

# Model-Based Policy Search Using Monte Carlo Gradient Estimation with Real Systems Application

Fabio Amadio<sup>1</sup>, Alberto Dalla Libera<sup>1</sup>, Riccardo Antonello<sup>1</sup>, Daniel Nikovski<sup>2</sup>, Ruggero Carli<sup>1</sup>, Diego Romeres<sup>2</sup>

**Abstract**—In this paper, we present a Model-Based Reinforcement Learning algorithm named Monte Carlo Probabilistic Inference for Learning Control (MC-PILCO). The algorithm relies on Gaussian Processes (GPs) to model the system dynamics and on a Monte Carlo approach to estimate the policy gradient. This defines a framework in which we ablate the choice of the following components: (i) the selection of the cost function, (ii) the optimization of policies using dropout, (iii) an improved data efficiency through the use of structured kernels in the GP models. The combination of the aforementioned aspects affects dramatically the performance of MC-PILCO. Numerical comparisons in a simulated cart-pole environment show that MC-PILCO exhibits better data-efficiency and control performance w.r.t. state-of-the-art GP-based MBRL algorithms. Finally, we apply MC-PILCO to real systems, considering in particular systems with partially measurable states. We discuss the importance of modeling both the measurement system and the state estimators during policy optimization. The effectiveness of the proposed solutions has been tested in simulation and in two real systems, a Furuta pendulum, and a ball-and-plate.

## I. INTRODUCTION

In recent years, reinforcement learning (RL) [1] has achieved outstanding results in many different environments, and has shown the potential to provide an automated framework for learning different control applications from scratch. However, model-free RL (MFRL) algorithms might require a massive amount of interactions with the environment in order to solve the assigned task. This data inefficiency puts a limit to RL's potential in real-world applications, due to the time and cost of interacting with them. In particular, when dealing with mechanical systems, it is critical to learn the task after the least possible amount of trials, to reduce wear and tear and avoid any damage to the system. A promising way to overcome this limit is model-based reinforcement learning (MBRL), which is based on the use of data from interactions to build a predictive model of the environment and to exploit it to plan control actions. MBRL increases data efficiency by using the model to extract more valuable information from the available data [2].

On the other hand, MBRL methods are effective only inasmuch as their models resemble accurately the real systems. Hence, deterministic models might suffer dramatically from model inaccuracy, and the use of stochastic models becomes necessary in order to capture uncertainty. Gaussian Processes

(GPs) [3] are a class of Bayesian models commonly used in RL methods precisely for their intrinsic capability to handle uncertainty and provide principled stochastic predictions [4][5]. PILCO (Probabilistic Inference for Learning COntrol) [6] is a successful MBRL algorithm that uses GP models and gradient-based policy search to achieve substantial data efficiency in solving different control problems, both in simulation as well as with real systems [7][8]. In PILCO, long-term predictions are computed analytically, approximating the distribution of the next state at each time instant with a Gaussian distribution by means of moment matching. In this way, the policy gradient is computed in closed form. However, the use of moment matching introduces also two relevant limitations. (i) Moment matching allows modeling of only unimodal distributions. This fact, besides being a potential incorrect assumption on the system dynamics, it introduces relevant limitations related to initial conditions. In particular, the restriction on the use of unimodal distributions complicates dealing with multimodal initial conditions, as well as being a potential limitation even when the system initial state is unimodal. For instance, in case that the initial variance is high, the optimal solution might be multimodal, due to dependencies on initial conditions. (ii) The computation of the moments is shown to be tractable only when considering Squared Exponential (SE) kernels and differentiable cost functions. In particular, the limitation on the kernel choice might be very stringent, as GPs with SE kernel impose smooth properties on the posterior estimator and might show poor generalization properties in data that have not been seen during training [9], [10], [11], [12].

PILCO has inspired several other MBRL algorithms, which try to improve it in different ways. Limitations due to the use of SE kernels have been addressed in Deep-PILCO [13], where the system evolution is modeled using Bayesian Neural Networks [14], and long-term predictions are computed combining particle-based methods and moment matching. Results show that, compared to PILCO, Deep-PILCO requires a larger number of interactions with the system in order to learn the task. This fact suggests that using neural networks (NNs) might not be advantageous in terms of data-efficiency, due to the considerably high amount of parameters needed to characterize the model. A more articulated approach has been proposed in PETS [15], where the authors use a probabilistic ensemble of NNs to model the uncertainty of the system dynamics. Despite the positive results in the simulated high-dimension systems, also the numerical results in PETS show that GPs are more data-efficient than NNs when considering low-dimensional systems, such as the cart-pole benchmark. An alternative route has been proposed in [16], where the authors use a

<sup>1</sup> Fabio Amadio, Alberto Dalla Libera, Riccardo Antonello and Ruggero Carli are with the Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy [amadiofa@dei.unipd.it, dallaliber@dei.unipd.it, antonello@dei.unipd.it, carlirug@dei.unipd.it].

<sup>2</sup> Diego Romeres and Daniel Nikovski are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139 [romeres@merl.com, nikovski@merl.com].

simulator to learn a prior for the GP model before starting the reinforcement learning procedure on the actual system to control. This simulated prior improves the performance of PILCO in areas of the state space with no available data points. However, the method requires an accurate simulator that may not always be available at the user.

Limitations due to the gradient-based optimization were addressed in Black-DROPS [17], which adopts a gradient-free policy optimization. In this way, also non-differentiable cost functions can be used, and the computational time can be improved with the parallelization of the black-box optimizer. With this strategy, Black-DROPS achieves similar data efficiency to PILCO’s, but significantly increases asymptotic performance.

Other approaches focused on improving the accuracy of long-term predictions, overcoming approximations due to moment matching. A first attempt has been proposed in [18], where long-term distributions are computed relying on particle-based methods. Based on the current policy and the one-step-ahead GP models, the authors simulate the evolution of a batch of particles sampled from the initial state distribution. Then, the particle trajectories are used to approximate the expected cumulative cost. The policy gradient is computed using the strategy proposed in PEGASUS [19], where by fixing the initial random seed, a probabilistic Markov decision process (MDP) is transformed into an equivalent partially observable MDP with deterministic transitions. Compared to PILCO, results obtained where not satisfactory. The poor performance was attributed to the policy optimization method, in particular, to inability to escape from the numerous local minima generated by the multimodal distribution. Another particle-based approach is PIPPS [20], where the policy gradient is computed with the so-called *reparameterization trick* [21] instead of the PEGASUS strategy. Given a distribution  $p_\theta(\cdot)$ , parameterized by  $\theta$ , the *reparameterization trick* provides an alternative method for generating samples from  $p_\theta(\cdot)$  such that those samples are differentiable w.r.t.  $\theta$ . The *reparameterization trick* has been introduced with successful results in stochastic variational inference (SVI) [21], [22]. In contrast with the results obtained in SVI, where just a few samples are needed to estimate the gradient, the authors of [20] highlighted several issues related to the gradient computed with the *reparameterization trick*, due to its exploding magnitude and random direction. To overcome these issues, they proposed the *total propagation algorithm*, where the *reparameterization trick* is combined with the likelihood ratio gradient. The algorithm performs similarly to PILCO with some improvements in the gradient computation and in the performance in the presence of additional noise.

In this work, we propose an MBRL algorithm named Monte Carlo Probabilistic Inference for Learning COntrol (MC-PILCO). Like PILCO, MC-PILCO is a policy gradient algorithm, which uses GPs to describe the one-step-ahead system dynamics and relies on a particle-based method to approximate the long-term state distribution instead than using moment matching. The gradient of the expected cumulative cost w.r.t. the policy parameters is obtained by backpropagation [23] on the associated stochastic computational graph, exploiting the *reparameterization trick*. Differently than in PIPPS, where they focused on obtaining accurate estimates of the gradient,

we have interpreted the optimization problem as a stochastic gradient descent (SGD) problem [24]. This problem has been studied in depth in the context of neural networks, where overparameterized models are optimized using noisy estimates of the gradient [25]. Analytical and experimental studies show that the shape of the cost function and the nonlinear activation function adopted can affect dramatically the performance of SGD algorithms [26], [27], [28]. Motivated by the results obtained in this field, w.r.t. the previous particles-based approaches, we considered the use of more complex policies and less peaked cost functions, i.e., less penalizing costs. During policy optimization we also considered the application of dropout [29] to the policy parameters, in order to improve the ability to escape from local minima, obtaining more performing policies. The effectiveness of the proposed choice has been ablated and analyzed in simulation. First, a simulated cart-pole, a common benchmark system, was considered to compare MC-PILCO with PILCO and Black-DROPS. Results show that MC-PILCO outperforms both PILCO and Black-DROPS, which can be considered state-of-the-art GP-based MBRL algorithms. Second, with the purpose to evaluate the behavior of MC-PILCO in a higher-dimensional system, we applied it to a simulated UR5 robotic arm. The considered task consists of learning a joint-space controller able to follow a desired trajectory and it was successfully accomplished. These results confirm that the *reparameterization trick* can be used effectively in MBRL, and Monte Carlo methods do not suffer of gradient estimation problems, as commonly asserted in literature, if properly considering the cost function, the use of dropout and complex/rich policies.

Moreover, differently from previous works which combined GPs with particle-based methods, we show a relevant advantage of this strategy, namely, the possibility of adopting different kernel functions. We considered the use of a kernel function given by the combination of an SE kernel and a polynomial kernel [30], as well as a semiparametrical model [10], [11], [12]. Results obtained both in simulation and in a real Furuta pendulum show that the use of such kernels significantly increases data efficiency, limiting the interaction time required to learn the tasks.

Finally, we have applied and analyzed MC-PILCO in real systems. Unlike simulated environments, where the state is typically assumed to be fully measurable, the state of real systems might be partially measurable. For instance, most of the time, only positions are directly measured in real robotic systems, while velocities are typically computed by means of estimators, such as state observers, Kalman filters, and numerical differentiation with low-pass filters. In particular, the controller, i.e., the policy, works with the output of online state estimators which, due to noise and real-time computation constraints, might introduce significant delays and discrepancies w.r.t. to the filtered data used during policy training. In this context, we verified that during policy optimization it is important to distinguish between the states generated by the models, which aim at describing the evolution of the real system state, and the states provided to the policy. Indeed, providing to the control policy the model predictions corresponds to assume to measure directly the system state, which, as mentioned

before, is not possible in the real system. This incorrect assumption might compromise the effectiveness of the trained policy into the real system, due to the presence of distortions caused by the online state estimators. Hence, during policy optimization, from the evolution of the system state predicted by the GPs models, we compute the estimates of the states observed by modeling both the measurement system and the online estimators used in the real system. Then we feed to the policy the estimates of the observed states. In this way, we aim at obtaining robustness w.r.t. the delays and distortions caused by online filtering. The effectiveness of the proposed strategy has been tested both in simulation and also with two real systems, a Furuta pendulum and a ball-and-plate system. The obtained performance confirms the importance of considering the presence of filters in the real system during policy optimization.

The article is structured as follows. In Section II, some background notions are provided: we state the general problem of model-based policy gradient methods, and present modelling approaches of dynamical systems with GPs. In Section III, we present MC-PILCO, our proposed algorithm, detailing the policy optimization and model learning techniques adopted. In Section IV, we analyze several aspects affecting the performance of MC-PILCO, such as the cost shape, dropout, and the kernel choice. In Section V, we compare MC-PILCO with PILCO and Black-DROPS using a simulated cart-pole benchmark system, test the MC-PILCO performance with a simulated UR5 robot, and also test the advantages of the particle-based approach when dealing with different distributions of the initial conditions. In Section VI, we discuss the application of the proposed algorithm to systems with partially measurable state. Experiments on a real Furuta pendulum and a ball-and-plate system are shown in Section VII. Finally, we draw conclusions in Section VIII.

## II. BACKGROUND

In this section, we first introduce the standard framework considered in model-based policy gradient RL methods, and then discuss how to use Gaussian Process Regression (GPR) for model learning. In the latter topic, we focus on three aspects: some background notions about GPR, the description of the model prediction for one-step-ahead, and finally, we discuss long term predictions, focusing on two possible strategies, namely, moment matching and particle-based method.

### A. Model-Based Policy Gradient

Consider the discrete-time system described by the unknown transition function  $f(\cdot, \cdot)$ ,

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}_t,$$

where, at each time step  $t$ ,  $\mathbf{x}_t \in \mathbb{R}^{d_x}$  and  $\mathbf{u}_t \in \mathbb{R}^{d_u}$  are, respectively, the state and the inputs of the system, while  $\mathbf{w}_t \sim \mathcal{N}(0, \Sigma_w)$  is an independent Gaussian random variable modeling additive noise. The cost function  $c(\mathbf{x}_t)$  is defined to characterize the immediate penalty for being in state  $\mathbf{x}_t$ .

Inputs are chosen according to a policy function  $\pi_\theta : \mathbf{x} \mapsto \mathbf{u}$  that depends on the parameter vector  $\theta$ .

The objective is to find the policy that minimizes the expected cumulative cost over a finite number of time steps  $T$ , i.e.,

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

with an initial state distributed according to a given  $p(\mathbf{x}_0)$ .

A model-based approach for learning a policy consists, generally, of the succession of several trials; i.e. attempts to solve the desired task. Each trial consists of three main phases:

- *Model Learning*: the data collected from all the previous interactions are used to build a model of the system dynamics (at the first iteration data are collected applying possibly random exploratory controls);
- *Policy Update*: the policy is optimized in order to minimize the cost  $J(\theta)$  according to the current model;
- *Policy Execution*: the current optimized policy is applied to the system and the data are stored for model improvement.

Model-based policy gradient methods use the learned model to predict the state evolution when the current policy is applied. These predictions are used to estimate  $J(\theta)$  and its gradient  $\nabla_{\theta} J$ , in order to update the policy parameters  $\theta$  following a gradient-descent approach.

### B. GPR and one-step-ahead predictions

A common strategy with GPR-based approaches consists of modeling the evolution of each state dimension with a distinct GP. Let's denote by  $\Delta_t^{(i)} = x_{t+1}^{(i)} - x_t^{(i)}$  the difference between the value of the  $i$ -th component at time  $t+1$  and  $t$ , and by  $y_t^{(i)}$  the noisy measurement of  $\Delta_t^{(i)}$  with  $i \in \{1, \dots, d_x\}$ . Moreover, let  $\tilde{\mathbf{x}}_t = [\mathbf{x}_t, \mathbf{u}_t]$  be the vector that includes the state and the input at time  $t$ , also called the GP input. Then, given the data  $\mathcal{D} = (\tilde{X}, \mathbf{y}^{(i)})$ , where  $\mathbf{y}^{(i)} = [y_{t_1}^{(i)}, \dots, y_{t_n}^{(i)}]^T$  is a vector of  $n$  output measurements, and  $\tilde{X} = \{\tilde{\mathbf{x}}_{t_1}, \dots, \tilde{\mathbf{x}}_{t_n}\}$  is the set of GP inputs, GPR assumes the following probabilistic model, for each state dimension,

$$\mathbf{y}^{(i)} = \begin{bmatrix} h^{(i)}(\tilde{\mathbf{x}}_{t_1}) \\ \vdots \\ h^{(i)}(\tilde{\mathbf{x}}_{t_n}) \end{bmatrix} + \begin{bmatrix} e_{t_1}^{(i)} \\ \vdots \\ e_{t_n}^{(i)} \end{bmatrix} = \mathbf{h}^{(i)}(\tilde{X}) + \mathbf{e}^{(i)},$$

where  $\mathbf{e}^{(i)}$  is a zero-mean Gaussian i.i.d. noise with standard deviation  $\sigma_i$ , and  $h^{(i)}(\cdot)$  is an unknown function modeled a priori as a zero-mean Gaussian Process and  $i \in \{1, \dots, d_x\}$ . In particular, we have  $\mathbf{h}^{(i)} \sim \mathcal{N}(0, K_i(\tilde{X}, \tilde{X}))$ , with the a priori covariance matrix  $K_i(\tilde{X}, \tilde{X}) \in \mathbb{R}^{n \times n}$  defined element-wise through a kernel function  $k_i(\cdot, \cdot)$ , namely, the element in  $j$ -th row and  $k$ -th column is given by  $k_i(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k})$ . A crucial aspect in GPR is the kernel choice. The kernel function encodes prior assumptions about the process. One of the most common choices for continuous functions is the SE kernel, defined as

$$k_{SE}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) := \lambda^2 e^{-\|\tilde{\mathbf{x}}_{t_j} - \tilde{\mathbf{x}}_{t_k}\|_{\Lambda}^2}, \quad (2)$$

where the scaling factor  $\lambda$  and the matrix  $\Lambda$  are kernel hyperparameters which can be estimated by marginal likelihood maximization. Typically,  $\Lambda$  is assumed to be diagonal, with the diagonal elements named lengthscales.

Remarkably, the posterior distribution of  $h^{(i)}(\cdot)$  can be computed in closed form. Let  $\tilde{\mathbf{x}}_t$  be a general GP input at time  $t$ . Then, the distribution of  $\hat{\Delta}_t^{(i)}$ , the estimate of  $\Delta_t^{(i)}$ , is Gaussian with mean and variance given by

$$\mathbb{E}[\hat{\Delta}_t^{(i)}] = k_i(\tilde{\mathbf{x}}_t, \tilde{X})\Gamma_i^{-1}\mathbf{y}^{(i)}, \quad (3)$$

$$\text{var}[\hat{\Delta}_t^{(i)}] = k_i(\tilde{\mathbf{x}}_t, \tilde{\mathbf{x}}_t) - k_i(\tilde{\mathbf{x}}_t, \tilde{X})\Gamma_i^{-1}k_i^T(\tilde{\mathbf{x}}_t, \tilde{X}), \quad (4)$$

with  $\Gamma_i$  and  $k_i(\tilde{\mathbf{x}}_t, \tilde{X})$  defined as

$$\begin{aligned} \Gamma_i &= (K_i(\tilde{X}, \tilde{X}) + \sigma_i^2 I), \\ k_i(\tilde{\mathbf{x}}_t, \tilde{X}) &= [k_i(\tilde{\mathbf{x}}_t, \tilde{\mathbf{x}}_{t_1}), \dots, k_i(\tilde{\mathbf{x}}_t, \tilde{\mathbf{x}}_{t_n})]. \end{aligned}$$

Recalling that the evolution of each state dimension is modeled with a distinct GP, and assuming that the GPs are conditionally independent given the current GP input  $\tilde{\mathbf{x}}_t$ , the posterior distribution for the estimated state at time  $t+1$  is

$$p(\hat{\mathbf{x}}_{t+1}|\tilde{\mathbf{x}}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}), \quad (5)$$

where

$$\boldsymbol{\mu}_{t+1} = \mathbf{x}_t + \left[ \mathbb{E}[\hat{\Delta}_t^{(1)}], \dots, \mathbb{E}[\hat{\Delta}_t^{(d_{\mathbf{x}})}] \right]^T, \quad (6)$$

$$\Sigma_{t+1} = \text{diag} \left( \left[ \text{var}[\hat{\Delta}_t^{(1)}], \dots, \text{var}[\hat{\Delta}_t^{(d_{\mathbf{x}})}] \right] \right). \quad (7)$$

### C. Long-term predictions with GP dynamical models

In MBRL, the policy  $\pi_{\theta}$  is evaluated and improved based on long-term predictions of the state evolution:  $p(\hat{\mathbf{x}}_1), \dots, p(\hat{\mathbf{x}}_T)$ . The exact computation of these quantities entails the application of the one-step-ahead GP models in cascade, considering the propagation of the uncertainty. More precisely, starting from a given initial distribution  $p(\mathbf{x}_0)$ , at each time step  $t$ , the next state distribution is obtained by marginalizing (5) over  $p(\hat{\mathbf{x}}_t)$ , that is,

$$p(\hat{\mathbf{x}}_{t+1}) = \int p(\hat{\mathbf{x}}_{t+1}|\hat{\mathbf{x}}_t, \pi_{\theta}(\hat{\mathbf{x}}_t), \mathcal{D})p(\hat{\mathbf{x}}_t)d\hat{\mathbf{x}}_t. \quad (8)$$

Unfortunately, computing the exact predicted distribution in (8) is not tractable. There are different ways to solve it approximately, here we discuss two main approaches: moment matching, adopted by PILCO, and a particle-based method, the strategy followed in this work.

1) *Moment matching*: Assuming that the GP models use only the SE kernel as a prior covariance, and considering a normal initial state distribution  $x_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$ , the first and the second moments of  $p(\hat{\mathbf{x}}_1)$  can be computed in closed form [31]. Then, the distribution  $p(\hat{\mathbf{x}}_1)$  is approximated to be a Gaussian distribution, whose mean and variance correspond to the moments computed previously. Finally, the subsequent probability distributions are computed iterating the procedure for each time step of the prediction horizon. For details about the computation of the first and second moments, we refer the reader to [31]. Moment matching offers the advantage of providing a closed form solution for handling uncertainty propagation through the GP dynamics model. Thus, in this setting, it is possible to analytically compute the policy gradient from long-term predictions. However, as already mentioned in Section I, the Gaussian approximation performed in moment

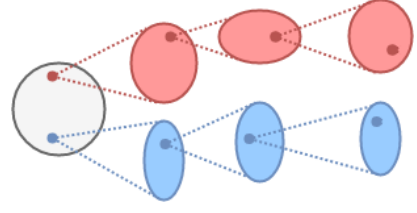


Fig. 1: Example of two particles propagating through the stochastic model (Gaussian distributions represented as ellipses).

matching is also the cause of two main weaknesses: (i) The computation of the two moments has been performed assuming the use of SE kernels, which might lead to poor generalization properties in data that have not been seen during training [9], [10], [11], [12]. (ii) Moment matching allows modeling only unimodal distributions, which might be a too restrictive approximation of the real system behavior.

2) *Particle-based method*: The integral in (8) can be approximated relying on Monte Carlo approaches, in particular on particle-based methods. Specifically,  $M$  particles are sampled from the initial state distribution  $p(\mathbf{x}_0)$ . Each one of the  $M$  particles is propagated using the one-step-ahead GP models (5). Let  $\mathbf{x}_t^{(m)}$  be the state of the  $m$ -th particle at time  $t$ , with  $m = 1, \dots, M$ . At time step  $t$ , the actual policy  $\pi_{\theta}$  is evaluated to compute the associated control. The GP model provides the Gaussian distribution  $p(\mathbf{x}_{t+1}^{(m)}|\mathbf{x}_t^{(m)}, \pi_{\theta}(\mathbf{x}_t^{(m)}), \mathcal{D})$  from which  $\mathbf{x}_{t+1}^{(m)}$ , the state of the particle at the next time step, is sampled. This process is iterated until a trajectory of length  $T$  is generated for each particle. The process is illustrated in Figure 1 for the sake of clarity. The long-term distribution at each time step is approximated with the distribution of the particles. Notice that this approach does not impose any constraint on the choice of the kernel function and the initial state distribution. Moreover, no approximations on the distribution  $p(\hat{\mathbf{x}}_t)$  are made. Therefore, particle-based methods do not suffer from the problems seen in moment matching, at the cost of being more computationally heavy. Specifically, the computation of (5) entails the computation of (3) and (4), which are, respectively, the mean and the variance of the delta states. Regarding the computational complexity, it can be noted that  $\Gamma_i^{-1}\mathbf{y}^{(i)}$  is computed a single time offline during the training of the GP model (same computation is needed in the moment matching case), and the number of operations required to compute (3) is linear w.r.t. the number of samples  $n$ . The computational bottleneck is the computation of (4), which is  $O(n^2)$ . Then, the cost of a single state prediction is  $O(d_{\mathbf{x}}n^2)$ , leading to a total computational cost of  $O(d_{\mathbf{x}}MTn^2)$ . Depending on the complexity of the system dynamics, the number of particles necessary to obtain a good approximation might be high, determining a considerable computational burden. Nevertheless, the computational burden can be substantially mitigated via GPU parallel computing, due to the possibility of computing the evolution of each particle in parallel.

### III. MC-PILCO

In this section, we present the proposed algorithm. MC-PILCO relies on GPR for model learning and follows a Monte Carlo sampling method to estimate the expected cumulative cost from particles trajectories propagated through the learned model. We exploit the *reparameterization trick* to obtain the policy gradient from the sampled particles and optimize the policy. This way of proceeding is very flexible, and allows using any kind of kernels for the GPs, as well as providing more reliable approximations of the system's behaviour. MC-PILCO, in broad terms, consists of the iteration of three main steps, namely, update the GP models, update the policy parameters, and execute the policy on the system. In its turn, the policy update is composed of three steps, iterated for a maximum of  $N_{opt}$  times:

- simulate the evolution of  $M$  particles, based on the current  $\pi_\theta$  and on the GP models learned from the previously observed data;
- compute  $\hat{J}(\theta)$ , an approximation of the expected cumulative cost, based on the evolution of the  $M$  particles;
- update the policy parameters  $\theta$  based on  $\nabla_\theta \hat{J}(\theta)$ , the gradient of  $\hat{J}(\theta)$  w.r.t.  $\theta$ , computed by backpropagation.

In the remainder of this section, we discuss in more depth the model learning step and the policy optimization step.

#### A. Model Learning

Here, we describe the model learning framework considered in MC-PILCO. We begin by showing the proposed one-step-ahead prediction model, and analyzing the advantages w.r.t. the standard model described in Section II-B. Then, we discuss the choice of the kernel functions. Finally, we briefly discuss the model's hyperparameters optimization and the strategy adopted to reduce the computational cost.

1) *One-step-ahead model*: Let the state be defined as  $\mathbf{x}_t = [\mathbf{q}_t^T, \dot{\mathbf{q}}_t^T]^T$ , where  $\mathbf{q}_t \in \mathbb{R}^{\frac{d_x}{2}}$  is the vector of the generalized coordinates of the system at time step  $t$ , and,  $\dot{\mathbf{q}}_t$  represents the derivative of  $\mathbf{q}_t$  w.r.t. time. MC-PILCO adopts a one-step-ahead model, hereafter denoted as *speed-integration* dynamical model, which exploits the intrinsic correlation between the state components  $\mathbf{q}$  and  $\dot{\mathbf{q}}$ . Indeed, when considering a sufficiently small sampling time  $T_s$  (small w.r.t. the application), it is reasonable to assume constant accelerations between two consecutive time-steps, obtaining the following evolution of  $\mathbf{q}_t$ ,

$$\mathbf{q}_{t+1} = \mathbf{q}_t + T_s \dot{\mathbf{q}}_t + \frac{T_s}{2} (\dot{\mathbf{q}}_{t+1} - \dot{\mathbf{q}}_t). \quad (9)$$

Let  $\mathcal{I}_q$  (respectively  $\mathcal{I}_{\dot{q}}$ ) be the ordered set of the dimension indices of the state  $\mathbf{x}$  associated with  $\mathbf{q}$  (respectively  $\dot{\mathbf{q}}$ ). The proposed *speed-integration* model learns only  $d_x/2$  GPs, each of which models the evolution of a distinct velocity component  $\Delta_t^{(i_k)}$ , with  $i_k \in \mathcal{I}_{\dot{q}}$ . Then, the evolution of the position change,  $\Delta_t^{(i_k)}$ , with  $i_k \in \mathcal{I}_q$  is computed according to (9) and the predicted change in velocity.

Many previous MBRL algorithms, see for instance [6], [17], adopted the standard model described in Section II-B, and hereafter denoted as *full-state* dynamical model. The *full-state*

model predicts the change of each state component with a distinct and independent GP. Doing so, the evolution of each state dimension is assumed to be conditionally independent given the current GP input, and it is necessary to learn a number of GPs equal to the state dimension  $d_x$ . Then, compared to the *full-state* model, the proposed *speed-integration* model halves the number of GPs to be learned, decreasing the cost of a state prediction to  $O(\frac{d_x}{2} MTn^2)$ . Nevertheless, this approach is based on a constant acceleration assumption, and works properly only when considering small enough sampling times. However, MC-PILCO can use also the standard *full-state* model, which might be more effective when sampling time is too high.

2) *Kernel functions*: Regardless of the GP dynamical model structure adopted, one of the advantages of the particle-based policy optimization method is the possibility of choosing any kernel functions without restrictions. Hence, we considered different kernel functions as examples to model the evolution of physical systems. But the reader can consider a custom kernel function appropriate for his application.

**Squared exponential (SE)**. The SE kernel described in (2) represents the standard choice adopted in many different works.

**SE + Polynomial (SE+P<sup>(d)</sup>)**. Recalling that the sum of kernels is still a kernel [3], we considered also a kernel given by the sum of a SE and a polynomial kernel. In particular, we used the Multiplicative Polynomial (MP) kernel, which is a refinement of the standard polynomial kernel, introduced in [30]. The MP kernel of degree  $d$  is defined as the product of  $d$  linear kernels, namely,

$$k_P^{(d)}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) := \prod_{r=1}^d \left( \sigma_{P_r}^2 + \tilde{\mathbf{x}}_{t_j}^T \Sigma_{P_r} \tilde{\mathbf{x}}_{t_k} \right).$$

where the  $\Sigma_{P_r} > 0$  matrices are distinct diagonal matrices. The diagonal elements of the  $\Sigma_{P_r}$ , together with the  $\sigma_{P_r}^2$  elements are the kernel hyperparameters. The resulting kernel is

$$k_{SE+P^{(d)}}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) = k_{SE}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) + k_P^{(d)}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}). \quad (10)$$

The idea motivating this choice is the following: the MP kernel allows capturing possible modes of the system that are polynomial functions in  $\tilde{\mathbf{x}}$ , which are typical in mechanical systems [9], while the SE kernel models more complex behaviors not captured by the polynomial kernel.

**Semi-Parametrical (SP)**. When prior knowledge about the system dynamics is available, for example given by physics first principles, the so called physically inspired (PI) kernel can be derived. The PI kernel is a linear kernel defined on suitable basis functions  $\phi(\tilde{\mathbf{x}})$ , see for instance [10]. More precisely,  $\phi(\tilde{\mathbf{x}}) \in \mathbb{R}^{d_\phi}$  is a, possibly nonlinear, transformation of the GP input  $\tilde{\mathbf{x}}$  determined by the physical model. Then we have

$$k_{PI}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) = \phi^T(\tilde{\mathbf{x}}_{t_j}) \Sigma_{PI} \phi(\tilde{\mathbf{x}}_{t_k}),$$

where  $\Sigma_{PI}$  is a  $d_\phi \times d_\phi$  positive-definite matrix, whose elements are the  $k_{PI}$  hyperparameters; to limit the number of hyperparameters, a standard choice consists in considering  $\Sigma_{PI}$  to be diagonal. To compensate possible inaccuracies of the physical model, it is common to combine  $k_{PI}$  with an SE kernel, obtaining so called semi-parametric kernels [12], [10], expressed as

$$k_{SP}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) = k_{PI}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}) + k_{SE}(\tilde{\mathbf{x}}_{t_j}, \tilde{\mathbf{x}}_{t_k}).$$

The rationale behind this kernel is the following:  $k_{PI}$  encodes the prior information given by the physics, and  $k_{SE}$  compensates for the dynamical components unmodeled in  $k_{PI}$ .

3) *Model optimization and reduction techniques*: In MC-PILCO, the GP hyperparameters are optimized by maximizing the marginal likelihood (ML) of the training samples, see [3]. In Section II-C2, we saw that the computational cost of a particle prediction scales with the square of the number of samples  $n$ , leading to a considerable computational burden when  $n$  is high. In this context, it is essential to implement a strategy to limit the computational burden of a prediction. Several solutions have been proposed in the literature, see [32] for an overview. We implemented a procedure inspired by [33], where the authors proposed an online importance sampling strategy. After optimizing the GP hyperparameters by ML maximization, the samples in  $\mathcal{D}$  are downsampled to a subset  $\mathcal{D}_r = (\tilde{X}_r, \mathbf{y}_r^{(i)})$ , which is then used to compute the predictions. This procedure first initializes  $\mathcal{D}_r$  with the first sample in  $\mathcal{D}$ , then, it computes iteratively the GP estimates of all the remaining samples in  $\mathcal{D}$ , using  $\mathcal{D}_r$  as training samples. Each sample in  $\mathcal{D}$  is either added to  $\mathcal{D}_r$  if the uncertainty of the estimate is higher than a threshold  $\beta^{(i)}$  or it is discarded. The GP estimator is updated every time a sample is added to  $\mathcal{D}_r$ . The trade-off between the reduction of the computational complexity and the severity of the approximation introduced is regulated by tuning  $\beta^{(i)}$ . The higher the  $\beta^{(i)}$ , the smaller the number of samples in  $\mathcal{D}_r$ . On the other hand, using values of  $\beta^{(i)}$  too high might compromise the accuracy of GP predictions.

## B. Policy Optimization

Here, we present the policy optimization strategy adopted in MC-PILCO. We start by describing the general-purpose policy structure considered. Later, we show how to exploit backpropagation and the *reparameterization trick* to estimate the policy gradient from particle-based long-term predictions. Finally, we explain how to implement dropout in this framework.

1) *Policy structure*: In all the experiments presented in this work, we considered an RBF network policy with outputs limited by an hyperbolic tangent function, properly scaled. We call this function *squashed-RBF-network*, and it is defined as

$$\pi_{\theta}(\mathbf{x}) = u_{max} \tanh \left( \frac{1}{u_{max}} \sum_{i=1}^{n_b} w_i e^{\|\mathbf{a}_i - \mathbf{x}\|_{\Sigma_{\pi}}^2} \right). \quad (11)$$

The policy parameters are  $\theta = \{\mathbf{w}, A, \Sigma_{\pi}\}$ , where  $\mathbf{w} = [w_1 \dots w_{n_b}]$  and  $A = \{\mathbf{a}_1 \dots \mathbf{a}_{n_b}\}$  are, respectively, the weights and the centers of the Gaussian basis functions, while  $\Sigma_{\pi}$  determines the shape of the Gaussian basis functions; in all experiments we assumed  $\Sigma_{\pi}$  to be diagonal. The maximum control action  $u_{max}$  is constant and chosen depending on the system to control. It is worth mentioning that MC-PILCO can deal with any differentiable policy, so more complex functions, such as deep neural networks could be considered too.

2) *Computation of the gradient*: MC-PILCO derives the policy gradient by applying the *reparameterization trick* to the computation of the estimated expected cumulative cost in (1), obtained relying on Monte Carlo sampling [34]. Given a control policy  $\pi_{\theta}$  and an initial state distribution  $p(\mathbf{x}_0)$ , the

evolution of a sufficiently high number of particles is predicted as described in Section II-C2. Thus, the sample mean of the costs incurred by the particles at time step  $t$  approximates each  $\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$ . Specifically, let  $\mathbf{x}_t^{(m)}$  be the state of the  $m$ -th particle at time  $t$ , with  $m = 1, \dots, M$  and  $t = 0, \dots, T$ . The Monte Carlo estimate of the expected cumulative cost is computed with the following expression,

$$\hat{J}(\theta) = \sum_{t=0}^T \left( \frac{1}{M} \sum_{m=1}^M c(\mathbf{x}_t^{(m)}) \right). \quad (12)$$

To compute the gradient of (12) w.r.t. the policy parameters, we use the *reparameterization trick* [21] to differentiate through the stochastic operations. The evolution of every particle  $\mathbf{x}_t^{(m)}$  at the next time step is sampled from the normal distribution  $p(\mathbf{x}_{t+1}^{(m)} | \mathbf{x}_t^{(m)}, \pi_{\theta}(\mathbf{x}_t^{(m)}), \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1})$ , defined in (6)-(7). However, instead of sampling directly from  $\mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1})$ , the *reparameterization trick* samples a point  $\epsilon$  from a zero-mean and unit-variance normal distribution of proper dimensions. Then, it maps this point to the desired distribution as  $\mathbf{x}_{t+1}^{(m)} = \boldsymbol{\mu}_{t+1} + L_{t+1}\epsilon$ , where  $L_{t+1}$  is the Cholesky decomposition of  $\Sigma_{t+1}$ , namely,  $L_{t+1}L_{t+1}^T = \Sigma_{t+1}$ . Now, it is possible to differentiate  $\mathbf{x}_{t+1}^{(m)}$  w.r.t. the distribution parameters, and  $\nabla_{\theta} \hat{J}$  is computed simply by backpropagation. We update the policy parameters using the Adam solver [35]; we will denote the Adam step size with  $\alpha_{lr}$ .

3) *Dropout*: To improve exploration in the parameters  $\theta$  and increase the ability of escaping from local minima during policy optimization, we considered the use of dropout [29]. The adopted procedure is described assuming that the policy is the squashed-RBF-network in (11); similar considerations can be applied to different policy functions. When dropout is applied to the policy in (11) at each evaluation of the policy function, the policy weights  $\mathbf{w}$  are randomly dropped with probability  $p_d$ . This operation is performed by scaling each weight  $w_i$  with a random variable  $r_i \sim \text{Bernoulli}(1 - p_d)$ , where  $\text{Bernoulli}(1 - p_d)$  denotes a Bernoulli distribution, assuming value  $1/(1 - p_d)$  with probability  $1 - p_d$ , and 0 with probability  $p_d$ . This operation is equivalent to defining a probability distribution for  $\mathbf{w}$ , obtaining a parameterized stochastic policy. In particular, as shown in [36], the distribution of each  $w_i$  can be approximated with a bimodal distribution, defined by the sum of two properly scaled Gaussian distributions with infinitely small variance  $\xi^2$ , namely,

$$p_d \mathcal{N}(0, \xi^2) + (1 - p_d) \mathcal{N}\left(\frac{w_i}{1 - p_d}, \xi^2\right).$$

The use of a stochastic policy during policy optimization allows increasing the entropy of the particles' distribution. This property increments the probability of visiting low-cost regions and escaping from local minima. In addition, we also verified that dropout can mitigate issues related to exploding gradients. This is probably due to the fact that the average of several different values of  $\mathbf{w}$  is used to compute the gradient and not a single value of  $\mathbf{w}$ , i.e., different policy functions are used, obtaining a regularization of the gradient estimates.

By contrast, the use of a stochastic policy might affect the precision of the obtained solution due to the additional entropy.

We also need to take in consideration that the final objective is to obtain a deterministic policy. For these reasons, we designed an heuristic scaling procedure to gradually decrease the dropout rate,  $p_d$ , until it equals 0. The scaling action is triggered by a monitoring signal  $s$ , defined from the statistics of the past history of  $\hat{J}$ . Define the cost change,  $\Delta\hat{J}_j = \hat{J}(\theta_j) - \hat{J}(\theta_{j-1})$ , where  $\theta_j$  denotes the policy parameters at the  $j$ -th optimization step. Then,  $s$  is computed as a filtered version of the ratio between  $\mathcal{E}[\Delta\hat{J}_j]$  and  $\sqrt{\mathcal{V}[\Delta\hat{J}_j]}$ , that are, respectively, the mean and the standard deviation of  $\Delta\hat{J}_j$  computed with an Exponential Moving Average (EMA) filter. The expression of  $s$  at the  $j$ -th optimization step is the following:

$$\begin{aligned} \mathcal{E}[\Delta\hat{J}_j] &= \alpha_s \mathcal{E}[\Delta\hat{J}_{j-1}] + (1 - \alpha_s) \Delta\hat{J}_j, \\ \mathcal{V}[\Delta\hat{J}_j] &= \alpha_s (\mathcal{V}[\Delta\hat{J}_{j-1}] + (1 - \alpha_s) (\Delta\hat{J}_j - \mathcal{E}[\Delta\hat{J}_{j-1}])^2), \\ s_j &= \alpha_s s_{j-1} + (1 - \alpha_s) \frac{\mathcal{E}[\Delta\hat{J}_j]}{\sqrt{\mathcal{V}[\Delta\hat{J}_j]}}, \end{aligned} \quad (13)$$

with  $\alpha_s$  a coefficient of the exponential moving average filter, which determines the memory of the filter. At each iteration of the optimization procedure, the algorithm checks if the absolute value of the monitoring signal  $s$  in the last  $n_s$  iterations is below the threshold  $\sigma_s$ , namely,

$$[|s_{j-n_s}| \dots |s_j|] < \sigma_s, \quad (14)$$

where  $<$  is an element-wise operator, and the condition in (14) is true if it is verified for all the elements. If the condition is verified,  $p_d$  is decreased by the quantity  $\Delta p_d$ , and both the learning rate of the optimizer,  $\alpha_{lr}$ , and  $\sigma_s$ , are scaled by an arbitrary factor  $\lambda_s$ . Then, we have:

$$p_d = p_d - \Delta p_d \quad (15a)$$

$$\alpha_{lr} = \lambda_s \alpha_{lr} \quad (15b)$$

$$\sigma_s = \lambda_s \sigma_s \quad (15c)$$

The procedure is iterated as long as

$$p_d \geq 0 \text{ and } \alpha_{lr} \geq \alpha_{lr_{min}}, \quad (16)$$

where  $\alpha_{lr_{min}}$  is the minimum value of the learning rate.

The rationale behind this heuristic scaling procedure is the following. The  $s_j$  signal is small, if  $\mathcal{E}[\Delta\hat{J}_j]$  is close to zero, or if  $\mathcal{V}[\Delta\hat{J}_j]$  is particularly high. The first case happens when the optimization reaches a minimum, while the high variance denotes that the particles' trajectories cross regions of the workspace where the uncertainty of the GPs predictions is high. In both cases, we are interested in testing the policy on the real system, in the first case to verify if the configuration reached solves the task, and in the second case to collect data where predictions are uncertain, and so to improve model accuracy. MC-PILCO with dropout is summarized in pseudo-code in Algorithm 1.

We conclude the discussion about policy optimization by reporting, in Table I, the optimization parameters used in all the proposed experiments, unless expressly stated otherwise. However, it is worth mentioning that some adaptation could be needed in other setups, depending on the problem considered.

parameter	description	value
$p_d$	dropout probability	0.25
$\Delta p_d$	$p_d$ reduction coeff.	0.125
$\alpha_{lr}$	Adam step size	0.01
$\alpha_{lr_{min}}$	minimum step size	0.0025
$\alpha_s$	EMA filter coeff.	0.99
$\sigma_s$	monitoring signal threshold	0.08
$n_s$	num. iterations monitoring	200
$\lambda_s$	$\sigma_s$ reduction coeff.	0.5

TABLE I: Standard values for the policy optimization parameters.

---

#### Algorithm 1: MC-PILCO

---

**init** policy  $\pi_{\theta}(\cdot)$ , cost  $c(\cdot)$ , kernel  $k(\cdot, \cdot)$ , maximum optimization steps  $N_{opt}$ , number of particles  $M$ , learning rate  $\alpha_{lr}$ , min. learning rate  $\alpha_{lr_{min}}$ , dropout probability  $p_d$ , dropout probability reduction  $\Delta p_d$  and other monitoring signal parameters:  $\sigma_s$ ,  $\lambda_s$ ,  $n_s$ .

Apply random control to system and collect data

**while** task not learned **do**

**1) Model Learning:**

        Learn GP models from sampled data - Sec. III-A;

**2) Policy Update:**

        Initialize monitoring signal  $s_0 = 0$ ;

**for**  $j = 1 \dots N_{opt}$  **do**

        Simulate  $M$  particles rollouts with GP models and current policy  $\pi_{\theta_j}(\cdot)$ ;

        Compute  $\hat{J}(\theta_j)$  from particles (12);

        Compute  $\nabla_{\theta} \hat{J}(\theta_j)$  through backpropagation;

$\pi_{\theta_{j+1}}(\cdot) \leftarrow$  Gradient-based policy update;

        Update monitoring signal  $s_j$  with (13);

**if** (14) is True **then**

            | Update  $p_d$ ,  $\alpha_{lr}$  and  $\sigma_s$  with (15);

**end**

**if** (16) is False **then**

            | **break**;

**end**

**end**

**3) Policy Execution:**

        apply updated policy to system and collect data

**end**

**return** final policy  $\pi_{\theta^*}(\cdot)$ , learned GP model;

---

## IV. ABLATION STUDIES

In this section, we analyze several aspects affecting the performance of MC-PILCO, such as the shape of the cost function, the use of dropout, the kernel choice, and the probabilistic model adopted, namely, *full-state* or *speed-integration* dynamical model. The purpose of the analysis is to validate the choices made in the proposed algorithm, and show the effect that they have in control of dynamical systems. MC-PILCO has been implemented in Python, exploiting the PyTorch library [37] automatic differentiation functionalities; the code is publicly available<sup>1</sup>.

We considered the swing-up of a simulated cart-pole, a classical benchmark problem, to perform the ablation studies. The system and the experiments are described in the following.

<sup>1</sup>Code available at <https://www.merl.com/research/license/MC-PILCO>

The physical properties of the system are the same as the system used in PILCO [6]: the masses of both cart and pole are 0.5 [kg], the length of the pole is  $L = 0.5$  [m], and the coefficient of friction between cart and ground is 0.1. The state at each time step  $t$  is defined as  $\mathbf{x}_t = [p_t, \dot{p}_t, \theta_t, \dot{\theta}_t]$ , where  $p_t$  represents the position of the cart and  $\theta_t$  the angle of the pole. The target states corresponding to the swing-up of the pendulum is given by  $p^{des} = 0$  [m] and  $|\theta^{des}| = \pi$  [rad]. The downward stable equilibrium point is defined at  $\theta_t = 0$  [rad]. As done in [6], in order to avoid singularities due to the angles,  $\mathbf{x}_t$  is replaced in the algorithm with the state representation  $\bar{\mathbf{x}}_t = [p_t, \dot{p}_t, \theta_t, \sin(\theta_t), \cos(\theta_t)]$ . The control action is the force that pushes the cart horizontally. In all following experiments, we considered white measurement noise with standard deviation of  $10^{-2}$ , and as initial state distribution  $\mathcal{N}([0, 0, 0, 0], \text{diag}([10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}]))$ . The sampling time is 0.05 seconds. The policy is a squashed-RBF-network with  $n_b = 200$  basis functions. It receives as input  $\bar{\mathbf{x}}_t$  and  $u_{max} = 10$  [N]. The number of particles has been set to  $M = 400$  in all the tests. The exploration trajectory is obtained by applying at each time step  $t$  a random control action sampled from  $\mathcal{U}(-10, 10)$ . GP reduction techniques have not been adopted.

The cost function optimized in MC-PILCO is the following,

$$c(\mathbf{x}_t) = 1 - \exp\left(-\left(\frac{|\theta_t| - \pi}{l_\theta}\right)^2 - \left(\frac{p_t}{l_p}\right)^2\right), \quad (17)$$

where  $l_\theta$  and  $l_p$  are named lengthscales. Notice that the lengthscales define the shape of  $c(\cdot)$ , the cost function goes to its maximum value more rapidly with small lengthscales. Therefore, higher cost is associated to the same distance from the target state with lower  $l_\theta$  and  $l_p$ . The lower the lengthscale the more selective the cost function. The absolute value on  $\theta_t$  is needed to allow different swing-up solutions to both the equivalent target angles of the pole  $\pi$  and  $-\pi$ .

All the comparisons consist in a Monte Carlo study composed of 50 experiments. Every experiment is composed of 5 trials each of length 3 seconds. The random seed varies at each experiment, corresponding to different explorations and initialization of the policy, as well as different realizations of the measurement noise. The performance of the learned policies is evaluated using the cost proposed in [6],

$$c^{pilco}(\mathbf{x}_t) = 1 - \exp\left(-\frac{1}{2}\left(\frac{d_t}{0.25}\right)^2\right), \quad (18)$$

where  $d_t^2 = p_t^2 + 2p_t L \sin(\theta_t) + 2L^2(1 + \cos(\theta_t))$  is the squared distance between the tip of the pole and its position at the unstable equilibrium point with  $p_t = 0$  [m]. We introduce this cost in order to have a common metric to compare both different setups of MC-PILCO and other MBRL algorithms, see Section V-A. For each trial, we report the median value and confidence interval defined by the 5-th and 95-th percentile of the cumulative cost obtained with  $c^{pilco}(\cdot)$ , as well as the success rates observed. We mark two values of the cumulative cost indicatively associated with an optimal and a sub-optimal swing-up, respectively. A solution is optimal if the pole oscillates only once before reaching the upwards equilibrium.

A sub-optimal solution is when the pole oscillates twice. Finally, we label a trial as "success" if  $|p_t| < 0.1$  [m] and  $170$  [deg]  $< |\theta_t| < 190$  [deg]  $\forall t$  in the last second of the trial.

#### A. Cost shaping

The first test regards the performance obtained varying the lengthscales of the cost function in (17). Reward shaping is a known important aspect of RL and here we will analyze it for MCPILCO. In Figure 2, we compare the evolution of the cumulative costs obtained with  $(l_\theta = 3, l_p = 1)$  and  $(l_\theta = 0.75, l_p = 0.25)$  and we report the observed success rates. The latter set of length-scales defines a more selective cost as the function shape becomes more skewed. In both cases, we adopted the *speed-integration* model with SE kernel and no dropout was used during policy optimization. The results

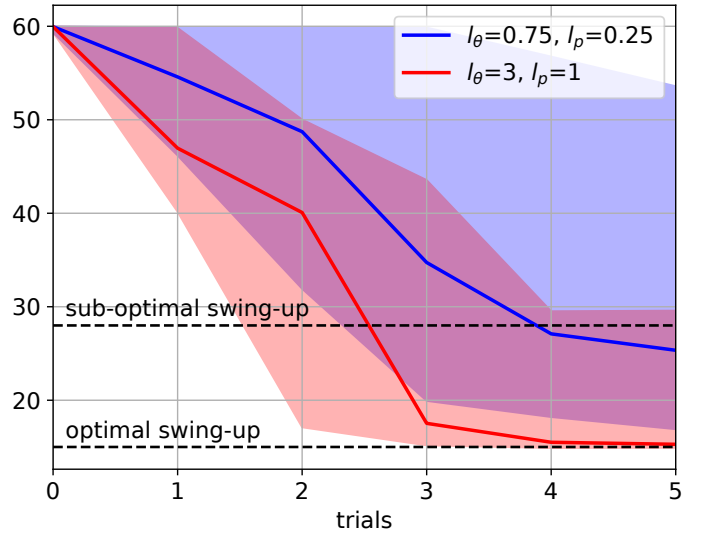


Fig. 2: Median and confidence interval of the cumulative cost  $c^{pilco}(\cdot)$  per trial obtained using  $(l_\theta = 3, l_p = 1)$  or  $(l_\theta = 0.75, l_p = 0.25)$ . In both cases, we used GP *speed-integration* models with SE kernels and no dropout was applied. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
$l=(0.75,0.25)$	0%	2%	30%	68%	74%
$l=(3,1)$	0%	12%	50%	72%	82%

show that with  $(l_\theta = 3, l_p = 1)$  MC-PILCO performs better. Indeed, with  $(l_\theta = 0.75, l_p = 0.25)$  MC-PILCO manages to find a solution only in the 74% of the experiments, while with  $(l_\theta = 3, l_p = 1)$  success rate is 82%. Observing the cumulative costs, it is possible to appreciate also a difference in the quality of the policies learned in the two cases. The optimal swing-up can be found in the majority of the experiments only when using  $(l_\theta = 3, l_p = 1)$ , while it has never been obtained with  $(l_\theta = 0.75, l_p = 0.25)$ .

This fact suggests that the use of too selective cost functions might decrease significantly the probability of converging to a solution. The reason might be that with small valued lengthscales,  $c(\mathbf{x}_t)$  is very peaked, resulting in almost null gradient, when the policy parameters are far from a good configuration, and increasing the probability of getting stuck in a local minimum. Instead, higher values of the lengthscales promote the presence of non-null gradients also far away from



the objective, facilitating the policy optimization procedure. These observations have already been made in PILCO, but they did not encounter difficulties in using a small lengthscale such as 0.25 in (18). This may be due to the analytic computation of the policy gradient made possible thanks to moment matching, as well as to the different optimization algorithm used. On the other hand, the lengthscales' values seems to have no effect on the precision of the learned solution. To confirm this, in Table II, rows 3-4, are reported the average distances from the target states obtained by successful policies at trial 5 during the last second of interaction. No significant differences in term of precision in reaching the targets can be observed.

### B. Dropout

In this test, we compared the results obtained using, or not, the dropout during policy optimization. In Figure 3, we compare the evolution of the cumulative cost obtained in the two cases and we show the obtained success rates. In both

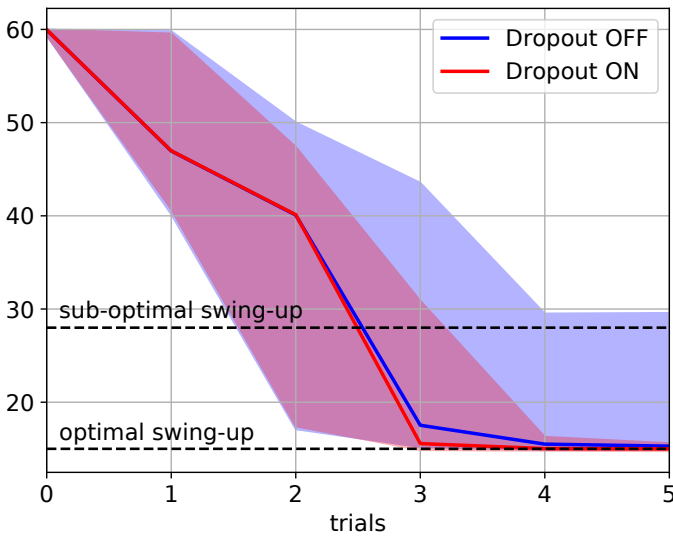


Fig. 3: Median and confidence interval of the cumulative cost  $c^{pilco}(\cdot)$  per trial obtained using, or not, dropout. In both cases, we adopted GP *speed-integration* model with SE kernels,  $l_\theta = 3$  and  $l_p = 1$ . Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Dropout OFF	0%	12%	50%	72%	82%
Dropout ON	0%	14%	78%	94%	100%

scenarios, we adopted the *speed-integration* model with SE kernel and a cost function with lengthscales ( $l_\theta = 3, l_p = 1$ ). When using dropout, MC-PILCO learned the optimal solution at trial 4 in the 94% of the experiments, and it managed to obtain it for all the random seeds by trial 5. Instead, without dropout, the optimal policy has not always been found, even in the last trial. Notice that, when dropout is not used, the upper bounds of the cumulative costs in the last two trials are higher, and the task cannot be always solved. Additionally, Table II, rows 2-4, shows that using dropout also helps in decreasing the cart positioning error at the end of the swing-up (in terms of both mean and standard deviation).

Empirically, we found that dropout not only helps in stabilizing the learning process and in finding better solutions

more consistently, but it can also improve the precision of the learned policies.

### C. Kernel function

In this test, we compared the results obtained using either the SE or the SE+P<sup>(2)</sup> kernel. In both cases, we adopted a *speed-integration* model, the cost function was defined with lengthscales ( $l_\theta = 3, l_p = 1$ ), and dropout was used. Figure 4 shows that SE+P<sup>(2)</sup> is faster to converge to the optimal solution than SE. With the SE+P<sup>(2)</sup> kernel, the algorithm learns the task at trial 3 in 90% of the cases, and it obtains a success rate of 100% at trial 4. On the other hand, when using the SE kernel, only at trial 5 the task is solved for all the random seeds. This can be explained by the capacity of a more structured kernel to learn a correct enough dynamics of the system also in areas of the state-action space with no available data points. In fact, some parts of the dynamics of the cart-pole system are polynomial functions of the GP input  $\tilde{x}_t = (\tilde{x}_t, \mathbf{u}_t)$  and the structure of the SE+P<sup>(2)</sup> improves the data-efficiency of the model learning.

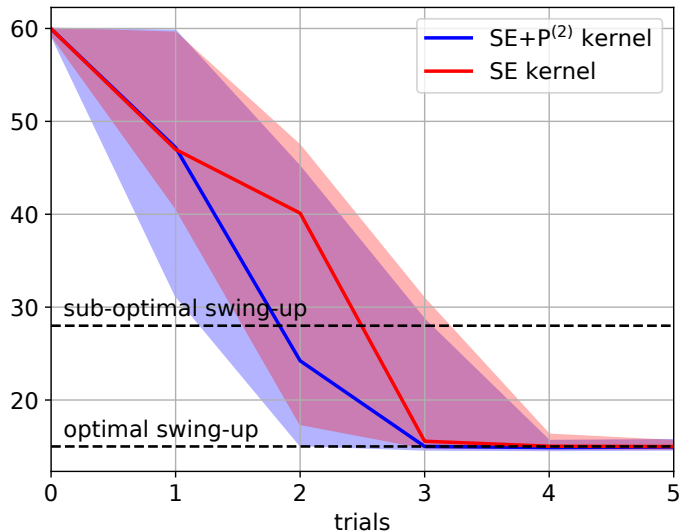


Fig. 4: Median and confidence interval of the cumulative cost  $c^{pilco}(\cdot)$  per trial obtained using GP *speed-integration* model with kernel SE or SE+P<sup>(2)</sup>. In both cases,  $l_\theta = 3, l_p = 1$ , and dropout was used. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
SE	0%	14%	78%	94%	100%
SE+P <sup>(2)</sup>	2%	42%	90%	100%	100%

### D. Speed-integration model

In this test, we compared the performance obtained by the proposed *speed-integration* dynamical model and by the standard *full-state* model. In both cases, SE kernels were chosen, the cost function was defined with lengthscales ( $l_\theta = 3, l_p = 1$ ), and dropout was used. Figure 5 shows that the *speed-integration* model obtains better performance at trials 2 and 3, with narrower confidence intervals and better success rates. On the contrary, during the last two trials, the success rate of the *full-state* model is slightly better. Recall, that in the *full-state*

model, positions and velocities are learned independently, while, in the *speed-integration* model, the position is computed as the integral of the velocity under a constant acceleration assumption. Then, the *speed-integration* model may reduce uncertainty in long-term predictions and facilitate the learning w.r.t. the counterpart when few data points have been collected. In fact, the *full-state* model could face some difficulties in learning the connection between positions and respective velocities from a limited amount of data. This reduction of the uncertainty may explain the narrower confidence intervals observed during the first trials of the experiment. On the other hand, when enough data points have been collected (trials 4 and 5), the improvement in precision obtained by *full-state* model is not very significant. Even with comparable performance, the choice of the *speed-integration* model is justified since it halves the number of GPs to learn, hence this structure is also improving the computational time.

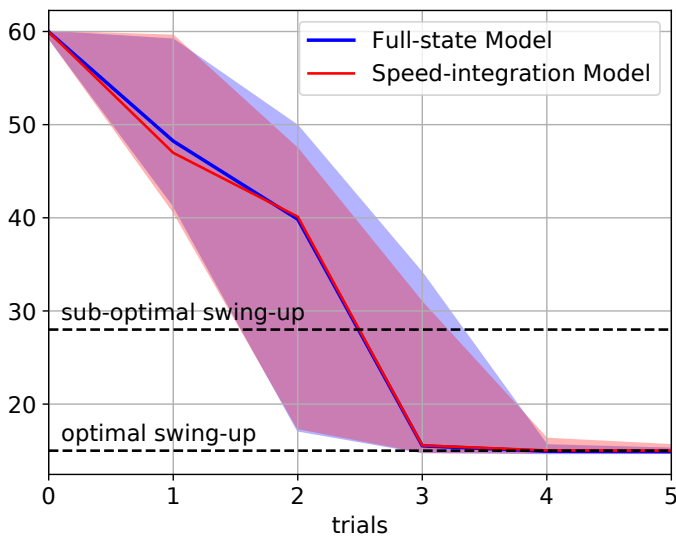


Fig. 5: Median and confidence interval of the cumulative cost  $c^{pilco}(\cdot)$  per trial obtained using *full-state* or *speed-integration* dynamical model. The kernels are in both cases SE,  $l_\theta = 3$ ,  $l_p = 1$  and dropout was used. Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Full-state	0%	10%	72%	98%	100%
Speed-int.	0%	14%	78%	94%	100%

## V. EXPERIMENTS IN SIMULATION

In this section, two simulated systems are considered. First, MC-PILCO is tested on a cart-pole system and compared to other policy gradient algorithms, namely PILCO and Black-DROPS. In the same environment, we tested the capability of MC-PILCO to handle bimodal probability distributions. Second, MC-PILCO learns a controller in joint space of a UR5 robot arm, considered as an example of a higher DoF system.

### A. Cart-pole: comparison with other methods

We tested PILCO<sup>2</sup>, Black-DROPS<sup>3</sup>, and MC-PILCO on the cart-pole system with the same setup described in Section IV.

<sup>2</sup>PILCO code available at <http://mlg.eng.cam.ac.uk/pilco>

<sup>3</sup>Black-DROPS code available at <https://github.com/resibots/blackdrops>

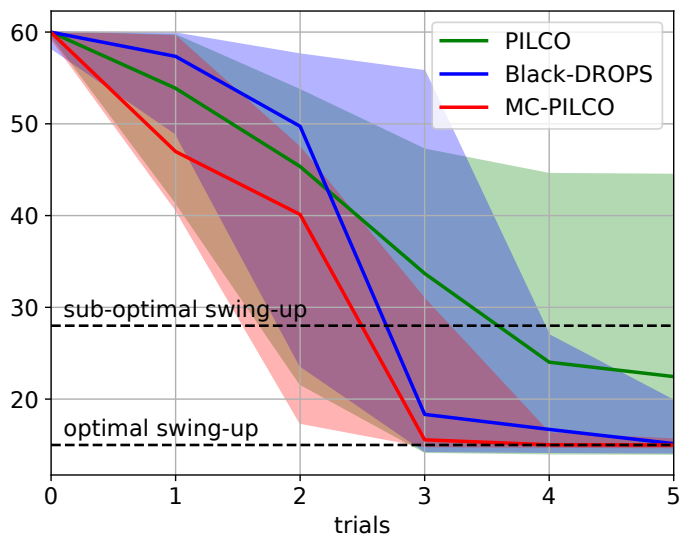


Fig. 6: Median and confidence interval of the cumulative cost  $c^{pilco}(\cdot)$  per trial obtained with PILCO, Black-DROPS and MC-PILCO (with *speed-integration* model, SE kernel, dropout activated,  $l_\theta = 3$  and  $l_p = 1$ ). Success rates are reported below.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
PILCO	2%	4%	20%	36%	42%
Black-DROPS	0%	4%	30%	68%	86%
MC-PILCO	0%	14%	78%	94%	100%

In MC-PILCO, we considered the cost function (17) with lengthscales ( $l_\theta = 3, l_p = 1$ ) and SE kernels, in order to have the same kernel functions in all the three algorithms. Results of the cumulative cost are reported in Figure 6. MC-PILCO achieved the best performance both in transitory and at convergence, by trial 5, it learned how to swing up the cart-pole with a success rate of 100%. In each and every trial, MC-PILCO obtained cumulative costs with lower median and less variability. On the other hand, the policy in PILCO showed poor convergence properties with only 42% of success rate after all the 5 trials. Black-DROPS outperforms PILCO, but it obtained worse results than MC-PILCO in each and every trial, with a success rate of only 86% at trial 5. Recall that MC-PILCO obtains even better performance when considering SE+P<sup>(2)</sup> kernel. Results in Table II, rows 1-2-6-7, also show that policies learned with MC-PILCO are more precise in reaching the target.

### B. Cart-pole: handling bimodal distributions

One of the main advantages of particle-based policy optimization is the capability to handle multimodal state evolutions. This is not possible when applying methods based on moment matching, such as PILCO. We verified this advantage by applying both PILCO and MC-PILCO to the simulated cart-pole system, when considering a very high variance on the initial cart position,  $\sigma_p^2 = 0.5$ , which corresponds to have unknown cart's initial position (but limited within a reasonable range). The aim is to be in a situation in which the policy has to solve the task regardless of the initial conditions and needs to have a bimodal behaviour in order to be optimal. Note that the situation described could be relevant in several real applications. We kept

		$e_p$ [m]	$e_\theta$ [rad]
1	S.I. SE+P <sup>(2)</sup> (3,1) drop. on	0.008 ± 0.011	0.011 ± 0.013
2	S.I. SE (3,1) drop. on	0.009 ± 0.016	0.011 ± 0.013
3	S.I. SE (0.75,0.25) drop. off	0.018 ± 0.040	0.013 ± 0.025
4	S.I. SE (3,1) drop. off	0.020 ± 0.045	0.013 ± 0.025
5	F.S. SE (3,1) drop. on	0.010 ± 0.017	0.011 ± 0.015
6	Black-DROPS	0.025 ± 0.032	0.033 ± 0.058
7	PILCO	0.027 ± 0.036	0.045 ± 0.057

TABLE II: Average distances from the target states ( $p_t = 0$  and  $\theta_t = \pm\pi$ ) obtained during the last second of interaction with the cart-pole by the successful policies learned by PILCO, Black-DROPS and the various MC-PILCO configurations analyzed in Section IV. Different configurations are labeled reporting the adopted dynamical model structure (*speed-integration*, S.I., or *full-state*, F.S.), kernel function, cost lengthscales, and if dropout was used or not. Values are reported as mean  $\pm 3 \times$  standard deviation, calculated over the total number of successful runs at trial 5.

the same setup used in previous cart-pole experiments, changing the initial state distribution to a zero mean Gaussian with covariance matrix  $\text{diag}([0.5, 10^{-4}, 10^{-4}, 10^{-4}])$ . MC-PILCO optimizes the cost in (17) with lengthscales ( $l_\theta = 3, l_p = 1$ ). We tested the policies learned by the two algorithms starting from nine different cart initial positions (-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2 [m]). In Section V-A, we observed that PILCO struggles to consistently converge to a solution and the high variance in the initial conditions accentuates this issue. Nevertheless, in order to make the comparison possible, we cherry-picked a random seed for which PILCO converged to a solution in this particular scenario. In Figure 7, we show the results of the experiment. MC-PILCO is able to handle the initial high variance. It learned a bimodal policy that pushes the cart in two opposite directions, depending on the cart’s initial position, and stabilizes the system in all the experiments. On the contrary, PILCO’s policy is not able to control the cart-pole for all the tested starting conditions. Its strategy is always to push the cart in the same direction, and it cannot stabilize the system when the cart starts far away from the zero position. The state evolution under MC-PILCO’s policy is bimodal, while PILCO cannot find this type of solutions because of the unimodal approximation enforced by moment matching.

In this example, we have seen that a multimodal state evolution could be the optimal solution, when starting from a unimodal state distribution with high variance, due to dependencies on initial conditions. In other cases, multimodality could be directly enforced by the presence of multiple possible initial conditions that would be badly modeled with a single unimodal distribution. MC-PILCO can handle all these situations thanks to its particle-based method for long-term predictions. Similar results were obtained when considering bimodal initial distributions.

### C. UR5 joint-space controller: high DoF application

The objective of this experiment is to test MC-PILCO in a more complex system with higher DoF. We used MC-PILCO to learn a joint-space controller for a UR5 robotic arm (6 DoF) simulated in MuJoCo [38]. Let the state at time  $t$  be  $x_t = [q_t, \dot{q}_t]^T$ , where  $q_t, \dot{q}_t \in \mathbb{R}^6$  are joint angles and

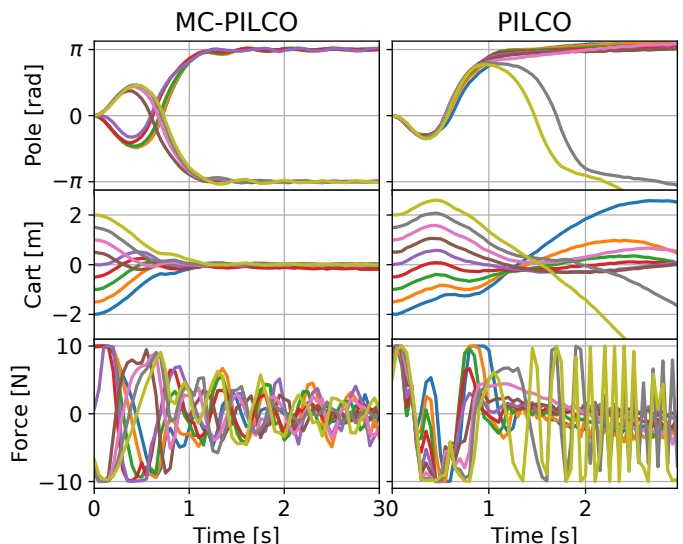


Fig. 7: (Left) MC-PILCO policy applied to the cart-pole system starting from nine different sparse cart initial positions, namely: -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2 [m]. The policy is able to complete the task in all cases, pushing the cart in different directions depending on its initial condition. (Right) PILCO policy applied starting from the same cart initial positions. This policy struggles to adapt to different starting conditions, and it cannot swing up the cart-pole when starting from the initial positions further away from zero.

velocities, respectively. The objective for the policy  $\pi_\theta$  is to control the torques  $\tau_t$  in order to follow a desired trajectory  $(\bar{q}_t^r, \bar{\dot{q}}_t^r)$  for  $t = 1, \dots, T$ . Let  $e_t = \bar{q}_t^r - q_t, \dot{e}_t = \bar{\dot{q}}_t^r - \dot{q}_t$  be position and velocity errors at time  $t$ , respectively. The policy is a multi-output squashed-RBF-network defined in (11) with  $n_b = 400$  Gaussian basis functions and  $u_{max} = 1$  [N·m] for all the joints, that maps states and errors into torques,  $\pi_\theta : q_t, \dot{q}_t, e_t, \dot{e}_t \mapsto \tau_t$ . The control scheme is represented in Figure 8. In this experiment, we considered a control

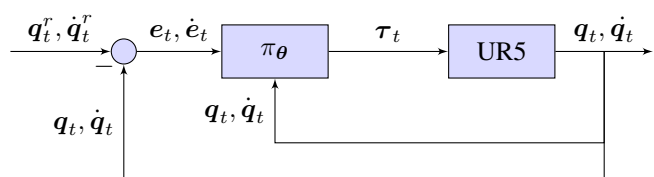


Fig. 8: Joint-space control scheme for UR5 robotic arm.

horizon of 4 seconds with a sampling time of 0.02 seconds. The reference trajectory has been calculated to make the end-effector to draw a circle in the X-Y operational space. The initial exploration, used to initialize the *speed-integration* dynamical model, is provided by a poorly-tuned PD controller. We used SE+P<sup>(1)</sup> kernels in the GP dynamical model. The GP reduction thresholds were set to  $10^{-3}$ . GP input was built using extended state  $\bar{x}_t = [\dot{q}_t, \sin(q_t), \cos(q_t)]$ .  $M = 200$  is the number of particles used for gradient estimation. The cost function considered is defined as,

$$c(x_t) = 1 - \exp \left( - \left( \frac{\|q_t^r - q_t\|}{0.5} \right)^2 - \left( \frac{\|\dot{q}_t^r - \dot{q}_t\|}{1} \right)^2 \right).$$

We assumed full state observability with measurements perturbed by a white noise with standard deviation of  $10^{-3}$ . The initial state distribution is a Gaussian centered on  $(\mathbf{q}_0^T, \dot{\mathbf{q}}_0^T)$  with standard deviation of  $10^{-3}$ . Policy optimization parameters are the same reported in Table I, with the exception of  $n_s = 400$  and  $\delta_s = 0.05$ , to enforce more restrictive exit conditions.

In Figure 9, we report the trajectory followed by the end-effector at each trial, together with the desired trajectory. MC-PILCO considerably improved the high tracking error obtained with the PD controller after only 2 trials (corresponding to 8 seconds of interaction with the system). The learned control policy followed the reference trajectory for the end-effector with a mean error of  $0.65 \pm 0.69$  [mm] (confidence interval computed as  $3 \times$  standard deviation), and a maximum error of 1.08 [mm].

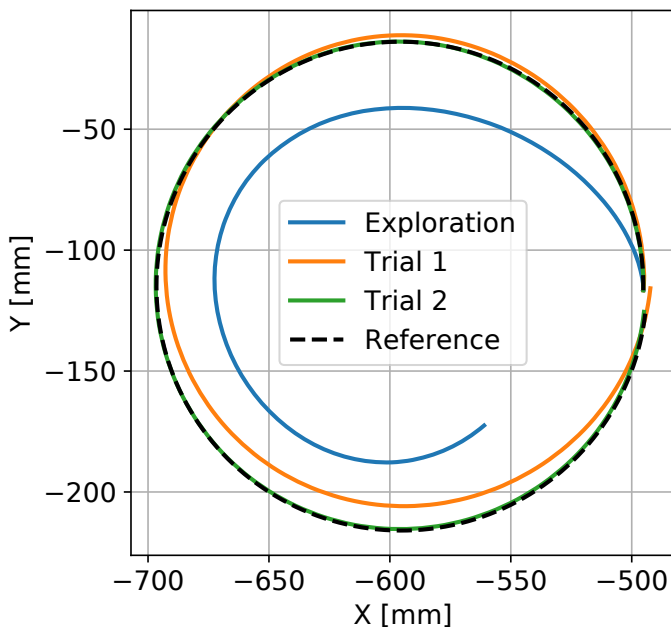


Fig. 9: End-effector trajectories obtained in exploration and for each trial of policy learning together with the desired circle. Let  $e_{ee}$  be the error between the desired and the actual end-effector trajectories. In the table below, we report, in millimeters, the maximum and mean errors ( $\pm 3 \times$  standard deviation) at each trial.

	Exploration	Trial 1	Trial 2
mean( $e_{ee}$ ) [mm]	$140.66 \pm 158.94$	$21.15 \pm 41.71$	$0.65 \pm 0.69$
max( $e_{ee}$ ) [mm]	196.70	40.79	1.08

## VI. MC-PILCO FOR PARTIALLY MEASURABLE SYSTEMS

In this section, we discuss the application of MC-PILCO to systems where the state is partially measurable, i.e., systems whose state is observable, but only some components of the state can be directly measured, while the rest must be estimated from measurements. For simplicity, we introduce the problem discussing the case of mechanical system where only positions (and not velocities) can be measured, but similar considerations can be done for any partially measurable system with observable state. Then, we describe MC-PILCO for partially measurable systems (MC-PILCO4PMS), a modified version of MC-PILCO, proposed to deal with such setups. The algorithm is validated in simulation as a proof of concept.

### A. MC-PILCO4PMS

Consider a mechanical systems where only joint positions can be measured. This can be described as a partially measurable system, where in the state  $\mathbf{x}_t = [\mathbf{q}_t^T, \dot{\mathbf{q}}_t^T]^T$  only  $\mathbf{q}_t$  is measured. Consequently, the  $\dot{\mathbf{q}}_t$  elements are estimated starting from the history of  $\mathbf{q}_t$  measurements through proper estimation procedures, possibly performing also denoising operations of  $\mathbf{q}_t$  in case that the measurement noise is high. In particular, it is worth distinguishing between estimates computed online and estimates computed offline. The former are provided to the control policy to determine the system control input, and they need to respect real-time constraints, namely, velocity estimates are causal and computations must be performed within a given interval. The latter, do not have to deal with such constraints. As a consequence, offline estimates can be more accurate, taking into account acausal information and limiting delays and distortions.

In this context, we verified that, during policy optimization, it is relevant to distinguish between the particle state predictions computed by the models and the data provided to the policy. Indeed, GPs should simulate the real system dynamics, independently of additional noise given by the sensing instrumentation, so they need to work with the most accurate estimates available; delays and distortions might compromise the accuracy of long-term predictions. On the other hand, providing to the policy directly the particles' states computed with the GPs during policy optimization, correspond to train the policy assuming available access directly to the system state, which, as mentioned before, is not possible in the setup considered. Indeed, considerable discrepancies between the particles' states and the state estimates computed online during the policy application to the real system might compromise the effectiveness of the policy. This approach differentiate from standard MBRL approaches where, typically, the effects of the online state estimators are not considered during training.

To deal with the above issues, we introduce MC-PILCO4PMS a modified version of MC-PILCO. With respect to the algorithm described in Section III, we propose the two following additions.

1) *Computation of the GPs training data:* We compute the state estimates used to train the GP models with offline estimation techniques. In particular, in our real experiments, we considered two options

- Computation of the velocities with the central difference formula, i.e.,  $\dot{\mathbf{q}}_t = (\mathbf{q}_{t+1} - \mathbf{q}_{t-1}) / (2T_s)$ , where  $T_s$  is the sampling time. This technique can be used only when the measurement noise is limited, otherwise the  $\dot{\mathbf{q}}$  estimates might be too noisy.
- Estimation of the state with a Kalman smoother [39], with state-space model given by the general equations relating positions, velocities, and accelerations. The advantage of this technique is that it exploits the correlation between positions and velocities, increasing regularization.

2) *Simulation of the online estimators:* During policy optimization, instead of simulating only the evolution of the particles states, we simulate also the measurement system and

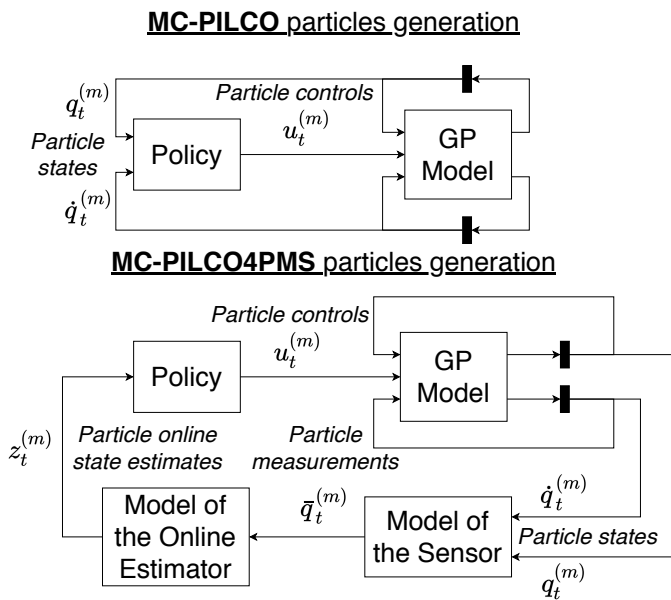


Fig. 10: Block schemes illustrating particles generation in MC-PILCO and MC-PILCO4PMS.

the online estimators. The state feed to the policy, denoted  $z_t$ , is computed to resemble the state that will be estimated online. Given the  $m$ -th particle, this is:

$$z_t^{(m)} = f_z \left( \bar{q}_t^{(m)} \dots \bar{q}_{t-m_q}^{(m)}, z_{t-1}^{(m)} \dots z_{t-1-m_z}^{(m)} \right),$$

where  $f_z$  denotes the online state estimator, with memory  $m_q$  and  $m_z$ , and  $\bar{q}_t^{(m)}$  is a fictitious noisy measurement of the  $m$ -th particle positions. More precisely, let  $q_t^{(m)}$  the positions of the  $x_t^{(m)}$  particle state, then, we have

$$\bar{q}_t^{(m)} = q_t^{(m)} + e_t^{(m)}, \quad (19)$$

where  $e_t^{(m)} \in \mathbb{R}^{d_x/2}$  is Gaussian i.i.d. noise with zero mean and covariance  $\text{diag}([\sigma_z^{(1)} \dots \sigma_z^{(d_x/2)}])$ . The  $\sigma_z^{(i)}$ s values must be tuned in accordance with the properties of the measurement system, e.g., the accuracy of the encoder. Then, the control input of the  $m$ -th particle is computed as  $\pi_\theta(z_t^{(m)})$ , instead than  $\pi_\theta(x_t^{(m)})$ . Differences in particles generation between MC-PILCO and MC-PILCO4PMS are summed up in the block scheme reported in Figure 10.

### B. Simulation results

Here, we test the relevance of modeling the presence of online estimators using the simulated cart-pole system, but adding assumptions that emulate a real world experiment. We considered the same physical parameters and the same initial conditions described in Section IV, but assuming to measure only the cart position and the pole angle. We modeled a possible measurement system that we would have in the real world as an additive Gaussian i.i.d. noise with standard deviation  $3 \cdot 10^{-3}$ . In order to obtain reliable estimates of the velocities, samples were collected at 30 [Hz]. The online estimates of the velocities were computed by means of causal numerical differentiation followed by a first order low-pass filter, with

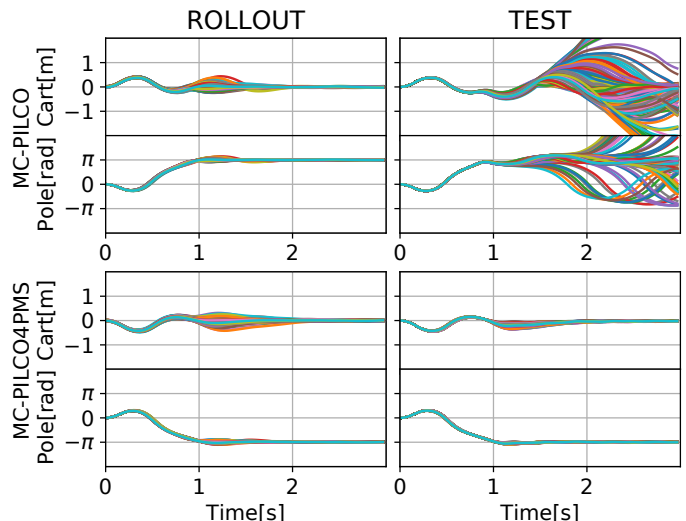


Fig. 11: Comparison of 400 simulated particles rollout (left) and the trajectories performed applying repetitively the policy 400 times in the system (right) with the simulated cart-pole system. Results obtained without simulating online filtering are on the top plots, while the ones obtained considering the low-pass filters are on the bottom. The plots refer to the policy learned after 5 trials with the system.

cutoff frequency 7.5 [Hz]. The velocities used to train the GPs were derived with the central difference formula. To verify the effectiveness of the strategy described in Section VI-A, two policy functions were trained. The first policy is obtained with MC-PILCO by neglecting the presence of online filtering during policy optimization and assuming direct access to the state predicted by the model. On the contrary, the second policy is trained with MC-PILCO4PMS, which models the presence of the online estimators as described in Section VI-A2. Exploration data were collected with a random policy. To avoid dependencies on initial conditions, such as policy initialization and exploration data, we fixed the same random seed in both experiments. In Figure 11, we report the results of a Monte Carlo study with 400 runs. On the left, the final policy is applied to the learned models (ROLLOUT) and on the right to the cartpole system (TEST). Even though the two policies perform similarly when applied to the models, which is all can be tested offline, the results obtained by testing the policies in the cartpole system are significantly different. The policy optimized with modeling the presence of online filtering solves the task in all 400 attempts. In contrast, in several attempts, the first policy does not solve the task, due to delays and discrepancies introduced by the online filter and not considered during policy optimization. We believe that these considerations on how to manipulate the data during model learning and policy optimization might be beneficial for other MBRL algorithms different from MC-PILCO.

## VII. EXPERIMENTS WITH REAL SYSTEMS

In this section, we test MC-PILCO4PMS when applied to real systems, considering the concepts introduced in Section VI. In particular, we experimented on two benchmark systems: a Furuta pendulum, and a ball-and-plate (Figure 12)<sup>4</sup>.

<sup>4</sup>A video of these experiments is available at <https://youtu.be/-73hmZYaHA>.

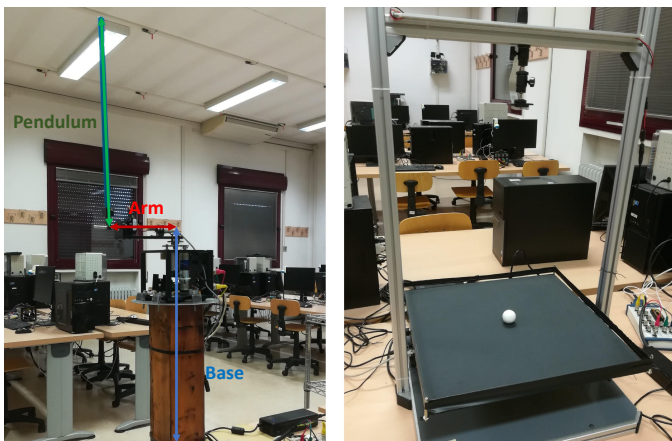


Fig. 12: (Left) Furuta pendulum used in the experiment while being controlled in the upward equilibrium point by the learned policy. (Right) Ball-and-plate system.

### A. Furuta Pendulum

The Furuta pendulum (FP) [40] is a popular benchmark system used in nonlinear control and reinforcement learning. The system is composed of two revolute joints and three links. The first link, called the base link, is fixed and perpendicular to the ground. The second link, called arm, rotates parallel to the ground, while the rotation axis of the last link, the pendulum, is parallel to the principal axis of the second link, see Figure 12. The FP is an under-actuated system as only the first joint is actuated. In particular, in the FP considered the horizontal joint is actuated by a DC servomotor, and the two angles are measured by optical encoders with 4096 [ppr]. The control variable is the motor voltage. Let the state at time step  $t$  be  $\mathbf{x}_t = [\theta_t^h, \dot{\theta}_t^h, \theta_t^v, \dot{\theta}_t^v]^T$ , where  $\theta_t^h$  is the angle of the horizontal joint and  $\theta_t^v$  the angle of the vertical joint attached to the pendulum. The objective is to learn a controller able to swing-up the pendulum and stabilize it in the upwards equilibrium ( $\theta_t^v = \pm\pi$  [rad]) with  $\theta_t^h = 0$  [rad]. The trial length is 3 seconds with a sampling frequency of 30 [Hz]. The cost function is defined as

$$c(\mathbf{x}_t) = 1 - \exp\left(-\left(\frac{\theta_t^h}{2}\right)^2 - \left(\frac{|\theta_t^v| - \pi}{2}\right)^2\right) + c_b(\mathbf{x}_t), \quad (20)$$

with

$$c_b(\mathbf{x}_t) = \frac{1}{1 + \exp\left(-10\left(-\frac{3}{4}\pi - \theta_t^h\right)\right)} + \frac{1}{1 + \exp\left(-10\left(\theta_t^h - \frac{3}{4}\pi\right)\right)}.$$

The first part of the function in (20) aims at driving the two angles towards  $\theta_t^h = 0$  and  $\theta_t^v = \pm\pi$ , while  $c_b(\mathbf{x}_t)$  penalizes solutions where  $\theta_t^h \leq -\frac{3}{4}\pi$  or  $\theta_t^h \geq \frac{3}{4}\pi$ . We set those boundaries to avoid the risk of damaging the system if the horizontal joint rotates too much. Offline estimates of velocities for the GP model have been computed by means of central differences. For the online estimation, we used causal numerical differentiation:  $\dot{\mathbf{q}}_t = (\mathbf{q}_t - \mathbf{q}_{t-1})/(T_s)$ , where  $T_s$  is the sampling time. Instead of  $\mathbf{x}_t$ , we considered the extended

state  $\bar{\mathbf{x}}_t = [\dot{\theta}_t^h, \dot{\theta}_t^v, \sin(\theta_t^h), \cos(\theta_t^h), \sin(\theta_t^v), \cos(\theta_t^v)]^T$  in GP input. The policy is a squashed-RBF-network with  $n_b = 200$  basis functions that receives as input  $\bar{\mathbf{z}}_t = [(\theta_t^h - \theta_{t-1}^h)/T_s, (\theta_t^v - \theta_{t-1}^v)/T_s, \sin(\theta_t^h), \cos(\theta_t^h), \sin(\theta_t^v), \cos(\theta_t^v)]^T$ . We used 400 particles to estimate the policy gradient from model predictions. The exploration trajectory has been obtained using as input a sum of ten sine waves of random frequencies and same amplitudes. The initial state distribution is assumed to be  $\mathcal{N}([0, 0, 0, 0]^T, \text{diag}([5 \cdot 10^{-3}, 5 \cdot 10^{-3}, 5 \cdot 10^{-3}, 5 \cdot 10^{-3}]))$ .  $M = 400$  particles were used for gradient estimation, and the GP reduction thresholds were set to  $10^{-3}$ . We solved the task using the three different choices of kernel functions described in Section III-A2: squared exponential (SE), squared exponential + polynomial of degree  $d$  (SE+P<sup>(d)</sup>) and semi-parametrical (SP). In Figure 13, we show the resulting trajectories for each trial. MC-PILCO4PMS managed to learn how to swing up the Furuta

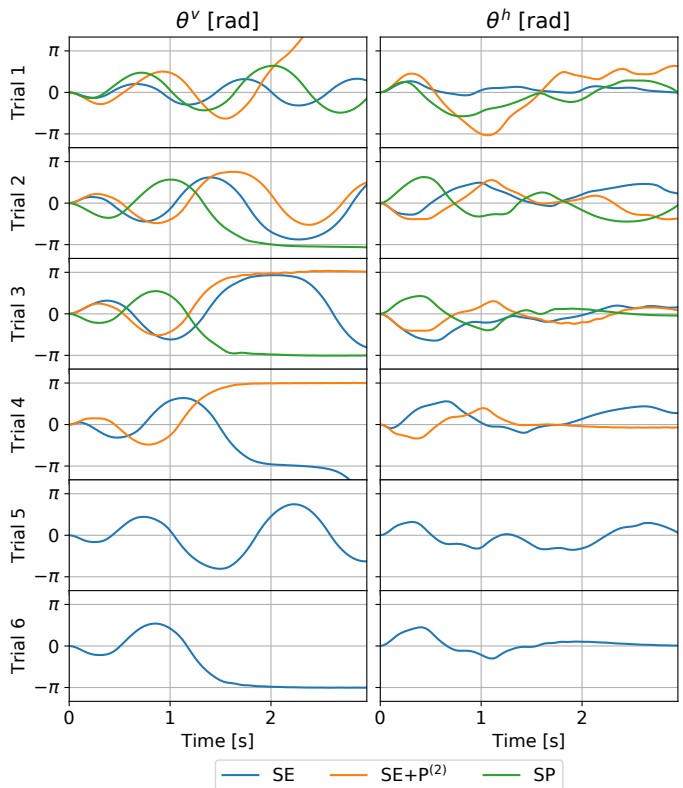


Fig. 13: (Left) Pendulum angle's trajectories for each trial. (Right) Horizontal joint angle's trajectories for each trial. For all the kernels, the angles are plotted up to the trial that solved the task.

pendulum in all cases. It succeeded at trial 6 with kernel SE, at trial 4 with kernel SE+P<sup>(2)</sup>, and at trial 3 with SP kernel. These experimental results confirm the higher data efficiency of more structured kernels and the advantage that MC-PILCO4PMS offers by allowing any kind of kernel function. Moreover, we can observe the effectiveness of the cost function (20) in keeping  $\theta_t^h$  always inside the desired boundaries in all the trials and for any kernel tested. Considering penalties similar to  $c_b(\mathbf{x}_t)$  inside the cost function could be enough to handle soft constraints also in other scenarios.

## B. Ball-and-Plate

The ball-and-plate system is composed of a square plate that can be tilted in two orthogonal directions by means of two motors. On top of it, there is a camera to track the ball and measure its position on the plate. Let  $(b_t^x, b_t^y)$  be the position of the center of the ball along X-axis and Y-axis, while  $\theta_t^{(1)}$  and  $\theta_t^{(2)}$  are the angles of the two motors tilting the plate, at time  $t$ . So, the state of the system is defined as  $\mathbf{x}_t = [b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, \theta_t^{(1)}, \theta_t^{(2)}, \dot{\theta}_t^{(1)}, \dot{\theta}_t^{(2)}]^T$ . The drivers of the motors allow only position control, and do not provide feedback about the motors angles. To keep track of the motor angles, we defined the control actions as the difference between two consecutive reference values sent to the motor controllers, and we limited the maximum input to a sufficiently small value, such that the motor controllers are able to reach the target angle within the sampling time. Then, in first approximation, the reference angles and the motor angles coincide, and we have  $u_t^{(1)} = \theta_{t+1}^{(1)} - \theta_t^{(1)}$  and  $u_t^{(2)} = \theta_{t+1}^{(2)} - \theta_t^{(2)}$ . The objective of the experiment is to learn how to control the motor angles in order to stabilize the ball around the center of the plate. Notice that the control task, with the given definition of inputs, is particularly difficult because the policy must learn to act in advance, and not only react to changes in the ball position. The cost function is defined as

$$c(\mathbf{x}_t) = 1 - \exp(-g_t(\mathbf{x}_t)), \quad \text{with}$$

$$g_t(\mathbf{x}_t) = \left(\frac{b_t^x}{0.15}\right)^2 + \left(\frac{b_t^y}{0.15}\right)^2 + \left(\theta_t^{(1)}\right)^2 + \left(\theta_t^{(2)}\right)^2.$$

The trial length is 3 seconds, with a sampling frequency of 30 [Hz]. Measurements provided by the camera are very noisy, and cannot be used directly to estimate velocities from positions. We used a Kalman smoother for the offline filtering of ball positions  $b_t^x, b_t^y$  and associated velocities  $\dot{b}_t^x, \dot{b}_t^y$ . In the control loop, instead, we used a Kalman filter [41] to estimate online the ball state from noisy measures of positions. When simulating the online estimator during policy optimization, we tried both to perturb and to not perturb the positions of the predicted particles with some additive noise. We obtained similar performance in the two cases, this result may be due to the fact that the Kalman filter is able to effectively filter out the white noise added to particles. Concerning the model, we need to learn only two GPs predicting the evolution of the ball velocity because we directly control motor angles, hence, their evolution is assumed deterministic. GP inputs,  $\tilde{\mathbf{x}}_t = [\tilde{\mathbf{x}}_t, u_t]$ , include an extended version of the state,  $\tilde{\mathbf{x}}_t = [b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, \sin(\theta_t^{(1)}), \cos(\theta_t^{(1)}), \sin(\theta_t^{(2)}), \cos(\theta_t^{(2)}), (\theta_t^{(1)} - \theta_{t-1}^{(1)})/T_s, (\theta_t^{(2)} - \theta_{t-1}^{(2)})/T_s]^T$  where angles have been replaced by their sines and cosines, and motor angular velocities have been estimated with causal numerical differentiation ( $T_s$  is the sampling time). The SE+P<sup>(1)</sup> kernel (10) is used, where the linear kernel acts only on a subset of the model inputs,  $\tilde{\mathbf{x}}_t^{lin} = [\sin(\theta_t^{(1)}), \sin(\theta_t^{(2)}), \cos(\theta_t^{(1)}), \cos(\theta_t^{(2)}), u_t]$ . The GP reduction threshold is reduced to  $10^{-4}$  because of the small distances that the ball can cover in one time step. We considered  $M = 400$  particles for policy gradient estimation. The policy is a multi-output RBF network (11),

with  $n_b = 400$  basis functions, that receives as inputs the estimates of  $(b_t^x, b_t^y, \dot{b}_t^x, \dot{b}_t^y, \theta_t^{(1)}, \theta_{t-1}^{(1)}, \theta_t^{(2)}, \theta_{t-1}^{(2)})$  computed with the Kalman filter; maximum angle displacement is  $u_{max} = 4$  [deg] for both motors. The policy optimization parameters used were the same described in Table I, with the difference that we used  $\alpha_{lr} = 0.006$  as initial learning rate. The reduction of the learning rate is related to the use of small lengthscales in the cost function, that are necessary to cope with the small range of movement of the ball. For the same reason, we set also  $\alpha_{lr_{min}} = 0.0015$  and  $\sigma_s = 0.05$ . Initial exploration is given by two different trials, in which the control signals are two triangular waves perturbed by white noise. Mostly during exploration and initial trials, the ball might touch the borders of the plate. In those cases, we kept data up to the collision instant. A peculiarity of this experiment in comparison to the others seen before is a wide range of initial conditions. In fact, the ball could be positioned anywhere on the plate's surface, and the policy must control it to the center. The initial distribution of  $b_0^x$  and  $b_0^y$  is a uniform  $\mathcal{U}(-0.15, 0.15)$ , which covers almost the entire surface (the plate is a square with sides of about 0.20 [m]). For the other state components,  $\theta_t^{(1)}$  and  $\theta_t^{(2)}$ , we assumed tighter initial distributions  $\mathcal{U}(-10^{-6}, 10^{-6})$ . MC-PILCO4PMS managed to learn a policy able to control the ball around the center starting from any initial position after the third trial, 11.33 seconds of interaction with the system. We tested the learned

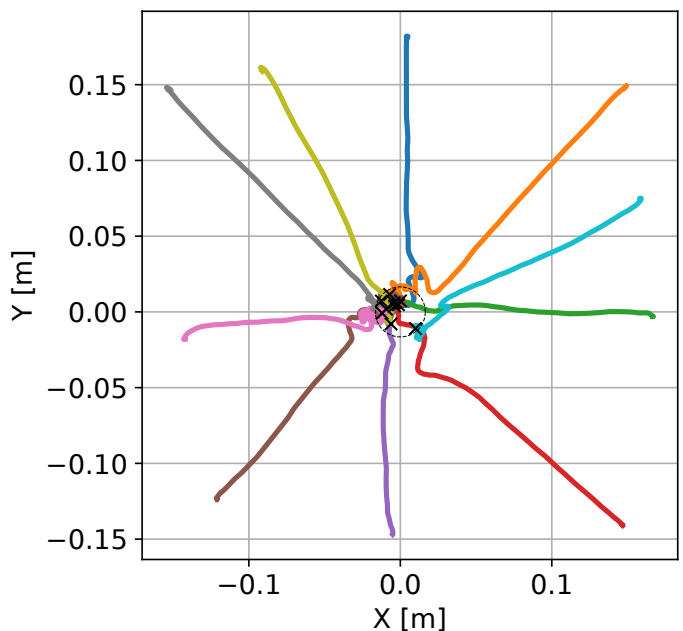


Fig. 14: Ten different ball trajectories obtained under the final policy learned by MC-PILCO4PMS. Steady-state positions are marked with black crosses. The dashed circle has the same diameter of the used ball.

policy starting from ten different points, see Figure 14. The mean steady-state error, i.e. the average distance of the final ball position from the center observed in the ten trials, was 0.0099 [m], while the maximum measured error was 0.0149 [m], which is lower than the ball radius of 0.016 [m].

## VIII. CONCLUSION

In this paper, we have presented the MBRL algorithm MC-PILCO. The proposed framework uses GPs to derive a probabilistic model of the system dynamics, and updates the policy parameters through a gradient-based optimization; the optimization exploits the *reparameterization trick* and approximates the expected cumulative cost relying on a Monte Carlo approach. Compared to similar algorithms proposed in the past, the Monte Carlo approach worked by focusing on two aspects, that are (i) proper selection of the cost function, and (ii) introduction of exploration during the policy optimization through the use of dropout. We compared MC-PILCO with PILCO and Black-DROPS, that are two state-of-the-art GP-based MBRL algorithms. MC-PILCO outperforms both the algorithms, exhibiting better data-efficiency and asymptotic performance. The results obtained in simulation confirm the effectiveness of the proposed solution, and show the relevance of the two aforementioned aspects when optimizing the policy combining the *reparameterization trick* with particles-based approaches. Moreover, we explored two advantages due to the particles-based approximation w.r.t. the moment-matching adopted in PILCO, that are, the possibility of using structured kernels, such as polynomial kernel and semiparametrical kernel, and the ability of handling multimodal distributions. In particular, results obtained in simulation and with real systems show that the use of structured kernels can increase data-efficiency, reducing the interaction-time required to learn the task. Finally, we analyzed common problems arising when trying to apply MBRL to real systems. In particular, we focused on systems with partially measurable states, which are particularly relevant in real applications. In this context, we proposed a modified algorithm called MC-PILCO4PMS, where we verified the importance of taking into account the state estimators used in the real system during policy optimization. Results have been validated in different simulated scenarios, namely a cart-pole and a robotic manipulator, and also on real systems, such as a Furuta pendulum and a ball-and-plate setup.

In future works, we are interested in testing the proposed algorithms in more challenging environments, e.g., manipulation tasks in real world environments. The issues regarding the impossibility of measuring directly the velocity states tackled in MC-PILCO4PMS could be further analyzed by considering the recently introduced "Derivative-free" framework [42]. Finally, the application to manipulation tasks will also require the introduction of safe exploration techniques and guarantees from the Safe RL state of the art [43].

## REFERENCES

- [1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [2] Christopher G Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of international conference on robotics and automation*, volume 4, pages 3557–3564. IEEE, 1997.
- [3] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.
- [4] Malte Kuss and Carl E Rasmussen. Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pages 751–758, 2004.
- [5] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.
- [6] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.
- [7] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems VII*, pages 57–64, 2011.
- [8] Marc Peter Deisenroth, Roberto Calandra, André Seyfarth, and Jan Peters. Toward fast policy search for learning legged locomotion. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1787–1792. IEEE, 2012.
- [9] A. D. Libera and R. Carli. A data-efficient geometrically inspired polynomial kernel for robot inverse dynamic. *IEEE Robotics and Automation Letters*, 5(1):24–31, 2020.
- [10] Diego Romeres, Devsh K Jha, Alberto DallaLibera, Bill Yezauris, and Daniel Nikovski. Semiparametrical gaussian processes learning of forward dynamical models for navigating in a circular maze. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3195–3202. IEEE, 2019.
- [11] Diego Romeres, Mattia Zorzi, Raffaello Camoriano, and Alessandro Chiuso. Online semi-parametric learning for inverse dynamics modeling. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2945–2950. IEEE, 2016.
- [12] D. Nguyen-Tuong and J. Peters. Using model knowledge for learning inverse dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 2677–2682, 2010.
- [13] Yarín Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 34, 2016.
- [14] David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.
- [15] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.
- [16] Mark Cutler and Jonathan P How. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2605–2612. IEEE, 2015.
- [17] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-box data-efficient policy search for robotics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51–58. IEEE, 2017.
- [18] Andrew James McHutchon et al. *Nonlinear modelling and control using Gaussian processes*. PhD thesis, Citeseer, 2015.
- [19] Andrew Y Ng and Michael I Jordan. Pegasus: A policy search method for large mdps and pomdps. *arXiv preprint arXiv:1301.3878*, 2013.
- [20] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pippis: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074, 2018.
- [21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [22] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.
- [23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.
- [24] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.
- [25] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.
- [26] Carlo Baldassi, Fabrizio Pittorino, and Riccardo Zecchina. Shaping the learning landscape in neural networks around wide flat minima. *Proceedings of the National Academy of Sciences*, 117(1):161–170, 2020.
- [27] Carlo Baldassi, Enrico M. Malatesta, and Riccardo Zecchina. Properties of the geometry of solutions and capacity of multilayer neural networks with rectified linear unit activations. *Phys. Rev. Lett.*, 123:170602, Oct 2019.



- [28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.
- [29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [30] Alberto Dalla Libera, Ruggero Carli, and Gianluigi Pillonetto. A novel multiplicative polynomial kernel for volterra series identification. *arXiv preprint arXiv:1905.07960*, 2019.
- [31] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.
- [32] J. Quinero Candela and CE. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1935–1959, December 2005.
- [33] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural Comput.*, 14(3):641–668, March 2002.
- [34] Russel E Caflisch et al. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 1998:1–49, 1998.
- [35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [36] Yarín Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org, 2016.
- [37] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Proceedings of Neural Information Processing Systems*, 2017.
- [38] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.
- [39] Garry A Einicke. Optimal and robust noncausal filter formulations. *IEEE Transactions on Signal Processing*, 54(3):1069–1077, 2006.
- [40] Benjamin Seth Cazzolato and Zebb Prime. On the dynamics of the furuta pendulum. *Journal of Control Science and Engineering*, 2011, 2011.
- [41] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.
- [42] A. Dalla Libera, D. Romeres, D. K. Jha, B. Yeramunis, and D. Nikovski. Model-based reinforcement learning for physical systems without velocity and acceleration measurements. *IEEE Robotics and Automation Letters*, 5(2):3548–3555, 2020.
- [43] Javier Garcia and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.