Keep it Simple: Data-efficient Learning for Controlling Complex Systems with Simple Models

Thomas Power[1] and Dmitry Berenson[1]

***Abstract*—When manipulating a novel object with complex dynamics, a state representation is not always available, for example for deformable objects. Learning both a representation and dynamics from observations requires large amounts of data. We propose Learned Visual Similarity Predictive Control (LVSPC), a novel method for data-efficient learning to control systems with complex dynamics and high-dimensional state spaces from images. LVSPC leverages a given simple model approximation from which image observations can be generated. We use these images to train a perception model that estimates the simple model state from observations of the complex system online. We then use data from the complex system to fit the parameters of the simple model and learn where this model is inaccurate, also online. Finally, we use Model Predictive Control and bias the controller away from regions where the simple model is inaccurate and thus where the controller is less reliable. We evaluate LVSPC on two tasks; manipulating a tethered mass and a rope. We find that our method performs comparably to state-of-the-art reinforcement learning methods with an *order of magnitude less data*. LVSPC also completes the rope manipulation task on a real robot with 80% success rate after only 10 trials, despite using a perception system trained only on images from simulation.**

***Index Terms*—Machine Learning for Robot Control; Motion and Path Planning**

I. Introduction

While recent machine learning methods have been effective for many manipulation tasks, they rely on access to large datasets of the system being manipulated [1], [2], [3]. Yet in many scenarios we do not have time to gather extensive training data with an object before performing a task. Sim-to-real transfer has been used to fine-tune parameters on limited real-world data when the real object is similar to those used in simulation [4], [5], but these methods struggle if the objects are significantly different. We would like to use prior knowledge about the object to reduce the data required for learning, but the question of *how* to effectively use prior knowledge when encountering a *novel* object remains open.

This paper addresses how to leverage dynamics models of simple systems when learning to control much more complex, but related, systems online. While it is possible to learn dynamics using only online data (e.g. [6]), we wish to use our knowledge of a simple model to make the learning much more data-efficient, and thus practical for real-world application. For example, consider a tethered mass being swung by a gripper (Figure 1). The dynamics of the system are complex and require a great deal of data to learn. However, if we treat the system as a cart with a rigid pendulum, we can predict the dynamics fairly accurately *for some subset of the state-action space*. We can exploit this subset to perform tasks such as bringing the mass to a target, even without a globally-accurate dynamics model. Simple models are often used in this way, for example in deformable object manipulation [7], [8] and control for humanoids [9].

To use knowledge of the dynamics of the simple model to control the more complex true system, we must know which states of the complex system correspond to which states of the simple system. What makes this problem especially difficult is that, while we can design a useful state representation for the simple system offline, we do not know what state representation to use for the complex system, so we cannot explicitly define a correspondence between states.

Our key insight for overcoming this problem is that the simple system (and its state representation) is a good approximation of the complex system when it gives rise to similar image observations to the complex system. By using a metric for observation similarity that reasons about uncertainty we can build a controller for the complex system and also learn where our approximation is inaccurate (to avoid visiting those parts of the state space). By utilizing domain randomization during training, we enable a single simple system state to elicit a wide variety of image observations; i.e. shapes, colors, and obstacles can vary while still producing an image we consider to be *visually-similar*. We use online system identification to estimate the parameters of the simple model, however, deciding which class of simple model to use for a given task is not within the scope of this paper. Here we made this decision manually but seek to automate selecting the class of simple model in future work.

This paper makes the following contributions: 1) Learned Visual Similarity Predictive Control (LVSPC), a novel framework for learning how to perform manipulation tasks with a complex system given only a simple model and images from a small number of trials online; 2) Evaluation of LVSPC on manipulating a tethered mass (using a cart-pole as a simple model) and a rope (using a rigid body as a simple model) (See Fig. 1) in simulation, showing large improvements in data-efficiency over baselines (PlaNet [6] and CURL [10]). LVSPC also completes the rope manipulation task *on a real robot* with 80% success rate after only 10 trials.

LVSPC consists of two phases: 1) Offline, we train an

Manuscript Received: October 16, 2020; Revised: December 21, 2020; Accepted: January 15, 2021. This paper was recommended for publication by Editor Dana Kulic upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF grant IIS-1750489 and ONR grant N00014-21-1-2118.

[1]Authors are with the University of Michigan, Ann Arbor, MI, USA. {tpower, dmitryb}@umich.edu

Digital Object Identifier (DOI): see top of this page

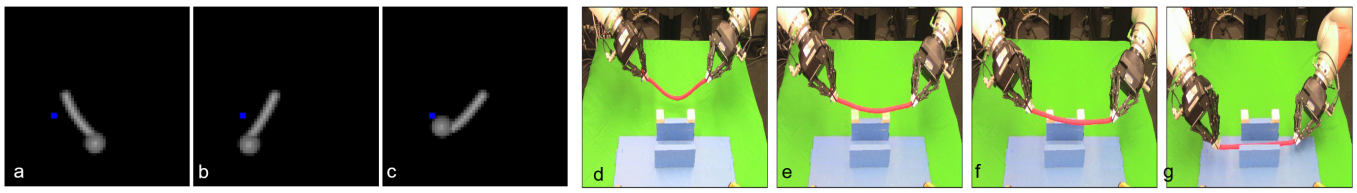

Fig. 1: *(a-c)*: LVSPC controlling a tethered mass to a desired position (blue) from images by treating it as a cart-pole; *(d-g)*: LVSPC brings a rope to a target location in a narrow passage between two obstacles while avoiding protrusions by treating the rope as a rigid object. The robot starts with the rope slack but pulls it taut to keep the approximation more accurate, allowing it to complete the task.

ensemble Convolutional Neural Network (CNN) perception system on image observations of the simple system, outputting an estimate of the simple system's state. 2) Online, given image observations of the complex system, we do system identification to estimate parameters of the simple system dynamics and learn a Gaussian Process (GP) that predicts where the simple model is accurate. We use the simple model and the GP to track the object via a Gaussian Process Unscented Kalman Filter (GPUKF) [11] and perform control via Model Predictive Path Integral Control (MPPI) [12], biasing the system away from inaccurate transitions.

II. Related Work

Dynamics from Images: Learning-based approaches using dynamics models for control with images observations have included learning dynamics models directly in image space [1], [3], [13]. Dynamics in image space are highly complex, and these methods require large amounts of data. Other methods learn dynamics in a lower-dimensional latent space [14], [2], [6], [15]. None of these methods incorporate prior knowledge. SE3-PoseNets [16] learn dynamics in pose-space from point cloud data. [17] use the positions of a set of ordered points as the representation of a rope and pre-trains a state estimator on ground truth in a simulator. Unlike LVSPC, neither of these methods use a given model approximation nor do they reason about model uncertainty.

Using simplified models: Simplified models have been widely explored in the legged robotics literature, in particular using spring-mass damper models [18], [9]. Simplified models have been used to generate trajectories for a lower-level controller to track with guarantees [19]. However, these guarantees require access to a high-fidelity model. Other work [20] has used a set of simple models and a selection mechanism to choose between them. [8] use a given simplified dynamics model and learns a classifier on whether a given transition is reliable. We use GP uncertainty to model transition reliability rather than a classifier. We also use image observations and perform tracking concurrently.

Incorporating model uncertainty: Previous work has shown that reasoning about model uncertainty can improve data efficiency [21], [22]. PILCO [21] uses a Gaussian Process dynamics model for model uncertainty and achieves high data efficiency on learning control policies. Gaussian Processes dynamics have also been used for the purpose of both avoiding uncertainty [23], or explicitly seeking it [24]. PETS [22] uses a probabilistic ensemble of neural networks to model uncertainty and is able to outperform PILCO on control tasks with high state dimension. These methods have only been demonstrated on tasks for which state is available, and not on image domains where parameterizing uncertainty can be difficult. LVSPC aims to combine modeling of uncertainty in the dynamics with strong priors to maintain high data efficiency when learning from images.

III. Problem Statement

We consider a nonlinear discrete-time system with state $x \in \mathcal{X}$ and controls $u \in \mathcal{U}$. The system has unknown true dynamics given by $x_{t+1} = f(x_t, u_t)$. We assume $\mathcal{X}$ may be arbitrarily high-dimensional and unobserved. Instead we may only have access to observations $o \in \mathcal{O}$ via an observation function at the current state $o_t = g(x_t)$.

We define a trial as a time-limited attempt to find a sequence of controls $\{u_1, ..., u_T\}$ such that the final state $x_T \in \mathcal{X}_{goal}$ where $\mathcal{X}_{goal}$ is the trial's goal region. We assume that we can fully observe when the system has reached the goal i.e. $o \in \mathcal{O}_{goal} \iff x \in \mathcal{X}_{goal}$. The goal in observation space is defined as $\mathcal{O}_{goal} = \{g(x) : x \in \mathcal{X}_{goal}\}$. We assume that data collection on the true system is expensive. The unknown dynamics and high-dimensional state make this problem intractable to solve with a small dataset. Instead we seek to model the system in a latent state of lower dimensionality $z \in \mathcal{Z}$ with *simple* dynamics $\hat{f}_\rho$ parameterized by ρ with input-dependent noise. The transition distribution, which we will denote as p_z for shorthand is given by

$$p(z_{t+1}|z_t, u_t) = \mathcal{N}(\hat{f}_\rho(z_t, u_t), Q(z_t, u_t)) \tag{1}$$

We assume that $\hat{f}_\rho$ is given and is differentiable with respect to (z, u, ρ). Q is an input-dependent uncertainty term. We also assume that the simple dynamics are Markovian. The simple system has the same observation space $\mathcal{O}$ and has a given observation function $o_t = \hat{g}(z_t)$. We assume that we can *a priori* specify some subset of the goal region in $\mathcal{Z}$ as $\mathcal{Z}_{goal}$, i.e that $\{\hat{g}(z) : z \in \mathcal{Z}_{goal}\} \subset \mathcal{O}_{goal}$. This could also be done by specifying $\mathcal{O}_{goal}$ directly (as is common in learning to control from images, e.g. [25]) and using this to infer z_{goal}. We then seek to design a feedback policy $u_t = \pi(z_t)$ such that $z_T \in \mathcal{Z}_{goal}$ for some time T. Our goal is to design π using $\hat{f}_\rho$ so that it achieves high success rate after a small number of trials.

IV. Methods

Our approach to this problem requires input in the form of a simple model approximation that is believed to accurately represent the dynamics over some subset of the complex system $(\mathcal{X}, \mathcal{U})$. By using this simple model in simulation we

Algorithm 1 LVSPC

Inputs: Simple model dynamics: $\hat{f}_\rho$; Simple model cost: c; Simple model renderer $\hat{g}$; Initial data size N; # Episodes K

Offline Training with simple system data

$\{y_i, o_i\}_{i=1}^N \leftarrow$ `CollectData`$(\hat{f}_\rho, \hat{g}, N)$;
$\phi \leftarrow$ `TrainStateEstimator`$(\{y_i, o_i\}_{i=1}^N)$;

Online Training with complex system data

$\mathcal{D} \leftarrow \emptyset$; $\rho, Q \leftarrow$ `Initialize`;
for k $\in \{1, ..., K\}$ **do**
  $p_z \leftarrow \mathcal{N}(\hat{f}_\rho(z_t, u_t), Q(z_t, u_t))$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup$; `Rollout`$(p_z, c, \phi)$;
  $\rho \leftarrow$; `FitSimpleSystem`$(\mathcal{D}, \hat{f}_\rho)$;
  $Q \leftarrow$ `FitGP`$(\mathcal{D}, \hat{f}_\rho, Q, \rho)$;

can generate large amounts of data. The key to our approach is to leverage this data and our knowledge of the simple system. We then reduce the problem of unsupervised representation and dynamics learning to that of supervised learning of a perception system for the simple model representation (offline), and then learning when this representation and the dynamics are accurate (online).

Our full method is shown in Algorithm 1 and Figure 2. The overall procedure is to first generate a dataset of images with corresponding simple model configurations and then to train a perception system to estimate these configurations from images. Once this perception system is trained offline, we move to the online execution/learning phase, where we must manipulate the never-before-seen complex system.

The goal of the online execution is to reach a given goal region. However, because the perception system and the simple model dynamics can only account for *some* complex model states, we must try to avoid states where the perception/dynamics are inaccurate. To this end, we collect data as we attempt the task and use that data to train a GP that captures the error in the simple model predictions. This error distribution is input into a Kalman Filter variant to better estimate the state and into a trajectory optimizer, which attempts to avoid regions of state space where the simple model predictions are inaccurate. The process of planning trajectories, executing one action, estimating the resulting state, and replanning a trajectory (Alg. 2) repeats until the goal (or a timeout) is reached.

A. Simple Model

The simple system state may contain elements which cannot be estimated from a single image, e.g. velocities. Thus we define the components of the simple state that can be noisily observed from a single image as latent observations y. We then have the non-linear discrete-time state space model with dynamics described in Eq. (1). In general there will be a non-linear mapping from z to y. In this paper we consider only a linear mapping, which is sufficient for our models:

$$y_t = C z_t + \epsilon, \tag{2}$$

For an n-dimensional simple model system ($z \in \mathbb{R}^n$) with m-dimensional ($m \leq n$) observations ($y \in \mathbb{R}^m$), $C = [I_{m\times m}, 0_{m\times n-m}]$ selects the latent observations from z. For example, if z is the position and velocity of a point, then y is only the position, which is all that can be observed from a single image. In the case where $\epsilon \sim \mathcal{N}(0, R)$ for positive-definite R we can use noisy measurements y to estimate z by filtering using non-linear techniques such as the Unscented Kalman Filter (UKF) [26]. We will show how to use a GP to learn $Q(z_t, u_t)$ in Eq. (1) from data in Sec. IV-D.

Algorithm 2 Rollout

Inputs: Transition distribution p_z; Simple model cost: c; CNN Ensemble ϕ

$\mathcal{D} \leftarrow \emptyset$; $\mu_1^z, \Sigma_1^z \leftarrow$ `Initialize`;
for t $\in \{1, ..., T\}$ **do**
  $\mu_t^y, \Sigma_t^y \leftarrow \phi(o_t)$;
  $y_t \sim \mathcal{N}(\mu_t^y, \Sigma_t^y)$;
  $\mu_t^z, \Sigma_t^z \leftarrow$`GPUKF`$(\mu_{t-1}^z, \Sigma_{t-1}^z, u_{t-1}, p_z, y_t)$;
  $u_t \leftarrow$ `MPPI`(μ_t^z, c, p_z);
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mu_t^y, \Sigma_t^y, u_t)$;
  `ExecuteAction`(u_t);
  if `AtGoal` **then** break;
return $\mathcal{D}$

B. Probabilistic CNN Ensemble for Perception

In order to use the simple model for the complex system, we need a perception system ϕ that maps images to simple model states (even if the image is generated from the complex system). We would also like a way to estimate how well a simple model state approximates the complex system at a given state, as this gives us an estimate of confidence in the simple system dynamics at this state. We use the uncertainty in the perception estimate as a proxy for correspondence between the simple state and the unknown complex state. The perception output is

$$\mu_t^y, \Sigma_t^y = \phi(o_t) \tag{3}$$

$$y_t \sim p(y_t|o_t) = \mathcal{N}(\mu_t^y, \Sigma_t^y), \tag{4}$$

where the variance Σ_t^y estimates the uncertainty, and ϕ is the perception system. We assume an isotropic Gaussian in Eq. 3, thus Σ_t^y can be described by a vector $\sigma_t^y \in \mathbb{R}^m$. Ensembles have been empirically shown to give useful estimates of prediction uncertainty, which can be used to evaluate if a given input is out-of-distribution w.r.t the training data [27]. Thus using ensembles avoids manually defining a similarity between the complex system observations and observations generated from the simple system. Instead we can input observation o_t from the complex system into our perception system, and if it produces a high-certainty estimate of y_t (i.e. where $||\sigma_t^y||$ is small), this implies that y_t is a good approximation for the complex system at time t.

We parameterize ϕ as a CNN ensemble which is trained with data generated from the simple system. Each CNN in the ensemble is a probabilistic CNN which outputs the parameters of a Gaussian, these are then combined into one Gaussian estimate. We train the CNN via supervised learning on observations of the simple system which we collect from simulation, along with correspond simple system states. Importantly, we assume that we can generate observations from

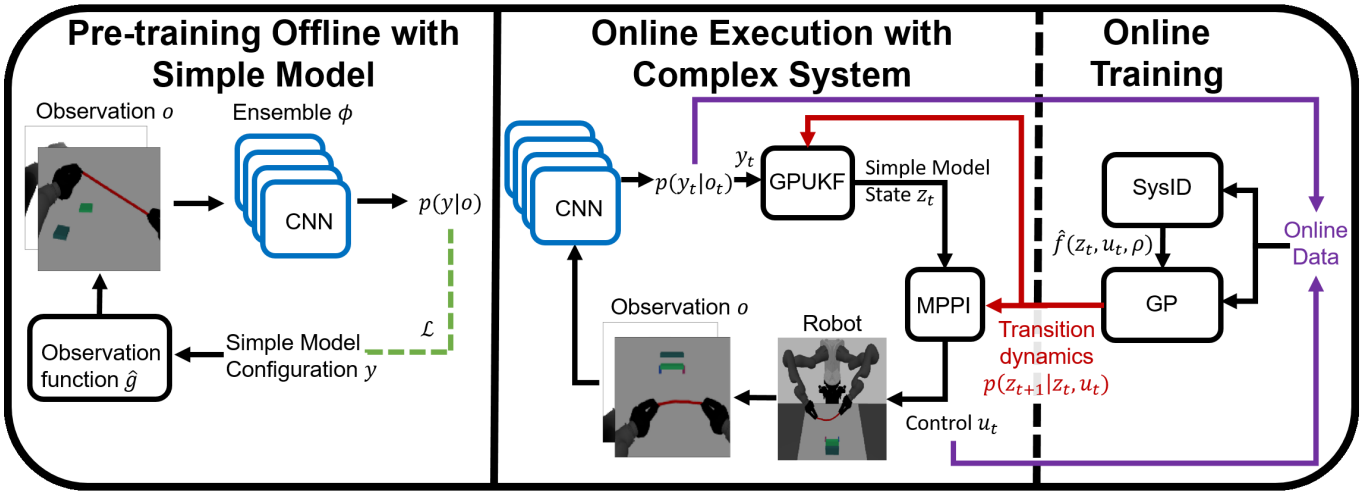

Fig. 2: Method overview. *Left*: Training the CNN ensemble on image observations generated from the simple system offline. ϕ is a CNN ensemble with variance used as a measure of uncertainty; *Center*: Online execution using the simple model CNN with GPUKF filtering and MPPI for control; *Right*: Procedure for fitting parameterized simple model and GP from observations of the complex system. The transition probability (red) is trained to predict the future uncertainty of ϕ, allowing us to avoid avoid areas where ϕ is not confident.

the simple system which are similar to the complex system observations. To avoid requiring precise knowledge of the complex system before generating the simple model data, we generate a diverse training set of observations from the simple model. For example, we generate cart-poles with varying pendulum length for the tethered mass scenario. By generating diverse observations via domain randomization, our notion of visual similarity means that there is a simple system with some appearance and system parameters that looks similar to the complex system. See in Fig. 3 for examples.

Given an o_t of the complex system online, we sample y_t from the output of the ϕ and use this along with the learned GP transition distribution (Sec. IV-D) to track a Gaussian distribution over the simple model state ($p(z_t|u_{1:t-1}, y_{1:t}) = \mathcal{N}(\mu_t^z, \sigma_t^z)$) with a GPUKF [11]—an extension to the UKF for GP dynamics. When predicting $p(z_{t+1}|u_{1:t}, y_{1:t})$ in the GPUKF we use the posterior mean of the GP (Sec. IV-D) to perform the unscented transform, while the process noise is the posterior covariance of the GP, $Q(z_t, u_t)$, evaluated at (μ_t^z, u_t).

C. System Identification

The simple model dynamics may be parameterized by ρ (for example mass, length, etc.) and in order to use it, we must estimate the ρ which best approximates the complex system. One approach is using the Kalman filter to jointly estimate ρ and the latent state z, but we found that this was not numerically stable. Instead we use maximum-likelihood estimation on observed trajectories from the complex system.

Given an observed trajectory of the complex system consisting of $\{o_t, u_t\}_{t=1}^T$ we encode the observations into $\{\mu_t^y, \sigma_t^y, u_t\}_{t=1}^T$. Since our trajectory may contain transitions which the simple model cannot accurately predict, we split the trajectory into N trajectories of length $K < T$, and discard trajectories with average uncertainties above threshold α so we are left with high-certainty sub-trajectories. For each sub-trajectory we rollout the actions $u_{1:T}$ using Eq. (1) and (2) to get estimated observations $\hat{y}_{1:T}$ and perform gradient ascent

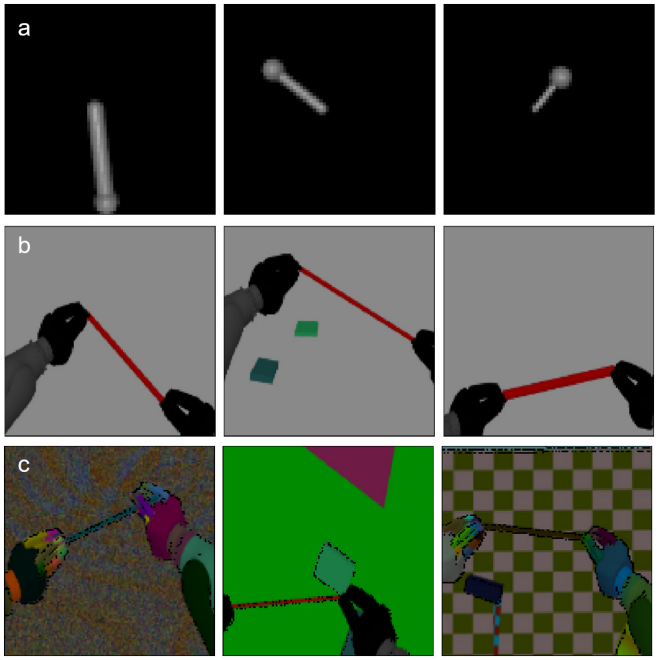

Fig. 3: Examples of data generated from the simple system for training the CNN ensemble. (a) Tethered mass experiment, showing different geometries of the cart-pole. (b) Simulated rope manipulation experiment, showing different geometries of rigid link, and differing number and geometries of objects. (c) Real robot rope manipulation experiment. We randomize textures, lighting, obstacle configuration, camera pose, and rigid link geometry and add noise.

on the parameters ρ and the trajectory initial states $\{z_1^i\}_{i=1}^N$ by maximizing the log likelihood of $\hat{y}_{1:T}$ in the distribution output by the CNN ensemble $\mathcal{N}(\mu_{1:T}^y, \sigma_{1:T}^y)$. The CNN weights are held constant. This process optimizes ρ to match the observed dynamics for high-certainty transitions in $(\mathcal{Z}, \mathcal{U})$.

D. Predicting Future Uncertainty with GP Regression

From ϕ we have a confidence in our simple model approximation at a given y (the uncertainty σ^y). To keep the system in regimes where the approximation is accurate we also need to predict the future uncertainty conditioned on actions. Our

uncertainty expresses uncertainty over the *validity* of the state as a description of the complex system, rather than the *value* of the state. Since we are using state uncertainty as a measure of confidence in the simple model approximation we model this uncertainty as state and action-dependent and use a GP with mean function $\hat{f}_\rho$ and kernel function $\mathcal{K}$ to model the transition distribution. The GP posterior is

$$p(z_{t+1}|z_t, u_t) = \mathcal{N}(\hat{f}_\rho(z_t, u_t, \rho) + \mu_f(z_t, u_t), Q(z_t, u_t)), \quad (5)$$

where μ_f and Q are typically found via conditioning on some training set. However in our case this is a Gaussian Process State Space Model (GPSSM) [28] with transition probability above and emission probability defined in Eq. (2). Training this GP is non-trivial as we do not have access to z directly. Instead we must jointly infer both the transition probability and z during training.

We use a Parametric Predictive GP (PPGP)[29] in order to train a GP with state-dependent aleatoric uncertainty via stochastic gradient descent. The uncertainty of the GP σ^z is used to predict the uncertainty of the CNN ensemble σ^y via Eq. (2). The PPGP is a sparse GP method which fits psuedo-inputs (ζ) and psuedo-ouputs ($\gamma \sim \mathcal{N}(m, S)$) such that conditioning the GP on (γ, ζ) approximates the true GP posterior. The GP parameters are thus (m, S, ζ) as well as the kernel hyper-parameters. The GP posterior contains an additional μ_f term compared with Eq. (1). This allows the GP posterior mean to deviate from that of the simple model, attempting to fit transitions which do not conform to the simple model dynamics. Since our representation is known to be insufficient to model the true dynamics of the system, we are conservative and do not allow the GP to fit such transitions by constraining $m = 0$ and thus $\mu_f = 0$. We compare to a variant of our method where we do not enforce $\mu_f = 0$ in our experiments.

We now describe how to train this GP using trajectories from the complex system of the form $\{\mu_t^y, \sigma_t^y, u_t\}_{t=1}^T$. We would like Eq. (2, 5) and an initial $p(z_1)$ to be able to reproduce the trajectory and uncertainties from the CNN. The learning objective to be minimized is then

$$\mathcal{L} = \mathcal{KL}(p(y_{1:T}|o_{1:T})||p(y_{1:T}|u_{1:T})), \quad (6)$$

where $\mathcal{KL}$ is the Kullback–Leibler divergence, $p(y_{1:T})$ represents the joint distribution $p(y_1, ..., y_T)$, $p(y_{1:T}|o_{1:T})$ is the output of the CNN, and $p(y_{1:T}|u_{1:T})$ is the prediction from the dynamics and Eq. (2). The GP predicted uncertainty σ_t^z is used with Eq. (2) to predict a latent observation uncertainty $\hat{\sigma}_t^y$. This objective aims to make the predicted uncertainty $\hat{\sigma}_t^y$ and the observed uncertainties σ_t^y consistent, i.e. the GP will predict the future uncertainty.

$p(y_{1:T}|o_{1:T})$ is fixed (i.e. we are not retraining the CNN online). Given this, we can rewrite the objective in terms of expectations over $p(y_{1:T}|o_{1:T})$

$$\mathcal{L} = -\mathbb{E}_{p(y_{1:T}|o_{1:T})}\left[\log p(y_{1:T}|u_{1:T})\right] + \mathcal{H}\left[p(y_{1:T}|o_{1:T})\right], \quad (7)$$

where $\mathcal{H}$ is the entropy and this entropy term can be dropped as it only depends on the pre-trained CNN. We can then optimize by maximizing the conditional expectation in Eq. (7) of $y_{1:T}$. To do this we construct a variational lower bound on $p(y_{1:T}|u_{1:T})$. This lower bound is given by

$$\begin{aligned} ELBO = \sum_{t=1}^{T} \mathbb{E}_{q(z_t)}\left[\log p(y_t|z_t)\right] - \mathcal{KL}(q(z_1)||p(z_1)) - \\ \sum_{t=2}^{T} \mathbb{E}_{q(z_{t-1})}\left[\mathcal{KL}\left(q(z_t) \,||\, p(z_t|z_{t-1}, u_{t-1})\right)\right], \end{aligned} \quad (8)$$

where the prior on the initial state is $p(z_1) \sim \mathcal{N}(0, I)$ and $q(z_t) = p(z_t|y_{1:t}, u_{1:t-1})$ is the GPUKF filtering distribution [11]. The final objective to minimize is given by $\mathcal{L}_1$

$$\mathcal{L}_1 = -\mathbb{E}_{p(y_{1:T}|o_{1:T})}[ELBO] \geq \mathcal{L} \quad (9)$$

To evaluate this objective we use the reparameterization trick to sample from the CNN and estimate gradients for $\mathcal{L}_1$. After performing this training procedure we obtain the transition distribution p_z, which is used by the GPUKF to perform filtering and by the MPC to predict future uncertainty.

E. Model Predictive Control

For MPC we use MPPI [12] with a cost c for the given task. To encourage the controller to keep the system in the domain of the simple model we add a cost to penalize the predicted uncertainty. Thus the cost function has the form $c(z, \sigma^z, u)$ (examples are shown in the experiments). Note that typically in this setting the expected cost is computed, but as mentioned in the previous section, our uncertainty does not express uncertainty over the *value* of the state. When rolling out a predicted trajectory with the model we propagate the expectation through the dynamics and record the one-step uncertainty for each step resulting in a trajectory $(\mu_t^z, \sigma_t^z, u_t)_{t=1}^T$ with which to calculate the cost. If we do not penalize this uncertainty, it will be ignored, which is equivalent to assuming the simple model is always accurate (we compare to this method in our experiments). Also, because we manually design the simple model state representation, we can incorporate additional information, such as avoiding collision, into the cost, which would have to be learned for an unsupervised learned representation.

V. Experiments

We evaluate LVSPC on 1) manipulating a tethered mass, and 2) placing a rope in a narrow opening vs. baselines in the low-data regime. An episode is a time-limited attempt to reach the goal (terminating early when the goal is reached). See the accompanying video for example task executions.

A. Environments

a) Tethered Mass: This task involves controlling a tethered mass by applying force to the base of the tether. The goal is to bring the mass to a target without the tether contacting the target (tether contact results in failure). We implement this system in MuJoCo [30]. There is a single actuated horizontal joint at the top of the tether (see Figure 4). Goals are randomly assigned at the start of each trial. This example demonstrates the applicability of LVSPC to highly-dynamic systems where velocity must be considered.

The simple system we choose here is the pendulum on a cart (i.e. a cart-pole); we choose this because we observed that when the tether is taut the system will behave like a pendulum. We use an analytical dynamics function for $\hat{f}_\rho$. We define $z = [p_{x_{cart}}, p_{x_{mass}}, p_{y_{mass}}, \dot{p}_{x_{cart}}, \dot{\theta}]$, where θ is the angle of the pendulum. We define the latent observations as $y = [p_{x_{cart}}, p_{x_{mass}}, p_{y_{mass}}]$ and thus $C = [I_{3\times3} \quad 0_{3\times2}]$. The parameters ρ are [`mass_cart, mass_pole, angular_damping`].

b) Rope Manipulation: This task consists of two KUKA iiwa 7-DOF arms holding the ends of a rope. The goal to bring the center of the rope to the center of a narrow gap between two obstacles. These obstacles have small protrusions on which the rope can become caught. We implement this environment in Gazebo with the ode45 back-end (Figure 4). The action space of the robot is $[\Delta p_L, \Delta p_R] \in \mathbb{R}^6$ where p_L, p_R are the left and right end-effector positions, respectively. We use a Jacobian-based method for inverse kinematics so that transitions in the robot's configuration space are smooth. The observations consists of RGBD data from an overheard Kinect. The goal and obstacle configuration for the task remain fixed across trials, but the starting locations of the end-effectors vary. We choose this example because it mimics cable installation, which is necessary for manufacturing and repair applications, where there are often narrow gaps and protrusions.

The simple system we choose here is to treat the rope as if it is a rigid link. The simple dynamics are then specified by adding a constraint that the gripper distances remain fixed. This approximation will be accurate so long as the rope is kept taut for the duration of the task. We define $z = y = [p_L, p_R]$ and $C = [I_{6\times6}]$. Since this model does not require dynamic parameters we forego the sysid step of our method.

B. Baselines

We compare LVSPC to two recent methods from the literature. The first method is PlaNet [6], a model-based reinforcement learning algorithm. PlaNet learns a low-dimensional state representation along with dynamics and cost functions. The second is CURL [10], which uses a contrastive loss to learn a representation in which to learn a policy and has shown state-of-the-art sample-efficiency. For each of these baselines we test them by training them directly on the task with the complex system. We also show results for when the baselines are pre-trained on the simple system and fine-tuned on the complex system to investigate if these methods can take advantage of the data from the simple system. Both baselines were originally proposed with RGB observations, and we extend them to use RGBD for the rope experiment.

We also test with three variants of LVSPC: 1) The full method which does both system identification and GP learning; 2) LVSPC without the GP, this is equivalent to only using the simple model for control, and assuming it will be sufficiently accurate for all transitions. We choose this variant to investigate whether learning and avoiding inaccurate areas of the simple model state space is helpful for task performance; and 3) LVSPC without constraining the GP posterior to be zero-mean, hence attempting to learn a better approximation of the dynamics in the simple system state space, rather than only where the simple model is accurate.

C. Simple Model Data

a) Tethered Mass: For pre-training the state estimator we generate 5000 trajectories of 20 time-steps from the cart-pole using random actions and render the cart-pole configurations to produce images. This corresponds to 100000 64×64 grayscale frames. For domain randomization, we vary the dimensions and parameters of the system (see Figure 3(a)).

b) Rope Manipulation: For pre-training the state estimator we generate 800 trajectories of 50 time-steps length using random actions from the rigid body system and render the configuration. This corresponds to 80000 $128\times128\times4$ RGBD frames. For domain randomization, we vary the dimensions of the rigid link and the obstacles, as well as the obstacle locations (examples shown in Figure 3(b)).

D. Cost Functions

For both LVSPC and PlaNet we use an MPC horizon of 40 and sample 1000 trajectories per timestep. We do not have a cost on control. CURL and PlaNet use the true environmental cost i.e. $c_{env}(x_t)$, whereas LVSPC and variants use an equivalent cost based on the simple model state with an uncertainty penalty $c(z_t, \sigma_t^z)$. The environmental costs use the true state from the simulator to calculate the cost (because CURL and PlaNet have no knowledge of the simple model), whereas LVSPC uses the simple model state to approximate this cost, effectively giving CURL and PlaNet an advantage.

a) Tethered Mass: The environmental cost consists of three parts; a euclidean distance to goal, a collision penalty for the tether and mass, and a penalty when the system goes out of view of the camera. The cost functions are $c(z_t, \sigma_t^z) = \delta_g \texttt{distToGoal}_{\mathcal{Z}} + \texttt{OffScreen}(z_t) + 10\texttt{checkCollision}(z_t) + \beta\sigma_t^z$ and $c_{env}(x_t) = \delta_g \texttt{distToGoal}_{\mathcal{X}} + \texttt{OffScreen}(x_t) + 10\texttt{checkCollision}(x_t)$, where β is a parameter on how heavily to weigh uncertainty, and δ_g is 0 if the goal is reached before time t and 1 otherwise. To balance exploiting vs. exploring we increase β from 0 to 2.0 in the first 10 episodes. This cost is not memoryless; δ_g depends on the state for times $t' < t$. This is because we only wish to hit the target, we do not have to reach the target and stay there.

b) Rope Manipulation: The environmental cost is the distance to the goal, computed by considering the centre of the rope to be a floating point, discretizing the 3D environment into a 8-connected graph and solve for the shortest path to the goal for every point in the graph. We do not penalize contact for the baselines, as we found that they could exploit contact to help complete the task. The cost for LVSPC penalizes contact (because the simple model is rigid), where we do a collision-check for the rigid-link approximation. The cost functions are $c(z_t, \sigma_t^z) = \texttt{distToGoal}_{\mathcal{Z}} + \beta\sigma_t^z + 100\texttt{checkCollision}(z_t)$ and $c_{\text{env}}(x_t) = \texttt{distToGoal}_{\mathcal{X}}$.

To balance exploiting vs. exploring we increase β from 0 to 1000 in the first 10 episodes.

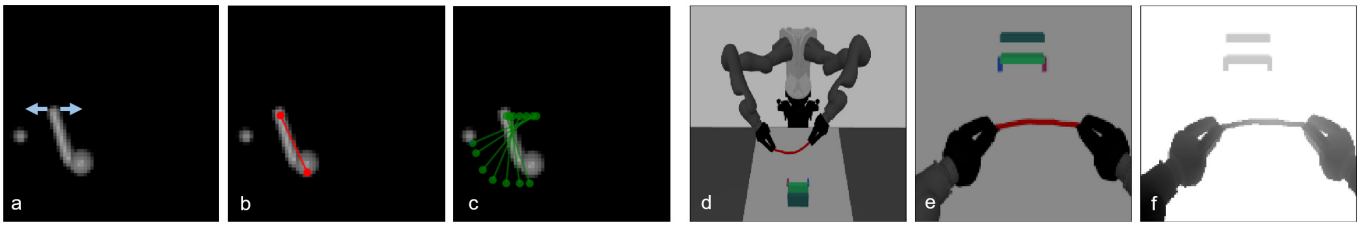

Fig. 4: *(a)* Tethered mass input image (64x64 grayscale) with the target (left) and the single prismatic joint (blue); *(b)* output from CNN ensemble and GPUKF estimation (red); *(c)* planned trajectory from MPPI (green). Only the first action from this trajectory is executed before replanning; *(d)* The rope manipulation environment. The goal is to bring the centre of the rope to the centre of the narrow gap. The sides of the gap have protrusions which can catch the rope; *(e, f)* Example RGB and D observations from overhead Kinect.

E. Network Architectures

Networks are implemented in PyTorch [31], and the GPs are implemented in GPytorch [32], which allows us to exploit parallelism on the GPU for GP inference when performing MPPI. Thus, for the rope manipulation experiment, an iteration of MPPI takes only 0.89s on average using an Intel i7-8700K CPU and an Nvidia 1080Ti GPU. For both experiments we use a CNN ensemble consisting of 10 networks. All convolutional filters have filter size 3×3 and stride 2 for downsampling, all layers other than the output layers use ReLU activations. We use the Adam optimizer with a learning rate of 10^{-3}, except when fine-tuning the pretrained CURL and PlaNet models where we use 10^{-4}.

For the GP dynamics model, we use 200 inducing points. We train an independent GP for each output dimension using the RBF kernel with automatic-relevance determination [33]. We use a learning rate of 10^{-2} to train the GP and perform sysid. For each experiment CURL and PlaNet use encoders with the same architecture as our CNN. The transition and reward models for PlaNet are the same architecture as [6]. The actor-critic architecture for CURL is the same as in [10]. Both CURL and PlaNet are trained end-to-end.

a) Tethered Mass: Each CNN consists first of 4 convolutional layers. There is then a fully connected layer with 2048 hidden units, followed by an output layer.

b) Rope manipulation: Each CNN seperately processes depth and RGB, consisting of an RGB module and a depth module which are combined downstream. Each module consists first of 4 convolutional layers. There is then a fully-connected layer with 512 hidden units. After passing the RGB image through both the RGB module, and the depth image through the depth module, the output from each module is combined and passed through a final hidden layer of 1024 units, followed by an output layer.

F. Results

a) Tethered Mass: An example of the system tracking and MPC is demonstrated in Figure 4. Our statistical results are shown in Figure 5(a, b). PlaNet achieves it's maximum performance at 200-300 episodes and has a success rate of approximately 26% with large variation. We see that CURL shows the highest asymptotic performance, with 97% after 400 episodes. Higher asymptotic performance is typical of model-free learning methods. Pre-training both PlaNet and CURL on data from the simple system results in improved initial performance, but lower final performance. In contrast, LVSPC achieves approximately 90% after 20 episodes, outperforming PlaNet and matching CURL's performance after 200 episodes, demonstrating 10x improved data efficiency. We also see that seeking to learn the dynamics in the simple state space with the GP results in substantially worse performance. This is likely because the simple state representation is insufficient to model the full complex dynamics.

b) Simulated rope manipulation: Our statistical results are shown in Figure 5(c, d). PlaNet's performance after 500 episodes is approximately 30%, while CURL solves the task with almost 100% success rate after 250 episodes. Pre-training CURL on data from the simple system results in improved initial performance, but lower final performance, however pretraining PlaNet led to poor performance which it could not recover from, getting caught on the obstacles in every episode. Our full method achieves 80% success rate after 20 episodes, again equivalent to CURL's performance after 200 episodes (thus we have 10x better data-efficiency) and outperforming PlaNet's final performance. We see that naively treating the rope as a rigid object results in approximately 46% success and almost all failures result from the rope snagging on the protrusions on the side of the gap. As in the tethered mass experiment, attempting to fit the complex dynamics in the simple mode space is ineffective, causing frequent snagging on obstacles.

G. Rope Manipulation on a Real Robot

Our simulation experiments show that LVSPC is effective at transferring within the same simulation environment. To validate that we can still use LVSPC when the simple model and complex environments are very different, we perform the rope manipulation experiment described above on a real robot using a perception system trained only in simulation. We use domain randomization to improve the transfer of the CNN ensemble to real data [34] (see Figure 3(c)). We observed better generalization when we randomized the pose of the camera and trained the CNN ensemble to produce an estimate in the camera frame instead of the world frame.

We perform the experiment on the real robot over 5 random seeds. For each seed, after every 5 episodes we record the success rate on 10 test episodes. The results are shown in Table I. Using LVSPC we can complete this task with 80% success using only 10 episodes of data collected on the real robot. This experiment demonstrates that using LVSPC is promising for real-world tasks, as we only need data from simulation to train an effective perception system.

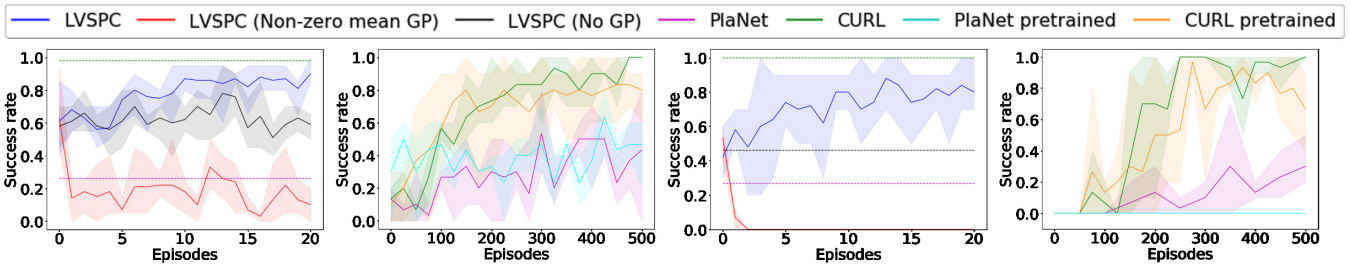

Fig. 5: Average Success over 10 test tasks vs number of episodes for both experiments. Shaded region shows minimum and maximum success rate over 5 runs for LVSPC and ablations and 3 runs for the baselines for a total of 50 and 30 test tasks for LVSPC and the baselines, respectively. *a)* LVSPC and ablations for tethered mass, dotted lines show baseline performance after 500 episodes. *b)* Baselines for tethered mass. *c)* LVSPC and ablations for rope, dotted lines show baseline performance after 500 episodes. *d)* Baselines for rope.

Episode	0	5	10	15	20
Success rate	0.3	0.7	0.8	0.78	0.82

TABLE I: Results over 5 random seeds for real robot experiment

VI. Conclusion

We have presented LVSPC for leveraging a given simple model approximation to improve data efficiency for control tasks on systems with complex dynamics from image observations. We demonstrated this method on two tasks, showing substantially improved performance in the low-data regime over recent reinforcement learning methods. We have also demonstrated that we can apply our framework to a real robot while only using simulated data for pre-training. We assumed that the user specifies a type of simple model, but choosing a simple model which can approximate the complex system is an open problem, made difficult by the requirement that it must be possible to complete the task while operating only in the regime where the simple model is accurate. In future work we intend to incorporate multiple simple models and create a way to decide which is most appropriate.

References

[1] A. Xie, F. Ebert, S. Levine, and C. Finn, "Improvisation through physical understanding: Using novel objects as tools with visual foresight," in *RSS*, 2019.
[2] A. Wang, T. Kurutach, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," in *RSS*, 2019.
[3] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *NeurIPS*, 2016.
[4] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *CoRL*, 2017.
[5] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *ICRA*, 2019.
[6] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *ICML*, 2019.
[7] S. Miller, J. van den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *IJRR*, vol. 31, no. 2, pp. 249–267, 2012.
[8] D. McConachie, T. Power, P. Mitrano, and D. Berenson, "Learning when to trust a dynamics model for planning in reduced state spaces," *RA-L*, vol. 5, no. 2, pp. 3540–3547, April 2020.
[9] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *IJRR*, vol. 20, no. 2, pp. 129–143, 2001.
[10] M. Laskin, A. Srinivas, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," *ICML*, 2020.
[11] J. Ko and D. Fox, "Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models," *AuRo*, vol. 27, pp. 75–90, May 2009.
[12] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *ICRA*, 2017.
[13] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," *ICRA*, 2016.
[14] E. Banijamali, R. Shu, M. Ghavamzadeh, H. H. Bui, and A. Ghodsi, "Robust locally-linear controllable embedding," in *AISTATS*, 2018.
[15] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *ICLR*, 2020.
[16] A. Byravan, F. Leeb, F. Meier, and D. Fox, "SE3-Pose-Nets: Structured Deep Dynamics Models for Visuomotor Control," in *ICRA*, 2018.
[17] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *RA-L*, vol. 5, no. 2, pp. 2372–2379, 2020.
[18] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *Humanoids*, 2014.
[19] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, Aggressive Quadrotor Flight via Reachability-Based Trajectory Design," in *DSCC*, 2019.
[20] D. Mcconachie and D. Berenson, "Estimating model utility for deformable object manipulation using multiarmed bandit methods," *T-ASE*, vol. 15, no. 3, pp. 967–979, July 2018.
[21] M. Deisenroth and C. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *ICML*, 2011.
[22] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *NeurIPS*, 2018.
[23] F. Farshidian and J. Buchli, "Risk sensitive, nonlinear optimal control: Iterative linear exponential-quadratic optimal control with gaussian noise," *arXiv preprint: 1512.07173*, 2015.
[24] S. Bechtle, Y. Lin, A. Rai, L. Righetti, and F. Meier, "Curious ilqr: Resolving uncertainty in model-based rl," in *CoRL*, 2019.
[25] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint 1812.00568*, 2018.
[26] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *AS-SPCC*, 2000, pp. 153–158.
[27] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *NeurIPS*, 2017.
[28] R. Frigola, Y. Chen, and C. E. Rasmussen, "Variational gaussian process state-space models," in *NeurIPS*, 2014.
[29] M. Jankowiak, G. Pleiss, and J. R. Gardner, "Parametric gaussian process regressors," in *ICML*, 2020.
[30] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.
[31] *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019.
[32] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," in *NeurIPS*, 2018.
[33] R. M. Neal, *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996.
[34] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS*, 2017.