Reducing Hardware Experiments for Model Learning and Policy Optimization

Sehoon Ha[1] and Katsu Yamane[2]

Abstract— **Conducting hardware experiment is often expensive in various aspects such as potential damage to the robot and the number of people required to operate the robot safely. Computer simulation is used in place of hardware in such cases, but it suffers from so-called simulation bias in which policies tuned in simulation do not work on hardware due to differences in the two systems. Model-free methods such as Q-Learning, on the other hand, do not require a model and therefore can avoid this issue. However, these methods typically require a large number of experiments, which may not be realistic for some tasks such as humanoid robot balancing and locomotion. This paper presents an iterative approach for learning hardware models and optimizing policies with as few hardware experiments as possible. Instead of learning the model from scratch, our method learns the difference between a simulation model and hardware. We then optimize the policy based on the learned model in simulation. The iterative approach allows us to collect wider range of data for model refinement while improving the policy.**

I. Introduction

Conducting hardware experiments is a cumbersome task especially with large, complex and unstable robots such as full-size humanoid robots. They may require multiple people to operate to ensure safety of both operators and the robot; control failures can cause major damage; and even a minor damage is difficult to troubleshoot due to complexity.

For this reason, simulation is often used to replace hardware experiments. Unfortunately, it is difficult to obtain accurate simulation models, and therefore it suffers from so-called simulation bias [1] in which policies tuned in simulation cannot realize the same task with the hardware system due to differences in the two systems.

This paper presents an iterative approach for model learning and policy optimization using as few experiments as possible. Instead of learning the hardware model from scratch, our method reduces the number of experiments by only learning the difference from a simulation model. The policy is then optimized through simulations using the learned model. We repeat this process iteratively so that we can refine the model because the improved policy is more likely to realize wider range of motions.

The assumption is that three things are essential to policy learning for complex robots:

- Learning only the difference from a model is essential to reduce the number of hardware experiments. The model can also be used for optimizing the initial policy.
- Iterative process is important for inherently unstable robots because we cannot collect enough data using a policy trained only in simulation.
- The learned model should be stocastic so that it can model sensor and actuator noises.

Our target task in this paper is balancing of bipedal robot on a bongoboard. To prove the concept, and to better control the noise conditions, we shall use two simulation models instead of a simulation model and a hardware system. One of the models is derived by Lagrangian dynamics assuming perfect contact conditions, while the other model is based on a 2D physics simulation engine with a more realistic contact model. These models are different enough that a policy optimized for the former cannot stabilize the latter.

The rest of the paper is organized as follows. In Section II, we first review the related work in machine learning literature in the context of robot control. Section III gives an overview of our framework, followed by more details on the model learning in Section IV and policy optimization in Section V. Section VI presents simulation results and analysis. We finally conclude the paper in Section VII.

II. Related Work

Difference between a robot and its simulation model becomes a serious problem when we try to use controllers obtained by model-based optimization or tuned in simulation. Classical parameter identification techniques [2] partially solve this problem by fitting model parameters to experimental data, but they are still limited to factors that can actually be modeled. Furthermore, these approaches assume that the data set is large enough to accurately estimate the parameters. In large and unstable systems such as humanoid robots, it is often difficult to collect enough data [3].

Another approach is model-free policy optimization, where the policy is improved through a number of hardware trials [4], [5]. Unfortunately, these methods generally require hundreds of trials, which is unrealistic for tasks such as humanoid balancing and locomotion. One way to overcome this issue is to limit the parameter space by using task-specific primitives [6] or to provide a good initial trajectory by human demonstration [7]. However, it is not clear how to extend these approaches to dynamically unstable robots or tasks that cannot described by joint trajectories.

A number of researchers have attempted to overcome the drawbacks of these approaches by combining simulation and real-world data [8]–[10]. Abbeel et al. [11] used an inaccurate model to estimate the derivative of the cost with respect to the policy parameters. Ko et al. [12] used Gaussian Process

[1]Sehoon Ha is with Georgia Institute of Technology. He was at Disney Research, Pittsburgh when this research was conducted. [2]K. Yamane is with Disney Research, Pittsburgh. kyamane@disneyresearch.com

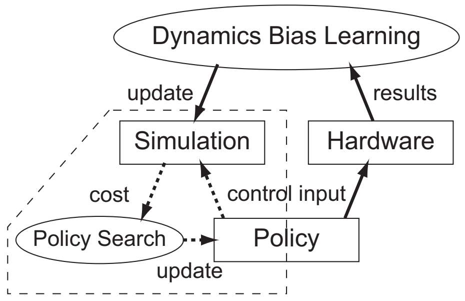

Fig. 1. Framework of our approach.

to model the difference between a nonlinear dynamics model and the actual dynamics and applied the model to reinforcement learning for yaw control of a blimp. However, they do not iterate the process to refine the model. Deisenroth et al. [13] also used Gaussian Process for learning the dynamics model from scratch. Similarly, Morimoto et al. [14] used Gaussian Process for learning simplified dynamics of human locomotion. Sugimoto et al. [15] used *sparse pseudo-input Gaussian Process* (SPGP) that accounts both variances of inputs and outputs to handle sensor noises. Instead, Tangkaratt et al. [16] used *least-squares conditional density estimation* (LSCDE) to learn the dynamics model without Gaussian assumption on the transitions. Cutler et al. [17] trained a policy in multiple fidelity simulators with discretized actions. Ross and Bagnell [18] theoretically proved that their iterative system identification method converges even the system is not in the assumed class. Please refer to Section 6 of [1] for more complete survey on this topic.

III. Overview

We developed an iterative reinforcement learning process to alternatley refine the model and policy. Figure 1 illustrates the approach.

The three main components are simulation, hardware, and policy. *Simulation* is based on a model of the robot hardware, and cheap to run. *Hardware* is the real robot and therefore more expensive to run. Both simulation model and robot hardware are controlled by control inputs computed by the *policy*.

The framework includes two iteration loops that run with different cycles. The outer loop (solid arrows) is the *dynamics bias learning* process that uses the experimental data from hardware to train the simulation model. The inner loop (dashed arrows) is the *policy search* process that uses the simulation model to optimize the policy based on a given cost function.

Our framework adapts some of the ideas used in prior work. Similarly to [12], we use Gaussian Process to model the difference between a dynamics model and actual robot dynamics. On the other hand, we also adopt the iterative learning scheme as in [11] because the performance of the initial controller is usually not good enough to learn accurate dynamics model. We also chose to directly optimize the policy parameters instead of learning the value function, as in [13].

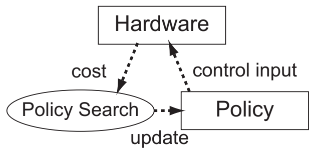

Fig. 2. Direct policy search.

We compare our framework with conventional direct policy search represented in Fig. 2. This approach only has the policy search loop that uses the hardware directly to obtain the control cost for policy search. It usually requires a large number of hardware trials, which is unrealistic for our target robots and tasks.

The goal of this work is to reduce the number of dynamics bias learning loops that involve hardware experiments. On the other hand, we can easily run many policy search loops because we only have to run simulations.

IV. Learning the Dynamics Model

A. Dynamics Bias Formulation

A general form of dynamics of a system with n states and m inputs can be written as

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{1}$$

where

$$\begin{aligned} \boldsymbol{x} \in \Re^n &\quad : \quad \text{robot state} \\ \boldsymbol{u} \in \Re^m &\quad : \quad \text{input} \\ \boldsymbol{f}\colon \Re^n \times \Re^m \rightarrow \Re^n &\quad : \quad \text{system dynamics function.} \end{aligned}$$

The goal of learning is to obtain $\boldsymbol{f}$ such that the model can accurately predict the system's behavior. In this paper, we employ one of the non-parametric models, Gaussian Process (GP) model. Learning $\boldsymbol{f}$ without prior knowledge, however, is expected to require a large amount of data to accurately model the system dynamics.

For many robots, we can obtain an approximate dynamics model by using, for example, Lagrangian dynamics. We denote such model by $\boldsymbol{f}'$. Instead of learning $\boldsymbol{f}$ that requires a large amount of data, our idea is to learn the difference between $\boldsymbol{f}'$ and the real dynamics:

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) + \boldsymbol{g}_D\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right) \tag{2}$$

where $\boldsymbol{g}_D\colon \Re^n \times \Re^m \rightarrow \Re^n$ is the difference model to be learned and D represents the set of data used for learning the model. In this paper, we call $\boldsymbol{g}_D$ as dynamics bias.

Our expectation is that $\boldsymbol{f}'$ is a good approximation of the system dynamics, and therefore learning $\boldsymbol{g}_D$ requires far smaller data set than learning $\boldsymbol{f}$ from scratch.

B. Gaussian Process

Gaussian Process (GP) [19] is a stochastic model that represents the relationship between r inputs $\tilde{\boldsymbol{x}} \in \Re^r$ and a scalar output y. For the covariance function, we use the sum of a squared exponential and noise functions:

$$k\left(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}'\right) = \alpha^2 \exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)^T \boldsymbol{\Lambda}^{-1}\left(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}'\right)\right) + \delta_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{x}}'}\sigma^2 \tag{3}$$

where α^2 is the variance of the latent function, σ^2 is the noise variance, and $\boldsymbol{\Lambda}^{-1}$ is a positive-definite matrix. Assuming that $\boldsymbol{\Lambda}^{-1}$ is a diagonal matrix whose elements are $\{l_1, l_2, \ldots, l_r\}$, the set of parameters $\boldsymbol{\theta} = \left(l_1, l_2, \ldots, l_r, \alpha^2, \sigma^2\right)$ is called hyper-parameters.

With N pairs of training inputs $\tilde{\boldsymbol{x}}_i$ and outputs y_i $(i = 1, 2, \ldots, N)$, we can predict the output for a new input $\tilde{\boldsymbol{x}}^*$ by

$$y^* = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y} \tag{4}$$

with variance

$$\sigma^2 = k(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*) - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_* \tag{5}$$

where $\boldsymbol{K} = \{k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)\} \in \Re^{N \times N}$ and $\boldsymbol{k}_* = \{k\left(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}_i\right)\} \in \Re^N$.

The hyper-parameters are normally optimized to maximize the marginal likelihood of producing the training data. In our setting, however, optimizing hyper-parameters often results in over-fitting due to the small number of training data. We therefore manually adjust the hyper-parameters by looking at the policy optimization results.

C. Learning

We collect the input and output data from hardware experiments to train the dynamics bias model. For multiple-output systems, we use one GP for each dimension and train each GP independently using the outputs obtained from the same set of inputs.

The inputs to the GP models are the current state and input, $\tilde{\boldsymbol{x}}_t = \left(\boldsymbol{x}_{t-1}^T \; \boldsymbol{u}_{t-1}^T\right)^T$, while the outputs are the difference between the measured state and the prediction of the simulation model:

$$\boldsymbol{\Delta}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1} - \boldsymbol{f}'\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}\right). \tag{6}$$

We collect a set of input and output pairs from hardware experiments.

The computational cost for learning increases rapidly as the training data increases. We therefore remove some of the samples from learning data set. First, we downsample the data because similar states do not improve model accuracy. We then remove the samples where the robot and board are no longer balancing on the wheel. Next, we discard the samples whose states are too far away from the static equilibrium state or too difficult to recover balance, since designing a controller in such areas of the state space does not make much sense. Finally, we discard the frames that are far from the prediction by the simulation model in order to remove outliers that may happen due to sensor errors in hardware experiments.

To summarize, samples with the following properties are not included in the training data:

1) The board touches the ground.
2) The board and wheel are detached.
3) The distance from the static equilibrium state is larger than a threshold.
4) The global angle of the robot body exceeds a threshold.
5) The global angle of the board exceeds a threshold.
6) The norm of the velocity exceeds a threshold.
7) The distance from the state predicted by the Lagrangian model is larger than a threshold.

D. Prediction

In policy search, we use the dynamics bias model to predict the next state $\boldsymbol{x}_t$ given the current state $\boldsymbol{x}_{t-1}$ and input $\boldsymbol{u}_{t-1}$. The GP model predicts the mean $\bar{\boldsymbol{\Delta}}_t$ and variance $\boldsymbol{\sigma}_t$ of the output, and the mean value is commonly used as the prediction. A problem with this method is that the prediction is not accurate if the input is far from any of the training data, especially when the training data is sparse as in our case. Here, we take advantage of the system dynamics model $\boldsymbol{f}'$ by weighing the prediction of the GP such that we rely on the model as the prediction variance becomes larger, i.e.,

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{f}'(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) + \exp\left(-d|\boldsymbol{\sigma}^2|^2\right)\bar{\boldsymbol{\Delta}}_t \tag{7}$$

where $d > 0$ is a user-defined coefficient. If $\left(\boldsymbol{x}_{t-1}^T \; \boldsymbol{u}_{t-1}^T\right)^T$ is far away from any learning data, then the last term of Eq.(7) is nearly zero, meaning that we mostly use the prediction by the model.

V. Data-Efficient Reinforcement Learning

Algorithm 1 summarizes our framework. The algorithm starts from an empty learning data set $D = \emptyset$ and the assumption that the simulation model is accurate, i.e., $\boldsymbol{g} = 0$. At each iteration, we first search for an optimal policy using the simulation model $\boldsymbol{f}' + \boldsymbol{g}$. If the optimal policy does not give satisfactory results with the simulation model, we clear the model and restart from scratch. Otherwise, we evaluate the policy by a few hardware experiments to obtain the maximum cost as well as a new data set D_i for learning. If the policy successfully achieves the control objective on hardware, we terminate the iteration. Otherwise, we append D_i to the existing data set and re-learn the dynamics bias model $\boldsymbol{g}$ and repeat the same process until the maximum number of iterations is reached.

The cost function for policy optimization is

$$Z = c(T - t_{fail}) + \max_{1 \le t \le T} \boldsymbol{x}_t^T \boldsymbol{R} \boldsymbol{x}_t + \sum_{t=0}^{T} \boldsymbol{u}_t^T \boldsymbol{Q} \boldsymbol{u}_t \tag{8}$$

where c is a user-defined positive constant, T is the number of simulation frames, t_{fail} is the frame at which the simulation failed, and $\boldsymbol{R} \in \Re^{n \times n}, \boldsymbol{Q} \in \Re^{m \times m} \ge 0$ are user-defined weight matrices. We set $c = Z_{max}$ to make sure that the cost function value always exceed Z_{max} if a policy fails to keep the robot balanced for T frames. The first term

Algorithm 1 Data-efficient reinforcement learning

```
Require: nominal model $\boldsymbol{f}$
 1: initialize $D = \emptyset$ and $\boldsymbol{g} = 0$
 2: $i \leftarrow 0$
 3: while $i < N_{out}$ do
 4:     $p \leftarrow$ policy optimized for $\boldsymbol{g}$
 5:     $Z_g \leftarrow$ evaluate policy $p$ on $\boldsymbol{g}$
 6:     if $Z_g > Z_{max}$ then
 7:         initialize the simulation model: $D = \emptyset$ and $\boldsymbol{g} = 0$
 8:     end if
 9:     $Z_r, D_i \leftarrow$ evaluate $p$ with hardware experiments
10:     if $Z_r < Z_{max}$ then
11:         break
12:     end if
13:     $D \leftarrow D \cup D_i$
14:     $\boldsymbol{g} \leftarrow \boldsymbol{g}_D$
15:     $i \leftarrow i + 1$
16: end while
```

penalizes policies that cannot balance the model for at least T frames. To determine failure, we use the criteria 1)–6) described in Section IV-C. The second term tries to minimze the maximum distance from the static equilibrium state. The third term considers the total energy consumption for control.

Any numerical optimization algorithm can be used for optimizing the policy p using the simulation model. We have found that the DIRECT algorithm [20] works best for our problem. Theoretically, the DIRECT algorithm is capable of finding the globally optimal solution relatively quickly. We terminate the algorithm when the relative change in the cost function value in an optimization step is under a threshold ϵ, or the number of cost function evaluations exceeds a threshold N_{in}. DIRECT also requires the upper and lower bounds for each optimization parameters.

VI. Results

While the final goal of this work is to optimize a policy for hardware systems, this paper focuses on proof of concept and uses two different simulation models in place of a simulation model and hardware. Using a well-controlled simulation environment also gives us the opportunity to explore different noise types and levels.

A. Bongoboard Balancing

The task we consider is balancing on bongoboard of a simple legged robot shown in Fig. 3(a). Specifically, we apply the output-feedback controller developed by Nagarajan and Yamane [21] and attempt to optimize the gains through model learning and policy search. The state of the system is $\boldsymbol{x} = (\alpha_w \ \alpha_b \ \theta_1^r \ \dot{\alpha}_w \ \dot{\alpha}_b \ \dot{\theta}_1^r)^T$ (see Fig. 3(a)), and the outputs we use for feedback control are $\boldsymbol{z} = (x_p \ \dot{x}_p \ \theta_1^r \ \dot{\theta}_1^r \ \alpha_f)^T$ as indicated in Fig. 3(b).

The system has three degrees of freedom, and the only input is the ankle torque. Therefore the number of states is $n = 6$ and the number of inputs to the model is $m = 1$. Then the number of inputs to the GP becomes $r = n + m = 7$.

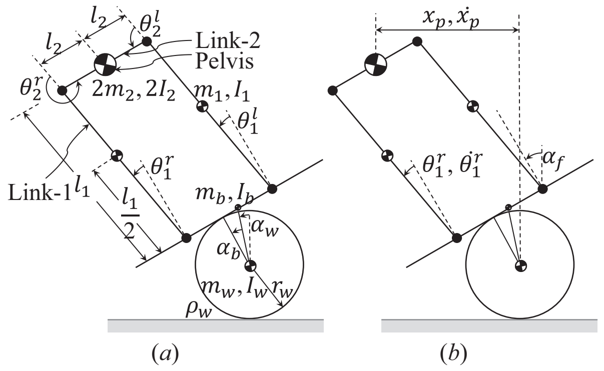

Fig. 3. Robot balancing on a bongoboard.

The output-feedback controller takes the five outputs of the models and compute the ankle torque by

$$u = \boldsymbol{H}\boldsymbol{z} \tag{9}$$

where $\boldsymbol{H} = (h_1 \ h_2 \ \dots \ h_5)$ is the feedback gain matrix. The policy search process computes the optimal values for the five elements of the gain matrix. In our implementation, we optimize a different set of parameters $\hat{\boldsymbol{h}}$ that are mapped to the elements of $\boldsymbol{H}$ by

$$h_i = \begin{cases} \exp(\hat{h}_i) - 1 & \text{if } \hat{h}_i \geq 0 \\ -\exp(-\hat{h}_i) + 1 & \text{if } \hat{h}_i < 0 \end{cases} \tag{10}$$

instead of directly optimizing h_i.

As mentioned above, we use two models in this paper, one corresponding to the *simulation* and the other corresponding to the *hardware* blocks for Fig. 1.

The first model is derived by the Lagrangian dynamics formulation as described in [21]. This model assumes perfect contact condition, i.e. no slip or detachment of contacts between the floor and wheel, the wheel and board, as well as the board and robot feet.

The second model, used in lieu of hardware, is based on a 2D physical simulation engine called Box2D [22], which uses maximal (Eulerian) coordinate system and a spring-and-damper contact model. To make the simulation realistic, we add three types of noise:

- Torque noise: a zero-mean gaussian noise of variance σ_τ^2 is added to the robot's ankle joint torque.
- Joint angle noise: a zero-mean Gaussian noise of variance σ_p^2 is added to the wheel (α_w), board (α_b), and robot (θ_1^r) angles used for feedback control.
- Joint velocity sensor noise: a zero-mean Guassian noise of variance σ_v^2 is added to the wheel ($\dot{\alpha}_w$), board ($\dot{\alpha}_b$), and robot ($\dot{\theta}_1^r$) anglular velocities.

We also randomly choose the initial states in Box2D simulations for collecting training data for dynamics bias model learning because it is impossible to set exact initial states in hardware experiments.

Even though both are simulation, the results may be different due to different contact models and coordinate

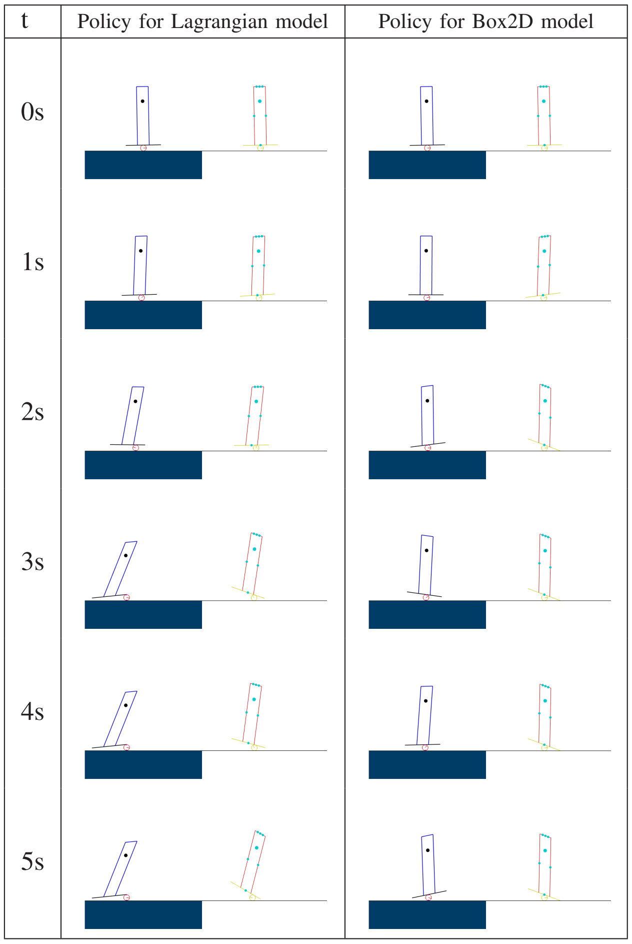

Fig. 4. Simulation result of a policy optimized for the Lagrangian model (left column) and Box2D model (right column). In each snapshot, the left and right figures are the Box2D and Lagrangian model simulations respectively.

systems. In fact, a policy optimized for the Lagrangian model does not always balance the robot in the second model, which justifies the need for our framework even in this simple setup. Figure 4 show an example of using a policy optmized for the Lagrangian and Box2D models for both the Box2D model and the Lagrangian model. Both policies can successfully balance the model for which they are designed, but not the other model. With the policy designed for the Lagrangian model, the Box2D simulation fails before $t = 3$ sec when the board leaves the wheel. The Lagrangian model simulation with the policy designed for Box2D model fails when the board hits the ground before $t = 2$ sec.

Table I summarizes the parameters we used for the experiments.

B. *Dynamics Bias Learning*

To ensure that the GP models can accurately predict the dynamics bias, we draw the vector field in the 3-dimensional subspace $(\alpha_w \ \alpha_b \ \theta_1^r)$ of the state spate. An example is shown in Fig. 5, where the cyan arrows represent the training data and red and blue arrows depict the prediction and ground truth computed at different states. This example uses 571

TABLE I

PARAMETERS USED FOR THE EXPERIMENTS.

Dynamics Bias Model	
$\boldsymbol{\Lambda}^{-1}$	$diag(1,1,1,1,1,1)$
α^2	1
σ^2	e^{-4}
d	1.0
Policy Optimization	
c	200
T	5000
Q	10^{-6}
$\boldsymbol{R}$	$diag(10,10,10,0.1,0.1,0.1)$
N_{out}	10
Z_{max}	200
experiments per iteration	2
DIRECT parameters	
parameter bounds	$-10 \leq \hat{h}_i \leq 10$
N_{in}	1000
ϵ	10^{-6}
Simulation Setting	
maximum torque	100 Nm
timestep	0.001 s

samples obtained from four Box2D simulations. As shown here, the corresponding red and blue arrows match well, indicating that the GP models can accurately predict the dynamics bias.

C. *Policy Search*

We run our method for different noise levels and inertial parameter error magnitudes to investigate the relationship between the number of experiments required and the discrepancy between the model and hardware. Furthermore, to test the robustness against model errors, we conducted the same set of experiments when the inertial parameters of the Box2D model are 20% larger than those in the Lagrangian model.

Table II shows the average number of experiments required to obtain a policy that can successfully balance the robot in Box2D simulation for 5 seconds. For reference, a 12-bit rotary encoder combined with a 50:1 gear measures the output joint angle at a resolution of 3.1×10^{-5} rad.

The results do not show any clear relationship between the noise level and the number of experiments required, which implies that larger noise or error does not necessarily require more experiments. Also, it is interesting that the numbers of experiments with inertial parameter errors are generally lower than their counterparts without errors. We suspect that the larger inertia lowered the natural frequency of the system, making the control easier in general.

Figure 6 shows three examples of cost function value change in Box2D simulation. The cost generally remains flat for a few iterations and then declines rapidly, probably when the dynamics bias model becomes accurate enough.

D. *Policy Performance*

Since the Box2D simulation includes noise, simulation results vary even if the robot starts from the same initial state and uses the same policy. We therefore compute the success rate from various initial states to evaluate the performance of a policy.

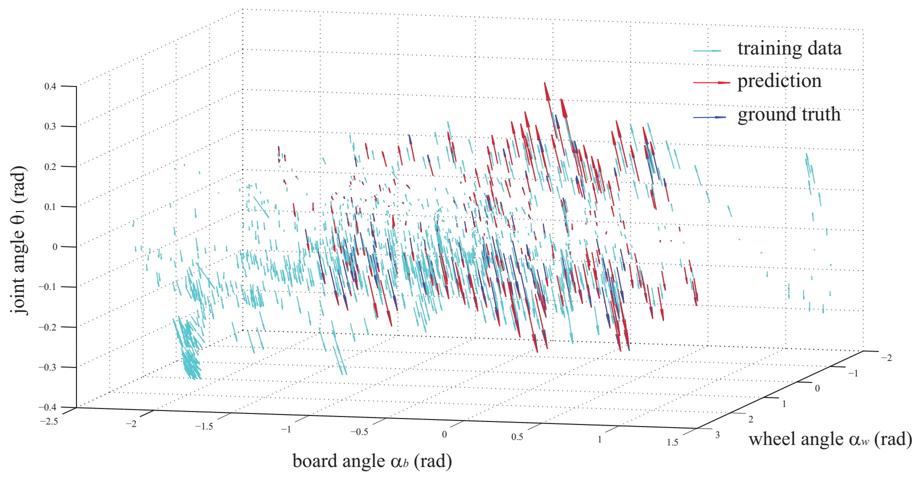

Fig. 5. Velocity field of the learned dynamics model. Cyan: training data; red: prediction; blue: ground truth.

TABLE II
AVERAGE NUMBER OF EXPERIMENTS REQUIRED AT DIFFERENT NOISE LEVELS AND INERTIAL PARAMETER ERRORS.

Torque σ_τ^2	Position σ_p^2	Velocity σ_v^2	# of experiments no error	# of experiments 20% error
0	0	0	6.4	2.8
0.001	0	0	7.3	3.5
0.01	0	0	9.5	4.8
0.1	0	0	5.5	2.5
0.1	1.0×10^{-6}	1.0×10^{-3}	7.5	3.5
0.1	2.0×10^{-6}	2.0×10^{-3}	4.4	4.4
0.1	4.0×10^{-6}	4.0×10^{-3}	7.0	3.3
0.1	8.0×10^{-6}	8.0×10^{-3}	9.6	5.0
0.1	1.6×10^{-5}	1.6×10^{-2}	4.6	5.5
0.1	3.2×10^{-5}	3.2×10^{-2}	4.0	3.5
0.1	6.4×10^{-5}	6.4×10^{-2}	6.0	4.0
0.1	1.28×10^{-4}	1.28×10^{-1}	4.0	3.6

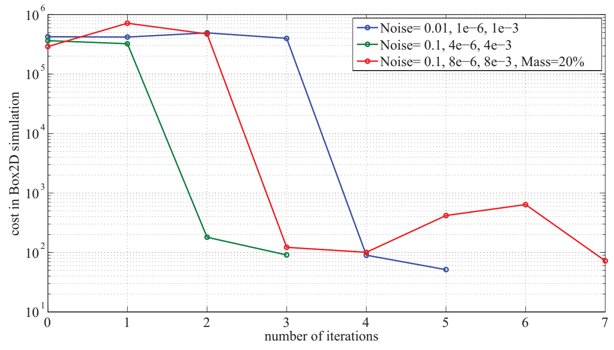

Fig. 6. Change of cost function value in Box2D simulations over iterations.

Figure 7 depicts the balancing success rates starting from various wheel and board angles, using a policy optimized with Box2D simulation without noise (a) and with noise (b). This result clearly shows that the policy optimized in noisy environment can successfully balance the robot from a wider range of initial states under noisy actuator and sensors.

VII. Conclusion and Future Work

This paper presented a framework for model learning and policy optimization of robots that are difficult to conduct experiments with. The key idea is to learn the difference between a model and hardware rather than learning the hardware dynamics from scratch. We also employ an iterative learning process to improve the model and policy This approach is particularly useful for tasks such as humanoid balancing and locomotion where a dynamics model is necessary to obtain a controller to collect the initial set of data.

We conducted numerical experiments through bongoboard balancing task of a simple bipedal robot, and demonstrated that the framework can compute a policy that successfully completes the test task with only several hardware experiments. The policy obtained from noisy simulation proved to have higher balancing performance than the one obtained from clean simulation. The number of hardware experiments did not show clear correlation with the noise level or magnitude of inertial parameter error.

Future work besides experiments with actual hardware system includes establishing a guideline for determining the hyper-parameters of GP. Also, we want to test the scalability of our framework by testing it on more complex robot models. Although Wang et al. [23] successfully predicted the fullbody motion with GP, learning dynamics of complex robots is likely to be suffered from the curse of dimensionality. One possible solution is learning a low dimensional projection of the full space using an abstract model [14] or a dimensionality reduction technique [24]. Another interesting direction would be to explore different representation of dynamics bias instead of the additive bias considered in this paper.

References

[1] J. Kober and J. Bagnell, J.A. amd Peters, “Reinforcement learning in robotics: A survey,” *the International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
[2] W. Khalil and E. Dombre, *Modeling, identification and control of robots*. London, U.K.: Hermès Penton, 2002.

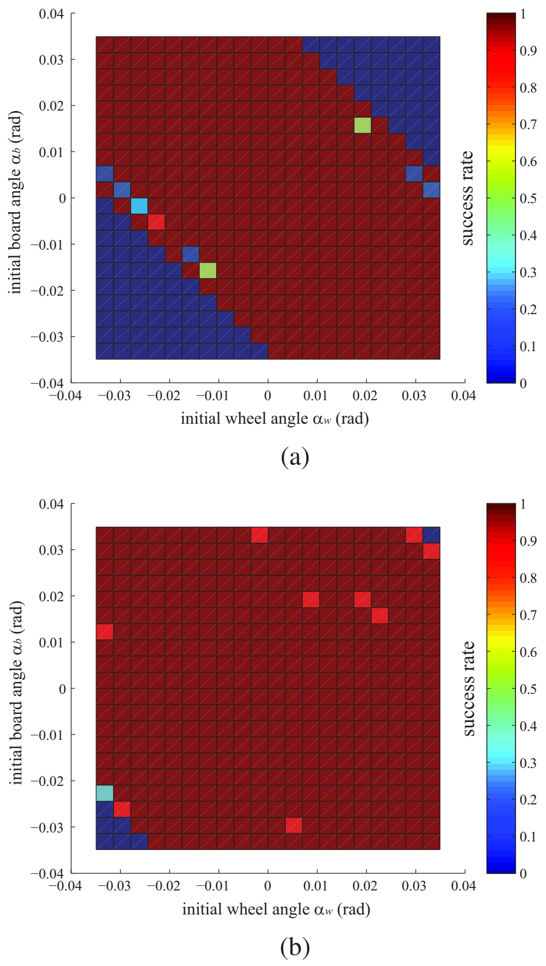

Fig. 7. Balancing success rate in Box2D simulation with noise, starting from various initial wheel and board angles. (a) The policy has been optimized with Box2D simulation without noise. (b) The policy has been optimized with Box2D simulation with noise.

[3] K. Yamane, "Practical kinematic and dynamic calibration methods for force-controlled humanoid robots," in *Proceedings of IEEE-RAS International Conference on Humanoids Robots*, Bled, Slovenia, October 2011, p. (in press).

[4] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[5] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems*, 2008, pp. 849–856.

[6] 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, pp. 79–91, 2004.

[7] C. Atkeson and S. Schaal, "Robot learning from demonstration," in *International Conference on Machine Learning*, 1997, pp. 12–20.

[8] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the 7th International Conference on Machine Learning*, 1990, pp. 216–224.

[9] A. Moore and C. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[10] J. Peng and R. Williams, "Incremental multi-step Q-Learning," *Machine Learning*, vol. 22, pp. 283–290, 1996.

[11] P. Abbeel, M. Quigley, and A. Ng, "Using inaccurate models in reinforcement learning," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 1–8.

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

[13] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 465–472.

[14] J. Morimoto, C. G. Atkeson, G. Endo, and G. Cheng, "Improving humanoid locomotive performance with learnt approximated dynamics via Gaussian processes for regression," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4234–4240, 2007.

[15] N. Sugimoto and J. Morimoto, "Trajectory-model-based reinforcement learning : Application to bimanual humanoid motor learning with a closed-chain constraint," *IEEE-RAS International Conference on Humanoid Robots*, 2013.

[16] V. Tangkaratt, S. Mori, T. Zhao, J. Morimoto, and M. Sugiyama, "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation." *Neural networks*, vol. 57, Sep. 2014.

[17] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3888–3895, 2014.

[18] S. Ross and J. Bagnell, "Agnostic system identification for model-based reinforcement learning," in *International Conference on Machine Learning*, 2012.

[19] C. Rasmussen and M. Kuss, "Gaussian Processes in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[20] D. Jones, C. Perttunen, and B. Stuckman, "Lipscitzian optimization without the Lipschitz constant," *Journal of Optimization Theory*, vol. 79, no. 1, pp. 157–181, 1993.

[21] U. Nagarajan and K. Yamane, "Universal balancing controller for robust lateral stabilization of bipedal robots in dynamic, unstable environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2014, pp. 6698–6705.

[22] "Box2d — a 2d physics engine for games," http://box2d.org/.

[23] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models," *Advances in neural information processing systems*, pp. 1441—-1448, 2005.

[24] E. L. Snelson, "Flexible and efficient Gaussian process models for machine learning," *Ph.D Thesis*, 2007.