FP: Model-based Reinforcement Learning for Variable Stiffness Actuators

Final Report
von

Yuchao Yin

Lehrstuhl für
Dynamische Mensch-Roboter-Interaktion für
Automatisierungstechnik
Technische Universität München

Prof. Ph. D. Dongheui Lee

Betreuer: M.Sc. Matteo Saveriano
Beginn: 16.10.2015
Abgabe: 24.05.2016

Abstract

In this work, we introduce PILCO, a practical, data-efficient model-based policy search method. PILCO employs a probabilistic non-parametric Gaussian Process dynamics model and incorporate the model uncertainty into long-term planning. PILCO learns good controller with random initializations and expert knowledge of the system is not required. We implement PILCO into three different tasks, the position control, the hammering task and the jumping task respectively.

Contents

Chapter 1

Introduction and Related Work

In the recent years reinforcement learning (RL) becomes more popular in the research of robotic area. The traditional reinforcement learning algorithm however has a main disadvantage. The required number of interactions with the real environment is impractically high. It may require tens if not hundreds or thousands of trails to learn a particular task. That will cause the data inefficiency.
In order to increase the data efficiency we usually apply the model-based methods, which require the informative prior knowledge of the dynamic systems. Unfortunately the model-based methods can suffer severely from model errors, because they assume that the learned model resembles the real environment sufficiently accurately.
In this report, we assume that the expert knowledge is not available. That means we have no realistic simulators or explicit differential equations for the dynamics. If we only have a few observed data (see Figure 1.1), there could be multiple transition functions which cover these data points. If we just choose one of them to describe our model, the model errors could be a big problem for the long-term predictions. By contrast, a probabilistic model places a posterior distribution over the transition functions and expresses the uncertainty about the model itself.

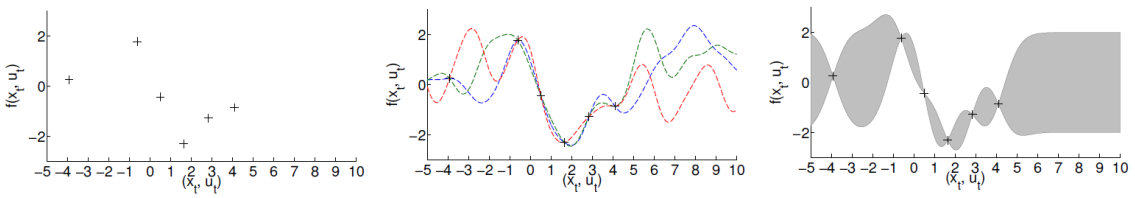


Figure 1.1: Effect of model errors

The authors of [1] propose PILCO algorithm and in this report we implement this algorithm in three different tasks. PILCO (probabilistic inference for learning control)

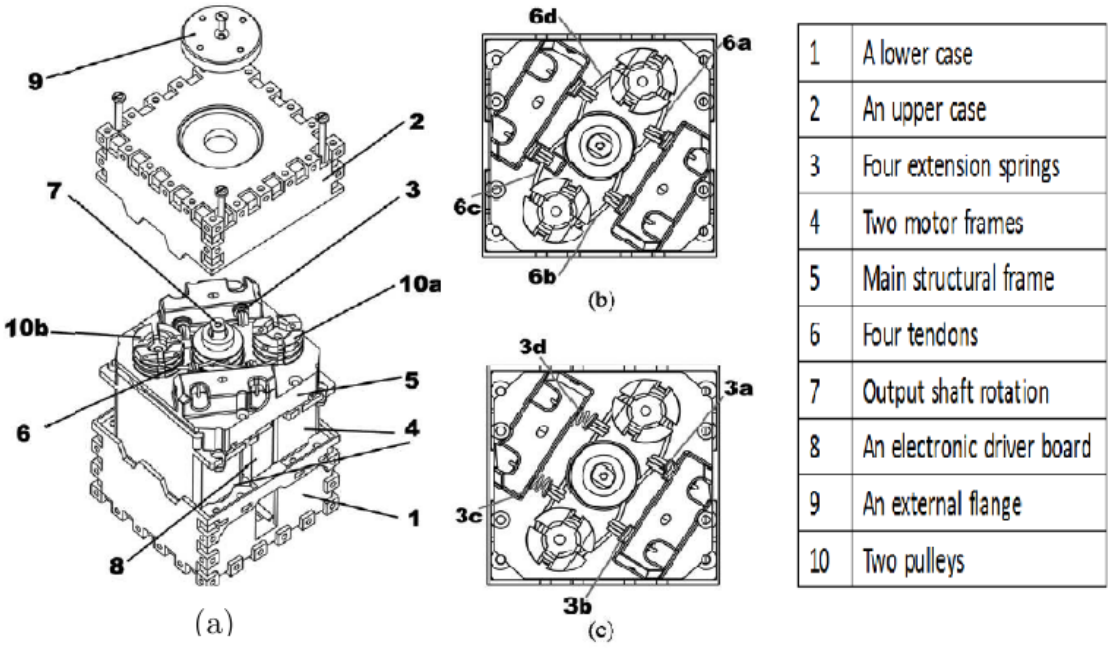
uses a probabilistic dynamics model, which is namely Gaussian processes (GPs), to overcome the model-bias problem. PILCO employs GPs to express the model uncertainty and incorporate the uncertainty into planning and policy evaluation. PILCO can also gain unprecedented data efficiency and could be directly applied to the robots.
For implementation of PILCO algorithm, we use VSA-Cube, a picture of which is shown in Figure 1.2. It is a smaller and lightweight actuator, unique for its ability to display Natural Motion. A VSA-Cube can present either a soft of rigid behavior as wished. This makes it suitable for safer motion and also possible for optimizing energy or speed performance. The usual tasks using VSA-Cube are hammering, grasping or throwing an object. These performances can be achieved by controlling not only the VSA-Cube position but also the stiffness of the VSA-Cube.

Figure 1.2: VSA-Cube

Using GP dynamics models is a common approach for learning robot dynamics [2] [3] [4]. These methods require the training data, which obtained either by motor babbling [2] or by demonstrations [4]. However, motor babbling is data-inefficient and demonstrations would contradict fully autonomous learning.
[5] [6] present algorithms with GP dynamics models in RL. Unfortunately, in [5] [6] value function models have to be maintained and they cannot directly deal with constraints in the state space.
For the purpose of data-efficient learning, [7] [8] present model-based RL methods. Shortcomings of these approaches are model-bias. In order to overcome these problems, PILCO learns a probabilistic dynamics model and incorporates model uncertainty into planning. PILCO neither requires sampling methods for planning, nor needs to maintain value function model.

Chapter 2

VSA-Cube Working Principle



1	A lower case
2	An upper case
3	Four extension springs
4	Two motor frames
5	Main structural frame
6	Four tendons
7	Output shaft rotation
8	An electronic driver board
9	An external flange
10	Two pulleys

Figure 2.1: Exploded 3D view of VSA-Cube with basic components high lighted (a). Two configurations with different values of stiffness: minimum stiffness (b) and maximum stiffness (c) with no external loads.

VSA-Cube is a variable stiffness servo actuator. It remains the features of a servo motor and moreover gives the possibility of adjusting the output shaft stiffness.
A VSA-Cube can be seen as a system with non-linear transmission that transforms input torques and velocities of its prime movers into a set of four new variables relative to the output shaft: torque, velocity, stiffness and stiffness velocity.
Figure 2.1 adapted from [9] shows the exploded 3D view of VSA-Cube with basic components and the table which listed the names of the corresponding components.

Figure 2.1(b) represents a low stiffness configuration, where springs and tendons are not loaded. In contrast, if the two pulleys rotate in opposite directions, two of the four tendons get loaded, stretching their springs and realizing a configuration of high stiffness as showed in Figure 2.1(c). Movement of the output shaft is realized by rotating the pulleys in the same direction. From the electrical point of view, the system is actuated by two digital servos and the position of the output shaft is monitored by one potentiometer [9].

Chapter 3

PILCO algorithm

We consider dynamical systems $x_t = f(x_{t-1}, u_{t-1})$ with continuous-valued states $x \in R^D$ and controls $u \in R^F$ and unknown transition function f .The policy search objective is to find a controller $\pi : x \to \pi(x) = u$ that 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.

Dynamics model learning

PILCO's probabilistic dynamics model is implemented as a GP, where we use tuples $(x_{t-1}, u_{t-1}) \in R^{D+F}$ as training inputs and differences $\Delta_t = x_t - x_{t-1} + \epsilon \in R^D, \epsilon \sim \mathcal{N}(0, \Sigma_\epsilon), \Sigma_\epsilon = \text{diag}([\sigma_{\epsilon_1}, ..., \sigma_{\epsilon_D}])$ as training targets. The covariance function is the squared exponential function which defined as

$$k(x, x') = \alpha^2 exp(-\frac{1}{2}(x - x')^T \wedge^{-1} (x - x')) \tag{3.2}$$

with $x := [x^T u^T]^T$. We define α^2 as variance of the latent function f and $\wedge := diag([l_1^2, ..., l_D^2])$.
Given n training inputs $X = [x_1, ..., x_n]$ and corresponding training targets $y = [\Delta_1, ..., \Delta_n]^T$, the GP hyper-parameters are learned by evidence maximization [10]. The GP yields one-step prediction, the predicted successor state x_{t+1} is Gaussian distributed

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(x_{t+1}|\mu_{t+1}, \Sigma_{t+1}), \tag{3.3}$$

$$\mu_{t+1} = x_t + E(\Delta_t), \Sigma_{t+1} = var_f[\Delta_t], \tag{3.4}$$

where the mean and variance of the GP prediction are

$$E_f[\Delta_t] = m_f(x_t) = k_\star^T (K + \sigma_w^2 I)^{-1} y = k_\star^T \beta, \tag{3.5}$$

$$var_f[\Delta_t] = k_{\star\star} - k_\star^T (K + \sigma_w^2 I)^{-1} k_\star, \tag{3.6}$$

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

Policy evaluation

PILCO uses the learned GP model to compute approximate long-term predictions $p(x_1|\pi), ..., p(x_T, |\pi)$ for a given controller π. Once the long-term predictions $p(x_1|\pi), ..., p(x_T, |\pi)$ are computed, the expected long-term cost

$$J^\pi(\theta) = \Sigma_{t=0}^T E(x_t)[c(x_t)], p(x_0) = \mathcal{N}(\mu_0, \Sigma_0) \tag{3.7}$$

can be computed analytically for many cost functions. In our implementations we generally use a quadratic cost function

$$c(x) = (x - x_{target})^T W (x - x_{target}), \tag{3.8}$$

where x_{target} is the target state and W is the weight matrix.

Policy improvement

We use gradient-based Quasi-Newton optimization methods, such as BFGS to improve the policy. The required gradients of the expected long-term cost J^π with respect to the policy parameters are computed analytically. The variance in the gradient estimate grows quickly with the number of parameters [11]. Further mathematically details are omitted here and refer to [12] for more information.
After policy improvement we do the roll-outs with the optimized policy π and record the new data for next iteration until the task is learned.
The general steps of PILCO is showed as follows:

Algorithm PILCO

1. **init:** Initialize random controller parameters.
   Apply random policy to simulation model and record data.
2. **REPEAT**
3. Learn GP dynamics model, using all data.
4. Model-based policy search.
5. **REPEAT**
6. Approximate inference for policy evaluation.
7. Gradient-based policy improvement.
8. Update controller parameters.
9. **UNTIL** convergence; **RETURN** parameters.
10. **SET** new policy.
11. Apply new policy to simuliation model and record data.
12. **UNTIL** task learned

Chapter 4

Experimental Results

In this chapter we implement PILCO algorithm to two different tasks. We present the experimental set-ups and results in the following.

4.1 Position Task

Set-up

Our task is to find a controller to change the input position, hereby the output position could achieve the desired position without oscillations. In our experimental the target position is 20 degree.
The code was written using Matlab and for simulation the VSA-Cube block was used, which is available at http://www.naturalmachinemotioninitiative.com/.
We created a simulation model in Matlab/Simulink to implement the code and to get some data back. Figure 4.1 shows the applied model, where the output position is connected with the desired position through a deterministic policy function. The stiffness is set to a constant value 50.

Results

We try to implement PILCO into the VSA-CubeBot [9], which serves as joints of our robot. VSA-CubeBot has mainly two input respectively the input position and the stiffness. We can change our input position to achieve the corresponding position with different stiffness. The stiffness of a VSA-CubeBot is adjustable in order to achieve different performance.
With a certain desired position and a determined stiffness, the output position as showed in Figure 4.2(b) has oscillations what cannot be ignored. For a safer performance we assume a policy model structure with the output position as model input and the desired position as control signal. The objective is to learn a controller to achieve the desired position without oscillations.

After ten trials the task is learned successfully. As showed in Figure 4.3(b), the output position achieved 20 degree smoothly without oscillations.

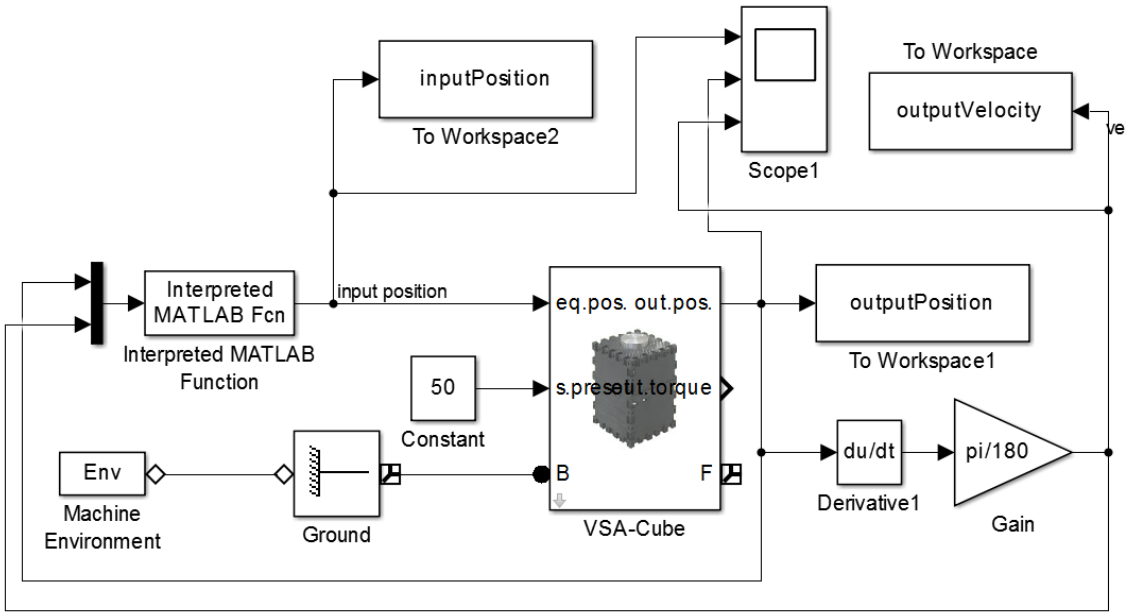
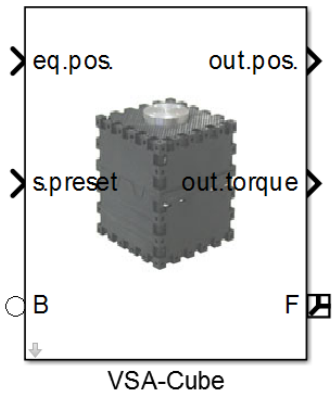
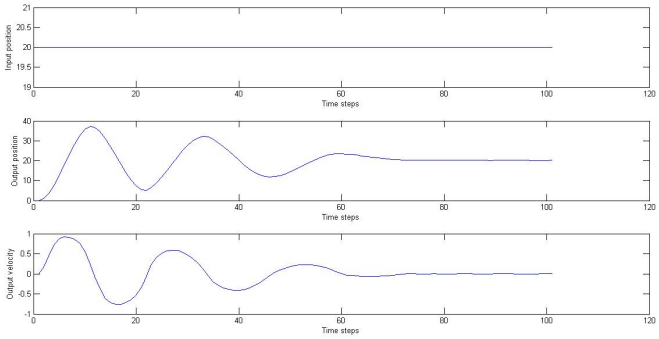


Figure 4.1: Simulation model of Position Task

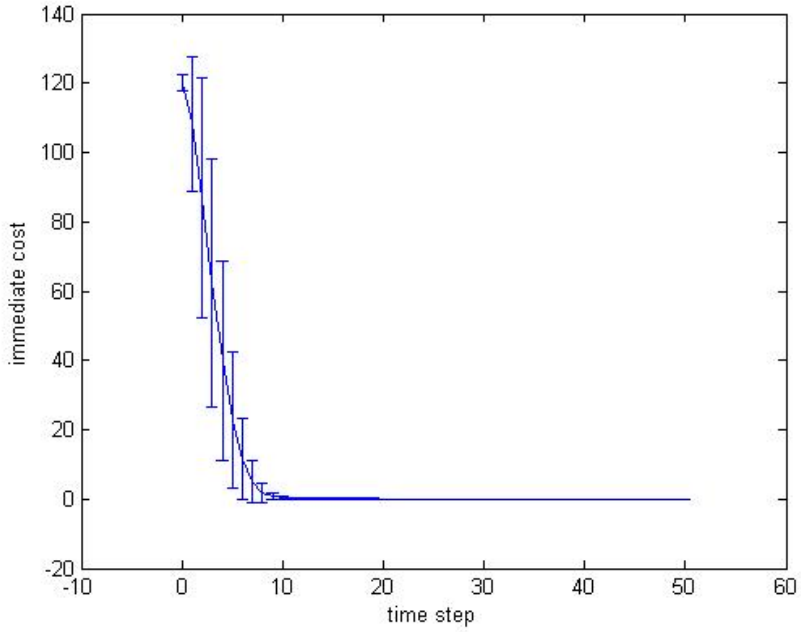


(a) VSA-Cube Block

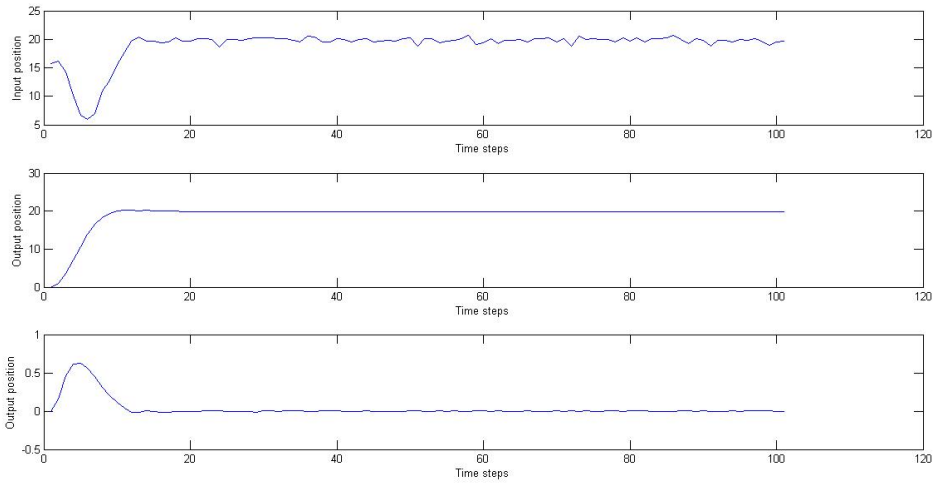


(b) The figures of one experiment with input position 20 degree and stiffness 25. The first figure is the constant input position. The second figure shows the output position with oscillations. The last position is the corresponding velocity.

Figure 4.2: Problem Description of Position Task



(a) Immediate Cost



(b) The first figure is the learned input position. The second figure shows the output position without oscillations and finally achieved desired position 20 degree. The last position is the corresponding velocity.

Figure 4.3: Experiment Results of Position Task

4.2 Hammering Task

Set-up

In this experiment we try to use VSA-Cube for the hammering task. The authors of [13] study optimal control of VSA and show that varying the spring stiffness during the execution of the hammering task improves the final performance substantially. The difference is that the PILCO algorithm is not an optimal control method. What we want to study in this work is to see if the PILCO algorithm can lead us to the success of the hammering task. For simplicity we use three constant stiffness values 5, 25 and 50. For each stiffness value we implement the PILCO algorithm and finally we compare these three performances.
The state consists of position and velocity. The target state is $[90, 9.5]$, which means it should reach 90 degree with the maximum velocity 9.5 rad/s. We only control the position as control input and we use linear policy for this situation. Figure 4.4 shows the simulation model of the hammering task. In the model we add a saturation block to the output position, which saturates the position between -90 degree to 90 degree.

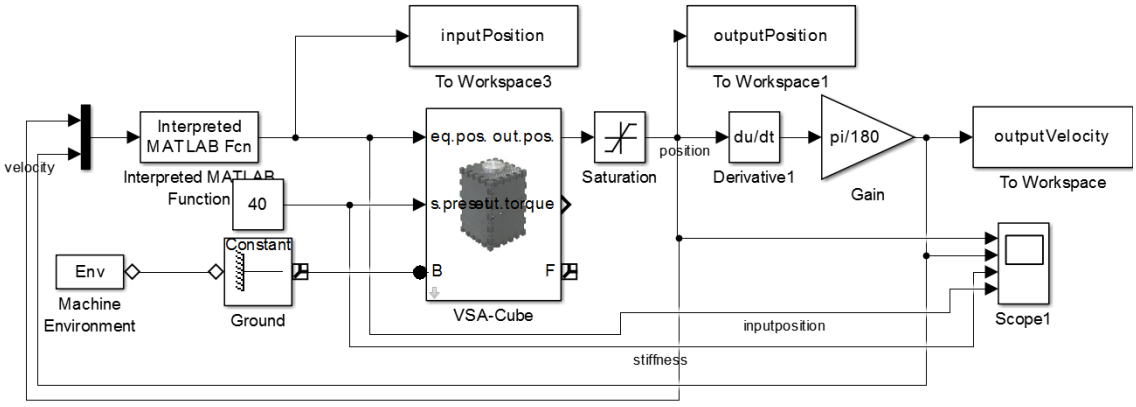


Figure 4.4: Simulation model of Hammering Task

Results

We only need circa 5 iterations to reach the target state for each constant stiffness value. The following figures show the results of three different stiffness. With stiffness 5 and stiffness 25, the target state can be reached at the end. The only difference is that with stiffness 25 it can reach the target state a little bit earlier and therefore the performance is better. With stiffness 50 the target state can never be reached as we expected. Here Figure 4.7 shows the result of one of the iterations.

In this figure we can see the output velocity when it reaches the 90 degree is getting bigger and bigger. We can deduce that at sometime the target state can be reached but this whole performance is not suitable for the hammering task. One point should be mentioned is that the immediate cost for the hammering task can not be converged because the whole performance is a dynamic action. In other words the robot can not stay in the target state. So even the PILCO algorithm is suitable for the stiffness 5 and 25, we can not confirm that the PILCO algorithm is suitable for all the situations in this experiment.

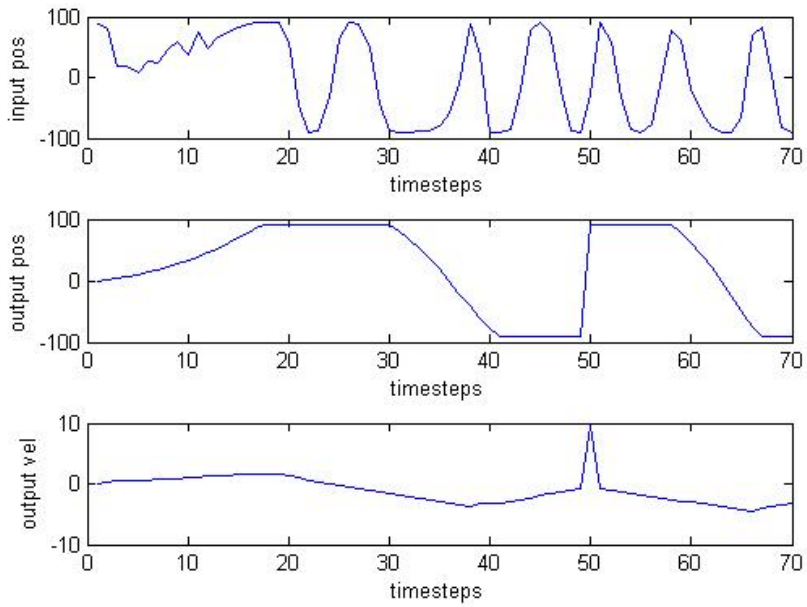


Figure 4.5: The result of Hammering Task with stiffness 5

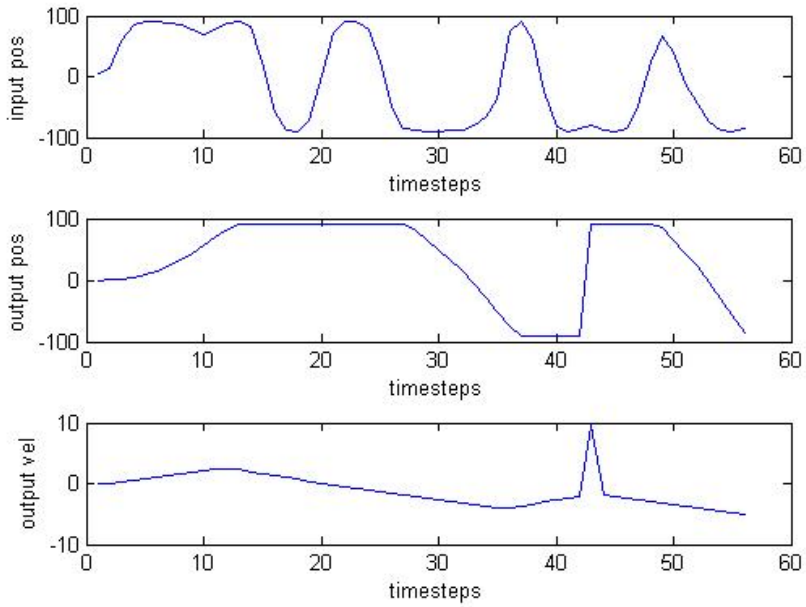


Figure 4.6: The result of Hammering Task with stiffness 25

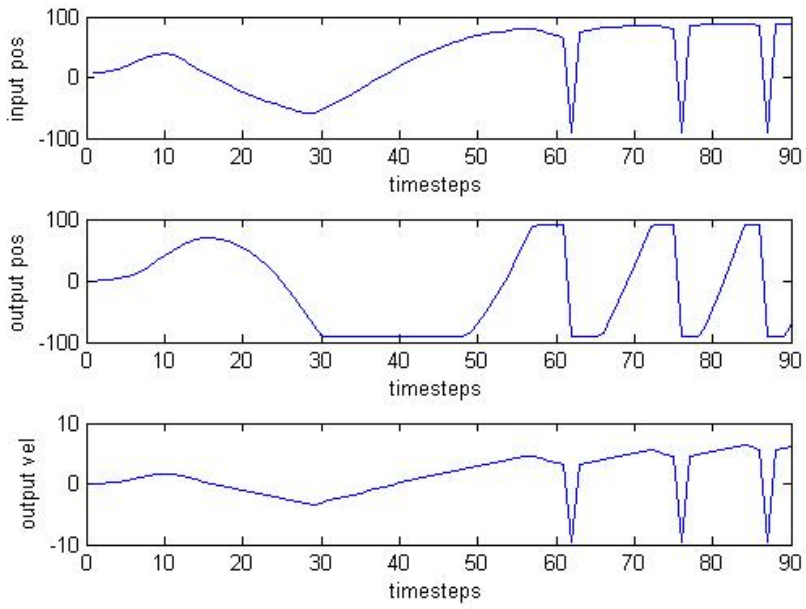


Figure 4.7: The result of Hammering Task with stiffness 50

4.3 Jumping Task

Set-up

For the jumping task we have a new simulation model which can also visualize the leg action of the robot meanwhile. The simulation model set-up is showed in Figure 4.8.

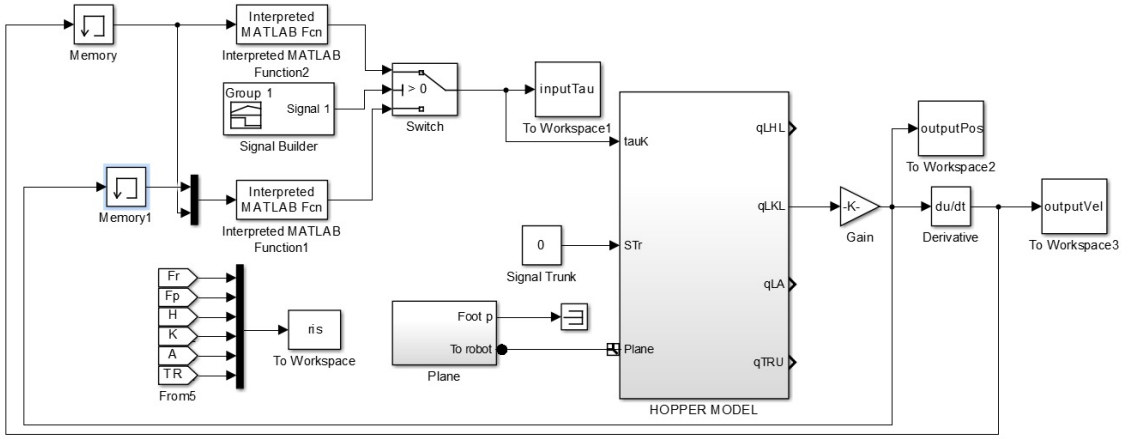


Figure 4.8: Simulation Model for Jumping Task

We control only the motor which serves as a knee-joint. The range of the control input limits the biggest angular velocity that the motor can reach and also it determines the biggest value of the force F. Figure 4.9 shows the force analysis of the robot. If we decompose it into two forces, one is perpendicular to the ground (f1) and another is parallel to the ground (f2), obviously we want to maximize f1 because f1 decides if the robot can jump or not. After analysis, the ideal case is that the motor gets the biggest velocity when it reaches the deepest position. We assume that the deepest position is -120 deg.

We divide the whole action phase into two parts and use different policy respectively. In the first phase we try to find a policy which lets the robot reach the deepest position as soon as possible. This phase we call it preparation phase. The robot saves energy in this phase. Figure 4.8 shows the simulation model of the jumping task. When the robot reaches the deepest position we switch the controller to the second policy. In the second policy our objective is let the robot stretch his leg with the biggest velocity and finally reach 0 deg smoothly for safe performance.

In the first phase we only consider the velocity as input. In the second phase we consider not only the velocity but also the position. We assume that the range of the control input is from -200 to 200. We use linear policy and a quadratic function as before.

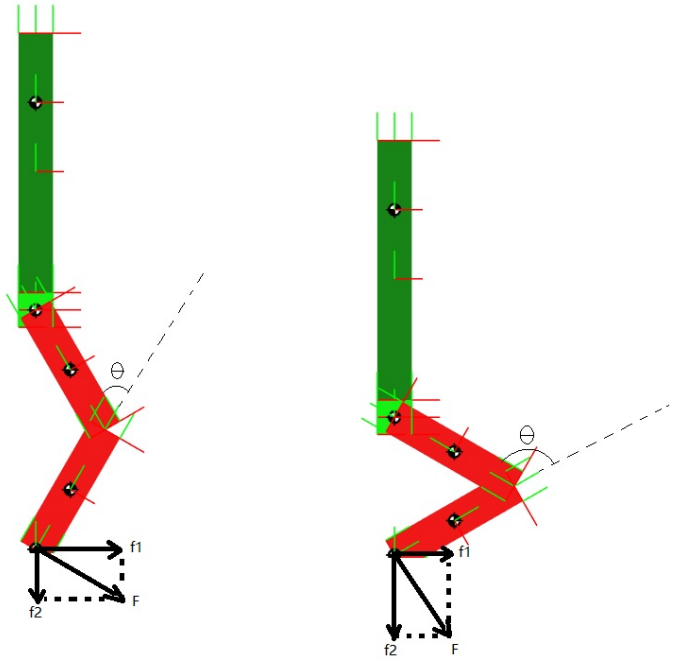


Figure 4.9: Force Analysis

Results

It needs approximately under 10 rollouts to learn the policies. Figure 4.10 provides the results. The initial state is defined as $x = [p, v]' = [-60, 0]'$. The robot is jumping at the point where we switch the first policy to the second policy. As expected the robot will finally reach 0 deg smoothly. Figure 4.11 shows the immediate cost for the second policy.
The visualization of the simulation shows that the robot bends his leg first and jumps when it reachs the deepest position, finally it straightens his leg slowly in the air.

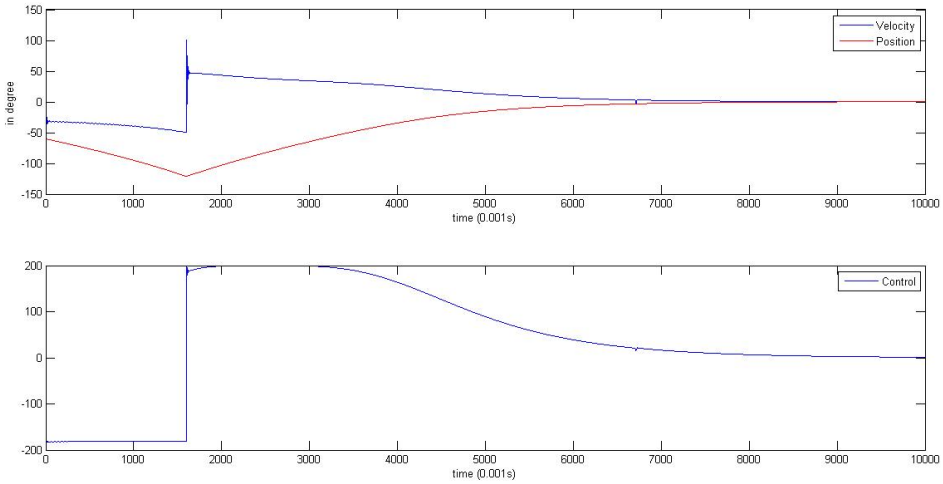


Figure 4.10: The first figure shows the output position and the corresponding velocity. The second figure is the control input.

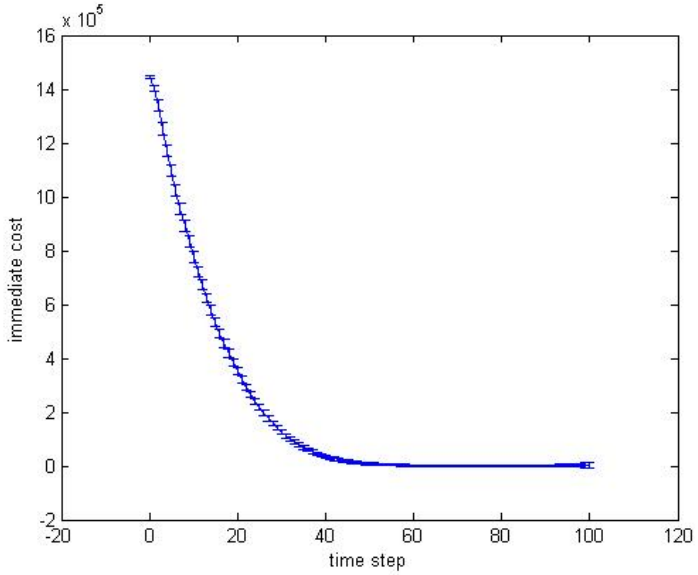


Figure 4.11: Immediate Cost for the second policy

Chapter 5

Discussion

In this report we introduced PILCO algorithm, a practical model-based policy search method. However it is not an optimal control method but only find a solution for the task. In some cases different control trajectories may have the same consequences , i.e.the target state can be reached successfully. That's why we exit the iteration as long as the task is learned. What we can guarantee is PILCO only finds the locally optimum because the optimization problem for learning the policy parameters is not convex.
The learned dynamics model is also crucial to the policy search. The GP dynamics model is only confident in the areas of the state space previously observed. When the initial policy makes the first state trajectory far from the target state, the number of total rollouts could be increased.
Since for the policy improvement the objective is to minimize the sum of the cost of each step, whether the learned task could stay in the target state or not plays a significant roll. For the position task or the pendulum task, the state trajectory could keep with the target state at the end, that leads to the lowest total cost. But for the hammering task or the jumping task, our target is let the motor reach some certain position with the biggest velocity. This state could be only reached one time and could be never kept. If we apply PILCO algorithm to these tasks, we can never guarantee that the target state can be reached. That's also the reason why we split the whole jumping task into two phases and use two different policies respectively.
One special aspect of the jumping task we should mention is that whether the motor can reach the biggest velocity or not depends on the previous state trajectory, i.e.in order to get the biggest velocity, the previous state trajectory may have several oscillations and the magnitude of the oscillations may already let the robot jump what is not expected. So the biggest velocity we got in the jumping task is smaller than the general biggest motor velocity.
PILCO needs the gradients of an approximation to the expected return J^π for indirect policy search. Thus we should guarantee that PILCO obtains nonzero gradients. It depends on the state distribution $p(x_1), ..., p(x_T)$ along a predicted trajectory and the width parameter of the immediate cost. If the cost is very peaked and the dy-

namics model is very poor, PILCO could get zero gradients and stuck in a local optimum.

Chapter 6

Conclusion

We introduced PILCO algorithm, a practical model-based policy search method using analytic gradients for policy improvement. PILCO is used promisingly for reducing model-bias in model learning, long-term learning and policy learning. PILCO doesn?t need any expert knowledge, such as demonstrations or some prior information of the dynamical system. In this report, we apply PILCO to three different tasks. We realize that PILCO does not suitable for all the tasks. But after some specific set-ups, such as splitting the task and using two policies for the jumping task, PILCO can still be successfully implemented. PILCO algorithm shows us that Bayesian inference and non-parametric models for learning controllers is not only possible but also practicable.

List of Figures

Bibliography

[1] C. E. R. Marc P Deisenroth, "Pilco: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML)*, pp. 465–472, 2011.

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

[3] J. Ko and D. Fox, "Learning gp-bayesfilters via gaussian process latent variable models," in *In Proceedings of robotics: science and systems (RSS*, 2009.

[4] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, pp. 2015–2034, Nov. 2009.

[5] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16*, pp. 751–759, MIT Press, 2004.

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

[7] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, pp. 1040–1046, MIT Press, 1997.

[8] 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.

[9] M. Catalano, G. Grioli, M. Garabini, F. Bonomo, M. Mancinit, N. Tsagarakis, and A. Bicchi, "Vsa-cubebot: A modular variable stiffness platform for multiple degrees of freedom robots," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5090–5095, May 2011.

[10] C. Rasmussen and C. Williams, "Gaussian processes for machine learning," The MIT Press, 2006.

[11] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IROS*, pp. 2219–2225, 2006.

[12] 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.

[13] M. Garabini, A. Passaglia, F. Belo, P. Salaris, and A. Bicchi, "Optimality principles in variable stiffness control: The vsa hammer," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3770–3775, Sept 2011.