2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

Data Efficient Learning of Robust Control Policies

Susmit Jha[1] and Patrick Lincoln[1]

Abstract— **This paper investigates data-efficient methods for learning robust control policies. Reinforcement learning has emerged as an effective approach to learn control policies by interacting directly with the plant, but it requires a significant number of example trajectories to converge to the optimal policy. Combining model-free reinforcement learning with model-based control methods achieves better data-efficiency via simultaneous system identification and controller synthesis. We study a novel approach that exploits the existence of approximate physics models to accelerate the learning of control policies. The proposed approach consists of iterating through three key steps: evaluating a selected policy on the real-world plant and recording trajectories, building a Gaussian process model to predict the reality-gap of a parametric physics model in the neighborhood of the selected policy, and synthesizing a new policy using reinforcement learning on the refined physics model that most likely approximates the real plant. The approach converges to an optimal policy as well as an approximate physics model. The real world experiments are limited to evaluating only promising candidate policies, and the use of Gaussian processes minimizes the number of required real world trajectories. We demonstrate the effectiveness of our techniques on a set of simulation case-studies using OpenAI gym environments.**

I. INTRODUCTION

Synthesis of control policies for dynamical systems is critical in many domains such as networks, robotics, cyber-physical systems and systems biology. While synthesis of control for known dynamics model has been well-studied in literature, the recent success in data-driven learning has inspired a number of techniques for synthesizing control of plants with unknown dynamics. These techniques for learning control policies when plant's dynamics are unknown can be broadly classified into two classes: model-based methods that attempt at learning the system dynamics before synthesizing control policies, and model-free methods that search for best control policies for a given task without explicitly learning the system dynamics. Model-based approaches learn plant models such as differential equations, Gaussian processes or Markov decision processes (MDP), and then use corresponding control synthesis techniques. Model-free methods directly learn policies but do not easily generalize to unobserved regions of the plant's behavior.

Reinforcement learning methods can be used for control synthesis by approximately learning the Q-values [12], and then picking the action for each state that maximizes the Q-value. Reinforcement learning methods such as direct policy search and trust region policy optimization (TRPO) [18] are examples of model-free approaches for synthesizing control policy directly. The use of deep neural networks in representing policies in reinforcement learning enables the application of scalable stochastic gradient descent based optimization methods, and is accredited with recent success in synthesizing control for highly nonlinear stochastic systems. Unfortunately, these techniques are typically very 'data hungry' requiring a lot of training data before the algorithms converge to a good control policy. Further, some of these training trajectories could be potentially unsafe for the physical system. Consequently, the use of reinforcement learning in real world becomes impractical if all learning data is directly obtained through experiments on the physical plant. This issue is often alleviated in practice by using simulation models for early stages of training. These simulation models use off-the-shelf physics engines that provide convenient platforms for modeling the dynamics of the robots or other plants that will interact with the controller. Reinforcement learning can be used with the simulation models but the learned policy may not work due to model inaccuracies. This is often referred as *reality gap*.

This paper addresses this challenge of *reality gap* to enable efficient learning of robust control policies using approximate physics models. We present a novel approach for more accurate model identification using off-the-shelf physics engines that reduces the number of real-world experiments. The plant to be controlled can often be modeled at a high-level using physics engines but the inaccuracies arise due to unknown parameters in the models. For instance, a simple example of a cart-pole can be easily qualitatively modeled using a physics engine but parametric details such as the mass of the cart, the friction between the wheels and the floor, and the length of the pole are difficult to get exactly correct. These inaccuracies can cause the behavior of the simulation model to diverge from the real-world experiments. An eager approach would be to let system identification learn the model parameters as accurately as possible before controller synthesis. But such an eager approach can potentially lead to wasted effort and needlessly large number of real-world trajectories because the discovery of optimal control policy may not actually require very accurate estimation of some of the parameters. Simulation models can often describe only a narrow slice of the overall behavior of a plant, and so, configuring these models requires us to know what behaviors the plant is expected to exhibit, which in turn, requires us to know the optimal policy that is yet to be synthesized. This

The authors acknowledge support from the National Science Foundation (NSF) Cyber-Physical Systems #1740079 project, NSF Software & Hardware Foundation #1750009 project, and US ARL Cooperative Agreement W911NF-17-2-0196 on Internet of Battlefield Things (IoBT).

[1] The authors are with Computer Science Laboratory, SRI International, Menlo Park, USA. Emails: `susmit.jha@sri.com`, `patrick.lincoln@sri.com`

creates a cyclic dependency between system identification for estimating the model parameters, and reinforcement learning for finding the optimal policy. In this paper, we adopt a novel approach of dovetailing identification of model parameters, and reinforcement learning that facilitates data efficient learning by minimizing the number of real-world trajectories.

The rest of the paper is organized as follows. In Section II, we describe relevant related work. We present the proposed approach in Section III and demonstrate its effectiveness with experiments in Section IV. We discuss the limitations of the proposed approach and mention ongoing work in conclusion in Section V.

II. RELATED WORK

The emergence of deep neural networks as high-capacity function approximators has led to increased success in the field of reinforcement learning [12], [19], [9], [10]. However, a significant roadblock in the widescale application of these methods is their reliance on large amount of data. One approach to address this data scarcity in the real-world is to first learn a policy using a simulation model and then transfer the learned policy to the real system. However, the environment and physics of the simulator are often not exactly the same as the real world. This causes the behavior of the simulation model to diverge from the real system. This reality gap results in unsuccessful transfer if the learned policy is not robust to modeling errors in the simulator, and several techniques have been proposed recently to accomplish this transfer [6], [16]. In contrast to these efforts, we do not aim at successful transfer of policy learned on approximate models to the real-world. Instead, we aim at learning a good enough approximation of the real-world to enable discovery of optimal policy minimizing the number of real world experiments.

Another closely related area is that of model-free learning that integrates physics engines [4], [2], [22] with end to end learning. These techniques use the physics models to learn end to end mapping from the observations to control inputs. These techniques do not try to refine the accuracy of physics model to reduce the number of real-world trajectories, but instead use physics model to represent part of the pipeline mapping observations to control inputs. In contrast, we attempt at learning a good enough approximation of the physical model itself. While model-free learning techniques have been studied for a long time [20], and their effectiveness in synthesizing control has been also established [18], [12], our approach attempts to make these approaches more data efficient.

In contrast to model-free learning, model-based control involves explicitly learning the unknown system dynamics. This dynamics model is then used for discovering the optimal policy. It has been previously reported in model-based control that even approximate models can yield near optimal policy [1], [7]. This motivates our effort to learn model parameters simultaneously with the policy where the attempt is not to learn an accurate model but just good enough to learn a near optimal policy. System identification [14], [13] to build dynamics model has also been independently studied in control literature outside of reinforcement learning context. In contrast, we consider parametric physics models and focus on learning approximate value of these parameters instead of discovering the physics model from scratch.

Data efficient learning of dynamics model for synthesizing control policy has also received significant attention recently. Deisenroth et al [8] developed a data-efficient reinforcement learning method by incorporating state-space constraints in the learning process and demonstrated the success on a set of stacking tasks. Zhu et al [23] proposed a fast model identification approach comprising of Gaussian process based estimation of parameter probability, and TRPO based policy learning that is similar to the technique presented in the paper. The choice of parameter for selecting real-world experiments in [23] is greedy while we rely on entropy search. Chatzilygeroudis et al [5] proposed a data-efficient model-based RL algorithm, called BlackDROPS (Black-box Data-efficient RObot Policy Search) that replaces the gradient-based optimization algorithm with a parallel, blackbox algorithm that takes into account the model uncertainties. Saveriano et al. [17] developed an approach for policy improvement with residual model learning (PI-REM) which focuses on learning the residual dynamics between the simulator and reality. Such a residual learning approach can be combined with the parametric learning technique presented in this paper if qualitative models of residuals are available.

III. APPROACH

We reduce the number of real-world trajectories required for learning robust control policies by avoiding the extra effort to learn accurate models of the plant that is universally valid over the entire state-action space. Instead, we use an iterative method to learn model parameters around the optimal policy. The overall approach is sketched in Figure 1. We start with a random safe policy and execute it on the plant to generate real-world trajectories. These trajectories are used to estimate a probability distribution over the parameter values that indicates the likelihood of a parameter value to be correct. This probability distribution is refined to be of low entropy by selecting some values of the parameters and obtaining trajectories from the simulation model configured with these values. The low entropy probability distribution is then used to sample a candidate parameter value. This candidate is accepted as the final parameter value if two conditions are satisfied. First, the candidate parameter is close to the most likely parameter predicted by the distribution. Second, the value function attained by the policy learned from the simulation engine configured to this parameter value is within a small threshold of the maximum value function attained in the parameter's neighborhood. This stopping criteria is guided by the guarantee on the KL divergence between successive policy revisions of the Trusted Region Policy Optimization (TRPO) algorithm used for reinforcement learning in our approach. If this criteria is not met, then we run the learned policy on the real system and add the corresponding trajectory to our training set. The estimation of probability distribution of parameter values is repeated, and we again sample a parameter value from the distribution and check whether the stopping criteria is satisfied. In rest of this section, we detail the critical steps of the algorithm.

A. Reinforcement learning

We use reinforcement learning methods for synthesizing control policies using the simulation model. The dynamics model of the plant to be controlled comprises of states S. A policy π is a mapping from states to actions A. A reward R is provided for a given state and selected action which encodes the task specification. The goal of controller synthesis is to find an optimal policy that maximizes the total expected reward. Typically, a value function $V_\pi(s)$ is associated to each state that denotes the long term expected reward of the state s. Model-based reinforcement learning attempts at learning the transition probability $P(s'|s,a)$ of getting to state s' from state s on action a. But as the space of states and actions become large, learning these transition probabilities becomes difficult. This problem is particularly severe in continuous control problems where the action space is continuous, and discretization results into a large number of actions. A Q-value function $Q_\pi(s,a)$ can also be associated to each state and action that denotes the expected reward of executing action a in state s. Deep learning implementations (DQN) [12] of Q-learning have proved to be very effective in learning control policies over high-dimensional states such as those in Atari games [11]. The table size of Q-value also grows rapidly with the increase in the number of actions.

In our attempt to make the process of learning control policies more data-efficient, we chose to focus on model-free reinforcement learning methods that scale better with large state-action spaces. In particular, we use Trust Region Policy Optimization (TRPO) [18] algorithm for reinforcement learning. TRPO directly searches over policies using gradient of the policy network. A key characteristic of TRPO is that the KL divergence between updated policy in the current iteration and the previous policy is guaranteed to have a bounded KL divergence. This enables the use of previous policy to evaluate different possible models of the plant. The search for best possible model is continued until the set of likely plant models predict similar values for the previous policy. Since the new policy is not much different than the previous policy, any of these models can be used to search for the next policy.

B. GP based estimation of model parameters probability

Let β denote the unknown discrete parameters of the plant model. We assume that the values of these parameters lie in a finite set B. In case of continuous parameters such as friction or length or mass, we assume that a suitable discretization has been performed to obtain B. Our algorithm maintains a probability distribution over $\beta \in B$ and iteratively refines this distribution. The unknown parameterized dynamics of the plant for a control policy π is given by

$$x_{t+1} = F(x_t, \pi(x_t), \beta)$$

where a simulator using off-the-shelf physics engine implements the function F but the parameters β are not known and need to be learned simultaneously with the optimal controller. We also parameterize the value function as $V_\pi(s,\beta)$ to denote the expected reward for state s computed using the simulation model with parameter value β.

The overall algorithm comprises of dovetailed model identification to find good approximations of β followed by controller synthesis to find optimal policy π. Giving an initial policy π_0 and initial distribution P over the parameters β, we repeat the following three steps for $t = 0, 1, 2, \ldots$

1) Roll out the policy π_t on the real-world plant and collect the trajectories.
2) Update distribution P of the parameters β by sampling β and running simulations using the physics model, and learn a Gaussian process approximation of the model-deviation function.
3) Sample parameter value β_{t+1} from B according to the updated distribution, and use TRPO to find optimal policy for the model with β_{t+1}.

Given real-world trajectories $\mathcal{T}$ with different policies π_t, we can collect the observed current and next states along with the action, $(s,a,s') \in \mathcal{T}$ where the action $a = \pi_t(s)$ to compute the average deviation of the observations from the predictions of the simulation model with the parameters set to β:

$$\Delta(\beta) = \frac{1}{|\mathcal{T}|} \sum_{(s,a,s') \in \mathcal{T}} ||s' - F(s,a,\beta)||_2$$

The overall goal of the model identification step can be formulated as finding β that minimizes the function $\Delta(\beta)$. But the function Δ itself is not fixed and will change as we gather more real-world trajectories in $\mathcal{T}$. Black-box optimization techniques can be used to compute the minimum of this function but these methods would overfit the parameter β to the observations selecting the best possible model for current observations. These techniques do not allow trading off exploration and exploitation needed to design new experiments and collect more data to learn better estimate of β.

This motivates our choice of approximating the model-deviation function Δ as a Gaussian process (GP) [15] mapping the parameters β to the average deviation between the model trajectories and the observed real-world trajectories. Let Δ be approximated by a GP inferred from the multiple policies learned and rolled out for different sampled model parameters. The mean and covariance matrix of this learned GP is denoted by μ and Σ, respectively.

If Δ_{GP} denotes the GP learned from the sampled β and corresponding observed averaged model-deviation $\Delta(\beta)$. We can revise the probability distribution over the model

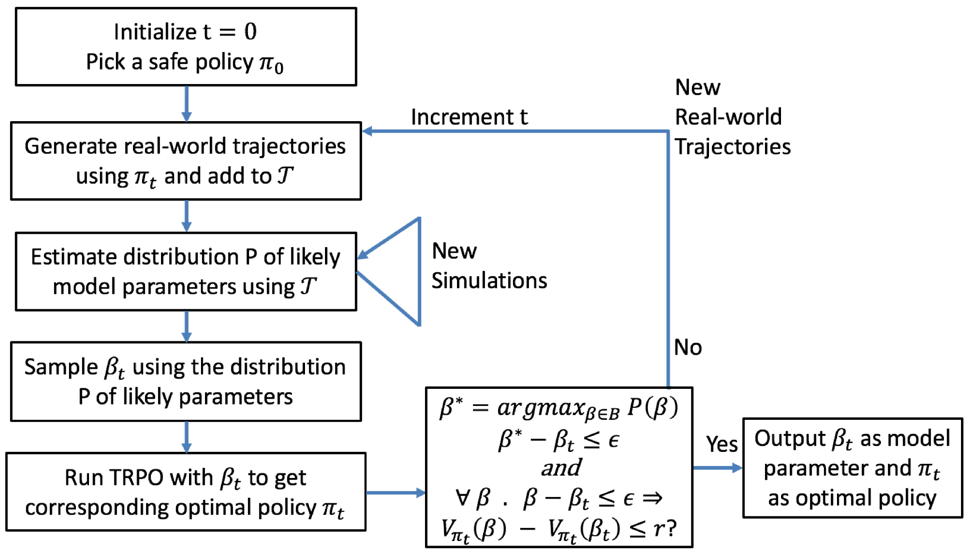
Initialize t = 0
Pick a safe policy π_0
Generate real-world trajectories using π_t and add to $\mathcal{T}$
Increment t
New Real-world Trajectories
Estimate distribution P of likely model parameters using $\mathcal{T}$
New Simulations
Sample β_t using the distribution P of likely parameters
No
Run TRPO with β_t to get corresponding optimal policy π_t
$\beta^* = argmax_{\beta \in B}\, P(\beta)$
$\beta^* - \beta_t \leq \epsilon$
and
$\forall \beta\ .\ \beta - \beta_t \leq \epsilon \Rightarrow$
$V_{\pi_t}(\beta) - V_{\pi_t}(\beta_t) \leq r$?
Yes
Output β_t as model parameter and π_t as optimal policy

Fig. 1. Overall algorithm for learning model parameters in loop with TRPO based policy search

parameters from this learned GP as follows:

$$P(\beta) = P(\beta = \arg\min_{b \in B} \Delta_{GP}(b))$$

Notice that Δ_{GP} is itself a random function that can be sampled from the learned Gaussian process. Given the mean μ and the covariance matrix Σ of the learned GP, let $p_{\mu,\Sigma}$ denote the probability of the model-deviation function to be Δ_{GP}. For a sampled Δ_{GP}, we can find the parameter $\beta_{\Delta_{GP}}$ that minimizes the sampled function. We use a 0-1 indicator function $\delta(\Delta_{GP}, \beta)$ which is 1 if and only if the parameter corresponding to function Δ_{GP} is not less than β, that is,

$$\delta(\Delta_{GP}, \beta) = 1\ if\ \beta = \beta_{\Delta_{GP}}\ and\ 0\ o.w.$$

The revised probability distribution can now be written as

$$P(\beta) = \int_{\Delta} p_{\mu,\Sigma}(\Delta)\ \cdot\ \delta(\Delta, \beta)\ \ d\Delta$$

The probability of the model parameter being β is the sum of probabilities of the model-deviation functions in which the parameter β minimizes the deviation.

We can use Monte Carlo sampling to approximate this revision of probability distribution. From the GP learned to model the deviation function Δ, we can sample $\Delta_1, \Delta_2, \ldots, \Delta_N$ functions from the GP and then evaluate these functions on all the parameter values in B. We can then approximate the revised probability distribution as:

$$P(\beta) \approx r/N$$

where r is the number of sampled deviation functions in which the parameter β minimizes the deviation function.

This revision of the probability distribution is continued till the entropy of the distribution continues to significantly decrease, or we reach the maximum number of samples allowed for β. We use a greedy approach to reduce entropy of the distribution P by picking the next parameter to try as

$$\beta_{next} = \arg\max_{\beta \in B} -P(\beta)\log(P(\beta))$$

We can simulate the plant using off-the-shelf physics engines with unknown parameters set to β_{next}. This is then used to approximate $\Delta(\beta)$ using states which are visited in both the simulation and the real world trajectory. The newly computed $\Delta(\beta)$ is added to the data used to learn the Gaussian Process approximation Δ_{GP} again. This repetition is expected to reduce the overall entropy of the distribution P given by

$$\sum_{\beta \in B} -P(\beta)\log(P(\beta))$$

Intuitively, as the entropy decreases, the mass of the probability distribution shifts to one or few most likely parameter values that minimize the model-deviation function. If the entropy is high, it implies that we do not have a good estimate of what model parameter is likely to minimize the model deviation.

We make two important observations regarding the efficiency of the method used to approximate the distribution of model parameters P:

- The computation of next parameter β_{next} to be tried only requires a simple maximization of the contribution to entropy of each of the parameters in P and does not require any simulation of the physics engine model, or real-world trajectories.
- Once the next parameter β_{next} has been selected, we do not collect any new real world trajectory but instead just compute $\Delta(\beta_{next})$ for retraining the Gaussian process learning the model-deviation function.

C. Gathering real-world trajectories and stopping criteria

The distribution over the parameter values is revised using new simulation trajectories till the reduction in entropy falls below a threshold. We use this low entropy distribution to sample a parameter β_t. We run the TRPO reinforcement learning on the simulation model to obtain an optimal policy

π_t. We then check whether the parameter β_t and corresponding policy π_t satisfy the stopping criteria consisting of two conditions:

- The parameter β_t is close to the most likely parameter according to the estimated probability distribution P, that is, $\arg\max_{\beta \in B} P(\beta) - \beta_t \le \epsilon$.
- The value function of the policy π_t for the simulation model with parameters set to β_t is at most r threshold away from the policy's value function with respect to any parameter value within ϵ of the parameter β_t.

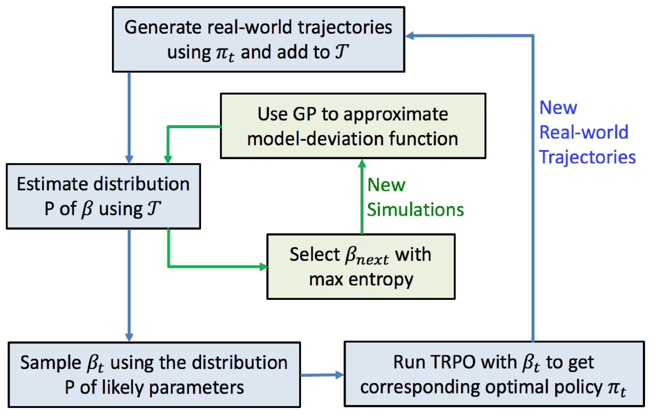
Generate real-world trajectories using π_t and add to $\mathcal{T}$
Use GP to approximate model-deviation function
New Real-world Trajectories
Estimate distribution P of β using $\mathcal{T}$
New Simulations
Select β_{next} with max entropy
Sample β_t using the distribution P of likely parameters
Run TRPO with β_t to get corresponding optimal policy π_t

Fig. 2. Nested parameter estimation and policy learning: Simulation trajectories are used to refine the Gaussian process estimation of model-deviation function, and construct low entropy estimation of probability distribution of the parameters. Real-world trajectories are gathered using parameter values sampled from the estimated distribution if the stopping criteria is not yet met.

If both these criteria are satisfied, then we can stop running the TRPO reinforcement learning algorithm and use the policy π_t as the discovered optimal policy. TRPO guarantees that the KL divergence between any two consecutive policies π_t and π_{t+1} is bounded. The value function of π_t is also within r threshold of the value function for any parameter β' that is within ϵ of the discovered parameter β_t. Since β_t is within ϵ of the most likely parameter given by the probability distribution, the value function for policy π_t is a good approximation of the optimal policy function that could be learned by further running TRPO.

If these conditions are not satisfied, we run the policy π_t on the real system and include this real-world trajectory in the set $\mathcal{T}$. This is used to recompute the model-deviation function Δ, and then relearn a GP approximation of the model-deviation function. Thus, our approach comprises of two nested loops as illustrated in Figure 2. The inner loop uses simulation environment to construct more accurate estimate of the probability distribution of the parameter values. This inner loop does not require any real-world trajectories. The outer loop uses real-world trajectories and is run very sparsely. This ensures the data efficiency of our approach for learning control policy.

IV. EXPERIMENTS

We now demonstrate the effectiveness of our approach using OpenAI gyms [3] control environments with the MuJoCo [21] physics simulator. We use simulation model with an unperturbed parameter value as a proxy for the real-world system. We construct the set B by considering 10% perturbation on the actual parameter. This defines the parameter search space. The goal is to find an optimal policy for the unperturbed model using a simulation model configured with parameters in the set B.

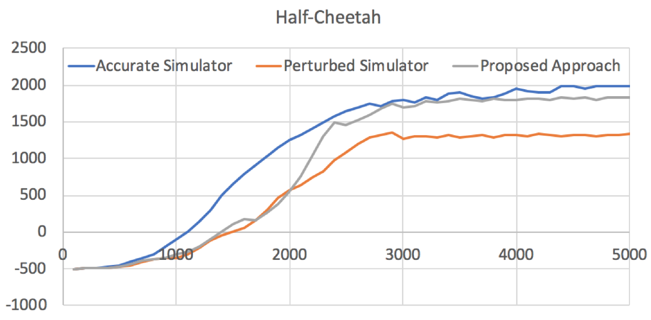

Fig. 3. Expected reward on Half-Cheetah environment

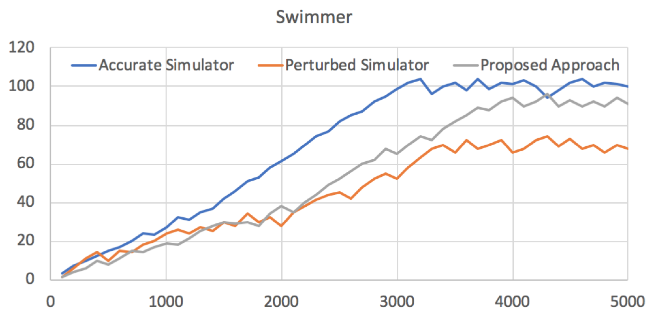

Fig. 4. Expected reward on Swimmer environment

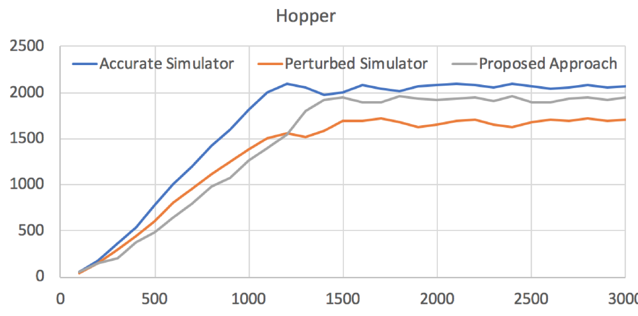

Fig. 5. Expected reward on Hopper environment

We use the following three OpenAI gym environments in our experimental evaluation:

- HalfCheetah: This is a planar biped robot with 8 rigid links, including two legs and a torso, along with 6 actuated joints. The 17 dimensional state space includes joint angles and joint velocities.
- Swimmer: This is a planar robot with 3 links and 2 actuated joints in a viscous container. The 8 dimensional state space includes joint angles and joint velocities.

- Hopper: This is a planar monopod robot with 4 rigid links, corresponding to the torso, upper leg, lower leg, and foot, along with 3 actuated joints. The 11 dimensional state space includes joint angles and joint velocities.

For each of the environments, we conducted three experiments each using a batch size of 20,000. We choose a large batch size to minimize variance in estimating value function of a candidate policy.

- We first ran TRPO reinforcement learning algorithm on the gym environment with the correct parameter values without perturbation. This provides the baseline for the maximum achievable reward.
- We also ran TRPO reinforcement learning algorithm by considering the worst perturbation in the parameters which allows us to compare the advantage of identifying parameters more accurately.
- Finally, we ran the algorithm presented in this paper.

For each of these experiments, we compute the expected reward for the revised policy in each iteration. We plot the expected rewards for the four environments in Figure 3, 4 and 5. While running TRPO algorithm with perturbed simulator produces significantly lower expected reward than the reward obtained by using an accurate simulator, the proposed approach is able to identify optimal policy and parameters using fewer number of real-world trajectories.

V. CONCLUSION

Learning optimal control policy of a plant in absence of an accurate model is difficult. A direct approach to model identification entails generating a large number of real-world trajectories by randomly initializing the policy and performing roll-outs. The search for best model parameters can be framed as an optimization problem that tries to minimize the deviation of the model behavior from the observed real-world trajectories. Learning an accurate dynamics model requires sufficiently sampling the state-action space of the dynamical system by suitably initializing the policy and performing a number of roll-outs. This direct approach is rather wasteful for two reasons. First, we need to learn the dynamics of the plant only close to the optimal policy for evaluating the optimal policy and its close-by candidates. Second, we need the dynamics model to be just good enough to infer the optimal policy. In many cases, an approximate dynamics model can provide approximate relative ordering of the candidate policies, and is thus, sufficient to identify a near-optimal policy. But exploiting these for data efficient learning of control policies is difficult due to the strong coupling between learning dynamics model and finding optimal policy. In this paper, we addressed this challenge by proposing an iterative search procedure that estimates the parameters of the dynamics model using Gaussian processes, and finds optimal policies using TRPO reinforcement learning algorithm, alternately. We demonstrate that the proposed approach converges to a near optimal policy requiring fewer real-world trajectories than a direct data-driven approach to policy learning.

REFERENCES

[1] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 1–8.

[2] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 5074–5082.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[4] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 173–180.

[5] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 51–58.

[6] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

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

[8] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[9] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[13] J.-I. Nagumo and A. Noda, "A learning method for system identification," *IEEE Transactions on Automatic Control*, vol. 12, no. 3, pp. 282–287, 1967.

[14] O. Nelles, *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer Science & Business Media, 2013.

[15] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71.

[16] 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.

[17] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-efficient control policy search using residual dynamics learning," in *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[20] R. S. Sutton, A. G. Barto, F. Bach, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[21] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[22] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," in *Advances in neural information processing systems*, 2015, pp. 127–135.

[23] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for data-efficient policy search," in *IJCAI*, 2018, pp. 3249–3256.