PI-REM: Policy Improvement with REsidual Model learning

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

[1]Human-centered Assistive Robotics, Technical University of Munich,
Karlstrasse 45, 80333 Munich, Germany
{matteo.saveriano,yuchao.yin,pietro.falco,dhlee}@tum.de
[2]Institute of Robotics and Mechatronics, German Aerospace Center (DLR),
Münchenerstrasse 20, 82234 Weßling, Germany

Abstract. In this work, we analyze the benefits of combining policy initialization and residuals learning to speed-up the policy search in model-based reinforcement learning. In particular, we leverage the Policy Improvement with REsidual Model learning (PI-REM), a model-based and data-efficient approach for reinforcement learning. PI-REM employs a simplified model of the system, in the form of a dynamical system, to search for an initial control policy. Moreover, the approach exploits sensory data to fit a probabilistic model of the residual difference between the measured state and the state of the simplified model. The learned residual model is then combined with the simplified dynamics to predict the real state of system and to search for an optimal policy. Simulation results show that the proposed approach effectively reduces the number of rollouts required to find an optimal control policy.

1 Introduction

The Reinforcement Learning (RL) framework allows a robot to learn a task by self-practice [10], i.e. by performing a number of trials (rollouts) and exploiting sensory data to improve its behavior. The effectiveness of RL is demonstrated in a variety of challenging robotics applications, including jumping [9], pancake flipping [16], and variable impedance control [12]. However, when applied to robotics and control problems, RL suffers from two main disadvantages [10]: *i)* state and action spaces are continuous-valued and high dimensional and *ii)* performing a rollout on real devices is extremely time consuming. A common approach to overcome problem *i)* is to employ parameterized control policies with a discrete number of parameters [4,9,16,19]. A possible way to alleviate problem *ii)* is to provide a good initial policy, for example via kinesthetic teaching [13]. Despite the possibility of properly initializing the policy, reducing the rollouts required to find an optimal policy is still an open problem.

Recent advances in RL research [2,5,15] show that model-based approaches can be effectively exploited to significantly reduce the number of rollouts. Model-based RL approaches employ sensory data to fit a model of the system to control.

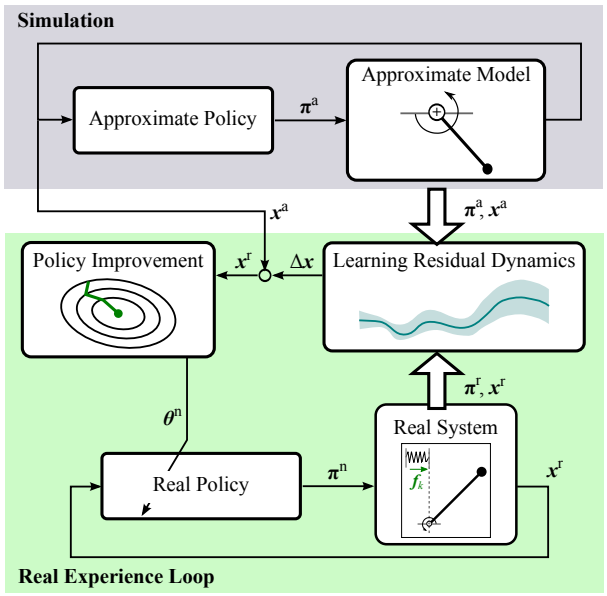


Fig. 1. Overview of the PI-REM algorithm.

The learned model is then exploited to predict the behavior of the system, reducing the number of iterations on the real robot required to find the optimal policy. It is known that model-based approaches suffer from the model–bias problem [11, 14, 17], since they assume that the learned model accurately represents the real robot. However, the model–bias problem can be significantly alleviated by explicitly considering uncertainty in the system model [2, 15].

The work in [5] employs an approximate model of the system to control and optimal control techniques to find a proper initial control policy. Such initial policy is a good starting point to search for a control policy for the real system, which is obtained by using model-free RL. This combination of model-based initialization and model-free learning significantly reduces the number of needed rollouts. Similarly, the algorithm in [2] exploits optimal control to find an initial policy, but it also learns a locally linear model of the system from sensory data.

The approach in [15], namely the Probabilistic Inference for Learning Control (PILCO), exploits Gaussian Processes (GP) [18] to learn a probabilistic model of the system from sensory data. Hence, PILCO assumes that model uncertainties are Gaussian distributed. Model uncertainties are explicitly considered in the long-term predictions used to evaluated the current policy, which allows PILCO to outperform several state-of-the-art approaches in terms of required rollouts [15]. Nevertheless, PILCO can find the optimal policy if the learned model reliably represents the system to control. In order to rapidly fit an accurate model, the state space is explored as much as possible in the first rollouts by applying a random policy. Despite the random initialization, PILCO needs

several samples to fit a reliable GP model which increases the computation time and reduces the scalability to high dimensional problems.

The work in [8] exploits black-box optimization and parallel computation to significantly reduce the learning time. However, multi-core processors suitable for parallel computation are not always available on robotic devices. The work in [6,7] encode prior information on the robot model in a non-zero GP mean. In particular, [7] considers a linear prior, while [6] assumes a GP prior.

The Policy Improvement with REsidual Model learning (PI-REM), originally proposed in [3], leverages an approximate model of the robot to find a proper initial control policy, and it exploits sensory data to learn a model of the residual difference between approximate and real models (see Fig. 1). PI-REM assumes that the approximate model is easy to analytically derive, while it relies on machine learning to compensate for unmodeled (or hard to model) dynamics. This work aims at showing that PI-REM quickly converges towards an optimal policy because it combines policy initialization and residual dynamics learning in a fruitful manner. To this end, we compare PI-REM and PILCO in two simulated scenarios, measuring the learning time and performed rollouts. Moreover, to show that learning residual dynamics is crucial for the performance of PI-REM, PILCO is initialized either with a random policy or with the same initial policy used by PI-REM. Obtained results show that, on average, PI-REM spends less rollouts than PILCO to find an optimal policy.

2 The PI-REM Approach

As for any RL algorithm, the goal of PI-REM 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 $\mathbb{E}[\cdot]$ indicates the expected value, $c(\boldsymbol{x}_t)$ is the cost of being in state $\boldsymbol{x}$ at time t, $\boldsymbol{\theta}$ are the parameters used to represent the continuous policy function $\boldsymbol{\pi}(\boldsymbol{x}, \boldsymbol{\theta})$. PI-REM assumes that the system to control is modeled using a non-linear dynamical system, i.e.

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

where $\boldsymbol{x}^r \in \mathbb{R}^d$ is the state of the system, $\boldsymbol{u}^r \in \mathbb{R}^f$ is the control input, $\boldsymbol{f}^r \in \mathbb{R}^d$ is a continuous and continuously differentiable function. As in [3], the superscript r indicates that (2) is the real (exact) model of the system.

2.1 Residual Dynamics: Definition and Learning

Deriving the exact model in (2) can be complicated in real cases. Hence, PI-REM assumes that the function $\boldsymbol{f}^r(\cdot, \cdot)$ in (2) has *i)* a deterministic and known part and *ii)* an additive unknown part. These assumptions allow us to rewrite (2) 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 $\boldsymbol{f}^a(\cdot,\cdot)$ is a known and deterministic function, $\boldsymbol{f}^u(\cdot,\cdot)+\boldsymbol{w}$ is an unknown and stochastic term, and $\boldsymbol{w} \in \mathbb{R}^d$ is an additive Gaussian noise. The so-called *approximate* model

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

describes the known part of (3) and it is known a priori. PI-REM exploits the approximate model (4) in two ways. First, it finds in simulation a control policy $\boldsymbol{u}^a = \boldsymbol{\pi}^a(\boldsymbol{x}^a)$ for (3). $\boldsymbol{\pi}^a$ is a good starting point to search for a control policy $\boldsymbol{\pi}^r$ for the real robot. Second, PI-REM uses sensory data to adjust the approximate model in order to obtain a reliable representation of the system to control.

In particular, the unknown additive dynamics $\boldsymbol{f}^u(\cdot,\cdot)+\boldsymbol{w}$ in (3) is learned using Gaussian Processes (GP) [18]. More specifically, it holds that

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

where $\Delta_t = (\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t)$, $\Delta\boldsymbol{x}_t = \boldsymbol{x}_t^r - \boldsymbol{x}_t^a$, $\Delta\boldsymbol{u}_t = \boldsymbol{u}_t^r - \boldsymbol{u}_t^a$, $\tilde{\boldsymbol{x}}_t^r = (\boldsymbol{x}_t^r, \boldsymbol{u}_t^r)$, and $\tilde{\boldsymbol{x}}_t^a = (\boldsymbol{x}_t^a, \boldsymbol{u}_t^a)$. To derive the result in (5) we leverage the zero-order Taylor series expansion and the relationship $\tilde{\boldsymbol{x}}_t^r = \tilde{\boldsymbol{x}}_t^a + \Delta_t$ to rewrite (3) as

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

where $\boldsymbol{r}^a(\Delta_t)$ and $\boldsymbol{r}^u(\Delta_t)$ are the *residual* errors committed by stopping the Taylor series expansion at the zero-order. Given (6), the difference between the real (3) and approximate (4) dynamics becomes

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

where we pose $\boldsymbol{r}(\Delta_t) = \boldsymbol{r}^a(\Delta_t) + \boldsymbol{r}^u(\Delta_t)$. Note that the assumption of additive Gaussian noise allows to represent the stochastic process (7) as a GP.

Equation (7) shows that the training inputs of the GP are $\{\tilde{\boldsymbol{x}}_t^a, \Delta_t\}_{t=0}^{N-1}$, while the training outputs are $\{\boldsymbol{y}_t = \Delta\boldsymbol{x}_{t+1} - \Delta\boldsymbol{x}_t\}_{t=0}^{N-1}$. In order to collect the training data, at the n^{th} rollout the current policy $\boldsymbol{\pi}^n$ is applied to the approximate model and to the robot, obtaining $\mathcal{X}^a = \{\boldsymbol{x}_t^a, \boldsymbol{u}_t^a\}_{t=0}^N$ and $\mathcal{X}^r = \{\boldsymbol{x}_t^r, \boldsymbol{u}_t^r\}_{t=0}^N$ respectively. As already mentioned, the procedure is initialized with $\boldsymbol{\pi}^1 = \boldsymbol{\pi}^a$. The tuples $\mathcal{X}^d = \{\Delta_t = (\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t)\}_{t=0}^N$ are then computed through the element-wise difference $\mathcal{X}^r - \mathcal{X}^a$. Considering 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 state $\Delta\boldsymbol{x}_{t+1}$ predicted (one-step) with a GP is specified by mean and variance

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

where $\boldsymbol{K}_{\tilde{x}^*\tilde{X}} = k(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{X}})$, $\boldsymbol{K}_{\tilde{X}\tilde{x}^*} = \boldsymbol{K}_{\tilde{x}^*\tilde{X}}^{\mathrm{T}}$, and the generic element ij of the matrix $\boldsymbol{K}_{\tilde{X}\tilde{X}}$ is given by $K_{\tilde{X}\tilde{X}}^{ij} = k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$. The kernel $k(\cdot,\cdot)$ is usually the squared exponential covariance function $k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j) = \sigma_k^2 \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j)\right)$, 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 training data via evidence maximization [18].

2.2 Control Policy and Cost Function

In robotics applications, the continuous-valued policy $\boldsymbol{\pi}$ is usually parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$ to discretize the searching space. In this work, the policy is

$$\boldsymbol{\pi}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i}^{D} k(\boldsymbol{x}^*, \boldsymbol{c}_i) \left(\boldsymbol{K}_{CC} + \sigma_\pi^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}_\pi, \tag{9}$$

that represents the mean of a GP. In (9), $\boldsymbol{x}^*$ is the current state and $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_D]$ are the centers of the squared exponential covariance function $k(\cdot, \cdot)$. The vector $\boldsymbol{y}_\pi$ contains the GP training targets, while $\boldsymbol{K}_{CC}$ is defined as in (8). $\boldsymbol{\theta} = [\boldsymbol{C}, \boldsymbol{\Lambda}, \boldsymbol{y}_\pi]$ are learnable parameters of the policy (9).

The optimal policy minimizes the expected return $J^\pi(\boldsymbol{\theta})$ in (1). This work employs the saturating cost function

$$c(\boldsymbol{x}_t^r) = 1 - \exp\left(-\frac{1}{2\sigma_c^2} \|\boldsymbol{x}_t^r - \boldsymbol{x}_g\|^2\right) \in [0, 1], \tag{10}$$

where $\boldsymbol{x}_g$ denotes the target state and σ_c^2 is the spread.

2.3 Policy Evaluation using GP Predictions

In order to compute the expected costs and to minimize the expected return in (1), the long-term predictions $p(\Delta\boldsymbol{x}_1|\boldsymbol{\pi}), ..., p(\Delta\boldsymbol{x}_N, |\boldsymbol{\pi})$, computed from $p(\Delta\boldsymbol{x}_0)$, are needed. Computing the exact long-term predictions is analytically intractable [15]. Hence, we approximate the predictive distribution $p(\Delta\boldsymbol{x}_{t+1}|\boldsymbol{\pi})$ with a Gaussian, specified by mean and covariance

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \boldsymbol{\mu}^\Delta, \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}^\Delta + \operatorname{cov}(\Delta\boldsymbol{x}_t, \boldsymbol{\Delta}_t) + \operatorname{cov}(\boldsymbol{\Delta}_t, \Delta\boldsymbol{x}_t), \end{aligned} \tag{11}$$

where $\boldsymbol{\mu}^\Delta$ and $\boldsymbol{\Sigma}^\Delta$ are computed with the moment matching algorithm in [15], and the covariance $\operatorname{cov}(\cdot, \cdot)$ is computed as in [20].

Given the long-term predictions, the estimated real state is computed as $\boldsymbol{x}_t^r = \boldsymbol{x}_t^a + \Delta\boldsymbol{x}_t$. Note that, by substituting $\Delta\boldsymbol{x}_t = \boldsymbol{x}_t^r - \boldsymbol{x}_t^a$ into (10), it is possible to write

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

Hence, the expected long-term cost can be expressed as $J^\pi(\boldsymbol{\theta}) = \sum_{t=0}^{N} \mathbb{E}_{x_t^r}[c(\boldsymbol{x}_t^r)]$ $= \sum_{t=0}^{N} \mathbb{E}_{\Delta x_t}[c(\Delta\boldsymbol{x}_t)]$. The approximate state $\boldsymbol{x}_t^a$ is computed off-line by applying the initial policy $\boldsymbol{\pi}^a$. This implies that the $\Delta\boldsymbol{x}_{g,t}$ for $t = 1, \ldots, N$ can be pre-calculated. In other words, by minimizing $c(\Delta\boldsymbol{x}_t)$ the real system is forced to

follow the same trajectory of the approximate model. Having assumed a Gaussian predictive distribution $p(\Delta\boldsymbol{x}_t|\boldsymbol{\pi}) = \mathcal{N}(\Delta\boldsymbol{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, the real expected cost in (1) becomes

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

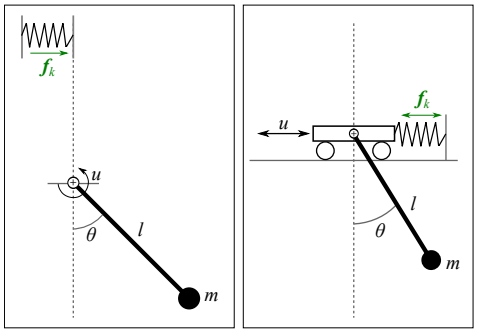
where $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ are defined as in (11). As shown in [15], the integral in (12) can be analytically computed and it is differentiable. This permits the adoption of gradient-based approaches to improve the current policy. The PI-REM algorithm is summarized in Algorithm 1.

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

```
Learn a policy π^a for the approximate model (4)
Compute the target set Δx_{g,t} = x_g − x_t^a and initialize π^n = π^a
while task learning is not completed do
    Apply π^n to the real robot and to the approximate model
    Calculate new training data X^d = X^r − X^a
    Learn GP model for the dynamics (7)
    while J^π(θ) not minimized do
        Policy evaluation using (11) and (12)
        Gradient-based policy improvement [15]
    end while
    return θ*
end while
return π*
```

3 Results

Presented results show that the combination of policy initialization and residual dynamics learning exploited by PI-REM significantly reduces the number of rollout performed on the real system. PI-REM is here applied to learn a pendulum swing-up and a cart–pole swing-up task (see Fig. 2). The presented approach is compared with PILCO [15]. For a fair comparison, the same parameters for PI-REM and PILCO are used. Moreover, PILCO is initialized both with a random policy and with the initial policy $\boldsymbol{\pi}^a$ used by PI-REM. This aims at showing that a good policy initialization is not sufficient to achieve the performance of PI-REM. The performance of each approach is measured in terms of rollouts performed on the real system, total time duration of the rollouts (real experience), and total training time (simulation and real). PI-REM and PILCO are implemented in Matlab®. In all the tests, the experted reward is measured using the saturating cost function in (10). The control policy in (9) is bounded in the interval $[-u_{max}, u_{max}]$ by the squashing function $\sigma(x) = u_{max}\frac{9\sin(x)+\sin(3x)}{8}$.



(a) (b)

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

Pendulum swing-up The goal is to find a control policy that balances the pendulum in Fig. 2(a) in the vertical position ($\theta_g = -\pi\,\text{rad}$), starting from $\theta_0 = 0\,\text{rad}$. Apart from swinging-up, the pendulum has to compensate for the external forces $\boldsymbol{f}_k$ generated by the interaction with an elastic environment. The interaction forces are generated by a spring placed at the vertical position $\theta = \pi\,\text{rad}$. We use the standard pendulum model $\ddot{\theta}_t(\frac{1}{4}ml^2 + I) + \frac{1}{2}mlg\sin(\theta_t) = u_t - b\dot{\theta}_t$ as approximate model, neglecting the external forces $\boldsymbol{f}_k$. The parameters of the pendulum are: the mass $m = 1\,\text{kg}$, the length $l = 1\,\text{m}$, the moment of inertia of the pendulum around the midpoint $I = \frac{1}{12}ml^2$, the friction coefficient $b = 0.01\,\text{sNm/rad}$, and the gravity acceleration $g = 9.81\,\text{m/s}^2$. u_t is the control torque. The vector $\boldsymbol{x} = [\dot{\theta}, \theta]^{\text{T}}$ is the state of the pendulum, while the aim is to reach the goal $\boldsymbol{x}_g = [0, -\pi]^{\text{T}}$. The elastic force $\boldsymbol{f}_k$ is assumed proportional to the stiffness of the spring. Results for three different stiffness values, namely 100, 200, and 500 N/m, are reported Tab. 1. It is worth noticing that the time of real experience in Tab. 1 is a multiple of the number of rollouts, since the duration of each rollout is fixed. On the contrary, the learning time does not grow linearly with the rollouts. This is because the time to train a GP model is not linearly growing with the size of the training data. In Tab. 1, PILCO+$\boldsymbol{\pi}^a$ indicates PILCO initialized with the policy $\boldsymbol{\pi}^a$, i.e. in the first rollout PILCO+$\boldsymbol{\pi}^a$ uses the policy learned for the approximate model instead of a random policy.

Stiffness 100 N/m: In this case, the control input is bounded to $u_{max} = 5\,\text{Nm}$, the total duration of each rollout is $T = 4\,\text{s}$, sampled at $dt = 0.1\,\text{s}$. As reported in Tab. 1, PI-REM needs 340 s of simulated time to find the policy $\boldsymbol{\pi}^a$ for the approximate model. Recall that $\boldsymbol{\pi}^a$ is used to perform the first rollut on the real system (see Algorithm 1). The time to find $\boldsymbol{\pi}^a$ depends on the used approach. One can exploit either a reinforcement learning or an optimal control technique to search for $\boldsymbol{\pi}^a$. For simplicity, in this work we exploit PILCO to search for a control policy for the approximate model. However, being the approximate model known, other choices like optimal control techniques are better suited in

Table 1. Results for the pendulum swing-up.

	Stiffness [N/m]	Real rollouts [#]	Real experience [s]	Simulation time [s]	Real time [s]
PI-REM	100	**2**	**8**	340	**212**
PILCO	100	6	24	**0**	1306
PILCO+$\boldsymbol{\pi}^a$	100	5	20	340	992
PI-REM	200	**3**	**12**	410	**410**
PILCO	200	4	16	**0**	624
PILCO+$\boldsymbol{\pi}^a$	200	4	16	410	624
PI-REM	500	**2**	**4**	602	**602**
PILCO	500	3	6	**0**	821
PILCO+$\boldsymbol{\pi}^a$	200	3	6	602	821

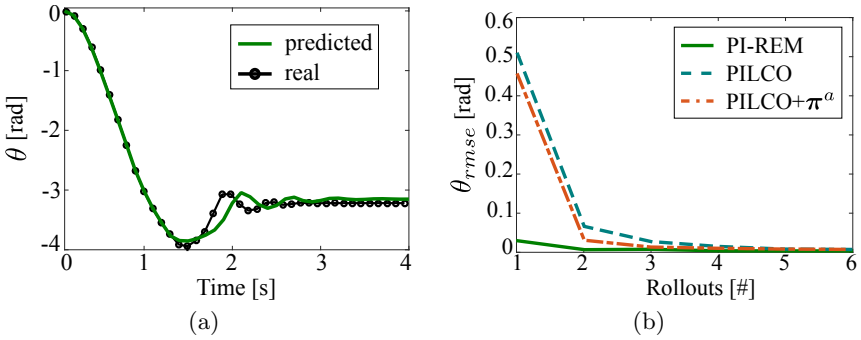


Fig. 3. Pendulum model learning results (stiffness 100 N/m). (a) State predicted and measured after one iteration of PI-REM. (b) Root mean square of the state prediction error obtained for PI-REM and PILCO in different rollouts.

this case. It is worth noticing that the simulation time does not require any human supervision and it is less expensive than performing rollouts on the real device. Regarding the interactions with the real system, PI-REM needs only 2 rollouts and 8 s of real experience to find a control policy, while PILCO spends 6 iterations and 24 s of real experience. Initializing PILCO with the optimal policy $\boldsymbol{\pi}^a$ computed for the approximate model slightly improves the performance of PILCO in terms of real rollouts and experience. However, the performance are worse than PI-REM. Indeed, PI-REM exploits approximate model and residual learning to reduce the state prediction error, rapidly finding a reliable model. This result is shown in Fig. 3. In particular, Fig. 3(a) shows that the approximate model properly represents the system until the pendulum touches the spring. As illustrated in Fig. 3(b), PI-REM properly estimates the state after 2 iterations. On the contrary, PILCO and PILCO+$\boldsymbol{\pi}^a$ need 4 and 5 iterations respectively to learn a proper model of the system. The policy found by PI-REM after 2 iterations is depicted in Fig. 4(a). As expected, the policy is bounded ($u(x) \in [-5, 5]$ Nm) and it is able to swing-up the pendulum and keep it in the unstable

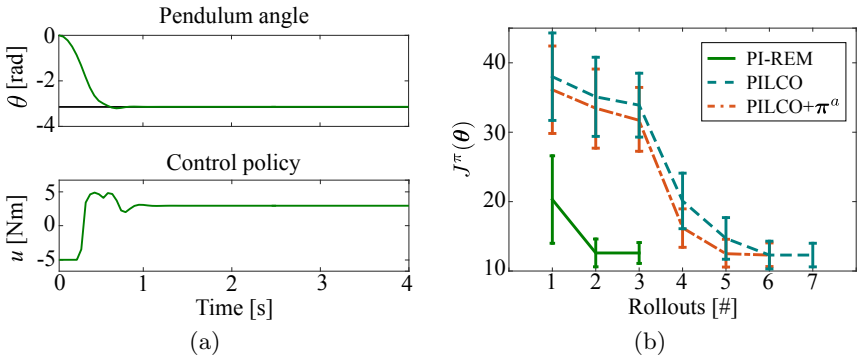


Fig. 4. Results for the pendulum swing-up (stiffness 100 N/m). (a) Pendulum angle and control policy obtained with PI-REM after 2 iterations. The black solid line represents the goal. (b) Evolution of the expected return $J^\pi(\boldsymbol{\theta})$ for different rollouts (mean and std over 5 executions).

position. The policy found by PILCO is not shown because, having used the same parameterization and cost function for the two approaches, PILCO and PI-REM learn (almost) the same policy. The evolution of the expected return for PI-REM, PILCO, and PILCO+$\boldsymbol{\pi}^a$ is illustrated in Fig. 4(b).

Stiffness 200 N/m: To properly simulate the system with a stiffness equal to 200 N/m, we reduce the sampling time to 0.05 s. This value is sufficient to avoid numerical instabilities in the integration of the real model dynamics. Moreover, the maximum control input is set to 15 Nm in order to fulfill the task. As shown in Tab. 1, PI-REM finds the control policy in 3 iterations and 12 s of real experience.

Stiffness 500 N/m: In this case, we further reduce the sampling time to 0.0125 s. The prediction horizon is also reduced to 2 s to speed-up the learning, while the maximum control input is set to 25 Nm. PI-REM finds the control policy in 2 iterations and 4 s of real experience.

Cart–pole swing-up The goal is to find a control policy that balances the pendulum on the cart in the inverted position ($\theta_g = -\pi$ rad), starting from $\theta_0 = 0$ rad. The policy is the horizontal force u_t that makes the cart moving to the left or to the right. The cart–pole model

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

serves as an approximate model. The real cart–pole system, instead, is connected to a wall through a spring (see in Fig. 2(b)). Hence, the approximate model neglects the additive elastic force $\boldsymbol{f}_k$. The mass of the cart is $m_c = 0.5$ kg, while the pendulum has mass $m_p = 0.5$ kg and length $l = 0.5$ m. The state of the cart–pole system is $\boldsymbol{x} = [p, \dot{p}, \theta, \dot{\theta}]^\mathrm{T}$, where p and $\dot{p}$ are the position and

Table 2. Results for the cart–pole swing-up.

	Stiffness [N/m]	Real rollouts [#]	Real experience [s]	Simulation time [s]	Real time [s]
PI-REM	25	**2**	**8**	996	**218**
PILCO	25	5	20	**0**	996
PILCO+$\boldsymbol{\pi}^a$	25	5	20	996	996
PI-REM	50	**3**	**12**	405	**405**
PILCO	50	6	24	**0**	1415
PILCO+$\boldsymbol{\pi}^a$	50	6	24	405	1415
PI-REM	120	**15**	**30**	405	**2328**
PILCO	120	23	46	**0**	6816
PILCO+$\boldsymbol{\pi}^a$	120	21	42	405	5623

the velocity of the cart, while θ and $\dot{\theta}$ are the angle and the angular velocity of the pendulum respectively. Hence, the state of the cart–pole system is four dimensional, i.e. double the dimension of the single pendulum state. Results for three different stiffness values, namely 25, 50, and 120 N/m, are shown in Tab. 2. As for the pendulum case, the time of real experience is a multiple of the number of rollouts, while the learning time does not grow linearly with the rollouts. In Tab. 2, PILCO+$\boldsymbol{\pi}^a$ indicates PILCO initialized with the policy $\boldsymbol{\pi}^a$.

Stiffness 25 N/m: In this case, the control input is bounded to $u_{max} = 10\,\text{N}$, the total duration of the task is $T = 4\,\text{s}$, sampled at $dt = 0.1\,\text{s}$. As reported in Tab. 2, PI-REM finds a control policy in 2 iterations and 8 s of real experience. For comparison, PILCO and PILCO+$\boldsymbol{\pi}^a$ need 5 iterations and 20 s of real experience.

Stiffness 50 N/m: In this case, the maximum control input has to be set to 15 N. PI-REM finds the policy in 3 rollouts and 12 s of real experience.

Stiffness 120 N/m: To properly simulate the system with a stiffness equal to 120 N/m, we reduce the sampling time to 0.05 s. This value is sufficient to avoid numerical instabilities in the integration of the real model dynamics. The prediction horizon is also reduced to $T = 2\,\text{s}$ to speed-up the learning process. With this settings PI-REM finds the control policy in 15 iterations and 30 s of real experience, as shown in Tab. 2. Figures 5(a) and 5(b) show that the learned policy drives the cart–pole system towards the goal $\boldsymbol{x}_g = [0, 0, \pi, 0]^{\text{T}}$. As expected, the control policy is bounded to $u(x) \in [-15, 15]\,\text{N}$ (see Fig. 5(c)). PILCO and PILCO+$\boldsymbol{\pi}^a$ need 23 and 21 rollouts respectively to find the optimal policy. The policy found by PILCO is not shown because, having used the same parameterization and cost function for the two approaches, PILCO and PI-REM learn (almost) the same control policy. The evolution of the expected return for PI-REM, PILCO, and PILCO+$\boldsymbol{\pi}^a$ is illustrated in Fig. 5(d).

In the considered simulations, PI-REM performs better than PILCO in terms of interactions with the real system. Hence, PI-REM increases the simulation time, where human intervention is not required, to significantly reduce the real time. In terms of rollouts on the real system, PI-REM takes from 25% to 67%

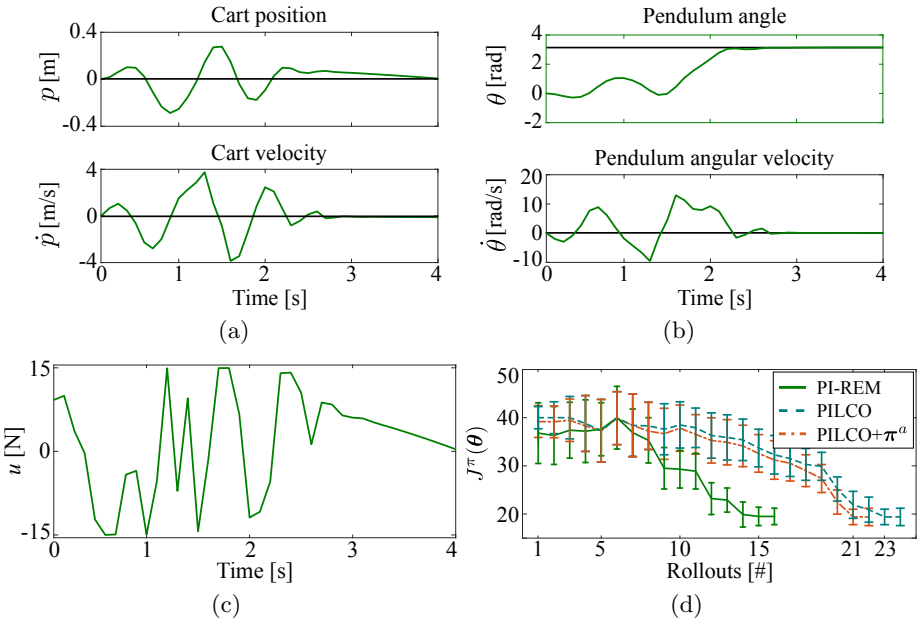


Fig. 5. Results for the cart–pole swing-up (stiffness 120 N/m). (a)–(b) State of the cart–pole system and (c) learned control policy after 15 iterations of PI-REM. The black solid line represents the goal. (d) Evolution of the expected return $J^{\pi}(\boldsymbol{\theta})$ for different rollouts (mean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

less iterations than PILCO. Consequently, the time of real experience is reduced from 25% to 67%, i.e. from a minimum of 2 s to a maximum of 16 s less than PILCO. Interestingly, results show that initializing PILCO with a proper policy, i.e. $\boldsymbol{\pi}^a$, is not sufficient to reach the performance of PI-REM. In other words, the combination of a proper policy initialization and residual dynamics learning is essential to rapidly converge to an optimal policy.

4 Conclusion and Future Work

This work demonstrated that combining a proper policy initialization and residual dynamics learning is beneficial to quickly find an optimal control policy. Such a combination is the key idea of PI-REM, the model-based and data-efficient reinforcement learning approach exploited in this study. PI-REM leverages a simplified model to find an initial policy, and sensory data to model the residual difference between simplified and real systems. In this work, PI-REM is compared with PILCO, a prominent approach for model-based reinforcement learning. Obtained results confirm our claim that a proper policy initialization alone is not sufficient to reach the performance of PI-REM in terms of performed rollouts. As summarized in Tab. 1 and 2, even when initialized with the control policy

found for the approximate model, PILCO needs more rollouts than PI-REM to find a control policy.

Our future research will investigate how accurate the approximate model has to be in order to maintain enhanced performance. We expect that a very inaccurate model will significantly affect the performance of PI-REM. In this case, the performance of PI-REM will be similar or slightly worse than classical black-box approaches for policy search like PILCO. Nevertheless, a formal quantification of the inaccuracies that the learning framework can tolerate without compromising the performance is still an open problem to investigate.

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. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics - Modelling, Planning and Control. Springer, Heidelberg (2009)
2. Levine, S., Abbeel, P.: Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. In: NIPS, pp. 1071–1079 (2014)
3. Saveriano, M., Yin, Y., Falco, P., Lee, D.: Data-Efficient Control Policy Search using Residual Dynamics Learning. In: IROS, pp. 4709–4715 (2017)
4. Peters, J., Muelling, K., Altun, Y.: Relative Entropy Policy Search. In: Conference on Artificial Intelligence (AAAI), pp. 1607–1612 (2010)
5. Farshidian, F., Neunert, M., Buchli, J.: Learning of Closed-Loop Motion Control. International Conference on Intelligent Robots and Systems, pp. 1441–1446 (2014)
6. Cutler, M., Jonathan P.H.: Efficient reinforcement learning for robots using informative simulated priors. In: ICRA, pp. 2605–2612 (2015)
7. Bischoff, B., Nguyen–Tuong, D., van Hoof, H., McHutchon, A., Rasmussen, C.E., Knoll, A., Peters, J., Deisenroth M.P.: Policy Search For Learning Robot Control Using Sparse Data. In: ICRA, pp. 3882–3887 (2014)
8. Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., Mouret, J.-B.: Black-Box Data-efficient Policy Search for Robotics. In: IROS, pp. 51–58 (2017)
9. Theodorou, E., Buchli, J., Schaal, S.: A generalized path integral control approach to reinforcement learning. J. of Mach. Lear. Res. 11, 3137–3181 (2010)
10. Kober, J., Bagnell, D., Peters, J.: Reinforcement learning in robotics: a survey. The International Journal of Robotics Research 32(11), 1238–1274 (2013)
11. Atkeson, C.G., Santamaria, J.C.: A Comparison of Direct and Model-Based Reinforcement Learning. In: ICRA, pp. 3557–3564 (1997)
12. Winter, F., Saveriano, M., Lee, D.: The Role of Coupling Terms in Variable Impedance Policies Learning. In: HFR (2016)
13. Saveriano, M., An, S., Lee, D.: Incremental Kinesthetic Teaching of End-Effector and Null-Space Motion Primitives. In: ICRA, pp. 3570–3575 (2015)

14. Atkeson, C.G., Schaal, S.: Robot Learning From Demonstration. In: International Conference on Machine Learning, pp. 12–20 (1997)
15. Deisenroth, M.P., Fox, D., Rasmussen, C.E.: Gaussian Processes for Data-Efficient Learning in Robotics and Control. TPAMI 37(2), 408–423 (2015)
16. Calinon, S., Kormushev, P., Caldwell, D.: Compliant Skills Acquisition and Multi-optima Policy Search with EM-based Reinforcement Learning. Robotics and Autonomous Systems 61(4), 369–379 (2013)
17. Schneider, J.G.: Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In: NIPS, pp. 1047–1053 (1996)
18. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
19. Kober, J., Peters, J.: Policy Search for Motor Primitives in Robotics. In: Advances in Neural Information Processing Systems, pp. 849–856 (2009)
20. Deisenroth, M.P.: Efficient Reinforcement Learning using Gaussian Processes. KIT Scientific Publishing (2010)