Data-efficient Domain Randomization with Bayesian Optimization

Fabio Muratore[1,2] and Christian Eilers[1,2] and Michael Gienger[2] and Jan Peters[1]

***Abstract*—When learning policies for robot control, the required real-world data is typically prohibitively expensive to acquire, so learning in simulation is a popular strategy. Unfortunately, such polices are often not transferable to the real world due to a mismatch between the simulation and reality, called 'reality gap'. Domain randomization methods tackle this problem by randomizing the physics simulator (source domain) during training according to a distribution over domain parameters in order to obtain more robust policies that are able to overcome the reality gap. Most domain randomization approaches sample the domain parameters from a fixed distribution. This solution is suboptimal in the context of sim-to-real transferability, since it yields policies that have been trained without explicitly optimizing for the reward on the real system (target domain). Additionally, a fixed distribution assumes there is prior knowledge about the uncertainty over the domain parameters. In this paper, we propose Bayesian Domain Randomization (BayRn), a black-box sim-to-real algorithm that solves tasks efficiently by adapting the domain parameter distribution during learning given sparse data from the real-world target domain. BayRn uses Bayesian optimization to search the space of source domain distribution parameters such that this leads to a policy which maximizes the real-word objective, allowing for adaptive distributions during policy optimization. We experimentally validate the proposed approach in sim-to-sim as well as in sim-to-real experiments, comparing against three baseline methods on two robotic tasks. Our results show that BayRn is able to perform sim-to-real transfer, while significantly reducing the required prior knowledge.**

***Index Terms*—Reinforcement Learning, Transfer Learning**

I. Introduction

PHYSICS simulations provide a possibility of generating vast amounts of diverse data at a low cost. However, sample-based optimization has been known to be optimistically biased [1], which means that the found solution appears to be better than it actually is. The problem is worsened when the data used for optimization does not originate from the same environment, also called domain. In this case, we observe a simulation optimization bias, which leads to an overestimation of the policy's performance [2]. Generally, there are two ways to overcome the gap between simulation and reality. One can improve the generative model to closely match the reality, e.g. by using system identification. Increasing the model's accuracy has the advantage of leading to controllers with potentially higher performance, since the learner can focus on a single domain. On the downside, this goes in line with a reduced transferability of the found policy, which is caused by the previously mentioned optimistic bias, and aggravated if the model does not include all physical phenomena. Moreover, we might face a situation where it is not affordable to improve the model. Alternatively, one can add variability to the generative model, e.g. by turning the physics simulator's parameters into random variables. Learning from randomized simulations poses a harder problem for the learner due to the additional variability of the observed data. But the recent successes in the field of sim-to-real transfer argue for domain randomization being a promising method [3, 4].

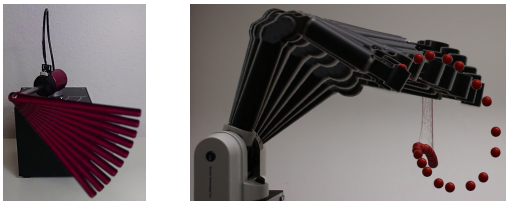


Figure 1: Evaluation platforms: (left) underactuated swing-up and balance task on the Quanser Furuta pendulum, (right) ball-in-a-cup task on the Barrett WAM robotic arm.

State-of-the-art approaches commonly randomize the simulator according to a static handcrafted distribution [5, 6, 7, 8]. Even though static randomization is in many cases sufficient to cross the reality gap, it is desirable to automate the process as far as possible. Moreover, using a fixed distribution does not allow to update the prior knowledge or incorporate the uncertainty over domain parameters. Most importantly, closing the feedback loop over the real system will lead to policies with higher performance on the target domain since the feedback enables the optimization of the domain parameter distribution. However, approaches which adapt an distribution over simulators, yield to additional challenges. For example algorithms that intertwine system identification and policy optimization, e.g., [4, 9], introduce a circular dependency since both subroutines depend on the sensible outputs of the other. One possible failure case is a policy which does not excite the system well enough, resulting in bad updates the simulator's parameters. The sim-to-real algorithm presented in this paper does not require any system identification.

Contributions: We advance the state-of-the-art by introducing Bayesian Domain Randomization (BayRn), a method which is able to close the reality gap by learning from randomized simulations and adapting the distribution over simulator parameters based solely on real-world returns. The use usage of Bayesian Optimization (BO) for sampling the next training environment makes BayRn sample efficient w.r.t. real-world

Manuscript received: October, 15, 2020; Revised December, 12, 2020; Accepted January, 05, 2021.

[1]Fabio Muratore, Christian Eilers and Jan Peters are with the Intelligent Autonomous Systems Group, Technical University Darmstadt, Germany. fabio@robot-learning.de

[2]Fabio Muratore, Christian Eilers and Michael Gienger are with the Honda Research Institute Europe, Offenbach am Main, Germany.

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

data. The proposed algorithm can be seen as a way to vastly automate the finding of a source domain distribution in sim-to-real settings, which is typically done by trial and error. We validate our approach by conducting a sim-to-sim as well as two sim-to-real experiments on an underactuated nonlinear swing-up task, and on a ball-in-a-cup task (Figure 1). The sim-to-sim setup examines the domain parameter adaptation mechanism of BayRn, and shows that the belief about the domain distribution parameters converges to a specified ground truth parameter set. In the sim-to-real experiments, we compare the performance of policies trained with BayRn against multiple baselines based on a total number of 700 real-world rollouts. Moreover, we demonstrate that BayRn is able to work with step-based as well as episodic Reinforcement Learning (RL) algorithms as policy optimization subroutines.

The remainder of this paper is organized as follows: first, we introduce the necessary fundamentals (Section II) for BayRn (Section III). Next, we evaluate the devised method experimentally (Section IV). Subsequently, we put BayRn into context with the related work (Section V). Finally, we conclude and mention possible future research directions (Section VI).

II. Background and Notation

Optimizing control policies for Markov Decision Processes (MDPs) with unknown dynamics is generally a hard problem (Section II-A). It is specifically hard due to the simulation optimization bias [2], which occurs when transferring the polices learned in one domain to another. Adapting the source domain based on real-world data requires a method suited for expensive objective function evaluations. BO is a prominent choice for these kind of problems (Section II-B).

A. Markov Decision Process

Consider a time-discrete dynamical system

$$\boldsymbol{s}_{t+1} \sim \mathcal{P}_{\boldsymbol{\xi}}\left(\boldsymbol{s}_{t+1} \middle| \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\xi}\right), \quad \boldsymbol{s}_0 \sim \mu_{0,\boldsymbol{\xi}}(\boldsymbol{s}_0 | \boldsymbol{\xi}),$$
$$\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}), \quad \boldsymbol{\xi} \sim \nu(\boldsymbol{\xi}; \boldsymbol{\phi}),$$

with the continuous state $\boldsymbol{s}_t \in \mathcal{S}_{\boldsymbol{\xi}} \subseteq \mathbb{R}^{n_s}$, and continuous action $\boldsymbol{a}_t \in \mathcal{A}_{\boldsymbol{\xi}} \subseteq \mathbb{R}^{n_a}$ at time step t. The environment, also called domain, is instantiated through its parameters $\boldsymbol{\xi} \in \mathbb{R}^{n_\xi}$ (e.g., masses, friction coefficients, or time delays), which are assumed to be random variables distributed according to the probability distribution $\nu\colon \mathbb{R}^{n_\xi} \to \mathbb{R}^+$ parametrized by $\boldsymbol{\phi}$. These parameters determine the transition probability density function $\mathcal{P}_{\boldsymbol{\xi}}\colon \mathcal{S}_{\boldsymbol{\xi}} \times \mathcal{A}_{\boldsymbol{\xi}} \times \mathcal{S}_{\boldsymbol{\xi}} \to \mathbb{R}^+$ that describes the system's stochastic dynamics. The initial state $\boldsymbol{s}_0$ is drawn from the start state distribution $\mu_{0,\boldsymbol{\xi}}\colon \mathcal{S}_{\boldsymbol{\xi}} \to \mathbb{R}^+$. Together with the reward function $r\colon \mathcal{S}_{\boldsymbol{\xi}} \times \mathcal{A}_{\boldsymbol{\xi}} \to \mathbb{R}$, and the temporal discount factor $\gamma \in [0, 1]$, the system forms a MDP described by the set $\mathcal{M}_{\boldsymbol{\xi}} = \{\mathcal{S}_{\boldsymbol{\xi}}, \mathcal{A}_{\boldsymbol{\xi}}, \mathcal{P}_{\boldsymbol{\xi}}, \mu_{0,\boldsymbol{\xi}}, r, \gamma\}$. The goal of a Reinforcement Learning (RL) agent is to maximize the expected (discounted) return, a numeric scoring function which measures the policy's performance. The expected discounted return of a stochastic domain-independent policy $\pi(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta})$, characterized by its parameters $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^{n_\theta}$, is defined as

$$J(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{s}_0) = \mathbb{E}_{\boldsymbol{\tau} \sim p(\boldsymbol{\tau})}\Big[\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big| \boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{s}_0\Big].$$

While learning from experience, the agent adapts its policy parameters. The resulting state-action-reward tuples are collected in trajectories, a.k.a. rollouts, $\boldsymbol{\tau} = \{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t\}_{t=0}^{T-1}$, with $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. To keep the notation concise, we omit the dependency on $\boldsymbol{s}_0$.

B. Bayesian Optimization with Gaussian Processes

Bayesian Optimization (BO) is a sequential derivative-free global optimization strategy, which tries to optimize an unknown function $f\colon \mathcal{X} \to \mathbb{R}$ on a compact set $\mathcal{X}$ [10]. In order to do so, BO constructs a probabilistic model, typically a Gaussian Process (GP), for f. GPs are distributions over functions $f \sim \mathcal{GP}(m, k)$ defined by a prior mean $m\colon \mathcal{X} \to \mathbb{R}$ and positive definite covariance function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ called kernel. This probabilistic model is used to make decisions about where to evaluate the unknown function next. A distinctive feature of BO is to use the complete history of noisy function evaluations $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=0}^n$ with $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \sim \mathcal{N}\left(y \middle| f(\boldsymbol{x}_i), \varepsilon\right)$ where ε is the variance of the observation noise. The next evaluation candidate is then chosen by maximizing a so-called acquisition function $a\colon \mathcal{X} \to \mathbb{R}$, which typically balances exploration and exploitation. Prominent acquisition functions are Expected Improvement and Upper Confidence Bound. Through the use of priors over functions, BO has become a popular choice for sample-efficient optimization of black-box functions that are expensive to evaluate. Its sample efficiency plays well with the algorithm introduced in this paper where a GP models the relation between domain distribution's parameters and the resulting policy's return estimated from real-world rollouts, i.e. $\boldsymbol{x} \equiv \boldsymbol{\phi}$ and $y \equiv \hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)$. For further information on BO and GPs, we refer the reader to [10] as well as [11].

III. Bayesian Domain Randomization (BayRn)

The problem of source domain adaptation based on returns from the target domain can be expressed in a bilevel formulation

$$\boldsymbol{\phi}^\star = \arg\max_{\boldsymbol{\phi} \in \Phi} J^{\text{real}}(\boldsymbol{\theta}^\star(\boldsymbol{\phi})) \quad \text{with} \tag{1}$$

$$\boldsymbol{\theta}^\star(\boldsymbol{\phi}) = \arg\max_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{\boldsymbol{\xi} \sim \nu(\boldsymbol{\xi}; \boldsymbol{\phi})}[J(\boldsymbol{\theta}, \boldsymbol{\xi})], \tag{2}$$

where we refer to (1) and (2) as the upper and lower level optimization problem respectively. Thus, the two equations state the goal of finding the set of domain distribution parameters $\boldsymbol{\phi}^\star$ that maximizes the return on the real-world target system $J^{\text{real}}(\boldsymbol{\theta}^\star(\boldsymbol{\phi}))$, when used to specify the distribution $\nu(\boldsymbol{\xi}; \boldsymbol{\phi})$ during training in the source domain. The space of domain parameter distributions is represented by Φ. In the following, we abbreviate $\boldsymbol{\theta}^\star(\boldsymbol{\phi})$ with $\boldsymbol{\theta}^\star$. At the core of BayRn, first a policy optimizer, e.g., an RL algorithm, is employed to solve the lower level problem (2) by finding a (locally) optimal policy $\pi(\boldsymbol{\theta}^\star)$ for the current distribution of stochastic environments. This policy is evaluated on the real system for n_τ rollouts, providing an estimate of the return $\hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)$. Next, the upper level problem (1) is solved using BO, yielding a new domain parameter distribution which is used to randomize the simulator. In this process the relation between the domain

Algorithm 1: Bayesian Domain Randomization

input : domain parameter distribution $\nu(\boldsymbol{\xi};\boldsymbol{\phi})$, parameter space $\Phi = [\boldsymbol{\phi}_{\min}, \boldsymbol{\phi}_{\max}]$, algorithm `PolOpt`, Gaussian Process $\mathcal{GP}$, acquisition function a, hyper-parameters n_{init}, n_τ, J^{succ}

output: maximum a posteriori domain distribution parameter $\boldsymbol{\phi}^\star$ and policy $\pi(\boldsymbol{\theta}^\star)$

1 Initialize empty data set and n_{init} policies randomly
2 $\mathcal{D} \leftarrow \{\}$; $\pi\big(\boldsymbol{\theta}_{1:n_{\text{init}}}\big) \leftarrow \boldsymbol{\theta}_{1:n_{\text{init}}} \sim \Theta$
3 Sample n_{init} source domain distribution parameter sets and train in randomized simulators
4 $\boldsymbol{\phi}_{1:n_{\text{init}}} \leftarrow \boldsymbol{\phi}_{1:n_{\text{init}}} \sim \Phi$
5 $\boldsymbol{\theta}^\star_{1:n_{\text{init}}} \leftarrow \texttt{PolOpt}\big[\pi\big(\boldsymbol{\theta}_{1:n_{\text{init}}}\big), \nu(\boldsymbol{\xi};\boldsymbol{\phi}_{1:n_{\text{init}}})\big]$
6 Evaluate the n_{init} policies on the target domain for n_τ rollouts and estimate the return
7 $\hat{J}^{\text{real}}\big(\boldsymbol{\theta}^\star_{1:n_{\text{init}}}\big) \leftarrow 1/n_\tau \sum_{j=1}^{n_\tau} J_j^{\text{real}}\big(\boldsymbol{\theta}^\star_{1:n_{\text{init}}}\big)$
8 Augment the data set and update the GP's posterior distribution
9 $\mathcal{D} \cup \{\boldsymbol{\phi}_i, \hat{J}^{\text{real}}(\boldsymbol{\theta}^\star_i)\}_{i=1}^{n_{\text{init}}}$; $\mathcal{GP}(m,k) \leftarrow \mathcal{GP}\big(m,k\big|\mathcal{D}\big)$
10 **do** ▷ `Sim-to-real loop`
11 Optimize the GP's acquisition function
12 $\boldsymbol{\phi}^\star \leftarrow \arg\max_{\boldsymbol{\phi}\in\Phi} a(\boldsymbol{\phi}, \mathcal{D})$
13 Train a policy using the obtained domain distribution parameter set
14 $\boldsymbol{\theta}^\star \leftarrow \texttt{PolOpt}[\pi(\boldsymbol{\theta}), \nu(\boldsymbol{\xi};\boldsymbol{\phi}^\star)]$
15 Evaluate the policy on the target domain for n_τ rollouts and estimate the return
16 $\hat{J}^{\text{real}}(\boldsymbol{\theta}^\star) \leftarrow 1/n_\tau \sum_{j=1}^{n_\tau} J_j^{\text{real}}(\boldsymbol{\theta}^\star)$
17 Augment the data set and update the GP's posterior distribution
18 $\mathcal{D} \cup \{\boldsymbol{\phi}^\star, \hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)\}$; $\mathcal{GP}(m,k) \leftarrow \mathcal{GP}\big(m,k\big|\mathcal{D}\big)$
19 **while** $\hat{J}^{real}(\boldsymbol{\theta}^\star) < J^{succ}$ *and* $n_{iter} \leq n_{iter,max}$
20 Train the maximum a posteriori policy (repeat the Lines 12 and 14 once)

distribution's parameters $\boldsymbol{\phi}$ and the resulting policy's return on the real system $\hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)$ is modeled by a GP. The GP's mean and covariance is updated using all recorded inputs $\boldsymbol{\phi}$ and the corresponding observations $\hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)$. Finally, BayRn terminates when the estimated performance on the target system exceeds J^{succ} which is the task-specific success threshold. Since the GP requires at least a few (about 5 to 10) samples to provide a meaningful posterior, BayRn has an initialization phase before the loop. In this phase, n_{init} source domains are randomly sampled from Φ, and subsequently for each of these domains a policy is trained. After evaluating the n_{init} initial policies, the GP is fed with the inputs $\boldsymbol{\phi}_{1:n_{\text{init}}}$ and the corresponding observations $\hat{J}^{\text{real}}\big(\boldsymbol{\theta}^\star_{1:n_{\text{init}}}\big)$. The complete BayRn procedure is summarized in Algorithm 1. In principle, there are no restrictions to the choice of algorithms for solving the two stages (1) and (2). For training the GP, we used the BO implementation from BoTorch [12] which expects normalized inputs and standardized outputs. Notably, we decided for the expected improvement acquisition function and a zero-mean GP prior with a Matérn 5/2 Kernel.

Connection to System Identification: Unlike related methods (Section V), BayRn does not include a term in the objective function that drives the system parameters to match the observed dynamics. Instead, the BO component in BayRn is free to adapt the domain distribution parameters $\boldsymbol{\phi}$ (e.g., mean or standard deviation of a body's mass) while learning in simulation such that the resulting policies perform well in the target domain. This can be seen as an indirect system identification, since with increasing iteration count the BO process will converge to sampling from regions with high real-world return. There is a connection to control as inference approaches which interpret the cost as a log-likelihood function under an optimality criterion using a Boltzmann distribution construct [13, 14]. Regarding BayRn, the sequence of sampled domain distribution parameter sets strongly depends on the acquisition function and the complexity of the given problem. We argue that excluding system identification from the upper level objective (1) is sensible for the presented sim-to-real algorithm, since it learns from a randomized physics simulator, hence attenuates the benefit of a well-fitted model.

IV. Experiments

We study Bayesian Domain Randomization (BayRn) on two different platforms: 1) an underactuated rotary inverted pendulum, also known as Furuta pendulum, with the task of swinging up the pendulum pole into an upright position, and 2) the tendon-driven 4-DoF robot arm WAM from Barrett, where the agent has to swing a ball into a cup mounted as the end-effector. First, we set up a simplified sim-to-sim experiment on the Furuta pendulum to check if the proposed algorithm's belief about the domain distribution parameters converges to a specified set of ground truth values. Next, we evaluate BayRn as well as the baseline methods SimOpt [4], Uniform Domain Randomization (UDR), and Proximal Policy Optimization (PPO) [15] or Policy learning by Weighting Exploration with the Returns (PoWER) [16] in two sim-to-real experiments. Additional details on the system description can be found in Appendix A. Furthermore, an extensive list of the chosen hyper-parameters can be found in Appendix B. A video demonstrating the sim-to-real transfer of the policies learned with BayRn can be found at `www.ias.informatik.tu-darmstadt.de/Team/FabioMuratore`. Moreover, the source code of BayRn and the baselines is available at [17].

A. Experimental Setup

All rollouts on the Furuta pendulum ran for 6 s at 100 Hz, collecting 600 time steps with a reward $r_t \in]0, 1]$. We decided to use a Feedforward Neural Network (FNN) policy in combination with PPO as policy optimization (sub)routine (Table IIa). Before each rollout, the platform was reset automatically. On the physical system, this procedure includes estimating the sensors' offsets as well as running a controller which drives the

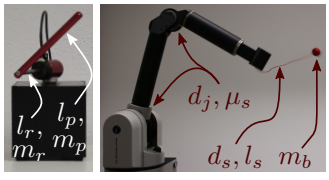


Figure 2: Platforms with annotated domain parameters

Table I: Range of domain distribution parameter values ϕ used during the experiments. All domain parameters were randomized such that they stayed physically plausible.

(a) swing-up and balance

Parameter	Range	Unit
pendulum pole mass mean	$\mathbb{E}[m_p] \in [0.0192, 0.0288]$	kg
pendulum pole mass var.	$\mathbb{V}[m_p] \in [5.76\text{e}{-10}, 5.76\text{e}{-6}]$	kg^2
rotary pole mass mean	$\mathbb{E}[m_r] \in [0.076, 0.114]$	kg
rotary pole mass var.	$\mathbb{V}[m_r] \in [9.03\text{e}{-9}, 9.03\text{e}{-5}]$	kg^2
pendulum pole length mean	$\mathbb{E}[l_p] \in [0.1032, 0.1548]$	m
pendulum pole length var.	$\mathbb{V}[l_p] \in [1.66\text{e}{-8}, 1.66\text{e}{-4}]$	m^2
rotary pole length mean	$\mathbb{E}[l_r] \in [0.068, 0.102]$	m
rotary pole length var.	$\mathbb{V}[l_r] \in [7.23\text{e}{-9}, 7.23\text{e}{-5}]$	m^2

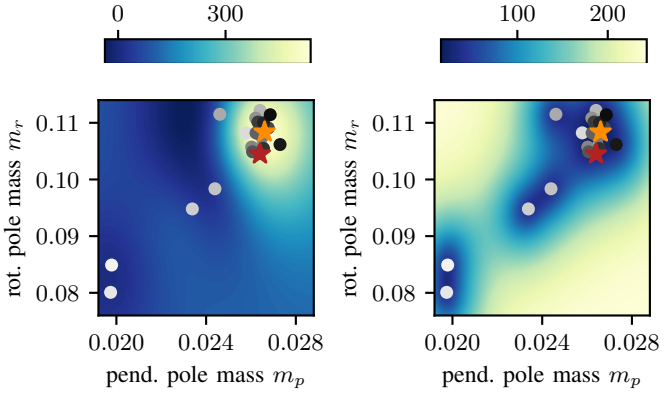
(b) ball-in-a-cup

Parameter	Range	Unit
string length mean	$\mathbb{E}[l_s] \in [0.285, 0.315]$	m
string length variance	$\mathbb{V}[l_s] \in [9\text{e}{-8}, 2.25\text{e}{-4}]$	m^2
string damping mean	$\mathbb{E}[d_s] \in [0, 2\text{e}{-4}]$	N/s
string damping variance	$\mathbb{V}[d_s] \in [3.33\text{e}{-13}, 8.33\text{e}{-10}]$	N^2/s^2
ball mass mean	$\mathbb{E}[m_b] \in [0.0179, 0.0242]$	kg
ball mass variance	$\mathbb{V}[m_b] \in [4.41\text{e}{-10}, 4.41\text{e}{-6}]$	kg^2
joint damping mean	$\mathbb{E}[d_j] \in [0.0, 0.1]$	N/s
joint damping variance	$\mathbb{V}[d_j] \in [3.33\text{e}{-8}, 2.08\text{e}{-4}]$	N^2/s^2
joint stiction mean	$\mathbb{E}[\mu_s] \in [0, 0.4]$	–
joint stiction variance	$\mathbb{V}[\mu_s] \in [1.33\text{e}{-6}, 3.33\text{e}{-3}]$	–

device to its initial position with the rotary pole centered and the pendulum hanging down. In simulation, the reset function causes the simulator to sample a new set of domain parameters $\boldsymbol{\xi}$ (Figure 2). Due to the underactuated nature of the dynamics, the pendulum has to be swung back and forth to put energy into the system before being able to swing the pendulum up.

The Barrett WAM was operated at 500 Hz with an episode length of 3.5 s, i.e., 1750 time steps. For the ball-in-cup task, we chose a RBF-policy commanding desired deltas to the current joint angles and angular velocities, which are passed to the robots feed-forward controller. Hence, the only input to the policy is the normalized time. At the beginning of each rollout, the robot is driven to an initial position. When evaluating on the physical platform, the ball needs to be manually stabilized in a resting position. Once the rollout has finished, the operator enters a return value (Appendix A).

In the sim-to-real experiments, we compare BayRn to SimOpt, UDR, and PPO or PoWER. For every algorithm, we train 20 polices and execute 5 evaluation rollouts per policy. PPO as well as PoWER are set up to learn from simulations where the domain parameters are given by the platforms' data sheets or CAD models. These sets of domain parameters are called nominal. Hence, PPO and PoWER serve as a baseline representing step-based and episodic RL algorithms without domain randomization or any real-world data. UDR augments an RL algorithm, here PPO or PoWER, and can be seen as the straightforward way of randomizing a simulator, as done in [7]. Each domain parameter ξ is assigned to an independent probability distribution, specified by its parameters $\boldsymbol{\phi}$, i.e. mean and variance, (Table I). Thus, we include UDR as a baseline method for static domain randomization. Note that UDR can, in contrast to BayRn and SimOpt, be easily parallelized which reduces the time to train a policy significantly. With SimOpt, Y. Chebotar et al. [4] presented a trajectory-based framework



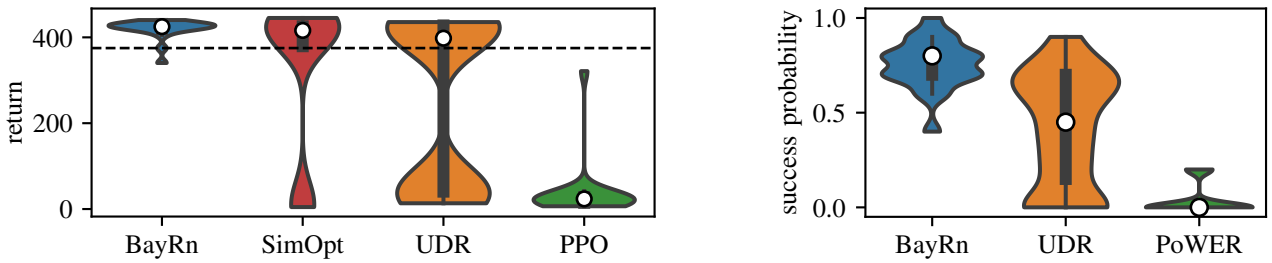
(a) posterior mean (b) posterior std.

Figure 3: Target domain returns (a) and the associated standard deviation (b) modeled by the GP learned with BayRn in a sim-to-sim setting (brighter is higher). The ground truth domain parameters as well as the maximum a posteriori domain distribution parameters found by BayRn are displayed as a red and orange star, respectively. The circles mark the sequence of domain parameter configurations (darker is later).

for closing the reality gap, and validated it on two state-of-the-art sim-to-real robotic manipulation tasks. SimOpt iteratively adapts the domain parameter distribution's parameters by minimizing discrepancy between observations from the real-world system and the simulation. While BayRn formulates the upper level problem (1) solely based on the real-world returns, SimOpt minimizes a linear combination of the L1 and L2 norm between simulated and real trajectories. Moreover, SimOpt employs Relative Entropy Policy Search (REPS) [18] to update the simulator's parameters, hence turning (1) into an RL problem. The necessity of real-world trajectories renders SimOpt unusable for the ball-in-a-cup task since the feed-forward policy is executed without recording any observations. Thus, there are no real-world trajectories with which to update the simulator. BayRn (Section III), SimOpt and UDR randomize the same domain parameters with identical nominal values. At the beginning of each sim-to-real experiment (Section IV-C), the domain distribution parameters $\boldsymbol{\phi}$ are sampled randomly from their ranges (Table I). The main difference is that BayRn and SimOpt adapt the domain distribution parameters, while UDR does not. We chose normal distributions for masses and lengths as well as uniform distributions for parameters related to friction and damping.

B. Sim-to-sim Results

Before applying BayRn to a physical system, we examine the domain distribution parameter sampling process of the BO component in simulation. In order to provide a (qualitative) visualization, we chose to only randomize the means of the poles' masses, i.e., $\boldsymbol{\phi} = [\mathbb{E}[m_r], \mathbb{E}[m_p]]^\mathsf{T}$. Thus, for this sim-to-sim experiment the domain distribution parameters $\boldsymbol{\phi}$ are synonymous to the domain parameters $\boldsymbol{\xi}$. Apart from that, the hyper-parameters used for executing BayRn are identical to the ones used in the sim-to-real experiments (Appendix B). As stated in Section III, BayRn was designed without an



(a) swing-up and balance (b) ball-in-a-cup

Figure 4: Performance of the different algorithms across both sim-to-real tasks. For each algorithm 20 policies have been trained, varying the random seed, and evaluated 5 times to estimate the mean return per policy (700 rollouts in total). The median performance per algorithm is displayed by white circles, and the inner quartiles are represented by thick vertical bars. A dashed line in (a) marks an approximate threshold where the task is solved, i.e., the rotary pole is stabilized on top in the center. SimOpt was not applicable to our open-loop ball-in-a-cup task (b) because of its requirement for recorded observations.

(explicit) system identification objective. However, we can see from Figure 3a that the maximizer of the GP's mean function $\boldsymbol{\phi}^\star = [0.0266, 0.1084]^\mathsf{T}$ closely match the ground truth parameters $\boldsymbol{\phi}_{\text{GT}} = [0.0264, 0.1045]^\mathsf{T}$. Moreover, Figure 3b displays how the uncertainty about the target domain return is reduced in the vicinity of the sampled parameter configurations. There are two decisive factors for the domain distribution parameter sampling process: the acquisition function (Algorithm 1 Line 12), and the quality of the found policy (Algorithm 1 Line 14). Concerning the latter, a failed training run is indistinguishable to a successful one which fails to transfer to the target domain, since the GP only observes the estimated real-world return $\hat{J}^{\text{real}}(\boldsymbol{\theta}^\star)$.

C. Sim-to-real Results

Figure 4 visualizes the results of the sim-to-real experiment described in Section IV-A. The discrepancy between the performance of PPO and PoWER and the other algorithms reveals that domain randomization was the decisive part for sim-to-real transferability. To verify that the PPO and PoWER learned meaningful policies, we checked them in the nominal simulation environments (not reported) and observed that they solve the tasks excellently. In Figure 4a, we see that each median performance of BayRn, SimOpt, and UDR are above the success threshold. However, UDR has a significantly higher variance. SimOpt solves the swing-up and balance task in most cases. However, we noticed that the system identification subroutine can converge to extreme domain distribution parameters, rendering the next policy useless, which then yields a collection of poor trajectories for the next system identification, resulting in a downward spiral. BayRn on the other side relies on the policy optimizer's ability to robustly solve the simulated environment (Section IV-B). This problem can be mediated by re-running the policy optimization in case a certain return threshold in simulation has not been exceeded. For the ball-in-a-cup task, Figure 4b shows an improvement of sim-to-real transfer for BayRn, especially since the tasks open-loop design amplifies domain mismatch. During the experiments, we noticed that UDR sometimes failed unexpectedly. We suspect the a high dependency on the initial state to be the reason for that.

Comparing the Furuta pendulum's nominal domain parameters $\boldsymbol{\phi}_{\text{nom}} = [m_p, m_r, l_p, l_r]^\mathsf{T} = [0.024, 0.095, 0.129, 0.085]^\mathsf{T}$ to the means among BayRn's final estimate $\boldsymbol{\phi}^\star_{\text{mean}} = [0.023, 0.098, 0.123, 0.087]^\mathsf{T}$, we see that the domain parameters' means changed by less than 10 % each. Complementary the variances among BayRn's final estimate are $\boldsymbol{\phi}^\star_{\text{var}} = [6.29\text{e}{-8}, 5.67\text{e}{-6}, 4.10\text{e}{-5}, 1.19\text{e}{-5}]^\mathsf{T}$, indicating a higher uncertainty on the link lengths (relative to the means). Thus, the final domain parameters are well within the boundaries of the BO search space (Table I). In combination, these small differences result in significantly different system dynamics. We believe this to be the reason why the baselines without domain randomization completely failed to transfer.

V. Related Work

We divide the related research on robot reinforcement learning from randomized simulations into approaches which use static (Section V-A) or adaptive (Section V-B) distributions for sampling the physics parameters. Bayesian Domain Randomization (BayRn) as introduced in Section III belongs to the second category.

A. Domain Randomization with Static Distributions

Learning from a randomized simulator with fixed domain parameter distributions has bridged the reality gap in several cases [3, 6, 2]. Most prominently, the robotic in-hand manipulation reported in [3] showed that domain randomization in combination with careful model engineering and the usage of recurrent neural networks enables direct sim-to-real transfer on an unprecedented difficulty level. Similarly, Lowrey et al. [6] employed Natural Policy Gradient to learn a continuous controller for a positioning task, after carefully identifying the system's parameters. Their results show that the policy learned from the identified model was able to perform the sim-to-real transfer, but the policies learned from an ensemble of models were more robust to modeling errors. Mordatch et al. [5] used finite model ensembles to run trajectory optimization on a small-scale humanoid robot. In contrast, Peng et al. [7] combined model-free RL with recurrent neural network policies trained on experience replay in order to push an object

by controlling a robotic arm. The usage of risk-averse objective function has been explored on MuJoCo tasks in [19]. The authors also provide a Bayesian point of view.

Cully et al. [20] can be seen as an edge case of static and adaptive domain randomization, where a large set of policies is learned before execution on the physical robot and evaluated in simulation. Every policy is associated to one configuration of the so-called behavioral descriptors, which are related but not identical to domain parameters. In contrast to BayRn, there is no policy training after the initial phase. Instead of retraining or fine-tuning, the algorithm suggested in [20] reacts to performance drops, e.g. due to damage, by using BO to sequentially select a pretrained policy and measure its performance on the robot. The underlying GP models the mapping from behavior space to performance. This method demonstrated impressive damage recover abilities on a robotic locomotion and a reaching task. However, applying it to RL poses big challenges. Most notably, the number of policies to be learned in order to populate the map, scales exponentially with the dimension of the behavioral descriptors, potentially leading to a very large number of training runs.

Aside from to the previous methods, Muratore et al. [2] propose an approach to estimate the transferability of a policy learned from randomized physics simulations. Moreover, the authors propose a meta-algorithm which provides a probabilistic guarantee on the performance loss when transferring the policy between two domains form the same distribution.

Static domain randomization has also been successfully applied to computer vision problems. A few examples that are: (i) object detection [21], (ii) synthetic object generation for grasp planning [8], and (iii) autonomous drone flight [22].

B. Domain Randomization with Adaptive Distributions

Ruiz et al. [23] proposed the meta-algorithm which is based on a bilevel optimization problem highly similar to the one of BayRn (1, 2). However, there are two major differences. First, BayRn uses Bayesian optimization on the acquired real-wold data to adapt the domain parameter distribution, whereas "learning to simulate" updates the domain parameter distribution using REINFORCE. Second, the approach in [23] has been evaluated in simulation on synthetic data, except for a semantic segmentation task. Thus, there was no dynamics-dependent interaction of the learned policy with the real world.

With SPRL, Klink et al. [24] derived a relative entropy RL algorithm that endows the agent to adapt the domain parameter distribution, typically from easy to hard instances. Hence, the overall training procedure can be interpreted as a curriculum learning problem. The authors were able to solve sim-to-sim goal reaching problems as well as a robotic sim-to-real ball-in-a-cup task, similar to the one in this paper. One decisive difference to BayRn is that the target domain parameter distribution has to be known beforehand.

The approach called Active Domain Randomization (ADR) [25] also formulates the adaption of the domain parameter distribution as an RL problem where different simulation instances are sampled and compared against a reference environment based on the resulting trajectories. This comparison is done by a discriminator which yields rewards proportional to the difficulty of distinguishing the simulated and real environments, hence providing an incentive to generate distinct domains. Using this reward signal, the domain parameters of the simulation instances are updated via Stein Variational Policy Gradient. Mehta et al. [25] evaluated their method in a sim-to-real experiment where a robotic arm had to reach a desired point. The strongest contrast between BayRn and ADR is they way in which new simulation environments are explored. While BayRn can rely on well-studied BO with an adjustable exploration-exploitation behavior, ADR can be fragile since it couples discriminator training and policy optimization, which results in a non-stationary process where distribution of the domains depends on the discriminator's performance.

Paul et al. [26] introduce Fingerprint Policy Optimization which, like BayRn, employs BO to adapt the distribution of domain parameters such that using these for the subsequent training maximizes the policy's return. At first glance the approaches look similar, but there is a major difference in how the upper level problem (1) is solved. Fingerprint Policy Optimization models the relation between the current domain parameters, the current policy and the return of the updated policy with a GP. This design decision requires to feed the policy parameters into the GP which is prohibitively expensive if done straightforwardly. Therefore, abstractions of the policy, so-called fingerprints, are created. These handcrafted features, e.g., the Gaussian approximation of the stationary state distribution, replace the policy to reduce the input dimension. The authors tested Fingerprint Policy Optimization on three sim-to-sim tasks. Contrarily, BayRn has been designed without the need to approximate the policy. Moreover, we validated the presented method in sim-to-real settings.

Yu et al. [9] intertwine policy optimization, system identification, and domain randomization. The proposed method first identifies bounds on the domain parameters which are later used for learning from the randomized simulator. The suggested policy is conditioned on a latent space projection of the domain parameters. After training in simulation, a second system identification step is executed to find the projected domain parameters which maximize the return on the physical robot. This step runs BO for a fixed number of iterations and is similar to solving the upper level problem in (1). The algorithm was evaluated on the bipedal walking robot Darwin OP2.

In Ramos et al. [27], likelihood-free inference in combination with mixture density random Fourier networks is employed to perform a fully Bayesian treatment of the simulator's parameters. Analyzing the obtained posterior over domain parameters, Ramos et al. showed that BayesSim is, in a sim-to-sim setting, able to simultaneously infer different parameter configurations which can explain the observed trajectories. The key difference between BayRn and BayesSim is the objective for updating the domain parameters. While BayesSim maximizes the model's posterior likelihood, BayRn updates the domain parameters such that the policy's return on the physical system is maximized. The biggest advantage of BayRn over BayesSim is its ability to work with very sparse real-world data, i.e. only the scalar return values.

VI. Conclusion

We have introduced Bayesian Domain Randomization (BayRn), a policy search algorithm tailored to crossing the reality gap. At its core, BayRn learns from a randomized simulator while using Bayesian optimization for adapting the source domain distribution during learning. In contrast to previous work, the presented algorithm constructs a probabilistic model of the connection between domain distribution parameters and the policy's return after training with these parameters in simulation. Hence, BayRn only requires little interaction with the real-world system. We experimentally validated that the presented approach is able to solve two nonlinear robotic sim-to-real tasks. Comparing the results against baseline methods showed that adapting the domain parameter distribution lead to policies with higher median performance and less variance. In order to improve the scalability of the Bayesian optimization subroutine to higher numbers of domain distribution parameters, one could for example incorporate quantile Gaussian processes [28], which have shown to scales up to problems with 60-dimensional input.

Acknowledgments

Fabio Muratore gratefully acknowledges the financial support from Honda Research Institute Europe.

Jan Peters received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 640554.

References

[1] B. F. Hobbs and A. Hepenstal, "Is optimization optimistically biased?" *Water Resources Research*, vol. 25, no. 2, pp. 152–160, 1989.

[2] F. Muratore, M. Gienger, and J. Peters, "Assessing transferability from simulation to reality for reinforcement learning," *PAMI*, vol. PP, pp. 1–1, 11 2019.

[3] OpenAI *et al.*, "Learning dexterous in-hand manipulation," *ArXiv eprints*, vol. 1808.00177, 2018.

[4] Y. Chebotar et al., "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *ICRA, Montreal, QC, Canada, May 20-24*, 2019, pp. 8973–8979.

[5] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *IROS, Hamburg, Germany, September 28 - October 2*, 2015, pp. 5307–5314.

[6] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system," in *SIMPAR 2018, Brisbane, Australia, May 16-19*, 2018, pp. 35–42.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *ICRA, Brisbane, Australia, May 21-25*, 2018, pp. 1–8.

[8] J. Tobin *et al.*, "Domain randomization and generative models for robotic grasping," in *IROS, Madrid, Spain, October 1-5*, 2018.

[9] W. Yu, V. C. V. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," in *IROS, Macau, SAR, China, November 3-8*. IEEE, 2019, pp. 3503–3510.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *NIPS, Lake Tahoe, Nevada, United States, December 3-6*, 2012, pp. 2960–2968.

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, ser. Adaptive computation and machine learning. MIT Press, 2006.

[12] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "BoTorch: Programmable bayesian optimization in pytorch," *ArXiv e-prints*, 2019.

[13] M. Toussaint, "Robot trajectory optimization using approximate inference," in *ICML Montreal, Quebec, Canada, June 14-18*, vol. 382, 2009, pp. 1049–1056.

[14] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," in *IJCAI, Beijing, China, August 3-9*, 2013, pp. 3052–3056.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv e-prints*, 2017.

[16] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1-2, pp. 171–203, 2011.

[17] F. Muratore, "SimuRLacra - a framework for reinforcement learning from randomized simulations," https://github.com/famura/SimuRLacra, 2020.

[18] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search," in *AAAI, Atlanta, Georgia, USA, July 11-15*, 2010.

[19] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," in *ICLR, Toulon, France, April 24-26*, 2017.

[20] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[21] J. Tobin *et al.*, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS, Vancouver, BC, Canada, September 24-28*, 2017, pp. 23–30.

[22] F. Sadeghi and S. Levine, "CAD2RL: real single-image flight without a single real image," in *RSS, Cambridge, Massachusetts, USA, July 12-16*, 2017.

[23] N. Ruiz, S. Schulter, and M. Chandraker, "Learning to simulate," *ArXiv e-prints*, vol. 1810.02513, 2018.

[24] P. Klink, H. Abdulsamad, B. Belousov, and J. Peters, "Self-paced contextual reinforcement learning," *ArXiv e-prints*, vol. 1910.02826, 2019.

[25] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *CoRL, Osaka, Japan, October 30 - November 1*, vol. 100. PMLR, 2019, pp. 1162–1176.

[26] S. Paul, M. A. Osborne, and S. Whiteson, "Fingerprint policy optimisation for robust reinforcement learning,"

ArXiv e-prints, vol. 1805.10662, 2018.

[27] F. Ramos, R. Possas, and D. Fox, "Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators," in *RSS, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26*, 2019.

[28] R. Moriconi, K. S. S. Kumar, and M. P. Deisenroth, "High-dimensional bayesian optimization with projections using quantile gaussian processes," *Optim. Lett.*, vol. 14, no. 1, pp. 51–64, 2020.

[29] "mujoco-py," Online. [Online]. Available: https://github.com/openai/mujoco-py

APPENDIX A
MODELING DETAILS ON THE PLATFORMS

The Furuta pendulum (Figure 1) is modeled as an underactuated nonlinear second-order dynamical system given by the solution of

$$\begin{bmatrix} J_r + m_p l_r^2 + \frac{1}{4} m_p l_p^2 (\cos(\alpha))^2 & \frac{1}{2} m_p l_p l_r \cos(\alpha) \\ \frac{1}{2} m_p l_p l_r \cos(\alpha) & J_p + \frac{1}{4} m_p l_p^2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} \tau - \frac{1}{2} m_p l_p^2 \sin(\alpha)\cos(\alpha)\,\dot{\theta}\dot{\alpha} - \frac{1}{2} m_p l_p l_r \sin(\alpha)\,\dot{\alpha}^2 - d_r \dot{\theta} \\ -\frac{1}{4} m_p l_p^2 \sin(\alpha)\cos(\alpha)\,\dot{\theta}^2 - \frac{1}{2} m_p l_p g \sin(\alpha) - d_p \dot{\alpha} \end{bmatrix},$$

with the rotary angle θ and the pendulum angle α, which are defined to be zero when the rotary pole is centered and the pendulum pole is hanging down vertically. While the system's state is defined as $\boldsymbol{s} = [\theta, \alpha, \dot{\theta}, \dot{\alpha}]^\mathsf{T}$, the agent receives observations $\boldsymbol{o} = [\sin(\theta), \cos(\theta), \sin(\alpha), \cos(\alpha), \dot{\theta}, \dot{\alpha}]^\mathsf{T}$. The horizontal pole is actuated by commanding a motor voltage (action) a which regulates the servo motor's torque $\tau = k_m(a - k_m\dot{\theta})/R_m$. One part of the domain parameters is sampled from distributions specified by in Table Ia, while the remaining domain parameters are fixed at their nominal values given in [17]. We formulate the reward function based on an exponentiated quadratic cost

$$r(\boldsymbol{s}_t, a_t) = \exp\left(-\left(\boldsymbol{e}_t^\mathsf{T} \boldsymbol{Q} \boldsymbol{e}_t + a_t R a_t\right)\right) \quad \text{with}$$
$$\boldsymbol{e}_t = \left(\begin{bmatrix} 0 & \pi & 0 & 0 \end{bmatrix} - \boldsymbol{s}_t\right) \bmod 2\pi.$$

Thus, the reward is in range $]0, 1]$ for every time step.

The 4-DoF Barrett WAM (Figure 1) is simulated using MuJoCo, wrapped by mujoco-py [29]. The ball is attached to a string, which is mounted to the center of the cup's bottom plate. We model the string as a concatenation of 30 rigid bodies with two rotational joints per link (no torsion). This specific ball-in-a-cup instance can be considered difficult, since the cups's diameter is only about twice as large as the ball's, and the string is rather short with a length of 30 cm. Similar to the Furuta pendulum, one part of the domain parameters is sampled from distributions specified by in Table Ib, while the remaining domain parameters are fixed at their nominal values given in [17]. Since the feed-forward policy is executed without recording any observations, we define a discrete ternary reward function

$$r(\boldsymbol{s}_T, \boldsymbol{a}_T) = \begin{cases} 1 & \text{if the ball is in the cup,} \\ 0.5 & \text{if the ball hit the cup's upper rim,} \\ 0 & \text{else} \end{cases}$$

where the final reward given by the operator after the rollout ($r(\boldsymbol{s}_t, \boldsymbol{a}_t) = 0$ for $t < T$) when running on the real system. We found the separation in three cases to be helpful during learning and easily distinguishable from the others. While training in simulation, successful trials are identified by detecting a collision between the ball and a virtual cylinder inside the cup. Moreover, we have access to the full state, hence augment the reward function with a cost term that punishes deviations from the initial end-effector position.

APPENDIX B
PARAMETER VALUES FOR THE EXPERIMENTS

Table II lists the hyper-parameters for all training runs during the experiments in Section IV. The reported values have been tuned but not fully optimized.

Table II: Hyper-parameter values for training the policies in Section IV. The domain distribution parameters $\boldsymbol{\phi}$ are listed in Table I.

(a) swing-up and balance

Hyper-parameter	Value
common	
`PolOpt`	PPO
policy / critic architecture	FNN 64-64 with tan-h
optimizer	Adam
learning rate policy	5.97e−4
learning rate critic	3.44e−4
PPO clipping ratio ϵ	0.1
iterations n_{iter}	300
step size Δt	0.01 s
max. steps per episode T	600
min. steps per iteration	$20T$
temporal discount γ	0.9885
adv. est. trade-off factor λ	0.965
success threshold J^{succ}	375
$\boldsymbol{Q}$	diag(2e−1, 1.0, 2e−2, 5e−3)
R	3e−3
real-world rollouts n_τ	5
UDR specific	
min. steps per iteration	$30T$
SimOpt specific	
max. iterations n_{iter}	15
`DistrOpt` population size	500
`DistrOpt` KL bound ϵ	1.0
`DistrOpt` learning rate	5e−4
BayRn specific	
max. iterations $n_{\text{iter,max}}$	15
initial solutions n_{init}	5

(b) ball-in-a-cup

Hyper-parameter	Value
common	
`PolOpt`	PoWER
policy architecture	RBF with 16 basis functions
iterations n_{iter}	20
population size n_{pop}	100
num. importance samples n_{is}	10
init. exploration std σ_{init}	$\pi/12$
min. rollouts per iteration	20
max. steps per episode T	1750
step size Δt	0.002 s
temporal discount γ	1
real-world rollouts n_τ	5
UDR specific	
min. steps per iteration	$30T$
BayRn specific	
max. iterations n_{iter}	15
initial solutions n_{init}	5