Model-based Reinforcement Learning using a simplified robot model

handed in
MASTER'S THESIS

Yuchao Yin

born on the 10.04.1990
living in:
Lerchenauer Str.9
80809 Munich
Tel.: 0176-97421914

Chair of
AUTOMATIC CONTROL ENGINEERING
Technical University of Munich

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Dirk Wollherr

Supervisor:	M.Sc.Matteo Saveriano
Start:	20.07.2016
Intermediate Report:	27.10.2016
Delivery:	13.02.2017

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
UNIV.-PROF. DONGHEUI LEE

M A S T E R ' S T H E S I S
for
Yuchao Yin
Student ID 03656508, Degree EI

Model-based Reinforcement Learning using a simplified robot model

Problem description:

The goal of Reinforcement learning (RL) is to make an agent able to autonomously learn how to perform a certain task. The ability of learning tasks by self-practice is appealing for robotics, and several RL approaches have been proposed in the past decade [1]. When applied to robotics, a RL algorithm has to face three main problems: *i)* control policies are, in general, continuous functions of the robot state, *ii)* the state space of the robot can be quite big (e.g. humanoid robots), *iii)* it is tedious and time consuming to perform thousands of rollouts on a real robotic platform. Aforementioned problems can be partially alleviated via control policy parameterization and model-based RL approaches adoptions. Among the other model-based RL approaches, the Probabilistic Inference for Learning Control (PILCO) [2, 3] has been proved to require few real rollouts to find the optimal control policy. PILCO learns a model of the robot from sensory data and iteratively improves the model and the policy until the task is satisfied. A limitation of PILCO is that it has to explore as much as possible the robot state space to learn a reliable model. This is usually achieved by randomly initializing the control policy during the first rollout, which can generate unpredictable behaviors in multi degrees-of-freedom robotic platforms.

In this Master's thesis work the student has to investigate the possibility of using a simplified dynamical model of the robot for model-based reinforcement learning. Eventual inaccuracies of the simplified model will be corrected by an additive term, learned from sensory data.

Tasks:

- Literature overview on model-free and model-based Reinforcement learning
- Model-based RL algorithm implementation
- Comparison with state-of-the-art approaches and experimental evaluation

Bibliography:

[1] J. Kober, J. A. Bagnell and J. Peters. Reinforcement Learning in Robotics: A Survey in *The International Journal of Robotics Research*, 2013.
[2] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-based and Data-Efficient Approach to Policy Search, in *International Conference on Machine Learning*, 2011.
[3] M. P. Deisenroth, D. Fox and C. E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control, in *Transactions on Pattern Analysis and Machine Intelligence*, 2015.

Supervisor:	M. Sc. Matteo Saveriano
Start:	20.07.2016
Intermediate Report:	27.10.2016
Delivery:	13.02.2017

(D. Lee)
Univ.-Professor

Abstract

Reinforcement learning becomes more popular in the recent years. However, in many applications of robotics and control, the state and action spaces are often continuous valued and high dimensional. Even to learn a particular task the number of interactions with its environment may be very huge. Thus, many researchers focus on increasing the data efficiency. Model-based approaches are more efficient than model-free approaches. However, to get an accurate model is difficult. PILCO algorithm is a model-based method and uses GP model to overcome the model-bias problem. It is also proved to be more efficient. In this work, we assume that only a simplified model is available. We calculate the difference between two data sets from the simplified model and the real model and use the data difference to learn a GP dynamics model. Based on the learned GP model we use analytic gradients for policy improvement. Compared with PILCO our algorithm needs less iterations to learn a task.

Contents

Chapter 1

Introduction

In the recent years, reinforcement learning (RL) becomes more popular in the area of robotics. It is a framework to let a robot autonomously discover an optimal behavior through interactions with its environment. However, RL suffers from two main disadvantages in many applications of robotics and control. First of all, the state and actions spaces are often continuous valued and also high dimensional. Moreover, the number of interactions with its environment may impractically high. To increase data efficiency is one of the most important tasks in reinforcement learning.

Basically RL can be classified into two approaches: model-free approach and model-based approach. Model-free methods like Q-learning [1] or TD-learning [2] directly learn the policy on the data samples. The authors in [3] proposed a novel RL approach, namely CP-I^2, to simultaneously learn trajectories and variable impedance behaviors. As model free approaches, they don't require any model approximations and are applicable in many situations. One main problem is that the number of interactions with the real environment may be very huge. As in [4], even for a simple task with only two parameters, hundreds rollouts are required.

On the other hand, model-based methods estimate the transition model of the environment and use this learned model to search the (optimal) policy. They are very efficient compared with model-free methods. However, model-based methods strongly suffer from model-bias problem. They assume that the learned model for real environment is sufficiently accurate [5] [6] [7]. Optimization of inaccurate models might lead to disaster.

Model bias is especially a problem when the training data is too sparse. See figure 1.1, given a small data set of observed data, there are multiple transition functions which could have generated them [8]. Simply choose one of them will cause problems during long-term predictions.

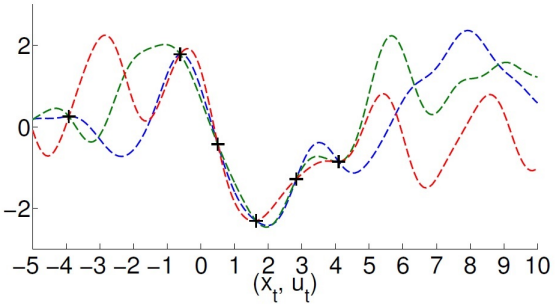


Figure 1.1: Effect of model errors: multiple plausible deterministic functions [8]

The authors in [9] use a probabilistic dynamics model, namely non-parametric Gaussian processes, to express the model uncertainty. Based on the learned GP model, the expected long-term cost can be predicted. The policy improvement is based on the analytic policy gradients of the cost function.

An accurate model in robot control is often very hard to obtain. For the sake of reducing the number of interactions with real robot and try to be more data efficient, we propose a new method using a simplified robot model in this work. We assume that the accurate model is not available and only a simplified model can be used. The initial policy is the optimal policy for the simplified model. We can get the difference of two data sets by applying the initial policy to the real environment and also to the simplified model. The difference is considered as the input of GP model. Therefore, GP model is to be learned for the sake of compensating the inaccuracy of the simplified model. The next step is to improve the policy using the learned GP model until the task is learned. Our new method is more efficient than PILCO [9] and needs less interactions with the real environment.

Chapter 2

Related work

The parameter uncertainty has been taken into consideration in robust and adaptive control [10] [11]. The authors in [12] proposed a new approach which takes into consideration the uncertainty of the model parameters by adding an extra term in the cost function of a minimum-variance controller. Through this, it has no requirements to perform prior open-loop plant identification.

Kolter [13] proposed an algorithm which uses inaccrate models to find good policies. In [13] an approximate policy gradient can be computed by simply substituting the signs for the derivatives themselves. The algorithm works well in many situations and requires no accurate for learning robot dynamics.

Using Gaussian processes dynamics models in Reinforcement Learning were previously presented in [14][15][16] . These methods require the training data, which is obtained either by motor babbling [14] or by demonstrations [16] . However, motor babbling is data-inefficient and demonstrations would contradict fully autonomous learning [14] [16].

The authors of [17][18] also proposed algorithms with GP dynamics models in Reinforcement Learning. Unfortunately, [17][18] cannot take constraints of the state space into account because of value function models. [19] proposed a novel Bayesian approach for estimation of value function in continuous spaces. Also [20] needs an explicit value function model. They all suffer from the same problem.

In Reinforcement Learning, an aggressive exploration is encouraged. However, in some cases this will cause catastrophic failures. [5] proposed a new algorithm to exploit model uncertainty estimates for safe dynamic control. In [5] the model uncertainty is treated as additive uncorrelated noise.

The authors of [9] propose PILCO algorithm, which uses a probabilistic dynamics model to overcome the model-bias problem. PILCO employs Gaussian processes

to express the model uncertainty and incorporate long-term model uncertainty into planning and policy evaluation. PILCO can gain unprecedented data efficiency and can be directly applied to robots.

However, PILCO needs to explore the state space as much as possible to gain a reliable model. The computational cost increases dramatically while the data sample size increases. States and actions need to also follow Gaussian distributions and the reward function should be an exponential form to guarantee that the policy evaluation is performed in a closed form and the policy gradients are computed analytically for policy improvement [21]. These requirements restrict PILCO in many situations.

Chapter 3

Model-based policy search

Through this work, we consider dynamical systems $x_t = f(x_{t-1}, u_{t-1}) + w$ with continuous-valued states $x \in R^D$, controls $u \in R^F$, addtive Gaussian noise w and unknown transition function f . The policy search objective is to find a controller $\pi : x \to \pi(x) = u$ which minimizes the expected return

$$J^{\pi}(\theta) = \Sigma_{t=0}^{T} E(x_t)[c(x_t)], \tag{3.1}$$

where $c(x_t)$ is the cost of being in state x at time t. We assume that function π is parameterized by θ.

We assume that a real dynamical system (indexed by R) is

$$x_t^R = f^R(x_{t-1}^R, u_{t-1}^R) + w. \tag{3.2}$$

Unfortunately, due to model uncertainties, it is hard to have a correct real model. Instead of exploring as much data as possible from scratch, in this work we decide to use a simplified approximate model (indexed by A) as

$$x_t^A = f^A(x_{t-1}^A, u_{t-1}^A) + w. \tag{3.3}$$

Then the difference of these two models can be written as

$$x_t^D = f^R(x_{t-1}^R, u_{t-1}^R) - f^A(x_{t-1}^A, u_{t-1}^A) + w = f^D(x_{t-1}^D, u_{t-1}^D) + w, \tag{3.4}$$

where $x_t^D = x_t^R - x_t^A$ and $u_t^D = u_t^R - u_t^A$. We also assume that the difference of these two models also follows Gaussian distribution.

At the beginning, we apply the initial policy $\pi_{initial}$ both to the approximate model and the real robot. After we have two data sets (x_t^A, u_t^A) and (x_t^R, u_t^R), we calculate

the difference of them and obtain tuples (x_t^D, u_t^D) which are considered as the input of Gaussian processes.

The real expected values in (3.1) can be written as

$$\mathrm{E}_{x_t^R}[c(x_t^R)] = \int c(x_t^R)\mathcal{N}(x_t^R|\mu_t^R, \Sigma_t^R)\mathrm{d}x_t^R, \tag{3.5}$$

where $c(x_t^R)$ is the cost function.

In this work, we implement a saturating cost function with spread σ_c,

$$c(x_t^R) = 1 - \exp(-\frac{1}{\sigma_c}||x_t^R - x_{target}||^2), \tag{3.6}$$

where x_{target} denotes the target state.

We substitute $x_t^D = x_t^R - x_t^A$ into the following function and we obtain

$$\begin{aligned} c(x_t^R) &= 1 - \exp((-\frac{1}{\sigma_c}||x_t^R - x_{target}||^2)) \\ &= 1 - \exp(-\frac{1}{\sigma_c}||x_t^D + x_t^A - x_{target}||^2)) \\ &= 1 - \exp(-\frac{1}{\sigma_c}||x_t^D - (x_{target} - x_t^A))||^2)) \\ &= 1 - \exp(-\frac{1}{\sigma_c}||x_t^D - x_{target,t}^D)||^2)) \\ &= c(x_t^D) \end{aligned}$$

Since x_t^A is the approximate state which is extracted from the initial policy, it is deterministic. We can directly calculate the $x_{target,t}^D$, where $t = 0...T$. From now on the expected real long-term cost can be rewritten as

$$J^\pi(\theta) = \Sigma_{t=0}^T \mathrm{E}_{x_t^D}[c(x_t^D)] = \Sigma_{t=0}^T \mathcal{E}_t^D. \tag{3.7}$$

3.1 GP dynamics model learning

The probabilistic dynamics model is implemented as a GP, where we use tuples $(x_{t-1}^D, u_{t-1}^D) \in R^{D+F}$ as training inputs and differences $\Delta_t^D = x_t^D - x_{t-1}^D + \epsilon \in R^D$as training targets, where$\epsilon \sim \mathcal{N}(0, \Sigma_\epsilon), \Sigma_\epsilon = \text{diag}([\sigma_{\epsilon_1}, ..., \sigma_{\epsilon_D}])$.

The covariance function is the squared exponential function which is defined as

$$k(x, x') = \alpha^2 \text{exp}(-\frac{1}{2}(x - x')^T \wedge^{-1} (x - x')) \tag{3.8}$$

with $x := [x^T u^T]^T$. We define α^2 as variance of the latent function f and $\wedge := \mathsf{diag}([l_1^2, ..., l_D^2])$.

Given n training inputs $X^D = [x_1^D, ..., x_n^D]$ and corresponding training targets $y = [\Delta_1^D, ..., \Delta_n^D]^T$, the GP hyper-parameters are learned by evidence maximization [22].

The GP yields one-step prediction, the predicted successor state x_{t+1}^D is Gaussian distributed

$$p(x_{t+1}^D | x_t^D, u_t^D) = \mathcal{N}(x_{t+1}^D | \mu_{t+1}^D, \Sigma_{t+1}^D), \tag{3.9}$$

$$\mu_{t+1}^D = x_t^D + \text{E}(\Delta_t^D), \Sigma_{t+1}^D = \text{var}_f[\Delta_t^D], \tag{3.10}$$

where the mean and variance of the GP prediction are

$$\text{E}_f[\Delta_t^D] = m_f(x_t^D) = k_\star^T (K + \sigma_w^2 I)^{-1} y = k_\star^T \beta, \tag{3.11}$$

$$\text{var}_f[\Delta_t^D] = k_{\star\star} - k_\star^T (K + \sigma_w^2 I)^{-1} k_\star, \tag{3.12}$$

where $k_\star := k(X^D, x_t^D)$, $k_{\star\star} := k(x_t^D, x_t^D)$ and $\beta := (K + \sigma_w^2 I)^{-1} y$, where K is the kernel matrix with entries $K_{ij} = k(x_i^D, x_j^D)$.

3.2 Policy evaluation

The learned GP model is used to compute the approximate long-term predictions $p(x_1^D|\pi), ..., p(x_T^D, |\pi)$ for a given controller π. Once the long-term predictions are computed and a real state can be calculated as $x_t^R = x_t^D + x_t^A$, where x_t^A is deterministic. The expected real long-term cost

$$J^\pi(\theta) = \Sigma_{t=0}^T \mathrm{E}_{x_t^R}[c(x_t^R)] = \Sigma_{t=0}^T \mathrm{E}_{x_t^D}[c(x_t^D)] \tag{3.13}$$

can be computed analytically.

3.3 Policy Improvement

In this work, we turn to CMA-ES for the simplified model to find an initial optimal policy. During the learning process, we use analytic gradients method for policy improvement. We describe the main approaches in the following subsections.

3.3.1 Optimization by using CMA-ES

The Covariance Matrix Adaption Evolution Strategy (CMA-ES) developed by Hansen and Ostermeier is a stochastic method for numerical optimization of non-linear or non-convex continuous optimization problem [23]. It does not presume the existence of the gradients. Thus it is suitable in many situations.

In the CMA-ES, a population of new candidate solutions is generated by sampling a multivariate normal distribution:

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)} \mathcal{N}(0, C^{(g)}), \qquad \text{for } k = 1, ..., \lambda \tag{3.14}$$

where m^g is the mean value and $C^{(g)}$ is the covariance matrix of the search distribution at generation g [23]. They are adapted in each generation along with the general step size σ.

The new mean $m^{(g+1)}$ of the search distribution is defined as a weighted average of several best-ranking individuals of the last generation [23]. We assume that the population contains enough information to reliably estimate the covariance matrix C. Its updating is fairly sophisticated and the details can be found in [24].

For convenience, the CMA-ES method is designed so that it doesn't need many user interactions and manual setting of parameters. The population size λ and the initial step size $\sigma^{(0)}$ can be changed at the beginning. The parent population size is defined as $\mu = \frac{1}{2}\lambda$. The population size is by default very small: $\lambda_{def} = 4 + \lfloor 3 \log(n) \rfloor$, where

n is the problem dimension.

Since the optimal policy is learned for approximate model, we consider this policy as our initial policy $\pi_{initial}$.

3.3.2 Gradient-based Optimization

We use gradient information $\mathrm{d}J^{\pi}(\theta)/\mathrm{d}\theta$ to search policy parameters θ, which minimize the expected long-term cost function in (3.7). We assume the cost function is differentiable and also the control distribution μ_u^D and Σ_u^D are also differentiable and can be computed analytically.

With $\mathcal{E}_t^D := \mathrm{E}_{x_t^D}[c(x_t^D)]$ the gradient $\mathrm{d}J^{\pi}/\mathrm{d}\theta$ can be written as

$$\frac{\mathrm{d}J^{\pi}(\theta)}{\mathrm{d}\theta} = \Sigma_{t=0}^{T} \frac{\mathrm{d}\mathcal{E}_t^D}{\mathrm{d}\theta}. \tag{3.15}$$

.

Then we apply the chain-rule to the gradient and we obtain

$$\frac{\mathrm{d}\mathcal{E}_t^D}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{E}_t^D}{\mathrm{d}p(x_t^D)} \frac{\mathrm{d}p(x_t^D)}{\mathrm{d}\theta} = \frac{\partial \mathcal{E}_t^D}{\partial \mu_t^D} \frac{\mathrm{d}\mu_t^D}{\mathrm{d}\theta} + \frac{\partial \mathcal{E}_t^D}{\partial \Sigma_t^D} \frac{\mathrm{d}\Sigma_t^D}{\mathrm{d}\theta}, \tag{3.16}$$

where $p(x_t^D) = \mathcal{N}(x_t^D | \mu_t^D, \Sigma_t^D)$.

We continually apply the chain-rule to (3.16) and we have

$$\frac{\mathrm{d}p(x_t^D)}{\mathrm{d}\theta} = \frac{\partial p(x_t^D)}{\partial p(x_{t-1}^D)} \frac{\mathrm{d}p(x_{t-1}^D)}{\mathrm{d}\theta} + \frac{\partial p(x_t^D)}{\partial \theta}, \tag{3.17}$$

$$\frac{\partial p(x_t^D)}{\partial p(x_{t-1}^D)} = \left\{ \frac{\partial \mu_t^D}{\partial p(x_{t-1}^D)}, \frac{\partial \Sigma_t^D}{\partial p(x_{t-1}^D)} \right\}. \tag{3.18}$$

Computing $\mathrm{d}\Sigma_t^D/\mathrm{d}\theta$ is similar to computing $\mathrm{d}\mu_t^D/\mathrm{d}\theta$. We calculate the derivative for $\mathrm{d}\mu_t^D/\mathrm{d}\theta$ as

$$\frac{\mathrm{d}\mu_t^D}{\mathrm{d}\theta} = \frac{\partial \mu_t^D}{\partial \mu_{t-1}^D} \frac{\mathrm{d}\mu_{t-1}^D}{\mathrm{d}\theta} + \frac{\partial \mu_t^D}{\partial \Sigma_{t-1}^D} \frac{\mathrm{d}\Sigma_{t-1}^D}{\mathrm{d}\theta} + \frac{\partial \mu_t^D}{\partial \theta}, \tag{3.19}$$

where

$$\frac{\partial \mu_t^D}{\partial \theta} = \frac{\partial \mu_{\Delta}^D}{\partial p(u_{t-1}^D)} \frac{\partial p(u_{t-1}^D)}{\partial \theta}, \tag{3.20}$$

where $p(u_t^D)$ is given as $\mathcal{N}(u_t^D|\mu_u^D, \Sigma_u^D)$ and the corresponding partial derivatives depend on the policy representation.

By applying the chain-rule the above individual partial derivatives can be computed analytically [25].

After policy improvement, we apply the new policy π to the real robot and obtain data set X^R. We calculate the difference between the original data set X^A. The new difference set X^D is recorded for the next iteration until the task is learned.

The general steps of our algorithm are showed as follows:

Algorithm Policy Search with Approximate Model

---

1. **init:** Apply initial optimal policy $\pi_{initial}$ both to
   the approximate model and the real robot.
2. Record X^A and X^R, calculate the difference X^D.
3. Calculate the new target set $x_{target}^D = x_{target} - x^A$.
4. **REPEAT**
5. Learn GP dynamics model for the model difference.
6. Policy search:
7. **REPEAT**
8. Approximate inference for policy evaluation.
9. Policy improvement
10. Update control parameters.
11. **UNTIL** convergence; **RETURN** parameters.
12. **SET** new policy π.
13. Apply new policy π to the real robot and record data X^R.
14. Calculate the new model difference X^D.
15. **UNTIL** task learned

Chapter 4

Experimental Results

In this Chapter, we implemented our algorithm to two different cases, namely single pendulum and cart pole swing up. However, in our cases two systems are more complicated than the original settings. The single pendulum system has an additive spring and the cart pole system have an elastic band, which connects the middle of the track with the middle of the cart.

Compared with standard PILCO, our Algorithm is proved to be more efficient.

In the following, we use a saturating cost function with spread σ_c, i.e.,

$$c(x) = 1 - \mathsf{exp}(-\frac{1}{2}\sigma_c^2||x - x_{target}||^2) \qquad \in [0, 1], \tag{4.1}$$

where x_{target} is a target state.

The policy is concatenation of two functions: the RBF controller, which is parametrized as the mean of a GP with a squared exponential covariance function, and squashing function

$$\sigma(x) = u_{max}\frac{9\mathsf{sin}x + \mathsf{sin}(3x)}{8}, \tag{4.2}$$

which is the third-order Fourier series expansion of a trapezoidal wave, normalized to the interval $[-u_{max}, u_{max}]$.

4.1 Single pendulum

We implemented the new algorithm to a complicated pendulum system. As showed in figure 4.1, there is an additive part on the top of the system. When the ball moves upwards and touches the spring, it will get an opposing force against itself. Typical values are: $m = 1\mathsf{kg}$ and $l = 1\mathsf{m}$. The task is to control the torque u in order to balance the ball in the inverted position with $\phi = -\pi$.

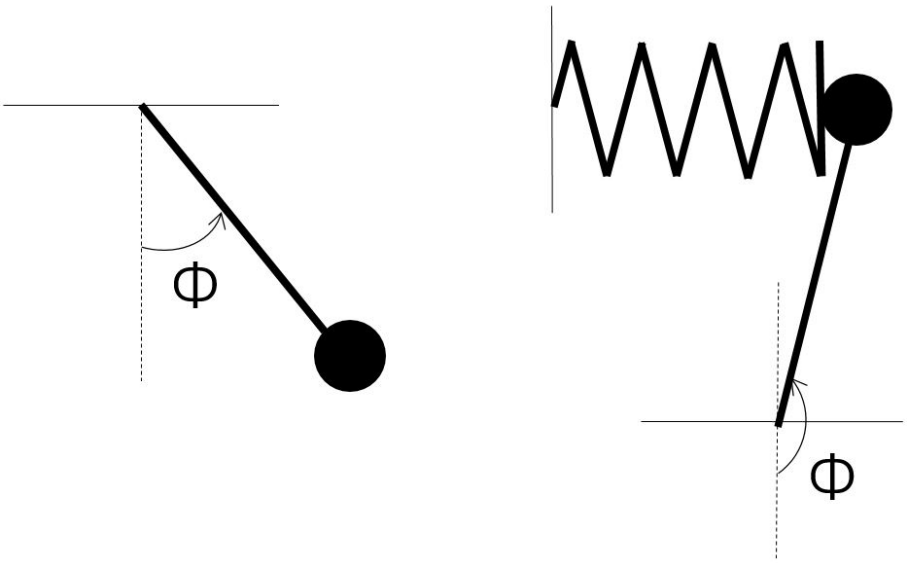


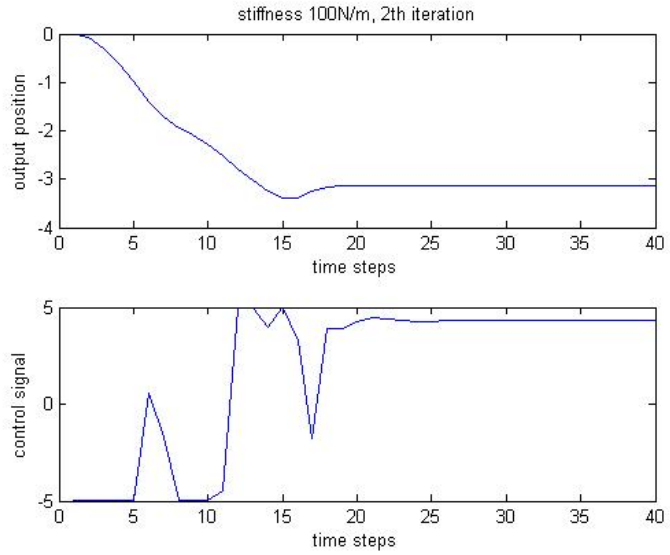
Figure 4.1: Pendulum with spring

The simplified model that we used is defined as

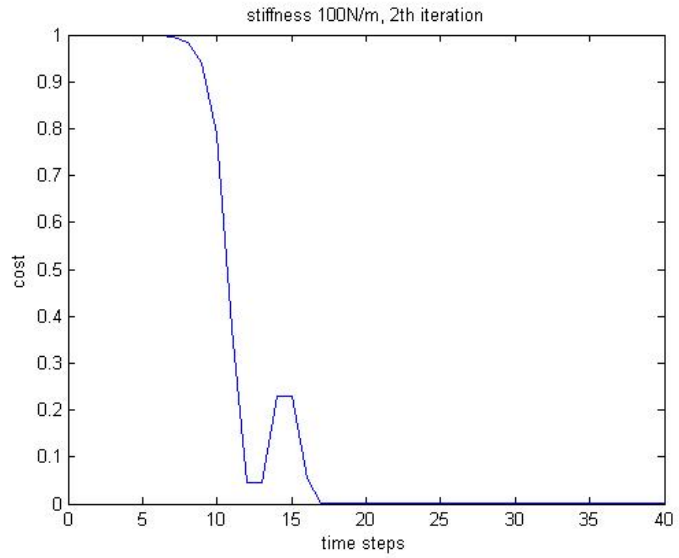
$$\ddot{\phi}\,(\frac{1}{4}ml^2 + I) + \frac{1}{2}mlg\mathsf{sin}\phi = u - b\,\dot{\phi}, \tag{4.3}$$

where g is the acceleration of gravity, $I = \frac{1}{12}ml^2$ is the moment of inertia of a pendulum around the pendulum midpoint and b is a friction coefficient. If the ball moves towards the spring it will get an opposing force F. Theoretically the torque u should be increased to compensate it.
The state is defined as $x = [\dot{\phi}, \phi]$

Stiffness 100 N/m First we set the stiffness to 100. u_{max} is by default 5 and prediction time $T = 4$s and sampling time $dt = 0.1$s. Only after 2 iterations, the ball can stay in the inverted position with $\phi = -180$. Figure 4.2 shows the ball position and the corresponding torque value.



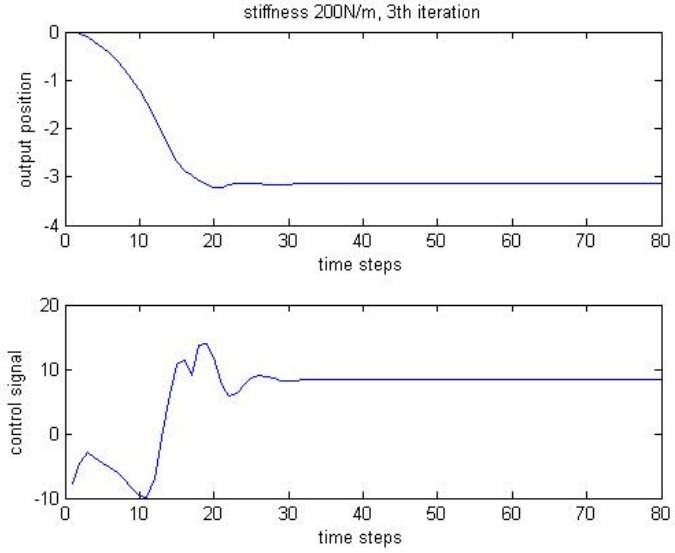
(a) Output position and control signal



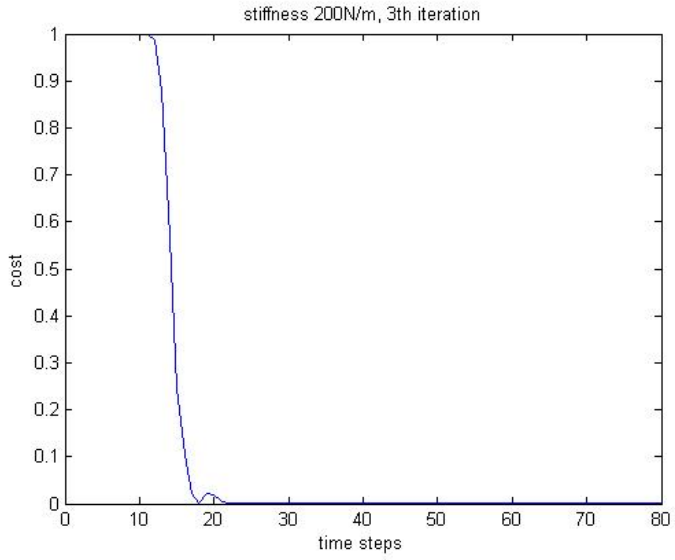
(b) Cost value

Figure 4.2: Results of stiffness 100 N/m

Stiffness 200 N/m With the stiffness 200 we reduce the sampling time to 0.05s. The maximum control signal is 15. The task can be learned until 3th iteration.Figure 4.3 shows the results.



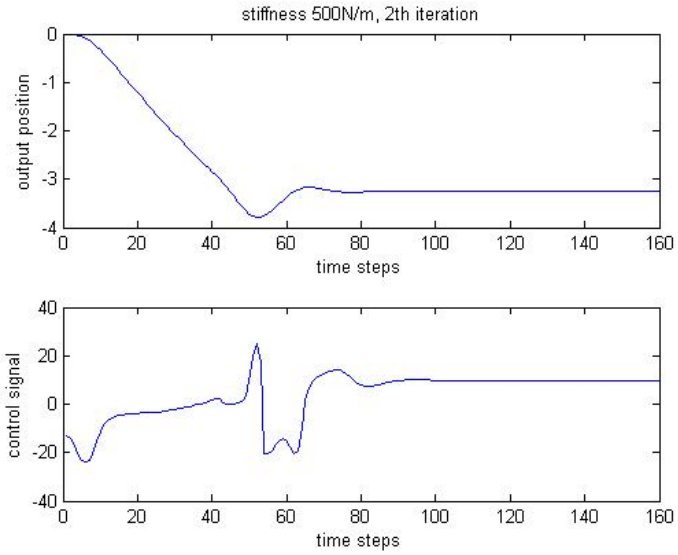
(a) Output position and control signal



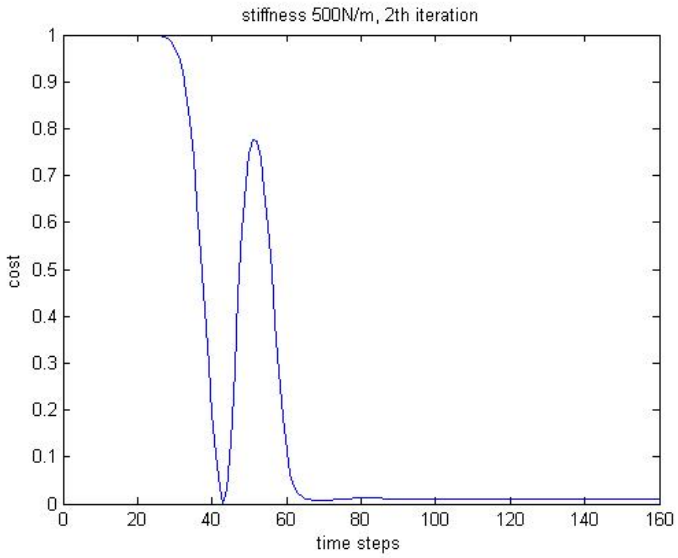
(b) Cost value

Figure 4.3: Results of stiffness 200 N/m

Stiffness 500 N/m When the stiffness is big enough, the sampling time should be reduced. We set the sampling time to 0.0125s and to save the computational cost the prediction time is also reduced to 2s. To compensate the spring force successfully we enlarge the maximum torque to 25. Final results are showed in figure 4.4.



(a) Output position and control signal



(b) Cost value

Figure 4.4: Results of stiffness 500 N/m

We recorded the total time for policy learning and compared it with the standard PILCO algorithm as showed in the following tables.

Stiffness 100 N/m

	Number of iterations	Total used time
Our Algorithm	2	212 s
PILCO	6	1306 s

Stiffness 200 N/m

	Number of iterations	Total used time
Our Algorithm	3	410 s
PILCO	4	624 s

Stiffness 500 N/m

	Number of iterations	Total used time
Our Algorithm	2	602 s
PILCO	3	821 s

In this experiment, we implemented our algorithm to three different situations by changing the spring stiffness. When the stiffness is enlarged, the system is more 'sensitive'. The input control torque should be enlarged to compensate the force of the spring and the sampling time should also be reduced.

In the next section, we applied our algorithm to a cart-pole swing-up system, which is more complicated than single pendulum.

4.2 Cart-pole swing-up

We also applied our algorithm to a cart-pole swing-up system. As showed in 4.5, the middle of the cart is connected to the middle of the track through an elastic band. The additive force of the spring will affect the cart and makes the system complicated. Through a horizontal force u the cart can move along a track and swing up a pendulum and to balance it in the inverted position in the middle of the track.

The mass of the cart is 0.5kg, the mass of the pendulum is 0.5kg and the length of the pendulum is 0.5m. The state consists of the position of the cart, the velocity of the cart, the angle of the pendulum and the angular velocity of the pendulum.

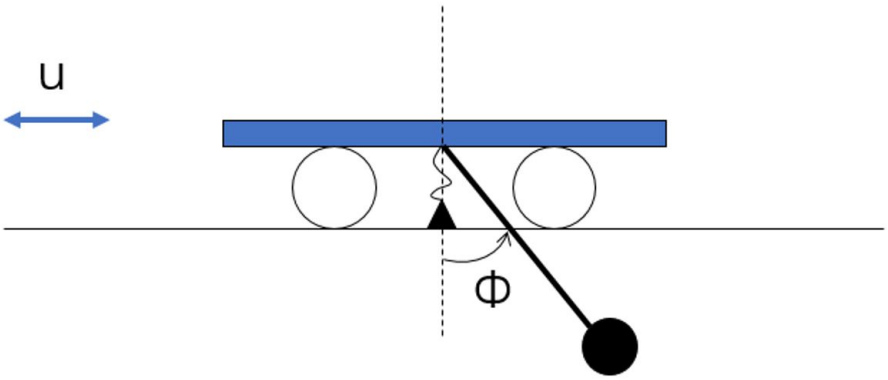
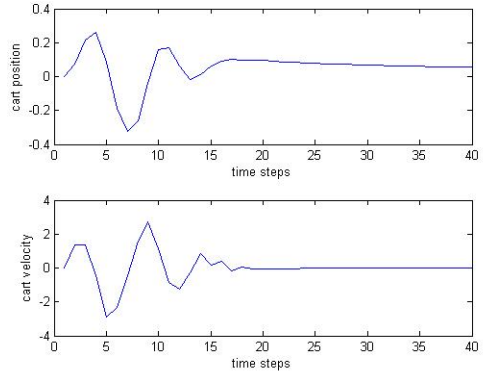


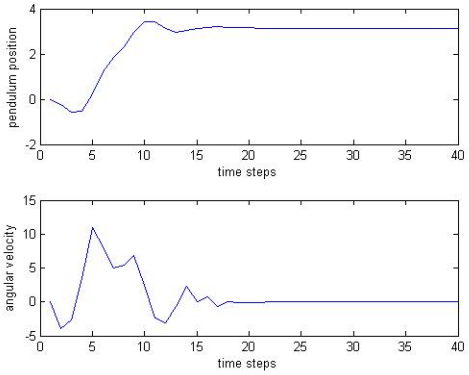
Figure 4.5: Cart-pole with an elastic band

We also changed the stiffness value to see if our algorithm is applicable in different situations.

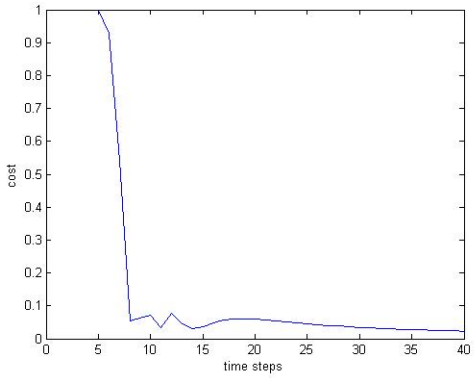
Stiffness 25 N/m First of all, we use an elastic band with stiffness 25 N/m. Our algorithm only needs 2 iterations while the standard PILCO algorithm needs 5 iterations. Figure 4.6 shows the results.



(a) Cart position and cart velocity



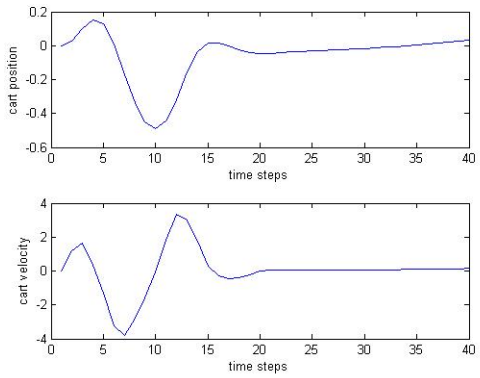
(b) Pendulum position and angular velocity



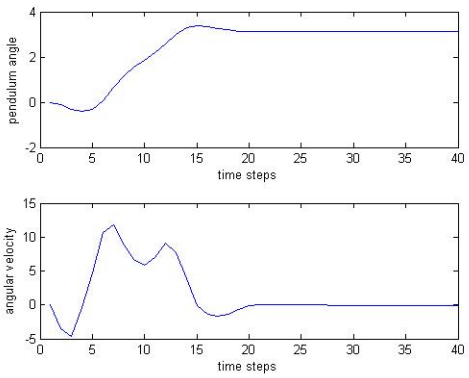
(c) Cost value

Figure 4.6: Cart-pole swing-up stiffness 25 N/m after 2 iterations

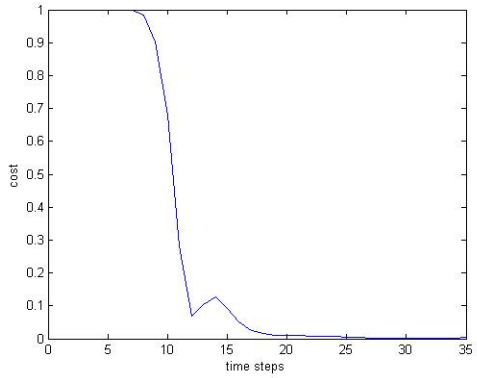
Stiffness 50 N/m In this case we change the stiffness to 50 N/m. And we enlarge the maximum force to compensate the additive force of elastic band. Our algorithm only needs 3 iterations to learn the policy. The results are showed in figure 4.7.



(a) Cart position and cart velocity



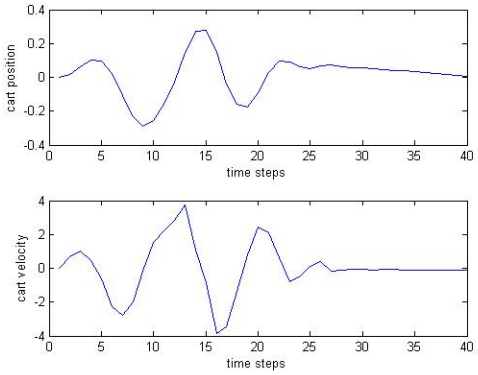
(b) Pendulum position and angular velocity



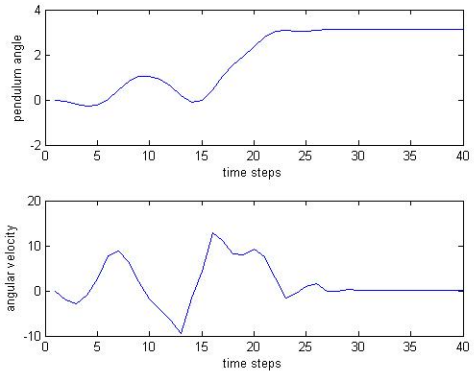
(c) Cost value

Figure 4.7: Cart-pole swing-up stiffness 50 N/m after 3 iterations

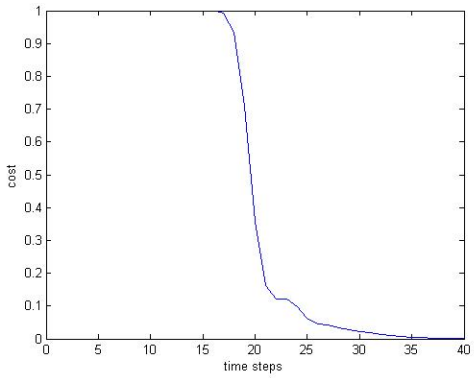
Stiffness 120 N/m When we enlarge the stiffness of the elastic band, it needs more iterations to find an optimal policy. Now maximum control signal is set to 15 and we also reduce the sampling time to $0.05s$. As showed in figure 4.8, after 15 iterations it can finally find the optimal policy.



(a) Cart position and cart velocity



(b) Pendulum position and angular velocity



(c) Cost value

Figure 4.8: Cart-pole swing-up stiffness 120 N/m after 15 iterations

We also recorded the total time for policy learning and compared it with the standard PILCO algorithm as showed in the following tables.

Stiffness 25 N/m

	Number of iterations	Total used time
Our Algorithm	2	218 s
PILCO	5	996 s

Stiffness 50 N/m

	Number of iterations	Total used time
Our Algorithm	3	405 s
PILCO	6	1415 s

Stiffness 120 N/m

	Number of iterations	Total used time
Our Algorithm	15	2328 s
PILCO	23	6816 s

When the stiffness is enlarged, we reduce the sampling time and enlarge the control input. So the size of the training data set will also be increased. Consequently the evaluation time for policy learning will be longer. In cart-pole swing-up with an elastic band system, our algorithm is also proved to be more data efficient than PILCO algorithm.

Chapter 5

Discussion

Model-based policy search method suffers from model-bias problem. Using an inaccurate model for policy learning might damage the robot. Our algorithm uses a GP forward model to account for model-bias problem . Based on the learned GP model we compute the gradients of an approximation to the expected long-term return J^{π} for policy search. Using gradient-based non-convex optimization methods like 'LBFGS' the optimized policy parameter θ^{*} can be obtained [9].

The accuracy of the learned GP model plays a significant role for policy search. PILCO needs to explore as much data as possible to get an available model. Thus, the computational cost increases dramatically in the learning process. However, our algorithm provides us with the opportunity to directly explore data near target state. Therefore, it is more data efficient.

However, it is not an optimal control method but only a solution for a specified task. In some cases, different control trajectories may have the same consequence, i.e., the target state can be reached successfully. That's why we stop the learning process as long as the task is learned. Our algorithm only finds the local optimum because the optimization method, which is used for learning the policy parameters is non-convex.

Control signals of real applications are often bounded. This fact should also be taken into account during planning. Thus, we use a squashing function for policy π, which limits the policy to the interval $[-u_{max}, u_{max}]$.

In classical control task, a quadratic cost function is often be used. Unfortunately, a quadratic cost function only focuses on the worst deviation along the predicted trajectory. In the early stage of learning process the model uncertainty can not be ignorable, what makes gradients of the cost function more useless. Moreover, because we use analytic gradients for policy improvement, the policy function and the cost function need to be differentiable.

Chapter 6

Conclusion

In this work, we propose a new model-based approach with a simplified model. As long as we have a simplified model, we can directly apply the CMA-ES optimization method to find an optimal policy $\pi_{initial}$. Obviously, this policy is not the optimal policy for the real robot.

Instead of directly scratching data through interaction with real environment, we apply the policy $\pi_{initial}$ both to the real robot and the approximate model. Since we have two data sets, we calculate the difference of these two sets and learn the GP dynamics model using this difference. In this way, more prior information of the system near target state can be obtained. Therefore, it shows more data efficiency than PILCO algorithm.

The authors in [26] employed PILCO algorithm to constrained environments. This is very useful and also realistic in real robot applications. In the future work, we can also take some constraints into consideration and make our algorithm more flexible.

We applied our algorithm to two different tasks, namely single pendulum and cart-pole swing-up. The results of both tasks are proved that our algorithm has gained unprecedented speed of learning.

List of Figures

Bibliography

[1] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[2] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[3] M. S. Florian Winter and D. Lee, "The role of coupling terms in variable impedance policies learning," in *9th International Workshop on Human-Friendly Robotics(HFR)*, 2016.

[4] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems 21*, (Red Hook, NY, USA), pp. 849–856, Max-Planck-Gesellschaft, Curran, June 2009.

[5] J. G. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," in *in Neural Information Processing Systems 9*, pp. 1047–1053, The MIT Press, 1996.

[6] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, (San Francisco, CA, USA), pp. 12–20, Morgan Kaufmann Publishers Inc., 1997.

[7] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *IN INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pp. 3557–3564, IEEE Press, 1997.

[8] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 408–423, Feb 2015.

[9] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *In Proceedings of the International Conference on Machine Learning*, 2011.

[10] D.C.McFarlane and K. Glover., "Lecture notes in control and information sciences."

[11] K. J. Astrom and B. Wittenmark, *Adaptive Control*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1994.

[12] S. Fabri and V. Kadirkamanathan, *Dual Adaptive Control of Nonlinear Stochastic Systems using Neural Networks*, vol. 2, pp. 245–253. 1998.

[13] J.Z.Kolter, *Learning and control with inaccurate models*. PhD thesis.

[14] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 742–747, April 2007.

[15] J. Ko and D. Fox, *Learning GP-Bayesfilters via Gaussien process latent variable models*, vol. 5, pp. 225–232. MIT Press Journals, 2010.

[16] D. Nguyen-tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression."

[17] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16*, (Cambridge, MA, USA), pp. 751–759, Max-Planck-Gesellschaft, MIT Press, June 2004.

[18] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomput.*, vol. 72, pp. 1508–1524, Mar. 2009.

[19] Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: The gaussian process approach to temporal difference learning.," in *ICML* (T. Fawcett and N. Mishra, eds.), pp. 154–161, AAAI Press, 2003.

[20] A. Wilson, A. Fern, and P. Tadepalli, "Incorporating domain models into bayesian optimization for rl," in *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part III*, ECML PKDD'10, (Berlin, Heidelberg), pp. 467–482, Springer-Verlag, 2010.

[21] T. Z. J. M. M. S. Voot Tangkaratt, Syogo Mori, "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation," *Neural Networks*, vol. 57, pp. 128–140, 2014.

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

[23] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, pp. 159–195, June 2001.

[24] N. Hansen, "The CMA evolution strategy: A tutorial," *CoRR*, vol. abs/1604.00772, 2016.

[25] M. Deisenroth, *Efficient Reinforcement Learning using Gaussian Processes*. KIT Scientific Publishing, 2010.

[26] M. Deisenroth, C. Rasmussen, and D. Fox, *Learning to control a low-cost manipulator using data-efficient reinforcement learning*, vol. 7, pp. 57–64. MIT Press Journals, 2012.

License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit http://creativecommons.org or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.