Journal of Machine Learning Research 1 (2018) 1-48 Submitted 4/18; Published 10/00

Towards Closing the Reality Gap of Robotic Simulators through Task-oriented Bayesian Optimization

Shaojun Zhu SHAOJUN.ZHU@CS.RUTGERS.EDU
David Surovik DAVID.SUROVIK@CS.RUTGERS.EDU
Kostas E. Bekris KOSTAS.BEKRIS@CS.RUTGERS.EDU
Abdeslam Boularias ABDESLAM.BOULARIAS@CS.RUTGERS.EDU
Department of Computer Science
Rutgers University
Piscatway, NJ 08854-8019, USA

Editor: Roberto Calandra, Roman Garnett, Javier González, Frank Hutter and Bobak Shahriari

Abstract

Although robotic simulators continue to increase in sophistication, their real-world deployment for robot control and learning remains limited. This is primarily due to insufficient accuracy in the context of complex physical interactions. This work aims to close the reality gap of off-the-shelf physics engines for robotic tasks via three interrelated contributions. First, Bayesian optimization is adapted for time-efficient identification of the mechanical parameters of robots or objects they interact with given real-world trajectories. Examples of such physical parameters include the mass of objects in the environment and friction coefficients of surfaces. The second contribution relates to the daunting model complexity of modern and versatile robotic systems. This issue is approached by projecting the model identification challenge into an appropriate lower dimensional parameter space through the use of an autoencoder network. Finally, the Bayesian optimization process is integrated with policy learning in simulation to ensure that the model captures the intended closed-loop behavior within a desired level of confidence. This ensures that the results remain effective for real-world deployment, while avoiding a waste of effort on irrelevant aspects of the model. Hardware experiments for a simple robotic manipulation task show that the proposed framework achieves good data-efficiency against model-free policy search strategies. Furthermore, the proposed model identification process is applied on a simulated locomotion task involving a high-dimensional, compliant tensegrity robot. The proposed approach results in more accurate parameter identification than alternatives given the same time budget and increases the precision of locomotion control. Experiments in simulation also indicate that the proposed integration of policy search with the Bayesian model identification improves the time-efficiency of model-based reinforcement learning.

Keywords: Model Identification, Bayesian Optimization, Policy Search

1. Introduction

Physics engines are an important tool for various fields ranging from computer graphics and video games, to aeronautical, bio-mechanical, and automotive industries. They are also quickly gaining popularity in robotics for testing and validation of control strategies in advance of more costly and risky deployment on a real robot. In a typical robotic application, a physics engine receives as input

©2018 Shaojun Zhu, David Surovik, Kostas E. Bekris and Abdeslam Boularias.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v1/zhu18a.html.

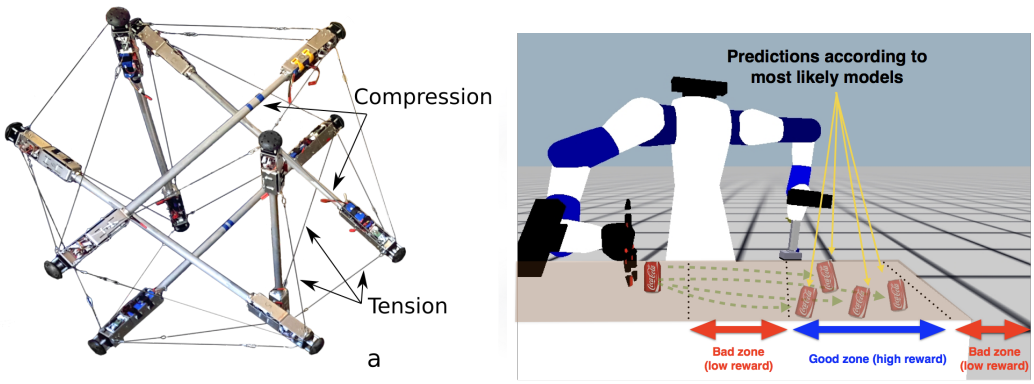

Figure 1: Left: NASA Tensegrity robot prototype (SUPERball), a robotic icosahedron with 6 rods and 24 cables (Sabelhaus et al., 2015). Right: Example of a stopping criterion for model identification in a manipulation task (pushing an unknown object). If all high-probability model parameters predict a high reward for an action then there is no point in identifying more accurate tunings.

mechanical and mesh models of the robot and objects in the robot's surroundings. Given control forces or robot torques, the engine returns the poses and velocities of the robot and the objects.

As manually-tuned controllers are typically insufficient for complex systems, learning from experience is a promising direction for robot manipulation and locomotion. Specifically, reinforcement learning (RL) provides an elegant framework for enabling robots to acquire new skills from trial and error and with minimal human effort. Current RL algorithms, however, generally require large amounts of training data to achieve reasonable performance. Acquiring this data is a time-consuming process that also accelerates wear and tear on robotic hardware. For this reason, learning in simulation using physics engines is comparatively a cheaper and faster process.

Discrepancies between a physical model and the real system, however, often result in poor effectiveness of policies learned in simulation. This problem is known as *the reality gap*. The accuracy of a physics engine depends upon several factors. For instance, the model assumed by the engine, such as Coulomb's law of friction, is a simplification of the underlying real-world phenomena. The numerical algorithm for solving the differential equations of motion may introduce additional errors. Moreover, the values of the robot's and the objects' mechanical parameters, such as mass, friction, and elasticity, may be inaccurate. Despite recent and ongoing progress [1], simulators inevitably can only approximate real systems, while sacrificing accuracy in order to limit computational costs. Compliant components, such as those of the tensegrity robot shown in Figure 1, and contact-related tasks, such as object manipulation, make precise and accurate modeling even more difficult to achieve.

The challenges in building accurate models have been studied extensively in classical system identification (Ljung, 2010), which usually assumes some parametrized form of an analytical model function and estimates its parameters by minimizing an empirical prediction error. Nevertheless, analytical models can be overly simplistic and often fail to capture the true complex dynamics of a robotic system. This paper revisits this classic problem in the new context of modern physics engines and robotic systems.

Contribution 1: Figure 2 presents the high-level framework for model identification, which uses off-the-shelf physics engines and fine-tunes the mechanical parameters of a given model to perform a given robotic task. Real trajectories are generated by controlling the robot according to

1. Example engines: Bullet: www.bulletphysics.org, DART: http://dartsim.github.io, Havok: www.havok.com, MuJoCo: www.mujoco.org, NASA tensegrity robotics toolkit: https://ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/NTRT/, PhysX: www.geforce.com/hardware/technology/physx

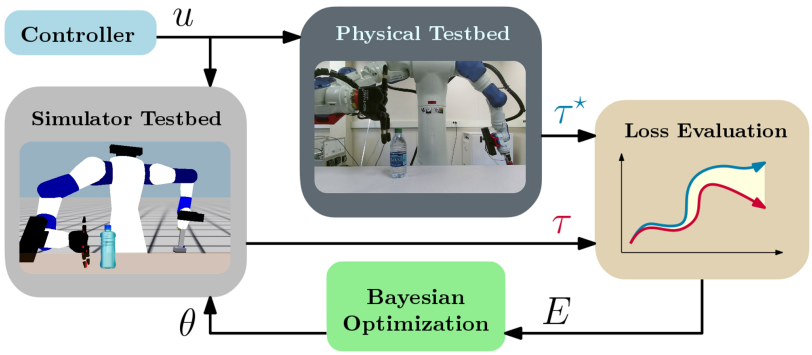

Figure 2: Model identification with Bayesian Optimization: applying control commands u from a controller to a physical testbed (e.g., a real robot), ground-truth trajectories τ^* can be collected. Applying the same control commands to a simulator testbed (e.g., a physics-based robot simulator), an error E can be computed between τ^* and the simulated trajectories τ. Bayesian optimization is applied to search for the optimal tuning θ^*, which minimizes E.

a certain policy and recorded. Then, the approach searches for the best parameter vector of the model so that the simulated trajectories match reality as closely as possible. This work assumes that the structure of the dynamics model as well as the object and robot geometries are known a priori. For example, in the experiment illustrated in Figure 2, the shape of the pushed bottle and the differential equation of its motion are both known for a given model of the problem. The parameters of the model that are not known in this example are the bottle's mass and the corresponding friction coefficients. In both cases, these parameters are assumed to be homogenous. Given these two parameters and a manipulation action, a physics engine can solve the equation of motion to simulate the object's trajectory. We refer to the vector of values, such as the mass and the friction in the example, as the model parameter vector. In general, a model parameter vector contains any unknown mechanical properties of the object. A specific assignment of values to the model parameters is referred to as a *tuning*. The objective is to achieve a tuning of the model parameter vector that results in simulations that are close enough to reality and allow the completion of target tasks.

Tuning the model parameter vector corresponds to searching in the space of values for the corresponding parameters. This search is performed through an "anytime black-box Bayesian Optimization" (BO) approach, where a probability distribution (belief) on the optimal parameter values is repeatedly updated. The term *black-box* refers to optimization methods where the analytical form of the objective function and its derivative are not available. Only values of the objective function at various points can be accessed. This work adopts this strategy, given that most current physics engines do not provide the derivatives of the predicted motions. The *anytime* property indicates that the algorithm keeps searching for a better solution — i.e., the parameter values with the highest probability — until the computation time exceeds a predefined budget.

While the proposed approach is in line with traditional system identification and model-based RL, the parametrizations identified here are those of the physics engines, which are more expressive and accurate than analytical equations of motion. Gaussian Processes (GPs) are widely used in model-based RL, such as in the Probabilistic Inference for Learning COntrol (PILCO) algorithm (Deisenroth et al., 2011), where the dynamics model itself is represented by GPs. BO has been shown to be an efficient tool for optimizing both dynamics models (Bansal et al., 2017) and controllers (Marco et al., 2016). To the best of the authors' knowledge, this is the first method that uses BO to optimize the mechanical parameters of a robot simulator and improve its accuracy.

Contribution 2: Additional challenges, however, emerge when one tries to utilize BO in the context of robotic applications. Modern robots often exhibit high-dimensional states and actions

due to their many degrees of freedom (Kober et al., 2013), which leads to high-dimensional parameterizations within simulation as well. In the context of compliant locomotion systems, the SUPERball tensegrity planetary exploration rover is a structurally compliant platform that can distribute forces into linear elements as pure compression or tension (Caluwaerts et al., 2014). This robot's tensile elements can be actuated, enabling it to effectively adapt to complex contact dynamics in unstructured terrains.

Tensegrity robots are inherently high-dimensional and highly dynamic systems, warranting the use of a physics-based simulator for more accurate modeling and control. This accuracy critically depends upon the values of the robot's physical parameters, such as the density of its rigid elements and the elasticity of its tensile elements. While a manual process can be followed to fine-tune a simulation to match the behavior of a real prototype (Mirletz et al., 2015), it is desirable to automate this calibration. Using the approach outlined in Figure 2 directly, however, is still computationally expensive due to the large number of parameters.

This work proposes to embed the physical parameters of a robot in a low-dimensional space to further improve the efficiency of the Bayesian optimization process while avoiding any significant loss of accuracy. A Variational Auto Encoder (VAE) (Kingma and Welling, 2014) is used for learning an embedding that preserves the dynamics of the system. The dimensionality reduction process is tied to the task at hand by first learning a simplified dynamics model, and then utilizing the learned dynamics to train an auto-encoder in the parameter space. BO is conducted afterward in the embedded space, and its outcome is projected back to the original space to be used by the physics engine for prediction and control purposes.

Contribution 3: Since BO is an any-time approach, the question that arises is how accurate the identified tunings must be in order to find a successful policy. Instead of spending time searching for the tuning that best explains the observed data, the optimization can stop after obtaining a tuning that is sufficiently accurate for finding a good policy. Answering this question exactly, however, is difficult, because it would require knowing in advance the optimal policy for the real system. In the example shown in Figure 1 (right), the goal is for the robot to roll an object to the area near the end of the table. When all of the most likely tunings predict that the object will end up somewhere in the desired region, there is no need to further determine which of these tunings is most accurate.

The proposed model identification process is integrated with an iterative policy learning process. An algorithm called Value Guided Model Identification (VGMI) is proposed to efficiently halt the model identification process whenever the most likely tunings predict almost the same value for a previously optimized policy. In other terms, if all the tunings with high probability predict a similar value for the previous policy, then any of these tunings could be used for searching for the next policy. This is based on the assumption that the policy learning algorithm updates policies smoothly by explicitly bounding the divergence between consecutive policies.

Summary of Results: Empirical evaluations performed in simulation and on a real robotic manipulation task demonstrate the benefits of the proposed method. First of all, the experiments show that learning in a physics engine with an identified tuning can be more data-efficient than other model-based methods, such as PILCO (Deisenroth et al., 2011), and model-free ones, such as PoWER (Kober and Peters, 2009). This paper also reports experiments performed using the NASA Tensegrity Robotics Toolkit simulator to demonstrate the efficiency of the proposed model identification framework in high-dimensional spaces. The proposed method is able to efficiently identify the parameters that produce a simulation that most closely matches the observed ground-truth trajectories of the tensegrity platform. Simulation experiments performed in the OpenAI Gym (Brockman et al., 2016) using the MuJoCo simulator indicate that the proposed VGMI algorithm is more time-efficient than competing alternatives, and that it improves the data-efficiency of policy search algorithms, such as TRPO (Schulman et al., 2015).

Outline: The following section provides a more comprehensive review of the literature and the relative contribution of this paper. Section 3 formally states the problem. Section 4 explains in detail the main algorithm of using Bayesian Optimization for model identification with physical

simulation. Section 5 describes the challenges in comparatively higher dimensions, such as the tensegrity robot, and the proposed solution in this context. Section 6 shows how the proposed model identification algorithm can be used along with policy search algorithms. This integration results in a task-driven method for deciding when to stop the model identification process. Both simulation and real robot experimental results are presented in Section 7. Finally, Section 8 concludes the paper with a discussion of the findings from this work.

2. Related Work

Robot modeling relies on accurate knowledge of the kinematics and the dynamics of the robot. *Classical system identification* tunes the parameters of a dynamics model to minimize the difference between the model's output signals and real-world response data for the same input signals (e.g., using least squares) (Swevers et al., 1997; Ljung, 1999). There have also been successful attempts to combine parametric rigid body dynamics models with nonparametric model learning for approximating the inverse dynamics of a robot (Nguyen-Tuong and Peters, 2010).

However, analytical models inevitably do not capture all aspects of a robot's dynamics. Erroneous parameter values, unmodeled or incorrectly modeled contact, and soft materials are common causes of this discrepancy. Instead of relying on analytical models, *model-based reinforcement learning* explicitly learns the unknown dynamics of the robot, often from scratch, and searches for an optimal policy accordingly. Numerous model-based RL algorithms have been applied in robotic tasks (Dogar et al., 2012; Lynch and Mason, 1996; Scholz et al., 2014; Zhou et al., 2016). For example, it has been shown that using inaccurate models can still allow a policy to accurately steer an RC car (Abbeel et al., 2006).

Particularly, *Gaussian Processes* (GPs) have been widely used to model dynamical systems. For example, the PILCO algorithm has been proven efficient in utilizing a small amount of data to learn dynamical models represented by GPs as well as optimal policies (Deisenroth et al., 2011). By replacing the gradient-based optimization algorithm with the parallel, black-box Covariance Matrix Adaptation algorithm (CMA) (Hansen, 2006), the Black-DROPS method 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 in a closely related work for automatically designing open-loop reference trajectories (Tan et al., 2016), wherein trajectory optimization was performed in a physics simulator before real-world execution and real robot trajectories were used to fine-tune the simulator's parameters, and the focus was the gains on the actuators. This latter approach is similar in spirit to our proposed approach. The main difference between our approach and (Tan et al., 2016) is the use of BO instead of evolution strategies, in addition to the task-driven stopping condition. Empirical comparisons of the two methods are reported in Section 7

Bayesian optimization (BO) (Shahriari et al., 2016) is a framework that builds a probabilistic model of an objective function that does not have a known closed form, and searches for the optimums of that function by exploring unknown input parameters while exploiting the most promising regions of the parameter space. A GP is often used to model the unknown function. BO has been frequently used in robotics for direct policy optimization (Pautrat et al., 2017), such as gait optimization (Calandra et al., 2016). BO could also be used to optimize a policy while balancing the trade-off between simulation and real world experiments (Marco et al., 2017), or to learn a locally linear dynamics model while aiming to maximize the control performance (Bansal et al., 2017).

Dimensionality Reduction is widely used for computational speed-up or better interpretability. Principle Component Analysis (PCA) (Tipping and Bishop, 1999) is one classical method to achieve this purpose by assuming a linear mapping. The Gaussian Process Latent Variable Model (GP−LVM) (Lawrence, 2005) extends PCA so that the mapping can be non-linear and non-parametric. Random embedding is another efficient and effective dimensionality reduction technique that has been studied in the context of BO (Wang et al., 2016). *Autoencoders* are also a popular approach for learning low-dimensional embeddings. An autoencoder is a neural network

that learns to reconstruct its input after projecting it into a latent space with lower dimensionality (Vincent et al., 2010). Autoencoders have been shown to be useful in unsupervised learning of low dimensional representations. A variational autoencoder (VAE) adds an additional constraint that the latent space follows a prior distribution, usually assumed to be Gaussian (Kingma and Welling, 2014). This additional constraint makes the model more useful as a generative model, as it also learns to generate output from the prior distribution in addition to the reconstruction.

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 better data-efficiency compared to methods that learn a model from scratch, such as PILCO (Cutler and How, 2015). Policy Improvement with REsidual Model learning (PI-REM) (Saveriano et al., 2017) has also shown improvement over PILCO by utilizing a simulator and only modeling the residual dynamics between the simulator and the real robot. Similarly, using data from simulation as the mean function of a 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). Optimizing the model's parameters in simulation to match with actual data collected on the robot could also help avoid damaging the robot (Bongard et al., 2006). For instance, a recent work proposed an efficient model identification method for learning to predict physical parameters, such as mass and friction (Yu et al., 2017), before searching for an optimal policy. However, this latter work was tested only in simulation.

Model-free methods search for a policy that best solves the the task without explicitly learning the system's dynamics (Sutton and Barto, 1998; Kober et al., 2013), and have been successfully used in robotics to perform complex maneuvers (Peters et al., 2010). The Policy learning by Weighting Exploration with the Returns (PoWER) algorithm (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 achieved great success in some MuJoCo environments. Model-free methods, however, tend to require a lot of training data and can also jeopardize the safety of a robot.

End-to-end learning is an increasingly popular framework (Levine et al., 2016; Agrawal et al., 2016; Fragkiadaki et al., 2016; Wu et al., 2015; Byravan and Fox, 2017; Finn and Levine, 2016), that is based on learning a direct mapping of sensory inputs to controls. This type of approaches usually requires large numbers of physical experiments in order to produce effective results.

Our proposed approach is a model-based method that utilizes a physics engine and concentrates on identifying only the mechanical properties of the simulated objects instead of learning the model from scratch. Furthermore, it utilizes BO and identifies a dimensionality reduction process for dealing with high-dimensional model identification challenges efficiently. An additional contribution is tying the model-identification part with policy search algorithms. Instead of searching for the most accurate tuning, the framework searches for a tuning that is accurate enough to predict the value of a policy that is not too divergent from the searched policy. Therefore, our approach can be integrated with many policy search algorithms that explicitly impose smooth gradual changes in the returned policies, such as the TRPO algorithm (Schulman et al., 2015).

3. Problem Setup

This work considers the availability of a physics engine, which can be used to simulate a robot's actions in a given environment. The physics engine has access to a model of the robot and objects in the robot's surroundings. The model is parametrized with the physical properties of the robot and the objects, such as their masses and friction coefficients. The physical parameters can be completely or partially unknown. The robot needs to perform a given task, which is specified by a reward (or cost) function. Given observed trajectories of the robot and the objects, the objective consists of identifying physical properties of the robot and objects and searching for an optimal policy in simulation, using the identified parameters, to complete the target task, both in simulation and real hardware.

The parameters of the model are represented as a D-dimensional vector $\theta \in \Theta$. The parameter tuning space Θ contains the possible combinations of physical values of these parameters and can be specified using expert knowledge or as a neighborhood given an initial estimate. A real-world trajectory τ^* is a state-action sequence $(x_0, \mu_0, x_1, \mu_1, \ldots, x_{t-1}, \mu_{t-1}, x_t)$, where x_i is the state of the robot and objects, and μ_i is the action applied at time step i. A corresponding simulated trajectory τ for assumed tuning θ is obtained by starting at an initial state $\hat{x}_0$, which is set to be the same as the initial state of the corresponding real trajectory, i.e., $\hat{x}_0 = x_0$, and applying the same control sequence $(\mu_0, \mu_1, \ldots, \mu_{t-1})$. Thus, the simulated trajectory τ results in a state-action sequence $(\hat{x}_0, \mu_0, \hat{x}_1, \mu_1, \ldots, \hat{x}_{t-1}, \mu_{t-1}, \hat{x}_t)$, where $\hat{x}_{i+1} = f(x_i, \mu_i, \theta)$ is the next state predicted by the physics engine using the tuning θ. In other words, the state of the simulation is reset at each time-step to the real observed state. This helps so that the real-world trajectory τ^* and the simulated one τ do not diverge significantly one from the other. Reseting the states in simulation to the observed ones is done only during the search for tuning θ, and not in the subsequent policy search phase which is done fully in simulation given identified θ^*. The current work assumes that the full state is observable.

The goal then is to identify optimal tuning θ^*, which results in simulated state outcomes $\hat{x}_{i+1}$ as close as possible to the real observed states x_{i+1}. In other terms, the objective of this work is to solve the following black-box optimization problem:

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

where $E(\theta)$ is the error function that arises from using tuning $\theta \in \Theta$ in the physics engine.

The following sections describe three challenges in the process of solving the above optimization problem, along with the proposed solutions.

- Section 4: How to identify near-optimal tunings online and on the fly, given a limited time budget. For example, a robot uses a fine-tuned physics model of its different joints, but continuously monitors its sensory inputs and compares it to the model's predictions. This allows a robot to quickly identify the changes in its physical properties and adapt to them in the case of an environmental or robot change.
- Section 5: How to deal with the curse of dimensionality when the parameter space is large. This is especially important for high-dimensional, highly-dynamic systems, such as the tensegrity robot considered in this work.
- Section 6: How to integrate the proposed model identification framework with available policy search algorithms. In order to learn new skills in simulation that can be useful in the real world, the tuning and the corresponding data used for learning must accurately represent intended behaviors.

4. Model Identification with Greedy Entropy Search

Figure 3 provides an overview of the proposed approach. While this example shows a non-prehensile object manipulation application, the same approach can be used in other scenarios such as locomotion of a tensegrity robot. In object manipulation problems, the targeted mechanical properties correspond to the object's mass and friction coefficients of the object. For the tensegrity robot, the physical properties correspond to the density, length, radius, stiffness, damping factor, pretension, motor radius, motor friction, and motor inertia of the various rigid and tensile elements.

The pseudocode for online model identification is given in Algorithm 1, which takes several inputs. Input Θ is the space of all possible values of the tuning vector θ, and in practice is represented by a uniform grid. For example, if the model corresponds to the mass and the coefficient of friction of an object, then $\Theta = \mathbb{R}^{+^2}$, i.e., the set of all two-dimensional vectors of positive real numbers. In Algorithm 1, however, Θ is defined in this example as a regular grid in the two-dimensional space $[0, max_mass] \times [0, max_friction]$ where max_mass and $max_friction$ are

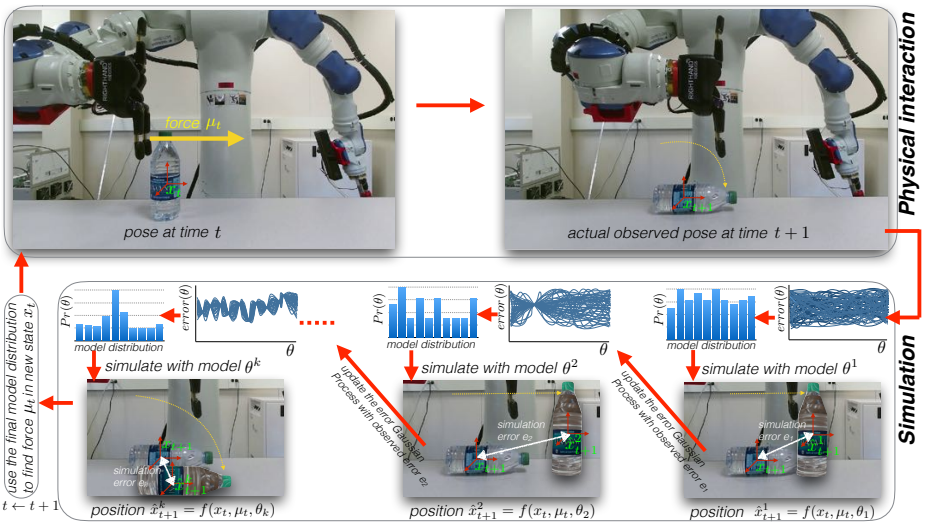

Figure 3: Tuning a model of an object (bottle) through physical simulations. The key idea is to search for model parameters that close the gap between simulation with a physics engine and reality, by using an anytime BO. The process in this figure is repeated after each time-step t, i.e. after every real-world action. In practice, it is more efficient to do the model identification only after a certain number of time-steps.

chosen in advance and the resolution of the grid is chosen based on the available computational resources. An example of a tuning vector $\theta \in \Theta$ would be $\theta = [1.3, 0.4]$, which is interpreted as a mass of 1.3 kg and a coefficient of friction of 0.4. Note that more efficient discretization methods can also be used here, such as sampling values from prior probabilities of their optimality, or by starting with a coarse resolution and iteratively refining it around the most promising regions. Also required are the minimum and maximum number of iterations k_{min}, k_{max}, and one or multiple real-world state-action trajectories $\tau^* = (x_0, \mu_0, x_1, \mu_1, \ldots, x_{t-1}, \mu_{t-1}, x_t)$. For instance, in the case of object manipulation, x_t is the 6D pose (position and orientation) of the manipulated object at time t and μ_t is a vector describing a force applied by the robot's fingertip on the object at time t. Applying a force μ_t results in changing the object's pose from x_t to x_{t+1}. Other optional inputs include a policy π for the task at hand, a threshold of the tuning confidence η and an error threshold ϵ for the policy's value. These inputs will be used later in Section 6.

From an initial distribution P_0 (assuming uniform distribution when no prior knowledge is available), the algorithm samples a parameter vector θ_0 and uses it to simulate the same control sequence that was executed by the robot in τ^* (line 5). Errors are calculated by comparing the simulated trajectory τ and the ground truth trajectory τ^* (line 6). Recall that the goal is to find θ^* that minimizes $E(\theta)$ as defined in Section 3. A belief about the $E(\theta)$ is maintained as a probability measure over the space of all functions $E : \mathbb{R}^D \rightarrow \mathbb{R}$, and is represented by a Gaussian process (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 vector of hypothesized physical properties of the robot or object, and $E(\theta_k)$ is the accumulated distance between actual observed states and states that are obtained from simulation using θ_k.

High-fidelity simulations are computationally expensive. It is therefore important to minimize the number of simulations, i.e., evaluations of the function E, while searching for the optimal parameters that solve Equation 1. To solve this problem, this work proposes a novel simplified variant of the BO technique of Entropy Search (Hennig and Schuler, 2012). Our method, which provides improved computational efficiency in practice, is referred to as *Greedy Entropy Search*.

Algorithm 1: Model Identification with Greedy Entropy Search

Input: Real-world trajectory data $\tau^* = \{(x_i, \mu_i, x_{i+1})\}$ for $i = 0, \ldots, t-1$,
the tuning space Θ,
minimum and maximum number of iterations (evaluated tunings) k_{min}, k_{max},
a control policy π,
tuning confidence threshold η,
value error threshold ϵ ;

Output: Probability distribution P over Θ;

Sample $\theta_0 \sim$ Uniform(Θ); $L \leftarrow \emptyset$; $k \leftarrow 0$; $stop \leftarrow false$;
repeat
  /* Calculating the accuracy of tuning 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 $\hat{x}_{i+1} = f(x_i, \mu_i, \theta_k)$;
    $l_k \leftarrow l_k + \|\hat{x}_{i+1} - x_{i+1}\|_2$;
  $L \leftarrow L \cup \{(\theta_k, l_k)\}$;
  Calculate $GP(m, K)$ on error function E, where $E(\theta) = l$, using data $(\theta, l) \in L$;
  /* Monte Carlo sampling */
  Sample $E_1, E_2, \ldots, E_n \sim GP(m, K)$;
  /* E_i is a discretized definition of a function on Θ */
  foreach $\theta \in \Theta$ **do**

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

  /* Selecting the next tuning to evaluate */
  $\theta_{k+1} = \arg\min_{\theta \in \Theta} P(\theta) \log(P(\theta))$;
  $k \leftarrow k + 1$;
  /* Checking the stopping condition */
  if $k \geq k_{min}$ **then**
    $stop \leftarrow CheckStoppingCondition(\{P(\theta)\}, \pi, \eta, \epsilon)$;
  if $k = k_{max}$ **then**
    $stop \leftarrow true$;
until *stop = true*;

The probability distribution P on the identity of the best tuning θ^*, returned by the algorithm, is computed from the learned GP as

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

where H is the Heaviside step function, i.e., $H(E(\theta') - E(\theta)) = 1$ if $E(\theta') \geq E(\theta)$ and $H(E(\theta') - E(\theta)) = 0$ otherwise, and $p_{m,K}(E)$ is the probability of a function E according to the learned GP

mean m and covariance K. Intuitively, $P(\theta)$ is the expected number of times where θ happens to be the minimizer of E, when E is a function sampled randomly according to GP density $p_{m,K}$. Further explanation of this equation can be found in the appendix of (Hennig and Schuler, 2012).

Distribution P from Equation 3 does not have a closed-form expression. Therefore, a *Monte Carlo* sampling is employed for estimating P. Specifically, the process samples vectors containing values that E could take, according to the learned Gaussian process, in the space Θ (line 9). Sampled vectors E_i are discretized definitions of functions on Θ. The dimension of each vector E_i is the same as the number of tunings in the finite set of tunings given by the grid Θ. For each $\theta \in \Theta$, $E_i(\theta)$ is a predicted value of the reality gap (error as defined in Equation 1) that would occur if tuning θ would be used. $P(\theta)$ is estimated by counting the fraction of sampled vectors of the values of E where θ happens to have the lowest value, as indicated in line 10-11 in Algorithm 1.

Finally, the computed distribution P is used to select the next vector θ_{k+1} to use as a physical model in the simulator (line 12). The entropy of P is given as $\sum_{\theta\in\Theta} -P_{min}(\theta)\log\left(P_{min}(\theta)\right)$. 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. Therefore, the next vector θ_{k+1} should be selected such that the entropy of P would decrease after adding the data point $(\theta_{k+1}, E(\theta_{k+1}))$ to train the GP and re-estimate P using the new mean m and covariance K in Equation 3.

Entropy Search methods follow this reasoning and use Monte Carlo 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 Monte Carlo sampling makes this method unpractical for online model identification. Instead, this work proposes a simple heuristic for choosing the next θ_{k+1}. In *Greedy Entropy Search*, the next θ_{k+1} is chosen as the point that contributes the most to the entropy of P, i.e.,

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

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, it 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 with a high uncertainty -$P(\theta_{k+1})\log\left(P(\theta_{k+1})\right)$.

This process is repeated until a predefined maximum number of iterations is reached (line 16-17), or when a stopping condition becomes true as will be explained in Section 6. Notably, the resulting physical parameter values are not guaranteed to be close to the true values and may not be effective in dynamics regimes that are not represented by the reference data.

The computational complexity of the algorithm is $O(k_{max}N^3)$, where k_{max} is the maximum number of iterations (outer loop), N is the number of input values (e.g, grid size) and $O(N^3)$ is the complexity of sampling functions from the GP, which dominates the other inner loops.

5. Coping with High Dimensionality

For problems where the space Θ of physical parameters has a high dimension D, the method presented in Algorithm 1 is not practical because of the exponential complexity of lines 10-11, where Θ must be sampled with sufficient resolution. This is a common problem in global search, derivative-free, methods. In fact, it has been shown that BO techniques do not perform better than a random search when the dimension of the search space is too large (ten dimensions in the experiments reported by Ahmed et al. (2016)). Therefore, Algorithm 1 cannot be directly deployed on robotic platforms with a large number of joints and parameters, such as Tensegrity robots or compliant dexterous hands.

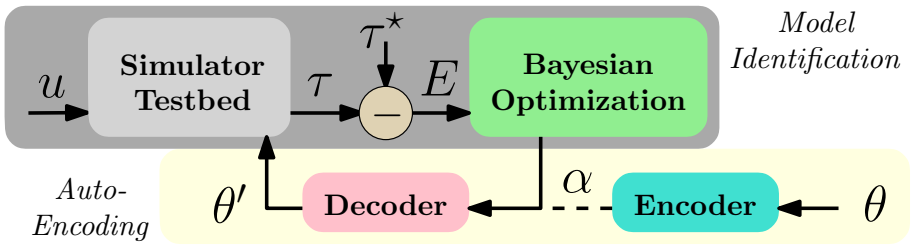

Figure 4: The auto-encoder is trained first to learn the latent low-dimensional embedding. Then, BO is performed in this low-dimensional space to identify the optimal model parameter. The decoder is used to reconstruct the original physical parameters. The encoder is removed once the decoder is trained, because the search is performed directly in the low-dimensional space of α.

Dimensionality reduction is a popular solution to the problem of searching in high-dimensional spaces. This solution is particularly appealing in the context of this work because we are more interested in the accuracy of the predicted trajectory than in identifying the true underlying physical parameters. Mechanical models of motion tie together several physical parameters of an object. For example, in Coulomb's model, the mass and the friction of an object are used in a linear function to predict its planar sliding motion. Therefore, one can linearly map these two parameters to a single parameter and still make accurate predictions. Similarly, random embedding (REMBO) (Wang et al., 2016) is an effective dimensionality reduction technique in the context of BO.

However, the performance of linear mappings is limited to capturing linear relationships between physical parameters. The dynamics of complex robots and objects are known to be highly nonlinear. An autoencoder is suitable for such a task as it learns to reconstruct the input after projecting it to a lower-dimensional latent space. A variational autoencoder (VAE) adds an additional constraint that the latent space follows a prior distribution, usually assumed to be Gaussian (Kingma and Welling, 2014). This additional constraint makes the model more useful as a generative model, as it also learns to generate output from the prior distribution in addition to reconstruction. The VAE can be trained using the physics engine, by randomly sampling different tunings.

VAEs have been shown to be useful in unsupervised learning of low-dimensional representations of spaces that have a low intrinsic dimension, such as images. The intrinsic dimension of the tuning space Θ is generally equal to the number of physical parameters when there are no constraints on the tuning, i.e. the tuning parameters are uniformly distributed. Therefore, training a VAE just to reduce the dimensionality of Θ, without utilizing any knowledge about the optimization goal, would not be an effective approach. For example, when simply describing an object on a table, both its mass and friction need to be known. Hence, the intrinsic dimension of the object's model is 2. But if we are only interested in predicting how the object moves when pushed forward, we only need one parameter, which is the product of its mass and its coefficient of friction. Hence, the intrinsic dimension of the object's model becomes 1, instead of 2. It is only in the context of the pushing control problem that a correlation between mass and friction comes into play.

In the proposed approach, we train a VAE such that the task-related predictions of the physics engine are approximately the same when one uses the given physical parameters, or parameters reconstructed by the VAE , (i.e., by projecting the given parameters to a low dimensional space and projecting them back to the original space). This may further improve the performance when using the identified tuning on the task. This concept is similar to learning a locally linear dynamics model while aiming to maximize the performance of the controller (Bansal et al., 2017).

We adapt the VAE and combine it with the BO process, as shown in Figure 4. First, the VAE is trained with randomly sampled tuning θ to learn a nonlinear mapping of the physical parameter vector to a low-dimension embedding α. The decoder component is used to project the low-dimensional α back to a value θ' in the original tuning space, which can be used in the physics

simulation. Thus, the BO process as detailed in Algorithm 1 can be performed in a computationally efficient manner. Specifically, line 10 of the algorithm is replaced by a process that uniformly samples α vectors in the low-dimensional space and maps them to the original space Θ by using the decoder. This process is computationally more efficient then searching directly in the high-dimensional space Θ, by discretizing it into a regular grid for example. Indeed, the low-dimensional space of α vectors can be quickly covered with uniform samples. Only this step is modified, the rest of the algorithm is the same. In line 11, the probability of each high-dimensional tuning $\theta \in \Theta$ (decoded from a sampled low-dimensional vector α) is estimated from the sampled error functions.

To do so, a simplified dynamics model of the system, represented by a neural network, is trained first. By varying the input tuning θ, the network also captures sensitivity to variation of physical parameters. Because of the complexity of the system, it is extremely difficult to fit a full dynamics model for the Tensegrity robot. One way to simplify the problem is to consider the dynamics of the system in a feature space that has much lower dimensionality but still describes the system dynamics meaningfully. We therefore instead learn the dynamics in the simplified feature space of the height of the center-of-mass of the robot[2], instead of the 126-dimensional full state space. We found from our experiments that using the full state as input and feature as output results in better accuracy than using only the feature both input and output.

Thus, the simplified dynamics model takes the tuning θ, full state x_t, and action u_t as input and predicts the height of the center-of-mass at the next time step $\phi(x_{t+1})$, as shown in Fig. 5. In the case of the Tensegrity robot, θ is the 15-dimensional parameter vector, x_t is the 126-dimensional full state, u_t is the 24-dimensional action and $\phi(x_{t+1})$ is the 1-dimensional height of the center-of-mass mass of the robot. The ground-truth center-of-mass is computed directly from the full state as the average of the centers of all the bars, provided by the simulator utilizing parameter θ.

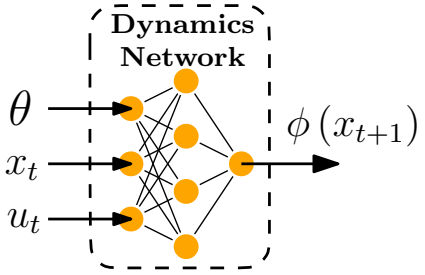

Figure 5: The simplified dynamics model takes the tuning θ, full state x_t, and action u_t as input and predicts the height of the center-of-mass at the next time step, $\phi(x_{t+1})$.

Using the learned simplified dynamics model, we train an auto-encoder to find the low-dimensional representation of the tuning space, as shown in Figure 6. During this process, the dynamics network is fixed and only the weights of the auto-encoder are updated. Instead of aiming to explicitly minimize the reconstruction error as in Figure 4, the goal is to minimize the error in the resulting trajectory using the reconstructed tuning θ'.

The dynamics network accommodates varying θ inputs. It is trained to predict accurate state features for any value of θ given to the network as part of its input. When θ' (output of the VAE) is used instead of θ and there is an error in the predicted feature ϕ, the VAE alone is responsible for the mistake of returning a θ' that does not have the same effect as θ. The error's gradient is thus back-propagated to update the weights of the VAE. The only purpose of the dynamics network in this approach is to train the VAE. The dynamics network is needed only to compute the derivative of the error of $\phi(x)$ (state features) with respect to θ and then back-propagate that derivative to train the VAE. In a sense, the dynamics network is a differentiable clone of the physics engine, but one that can predict only certain features of the state, such as the robot's center of mass.

Once the decoder is trained, we search for the best parameters in the low-dimensional space using BO, and use the decoder to project those identified parameters into the high-dimensional

2. The selection of the state representation is not a focus of the paper. The search for optimal state representation is left for a future work.

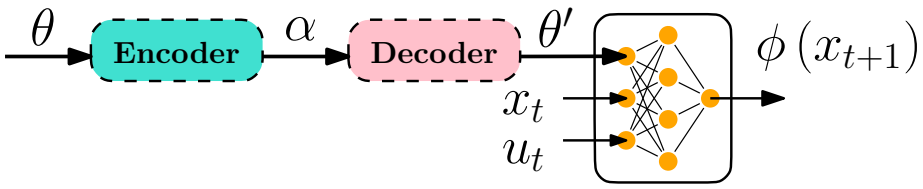

Figure 6: The auto-encoder is trained to learn the latent low-dimensional embedding that is sufficient for predicting key features ϕ of the state transition function, such as the center of the robot. BO is subsequently performed to identify the optimal tuning in this low-dimensional space . The decoder is used to reconstruct the original 15-dimensional tuning in order to perform physical simulations and search for a policy.

space to make them usable in the physics engine. One can also use the dynamics network to directly search for the best parameters θ, but that does not solve the high-dimensionality problem.

One may avoid learning a dynamics network all together by using the simulator in place of the dynamics network in Figure 6. However, most physics engines are not differentiable with respect to their physical parameters, and thus one cannot back-propagate the gradient of the error in the predicted trajectories to the encoding and decoding parts of the network. One remedy could be to calculate the gradient through the simulator numerically through finite differences, but this dramatically increases the computational cost as each value needs to be obtained through physical simulation. Although a differentiable physics engine (Degrave et al., 2016) could prove effective here, the accuracy and efficiency of such engines remain to be tested. Therefore, we train a dynamics network using the physics engine and use that network as a differentiable surrogate (or clone) for the physics engine.

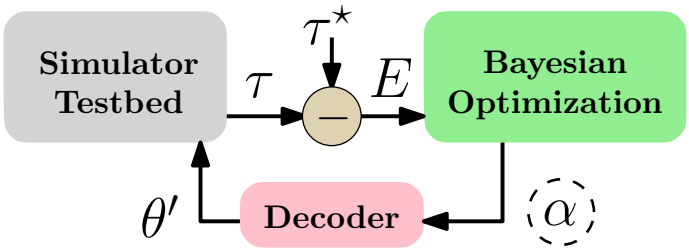

Figure 7: The decoder is used to reconstruct the original 15-dimensional tuning in order to perform simulations with the physics engine.

Finally, BO is performed using the decoder part only, as shown in Figure 7, which corresponds to the last step of Figure 4, when the decoder is trained and the encoder is no longer needed. The difference is that the decoder trained with the dynamics network should provide a low dimensional embedding that preserves physical properties that are relevant to the task at hand, which further improves the efficiency of the BO process. Note that the training of the VAE is performed off-line, while the BO search in the latent space could be performed online, using the learned decoder.

To provide the intuition behind this approach, we use an illustrative toy example of pushing a point-mass along a single dimension.

Intuitive toy example: Consider a cube of mass m resting on a surface, as in Figure 8, which can be represented by a point that can only move along one axis. Assuming uniform and constant coefficient of kinetic friction μ between the cube and its resting surface, an impulse Ft is applied to the cube to cause displacement across some distance S. Applying Newton's Laws and Kinematic Equations, we have the following equations:

$$Ft = \mu m t_1 = m v_0 \tag{4}$$

$$S = v_0 t_1 / 2 \tag{5}$$

where v_0 is the initial velocity of the cube after the impulse and t_1 is the duration of the cube's movement. Solving the above equations, we have:

$$S(m, \mu) = (Ft)^2 / (2m^2 \mu) \tag{6}$$

We use this equation solely to generate training and testing data, in the same manner as a black-box simulator. This parallels later use of a physics simulator for the Tensegrity robot without direct exposure of the differential equations of motion. Our goal is to identify m and μ, given only the initial impulse Ft and the displacement S, without mathematical analysis of the system dynamics equations.

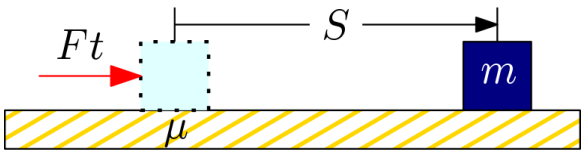

Figure 8: 1-D point pushing example.

This inverse problem, however, is ill-posed due to the fact that different values of m and μ can result in the same S for a given value of Ft. For example, assuming $Ft = 1$, both $m = 1, \mu = 0.32$ and $m = 0.8, \mu = 0.5$ will result in $S = 1.5625$. In other words, as long as the value of $m^2\mu$ is uniquely identified, so is the displacement S. Thus, if the task is to predict S, it is not necessary to individually identify the two parameters m and μ; the scalar value $m^2\mu$ alone can still uniquely determine the system. The goal is then to identify this one-dimensional representation automatically from data, in order to illustrate the concept of learning low-dimensional parameters.

Firstly, we train a dynamics network to predict the displacement S given inputs m, μ, and Ft, similar to Fig. 5. In this case $\theta = [m, \mu]$ and $u = Ft$. The input state x_t can be omitted as we assume the point is always at the origin before being pushed. As the dynamics are simple enough that no feature space is needed, the output of the network is simply x_{t+1}. Secondly, we use the resulting dynamics network to train the auto-encoder to reconstruct m and μ, similar to Fig. 6. During this step, the weights in the dynamics network are fixed. The encoder module is designed to receive input m, μ and output one-dimensional α. A unique aspect of the auto-encoder is that, instead of using reconstruction error of m and μ as the loss function, it passes the reconstructed parameters to the dynamics network and uses the displacement error to train the auto-encoder. The gradient of the displacement error is back-propagated through the auto-encoder. The goal of the auto-encoder is to reconstruct a tuning resulting in similar dynamics, as predicting dynamics is the primary concern. The goal is not to retrieve m and μ but any two number $\hat{m}$ and $\hat{\mu}$ that produce the same motion. Implicitly, the neural network learns parameters that satisfy $m^2\mu = \alpha$ and $\alpha = \hat{m}^2\hat{\mu}$. Here, α is the low-dimensional representation of (m, μ) that preserves the information about the pushing dynamics. Our `VAE` approach was inspired from this observation and aims to generalize the idea of parameter compression to models with higher dimensions and less intuitive dynamics.

6. Model Identification and Policy Search

In this section, we show how to integrate the proposed model identification method with policy search algorithms. Specifically, we show how policies returned by an algorithm such as TRPO can be used to determine if the identified tunings are good enough for the task at hand, and to decide

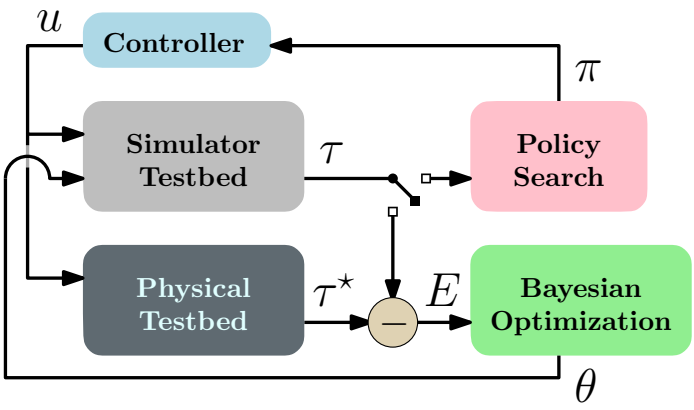

Figure 9: The proposed framework for iterative model identification and policy search: (1) data collection using the real robot (physical testbed) and current policy (2) model identification using a simulator testbed (3) policy search in simulator testbed using the best-identified tuning. Steps (1), (2), and (3) are done iteratively.

on when to stop searching for a more accurate tuning and start searching for an improved policy by using the identified tuning.

The robot's task is specified by a reward function R that maps state-actions (x, μ) onto real numbers. A policy π returns an action $\mu = \pi(x)$ for state x. The value $V^\pi(\theta)$ of policy π given tuning 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 tuning parameters θ. For simplicity, we focus here on systems with deterministic dynamics, without loss of generality.

6.1 Main Algorithm

Given the reward function R and a simulator with tuning 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 our system is presented in Algorithm 2, and also shown in Figure 9. This meta-algorithm consists in repeating three main steps: (1) data collection using the real robot (line 6), (2) model identification using a simulator (line 8), and (3) policy search in simulation using the best-identified tuning (line 9).

6.2 Value-Guided Model Identification

The proposed model identification process, called Value-Guided Model Identification (`VGMI`), is presented in Algorithm 1. Instead of using a pre-defined time budget, we use the function of Algorithm 3 to verify a stopping condition based on the value of the current best policy π.

The current best policy π is the one executed in the previous iteration of the main algorithm (Algorithm 2). 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 $|V^\pi(\theta) - V^\pi(\theta^*)|$ for two given tunings 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 tunings with high probabilities, and $|V^\pi(\theta) - V^\pi(\theta^*)| \leq \epsilon$ then there is no point in continuing

Algorithm 2: Main Loop

1 $t \leftarrow 0$;
2 Initialize distribution P over Θ to a uniform distribution;
3 Randomly initialize policy π;
4 Initialize a policy search algorithm (e.g. `TRPO`) with π;
5 **repeat**
6 Execute policy π for H times on the real robot, and collect a new real-world trajectory $\tau^* = \{(x_i, \mu_i, x_{i+1})\}$ for $i = t, \ldots, t + H - 1$;
7 $t \leftarrow t + H$;
8 Run Algorithm 1 with Algorithm 3 (for the stopping criterion), using collected τ^* and current policy π, for updating distribution P;
9 Run the policy search algorithm in the simulator with the tuning $\arg\max_{\theta \in \Theta} P(\theta)$ to find an improved policy π'; $\pi \leftarrow \pi'$;
10 **until** *Timeout*;

Algorithm 3: Value-Guided Model Identification

1: **function** CHECKSTOPPINGCONDITION($P(\theta), \pi, \eta, \epsilon$)
2: $stop \leftarrow false$;
3: $\theta^* \leftarrow \arg\max_{\theta \in \Theta} P(\theta)$;
4: Calculate values $V^\pi(\theta)$ for all tunings θ that have a probability $P(\theta) \geq \eta$ by using the physics engine for simulating trajectories with tunings θ and using Monte Carlo to estimate $V^\pi(\theta)$ from the sampled trajectories ;
5: **if** $\sum_{\theta, \forall P(\theta) \geq \eta} P(\theta) |V^\pi(\theta) - V^\pi(\theta^*)| \leq \epsilon$ **then**
$stop \leftarrow true$;
6: **return** stop
7: **end function**

the optimization to find out which one of the two tunings is actually the most accurate because both tunings would result in similar policies. The same argument could be used when there are more than two tunings with high probabilities.

The stopping condition verified by Algorithm 3 is requested by Algorithm 1 in lines 14-15. The current best tuning θ^* (line 3) is selected along with all other tunings with confidence $P(\theta)$ greater than a threshold η. The values V^π of the current best policy π are then obtained by simulating under different selected high-confidence tunings θ that satisfy $P(\theta) \geq \eta$ (line 4). If the sum of the weighted absolute differences between values according to the best tuning and other high-confidence tunings is smaller than a threshold ϵ, which can be set proportionally to the maximum reward (line 5), then the model identification process can be concluded, and the main loop in Algorithm 2 continues from line 5. In practice, these criteria indicate that there is no benefit to continuing the process but do not necessarily ensure that the desired task is fulfilled. Task success may further require that the tuning space Θ and the form of the model, which necessarily omits some real-world subtleties, are sufficient for approximating the true system in context.

7. Evaluation

Experimental evaluations are performed in both simulation and on a real robotic manipulation task to assess the proposed model identification method. First, we report simulation experiments performed in the OpenAI Gym (Brockman et al., 2016) with the MuJoCo simulator to verify the advantages of the Greedy Entropy Search over BO alternatives. Second, real robot manipulation experiments show that the data-efficiency of policy search is highest when using an analytical model of the system dynamics with unknown parameters and tuning its parameter values from data. The proposed approach outperformed methods such as PILCO that do not rely on analytical models, or avoid dynamics modeling altogether such as in PoWER. Third, experiments are performed using the NASA Tensegrity Robotics Toolkit to evaluate the efficiency of the proposed model identification framework, especially in high-dimensional spaces. The proposed method is able to efficiently identify parameters that produce simulated trajectories within a small distance of the observed ground-truth trajectories. Distance here is defined as the average Euclidean distance between the ground-truth 3D positions of each rod's center in the Tensegrity robot and the corresponding 3D positions in simulation, predicted by executing the same sequence of actions as in the ground-truth trajectories. Finally, simulation experiments with the MuJoCo simulator show that VGMI is more efficient computationally, in general, than other model identification alternatives.

7.1 Evaluating GES

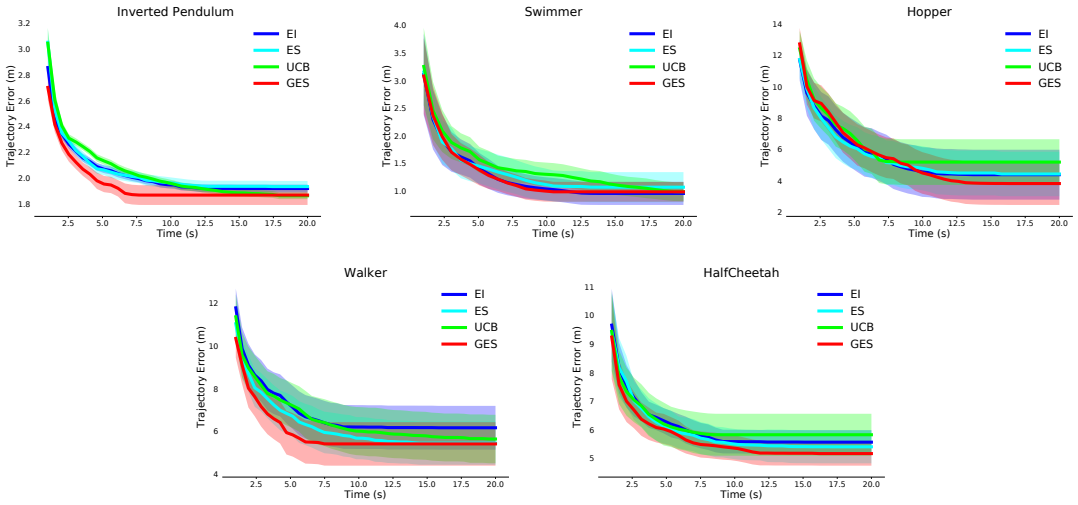

Figure 10: Model identification using the MuJoCo simulator with four different acquisition functions: Expected Improvement (EI), Entropy Search (ES), Upper Confidence Bound (UCB), and the proposed Greedy Entropy Search (GES).

This section performs a limited ablation study where we fix all the components of the proposed approach, and vary only the acquisition function. Specifically, the proposed Greedy Entropy Search (GES) method is compared to certain search methods in the OpenAI Gym (Brockman et al., 2016) with the MuJoCo simulator. Note that the comparison is tied to this specific environment and problem, and one cannot conclude from this study whether the GES method is better or worse than the others in general. Fig. 10 presents the corresponding results.

The unknown parameters that need to be tuned are the masses of the robot's joints. The dimension of tuning space Θ varies depending on the robot. **Inverted Pendulum**: A pendulum is connected to a cart, which moves linearly. The dimensionality of Θ is $D = 2$, one for the mass of

the pendulum and one for the cart. **Swimmer**: The swimmer is a 3-link planar robot, with $D = 3$. **Hopper**: The hopper is a 4-link planar monopod robot, with $D = 4$. **Walker2D**: The walker is a 7-link planar biped robot, with $D = 7$. **HalfCheetah**: The halfcheetah is a 7-link planar cheetah robot, with $D = 7$.

For each environment, the simulator is initiated with the corresponding ground-truth model and an optimal policy is learned in the ground-truth simulator. All the policies are trained with `TRPO` (Schulman et al., 2015) implemented in rllab (Duan et al., 2016). The policy network has two hidden layers with 32 neurons each. The learned policies are used to generate ground-truth state-action trajectories that are used for identifying the masses of each robot's joints.

An inaccurate tuning of the robot's model is used as prior for model identification. The masses of the joints were randomly increased or decreased by adding noise between 10% and 50% of the actual values. The goal is to tune the masses so that the trajectories from the inaccurate simulators become similar to the ground-truth trajectories when both are using the same actions.

In order to demonstrate the benefit of the proposed `GES` method during model identification, we compare to the original `ES` method (Hennig and Schuler, 2012), and to other popular `BO` methods including Expected Improvement (`EI`) (Močkus, 1975) and Gaussian process upper confidence bound (`GP-UCB`, denoted by `UCB`) (Auer, 2002). Fig. 10 demonstrates that `GES` results in improved convergence overall, relative to the alternatives, including the original `ES` approach.

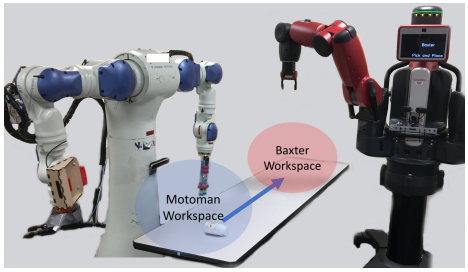

Figure 11: Robot Pushing Experiments

7.2 Model Identification with BO: Real Robot Pushing Experiments

Setup: Consider Fig. 11 and 12, where a *Motoman* robot assists a *Baxter* robot to pick up a bottle. The object 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 that pushes the bottle 100 cm away inside *Baxter*'s reachability region. If the robot simply executes 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 that none of the two robots can reach. Both outcomes are undesirable as they would require human intervention to reset the object.

The goal is to determine the optimal value of the policy's free parameter, i.e., the pushing velocity of the robot's hand. The pushing direction is always towards the target position and the hand pushes the geometric center of the object, which was manually placed on a marked initial position. The proposed approach first searches for the best tuning θ^*, corresponding to the mass and

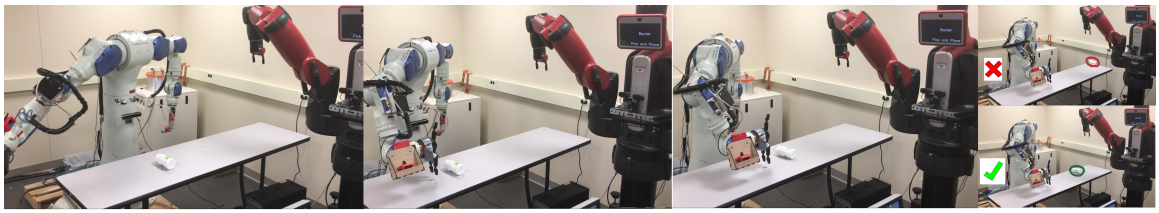

Figure 12: Experiment where the *Motoman* pushes the object into Baxter's workspace after identifying the object's physical parameters.

friction coefficients of the pushed object, through Algorithm 1. The domain of the mass parameter is $]0, 1]kg$ and the domain of the friction factor is $]0, 1]$. The domain of these two parameters is discretized into a 500×500 regular grid. Since the dimension of the model parameter space in this experiment is low (two), there is no need here for the dimensionality reduction technique presented in Section 5. The goal of this experiment is to evaluate the performance of Algorithm 1 alone, and to compare it to representative model-based and model-free reinforcement learning algorithms which previously were evaluated on comparable open-loop tasks, such as robot table tennis. To collect the ground-truth trajectories needed by Algorithm 1, the robot pushes the bottle gently in random directions with velocities that do no exceed $0.1m/s$ to make sure that the bottle does not move away from the robot's workspace while collecting the data. This random data-gathering policy is manually set. The state of the system is given by the angle of rotation of the principal axis of the object, in addition to its 2D position on the table. The object in this experiment is set so that its principal axis is parallel to the support surface. The action is the velocity of pushing the object in the direction of its secondary axis, which causes the bottle to roll for a certain distance.

Once a tuning is identified, a pushing policy π^* is searched in simulation using the identified tuning. Since the pushing policy is just a velocity command, open-loop, and one-dimensional, there is no need to use other policy optimization methods such as TRPO for finding policy π^*. Pushing policy π^* is returned by uniformly sampling 20 different pushing velocities ranging from 0 to $1.00m/s$ in the simulation and selecting the one that achieves the best expected reward. The method is compared to two `RL` methods: `PoWER` (Kober and Peters, 2009) and `PILCO` (Deisenroth et al., 2011). The reward function is $r = e^{-dist}$, where $dist$ is the distance between the object position after pushing and the desired target position. For `PILCO`, the state space is the object's 3D position (2D pose on the table and rotation angle).

Results: Two metrics are used for evaluation: 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. 13 shows that in the real-world experiments the proposed method achieves both lower final object location error and fewer object drops compared to alternatives. 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 in addition to the policy, but its model is not as accurate as that of `VGMI`. Since a very simple policy search is used for `VGMI`, the performance is expected to improve with more advanced policy search, such as combining `PoWER` with `VGMI`.

These results are not surprising given the fact that the physics engine, used by `VGMI`, already includes a structure that accounts for a large amount of prior knowledge that is absent in model-free RL and most model-based RL techniques. For example, the physics engine already incorporates laws of friction, contact models, and rigid-body dynamics, whereas the other compared methods need to implicitly acquire this knowledge from data and from scratch. While such a comparison may seem unfair, we have included it here to illustrate the practical benefit of taking advantage of physics engines for solving rigid-object manipulation tasks, which are commonly used as benchmarks for evaluating model-free RL algorithms as well as statistical model-based ones (such as `PILCO`). In this sense, this experiment compares three different paradigms of RL rather than three specific algorithms, represented here by `VGMI`, `PoWER` and `PILCO`. The fundamental question here is: do we need to re-learn basic models of mechanics from data, or shall we take advantage of our knowledge of these models and focus only on identifying unknown parameters of these models? While the answer is not simple, we conjecture that it depends on the amount of available data relative to the complexity of the task. Methods that utilise physics engines can learn from small amounts of data, but they cannot recover from structural inaccuracies of the engines, while model-agnostic methods do not suffer from such a problem, given sufficient training data.

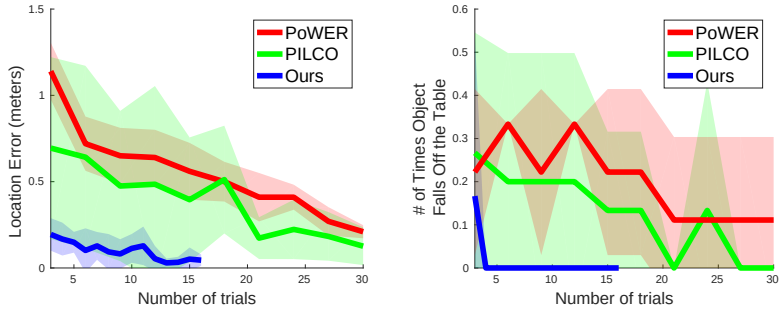

Figure 13: Pushing policy optimization results using a *Motoman* robot. Our proposed approach achieves both lower final object location error and fewer object drops compared to alternatives.

7.3 Experiments on Model Identification with Dimensionality Reduction

7.3.1 TOY EXAMPLE: 1-D POINT PUSHING

In this toy example, we use Equation 6 as the "simulator" to generate data. 20,000 training data points are generated by randomly sampling m, μ, F and t. 2,000 additional data points are generated for testing. As a proof of concept, we only compare model identification using `BO` in the original 2D tuning space and in the 1D latent space in the auto-encoder.

The dynamics network has three hidden layers with 64, 128, and 64 hidden units and ReLU activation functions. The encoder network has two layers with 32, 1 units, and decoder has two layers with 1, 32 units. Both the encoder and the decoder only have ReLU on the first hidden layer.

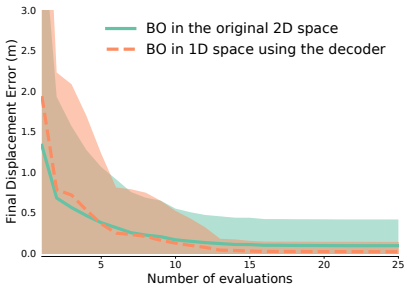

Figure 14: 1-D point pushing result: Model identification in the 1D space is more efficient than in the original 2D tuning space.

The result is shown in Fig. 14. After training, the decoder is able to reconstruct m and μ that result in negligible errors in the predicted displacement. Compared to search in the original 2D tuning space, using the decoder to optimize in 1D space is computationally more efficient.

7.3.2 TENSEGRITY ROBOT

Setup: This experiment aims to identify the 15 parameters of the model of the Tensegrity SUPERball robot in NASA's Tensegrity Robotics Toolkit. The complex dynamics and high dimensionality of the robot make this problem very hard.

The data gathering policy is obtained by taking into account the system's geometric configuration alone, without accounting for dynamics (Littlefield et al., 2017). While keeping the base triangle in full contact with the ground, this policy commands a change in cable lengths that corresponds to a desired shift in the system's center-of-mass. By displacing this value relative to the supporting base triangle, the system can be made dynamically unstable, causing a forward flop.

By using the geometric controller in simulation, 1200 trajectories are generated as training data by sampling the 15 parameters within the $\pm 10\%$ range of the ground truth. The assumption is that such error can appear during the robot modeling process and the proposed approach should be able to correct it and minimize the reality gap. Example trajectories can be found at `https://sites.google.com/view/tensegrity/`.

The proposed approach using BO with an auto-encoder and a dynamics network (BO+AE & DyNet 5D) is compared with: 1) random search (Random Search 15D), where random values of the parameters are selected within the $\pm 10\%$ range in the original 15-dimensional tuning space, 2) Bayesian optimization (BO 15D) in the original 15-dimensional tuning space, 3) Random EMbedding Bayesian Optimization (REMBO 5D) (Wang et al., 2016), and 4) Bayesian optimization with variational auto-encoder (BO+VAE 5D). BO+AE & DyNet 5D, REMBO 5D, and BO+VAE 5D all reduce the dimensionality of the tuning space from 15 to 5.

The encoder and decoder of the VAE used in the experiment are both two-layer neural networks. The input dimension of the encoder and the output dimension of the decoder is 15, which is the dimension of the tuning space. The latent space is 5-dimensional. Between them is one layer of 400 dimensions. This dimension is chosen through cross-validation by balancing accuracy and network complexity. The prior distribution of the latent space in the VAE is assumed to be $N(0, 1)$. Based on the three-sigma rule, when sampling between $[-3, 3]$, this interval should cover 99.7% of the latent space when the VAE is optimized. For REMBO, each time a random projection matrix is generated to project the parameters into $[0, 1]$.

The dynamics network is a four-layer neural network with dimensions 128, 64, 32, and 1. The encoder and decoder with the dynamics network are also two-layer neural networks but much narrower than the VAE with only 10 and 5 dimensions for each layer.

Results: Fig. 15 shows the average error between the trajectories using the tunings identified by different methods and the trajectories generated from the ground-truth simulator. Trajectory error is computed as the norm of the differences in node positions. When optimizing in the original 15-dimensional space, as a data-efficient global optimization method, BO outperformed random search. Further improvements are achieved by dimensionality reduction, making the search more efficient. Use of a pre-trained decoder ensures that the reduction captures an effective tuning subspace before model identification begins, resulting in the smallest error after only 1 second. BO with the autoencoder and the learned dynamics network (BO with AE and Dynamics Net) in the 5-dimensional space achieves the lowest error for any time allowance, outperforming the method using REMBO or VAE. This shows that a better learned latent embedding enables more efficient parameter search in the BO process.

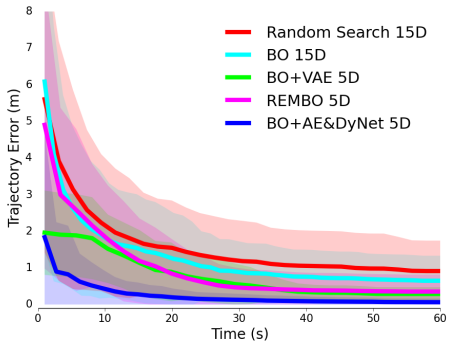

Figure 15: Test trajectory errors of different methods for the Tensegrity robot as a function of time budget for the model identification process.

Table 1 provides the errors for each of the final identified parameters. Interestingly, although it achieved the lowest trajectory error, BO with AE and Dynamics Net did not identify all parameters with the lowest error. Specifically, it was found to be the case that parameters like rod_length, rod_space, rod_length_mp, motor_radius, motor_friction, and motor_inertia are not actively used in the current simulation model of SUPERball. Thus even methods like VAE may be able to get a lower reconstruction error on the parameters themselves, BO with AE and Dynamics Net is able to achieve lower trajectory error as it ties the model identification process with the dynamics and the task-related features (height of the center of mass). Additionally, some parameters may have a stronger influence on the robot dynamics.

	Random Search 15D	BO 15D	REMBO 5D	BO+VAE 5D	BO+AE&DyNet 5D
density	5.01±2.86	4.10 ± 2.66	1.85±1.88	1.96 ± 0.46	**1.30**±0.05
radius	2.49±1.94	2.08 ± 1.73	1.86± 1.84	1.43±0.28	**0.30**±0.03
density_mp	5.40±2.96	5.19 ± 2.66	1.89± 1.86	2.38±0.39	**1.00**±0.18
radius_mp	4.78±2.78	5.36 ± 2.97	1.94± 1.94	2.00±0.55	**0.69**±0.43
stiffnessActive	4.49±2.68	4.44 ± 2.79	1.84± 1.90	**1.68**±0.46	1.71±0.03
damping	4.62±2.75	4.33 ± 2.78	1.81± 1.89	**2.02**±0.44	2.26±0.15
rod_length	5.05±2.75	4.72 ± 2.69	**1.90**± 1.88	2.04±0.31	6.25±0.59
rod_space	4.96 ±2.81	4.87 ± 2.74	1.88± 1.84	**1.68**±0.36	5.20±0.38
rod_length_mp	4.89 ± 2.81	5.22 ± 2.87	1.88± 1.96	**1.70**±0.58	4.06±0.23
pretension	5.10 ± 2.83	5.10 ± 3.01	1.93± 1.89	1.58±0.50	**1.38**±0.34
maxTens	4.99 ± 2.87	4.66 ± 2.80	1.86± 1.83	**1.85**±0.42	5.48±0.12
targetVelocity	4.85 ± 2.62	5.31 ± 3.01	1.84± 1.90	2.06±0.62	**0.49**±0.31
motor_radius	5.11 ± 2.90	4.38 ± 2.81	1.90± 1.91	**1.79**±0.66	4.90±0.23
motor_friction	5.10 ± 2.71	5.32 ± 3.17	1.89± 1.82	2.19±0.27	**0.65**±0.03
motor_inertia	4.78 ± 2.80	4.87 ± 3.01	1.83± 1.88	2.00±0.45	**1.86**±0.06

Table 1: Identified Parameter Error (%). Best result on each parameter is highlighted.

7.4 Experiments on Value-Guided Model Identification

In this section, `VGMI` is compared against a) Covariance Matrix Adaptation Evolution Strategy (`CMA-ES`) (Tan et al., 2016) and b) Least Squares (`LS`) optimization using `L-BFGS-B` algorithm (Swevers et al., 1997).

7.4.1 SETUP

All methods optimize the objective function of Eq. 1. The experiments are performed in the same simulation environment as in Section 7.1.

The number of iterations for policy search is varied based upon problem difficulty. The main loop in Alg. 2 is executed for 20 iterations for Inverted Pendulum, 100 iterations for Swimmer, 100 iterations for Hopper, 200 iterations for Walker2D and 400 iterations for HalfCheetah. `VGMI` follows Alg. 1 and is executed every 10 iterations, i.e., $H = 10$ in Algorithm 2. The same iterations are used for `CMA` and `LS` and they are also given the same time budget for model identification at each iteration. All results shown here represent the mean and standard deviation of 20 independent trials. The tuning confidence threshold is set as $\eta = 0.1$, determined empirically based on the overall performance of the algorithm on the various tasks in MuJoCo.

7.4.2 RESULTS

Figures 16 and 17 show the results of these experiments. The cumulative reward per trajectory on the ground-truth system is plotted in Fig. 16 as a function of the total runtime. `VGMI` always achieves the highest reward across the five benchmarks. `CMA` performs better than `LS`, but relies upon a large number of physically simulated trajectories.

Fig. 17 shows that `VGMI` automatically stops the model identification process without manually setting a time budget. The red line is the threshold ϵ of the error in the predicted value function used by `VGMI`. The green line is where `VGMI` decides to stop based on threshold ϵ and the criterion in Algorithm 3. We also ran the same algorithm with increasing pre-set time budgets (x-axis) instead of using the stopping criterion, and empirically estimated the reality gap in predicting the value of

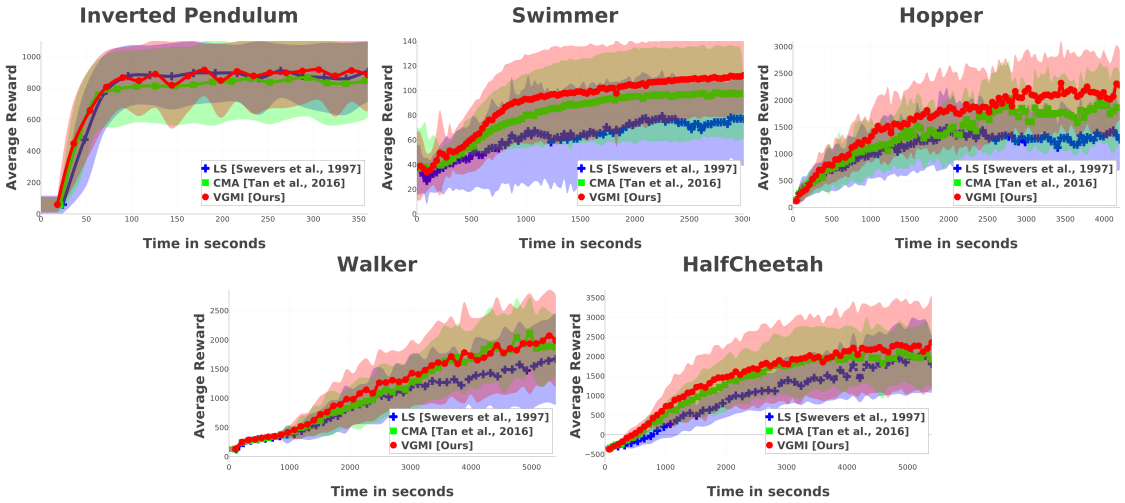

Figure 16: Average reward as a function of total runtime, including both model identification and policy search times.

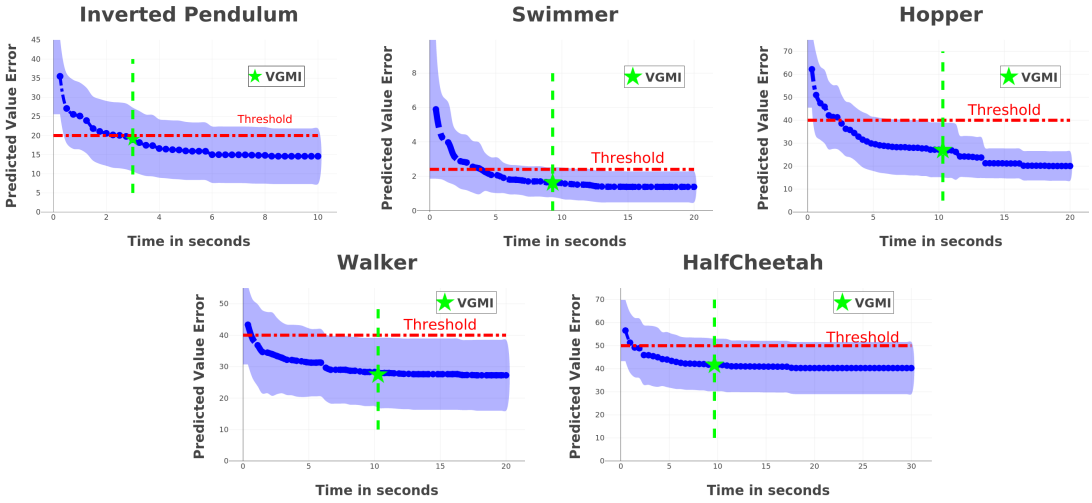

Figure 17: The error in predicting the value of the next policy ($|V^{\pi'}(\theta) - V^{\pi'}(\theta^*)|$) as a function of time spent in the model identification loop during one iteration of the algorithm. When `VGMI` decides to stop (green line), the actual value error is indeed always below the threshold used in its stopping criterion. The error is computed *a posteriori* because the best policy π' in the next round of the main loop of Algorithm 2 is not known to `VGMI` when it decides to stop the model identification and returns θ. Ground-truth parameters vector θ^*, used in computing the errors reported here, is also unknown to the algorithm as it is the objective of the search.

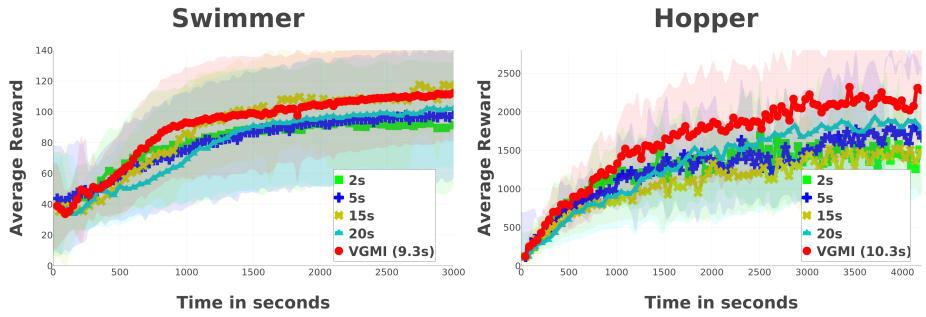

Figure 18: Alternate schemes for setting the per-iteration time budget for model identification are compared. The average reward per trajectory is reported here as a function of the total runtime, which includes both the model identification and the policy search times. The time budget for policy search is fixed and is the same for all the compared schemes, while the time budget for model identification is varied. VGMI achieves the best performance by automatically deciding when the tuning is sufficiently accurate.

the best policy using the best-identified tuning and the ground-truth tuning. This shows that VGMI does not stop prematurely. When it chooses to stop (green line), the error $|V^{\pi'}(\theta) - V^{\pi'}(\theta^*)|$ of the predicted value of the next policy π' (blue curve) is indeed below the predefined threshold ϵ. This also shows that the advantage of further tuning the identified tuning quickly diminishes past the stopping point.

Fig. 18 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.

8. Discussion

This work proposes a practical approach that exploits the availability of efficient physics engines and Bayesian optimization methods for model identification to increase the data-efficiency of reinforcement learning algorithms. Instead of searching for the most accurate tuning of the physical model, the objective is to identify a tuning that is good enough to predict the value function of any policy that is close to the current best policy. Therefore, the proposed approach can be used in combination with any policy search algorithm that guarantees smooth changes in the learned policy. Both simulated and real robot experiments show that the proposed algorithm can decrease the number of rollouts needed to find a satisfactory policy. To cope with high-dimensional parameter spaces, a projection to a lower-dimensional space by learning a latent embedding with an auto-encoder is presented.

8.1 Interesting Extensions

An interesting extension of this work involves the identification of controls during the learning process to help quickly minimize the error. This could be viewed as a robust control process, which utilizes Bayesian Optimization's output in terms of a belief distribution of the identified parameters, so as to minimize entropy and maximize the safety of the experimentation process. Furthermore, it is interesting to compare the generality of the learned parameter tuning and resulting control schemes that utilize them against completely model-free and end-to-end approaches for RL and control. Future work will include further analysis of the proposed method's properties, such as

expressing the conditions under which the inclusion of the model identification approach reduces the need for physical rollouts, and quantifying the associated speed-up in convergence. Application of the method to alternative physical tasks is also of interest.

8.2 Structural Disparity Between Simulated Model and Reality

The setup in the current work assumes that there exists a parameter tuning θ^* of the known model that makes the simulation match reality. If the model structure embedded in the simulator does not resemble reality, then the resulting policy would perform poorly in the real world. When the algorithm converges, according to the proposed stopping condition, there is no guarantee that the returned policy will perform well if the model structure is wrong.

It is not obvious, however, how to solve this problem efficiently when the difference between the simulated model and reality is structural, and not just a matter of parameter tuning. For example, the object may be compliant whereas it is simulated as a rigid object. In this case, there is no guarantee that the tuning process will result in a predictive model for the real world task.

Nevertheless, the proposed stopping criterion is helpful even in this case. Letting the algorithm run for longer in the case of structural model disparities would not result in a better policy. The algorithm will just continue to search within a set of models that are all structurally inaccurate. The purpose of the stopping criteria is to avoid pointless optimization as soon as it becomes clear that further optimization would not make any difference on the quality of the returned policy, which could be good or bad depending on the embedded model structure and collected data, but not because the algorithm stopped prematurely.

The case of an unknown model structure can perhaps be approached by considering a bigger search space that includes the structure of the model. The proposed technique could potentially be generalized to efficiently search in that space as well given a parameterization of the model's structure.

8.3 Noise in Observations

The proposed formulation assumes that the recorded trajectories are noise-free. In the real-world robot pushing experiments, there is noise that stems from the depth sensor and the estimated pose of the pushed object. While the current framework does not explicitly model this noise, the parameters returned by the Bayesian Optimization routine were accurate enough for the considered task.

This result may be related to the possibility that the noise is mostly additive and white. In this case, the optimal parameters also minimize the least-squares objective function used by the Bayesian Optimization. In general, noise in the recorded trajectories can lead to errors in the identified parameters, especially when distant outlier points are included in the data.

One possible solution is to start with a rough initial transition and observation model (such as a linear-Gaussian) and infer the most likely trajectory from the observed one by using the Viterbi algorithm. The most likely trajectory could then be given to the BO to identify the physical parameters. A more elaborate solution is to include the noise parameters (such as a mean and covariance of a Gaussian) in the search. The Bayesian Optimization is then used to search for the joint model and noise parameters that reduce the reality gap.

References

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

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

M.O. Ahmed, B. Shahriari, and M. Schmidt. Do we need "harmless" bayesian optimization and "first-order" bayesian optimization? In *NIPS BayesOPT Workshop*, 2016.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3 (Nov):397–422, 2002.

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

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

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

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

Roberto Calandra, André Seyfarth, Jan Peters, and Marc P. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 76(1):5–23, 2016. ISSN 1573-7470.

K. Caluwaerts, J. Despraz, A. Iscen, A. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral. Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of The Royal Society Interface*, 11 (98), 2014.

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.

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

Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016.

M. Deisenroth, C. Rasmussen, and D. Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Robotics: Science and Systems (RSS)*, 2011.

Mehmet Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-Based Grasp Planning Through Clutter. In *Robotics: Science and Systems VIII*, July 2012.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, pages 1329–1338, 2016.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *ICRA 2017*, 2016.

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *ICLR*, 2016.

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

Philipp Hennig and Christian J. Schuler. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

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

Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *NIPS*, pages 849–856, 2009.

Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(Nov):1783–1816, 2005.

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.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Z. Littlefield, D. Surovik, W. Wang, and K. E. Bekris. From quasi-static to kinodynamic planning for spherical tensegrity locomotion. In *International Symosium on Robotics Research (ISRR)*, Puerto Varas, Chile, 12/2017 2017.

Lennart Ljung, editor. *System Identification (2Nd Ed.): Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999. ISBN 0-13-656695-2.

Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *IJRR*, 18, 1996.

Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. Automatic lqr tuning based on gaussian process global optimization. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 270–277. IEEE, 2016.

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.

B. T. Mirletz, I.-W. Park, R. D. Quinn, and V. SunSpiral. Towards bridging the reality gap between tensegrity simulation and robotic hardware. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

J Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

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

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.

Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the AAAI*, pages 1607–1612, 2010.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005. ISBN 026218253X.

Andrew P Sabelhaus, Jonathan Bruce, Ken Caluwaerts, Pavlo Manovi, Roya Fallah Firoozi, Sarah Dobi, Alice M Agogino, and Vytas SunSpiral. System design and locomotion of superball, an untethered tensegrity robot. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 2867–2873, 2015.

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

Jonathan Scholz, Martin Levihn, Charles L. Isbell, and David Wingate. A Physics-Based Model Prior for Object-Oriented MDPs. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. Optimal robot excitation and identification. *IEEE transactions on robotics and automation*, 13(5):730–740, 1997.

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

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

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.

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 Robotics: Science and Systems*, July 2017.

Jiaji Zhou, Robert Paolini, J. Andrew Bagnell, and Matthew T. Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 372–377, 2016.