Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

Model Learning for Two-Wheeled Robot Self-Balance Control

Enbo Li, Haibo Feng*, Haitao Zhou, Xu Li, Yanwu Zhai, Songyuan Zhang, Yili Fu*
State Key Laboratory of Robotics and System
Harbin Institute of Technology
Harbin, Heilongjiang Province, 150001, China
Email: haibo_feng@hotmail.com, meylfu@hit.edu.cn
*Corresponding author

Abstract—Two-wheeled robots have many advantages over other mobile robots, but they are difficult to self-balance compared with other wheeled robots. Reinforcement learning (RL) is a general framework for sequential decision-making problems. So far, there are many applications of reinforcement learning to solve robot control problems, but most of them are used in simulators because of the large amount of data required. In addition, due to the reality gap, the policy learned in a simulation environment cannot be transferred directly to a real robot. Real robot data often expensive due to the potential damage to the robot. Model-based methods require far fewer robot data than model-free methods, but these methods have the problem of model bias. In this paper, we use a model-based reinforcement learning method to achieve self-balance of a two-wheeled robot. We present a model learning method that can reduce the problem of model bias. Our method combines the simulator and a few real robot data to learn a probabilistic dynamics model of the robot through an iterative way, which requires no expertise and can learn from scratch. Then the control policy is optimized based on the learned model.

Index Terms—two-wheeled robot; reinforcement learning; balance control; model learning.

I. Introduction

Mobile robots are ubiquitous in our life and have different applications in many fields. Two-wheeled robots have many advantages over other mobile robots, such as high maneuverability, but they are difficult to control compared with self-stable robots. Most of the existing robot controllers are designed by engineers or experts [1]. This process usually requires the use of Euler–Lagrange equation, Newton's law of motion or Kane method to obtain the nonlinear dynamic model of two-wheeled robots and is usually accompanied by assumptions such as ideal rigid body and ignore friction. In addition, the model is usually linearized to simplify the model analysis process and facilitate the controller design. These artificial constraints make it difficult to obtain highly adaptable controllers.

Reinforcement learning (RL) [2] is a general framework that enables a robot to autonomously search for an optimal policy by trial-and-error with its environment continuously from scratch without much human intervention. Recent years, research in the field of RL has made great progress in robot control, but most of them are implemented in simulation environments [3], [4]. This is because hardware experimental data is valuable and RL methods usually need millions of data volumes. To make matters worse, due to the reality gap, policies learned in simulators often have a poor performance on real robots. Reality gap is caused by the difference between the real robot and the simulation model, such as physical model parameters, actuator models and latency. Though, there are some demonstrations of robots learning to grasp directly on real robots [5], [6], it's challenging to apply this method to two-wheeled robots, due to the risk of robot falls and damage.

In this paper, we present a model learning method to learn the dynamics model of two-wheeled robot that combines simulation and a few real robot data. Our method uses an iterative approach to learn a probabilistic difference model between the simulation and the real robot data. Then a control policy is optimized based on the learned model. The main steps can be summarized as follows: 1) learning the simulation model and an initial policy in a simulator; 2) applying the learned policy to the real robot and collecting real robot data; 3) learning the probabilistic difference model between the simulation model and the real-world data then learning an optimal policy based on current real robot model; 4) repeat steps 2-3 until we get a policy with good performance. The framework of our method is shown in Fig. 1.

II. Related Work

In the past few decades, there has been a lot of research on the self-balance of two-wheeled robots. Kim et al. [7] derived the nonlinear model of the two-wheeled robot using Euler–Lagrange equation. Ha et al. [8] modeled only three states of the robot. These methods usually assume some ideal conditions that make the model inaccurate. Another approach to model dynamic systems is system identification, which needs a large data set to fit model parameters. Alarfaj et al. [9] estimated the model parameters of the discrete time state space by experimental methods. The sampling rate and order directly affect the accuracy of the model. Generally, the learned model is considered accurate enough and controllers are optimized based on these model. However, it is difficult to obtain a controller with good performance due to the problem of model bias.

Reinforcement learning is a good self-learning framework and has many applications in robot control. However, model-based RL methods also require learning a sufficiently accurate dynamics model of environment and then finding an optimal policy based on this model. Ng et al. [10] present a model-based RL method, in which a stochastic nonlinear helicopter model is obtained by model identification, a lot of work has been done on model accuracy and an autonomous inverted controller is designed according to the learned model. PILCO [11] reduces model bias by learning a probabilistic dynamics model. Instead of learning a single model, PILCO uses Gaussian process to rank all possible models. Ha et al. [12] also used Gaussian Process to learn a difference model between Lagrangian model and hardware model. Learning probabilistic robot dynamics system model can effectively reduce the problem of model bias.

Another commonly used reinforcement learning method is the model-free method. Model-free methods don't need to learn the dynamics model, but it is a huge challenge to apply this sort of methods to complex robots due to the problem of data-inefficiency. These methods improve the control policy by a large number of direct trial and error. In practice, constraint conditions are added to the real system to limit the policy search space [13] or to provide demonstrations [14].

Besides, combining simulators with real robot data seems like a promising approach. A policy can be pre-trained in simulation and then fine-tuned on robots [15], also the policies learned in the simulation environment can be transferred directly to the real environment, and the reality gap is reduced by system identification, adding noise, applying stochastic rigid body model parameters [16] and other methods [17]. Zhang et al. [18] proposed a method to learn visual-servo policies by domain transfer and combining real and simulation image data. More examples of applying reinforcement learning to real robots can be found in the survey [19].

III. Model Learning and Policy Search

RL methods typically require large amounts of data from the robots' interaction with the environment to learn complex skills [16]. We aim to combine the simulation data with a small amount of real robot data to learn a real robot model through an iterative way. Fig. 1 illustrates our method. First we need to use GP to learn the simulation model, and then learn an initial policy from the simulation model, as shown in step ①. Next the policy is applied to the real robot, in this way we will collect a few real robot data to learn a difference model. From the framework we can see that real robot model is the combination of the simulation model and the difference model. Step ② is a process of applying the policy learned from the real robot model to the real robot. In order to improve the accuracy

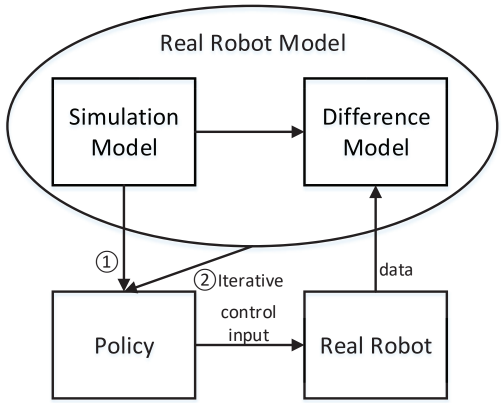

Fig. 1: Framework of our method

of the real robot model, we adopt an iterative method to constantly optimize the model. In the following, we will describe the main steps of the algorithm in detail: dynamics model formulation, difference model learning and policy search methods. Algrithom 1 summarizes our framework.

Algorithm 1 Model learning and policy search

```
Initialize random policy $p$ , $i = 0$ $f = 0$ $g = 0$ and $D_g = \emptyset$
1: $D_f \leftarrow$ apply $p$ to the simulation environment
2: $f \leftarrow D_f$ to learn a simulation model
3: $p \leftarrow$ policy optimized for $f$
4: while $i < N$ do
5:         $D_i \leftarrow$ apply $p$ to the real robot
6:         $D_g \leftarrow D_g \cup D_i$
7:         $g \leftarrow D_g$ learn a difference model
8:         $p \leftarrow$ policy optimized for $f + g$
9:         $i \leftarrow i + 1$
10: end while
```

A. Gaussian Process

Gaussian process (GP) provides a probability framework for functions sorting according to their plausibility by defining the probability distribution of functions [20].In standard GP regression [21], we assume that we have a training set $D:=\left\{\mathbf{X} := [\mathbf{x}_1, \cdots, \mathbf{x}_n]^T, \mathbf{y} := [y_1, \cdots, y_n]^T\right\}$, GP maps inputs $\mathbf{x} \in \Re^D$ to a scalar output $y \in \Re$, the equation can be expressed as $y_i = h(\mathbf{x}_i) + \varepsilon_i$, where $h : \Re^D \to \Re$ and $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise. In this paper, we will use a prior mean function $m_h \equiv 0$, and the squared exponential kernel (SE)

$$k_h(\mathbf{x}, \mathbf{x}') := \alpha^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Lambda^{-1} (\mathbf{x} - \mathbf{x}')\right) + \delta_{\mathbf{x},\mathbf{x}'}\sigma^2, \qquad \mathbf{x}, \mathbf{x}' \in \Re^D \quad (1)$$

Where α^2 is the signal variance of the latent function h, $\Lambda = diag\left(\left[\ell_1^2, \cdots, \ell_D^2\right]\right)$ is a positive-definite diagonal matrix with elements ℓ^i, $i = 1, \cdots, D$ and

$\delta_{\mathbf{x},\mathbf{x}'}$ is unity when $\mathbf{x} = \mathbf{x}'$ and zero otherwise. The set $\theta = \left(\ell_1, \cdots, \ell_D, \alpha^2, \sigma^2\right)$ is so-called hyper-parameters. Then the Gaussian distribution of any finite set of function values $h(\mathbf{X}) := [h(\mathbf{x}_1), \cdots, h(\mathbf{x}_n)]$ can be expressed as $h \sim \mathcal{GP}(m_h, k_h)$, GP hyper-parameters are learned by evidence maximization. Given a new input $\mathbf{x}^*$ we can predict the output

$$y^* = k_*^T K^{-1} y \tag{2}$$

and variance

$$\sigma^2 = k(\mathbf{x}^*, \mathbf{x}^*) - k_*^T K^{-1} k_* \tag{3}$$

where $K = \{k(\mathbf{x}_i, \mathbf{x}_j)\} \in \Re^{D\times D}$ and $k_* = \{k(\mathbf{x}^*, \mathbf{x}_i)\} \in \Re^D$. In this paper, we will use the GP to build up a simulation model and a difference model, the details will be described in the following sections.

B. Model Learning

The model learning process can be divided into two steps. First we need to learn the simulation model f, and the difference model g according to the simulation model, then combine the simulation model and difference model to obtain the real robot model. The model f and g are learned through GP respectively.

1) Learning simulation model: The dynamic system of the simulation model can be written as

$$\mathbf{x_t} = \mathbf{x_{t-1}} + f(\mathbf{x_{t-1}}, \mathbf{u_{t-1}}) \tag{4}$$

where $\mathbf{x_t} \in \Re^D$ is the robot state at time step t, $\mathbf{u} \in \Re^E$ is the control signal and $f : \Re^D \times \Re^E \to \Re^D$ is the simulation model. Since the model is learned in a simulation environment, we have enough data to learn a model with small variance. We use tuples $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \in \Re^{D+E}$ as training inputs and $\Delta_t = \mathbf{x}_t - \mathbf{x}_{t-1} \in \Re^D$ as training targets. Function f is implemented as a GP. The GP produces one-step predictions

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mu_{ft}, \mathbf{\Sigma}_{ft}) \tag{5}$$

$$\mu_{ft} = \mathbf{x}_{t-1} + E_f[\Delta_t] \tag{6}$$

where μ_{ft} is the mean and $\mathbf{\Sigma}_{ft}$ is the covariance matrix of the simulation model f at time step t. Since the variance of the prediction is small, the variance will be ignored in the following applications.

2) Learning difference model : Once we have the simulation model which is an approximate dynamics model of the real robot, we can use it to find an initial control policy using a model-based reinforcement learning algorithm and apply it to the real robot. Because of the model bias problem, the policy may have a poor performance on it, but we can collect real-world data through this process. Our idea is to use these data to learn the difference model g between f and the real-world data. The use of simulation model f effectively reduces the amount of real-world data used. The difference model g is represented by GP model. Combining simulation model and difference model, we get one-step prediction of the real robot model as

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mu_{gt} + \mu_{ft}, \mathbf{\Sigma}_{gt}) \tag{7}$$

where μ_{gt} is the mean and $\mathbf{\Sigma}_{gt}$ is the covariance matrix of the diffence model g in step t. The real robot dynamic model can be expressed as

$$\mathbf{x_t} = \mathbf{x_{t-1}} + f(\mathbf{x_{t-1}}, \mathbf{u_{t-1}}) + g(\mathbf{x_{t-1}}, \mathbf{u_{t-1}}) \tag{8}$$

where $g : \Re^D \times \Re^E \to \Re^D$. In order to get a more accurate model, we repeat the process of learning the difference model, learning policy and applying policy as shown in Fig. 1.

C. Policy Learning

Finding an optimal policy is an optimization problem that needs to find optimal policy parameters ψ^* by minimizing the expected long-term costs as shown in equation (9).

$$J^\pi(\psi) = \sum\nolimits_{t=0}^{T} E_{\mathrm{x}_t}[c(\mathrm{x}_t)], \mathrm{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{9}$$

Equation (9) samples several paths τ_i starting from different initial state distributions $p\left(\mathbf{x}_0^{(i)}\right)$. where $c(\mathrm{x}_t)$ is the cost function, specifying different cost functions for RL can generate controllers for different tasks of interest. $\pi_\psi(\mathrm{x}) = \mathbf{u}$ is the control signal for T steps. In our work we use RBF network $\tilde{\pi}(\mathbf{x})$ which is a nonlinear model to represent the parameterized policy model and a squashing function σ to ensure that the control signals $\mathbf{u} = \pi(\mathbf{x})$ do not go beyond the interval $[-\mathbf{u}_{\max}, +\mathbf{u}_{\max}]$. Then the nonlinear policy has the form $\mathbf{u} = \pi(\mathbf{x}) = \mathbf{u}_{\max}\sigma(\tilde{\pi}(\mathbf{x}))$, where σ is a squashing function defined as

$$\sigma(\mathrm{x}) = \frac{9\sin(\mathrm{x}) + \sin(3\mathrm{x})}{8} \tag{10}$$

The cost function for policy optimization is

$$c(\mathrm{x}) = 1 - \exp\left(-\frac{1}{2a^2} d(\mathrm{x}, \mathrm{x}_{t\arg et})^2\right) \in [0, 1] \tag{11}$$

where $\mathrm{x}_{\mathrm{t\,arg\,et}}$ is the target state, $d(\mathrm{x}, \mathrm{x}_{t\arg et})$ is the Euclidean distance between the current and the target state. a is a parameter which controls the shape of the cost function as shown in Fig. 2.

The gradient of the long-term cost $J^\pi(\psi)$ can be represented as

$$\frac{dJ^\pi(\psi)}{d\psi} = \sum_{t=0}^{T} \frac{d}{d\psi} E_{\mathrm{x}_t}[c(\mathrm{x}_t)|\pi_\psi] \tag{12}$$

Many RL algorithms can be used to solve the optimization problems [22], we also take probability into account in our policy search process. PILCO is a model-based RL method that takes the uncertainty of probabilistic models into account in long-term prediction and decision-making process and it provide an analytic method for policy improvement. In this paper we use the optimization

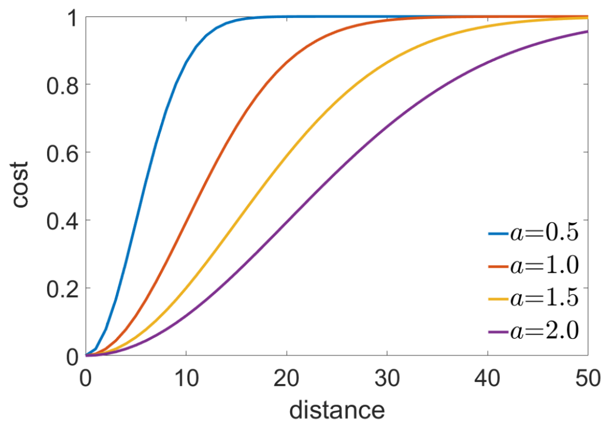

Fig. 2: Cost function.

method of PILCO. The derivative in equation (12) is given by

$$\frac{d}{d\psi}\mathrm{E}_{\mathrm{x}_t}\left[c\left(\mathrm{x}_t\right)\right]=\left(\frac{\partial}{\partial \mu_t}E_{\mathrm{x}_t}\left[c\left(\mathrm{x}_t\right)\right]\right)\frac{d\mu_t}{d\psi}+ \left(\frac{\partial}{\partial \Sigma_t}E_{\mathrm{x}_t}\left[c\left(\mathrm{x}_t\right)\right]\right)\frac{d\Sigma_t}{d\psi} \tag{13}$$

The required derivatives can be obtained recursively in the following steps

$$\frac{\partial}{\partial \mu}E_{\mathrm{x}_t}\left[c\left(\mathrm{x}_t\right)\right],\ \frac{\partial}{\partial \Sigma_t}E_{\mathrm{x}_t}\left[c\left(\mathrm{x}_t\right)\right] \tag{14}$$

The expressions in equation (14) is the partial derivative with respect to μ_t and Σ_t, where μ_t and Σ_t are the mean and covariance of the state distribution $p\left(\mathrm{x}_t\right)$, and the result depends on the expression of the cost function $c\left(\mathrm{x}_t\right)$.

$$\frac{d\mu_t}{d\psi}=\frac{\partial \mu_t}{\partial \mu_{t-1}}\frac{d\mu_{t-1}}{d\psi}+\frac{\partial \mu_t}{\partial \Sigma_{t-1}}\frac{d\Sigma_{t-1}}{d\psi}+\frac{\partial \mu_t}{\partial \psi} \tag{15}$$

$$\frac{d\Sigma_t}{d\psi}=\frac{\partial \Sigma_t}{\partial \mu_{t-1}}\frac{d\mu_{t-1}}{d\psi}+\frac{\partial \Sigma_t}{\partial \Sigma_{t-1}}\frac{d\Sigma_{t-1}}{d\psi}+\frac{\partial \Sigma_t}{\partial \psi} \tag{16}$$

For equation (15),(16) the partial derivatives $\frac{d\mu_{t-1}}{d\psi}$ and $\frac{d\Sigma_{t-1}}{d\psi}$ can be obtained through the iterative way and we can get

$$\frac{\partial \mu_t}{\partial \psi},\ \frac{\partial \Sigma_t}{\partial \psi} \tag{17}$$

through chainrule. A more detailed process can be found in paper [20].

IV. Experimental Results

In order to prove the feasibility of our method, our experiment is to learn the dynamic system model of a two-wheeled robot and then optimize a control policy based on the learned model.

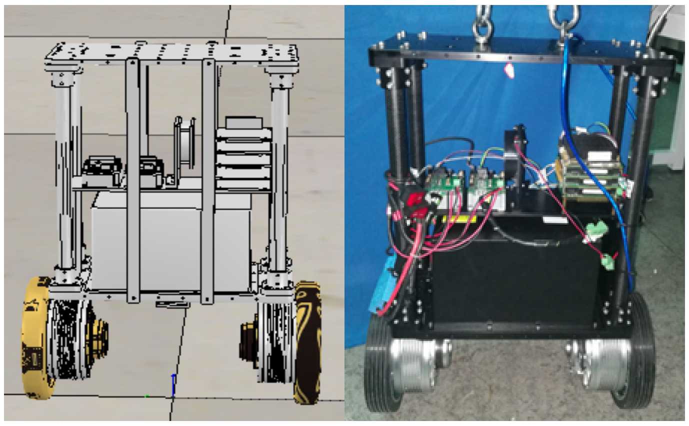

Fig. 3: Two-wheeled robot platform used in our experiment. (a) Simulation model in 3D physical simulation engine environment. (b) Real two-wheeled robot platform.

A. Experiment Platform

Fig. 3(a) is a two-wheeled robot in the 3D physical simulation engine environment, which is used to learn the simulation model. The real robot platform is shown in Fig. 3(b). Physical properties parameters of simulation environment model are set according to CAD model. The state $\mathbf{x}$ for two-wheeled robot is given as: $\mathbf{x}=\left[\theta,\ \dot{\theta},\ a\right]^T$, where θ is the platform's tilt angle in rad, $\dot{\theta}$ is the platform's angular velocity in rad/s, and a is the platform's acceleration in m/s^2. To simplify the experiment, the only control signal $\mathbf{u}=\pi\left(\mathbf{x}\right)$ is the joint torque of two-wheel motors. Therefore the dimension of robot states is $n=3$ and the number of control inputs to the model is $m=1$. So the total number of inputs to the GP models is $r=m+n=4$. We choose the initial state at $\theta=-10^\circ$ and added Gaussian noise with zero means. The platforms need to learn to recover from instability states.

GP has a good performance in model learning, but when the training data increases, there is a significant increase in computational cost. Therefore we need to switch to sparse GP approximations to remove some of the useless data.

B. Model Learning and Policy Search

As mentioned above, the parametric control policy is constructed by a RBF network $\tilde{\pi}\left(\mathbf{x}\right)$ and a squashing function $\sigma\left(\mathrm{x}\right)$. We set the maximum motor output torque to $\mathbf{u}_{\max}=20\ N.m$. The policy parameters is initialized with Gaussian noise. For the cost function in equation (11), the target state is $\mathrm{x}_{t\arg et}=\left[0,0,0\right]^{\mathrm{T}}$ and the shape parameter a=0.5 in our experiments.

In order to learn a good simulation model, we used random control signals for 20 times rollout on the simulation platform. When the maximum time step $H=100$ is reached or the platform tilt angle is greater than 30

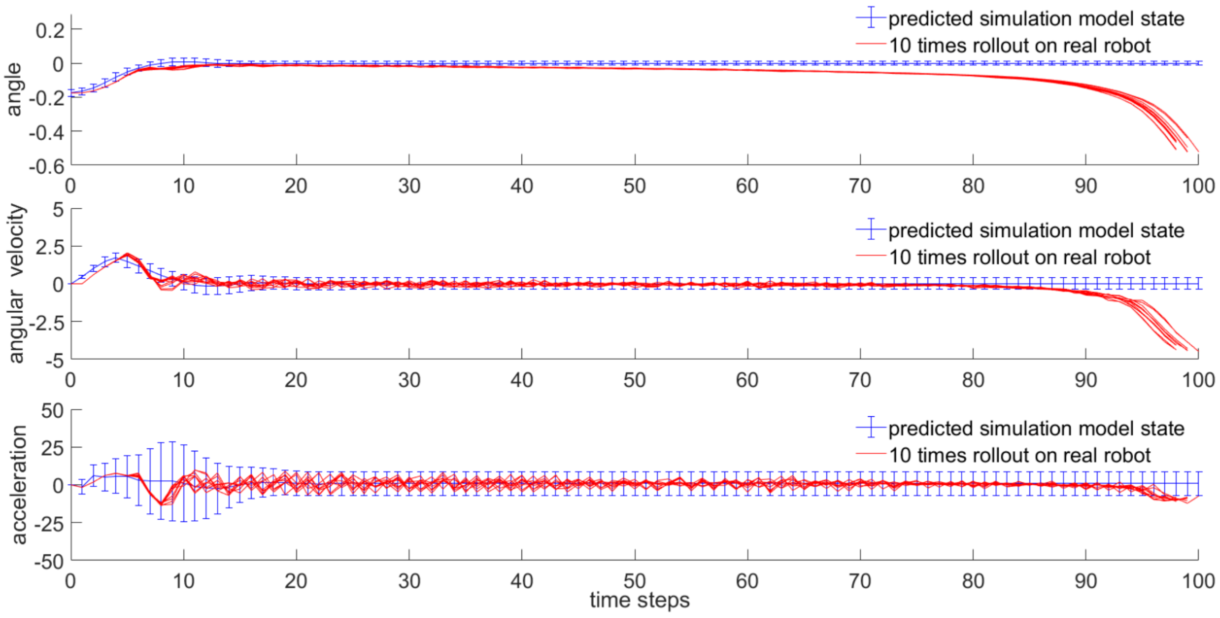

Fig. 4: Learning the simulation model and learning control policy based on the simulation model, and then directly applying the policy to the real robot platform. The blue line indicates the state distribution of each time step predicted on the simulation model according to the learned policy. The red line represents the state of 10 times the policy applied on the real robot platform.

degrees it will break in advance. So the total number of taining data set we used to learn the simulation model is less than 1000. After we have the simulation model, we can search the control policy and fit the difference model according to this model as shown in Fig. 1. Then the real platform model is obtained from simulation model and difference model.

C. Performance

We evaluate the performance of the learned policy by applying the learned policy to real robot platform. Fig. 4 shows the result of applying the policy learned from the simulation model directly to the real platform. Based on the policy we can get predicted state interval and real state trajectories. As we can see, the real state trajectory deviates from the predicted trajectory. Due to the model bias problem,the policy has poor performance on the real robot platform. After that, we modify the real robot model by adding the difference model based on the simulation model and then learn an optimal policy. Fig. 5 shows the result of applying the policy learned on the real robot platform after three times' update for the real robot model. We can see that the predicted state is highly consistent with the real robot state trajectory, which proves that the difference model plays an important role in the process of real robot model learning.

V. Conclusions

In this paper, we present a model learning method that can solve the model bias problem by combining the simulation data and real robot data to learn a probabilistic model. Moreover, this method is applied to the self-balance control of a two-wheeled robot, which proves the feasibility of the method. The key idea of our method is to learn a simulation model first and then learn the difference model iteratively based on the simulation model. For policy search process, it is necessary to take probability into account. We prove that we can find a policy to complete the task according to the model learned from a few hardware experiments. The use of simulation model can effectively reduce the amount of hardware data usage. Future work will focus on apply our method to more complex robots. In addition, combining traditional control theory with reinforcement learning method to control robot will be a promising method.

Acknowledgment

The authors would like to thank the Heilongjiang Post-doctoral Scientific Research Foundation (LBH-Q17068), the self-managed project of the State Key Laboratory of Robotics and System in Harbin Institute of Technology (SKLRS201801A01, SKLRS201801A02).

References

[1] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," Annual Reviews in Control, vol. 37, no. 1, pp. 89–103, 2013.
[2] R. S. Sutton, A. G. Barto et al., Introduction to reinforcement learning. MIT press Cambridge, 1998, vol. 135.

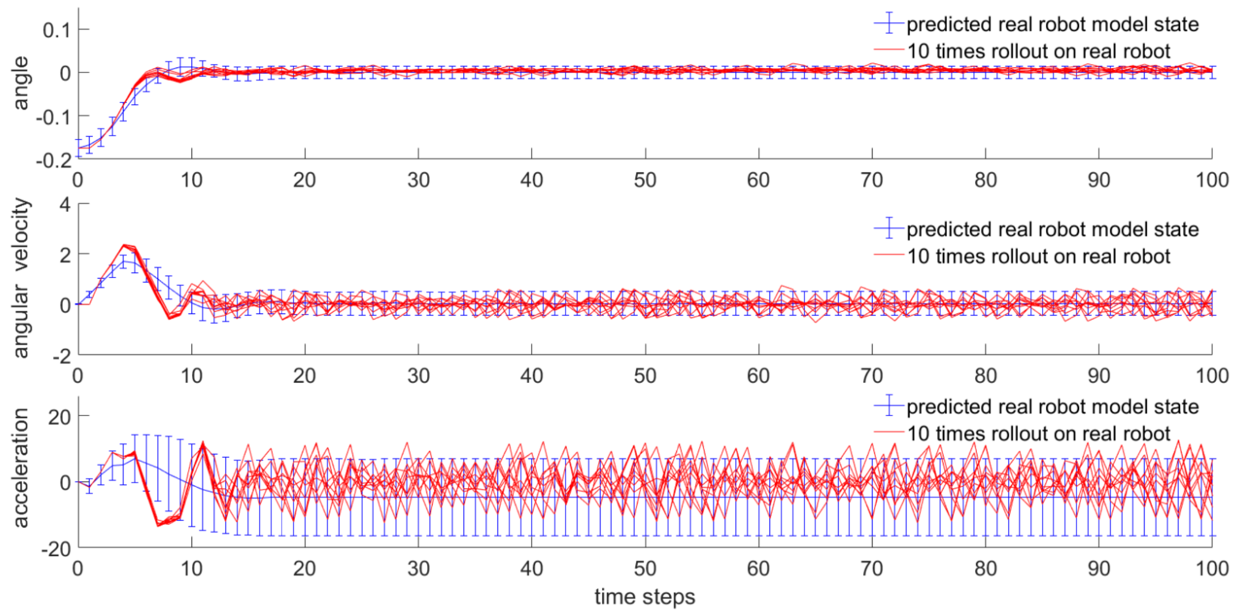

Fig. 5: Learning the real robot model (adding the difference model based on the simulation model)and then learning control policy based on the real robot model. The blue line indicates the state distribution of each time step predicted on the real robot model according to the learned policy. The red line represents the state of 10 times the policy applied on the real robot platform.

[3] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," arXiv preprint arXiv:1709.10087, 2017.

[4] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller et al., "Emergence of locomotion behaviours in rich environments," arXiv preprint arXiv:1707.02286, 2017.

[5] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 3381–3388.

[6] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," The International Journal of Robotics Research, vol. 37, no. 4-5, pp. 421–436, 2018.

[7] Y. Kim, S.-H. Lee, and D. H. Kim, "Dynamic equations of a wheeled inverted pendulum with changing its center of gravity," in 2011 11th International Conference on Control, Automation and Systems. IEEE, 2011, pp. 853–854.

[8] X. Ruan and J. Cai, "Fuzzy backstepping controllers for two-wheeled self-balancing robot," in 2009 International Asia Conference on Informatics in Control, Automation and Robotics. IEEE, 2009, pp. 166–169.

[9] M. Alarfaj and G. Kantor, "Centrifugal force compensation of a two-wheeled balancing robot," in 2010 11th International Conference on Control Automation Robotics & Vision. IEEE, 2010, pp. 2333–2338.

[10] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in Experimental Robotics IX. Springer, 2006, pp. 363–372.

[11] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in Proceedings of the 28th International Conference on machine learning (ICML-11), 2011, pp. 465–472.

[12] S. Ha and K. Yamane, "Reducing hardware experiments for model learning and policy optimization," in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 2620–2626.

[13] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, "Learning from demonstration and adaptation of biped locomotion," Robotics and autonomous systems, vol. 47, no. 2-3, pp. 79–91, 2004.

[14] G. Kniewasser, "Reinforcement learning with dynamic movement primitives-dmps," target, vol. 6, no. 8, p. 10.

[15] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," arXiv preprint arXiv:1610.04286, 2016.

[16] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," Science Robotics, vol. 4, no. 26, p. eaau5872, 2019.

[17] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," arXiv preprint arXiv:1804.10332, 2018.

[18] F. Zhang, J. Leitner, Z. Ge, M. Milford, and P. Corke, "Adversarial discriminative sim-to-real transfer of visuo-motor policies," The International Journal of Robotics Research, vol. 38, no. 10-11, pp. 1229–1245, 2019.

[19] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013.

[20] M. P. Deisenroth, Efficient reinforcement learning using Gaussian processes. KIT Scientific Publishing, 2010, vol. 9.

[21] C. Rusmassen and C. Williams, "Gaussian process for machine learning," 2005.

[22] K. P. Körding and D. M. Wolpert, "Bayesian decision theory in sensorimotor control," Trends in cognitive sciences, vol. 10, no. 7, pp. 319–326, 2006.