Data-Efficient Control Policy Search using Residual Dynamics Learning

Matteo Saveriano[1], Yuchao Yin[1], Pietro Falco[1] and Dongheui Lee[1,2]

***Abstract*— In this work, we propose a model-based and data efficient approach for reinforcement learning. The main idea of our algorithm is to combine simulated and real rollouts to efficiently find an optimal control policy. While performing rollouts on the robot, we exploit sensory data to learn a probabilistic model of the residual difference between the measured state and the state predicted by a simplified model. The simplified model can be any dynamical system, from a very accurate system to a simple, linear one. The residual difference is learned with Gaussian processes. Hence, we assume that the difference between real and simplified model is Gaussian distributed, which is less strict than assuming that the real system is Gaussian distributed. The combination of the partial model and the learned residuals is exploited to predict the real system behavior and to search for an optimal policy. Simulations and experiments show that our approach significantly reduces the number of rollouts needed to find an optimal control policy for the real system.**

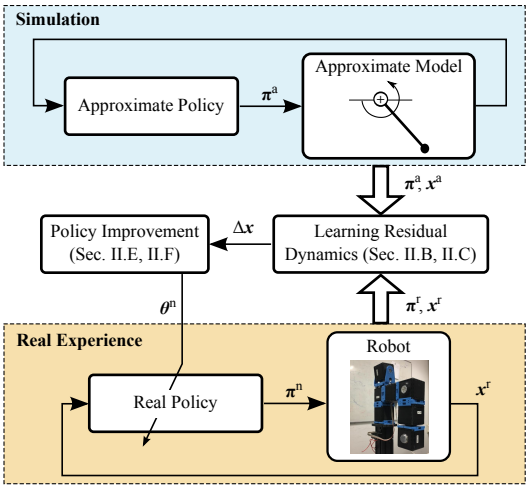


Fig. 1. Overview of the PI-REM algorithm.

I. Introduction

Robots that learn novel tasks by self-practice have the chance to be exploited in a wide range of situations, including industrial and social applications. Reinforcement Learning (RL) gives the agent the possibility to autonomously learn a task by trial and error [1]. In particular, the agent performs a number of trials (rollouts) and uses sensory data to improve the execution. In robotics and control applications, RL presents two main disadvantages [2]. First, the state and action spaces are continuous-valued and high dimensional, and existing approaches in RL to search continuous spaces are not sample-efficient. A common strategy [3]–[6] to reduce the search space to a discrete space consists in parameterizing the control policy with a discrete number of learnable parameters. Typical policy parameterizations adopt Gaussian mixture models [3], hidden Markov models [7], [8], or stable dynamical systems [5]. Second, it is extremely time consuming to perform a big number of trials (rollouts) on real devices. The rollouts can be reduced with a proper initialization of the policy, for example using kinesthetic teaching approaches [9]. However, reducing the rollouts needed to find an optimal policy is still an open problem.

RL algorithms can be classified into two categories, namely model-free and model-based approaches. Model-free methods [4]–[6] directly learn the policy on the data samples obtained after each rollout. Model-free approaches do not require any model approximation and are applicable in many situations. The problem of model-free approaches is that they may require many rollouts to find the optimal policy [4].

Model-based methods, instead, explicitly learn a model of the system to control from sensory data and use this learned model to search the optimal policy. The comparison in [10] shows that model-based approaches are efficient compared with model-free methods. However, model-based methods strongly suffer from the so-called model–bias problem. Indeed, they assume that the learned model is sufficiently accurate to represent the real system [11]–[13]. Learning an accurate model of the system is not always possible. This is the case, for instance, when the training data is too sparse. It is clear that searching for a control policy using inaccurate models might lead to poor results.

The Probabilistic Inference for Learning Control (PILCO) algorithm in [10] alleviates the model-bias problem by including uncertainty in the learned model. In particular, PILCO leverages Gaussian Processes (GP) [14] to learn a probabilistic model of the system which represents model uncertainties as Gaussian noise. The GP model allows to consider uncertainties in the long-term state predictions, and to explicitly consider eventual model errors in the policy evaluation. The comparison performed in [10] shows that PILCO outperforms state-of-the-art algorithms in terms of performed rollouts on the real robot in pendulum and cart–pole swing-up tasks. Despite the good performance, PILCO needs to learn a reliable model from sensory data to find an optimal policy. In order to explore the state space as much as possible and to rapidly learn a reliable model, PILCO applies a random policy at the first stage of the learning. In

[1]Human-centered Assistive Robotics, Technical University of Munich, Munich, Germany {matteo.saveriano, yuchao.yin, pietro.falco, dhlee}@tum.de

[2]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Germany

general, starting from a randomly initialized policy can be dangerous for real systems. The robot, in fact, can exhibit unstable behaviors in the first stages of the learning process, with the consequent risk of damages.

The work in [15] combines the benefits of model-free and model-based RL. Authors propose to model the system to control as a dynamical system and to find an initial control policy by using optimal control. The learned policy is then applied to the real system and improved using a model-free approach. The policy learned on the model represents a good starting point for policy search, significantly reducing the number of performed rollouts. The approach in [16] also uses optimal control to find an initial policy, but it uses sensory data to fit a locally linear model of the system.

In this paper, we propose a model-based RL approach that exploits an approximate, analytical model of the robot to rapidly learn an optimal control policy. The proposed Policy Improvement with REsidual Model learning (PI-REM) assumes that the system is modeled as a non-linear dynamical system. In real scenarios, it is not trivial to analytically derive the entire model, but it is reasonable to assume that part of the model is easy to derive. For instance, in a robotic application, the model of the robot in free motion has a well-known structure [17], but it is hard to model the sensor noise or the friction coefficient [18]. We refer to the known part of the model as the *approximate* model. Given the approximate model, we learn an optimal control policy in simulation. We exploit PILCO in this study, but other choices are possible. Directly applying the policy learned in simulation on a real robot does not guarantee to fulfill the task. Hence, we exploit sensory data to learn the residual difference between the real and the approximate model using Gaussian Processes (GP) [14]. The superposition of the approximate model and the learned residual term represents a good estimation of the real dynamics and it is adopted for model-based policy search. An overview of PI-REM is shown in Fig. 1.

In contrast to PILCO, our approach exploits the approximate model to be more data efficient and to converge faster to the optimal policy. Compared to the work in [15], PI-REM explicitly learns the residual difference to improve the model and speed-up the policy search. Indeed, the approximate model is globally valid and we use sensory data only to estimate local variations of that model. In contrast to [16], we do not constraint the system dynamics to be locally linear.

The rest of the paper is organized as follows. Section II describes our approach for model-based policy search. Experiments and comparisons are reported in Sec. III. Section IV states the conclusion and proposes future extensions.

II. Policy Improvement with REsidual Model learning (PI-REM)

A. Preliminaries

The goal of a reinforcement learning algorithm is to find a control policy $\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{x})$ that minimizes the expected return

$$J^{\pi}(\boldsymbol{\theta}) = \sum_{t=0}^{N} \mathbb{E}[c(\boldsymbol{x}_t)] \tag{1}$$

where $\mathbb{E}[\cdot]$ is the operator that computes the expected value of a random variable, and $c(\boldsymbol{x}_t)$ is the cost of being in state $\boldsymbol{x}$ at time t. The vector $\boldsymbol{\theta}$ represents a set of values used to parameterize the continuous-valued policy function $\boldsymbol{\pi}(\boldsymbol{x}, \boldsymbol{\theta})$.

In this work, we consider that the system to control is modeled as a non-linear dynamical system in the form

$$\boldsymbol{x}^r_{t+1} = \boldsymbol{f}^r(\boldsymbol{x}^r_t, \boldsymbol{u}^r_t) \tag{2}$$

where $\boldsymbol{x}^r \in \mathbb{R}^d$ is the continuous-valued state vector, $\boldsymbol{u}^r \in \mathbb{R}^f$ is the control input vector, $\boldsymbol{f}^r \in \mathbb{R}^d$ is a continuous and continuously differentiable function. The superscript r indicates that the dynamical system (2) is the real (exact) model of the system. The difference equation (2) can model a variety of physical systems, including robotic platforms.

B. Additive Unknown Dynamics

In general, it is not trivial to derive the exact model in (2). For example, although it is relatively simple to compute the dynamical model of a robotic manipulator in free motion, it is complicated to model eventual external forces. In many robotics applications, we can assume that the function $\boldsymbol{f}^r(\cdot,\cdot)$ in (2) consists of two terms: *i)* a deterministic and know term, and *ii)* an additive unknown term. Under this assumption, the dynamical system in (2) can be rewritten as

$$\boldsymbol{x}^r_{t+1} = \boldsymbol{f}^a(\boldsymbol{x}^r_t, \boldsymbol{u}^r_t) + \boldsymbol{f}^u(\boldsymbol{x}^r_t, \boldsymbol{u}^r_t) + \boldsymbol{w} \tag{3}$$

where the function $\boldsymbol{f}^a(\cdot,\cdot)$ is assumed known and deterministic, $\boldsymbol{f}^u(\cdot,\cdot) + \boldsymbol{w}$ is an unknown and stochastic term, and $\boldsymbol{w} \in \mathbb{R}^d$ is an additive Gaussian noise. The dynamical system

$$\boldsymbol{x}^a_{t+1} = \boldsymbol{f}^a(\boldsymbol{x}^a_t, \boldsymbol{u}^a_t) \tag{4}$$

represents the known dynamics in (3), and we call it the *approximate* model. Note that $\boldsymbol{x}^a_t$ and $\boldsymbol{x}^r_t$ represent the same state vector in different points of the state space $\mathbb{R}^d$.

The approximate model is known a priori. Hence, a control policy $\boldsymbol{u}^a = \boldsymbol{\pi}^a(\boldsymbol{x}^a)$ for (3) can be learned in simulation by using a reinforcement learning or an optimal control approach. In this work, we used the PILCO algorithm [10] to learn $\boldsymbol{\pi}^a$, but other choices are possible. It is worth noting that searching for the policy $\boldsymbol{\pi}^a$ requires only simulations, i.e., no experiments on the robot are performed at this stage. The learned policy $\boldsymbol{\pi}^a$ represents a good starting point to search for a control policy $\boldsymbol{\pi}^r$ for the real robot. This idea of initializing a control policy from simulations has been effectively applied in [15], where a linear system is used as approximate model. Apart form initializing a policy from simulation, this work exploits real data for the robot to explicitly learn the unknown dynamics $\boldsymbol{f}^u(\cdot,\cdot) + \boldsymbol{w}$ in (3). In other words, when the learned policy $\boldsymbol{\pi}^a$ is applied to the real robot, the measured states are used either to improve the current policy or to learn an estimate of the unknown dynamic. This significantly reduces the number of rollouts to perform on the real robot [10].

C. Learning Unknown Dynamics using Gaussian Processes

The unknown dynamics $\boldsymbol{f}^u(\cdot,\cdot)+\boldsymbol{w}$ in (3) is learned from sensory data using Gaussian Processes (GP) [14]. The goal of the learning process is to obtain an estimate $\hat{\boldsymbol{f}}^u$ of $\boldsymbol{f}^u+\boldsymbol{w}$ such that $\boldsymbol{f}^a(\cdot,\cdot)+\hat{\boldsymbol{f}}^u(\cdot,\cdot)\approx\boldsymbol{f}^a(\cdot,\cdot)+\boldsymbol{f}^u(\cdot,\cdot)+\boldsymbol{w}$. To learn the unknown dynamics, we exploit the real $\boldsymbol{x}_t^r$ and approximate $\boldsymbol{x}_t^a$ states, as well as the real $\boldsymbol{u}_t^r$ and approximate $\boldsymbol{u}_t^a$ control inputs.

For convenience, let us define $\Delta_t=(\Delta\boldsymbol{x}_t,\Delta\boldsymbol{u}_t)$, where $\Delta\boldsymbol{x}_t=\boldsymbol{x}_t^r-\boldsymbol{x}_t^a$ and $\Delta\boldsymbol{u}_t=\boldsymbol{u}_t^r-\boldsymbol{u}_t^a$. We also define $\tilde{\boldsymbol{x}}_t^r=(\boldsymbol{x}_t^r,\boldsymbol{u}_t^r)$ and $\tilde{\boldsymbol{x}}_t^a=(\boldsymbol{x}_t^a,\boldsymbol{u}_t^a)$. Using these definitions, it is straightforward to show that $\tilde{\boldsymbol{x}}_t^r=\tilde{\boldsymbol{x}}_t^a+\Delta_t$ and to rewrite the real dynamics in (3) as

$$\boldsymbol{f}^r(\tilde{\boldsymbol{x}}_t^r)=\boldsymbol{f}^r(\tilde{\boldsymbol{x}}_t^a+\Delta_t)=\boldsymbol{f}^a(\tilde{\boldsymbol{x}}_t^a+\Delta_t)+\boldsymbol{f}^u(\tilde{\boldsymbol{x}}_t^a+\Delta_t)+\boldsymbol{w} \tag{5}$$

Recalling that the zero-order Taylor series expansion of a function $\boldsymbol{f}(\boldsymbol{a}+\Delta\boldsymbol{a})=\boldsymbol{f}(\boldsymbol{a})+\boldsymbol{r}(\Delta\boldsymbol{a})$, we can write

$$\boldsymbol{f}^r(\tilde{\boldsymbol{x}}_t^a+\Delta_t)=\boldsymbol{f}^a(\tilde{\boldsymbol{x}}_t^a)+\boldsymbol{r}^a(\Delta_t)+\boldsymbol{f}^u(\tilde{\boldsymbol{x}}_t^a)+\boldsymbol{r}^u(\Delta_t)+\boldsymbol{w} \tag{6}$$

where $\boldsymbol{r}^a(\Delta_t)$ and $\boldsymbol{r}^u(\Delta_t)$ are the residual errors generated by stopping the Taylor series expansion at the zero-order. Considering (6), the difference between the real (3) and approximate (4) systems can be written as

$$\begin{aligned}\Delta\boldsymbol{x}_{t+1}&=\boldsymbol{f}^r(\tilde{\boldsymbol{x}}_t^a+\Delta_t)-\boldsymbol{f}^a(\tilde{\boldsymbol{x}}_t^a)\\&=\boldsymbol{f}^a(\tilde{\boldsymbol{x}}_t^a)+\boldsymbol{f}^u(\tilde{\boldsymbol{x}}_t^a)+\boldsymbol{r}(\Delta_t)+\boldsymbol{w}-\boldsymbol{f}^a(\tilde{\boldsymbol{x}}_t^a)\\&=\boldsymbol{f}^u(\tilde{\boldsymbol{x}}_t^a)+\boldsymbol{r}(\Delta_t)+\boldsymbol{w}=\hat{\boldsymbol{f}}^u(\tilde{\boldsymbol{x}}_t^a,\Delta_t)\end{aligned} \tag{7}$$

where we pose $\boldsymbol{r}(\Delta_t)=\boldsymbol{r}^a(\Delta_t)+\boldsymbol{r}^u(\Delta_t)$. GP assume that the data are generated by a set of underlying functions, whose joint probability distribution is Gaussian [14]. Having assumed a Gaussian noise term $\boldsymbol{w}$, we can conclude that the stochastic process (7) can be effectively represented as a GP.

Equation (7) shows that $\hat{\boldsymbol{f}}^u$ maps $\tilde{\boldsymbol{x}}_t^a$ and Δ_t into the next state $\Delta\boldsymbol{x}_{t+1}$. Hence, the training inputs of our GP are the tuples $\{\tilde{\boldsymbol{x}}_t^a,\Delta_t\}_{t=0}^{N-1}$, while the training outputs are $\{\boldsymbol{y}_t=\Delta\boldsymbol{x}_{t+1}-\Delta\boldsymbol{x}_t\}_{t=0}^{N-1}$. The training data are obtained as follows. At the n^{th} rollout, the current policy $\boldsymbol{\pi}^n$ is applied to the robot and to the approximate model. As already mentioned, for the first rollout we set $\boldsymbol{\pi}^1=\boldsymbol{\pi}^a$. After the trial, we have the two data sets $\mathcal{X}^a=\{\boldsymbol{x}_t^a,\boldsymbol{u}_t^a\}_{t=0}^N$ and $\mathcal{X}^r=\{\boldsymbol{x}_t^r,\boldsymbol{u}_t^r\}_{t=0}^N$. Hence, we have to simply calculate the element-wise difference $\mathcal{X}^r-\mathcal{X}^a$ to obtain the tuples $\mathcal{X}^d=\{\Delta\boldsymbol{x}_t,\Delta\boldsymbol{u}_t\}_{t=0}^N$, which contain the rest of the training data for the Gaussian process.

The state predicted with a GP $\Delta\boldsymbol{x}_{t+1}$ is Gaussian distributed, i.e., $p(\Delta\boldsymbol{x}_{t+1}|\tilde{\boldsymbol{x}}_t^a,\Delta_t)=\mathcal{N}(\Delta\boldsymbol{x}_{t+1}|\boldsymbol{\mu}_{t+1},\boldsymbol{\Sigma}_{t+1})$. Given N training inputs $\tilde{\boldsymbol{X}}=[\tilde{\boldsymbol{x}}_1^a,\Delta_1,\ldots,\tilde{\boldsymbol{x}}_N^a,\Delta_N]$, training outputs $\boldsymbol{y}=[y_1,\ldots,y_N]^{\mathrm{T}}$, and a query point $\tilde{\boldsymbol{x}}^*=[(\tilde{\boldsymbol{x}}_t^{a,*})^{\mathrm{T}}(\Delta_t^*)^{\mathrm{T}}]^{\mathrm{T}}$, the predicted (one-step) mean $\boldsymbol{\mu}_{t+1}$ and variance $\boldsymbol{\Sigma}_{t+1}$ are

$$\begin{aligned}\boldsymbol{\mu}_{t+1}&=\Delta\boldsymbol{x}_t+\boldsymbol{K}_{\tilde{x}^*\tilde{X}}\left(\boldsymbol{K}_{\tilde{X}\tilde{X}}+\sigma_n^2\boldsymbol{I}\right)^{-1}\boldsymbol{y}\\\boldsymbol{\Sigma}_{t+1}&=k(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{x}}^*)-\boldsymbol{K}_{\tilde{x}^*\tilde{X}}\left(\boldsymbol{K}_{\tilde{X}\tilde{X}}+\sigma_n^2\boldsymbol{I}\right)^{-1}\boldsymbol{K}_{\tilde{X}\tilde{x}^*}\end{aligned} \tag{8}$$

where $\boldsymbol{K}_{\tilde{x}^*\tilde{X}}=k(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{X}})$, $\boldsymbol{K}_{\tilde{X}\tilde{x}^*}=\boldsymbol{K}_{\tilde{x}^*\tilde{X}}^{\mathrm{T}}$, $k(\cdot,\cdot)$ is a covariance function and the generic element ij of the matrix $\boldsymbol{K}_{\tilde{X}\tilde{X}}$ is given by $K_{\tilde{X}\tilde{X}}^{ij}=k(\tilde{\boldsymbol{x}}_i,\tilde{\boldsymbol{x}}_j)$. In our approach, $k(\cdot,\cdot)$ is the squared exponential covariance function

$$k(\tilde{\boldsymbol{x}}_i,\tilde{\boldsymbol{x}}_j)=\sigma_k^2\exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{x}}_j)^{\mathrm{T}}\boldsymbol{\Lambda}^{-1}(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{x}}_j)\right) \tag{9}$$

where $\boldsymbol{\Lambda}=\mathrm{diag}(l_1^2,...,l_D^2)$. The tunable parameters $\boldsymbol{\Lambda}$, σ_k^2 and σ_n^2 are learned from the training data by using evidence maximization [14]. The standard GP formulation works for scalar outputs. For systems with a multidimensional state, we consider one GP for each dimension assuming that the dimensions are independent to each other.

D. Policy Parameterization and Cost Function

The control of a robotic device requires a continuous-valued control policy. It is clear that searching for a control policy in a continuous space is unfeasible in terms of computational cost. A common strategy [2] to solve this issue consists in assuming that the policy $\boldsymbol{\pi}$ is parameterized by $\boldsymbol{\theta}\in\mathbb{R}^p$, i.e., $\boldsymbol{\pi}(\boldsymbol{x},\boldsymbol{\theta})$. In this work, we assume that the policy is the mean of a GP in the form

$$\boldsymbol{\pi}(\boldsymbol{x},\boldsymbol{\theta})=\sum_i^D k(\boldsymbol{x}^*,\boldsymbol{c}_i)\left(\boldsymbol{K}_{CC}+\sigma_\pi^2\boldsymbol{I}\right)^{-1}\boldsymbol{y}_\pi \tag{10}$$

where $\boldsymbol{x}^*$ is the current state, $\boldsymbol{C}=[\boldsymbol{c}_1,\ldots,\boldsymbol{c}_D]$ are the centers of the Gaussian basis function $k(\cdot,\cdot)$ defined as in (9). The vector $\boldsymbol{y}_\pi$ plays the role of the GP training targets, while $\boldsymbol{K}_{CC}$ is defined as in (8). The tunable parameters of the policy (10) are $\boldsymbol{\theta}=[\boldsymbol{C},\boldsymbol{\Lambda},\boldsymbol{y}_\pi]$. As suggested in [10], we set the variance $\sigma_\pi^2=0.01$ in our experiments.

As already mentioned, the optimal policy minimizes the expected return $J^\pi(\boldsymbol{\theta})$ in (1). In this work, we adopt a saturating cost function $c(\boldsymbol{x}_t^r)$ with spread σ_c^2

$$c(\boldsymbol{x}_t^r)=1-\exp\left(-\frac{1}{2\sigma_c^2}\|\boldsymbol{x}_t^r-\boldsymbol{x}_g\|^2\right)\in[0,1] \tag{11}$$

where $\boldsymbol{x}_g$ denotes the goal (target) state[1].

E. Policy Evaluation

The learned GP model is used to compute the long-term predictions $p(\Delta\boldsymbol{x}_1|\boldsymbol{\pi}),...,p(\Delta\boldsymbol{x}_N,|\boldsymbol{\pi})$ starting from $p(\Delta\boldsymbol{x}_0)$. The long-term predictions, in fact, are needed to compute the expected costs and to minimize the expected return in (1). Compute the long-term predictions from the one-step predictions in (8) requires the predictive distribution

$$p(\boldsymbol{y}_t)=\iint p(\hat{\boldsymbol{f}}^u(\boldsymbol{\xi}_t)|\boldsymbol{\xi}_t)p(\boldsymbol{\xi}_t)\mathrm{d}\hat{\boldsymbol{f}}^u\mathrm{d}\boldsymbol{\xi}_t \tag{12}$$

where $\boldsymbol{\xi}_t=[(\tilde{\boldsymbol{x}}_t^a)^{\mathrm{T}}(\Delta_t)^{\mathrm{T}}]^{\mathrm{T}}$ and $\boldsymbol{y}_t=\Delta\boldsymbol{x}_{t+1}-\Delta\boldsymbol{x}_t$. Solving the integral in (12) is analytically intractable. Hence, we assume a Gaussian distribution $p(\boldsymbol{\Delta}_t)=\mathcal{N}(\boldsymbol{\Delta}_t|\boldsymbol{\mu}^\Delta,\boldsymbol{\Sigma}^\Delta)$. The mean $\boldsymbol{\mu}^\Delta$ and $\boldsymbol{\Sigma}^\Delta$ are obtained by applying the moment matching algorithm in [10]. The predictive distribution $p(\Delta\boldsymbol{x}_{t+1}|\boldsymbol{\pi})$ is also Gaussian with mean and covariance

$$\begin{aligned}\boldsymbol{\mu}_{t+1}&=\boldsymbol{\mu}_t+\boldsymbol{\mu}^\Delta\\\boldsymbol{\Sigma}_{t+1}&=\boldsymbol{\Sigma}_t+\boldsymbol{\Sigma}^\Delta+\mathrm{cov}(\Delta\boldsymbol{x}_t,\boldsymbol{\Delta}_t)+\mathrm{cov}(\boldsymbol{\Delta}_t,\Delta\boldsymbol{x}_t)\end{aligned} \tag{13}$$

[1]In [19], the author presents the benefits of the saturating cost (11) compared to a quadratic cost.

where the covariance $\text{cov}(\cdot,\cdot)$ is computed as in [19].

Once the long-term predictions are computed, the estimated real state can be calculated as $\boldsymbol{x}_t^r = \boldsymbol{x}_t^a + \Delta\boldsymbol{x}_t$, where $\boldsymbol{x}_t^a$ is the deterministic state of the approximate model (see Sec. II-A). Given that $\boldsymbol{x}_t^r = \boldsymbol{x}_t^a + \Delta\boldsymbol{x}_t$, the expected real long-term cost can be expressed as

$$J^\pi(\boldsymbol{\theta}) = \sum_{t=0}^{N} \mathbb{E}_{x_t^r}[c(\boldsymbol{x}_t^r)] = \sum_{t=0}^{N} \mathbb{E}_{\Delta x_t}[c(\Delta\boldsymbol{x}_t)] \qquad (14)$$

Indeed, by substituting $\Delta\boldsymbol{x}_t = \boldsymbol{x}_t^r - \boldsymbol{x}_t^a$ into (11) we obtain

$$\begin{aligned} c(\boldsymbol{x}_t^r) &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\|\boldsymbol{x}_t^r - \boldsymbol{x}_g\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\|\Delta\boldsymbol{x}_t + \boldsymbol{x}_t^a - \boldsymbol{x}_g\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\|\Delta\boldsymbol{x}_t - (\boldsymbol{x}_g - \boldsymbol{x}_t^a)\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\|\Delta\boldsymbol{x}_t - \Delta\boldsymbol{x}_{g,t}\|^2\right) = c(\Delta\boldsymbol{x}_t) \end{aligned}$$

The vector $\boldsymbol{x}_t^a$ is the approximate state computed by applying the initial policy $\boldsymbol{\pi}^a$ and it is deterministic. Hence, we can precalculate all the $\Delta\boldsymbol{x}_{g,t}$ for $t = 1,\ldots,N$. Note that by minimizing $c(\Delta\boldsymbol{x}_t)$ we force the real robot to follow the same trajectory of the approximate model. Recalling that the predictive distribution $p(\Delta\boldsymbol{x}_t|\boldsymbol{\pi}) = \mathcal{N}(\Delta\boldsymbol{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, we can rewrite the real expected costs in (1) as

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

where $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ are defined as in (13). The expected value $\mathbb{E}_{\Delta x_t}[c(\Delta\boldsymbol{x}_t)]$ in (15) can be computed analytically and it is differentiable, which allows the adoption of gradient based approaches for policy search.

F. Policy Search

We leverage the gradient $\partial J^\pi(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ to search the policy parameters $\boldsymbol{\theta}^*$ that minimize the expected long-term cost $J^\pi(\boldsymbol{\theta})$. In our formulation, the expected costs in (1) can be written in the form (15). Given that the expected costs have the form (15), and assuming that the moment matching algorithm is used for long-term predictions, the gradient $\partial J^\pi(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ can be computed analytically. The computation of the gradient $\partial J^\pi(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ is detailed in [10].

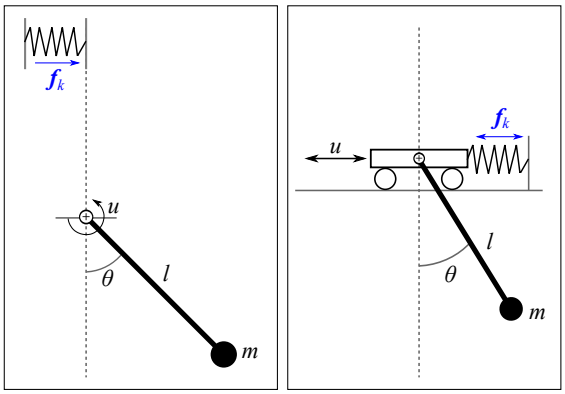
The result of the policy improvement is a set of policy parameters $\boldsymbol{\theta}^n$, where n indicates the n^{th} iteration. The control policy $\boldsymbol{\pi}^n = \boldsymbol{\pi}(\boldsymbol{x}, \boldsymbol{\theta}^n)$ drives the estimated real state $\boldsymbol{x}_t^a + \Delta\boldsymbol{x}_t$ towards the desired goal. The new policy $\boldsymbol{\pi}^n$ is applied to the real robot to obtain new training data and to refine the residual model. The process is repeated until the task is learned. PI-REM is summarized in Algorithm 1.

III. Experiments

Experiments in this section show the effectiveness of our approach in learning control policies for simulated and real physical systems. The proposed approach is compared with the PILCO algorithm in [10] considering the number of rollouts performed on the real system and the total time duration of the rollouts (real experience). Both PI-REM and PILCO are implemented in Matlab®.

Algorithm 1 Policy Improvement with REsidual Model learning (PI-REM)

```
Learn a policy π^a for the approximate model (4)
π^n = π^a
while task learning is not completed do
    Apply π^n to the approximate model and the real robot
    Calculate the new training set X^d = X^r − X^a
    Calculate the new target set Δx_{g,t} = x_g − x_t^a
    Learn GP model for the dynamics (7)
    while J^π(θ) not minimized do
        Policy evaluation using (13) and (15)
        Policy improvement (Sec. II-F and [10])
    end while
    return θ*
end while
return π*
```



(a) (b)

Fig. 2. (a) The pendulum interacting with an elastic environment. (b) The cart–pole system connected to a spring.

In all the experiments, we use the saturating cost function in (11). To consider bounded control inputs, the control policy in (10) is squashed into the interval $[-u_{max}, u_{max}]$ by using the function $\sigma(x) = u_{max}\frac{9\sin(x)+\sin(3x)}{8}$.

A. Simulation Results

We tested PI-REM in two different tasks, namely a pendulum swing-up and a cart–pole swing-up (see Fig. 2), and compared the results with PILCO [10]. For a fair comparison, we use the same parameters for PI-REM and PILCO.

1) Pendulum swing-up: The task consists in balancing the pendulum in the inverted position ($\theta_g = -\pi$ rad), starting from the stable position ($\theta_0 = 0$ rad). As shown in Fig. 2(a), the pendulum interacts with an elastic environment (modeled as a spring) when it reaches the vertical position ($\theta = \pi$ rad). The interaction generates an extra force $\boldsymbol{f}_k$ on the pendulum, which is neglected in our approximate model. Hence, the approximate model is the standard pendulum model

$$\ddot{\theta}_t(\frac{1}{4}ml^2 + I) + \frac{1}{2}mlg\sin(\theta_t) = u_t - b\dot{\theta}_t$$

where the mass $m = 1\,\text{kg}$, the length $l = 1\,\text{m}$, $I = \frac{1}{12}ml^2$ is the moment of inertia of a pendulum around the midpoint, $b = 0.01\,\text{sNm/rad}$ is the friction coefficient, $g = 9.81\,\text{m/s}^2$ is the gravity acceleration and u_t is the control torque. The state of the pendulum is defined as $\boldsymbol{x} = [\dot{\theta}, \theta]^{\text{T}}$, while the goal to reach is $\boldsymbol{x}_g = [0, -\pi]^{\text{T}}$. The external force $\boldsymbol{f}_k$ depends on the stiffness of the spring. We tested three different stiffness values, namely 100, 200 and 500 N/m. Results are reported is Tab. I. Note that the time of real experience in Tab. I is a multiple of the number of rollouts. This is because we keep the duration of the task fixed for each rollout.

TABLE I

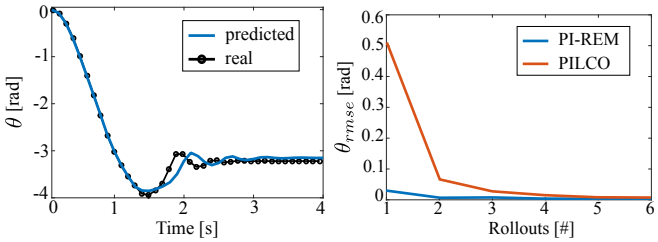
RESULTS FOR THE PENDULUM SWING-UP.

	Stiffness [N/m]	Real rollouts [#]	Real experience [s]
PI-REM	100	**2**	**8**
PILCO [10]	100	6	24
PI-REM	200	**3**	**12**
PILCO [10]	200	4	16
PI-REM	500	**2**	**4**
PILCO [10]	500	3	6

Stiffness 100 N/m: For a stiffness equal to 100 N/m, $u_{max} = 5\,\text{Nm}$ is sufficient to fulfill the task. The total duration of the task is $T = 4\,\text{s}$, while the sampling time is $dt = 0.1\,\text{s}$. Results in Tab. I show that PI-REM needs only 2 iterations and 8 s of real experience to learn the control policy. For comparison, PILCO needs 6 iterations and 24 s of real experience. The improvement mostly depends on the reduced state prediction error of PI-REM. As shown in Fig. 3(a), the approximate model accurately represents the system until the pendulum touches the spring. This lets PI-REM to properly estimate the state after 2 iterations (see Fig. 3(b)). PILCO, instead, spends 5 iterations to learn a proper model of the system. As shown in Fig. 4, after 2 iterations the pendulum is able to reach the goal $\boldsymbol{x}_g = [0, -\pi]^{\text{T}}$. The learned control policy is bounded ($u(x) \in [-5, 5]\,\text{Nm}$), and it drives the pendulum to the goal position in less than 1 s (see Fig. 4(a)). This results in an average expected return $J^{\pi}(\boldsymbol{\theta}) \approx 12$ (see Fig. 4(b)). PILCO needs more rollouts (6 instead of 2) to find the optimal policy. Since we used the same policy parameterization and cost function in both the approaches, PILCO and PI-REM learn (almost) the same control policy that generates a similar behavior of the pendulum.

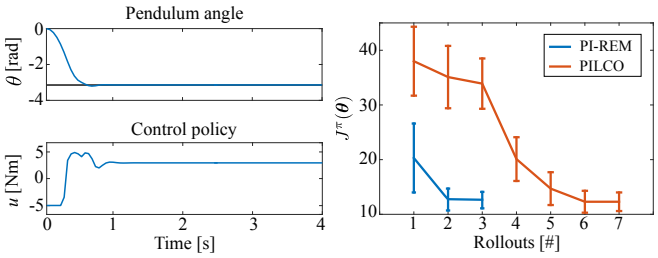
Stiffness 200 N/m: For a stiffness equal to 200 N/m, we have to reduce the sampling time to 0.05 s to avoid numerical instabilities in the simulation of the real model. Moreover, we increase the maximum control input to 15 Nm. In this case, our approach needs 3 iterations and 12 s of real experience to learn the control policy (see Tab. I).

Stiffness 500 N/m: For a stiffness equal to 500 N/m, we have to further reduce the sampling time to 0.0125 s. To speed-up the learning, we reduce the prediction time to 2 s. Moreover, we increase the maximum control input to 25 Nm. In this case, our approach needs 2 iterations and 4 s of real experience to learn the control policy (see Tab. I).



(a) (b)

Fig. 3. Pendulum model learning results (stiffness 100 N/m). (a) State predicted and measured after one iteration of PI-REM. (b) State prediction (root mean square) errors of PI-REM and PILCO for different iterations.



(a) (b)

Fig. 4. Results for the pendulum swing-up (stiffness 100 N/m). (a) Pendulum angle and control policy obtained with PI-REM after 2 iterations. The black solid line represents the goal. (b) Evolution of the expected return $J^{\pi}(\boldsymbol{\theta})$ for different rollouts (mean and std over 5 executions).

2) Cart–pole swing-up: The task consists in swinging-up the pendulum on the cart and balance it in the inverted position ($\theta_g = -\pi\,\text{rad}$), starting from the stable position ($\theta_0 = 0\,\text{rad}$). The control input is the horizontal force u_t that makes the cart moving to the left or to the right. As shown in Fig. 2(b), the cart is connected to a wall through a spring. The additive force $\boldsymbol{f}_k$ of the spring affects the motion of the cart. The mass of the cart is $m_c = 0.5\,\text{kg}$, the mass of the pendulum is $m_p = 0.5\,\text{kg}$, and the length of the pendulum is $l = 0.5\,\text{m}$. The state of the cart–pole system is $\boldsymbol{x} = [p, \dot{p}, \theta, \dot{\theta}]^{\text{T}}$, where p is the position of the cart, $\dot{p}$ is the velocity of the cart, θ and $\dot{\theta}$ are respectively the angle and the angular velocity of the pendulum. The goal state is $\boldsymbol{x}_g = [0, 0, \pi, 0]^{\text{T}}$. The cart–pole system is more complicated than the single pendulum, since the state has four dimensions instead of two. Also in this case, the approximate model does not consider the extra force $\boldsymbol{f}_k$. The approximate model of the cart–pole pendulum is then

$$(m_c + m_p)\ddot{p}_t + \frac{1}{2}m_p l\ddot{\theta}_t \cos(\theta_t) - \frac{1}{2}m_p l\dot{\theta}_t^2 \sin(\theta_t) = u_t - b\dot{p}_t$$
$$2l\ddot{\theta}_t + 3\ddot{p}_t \cos(\theta_t) + 3g \sin(\theta_t) = 0$$

We tested three different stiffness values, namely 25, 50 and 120 N/m. Results are reported is Tab. II. Recall that the time of real experience in Tab. II is a multiple of the number of rollouts.

Stiffness 25 N/m: For a stiffness equal to 25 N/m, $u_{max} = 10\,\text{N}$ is sufficient to fulfill the task. The total duration of the task is $T = 4\,\text{s}$, while the sampling time is $dt = 0.1\,\text{s}$. Results in Tab. II show that our approach needs only 2 iterations and 8 s of real experience to learn the control

TABLE II

RESULTS FOR THE CART–POLE SWING-UP.

	Stiffness [N/m]	Real rollouts [#]	Real experience [s]
PI-REM	25	**2**	**8**
PILCO [10]	25	5	20
PI-REM	50	**3**	**12**
PILCO [10]	50	6	24
PI-REM	120	**15**	**30**
PILCO [10]	120	23	46

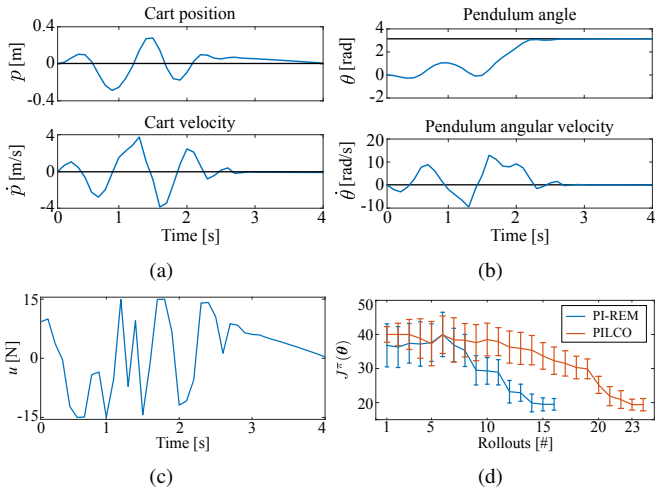


Fig. 5. Results for the cart–pole swing-up (stiffness 120 N/m). (a)–(b) State of the cart–pole system and (c) learned control policy after 15 iterations of PI-REM. The black solid line represents the goal. (d) Evolution of the expected return $J^\pi(\boldsymbol{\theta})$ for different rollouts (mean and std over 5 executions).

policy. For comparison, PILCO needs 5 iterations and 20 s of real experience.

Stiffness 50 N/m: For a stiffness equal to 200 N/m, we have to increase the maximum control input to 15 N in order to fulfill the task. In this case, PI-REM needs 3 iterations and 12 s of real experience to learn the policy (see Tab. II).

Stiffness 120 N/m: For a stiffness equal to 120 N/m, we have to reduce the sampling time to 0.05 s to avoid numerical instabilities in the simulation of the real model. We also reduce the duration of the task to $T = 2$ s to speed-up the learning process. In this case, our approach needs 15 iterations and 30 s of real experience to learn the control policy (see Tab. II). As shown in Fig. 5, after 15 iterations the cart–pole system reaches the goal $\boldsymbol{x}_g = [0, 0, \pi, 0]^{\mathrm{T}}$. The learned control policy is bounded ($u(x) \in [-15, 15]$ N), and it drives the cart–pole system to the goal position in less than 4 s (see Fig. 5(a) to 5(c)). This results in an average expected return $J^\pi(\boldsymbol{\theta}) \approx 20$ (see Fig. 5(d)). PILCO needs more rollouts (23 instead of 15) to find the optimal policy. Since we used the same policy parameterization and cost function in both the approaches, PILCO and PI-REM learn (almost) the same control policy that generates a similar behavior of the cart–pole system.

In the considered simulations, the proposed PI-REM performs better than PILCO. In terms of rollouts, our approach takes from 25% to 67% less iterations than PILCO. As a consequence, the time of real experience is reduced from 25% to 67% (from a minimum of 2 s to a maximum of 16 s less than PILCO).

B. Robot Experiment

PI-REM is applied to control a qbmove Maker variable stiffness actuator (VSA) [20]. As shown in Fig. 6, the arm consists of four VSA. The first joint of the arm is actuated, while the other three are not controlled. The task consists in swinging-up the robotic arm by controlling the actuated joint. As approximate model, we use the mass–spring system

$$m\ddot{\theta}_t + k(\theta_t - u_t) = 0 \tag{16}$$

where $m = 0.78$ kg is the mass of the unactuated joints, and $k = 14$ Nm/rad is the maximum stiffness of the qbmove Maker [20]. The variable θ_t represents the link position, while u_t is the commanded motor position. It is worth noticing that the approximate model in (16) neglects the length, inertia and friction of the arm in Fig. 6. Moreover, the model in (16) represents a VSA only in the region where the behavior of the spring is linear.

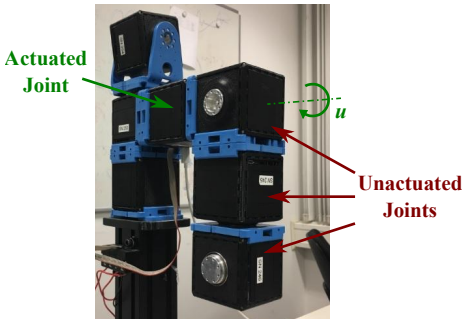


Fig. 6. The VSA arm used in the robotic experiment.

The goal is to learn a control policy u_t that drives the model from the initial state $\boldsymbol{x}_0 = [\theta_0, \dot{\theta}_0] = [\pi/2, 0]^{\mathrm{T}}$ to the goal $\boldsymbol{x}_g = [-\pi/2, 0]^{\mathrm{T}}$ in $T = 5$ s. The maximum input position is $u_{max} = 1.7$ rad. Joint velocities are computed from the measured joint angles using a constant velocity Kalman filter. The robot is controlled at 500 Hz via a Simulink® interface. Results in Tab. III show that our approach needs 3 iterations and 15 s of real experience to lean the task. For comparison, PILCO needs 5 iterations and 25 s of real experience to learn the same task. Hence, PI-REM shows an improvement of the performance of 40%. Figure 7 shows the learned policy, the cost evolution and the state of the robot after 3 iterations of PI-REM. Note that, for a better visualization, we show only the first two seconds of the task. As shown in Fig. 7(a), the robot effectively reaches the goal after approximatively 0.5 s. This results in an expected return $J^\pi(\boldsymbol{\theta}) \approx 9$ (see Fig. 7(d)). PILCO needs 5 rollouts (instead of 3) to find the optimal policy. As shown in Fig. 7(d), the expected return of PILCO is also $J^\pi(\boldsymbol{\theta}) \approx 9$. This indicates that PILCO and PI-REM learn a similar control policy that generates a similar behavior of the VSA pendulum.

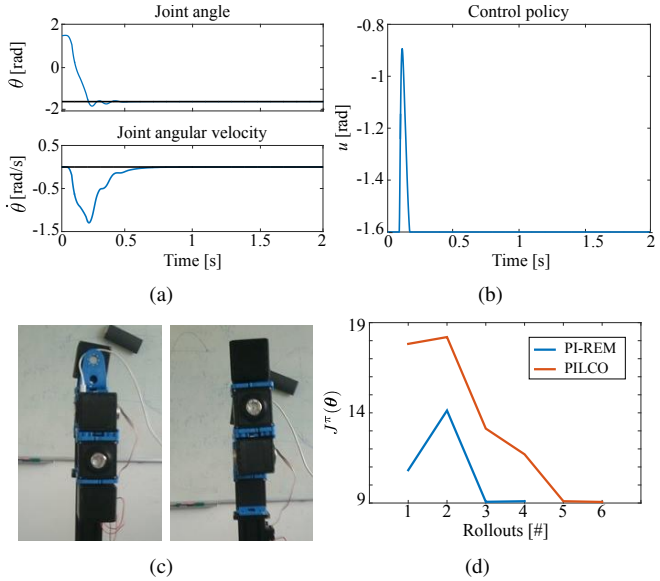


Fig. 7. Results for the VSA pendulum swing-up. (a) State of the VSA pendulum, (b) learned control policy, and (c) snapshots of the task execution after 3 iterations of PI-REM. The black solid line in (a) represents the goal. (d) Evolution of the expected return $J^{\pi}(\boldsymbol{\theta})$ for different rollouts.

Performed simulations and experiments show that learning a black-box model of the residual, instead of a black-box model of the entire system, can significantly improve the performance of the policy search algorithm.

TABLE III

RESULTS FOR THE VSA PENDULUM SWING-UP.

	Real rollouts [#]	Real experience [s]
PI-REM	**3**	**15**
PILCO [10]	5	25

IV. CONCLUSION AND FUTURE WORK

In this work, we proposed PI-REM, a model-based reinforcement learning approach that exploits a simplified model to find an optimal control policy for a given task. The proposed approach works in two steps. First, an optimal policy is found for the simplified model using standard policy search algorithms. Second, sensory data from the real robot are exploited to learn a probabilistic model that represents the difference between the real system and its simplified model. Differently from most model-based RL approaches, our method does not learn a black-box model of the entire system. PI-REM learns, instead, a black-box model of the differences between the real system and an approximated model, i.e., the model residual. As a consequence, our approach is suitable in applications where the system model is inaccurate or only partially available. The approach has been compared with PILCO, showing improvements in terms of performed rollouts and time of real experience.

Our future research will focus on testing the proposed approach in more complicated tasks with a high dimensional searching space. An important open question is how inaccurate the model can be in order to maintain enhanced performance. We expect that with a completely wrong model, the performance of the algorithm becomes similar or slightly worse than classical black-box approaches such as PILCO. However, formally quantifying the tolerated uncertainty still remains an open theoretical problem. Another potential limitation of our approach is that we assume additive model residuals. Evaluating the generality of this assumption more precisely will be a possible future research topic.

ACKNOWLEDGEMENTS

This work has been partially supported by the Technical University of Munich, International Graduate School of Science and Engineering, and by the Marie Curie Action LEACON, EU project 659265.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement learning: an introduction*, ser. A Bradford book. MIT Press, 1998.
[2] J. Kober, D. Bagnell, and J. Peters, "Reinforcement learning in robotics: a survey," *IJRR*, vol. 32, no. 11, pp. 1238–1274, 2013.
[3] S. Calinon, P. Kormushev, and D. Caldwell, "Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning," *RAS*, vol. 61, no. 4, pp. 369–379, 2013.
[4] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *NIPS*, 2009, pp. 849–856.
[5] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.
[6] J. Peters, K. Muelling, and Y. Altun, "Relative entropy policy search," in *Conference on Articial Intelligence (AAAI)*, 2010, pp. 1607–1612.
[7] D. Lee, H. Kunori, and Y. Nakamura, "Association of whole body motion from tool knowledge for humanoid robots," in *International Conference on Intelligent Robots and Systems*, 2008, pp. 2867–2874.
[8] H. Kunori, D. Lee, and Y. Nakamura, "Associating and reshaping of whole body motions for object manipulation," in *International Conference on Intelligent Robots and Systems*, 2009, pp. 5240–5247.
[9] M. Saveriano, S. An, and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *International Conference on Robotics and Automation*, 2015, pp. 3570–3575.
[10] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *TPAMI*, vol. 37, no. 2, pp. 408–423, 2015.
[11] J. G. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," in *in Neural Information Processing Systems 9*. The MIT Press, 1996, pp. 1047–1053.
[12] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *International Conference on Machine Learning*, 1997, pp. 12–20.
[13] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *International conference on robotics and automation*, 1997, pp. 3557–3564.
[14] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.
[15] F. Farshidian, M. Neunert, and J. Buchli, "Learning of closed-loop motion control," in *IROS*, 2014, pp. 1441–1446.
[16] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.
[17] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics - Modelling, Planning and Control*. Springer, 2009.
[18] G. D. Maria, P. Falco, C. Natale, and S. Pirozzi, "Integrated force/tactile sensing: The enabling technology for slipping detection and avoidance," in *ICRA*, 2015, pp. 3883–3889.
[19] M. Deisenroth, *Efficient Reinforcement Learning using Gaussian Processes*. KIT Scientific Publishing, 2010.
[20] M. G. Catalano, G. Grioli, M. Garabini, F. Bonomo, M. Mancini, N. Tsagarakis, and A. Bicchi, "Vsa-cubebot: A modular variable stiffness platform for multiple degrees of freedom robots," in *International Conference on Robotics and Automation*, 2011, pp. 5090–5095.