Fast Model Identification via Physics Engines for Data-Efficient Policy Search

Shaojun Zhu, Andrew Kimmel, Kostas E. Bekris, Abdeslam Boularias
Department of Computer Science, Rutgers University, New Jersey, USA
{shaojun.zhu, ask139, kostas.bekris, ab1544} @cs.rutgers.edu

Abstract

This paper presents a method for identifying mechanical parameters of robots or objects, such as their mass and friction coefficients. Key features are the use of off-the-shelf physics engines and the adaptation of a Bayesian optimization technique towards minimizing the number of real-world experiments needed for model-based reinforcement learning. The proposed framework reproduces in a physics engine experiments performed on a real robot and optimizes the model's mechanical parameters so as to match real-world trajectories. The optimized model is then used for learning a policy in simulation, before real-world deployment. It is well understood, however, that it is hard to exactly reproduce real trajectories in simulation. Moreover, a near-optimal policy can be frequently found with an imperfect model. Therefore, this work proposes a strategy for identifying a model that is just good enough to approximate the value of a locally optimal policy with a certain confidence, instead of wasting effort on identifying the most accurate model. Evaluations, performed both in simulation and on a real robotic manipulation task, indicate that the proposed strategy results in an overall time-efficient, integrated model identification and learning solution, which significantly improves the data-efficiency of existing policy search algorithms.

1 Introduction

Reinforcement learning (RL) typically requires a lot of training data before it can provide useful skills in robotics. Learning in simulation can help reduce the dependence on real-world data but the learned policy may not work due to model inaccuracies, also known as the "reality gap". This paper presents an approach for model identification by exploiting the availability of off-the-shelf physics engines [Erez *et al.*, 2015] for simulating robot and object dynamics in a given scene. One objective is to achieve data-efficiency for RL by reducing the need for many real-world robot experiments by providing better simulated models. It also aims to achieve time efficiency in the context of model identification by reducing the computational effort of this process.

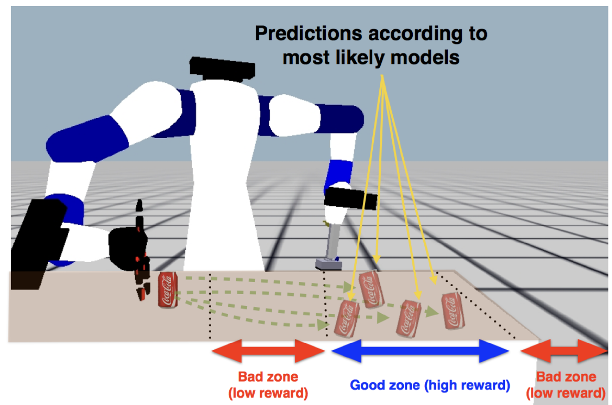

Figure 1: Example of a stopping criterion for model identification: if all high-probability models predict a high reward for a given action, there is no point in identifying a more accurate model.

The accuracy of a physics engine depends on several factors. The model assumed by the engine, such as Coulomb's law of friction, may be limited or the numerical algorithm for solving the differential equations of motion may introduce errors. Moreover, the robot's and the objects' mechanical parameters, such as mass, friction and elasticity, may be inaccurate. This work focuses on this last factor and proposes a method for identifying mechanical parameters used in physical simulations so as to assist in learning a robotic task.

Given initial real-world trajectories, potentially generated by randomly initializing a policy and performing roll-outs, the proposed approach searches for the best model parameters so that the simulated trajectories are as close as possible to the real ones. This is performed through an anytime black-box Bayesian optimization, where a belief on the optimal model is dynamically updated. Once a model with a high probability is identified, a policy search subroutine takes over the returned model and computes a policy to perform the target task. The subroutine could be a control method, such as a linear–quadratic regulator (LQR), or an RL algorithm that runs on the physics engine with the identified model. The obtained policy is then executed on the real robot and the newly observed trajectories are fed to the model identification module, to repeat the same process.

The question that arises is how accurate the identified model should be to find a successful policy. Instead of spending time searching for the most accurate model, the optimization can stop whenever a model that is sufficiently accurate to find a good policy is identified. Answering this question exactly, however, is difficult, because that would require knowing in advance the optimal policy for the real system.

The proposed solution is motivated by a key quality desired in many robot RL algorithms. To ensure safety, most robot RL algorithms constrain the changes in the policy between two iterations to be minimal and gradual. For instance, both Relative-Entropy Policy Search (REPS) [Peters *et al.*, 2010] and Trust Region Policy Optimization (TRPO) [Schulman *et al.*, 2015] algorithms guarantee that the KL divergence between an updated policy and the one in the previous iteration is bounded. Therefore, one can in practice use the previous best policy as a proxy to verify if there is a consensus among the most likely models on the best policy in the next iteration. This is justified by the fact that the updated policy is not too different from the previous one. Thus, model identification is stopped whenever the most likely models predict almost the same value for the previously computed policy, i.e., if all the high-probability models predict a similar value for the previous policy, then any of these models could be used for searching for the next policy.

Empirical evaluation performed in simulation and on a real robot show the benefits of the framework. The initial set of experiments are performed in the OpenAI Gym [Brockman *et al.*, 2016] with the MuJoCo simulator[1]. They demonstrate that the proposed model identification approach is more time-efficient than alternatives, and it improves the data-efficiency of policy search algorithms, such as TRPO. The second part is performed on a real-robot manipulation task. It shows that learning using a simulator with the identified model can be more data-efficient than other model-based methods, such as PILCO, and model-free ones, such as PoWER.

2 Related Work

The proposed method relates to *model-based RL*, where a dynamical system is physically simulated. Model-based learning involves explicitly learning the unknown dynamics of the system, and search for an optimal policy accordingly. Variants have been applied in robotics [Dogar *et al.*, 2012; Lynch and Mason, 1996; Scholz *et al.*, 2014; Zhou *et al.*, 2016], where it has been shown that using inaccurate models can still allow a policy to steer an RC car [Abbeel *et al.*, 2006]. For general-purpose model-based RL, the Probabilistic Inference for Learning COntrol (PILCO) algorithm has been shown to be efficient in utilizing a small amount of data to learn dynamical models and optimal policies [Deisenroth *et al.*, 2011]. Replacing the gradient-based optimization algorithm with a parallel, black-box algorithm (Covariance Matrix Adaptation (CMA)) [Hansen, 2006], Black-DROPS can be as data-efficient as PILCO without imposing any constraint on the reward function or the policy [Chatzilygeroudis *et al.*, 2017]. CMA was also used for automatically designing open-loop reference trajectories [Tan *et al.*, 2016]. Trajectory optimization was performed in a physical simulator before real-world execution. Real robot trajectories were used to optimize the simulator parameters.

The proposed method also relates to *learning from simulation*. Utilizing simulation to learn a prior before learning on the robot is a common way to reduce real-world data needs for policy search and provides data-efficiency over PILCO [Cutler and How, 2015]. Policy Improvement with REsidual Model learning (PI-REM) [Saveriano *et al.*, 2017] has shown improvement over PILCO by utilizing a simulator and only modeling the residual dynamics between the simulator and reality. Similarly, using data from simulation as the mean function of a Gaussian Process (GP) and combining with Iterative Linear Quadratic Gaussian Control (ILQG), GP-ILQG is able to improve incorrect models and generate robust controllers [Lee *et al.*, 2017].

In addition, the proposed method utilizes *Bayesian optimization (BO)* to efficiently identify the model. For direct policy search, a prior can also be selected from a set of candidates using BO [Pautrat *et al.*, 2017]. BO can also be used to optimize the policy while balancing the trade-off between simulation and real data [Marco *et al.*, 2017], or learn a locally linear dynamical model, while aiming to maximize control performance [Bansal *et al.*, 2017].

Traditional system identification builds a dynamics model by minimizing prediction error (e.g., using least squares) [Swevers *et al.*, 1997]. There have been attempts to combine parametric rigid body dynamics model with nonparametric model learning for approximating the inverse dynamics [Nguyen-Tuong and Peters, 2010]. In contrast to these methods, this work uses a physics engine, and concentrates on identifying model parameters instead of learning the models from scratch. Optimizing the simulated model parameters to match actual data collected on the robot can also help model robot damage [Bongard *et al.*, 2006]. Recent work performed in simulation proposed model identification for predicting physical parameters, such as mass and friction [Yu *et al.*, 2017].

Different from model-based methods, *model-free methods* search for a policy that best solves the task without explicitly learning the system's dynamics [Sutton and Barto, 1998; Kober *et al.*, 2013]. They have been shown effective in robotic tasks [Peters *et al.*, 2010]. Policy learning by Weighting Exploration with the Returns (PoWER) [Kober and Peters, 2009] is a model-free policy search approach widely used for learning motor skills. The TRPO algorithm [Schulman *et al.*, 2015] has been demonstrated in some MuJoCo simulator environments. *End-to-end learning* is an increasingly popular framework [Agrawal *et al.*, 2016; Wu *et al.*, 2015; Byravan and Fox, 2017; Finn and Levine,], which involves successful demonstrations of physical interaction and learning a direct mapping of the sensing input to controls. These approaches usually require many physical experiments to effectively learn. Overall, model-free methods tend to require a lot of training data and can jeopardize robot safety.

A *contribution* of this work is to link model identification with policy search. It does not search for the most accurate model within a time budget but for an accurate enough model, given an easily computable criterion, i.e., predicting a policy's value function that is not too different from the searched

[1]MuJoCo: `www.mujoco.org`

policy. This idea can be used with many policy search algorithms that allow for smooth changes during learning.

3 Proposed Approach

This section provides first a system overview and then the main algorithm, focusing on model identification.

3.1 System Overview and Notations

The focus is on identifying physical properties of robots or objects, e.g. mass and friction, using a simulator. These physical properties are concatenated in a single vector and represented as a D-dimensional vector $\theta \in \Theta$, where Θ is the space of possible physical parameters. Θ is discretized with a regular grid resolution. The proposed approach returns a distribution P on a discretized Θ instead of a single point $\theta \in \Theta$ given the challenge of perfect model identification. In other terms, there may be multiple models that can explain an observed movement of the robot with similar accuracy. The approach is to preserve all possible explanations and their probabilities.

The online model identification takes as input a prior distribution P_t, for time-step $t \geq 0$, on the discretized parameter space Θ. P_t is calculated based on the initial distribution P_0 and a sequence of observations $(x_0, \mu_0, x_1, \mu_1, \ldots, x_{t-1}, \mu_{t-1}, x_t)$. For instance, x_t can be the state of the robot at time t and μ_t is a vector describing a vector of torques applied to the robot at time t. Applying μ_t results in changing the robot's state from x_t to x_{t+1}. The algorithm returns a distribution P_{t+1} on the models Θ.

The robot's task is specified by a reward function R that maps state-actions (x, μ) into real numbers. A policy π returns an action $\mu = \pi(x)$ for state x. The value $V^{\pi}(\theta)$ of policy π given model θ is defined as $V^{\pi}(\theta) = \sum_{t=0}^{H} R(x_t, \mu_t)$, where H is a fixed horizon, x_0 is a given starting state, and $x_{t+1} = f(x_t, \mu_t, \theta)$ is the predicted state at time $t+1$ after simulating action μ_t in state x_t given parameters θ. For simplicity, the focus is on systems with deterministic dynamics.

3.2 Main Algorithm

Given a reward function R and a simulator with model parameters θ, there are many ways to search for a policy π that maximizes value $V^{\pi}(\theta)$. For example, one can use Differential Dynamic Programming (DDP), Monte Carlo (MC) methods, or if a good policy cannot be found with alternatives, execute a model-free RL algorithm on the simulator. The choice of a particular policy search method is open and depends on the task. The main loop of the system is presented in Algorithm 1. This meta-algorithm consists in repeating three main steps: (1) data collection using the real robot, (2) model identification using a simulator, and (3) policy search in simulation using the best identified model.

```
t ← 0;
Initialize distribution P over Θ to a uniform distribution;
Initialize policy π;
repeat
    Execute policy π for H iterations on the real robot, and
      collect new state-action-state data {(x_i, μ_i, x_{i+1})} for
      i = t, ..., t + H − 1;
    t ← t + H;
    Run Algorithm 2 with collected state-action-state data
      and reference policy π for updating distribution P;
    Initialize a policy search algorithm (e.g. TRPO) with π
      and run the algorithm in the simulator with the model
      arg max_{θ∈Θ} P(θ) to find an improved policy π′;
    π ← π′;
until Timeout;
```

Algorithm 1: Main Loop

3.3 Value-Guided Model Identification

The process, explained in Algorithm 2, consists of simulating the effects of actions μ_i on the robot in states x_i under various values of parameters θ and observing the resulting states $\hat{x}_{i+1}$, for $i = 0, \ldots, t$. The goal is to identify the model parameters that make the outcomes of the simulation as close as possible to the real observed outcomes. In other terms, the following black-box optimization problem is solved:

$$\theta^* = \arg\min_{\theta \in \Theta} E(\theta) \overset{def}{=} \sum_{i=0}^{t} \|x_{i+1} - f(x_i, \mu_i, \theta)\|_2, \qquad (1)$$

wherein x_i and x_{i+1} are the observed states of the robot at times i and $i+1$, μ_i is the action that moved the robot from x_i to x_{i+1}, and $f(x_i, \mu_i, \theta) = \hat{x}_{i+1}$, the predicted state at time $i+1$ after simulating action μ_i in state x_i using θ.

High-fidelity simulations are computationally expensive. It is therefore important to minimize the number of simulations, i.e., evaluations of function E, while searching for the optimal parameters that solve Eq. 1. This is solved by using the *Entropy Search* technique [Hennig and Schuler, 2012]. This method is well-suited because it explicitly maintains a belief on the optimal parameters. This is unlike other Bayesian optimization methods, such as *Expected Improvement*, which only maintain a belief on the objective function. The following description explains how this technique is adapted, and shows why keeping a distribution on all models is needed for deciding when to stop the optimization.

The error function E does not have an analytical form, it is gradually learned from a sequence of simulations with a small number of parameters $\theta_k \in \Theta$. To choose these parameters efficiently in a way that quickly leads to accurate parameter estimation, a belief about the actual error function is maintained. This belief is a probability measure over the space of all functions $E : \mathbb{R}^D \to \mathbb{R}$, and is represented by a Gaussian Process (GP) [Rasmussen and Williams, 2005] with mean vector m and covariance matrix K. The mean m and covariance K of the GP are learned from data points $\{(\theta_0, E(\theta_0)), \ldots, (\theta_k, E(\theta_k))\}$, where θ_k is a selected vector of physical properties of the robot, and $E(\theta_k)$ is the accumulated distance between actual observed states and states that are obtained from simulation using θ_k.

The probability distribution P on the identity of the best physical model θ^*, returned by the algorithm, is computed

Input: state-action-state data $\{(x_i, \mu_i, x_{i+1})\}$ for $i = 0, \ldots, t$
a discretized space of possible values of physical properties Θ,
a reference policy π,
minimum and maximum number of evaluated models k_{min}, k_{max},
model confidence threshold η,
value error threshold ε ;
Output: probability distribution P over Θ;
Sample $\theta_0 \sim \text{Uniform}(\Theta)$; $L \leftarrow \emptyset$; $k \leftarrow 0$; $stop \leftarrow false$;
repeat
  `// Calculating the accuracy of model` θ_k
  $l_k \leftarrow 0$;
  for $i = 0$ **to** t **do**
    Simulate $\{(x_i, \mu_i)\}$ using a physics engine with physical parameters θ_k and get the predicted next state $\hat{x}_{i+1} = f(x_i, \mu_i, \theta_k)$;
    $l_k \leftarrow l_k + \|\hat{x}_{i+1} - x_{i+1}\|_2$;
  end
  $L \leftarrow L \cup \{(\theta_k, l_k)\}$;
  Calculate $GP(m, K)$ on error function E, where $E(\theta) = l$, using data $(\theta, l) \in L$;
  `// Monte Carlo sampling`
  Sample $E_1, E_2, \ldots, E_n \sim GP(m, K)$ in Θ;
  foreach $\theta \in \Theta$ **do**

$$P(\theta) \approx \frac{1}{n} \sum_{j=0}^{n} \mathbf{1}_{\theta = \arg\min_{\theta' \in \Theta} E_j(\theta')} \tag{2}$$

  end
  `// Selecting the next model to evaluate`
  $\theta_{k+1} = \arg\min_{\theta \in \Theta} P(\theta) \log\big(P(\theta)\big)$;
  $k \leftarrow k + 1$;
  `// Checking the stopping condition`
  if $k \geq k_{min}$ **then**
    $\theta^* \leftarrow \arg\max_{\theta \in \Theta} P(\theta)$;
    Calculate the values $V^\pi(\theta)$ with all models θ that have a probability $P(\theta) \geq \eta$ by using the physics engine for simulating trajectories with models θ ;
    if $\sum_{\theta, \forall P(\theta) \geq \eta} P(\theta) |V^\pi(\theta) - V^\pi(\theta^*)| \leq \varepsilon$ **then**
      $stop \leftarrow true$;
    end
  end
  if $k = k_{max}$ **then**
    $stop \leftarrow true$;
  end
until *stop = true*;

Algorithm 2: Value-Guided Model Identification

from the learned GP as

$$\begin{aligned} P(\theta) &\stackrel{def}{=} P\big(\theta = \arg\min_{\theta' \in \Theta} E(\theta')\big) \\ &= \int_{E:\mathbb{R}^D \to \mathbb{R}} p_{m,K}(E) \Pi_{\theta' \in \Theta - \{\theta\}} H\big(E(\theta') - E(\theta)\big) \mathrm{d}E \end{aligned} \tag{3}$$

where H is the Heaviside step function, i.e., $H\big(E(\theta') - E(\theta)\big) = 1$ if $E(\theta') \geq E(\theta)$ and $H\big(E(\theta') - E(\theta)\big) = 0$ otherwise, and $p_{m,K}(E)$ is the probability of E according to the learned GP mean m and covariance K. Intuitively, $P(\theta)$ is the expected number of times that θ happens to be the minimizer of E when E is a function distributed according to the GP.

The distribution P from Eq. 3 does not have a closed-form expression. Therefore, *Monte Carlo* (MC) sampling is employed for estimating P. The process samples vectors $[E(\theta')]_{\theta' \in \Theta}$ containing values that E could take, according to the learned GP, in the discretized space Θ. Then $P(\theta)$ is estimated by counting the ratio of sampled vectors of the values of simulation error E where θ happens to make the lowest error, as indicated in Eq. 2 in Alg. 2.

Finally, the computed distribution P is used to select the next vector θ_{k+1} to use as a physical model in the simulator. This process is repeated until the entropy of P drops below a certain threshold, or until the algorithm runs out of the allocated time budget. The entropy of P is given as $\sum_{\theta \in \Theta} -P_{min}(\theta) \log\big(P_{min}(\theta)\big)$. When the entropy of P is close to zero, the mass of distribution P is concentrated around a single vector θ, corresponding to the physical model that best explains the observations. Hence, next θ_{k+1} should be selected so that the entropy of P would decrease after adding the data point $\big(\theta_{k+1}, E(\theta_{k+1})\big)\}$ to train the GP and re-estimate P using the new mean m and covariance K in Eq. 3.

Entropy Search methods follow this reasoning and use MC again to sample, for each potential choice of θ_{k+1}, a number of values that $E(\theta_{k+1})$ could take according to the GP in order to estimate the expected change in the entropy of P and choose the parameter vector θ_{k+1} that is expected to decrease the entropy of P the most. The existence of a secondary nested process of MC sampling makes this method impractical for online optimization. Instead, this work presents a simple heuristic for choosing the next θ_{k+1}. In this method, called *Greedy Entropy Search*, the next θ_{k+1} is chosen as the point that contributes the most to the entropy of P,

$$\theta_{k+1} = \arg\max_{\theta \in \Theta} -P(\theta) \log\big(P(\theta)\big). \tag{4}$$

This selection criterion is greedy because it does not anticipate how the output of the simulation using θ_{k+1} would affect the entropy of P. Nevertheless, this criterion selects the point that is causing the entropy of P to be high. That is a point θ_{k+1} with a good chance $P(\theta_{k+1})$ of being the real model, but also with a high uncertainty $P(\theta_{k+1}) \log\big(\frac{1}{P(\theta_{k+1})}\big)$. Initial experiments suggested that this heuristic version of Entropy Search is more practical than the original Entropy Search method because of the computationally expensive nested MC sampling loops used in the original method. The actual sampled values are not restricted to lie on the discretized grid. Once a grid value is selected as the candidate with the highest entropy, a local optimization process using L-BFGS takes place to further optimize the parameter value.

The stopping condition of Alg. 2 depends on the predicted value of a reference policy π. The reference policy is one that will be used in the main algorithm (Alg. 1) as a starting point in the policy search with the identified model. That is also the policy executed in the previous round of the main algorithm. Many policy search algorithms (such as REPS and TRPO) guarantee that the KL divergence between consecutive policies π and π' is minimal. Therefore, if the difference

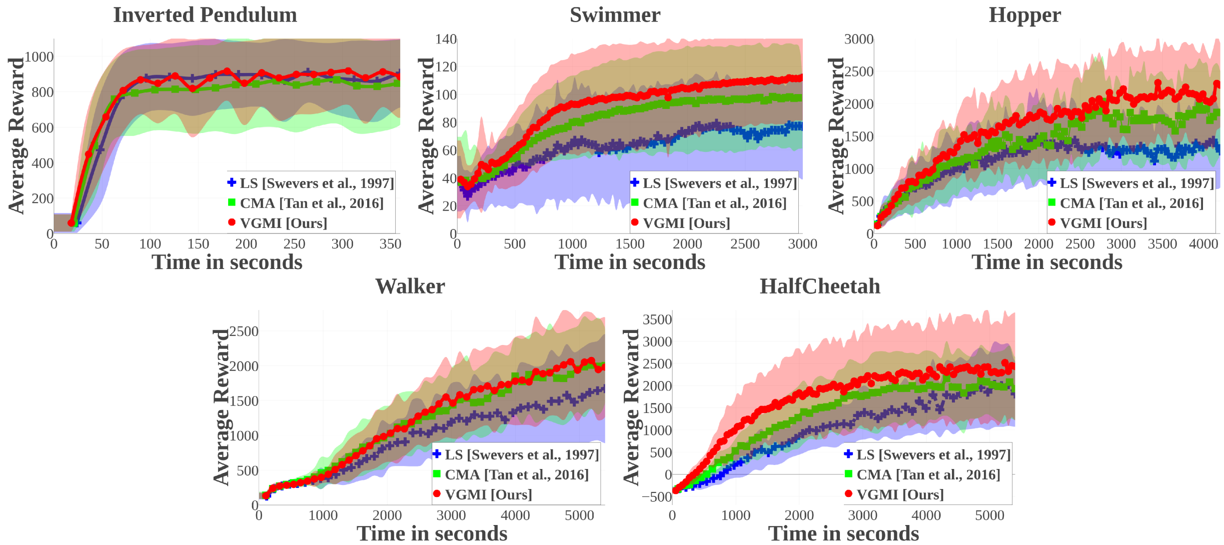

Figure 2: Average reward as a function of total time, including both model identification and policy search. (Best viewed in color.)

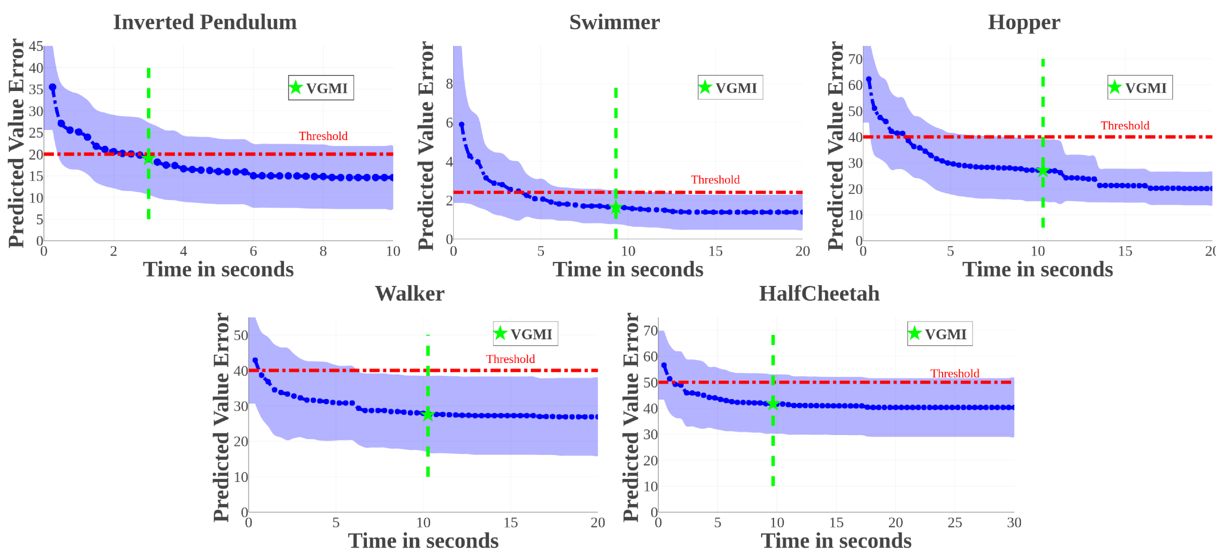

Figure 3: The error in predicting the value of the next policy ($|V^{\pi'}(\theta) - V^{\pi'}(\theta^*)|$) as a function of the time spent in the model identification process. When VGMI decides to stop (green line), the actual value error is indeed always below threshold used in its stopping criterion.

$|V^{\pi}(\theta) - V^{\pi}(\theta^*)|$ for two given models θ and θ^* is smaller than a threshold ε, then the difference $|V^{\pi'}(\theta) - V^{\pi'}(\theta^*)|$ should also be smaller than a threshold that is a function of ε and $KL(\pi \| \pi')$. A full proof of this conjecture is the subject of an upcoming work. In practice, this means that if θ and θ^* are two models with high probabilities, and $|V^{\pi}(\theta) - V^{\pi}(\theta^*)| \leq \varepsilon$ then there is no point in continuing the optimization to find out which one of the two models is actually the most accurate because both models will result in similar policies. The same argument could be used when there are more than two models with high probabilities.

4 Experimental Results

VGMI is evaluated both in simulation and on a real robot.

4.1 Experiments on RL Benchmarks in Simulation

Setup: The simulation experiments are performed in OpenAI Gym with the MuJoCo simulator. The space of unknown physical models θ is:

Inverted Pendulum (IP): A pendulum is connected to a cart, which moves linearly. The dimensionality of Θ is two, one for the mass of the pendulum and one for the cart.

Swimmer: The swimmer is a 3-link planar robot. Θ has three dimensions, one for the mass of each link.

Hopper: The hopper is a 4-link planar mono-pod robot. Thus, the dimensionality of Θ is four.

Walker2D: The walker is a 7-link planar biped robot. Thus, the dimensionality of Θ is seven.

HalfCheetah: The halfcheetah is a 7-link planar cheetah robot. The dimensionality of the space Θ is seven.

For each environment, the simulator is initiated with the ground truth mass, which is only used for rollouts to generate data for model identification. To create inaccurate simulators as prior models, the mass was randomly increased or decreased by ten to fifty percent. All the policies are trained with TRPO implemented in rllab [Duan *et al.*, 2016]. The policy network has 2 hidden layers with 32 neurons each.

VGMI is compared against a) Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Tan *et al.*, 2016] and b) Least Square (LS) optimization using L-BFGS-B algorithm [Swevers *et al.*, 1997]. All of them optimize the objective function of Eq. 1.

The number of iterations for policy search varies on problem difficulty. The main loop in Alg. 1 is executed for 20 iterations for IP, 100 iterations for Swimmer, 100 iterations for Hopper, 200 iterations for Walker2D and 400 iterations for HalfCheetah. VGMI follows Alg. 2 and is executed every 10 iterations, i.e., $H = 10$ in Alg. 1. The same iterations are used for CMA and LS and they are also given the same time budget for model identification. All the results are the mean and stand deviation of 20 independent trials.

Results: Performance is reported for total training time, i.e., including both model identification and policy search.

Fig. 2 shows the cumulative reward per trajectory on the ground truth system as a function of the total time. VGMI always achieves the highest reward across 5 benchmarks and earlier than CMA and LS. CMA performs better than LS, though relies on a large number of trajectories that needs to be physically simulated.

Fig. 3 shows that VGMI automatically stops the process without manually setting a time budget. The red line is the error threshold ε in the predicted value function of VGMI. The green line is where VGMI stops based on ε and Algorithm's 2 criterion. The same algorithm was also executed with increasing pre-set time budgets (x-axis). The reality gap in predicting the value of the best policy was empirically estimated using the best-identified and the ground truth model. This ensures that VGMI does not stop prematurely. When it stops (green line), the error $|V^{\pi'}(\theta) - V^{\pi'}(\theta^*)|$ (blue curve) is indeed below the predefined threshold ε, i.e., further tuning the identified model quickly does not provide many benefits past the stopping point.

Fig. 4 shows that the adaptive stopping criterion of VGMI performs better than different fixed time budgets for model identification. Thus, VGMI with the proposed stopping criterion effectively balances the trade-off between model identification and policy search.

Fig. 5 demonstrates that Greedy Entropy Search (GES) in the context of VGMI results in improved convergence relative to the original Entropy Search approach.

4.2 Real Robot Pushing Experiments

Setup: Consider Figs. 6 and 7, where the (*Motoman*) robot assists (*Baxter*) to pick up a bottle. The object is known but can be reached only by Motoman and not by Baxter. The intersection of the robots' reachable workspace is empty and Motoman must learn an action to push the bottle 1m away inside Baxter's reachability region. If the robot simply executes

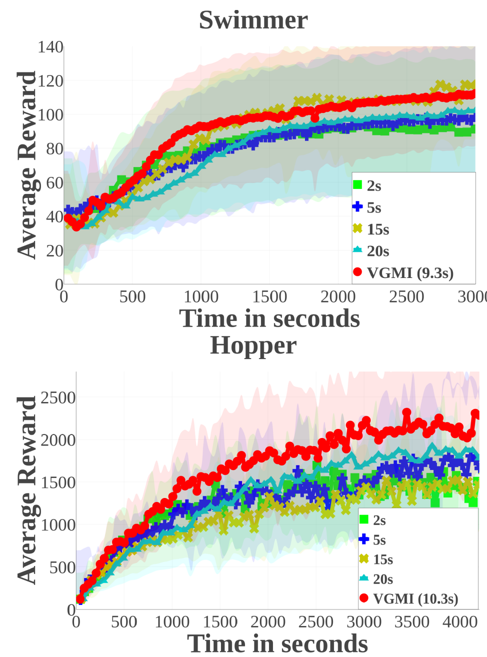

Figure 4: For a fixed total time budget for both model identification and policy search, VGMI perfoms better than manully selected time budgets for model identification. (Best viewed in color.)

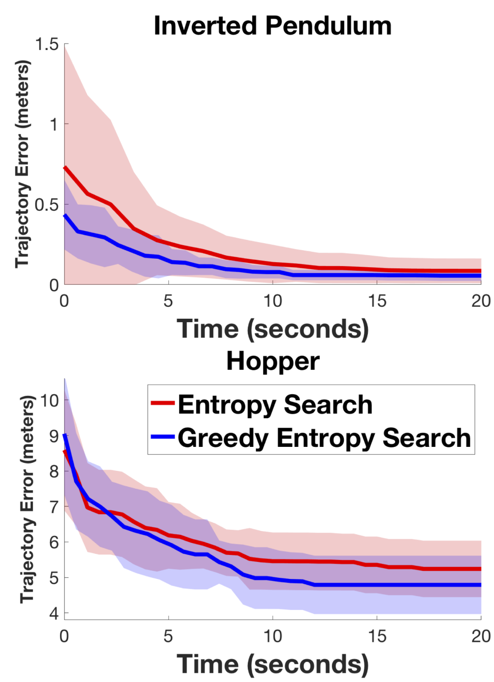

Figure 5: Model identification in Inverted Pendulum and Hopper environment using two variants of Entropy Search.

the maximum velocity push on the object, the object falls off the table. Similarly, if the object is rolled too slowly, it can get stuck in the region between the two robots' workspaces.

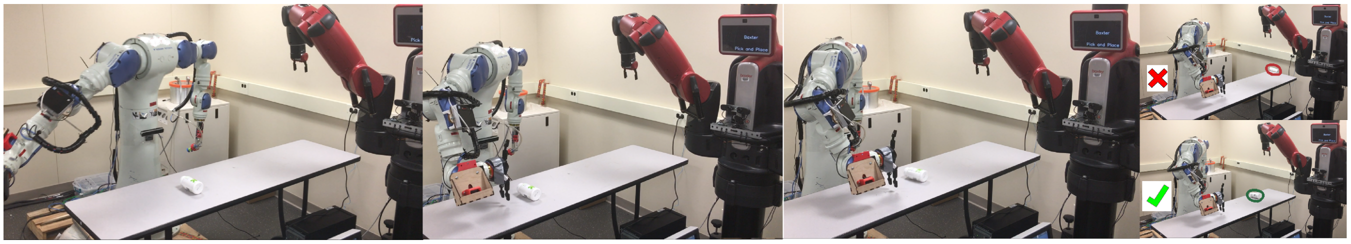

Figure 6: Experiment where the *Motoman* pushes the object into Baxter's workspace after identifying the problem's physical parameters.

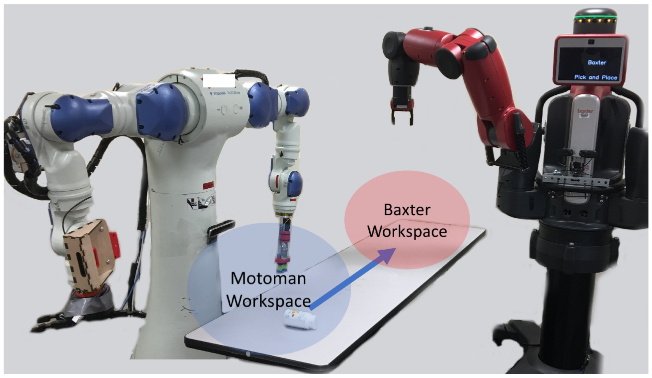

Figure 7: Baxter needs to pick up the bottle but cannot reach it, while the Motoman can. Motoman gently pushes the object locally without losing it and identifies the object's parameters from observed motions and a physics engine. The object is then pushed to Baxter using a policy learned in simulation with the identified parameters.

Both outcomes are undesirable as they would require human intervention to reset the scene.

The goal is to find an optimal policy with parameter η representing the robot hand's pushing velocity. The pushing direction is towards the target and the hand pushes the object at its geometric center. No human effort was needed to reset the scene since the velocity and pushing direction were controlled so the object is always in Motoman's workspace. Specifically, a pushing velocity limit is set. The approach iteratively searches for the best η by uniformly sampling 20 different velocities in simulation, and identifies the object model parameters θ^* (the mass and friction coefficient) using trajectories from rollouts by running VGMI as in Alg. 2. VGMI is executed after each rollout, i.e., $H = 1$ in Alg. 1. The method is compared to PoWER [Kober and Peters, 2009] and PILCO [Deisenroth *et al.*, 2011]. For PoWER, the reward function is $r = e^{-dist}$, where *dist* is the distance between the object position after pushing and the desired target. For PILCO, the state space is the 3D object position.

Results: Two metrics are used for evaluating performance: 1) The distance between the final object location after being pushed and the desired goal location; 2) The number of times the object falls off the table. A video of these experiments can be found on `https://goo.gl/Rv4CDa`.

Fig. 8 shows that in the real-world experiments the method achieves both lower final object location error and fewer number of object drops. The reduction in object drops is important for autonomous robot learning as it minimizes human effort. The model-free approach PoWER results in higher location error and more object drops. PILCO performs better than PoWER as it also learns a dynamical model together with the policy but the model is not as accurate as the VGMI one. Since simple policy search is used for VGMI, the performance is expected to be better is more advanced policy search methods, such as combining PoWER with VGMI.

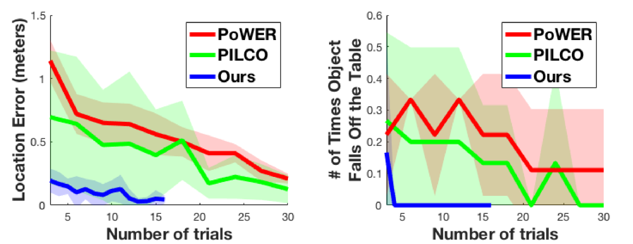

Figure 8: Pushing policy optimization results using a *Motoman* robot. VGMI achieves both lower final object location error and fewer object drops comparing to alternatives. (Best viewed in color)

5 Conclusion

This paper presents a practical approach that integrates physics engines and Bayesian optimization for model identification to increase data efficiency of RL. It identifies a model that is good enough to predict a policy's value function that is similar to the current optimal policy. The approach can be used with any policy search algorithm that guarantees smooth changes in the learned policy. It can also help the real-world applicability of motion planners that reason over dynamics and operate with physics engines [Bekris and Kavraki, 2008; Li *et al.*, 2016; Littlefield *et al.*, 2017]. Both simulated and real robot experiments show that VGMI can decrease the number of rollouts needed to learn an optimal policy. Future work includes analyzing the properties of the method, such as expressing the conditions under which the inclusion of the model identification reduces the need for physical rollouts and the speed-up in convergence in terms of physical rollouts. It is also interesting to apply the method to alternative physical tasks.

Acknowledgments

This work was partially sponsored by NSF IIS-1734492 and IIS-1723869, a NASA ECF'15 award to Kostas Bekris and a U.S. Army Research Lab Collaborative Agreement # W911NF-10-2-0016.

References

[Abbeel *et al.*, 2006] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proc. of ICML*. ACM, 2006.

[Agrawal *et al.*, 2016] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *NIPS*, 2016.

[Bansal *et al.*, 2017] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomlin. Goal-driven dynamics learning via bayesian optimization. In *IEEE CDC*, 2017.

[Bekris and Kavraki, 2008] K. E. Bekris and L. E. Kavraki. Informed and Probabilistically Complete Search for Motion Planning under Differential Constraints. In *STAIR*, 2008.

[Bongard *et al.*, 2006] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv:1606.01540*, 2016.

[Byravan and Fox, 2017] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *ICRA*. IEEE, 2017.

[Chatzilygeroudis *et al.*, 2017] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-Box Data-efficient Policy Search for Robotics. In *IROS*, September 2017.

[Cutler and How, 2015] Mark Cutler and Jonathan P How. Efficient reinforcement learning for robots using informative simulated priors. In *ICRA*. IEEE, 2015.

[Deisenroth *et al.*, 2011] M. Deisenroth, C. Rasmussen, and D. Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *R:SS*, 2011.

[Dogar *et al.*, 2012] Mehmet Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-Based Grasp Planning Through Clutter. In *R:SS*, July 2012.

[Duan *et al.*, 2016] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

[Erez *et al.*, 2015] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ODE and physx. In *IEEE ICRA*, 2015.

[Finn and Levine,] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *ICRA 2017*.

[Hansen, 2006] Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

[Hennig and Schuler, 2012] Philipp Hennig and Christian J. Schuler. Entropy Search for Information-Efficient Global Optimization. *JMLR*, 13:1809–1837, 2012.

[Kober and Peters, 2009] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *NIPS*, 2009.

[Kober *et al.*, 2013] J. Kober, J. Andrew (Drew) Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *IJRR*, 2013.

[Lee *et al.*, 2017] Gilwoo Lee, Siddhartha S Srinivasa, and Matthew T Mason. Gp-ilqg: Data-driven robust optimal control for uncertain nonlinear dynamical systems. *arXiv preprint arXiv:1705.05344*, 2017.

[Li *et al.*, 2016] Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically Optimal Sampling-based Kinodynamic Planning. In *IJRR*, volume 35, 2016.

[Littlefield *et al.*, 2017] Z. Littlefield, D. Surovik, W. Wang, and K. E. Bekris. From Quasi-Static To Kinodynamic Planning For Spherical Tensegrity Locomotion. In *ISRR*, 2017.

[Lynch and Mason, 1996] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, control- lability, and planning. *International Journal of Robotics Research*, 18, 1996.

[Marco *et al.*, 2017] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P. Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *ICRA*, pages 1557–1563, 2017.

[Nguyen-Tuong and Peters, 2010] Duy Nguyen-Tuong and Jan Peters. Using model knowledge for learning inverse dynamics. In *ICRA*, pages 2677–2682, 2010.

[Pautrat *et al.*, 2017] Rémi Pautrat, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Bayesian optimization with automatic prior selection for data-efficient direct policy search. *arXiv preprint arXiv:1709.06919*, 2017.

[Peters *et al.*, 2010] Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *AAAI*, 2010.

[Rasmussen and Williams, 2005] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[Saveriano *et al.*, 2017] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. Data-efficient control policy search using residual dynamics learning. In *Proc. of IROS*, 2017.

[Scholz *et al.*, 2014] Jonathan Scholz, Martin Levihn, Charles L. Isbell, and David Wingate. A Physics-Based Model Prior for Object-Oriented MDPs. In *ICML*, 2014.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[Swevers *et al.*, 1997] Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. Optimal robot excitation and identification. *IEEE TRO-A*, 13(5):730–740, 1997.

[Tan *et al.*, 2016] Jie Tan, Zhaoming Xie, Byron Boots, and C Karen Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *IROS*. IEEE, 2016.

[Wu *et al.*, 2015] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *NIPS*, pages 127–135, 2015.

[Yu *et al.*, 2017] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Proceedings of R:SS*, July 2017.

[Zhou *et al.*, 2016] Jiaji Zhou, Robert Paolini, J. Andrew Bagnell, and Matthew T. Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *ICRA*, pages 372–377, 2016.