1] has shown that model-based approaches to RL are more promising regarding data-efficiency and are therefore more suitable for RL in robotics and control.

Modeling the dynamics of a system using a deterministic model in the presence of large amounts of data can be useful and is often done in system identification. However, given only a limited amount of available data, the use of a deterministic model can have severe consequences. Looking at Fig. 1(b) with only a small set of observations, each gray curve represents a possible deterministic model passing through the observations. If a deterministic model is evaluated at a point that extrapolates the training data it will always return a function value in full confidence, assuming perfect model knowledge. In contrast, a GP can provide both the level of uncertainty (shaded red, Fig. 1) at that point and the most likely function value (red, Fig. 1).

By choosing this nonparametric probabilistic approach of modeling the dynamics with a GP, uncertainty can be explicitly accounted for in learning the model and evaluating a policy. Therefore, it is the basis of PILCO (Probabilistic Inference for Learning Control) [4][5][6], a data-efficient model-based policy search algorithm using GPs. Through approximate inference techniques for policy evaluation, PILCO is able to compute analytic policy gradients for policy improvement which allows the use of state-of-the-art gradient-based optimizers and results in a high learning speed for systems with low dimensions.

This paper will be structured as follows: After covering the necessary background for understanding PILCO and discussing related work, the algorithm is presented with its three main parts *model-learning*, *policy evaluation*, and *policy improvement*. The application of the algorithm to a physical systems is presented and finally modifications and further developments, as well as limitations of the algorithm are discussed.

II. Background

To introduce and understand PILCO, this chapter presents and discusses the basic concepts needed, such as RL and GPs. For a deeper introduction to the topics, it refers to further literature.

A. Introduction to Reinforcement Learning

RL is a subfield of machine learning in which an agent learns a task through interaction with its environment. The agent receives a reward for completing the task or for a specific actions depending on the problem set up. Through RL a machine can be taught to play games like chess or Go. But RL can also be applied to physical systems such as robots.

Considering such a physical system, its dynamics model with an unknown transition function f can be written as

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}, \quad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma_w^2 \mathbf{I}) \tag{1}$$

with $\mathbf{x}_t \in \mathbb{R}^D$ corresponding to the perceived states at time step t, Gaussian noise $\mathbf{w}$, and control inputs $\mathbf{u}_t \in \mathbb{R}^F$. The control inputs depend on a policy $\pi : \mathbb{R}^D \rightarrow \mathbb{R}^F$ which is used to map the perceived state to an action taken by the agent. This is formulated as $\mathbf{u}_t = \pi(\mathbf{x}_t, \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ as policy parameters.

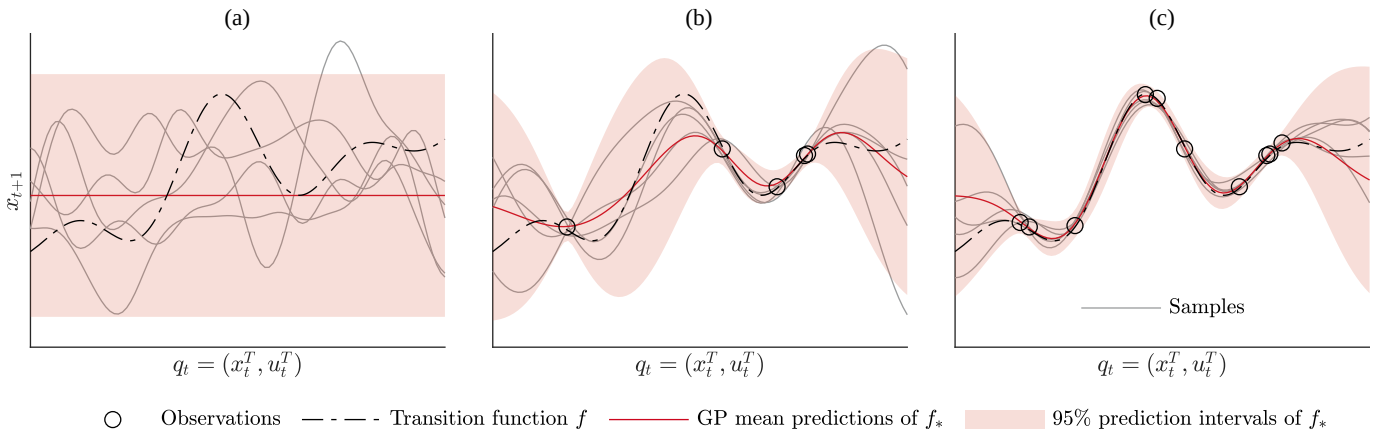

Fig. 1: Panel (a) shows the GP prior distribution using a squared-exponential covariance function (4). In panels (b) and (c) the prior is conditioned using five and ten random observations from the dynamics (1), respectively. In gray random samples drawn from the posterior distribution are shown, each representing a valid *deterministic* function given the observations. It can be seen that the variance of the posterior is small in regions with available observations, while in *unobserved territories* the variance is larger and the mean prediction approaches the mean of the prior distribution. The hyperparameters for the conditioned GPs are $l, \sigma_f^2, \sigma_w^2 = [2.7,\ 0.6^2,\ 0.005^2]$.

The RL goal in this setting is then to find optimal parameters $\boldsymbol{\theta}^*$ resulting in an optimal policy π^* which minimizes the long-term expected cost as

$$\min_{\boldsymbol{\theta}} J^{\pi}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \mathbb{E}[c(\mathbf{x}_t)|\boldsymbol{\theta})] \tag{2}$$

with $c(\mathbf{x}_t)$ corresponding to the cost of being in state $\mathbf{x}$ at time t. On a high level, the RL approach to finding an optimal policy can be compared to finding an optimal controller in optimal control. However, the main difference between the two approaches is that in the optimal control setting the transition function f in (1) is known, whereas in the RL approach it is not.

To find the minimum to the cost function $J^{\pi}(\boldsymbol{\theta})$, policy gradient methods can be used obtaining a gradient $\nabla_{\boldsymbol{\theta}} J^{\pi}(\boldsymbol{\theta})$ to update the current parameters $\boldsymbol{\theta}$. However, classic policy gradient methods used in model-free RL approaches require the changed policy to be evaluated in the environment to estimate its effect on the cost function. While this is a valid approach for simulations, the sheer amount of trial needed to find an optimal solution makes model-free approaches not applicable to physical systems. In contrast, the PILCO algorithm is a policy search method based on analytic gradients which utilizes a learned dynamics model to evaluate sub-optimal policies resulting in fewer conducted experiments with the system.

A big challenge in reinforcement learning is the trade-off between *exploration* and *exploitation*. While an agent needs to *exploit* the already known reward space – finding the local optimum – it also has to *explore* new options which might lead to a better selection of actions thus resulting in a higher long-term reward – exploring the global reward space for different maxima. For a deeper introduction to RL please refer to [14].

B. Introduction to Gaussian Processes

This section will give an introduction to GPs as discussed in [12]. A GP in a regression setting can be interpreted as a probability distribution over plausible functions and is fully defined by a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$. The covariance function defines the prior distribution given n^* test points $\mathbf{X}$ to

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}, \mathbf{X})) \tag{3}$$

with $K(\mathbf{X}, \mathbf{X})$ denoting the $n^* \times n^*$ matrix of the covariance function evaluated at all pairs of test points. This prior distribution is displayed in Fig. 1(a). Using a squared-exponential (SE) covariance function

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x - x')^2\right) \tag{4}$$

with the signal variance σ_f^2 and characteristic length scale l as hyperparameters, a GP can be used as a universal function approximator in regression.

Considering now a dynamic system as in (1), a GP can be used to model the unknown transition function f as $\mathbf{f}_*$. As training input $(\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{D+F}$ is chosen. The state and control vectors are merged to $\mathbf{q}_t := [\mathbf{x}_t^T, \mathbf{u}_t^T]^T$. Therefore, n training inputs are represented by $\tilde{\mathbf{Q}} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$. As training target $\mathbf{x}_{t+1} \in \mathbb{R}^D$ is chosen and represented in $\tilde{\mathbf{y}} = [\mathbf{x}_2, \ldots, \mathbf{x}_{n+1}]$. Combining the training targets and modeled function $\mathbf{f}_*$ to a multivariate Gaussian distribution assuming a prior mean of $\mathbf{0}$ yields in

$$\begin{bmatrix} \tilde{\mathbf{y}} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\tilde{\mathbf{Q}}, \tilde{\mathbf{Q}}) + \sigma_w^2 \mathbf{I} & K(\tilde{\mathbf{Q}}, \mathbf{Q}_*) \\ K(\mathbf{Q}_*, \tilde{\mathbf{Q}}) & K(\mathbf{Q}_*, \mathbf{Q}_*) \end{bmatrix}\right) \tag{5}$$

with n^* test points of interest as $\mathbf{Q}_* = [\mathbf{q}_1^*, \ldots, \mathbf{q}_{n^*}^*]$and σ_w^2 as variance of the noise in (1).

Conditioning the multivariate Gaussian distribution (5) to the training inputs $\tilde{\mathbf{Q}}$, training target $\tilde{\mathbf{y}}$ and the test points of interest $\mathbf{Q}_*$ yields in

$$\mathbf{f}_*|\mathbf{Q}_*, \tilde{\mathbf{y}}, \tilde{\mathbf{Q}} \sim \mathcal{N}(\bar{\mathbf{f}}_*, \mathrm{cov}[\mathbf{f}_*]) \tag{6}$$

with

$$\bar{\mathbf{f}}_* = K_{\mathbf{Q}_*\tilde{\mathbf{Q}}}[K_{\tilde{\mathbf{Q}}\tilde{\mathbf{Q}}} + \sigma_w^2\mathbf{I}]^{-1}\tilde{\mathbf{y}} \tag{7}$$

$$\mathrm{cov}[\mathbf{f}_*] = K_{\mathbf{Q}_*\mathbf{Q}_*} - K_{\mathbf{Q}_*\tilde{\mathbf{Q}}}[K_{\tilde{\mathbf{Q}}\tilde{\mathbf{Q}}} + \sigma_w^2\mathbf{I}]^{-1}K_{\tilde{\mathbf{Q}}\mathbf{Q}_*} \tag{8}$$

as the mean and covariance, respectively, resulting in a posterior probability distribution over plausible functions given the training data. In Fig. 1(b) and Fig. 1(c) the conditioning to training data is visualized.

The GP can provide a prediction corresponding to the mean $\bar{\mathbf{f}}_*$ as the *most likely* estimate for the test data as well as the *confidence* in the prediction in $\mathrm{cov}[\mathbf{f}_*]$. Taking this *confidence* measure into account is crucial for the exploratory behavior of the PILCO algorithm at the beginning of training [5][8].

However, the computational effort for a classical GP discussed in this paper scales with cubic complexity in training points n due to the inversion of $K_{\tilde{\mathbf{Q}}\tilde{\mathbf{Q}}}$ [10] (see (7) and (8)). Scalable GP methods have been proposed but are not further discussed, instead at this point it is referred to [10].

III. Related Work

MuJoCo [15] is a physics engine which allows for the validation of RL algorithms and their performance through simulations of complex tasks such as a walking robot, or easier tasks such as the inverted pendulum. This enables a direct comparison of algorithms on benchmark problems. While deep RL methods show promising performances [7], they can not yet be applied to physical systems due to the sheer amount of iterations needed. In physical systems, the number of iterations must be kept to a minimum to reduce wear and tear off the machines.

Therefore, data-efficient RL methods are needed. Data-efficiency can be achieved by incorporating domain knowledge. For example, successful trails to the task given different initial conditions can be presented to the agent through demonstration to *kick-start* the learning [13]. Assuming expert knowledge is either expensive or unavailable, another way of achieving data-efficiency is by increasing the *information extraction*. Comparing model-free and model-based RL approaches, [1] and [9] have shown significant advantages regarding data-efficiency for model-based RL. Crucially, in a model-based approach, an optimal policy does not have to be applied directly on the system to obtain information about its impact on the cost function (2). Instead, the model of the environment can be used to estimate the impact of the policy, resulting in data-efficiency.

To extract important features from data, nonparametric regression models, such as GPs, have shown promise. GPs for RL in the context of dynamical systems were proposed in [3]. However, in [3], GPs are used to model a value function from which policies are derived. In contrast, PILCO, as a policy search method, does not require an explicit value function. It uses a GP to learn the dynamics of the system and thus is able to search directly in the policy space through analytic gradients [4].

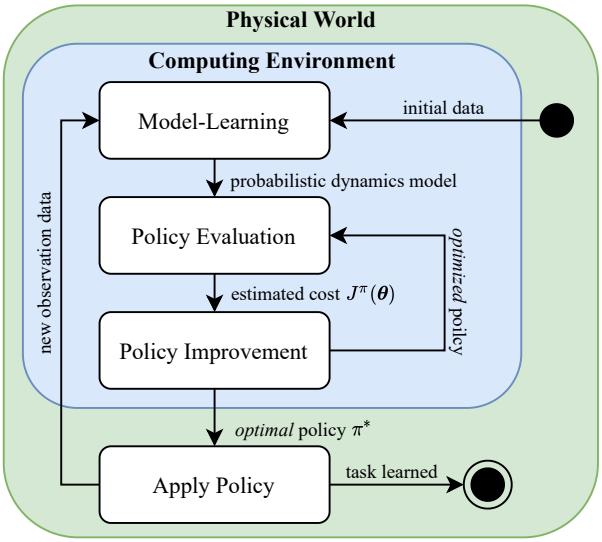

Fig. 2: High-level flowchart of the PILCO algorithm dividing the algorithm into two parts. One that is executed in a computing environment (blue) and another that is executed in the physical world (green). By only applying the *optimal* policy on the system in the physical world, the data-efficiency in terms of executed trials and required interaction time can be achieved.

IV. Methodology

Considering the dynamic system of (1), the PILCO algorithm aims to find the optimal parameters $\boldsymbol{\theta}^*$ to the optimal policy π^* which minimizes the expected long-term cost (2) in a model-based RL approach. The algorithm can be divided into three main steps.

1) *Learning a GP model* of (1) given the acquired observation data
2) Long-term *policy evaluation* to calculate the expected long-term cost $J^\pi(\boldsymbol{\theta})$
3) *Improving the policy* by analytically computing the gradient of the long-term cost with respect to the policy parameters $\boldsymbol{\theta}$.

The algorithm is displayed in a flowchart in Fig. 2 dividing it into parts which are executed in an computing environment or the physical world as well as in pseudo code in Alg. 1 for a more detailed overview. The three main steps are discussed further in the following sections.

A. Model-Learning

The GP model of the system dynamics is learned according to Sec. II-B with n training input as $\tilde{\mathbf{Q}} = [\mathbf{q_1}, \ldots, \mathbf{q_n}]$. However, instead of choosing $\mathbf{x}_{t+1}$ as training target, differences $\boldsymbol{\Delta}_t = \mathbf{x}_{t+1} - \mathbf{x}_t \in \mathbb{R}^D$ are chosen and represented as $\tilde{\mathbf{y}} = [\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_n]$.

The posterior probability distribution of the state $\mathbf{x}_{t+1}$ according to (6) can be formulated as

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \quad (9)$$

with

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \mathbb{E}_f[\boldsymbol{\Delta}_t] \quad (10)$$

$$\boldsymbol{\Sigma}_{t+1} = \mathrm{var}_f[\boldsymbol{\Delta}_t], \quad (11)$$

with the expected value $\mathbb{E}_f[\boldsymbol{\Delta}_t]$ and variance $\mathrm{var}_f[\boldsymbol{\Delta}_t]$ calculated according to (7) and (8), respectively, given the training target and a prior mean function $m \equiv 0$. Note that by choosing differences as training target the mean of the posterior state distribution in (9) does not go back to $\mathbf{0}$ in *unobserved territory* (as in Fig. 1), but to $\boldsymbol{\mu}_t$ (see 10).

As covariance function $k(\cdot, \cdot)$ a multivariate SE-kernel similar to (4) is used in [4], whose hyperparameters are learned in each learning iteration using likelihood maximization [12].

B. Policy Evaluation

During policy evaluation, the state evolution given a policy $p(\mathbf{x}_1|\pi), \ldots, p(\mathbf{x}_T|\pi)$ is computed to calculate the long-term cost $J^\pi(\boldsymbol{\theta})$. This requires mapping uncertain test inputs $p(\mathbf{q}_t)$ through the GP. To obtain the successor state, the change in state $\boldsymbol{\Delta}_t$ has to be computed. Assuming the test input is a joint Gaussian distribution $p(\mathbf{q}_t) = \mathcal{N}(\mathbf{q}_t|\boldsymbol{\mu}_{q,t}, \boldsymbol{\Sigma}_{q,t})$, the change in state can be expressed as

$$p(\boldsymbol{\Delta}_t) = \int\int p(f(\mathbf{q}_t)|\mathbf{q}_t)p(\mathbf{q}_t)\,\mathrm{d}f\,\mathrm{d}\mathbf{q}_t. \quad (12)$$

Solving this integral analytically is not traceable. Therefore, it is approximated by a Gaussian distribution for $p(\boldsymbol{\Delta}_t)$ using *moment matching*, see Fig. 3, by calculating the mean $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ and covariance $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ of (12) and assuming the distribution to be Gaussian. The desired predictive distribution of the successor state in (9) can then be approximated with

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_{\boldsymbol{\Delta}} \quad (13)$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{\boldsymbol{\Delta}} + \mathrm{cov}[\mathbf{x}_t, \boldsymbol{\Delta}_t] + \mathrm{cov}[\boldsymbol{\Delta}_t, \mathbf{x}_t]. \quad (14)$$

Algorithm 1 High-level PILCO algorithm [4]

Initial: Initialize parameters $\boldsymbol{\theta} \in \mathcal{N}(\mathbf{0}, \mathbf{I})$.
Apply random control signals for initial data.

1: **repeat**
2: *Learn GP model* using all recorded data.
3: **repeat**
4: Approximate inference for *policy evaluation* to get the expected long-term cost $J^\pi(\boldsymbol{\theta})$.
5: *Policy improvement:*
(a) Get analytic gradient $\nabla_{\boldsymbol{\theta}} J^\pi(\boldsymbol{\theta})$.
(b) Update parameters $\boldsymbol{\theta}$ by applying gradient-based optimization method (e.g. CG).
6: **until** *convergence*; **return** $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
7: $\pi^* \leftarrow \pi(\boldsymbol{\theta}^*)$.
8: Apply π^* on system and record data.
9: **until** *task learned*

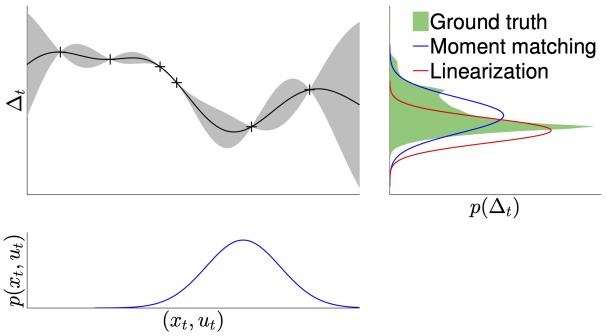

Fig. 3: *Figure form [4].* The uncertain, Gaussian-assumed, input distribution $p(\mathbf{q}_t) = p(\mathbf{x}_t, \mathbf{u}_t)$ is propagated through the GP model. The resulting distribution on $p(\boldsymbol{\Delta}_t)$ (shaded) is approximated using *moment matching* (blue) and *linearization* (red).

The process of calculating the successor state is summarized in Alg. 2.

While other approximation methods such as *linearization* are plausible, [4] and [8] discovered that using *moment matching* results in faster learning and is crucial for the exploratory behavior of the PILCO algorithm. Using *linearization* reduces the computational cost compared to *moment matching*, however, it tends to underestimate the predictive variances (Fig. 3) and therefore gets stuck in local minima.

Finally, to calculate the expected long-term cost $J^\pi(\boldsymbol{\theta})$, only the expected cost at each state in the state evolution is left to be computed as

$$\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)|\boldsymbol{\theta}] = \int c(\mathbf{x}_t)\mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\,\mathrm{d}\mathbf{x}_t \quad (15)$$

which can be done analytically by choosing a suitable cost function c.

Using the learned probabilistic dynamics model for policy evaluation is the key to the data-efficiency of PILCO. Instead of determining the influence of a policy on the cost function in an experiment by applying it to the system, the policy can first be evaluated using the GP dynamics model by predicting the state evolution. How a policy can be improved given the policy evaluation with the goal of only applying optimal policies to the system will be covered in the next section.

Algorithm 2 Computing the successor state distribution for policy evaluation [4]

Initial: $\mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$

1: Control distribution $p(\mathbf{u}_t) = p(\pi(\mathbf{x}_t, \boldsymbol{\theta}))$
2: Joint state-control distribution $p(\mathbf{q}_t) = p(\mathbf{x}_t, \mathbf{u}_t)$
3: Predictive GP distribution of change in state $p(\boldsymbol{\Delta}_t)$ using *moment matching*
4: Get distribution of successor state $p(\mathbf{x}_{t+1})$

C. Policy Improvement

To obtain analytical gradients of the cost function $J^\pi(\boldsymbol{\theta})$ a few assumptions have to be satisfied. On the one hand the expected cost in (15) has to be differentiable with respect to the moments of the state distribution and on the other hand the moments of the control distribution have to be differentiable with respect to $\boldsymbol{\theta}$ and can be computed analytically.

In the following $\mathcal{E}_t := \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$ is introduced as notation to reduce complexity. Applying the chain rule results in

$$\nabla_{\boldsymbol{\theta}} J^\pi(\boldsymbol{\theta}) = \frac{\mathrm{d}J^\pi(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}}, \tag{16}$$

$$\frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(\mathbf{x}_t)} \frac{\mathrm{d}p(\mathbf{x}_t)}{\mathrm{d}\boldsymbol{\theta}} := \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\mu}_t} \frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\Sigma}_t} \frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}} \tag{17}$$

with $\mathrm{d}\mathcal{E}_t/\mathrm{d}p(\mathbf{x}_t) = \{\mathrm{d}\mathcal{E}_t/\mathrm{d}\boldsymbol{\mu}_t, \mathrm{d}\mathcal{E}_t/\mathrm{d}\boldsymbol{\Sigma}_t\}$ for taking the derivative of the expected cost with respect to $p(\mathbf{x}_t)$. From here on the computation of gradients focuses on $\mathrm{d}\boldsymbol{\mu}_t/\mathrm{d}\boldsymbol{\theta}$ in (17). Computing the derivative of $\boldsymbol{\Sigma}_t$ is similar and further described in [4]. The gradient to the mean $\boldsymbol{\mu}_t$ is

$$\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\mu}_{t-1}} \frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\Sigma}_{t-1}} \frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\theta}}. \tag{18}$$

The derivatives of the mean $\boldsymbol{\mu}_{t-1}$ and covariance $\boldsymbol{\Sigma}_{t-1}$ with respect to the parameters $\boldsymbol{\theta}$ are known from the previous time step. The partial derivatives of $\boldsymbol{\mu}_t$ with respect to the previous state distribution $p(\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$ depend on the approximation method for (12) used during policy evaluation. In the appendix of [4], a detailed derivation of gradients is provided for using *moment matching* during policy evaluation. Therefore, what is left to calculate is the partial derivative of $\boldsymbol{\mu}_t$ with respect to the parameters in (18) as

$$\frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial p(\mathbf{u}_{t-1})} \frac{\partial p(\mathbf{u}_{t-1})}{\partial \boldsymbol{\theta}} \tag{19}$$

$$= \frac{\partial \boldsymbol{\mu}_{\boldsymbol{\Delta}}}{\partial \boldsymbol{\mu}_{\mathbf{u}}} \frac{\partial \boldsymbol{\mu}_{\mathbf{u}}}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\Sigma}_{\boldsymbol{\Delta}}}{\partial \boldsymbol{\mu}_{\mathbf{u}}} \frac{\partial \boldsymbol{\Sigma}_{\mathbf{u}}}{\partial \boldsymbol{\theta}}. \tag{20}$$

The partial derivatives with respect to $\boldsymbol{\theta}$ of the mean $\boldsymbol{\mu}_{\mathbf{u}}$ and covariance $\boldsymbol{\Sigma}_{\mathbf{u}}$ of $p(\mathbf{u}_t)$ depend on the policy representation.

Since the gradient of the accumulated expected cost in (16) can be calculated analytically, standard gradient-based optimization methods such as conjugate gradients (CG) or BFGS can be applied to find optimal policy parameters $\boldsymbol{\theta}^*$. After finding the optimal policy parameters, the optimal policy is applied onto the system, either solving the task or collecting data and executing another iteration according to Alg. 1.

V. Experimental Applications and Results

As a prove of concept in [4] multiple *real world* experiments have been conducted, such as a cart-pole swing up [5], block stacking by a fragile robot [6], and balancing of a robotic unicycle using a flywheel. Furthermore, in [2], PILCO was used to learn and control the throttle valve dynamics in a combustion engine. In the following, only the results of the cart-pole swing up are presented.

A. Cart-Pole Swing Up

In the cart-pole swing up, a pendulum is connected to a cart which moves horizontally on a track. By applying a horizontal force $u \in [-10\mathrm{N}, 10\mathrm{N}]$ the cart can be controlled. The objective of the cart-pole swing up is to find a policy, which swings up the pendulum and balances it in the upright position in the middle of the track.

The cost function was set up as the generalized binary saturating cost as

$$c(\mathbf{x}) = 1 - \exp\left(-\frac{1}{2\sigma_c^2} d(\mathbf{x}, \mathbf{x}_{target})^2\right) \in [0, 1] \tag{21}$$

with parameter σ_c^2 controlling the width of c and $d(\mathbf{x}, \mathbf{x}_{target})$ as the Euclidean distance between the current state and the target state. This cost function allows for a analytic integration of (15) [4].

Since the system is nonlinear, a nonlinear radial basis function (RBF) network as policy

$$\tilde{\pi}(\mathbf{x}, \boldsymbol{\theta}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{x}) \tag{22}$$

was chosen with $n = 50$ squared exponential basis functions

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \boldsymbol{\Lambda}^{-1} (\mathbf{x} - \mu_i)\right). \tag{23}$$

For the experiment the policy parameters are $\boldsymbol{\theta} = \{w_i, \boldsymbol{\Lambda}, \mu_i\} \in \mathbb{R}^{305}$. To consider the box constraints on the controls, a squashing function σ is applied onto the preliminary policy, mapping $\tilde{\pi}$ to a range of $[-1, 1]$ to obtain the final policy as $\pi(\mathbf{x}) = \mathbf{u}_{max}\sigma(\tilde{\pi}(\mathbf{x})) \in [-\mathbf{u}_{max}, \mathbf{u}_{max}]$. In [4] and [5] PILCO was able to successfully learn a sufficiently good performing controller from scratch without any prior knowledge of the system. Furthermore, the controller was robust to small disturbances. On average the total interaction time needed of was 20s (8 trials, each 2.5s long) and the algorithm converged in approximately 95% of the test runs to the correct solution and solving the task. The experiment was also performed by using only the mean prediction of the GP and neglecting the variance resulting in a *deterministic* model. In such a setting, the algorithm was not able to solve the task, which emphasizes the need for probabilistic modeling.

VI. Discussion

In this paper, PILCO, a data-efficient model-based RL algorithm for robotics and control, was presented. It consists of three main parts: *1) Model-learning*, *2) policy evaluation* and *3) policy improvement*. To model the dynamics of a system PILCO uses a GP which allows to explicitly consider uncertainty. This probabilistic approach to modeling dynamics using a GP enables PILCO to compute analytical gradients of the cost function $J^\pi(\boldsymbol{\theta})$ to find optimal policy parameters, rather than relying on gradient estimates as most policy gradient methods do. Furthermore, the probabilistic model is used to evaluate policies thus reducing the amount of trails needed. As a proof of concept, the application of the algorithm to

learning the cart-pole swing up from scratch was presented showing very little interaction time needed with the physical system.

A. Dimensionality and Scalability

However, there are some drawbacks to using a GP in this model-based RL setting. GPs scale cubically in complexity with the amount of observations, making them computationally very expensive for complex systems where many trials would be required for autonomous task learning. Therefore, DeepPILCO [8] adapted the PILCO algorithm to use a Bayesian deep dynamics model instead of a GP, allowing PILCO to scale linearly with respect to training data. However, this no longer allows the gradients of the cost function to be derived analytically, which is why stochastic optimization techniques were used in [8].

In addition to single-robot experiments, DeepPILCO was applied in a multi-agent real-world experiment in [9], where it was compared to a model-free RL approach in Deep Q-Learning (DQL). While DQL required significantly less computation time, DeepPLICO's sample efficiency led to the conclusion that DeepPILCO is superior in overall learning speed compared to DQL and will continue to be so as computational power further improves.

Also the usage of sparse GP models as in [10] could be promising to make PILCO applicable to more complex systems in robotics and control. By only choosing the most *informative* data points from the available training data, the approximation of the dynamics model (1) could still be accurate while reducing the computational impact of oversampled areas in the state space. In addition, incorporating domain knowledge in the form of tailored covariance functions could lead to further data-efficiency.

B. Safety and Safe Exploration

Another adaptation of the PILCO algorithm relates to safety and safe exploration. In [4], box constraints on the controls are considered by using a preliminary policy and a squashing function to obtain the final policy. However, constraints on the states of the system are not considered (e.g., the length of the track in the cart-pole swing up example). With this in mind, an adaptation to PILCO that incorporates state constraints into the algorithm was proposed in SafePILCO [11]. For this, the cost function was modified as

$$J^\pi(\boldsymbol{\theta}) = \xi Q^\pi(\boldsymbol{\theta}) + \sum_{t=1}^{T} \mathbb{E}[c(\mathbf{x}_t)|\boldsymbol{\theta})] \tag{24}$$

presenting a safety-performance trade-off with hyperparameter ξ and $Q^\pi(\boldsymbol{\theta})$ as the joint probability of all states $\mathbf{x}_1, \ldots, \mathbf{x}_T$ of the system being in the safe region $\mathcal{S}$ as

$$Q^\pi(\boldsymbol{\theta}) = \int_{\mathcal{S}} \cdots \int_{\mathcal{S}} p(\mathbf{x}_1, \ldots, \mathbf{x}_T|\boldsymbol{\theta}) \, \mathrm{d}\mathbf{x}_1 \ldots \mathrm{d}\mathbf{x}_T \,. \tag{25}$$

For a policy to be considered safe $Q^\pi(\boldsymbol{\theta}) > \epsilon$ has to hold with ϵ as a threshold. Moreover, (25) is also differentiable with respect to $\boldsymbol{\theta}$ considering box constraints on the states, so the analytical calculation of gradients is still possible [11]. If an optimal policy to (24) violates the ϵ threshold, it is not applied onto the system, but ξ is increased and the cost function (24) is again optimized.

In summary, [11] uses the analytic properties of policy evaluation and the state evolution obtained by it to minimize the risk that a policy violating the constraints is imposed onto the system. Thus, the probability of hardware damage to the system and the environment during learning can be minimized. However, due to the probabilistic nature, constraint violation when applying the SafePILCO algorithm can occur, making the choice of ϵ and the choice of covariance function crucial and domain dependent to still allow for exploration without constraint violation. Its effectiveness also still has to be proven in real-world experiments. To extend this approach, methods could be promising that can monitor the state evolution online during trials and switch to a backup policy in case of imminent danger of constraint violation.

VII. Conclusion

PILCO is able to achieve impressive data-efficiency through its model-based RL approach. Unlike other policy gradient methods, the gradient to the cost function can be calculated analytically and the impact of a policy can be evaluated using the learned probabilistic model. Therefore, only optimal policies given the model are applied to the physical system thus reducing wear and tear. However, the limiting factor of the algorithm is the required computing time, which makes the original algorithm only applicable to low-dimensional systems. To address this problem, the DeepPILCO [8] modification was introduced. Open questions, which arose in the original paper to PILCO [5] regarding safe exploration, were addressed by the extension SafePILCO [11]. Combining both adaptations, DeepPILCO and SafePILCO, could be very promising for the autonomous and safe learning of physical systems in the future.

References

[1] C. G. Atkeson and J.C. Santamaria. "A Comparison of Direct and Model-Based Reinforcment Learning". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. 1997.

[2] Bastian Bischoff et al. "Learning Throttle Valve Control Using Policy Search". In: *European Conference on Machine Learning*. Springer, 2014.

[3] Marc Peter Deisenroth, Dieter Fox, and Jan Peters. "Gaussian Process Dynamic Programming". In: *Neurocomputing* (2009).

[4] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2015).

[5] Marc Peter Deisenroth and Carl Edward Rasmussen. “PILCO: A Model-Based and Data-Efficient Approach to Policy Search”. In: *Proceedings of the International Conference on Machine Learning*. 2011.

[6] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. “Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning”. In: *Proceedings of Robotics: Science and Systems*. 2011.

[7] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *CoRR* (2016).

[8] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. “Improving PILCO with Bayesian Neural Network Dynamics Models”. In: *Data-Efficient Machine Learning Workshop, International Conference on Machine Learning*. 2016.

[9] Jingyi Huang and Andre Rosendo. “Deep Vs. Deep Bayesian: Reinforcement Learning on a Multi-Robot Competitive Experiment”. In: (2020).

[10] Haitao Lui et al. “When Gaussian Process Meets Big Data: A Review of Scalable GPs”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[11] Kyriakos Polymenakos, Alessandro Abate, and Stephan Roberts. “Safe Policy Search Using Gaussian Process Models”. In: (2019).

[12] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[13] Stefan Schaal. “Learning From Demonstration”. In: *Advances in Neural Information Processing Systems 9*. The MIT Press, 1997.

[14] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[15] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012.