Assessing Transferability from Simulation to Reality for Reinforcement Learning

Fabio Muratore, Michael Gienger, *Member, IEEE*, and Jan Peters, *Fellow, IEEE*

Abstract—Learning robot control policies from physics simulations is of great interest to the robotics community as it may render the learning process faster, cheaper, and safer by alleviating the need for expensive real-world experiments. However, the direct transfer of learned behavior from simulation to reality is a major challenge. Optimizing a policy on a slightly faulty simulator can easily lead to the maximization of the 'Simulation Optimization Bias' (SOB). In this case, the optimizer exploits modeling errors of the simulator such that the resulting behavior can potentially damage the robot. We tackle this challenge by applying domain randomization, i.e., randomizing the parameters of the physics simulations during learning. We propose an algorithm called Simulation-based Policy Optimization with Transferability Assessment (SPOTA) which uses an estimator of the SOB to formulate a stopping criterion for training. The introduced estimator quantifies the over-fitting to the set of domains experienced while training. Our experimental results in two different environments show that the new simulation-based policy search algorithm is able to learn a control policy exclusively from a randomized simulator, which can be applied directly to real system without any additional training on the latter.

Index Terms—Reinforcement Learning, Domain Randomization, Sim-to-Real Transfer.

1 Introduction

Exploration-based learning of control policies on physical systems is expensive in two ways. For one thing, real-world experiments are time-consuming and need to be executed by experts. Additionally, these experiments require expensive equipment which is subject to wear and tear. In comparison, training in simulation provides the possibility to speed up the process and save resources. A major drawback of robot learning from simulations is that a simulation-based learning algorithm is free to exploit any infeasibility during training and will utilize the flawed physics model if it yields an improvement during simulation. This exploitation capability can lead to policies that damage the robot when later deployed in the real world. The described problem is exemplary of the difficulties that occur when transferring robot control policies from simulation to reality, which have been the subject of study for the last two decades under the term 'reality gap'. Early approaches in robotics suggest using minimal simulation models and adding artificial i.i.d. noise to the system's sensors and actuators while training in simulation [1]. The aim was to prevent the learner from focusing on small details, which would lead to policies with only marginal applicability. This over-fitting can be described by the Simulation Optimization Bias (SOB), which is similar to the bias of an estimator. The SOB is closely related to the Optimality Gap (OG), which has been used by the optimization community since the 1990s [2, 3], but has not been transferred to robotics or Reinforcement Learning (RL), yet.

Figure 1: Evaluation platforms by Quanser [4]: (left) the 2 DoF Ball-Balancer, (right) the linear inverted pendulum, called Cart-Pole. Both systems are under-actuated nonlinear balancing problems with continuous state and action spaces.

Deep RL algorithms recently demonstrated super-human performance in playing games [5, 6] and promising results in (simulated) robotic control tasks [7, 8, 9]. However, when transferred to real-world robotic systems, most of these methods become less attractive due to high sample complexity and a lack of explainability of state-of-the-art deep RL algorithms. As a consequence, the research field of domain randomization has recently been gaining interest [10, 11, 12, 13, 14, 15, 16, 17]. This class of approaches promises to transfer control policies learned in simulation (source domain) to the real world (target domain) by randomizing the simulator's parameters (e.g., masses, extents, or friction coefficients) and hence train from a set of models instead of just one nominal model. Further motivation to investigate domain randomization is given by the recent successes in robotic sim-to-real scenarios, such as the in-hand manipulation of a cube [18], swinging a peg in a hole, or opening a drawer [17]. The idea of randomizing the simulator's parameters is driven by the fact that the

- *Fabio Muratore and Jan Peters are with the Intelligent Autonomous Systems Group, Technische Universität Darmstadt, Germany. Correspondence to fabio@robot-learning.de*
- *Fabio Muratore and Michael Gienger are with the Honda Research Institute Europe, Offenbach am Main, Germany.*
- *Jan Peters is with the Max Planck Institute for Intelligent Systems, Tübingen, Germany.*

Manuscript received June 21, 2019.

corresponding true parameters of the target domain are unknown. However, instead of relying on an accurate estimation of one fixed parameter set, we take a Bayesian point of view and assume that each parameter is drawn from an unknown underlying distribution. Thereby, the expected effect is an increase in robustness of the learned policy when applied to a different domain. Throughout this paper, we use the term robustness to describe a policy's ability to maintain its performance under model uncertainties. In that sense, a robust control policy is more likely to overcome the reality gap.

Looking at the bigger picture, model-based control (e.g., the linear-quadratic regulator) only considers a system's nominal dynamics parameter values, while robust control (e.g., $\mathcal{H}_\infty$ control) minimizes a system's sensitivity with respect to bounded model uncertainties, thus focuses the worst-case. In contrast to these methods, domain randomization takes the whole range of parameter values into account.

Contributions: we advance the state-of-the-art by

1) introducing a measure for the transferability of a solution, i.e., a control policy, from a set of source distributions to a different target domain from the same distribution,
2) designing an algorithm which, based on this measure, is able to transfer control policies from simulation to reality without any real-world data, and
3) validating the approach by conducting two sim-to-real experiments on under-actuated nonlinear systems.

The remainder of this paper is organized as follows: we explain the necessary fundamentals (Section 2) for the proposed algorithm (Section 3). In particular, we derive the Simulation Optimization Bias (SOB) and the Optimality Gap (OG). After validating the proposed method in simulation, we evaluate it experimentally (Section 4). Next, the connection to related work is discussed (Section 5). Finally, we conclude and discuss possible future research directions (Section 6).

2 Problem Statement and Notation

Optimizing policies for Markov Decision Processes (MDPs) with unknown dynamics is generally a hard problem (Section 2.1). Specifically, this problem is hard due to the simulation optimization bias (Section 2.2), which is related to the optimality gap (Section 2.3). We derive an upper bound on the optimality gap, show its monotonic decrease with increasing number of samples from the random variable. Moreover, we clarify the relationship between the simulation optimization bias and the optimality gap (Section 2.4). In what follows, we build upon the results of [2, 3].

2.1 Markov Decision Process

Consider a time-discrete dynamical system

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

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 the parameters

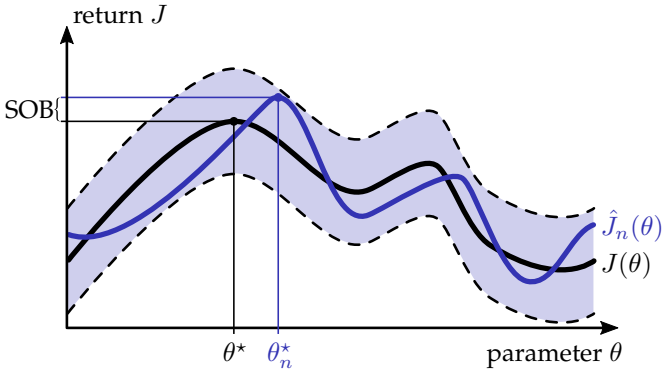


Figure 2: Simulation Optimization Bias (SOB) between the true optimum $\theta^\star$ and the sample-based optimum $\theta_n^\star$. The shaded region visualizes the standard deviation around $J(\theta)$, and $\hat{J}_n(\theta)$ is determined by a particular set of n sampled domain parameters.

$\boldsymbol{\xi} \in \mathbb{R}^{n_\xi}$ (e.g., masses, friction coefficients, or time delays), which are assumed to be random variables distributed according to a probability distribution 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 discount factor $\gamma \in [0, 1]$, the system forms a MDP described by the tuple $\mathcal{M}_{\boldsymbol{\xi}} = \langle \mathcal{S}_{\boldsymbol{\xi}}, \mathcal{A}_{\boldsymbol{\xi}}, \mathcal{P}_{\boldsymbol{\xi}}, \mu_{0,\boldsymbol{\xi}}, r, \gamma \rangle$.

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 policy $\pi_{\boldsymbol{\xi}}(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta})$, characterized by its parameters $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^{n_\theta}$, is defined as

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

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

2.2 Simulation Optimization Bias (SOB)

Augmenting the standard RL (2.1, 2.1) with the concept of domain randomization, i.e. maximizing the expectation of the expected return over all (feasible) realizations of the source domain, leads to the score

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})]$$

that quantifies how well the policy is expected to perform over an infinite set of variations of the nominal domain $\mathcal{M}_{\bar{\boldsymbol{\xi}}}$. When training exclusively in simulation, the true physics model is unknown and the true $J(\boldsymbol{\theta}, \boldsymbol{\xi})$ is thus inaccessible. Instead, we maximize the estimated expected return using a randomized physics simulator. Thereby, we update the policy parameters $\boldsymbol{\theta}$ with a policy optimization algorithm based on samples. The inevitable imperfections of physics simulations will automatically be exploited by any optimization method to achieve a 'virtual' improvement,

Table 1: Definition and of the expectation of the expected (discounted) return, the Simulation Optimization Bias (SOB), the Optimality Gap (OG), and its estimation. All approximations are based on n domains.

Name	Definition	Property
estimated expectation of the expected return	$\hat{J}_n(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i)$	$\mathbb{E}_{\boldsymbol{\xi}}\Big[\hat{J}_n(\boldsymbol{\theta})\Big] = J(\boldsymbol{\theta})$
simulation optimization bias	$\mathrm{b}\Big[\hat{J}_n(\boldsymbol{\theta}_n^\star)\Big] = \mathbb{E}_{\boldsymbol{\xi}}\Big[\max_{\hat{\boldsymbol{\theta}}\in\Theta} \hat{J}_n(\hat{\boldsymbol{\theta}})\Big] - \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}\Big[J(\boldsymbol{\theta}, \boldsymbol{\xi})\Big]$	$\mathrm{b}\Big[\hat{J}_n(\boldsymbol{\theta}_n^\star)\Big] \geq 0$
optimality gap at solution $\boldsymbol{\theta}^c$	$G(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})]$	$G(\boldsymbol{\theta}^c) \geq 0$
estimated optimality gap at solution $\boldsymbol{\theta}^c$	$\hat{G}_n(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c)$	$\hat{G}_n(\boldsymbol{\theta}^c) \geq G(\boldsymbol{\theta}^c)$

i.e., an increase of $J(\boldsymbol{\theta})$, in simulation. To formulate this undesirable behavior, we frame the standard RL problem as a Stochastic Program (SP)

$$J(\boldsymbol{\theta}^\star) = \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})] = \max_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta})\,, \tag{1}$$

with the optimal solution $\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta})$.
The SP above can be approximated using n domains

$$\hat{J}_n(\boldsymbol{\theta}_n^\star) = \max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}\in\Theta} \frac{1}{n}\sum_{i=1}^{n} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i)\,, \tag{2}$$

where the expectation is replaced by the Monte-Carlo estimator over the samples $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$, and $\boldsymbol{\theta}_n^\star = \arg\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta})$ is the solution to the approximated SP.

Sample-based optimization is guaranteed to be optimistically biased if there are errors in the domain parameter estimate, even if these errors are unbiased [2]. Since the proposed method randomizes the domain parameters $\boldsymbol{\xi}$, this assumption is guaranteed to hold. Regarding the SP (1) and its approximation, we define the Simulation Optimization Bias (SOB) to be

$$\mathrm{b}\Big[\hat{J}_n(\boldsymbol{\theta}_n^\star)\Big] = \underbrace{\mathbb{E}_{\boldsymbol{\xi}}\Big[\max_{\hat{\boldsymbol{\theta}}\in\Theta} \hat{J}_n(\hat{\boldsymbol{\theta}})\Big]}_{\text{sample optimum}} - \underbrace{\max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}\Big[J(\boldsymbol{\theta}, \boldsymbol{\xi})\Big]}_{\text{true optimum}} \geq 0. \tag{3}$$

The property $\mathrm{b}[\hat{J}_n(\boldsymbol{\theta}_n^\star)] \geq 0$ follows from the Jensen's inequality. A visualization of the SOB is depicted in Figure 2.

2.3 Optimality Gap (OG)

Intuitively, we want to minimize the SOB in order to achieve the highest transferability of the policy. Since computing the SOB (3) is intractable, the approach presented in this paper is to approximate the Optimality Gap (OG), which relates to the SOB as explained in the Section 2.4.

The OG at the solution candidate $\boldsymbol{\theta}^c$ is defined as

$$G(\boldsymbol{\theta}^c) = J(\boldsymbol{\theta}^\star) - J(\boldsymbol{\theta}^c) \geq 0, \tag{4}$$

where $J(\boldsymbol{\theta}^\star) = \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})]$ is the SP's optimal objective function value and $J(\boldsymbol{\theta}^c) = \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})]$ is the SP's objective function evaluated at the candidate solution [3]. Thus, $G(\boldsymbol{\theta}^c)$ expresses the difference in performance between the optimal policy and the candidate solution at hand. Unfortunately, computing the expectation over infinitely many domains in (4) is intractable. However, we can estimate $G(\boldsymbol{\theta}^c)$ from samples.

2.3.1 Estimation of the Optimality Gap

For an unbiased estimator $\hat{J}_n(\boldsymbol{\theta})$, e.g. a sample average with i.i.d. samples, he have

$$\mathbb{E}_{\boldsymbol{\xi}}\big[\hat{J}_n(\boldsymbol{\theta})\big] = \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})] = J(\boldsymbol{\theta})\,. \tag{5}$$

Inserting (5) into the first term of (4) yields

$$\begin{aligned} G(\boldsymbol{\theta}^c) &= \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}\Big[\hat{J}_n(\boldsymbol{\theta})\Big] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})] \\ &\leq \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta})\right] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})] \end{aligned} \tag{6}$$

as an upper bound on the OG. To compute this upper bound, we use the law of large numbers for the first term and replace the second expectation in (6) with the sample average

$$G(\boldsymbol{\theta}^c) \leq \max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c) = \hat{G}_n(\boldsymbol{\theta}^c)\,, \tag{7}$$

where $\hat{G}_n(\boldsymbol{\theta}^c) \geq 0$ holds.[1] Averaging over a finite set of domains allows for the utilization of an estimated upper bound of the OG as the convergence criterion for the policy search meta-algorithm introduced in Section 3.

2.3.2 Decrease of the Estimated Optimality Gap

The OG decreases in expectation with increasing sample size of the domain parameters $\boldsymbol{\xi}$. The expectation over $\boldsymbol{\xi}$ of the minuend in (7) estimated from $n+1$ i.i.d. samples is

$$\begin{aligned} \mathbb{E}_{\boldsymbol{\xi}}\Big[\hat{J}_{n+1}(\boldsymbol{\theta}_{n+1}^\star)\Big] &= \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \frac{1}{n+1}\sum_{i=1}^{n+1} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i)\right] \\ &= \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \frac{1}{n+1}\sum_{i=1}^{n+1} \frac{1}{n}\sum_{j=1, j\neq i}^{n+1} J(\boldsymbol{\theta}, \boldsymbol{\xi}_j)\right] \\ &\leq \mathbb{E}_{\boldsymbol{\xi}}\left[\frac{1}{n+1}\sum_{i=1}^{n+1} \max_{\boldsymbol{\theta}\in\Theta} \frac{1}{n}\sum_{j=1, j\neq i}^{n+1} J(\boldsymbol{\theta}, \boldsymbol{\xi}_j)\right] \\ &= \mathbb{E}_{\boldsymbol{\xi}}\Big[\hat{J}_n(\boldsymbol{\theta}_n^\star)\Big]\,. \end{aligned} \tag{8}$$

[1] This result is consistent with Theorem 1 and Equation (9) in [3] as well as the "type A error" in [2].

Taking the expectation of the OG estimated from $n+1$ samples $\hat{G}_{n+1}(\boldsymbol{\theta}^c)$ and then plugging in the upper bound from (8), we obtain the upper bound

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{G}_{n+1}(\boldsymbol{\theta}^c)\right] = \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_{n+1}(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c)\right] \leq \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c)\right] = \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{G}_n(\boldsymbol{\theta}^c)\right],$$

which shows that the estimator of the OG in expectation monotonically decreases with increasing sample size.[2]

2.4 Connection Between the SOB and the OG

The SOB can be expressed as the expectation of the difference between the approximated OG and the true OG. Starting from the formulation of the approximated OG in (7), we can take the expectation over the domains on both sides of the inequality and rearrange to

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{G}_n(\boldsymbol{\theta}^c)\right] - G(\boldsymbol{\theta}^c) \geq 0.$$

Using the definitions of $\hat{G}_n(\boldsymbol{\theta}^c)$ and $G(\boldsymbol{\theta}^c)$ from Table 1, the equation above can be rewritten as

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta})\right] - \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_n(\boldsymbol{\theta}^c)\right] - \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta},\boldsymbol{\xi})] + \mathbb{E}_{\boldsymbol{\xi}}\left[J(\boldsymbol{\theta}^c,\boldsymbol{\xi})\right] \geq 0. \tag{9}$$

Since $\hat{J}_n(\boldsymbol{\theta})$ is an unbiased estimator of $J(\boldsymbol{\theta})$, we have

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_n(\boldsymbol{\theta}^c)\right] = \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c,\boldsymbol{\xi})] = J(\boldsymbol{\theta}^c).$$

Hence, the left hand side of (9) becomes

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta})\right] - \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}\left[J(\boldsymbol{\theta},\boldsymbol{\xi})\right] = \mathrm{b}\left[\hat{J}_n(\boldsymbol{\theta}_n^\star)\right],$$

which is equal to the SOB defined in (3). Thus, the SOB is the difference between the expectation over all domains of the estimated OG $\hat{G}_n(\boldsymbol{\theta}^c)$ and the true OG $G(\boldsymbol{\theta}^c)$ at the solution candidate. Therefore, reducing the estimated OG leads to reducing the SOB.

2.5 An Illustrative Example

Imagine we were placed randomly in an environment either on Mars $\boldsymbol{\xi}_M$ or on Venus $\boldsymbol{\xi}_V$, governed by the distribution $\boldsymbol{\xi} \sim \nu(\boldsymbol{\xi};\phi)$. On both planets we are in a catapult about to be shot into the sky exactly vertical. The only thing we can do is to manipulate the catapult, modeled as a linear spring, according to the policy $\pi(\theta)$, i.e. changing the springs extension. Our goal is to minimize the maximum height of the expected flight trajectory $\mathbb{E}_{\boldsymbol{\xi}}[h(\theta,\boldsymbol{\xi})]$ derived from the conservation of energy

$$h(\theta,\boldsymbol{\xi}_i) = \frac{k_i(\theta - x_i)^2}{2mg_i},$$

with mass m, and domain parameters $\boldsymbol{\xi}_i = \{g_i, k_i, x_i\}$ consisting of the gravity acceleration constant, the catapult's spring stiffness, and the catapult's spring pre-extension. The domain parameters are the only quantities specific to Mars and Venus. In this simplified example, we assume that the domain parameters are not drawn from individual distributions, but that there are two sets of domain parameters $\boldsymbol{\xi}_M$ and $\boldsymbol{\xi}_V$ which are drawn from a Bernoulli distribution $\mathcal{B}(\boldsymbol{\xi}|\phi)$ where ϕ is the probability of drawing $\boldsymbol{\xi}_V$. Since minimizing $\mathbb{E}_{\boldsymbol{\xi}}[h(\theta,\boldsymbol{\xi})]$ is identical to maximizing its negative value $J(\theta,\boldsymbol{\xi}) := -\mathbb{E}_{\boldsymbol{\xi}}[h(\theta,\boldsymbol{\xi})]$, we rewrite the problem as

$$J(\theta^\star) = \max_{\theta\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}[J(\theta,\boldsymbol{\xi})].$$

Assume we experienced this situation n times and want to find the policy parameter maximizing the objective above without knowing on which planet we are (i.e., independent of $\boldsymbol{\xi}$). Thus, we approximate $J(\theta^\star)$ by

$$\hat{J}_n(\theta_n^\star) = \max_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} J(\theta,\boldsymbol{\xi}_i).$$

In this Bernoulli experiment, the return of a policy $\pi(\theta)$ fully determined by θ estimates to

$$\hat{J}_n(\theta) = \underbrace{\frac{n_M}{n} J(\theta,\boldsymbol{\xi}_M)}_{\text{proportion Mars}} + \underbrace{\frac{n_V}{n} J(\theta,\boldsymbol{\xi}_V)}_{\text{proportion Venus}}. \tag{10}$$

The optimal policy given the n domains fulfills the necessary condition

$$0 = \nabla_\theta \hat{J}_n(\theta_n^\star) = -\frac{n_M}{n}\frac{k_M(\theta_n^\star - x_M)}{mg_M} - \frac{n_V}{n}\frac{k_V(\theta_n^\star - x_V)}{mg_V}.$$

Solving for the optimal policy parameter yields

$$\theta_n^\star = \frac{x_M n_M k_M g_V + x_V n_V k_V g_M}{n_M k_M g_V + n_V k_V g_M} = \frac{x_M c_M + x_V c_V}{c_M + c_V}, \tag{11}$$

with the (mixed-domain) constants $c_M = n_M k_M g_V$ and $c_V = n_V k_V g_M$. Inserting (11) into (10) gives the optimal return value for n samples

$$\hat{J}_n(\theta^\star) = -\frac{n_M k_M}{2nmg_M}\left(\frac{x_V c_V - x_M c_V}{c_M + c_V}\right)^2 - \frac{n_V k_V}{2nmg_V}\left(\frac{x_M c_M - x_V c_M}{c_M + c_V}\right)^2.$$

Given the domain parameters in Table 2 of Appendix B, we optimize our catapult manipulation policy. This is done in simulation, since real-world trials (being shot with a catapult) would be very costly. Finding the optimal policy in simulation means solving the approximated SP (2), whose optimal solution is denoted by $\theta_n^\star$. We assume that the (stochastic) optimization algorithm outputs a suboptimal solution θ^c. In order to model this property, a policy parameter is sampled in the vicinity of the optimum $\theta^c \sim \mathcal{N}(\theta|\theta_n^\star;\sigma_\theta^2)$ with $\sigma_\theta = 0.15$. During the entire process, the true optimal policy parameter $\theta^\star$ will remain unknown. However, since this simplified example models the domain parameters to be one of two fixed sets ($\boldsymbol{\xi}_M$ or $\boldsymbol{\xi}_V$), $\theta^\star$ can be computed analogously to (11).

The Figures 3 and 4 visualize the evolution of the approximated SP with increasing number of domains n. Key observations are that the objective function value at the candidate solution $\hat{J}_n(\theta^c)$ is less than at the sample-based

[2] This result is consistent with Theorem 2 in [3].

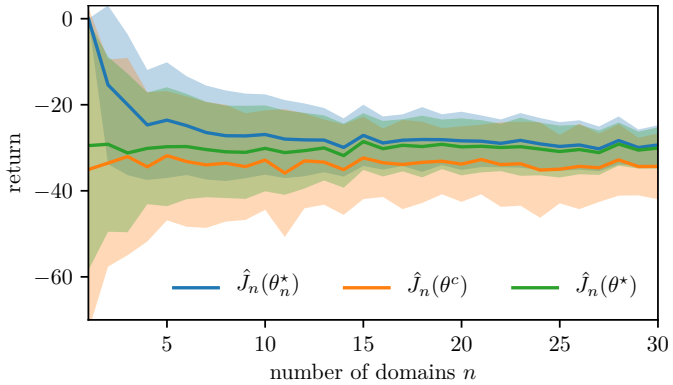


Figure 3: The estimated expected return evaluated using the optimal solution for a set of n domains $\hat{J}_n(\theta^\star_n)$, the candidate solution $\hat{J}_n(\theta^c)$, as well as the true optimal solution $\hat{J}_n(\theta^\star)$. Note that $\hat{J}_n(\theta^\star_n) > \hat{J}_n(\theta^c)$ holds for every instance of the 100 random seeds, even if the standard deviation areas overlap. The shaded areas show ± 1 standard deviation.

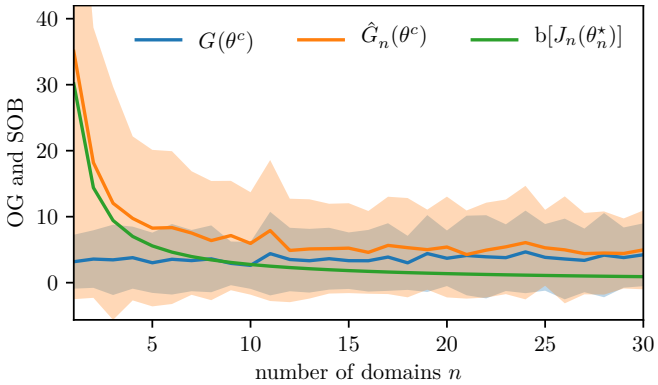


Figure 4: True optimality gap $G(\theta^c)$, the approximation from n domains $\hat{G}(\theta^c)$, and the simulation optimization bias $\mathrm{b}[\hat{J}_n(\theta^\star_n)]$. Note that $\hat{G}_n(\theta^c) \geq G(\theta^c)$ does not hold for every instance of the 100 random seeds, but is true in expectation. The variance in $G(\theta^c)$ is only caused by the variance in θ^c. The shaded areas show ± 1 standard deviation.

optimum $\hat{J}_n(\theta^\star_n)$ (Figure 3), and that with increasing number of domains the SOB $\mathrm{b}[\hat{J}(\theta^\star_n)]$ decreases monotonically while the estimated OG $\hat{G}_n(\theta^c)$ only decreases in expectation (Figure 4). When optimizing over $n = 30$ random domains in simulation, we yield a policy which leads to a $G_n(\theta^c) \approx 4.23\,\mathrm{m}$ higher (worse) shot compared to the best policy computed from an infinite set of domains and evaluated on this infinite set of domains, and a $\hat{G}_n(\theta^c) \approx 4.97\,\mathrm{m}$ higher (worse) shot compared to the best policy computed from a set of $n = 30$ domains and evaluated on the same finite set of domains. Furthermore, we can say that executing the best policy computed from a set of $n = 30$ domains will in reality result in a $\mathrm{b}[\hat{J}(\theta^\star_n)] \approx 0.911\,\mathrm{m}$ higher (worse) shot.

3 Simulation-Based Policy Optimization with Transferability Assessment

We introduce Simulation-based Policy Optimization with Transferability Assessment (SPOTA) [19], a policy search meta-algorithm which yields a policy that is able to directly transfer from a set of source domains to an unseen target domain. The goal of SPOTA is not only to maximize the expected discounted return under the influence of randomized physics simulations $J(\boldsymbol{\theta})$, but also to provide an approximate probabilistic guarantee on the suboptimality in terms of expected discounted return when applying the obtained policy to a different domain. The key novelty in SPOTA is the utilization of an Upper Confidence Bound on the Optimality Gap (UCBOG) as a stopping criterion for the training procedure of the RL agent.

One interpretation of (source) domain randomization is to see it as a form of uncertainty representation. If a control policy is trained successfully on multiple variations of the scenario, i.e., a set of models, it is legitimate to assume that this policy will be able to handle modeling errors better than policies that have only been trained on the nominal model $\bar{\boldsymbol{\xi}} = \mathbb{E}_\nu[\boldsymbol{\xi}]$. With this rationale in mind, we propose the SPOTA procedure, summarized in Algorithm 1.

SPOTA performs a repetitive comparison of solution candidates against reference solutions in domains that are in the references' training set but unknown to the candidates. As inputs, we assume a probability distribution over the domain parameters $\nu(\boldsymbol{\xi}; \boldsymbol{\phi})$, a policy optimization subroutine `PolOpt`, the number of candidate and reference domains n_c and n_r in conjunction with a nondecreasing sequence `NonDecrSeq` (e.g., $n_{k+1} = 2n_k$), the number of reference solutions n_G, the number of rollouts used for each OG estimate n_J, the number of bootstrap samples n_b, the confidence level $(1-\alpha)$ used for bootstrapping, and the threshold of trust β determining the stopping condition. SPOTA consists of four blocks: finding a candidate solution, finding multiple reference solutions, comparing the candidate against the reference solutions, and assessing the candidate solution quality.

Candidate Solutions are randomly initialized and optimized based on a set of n_c source domains (Lines 3–4). Practically, the locally optimal policy parameters are optimized on the approximated SP (2).

Reference Solutions are gathered n_G times by solving the same approximated SP as for the candidate but with different realizations of the random variable $\boldsymbol{\xi}$ (Lines 6–8). These n_G non-convex optimization processes all use the same candidate solution $\boldsymbol{\theta}^\star_{n_c}$ as initial guess.

Solution Comparison is done by evaluating each reference solution $\boldsymbol{\theta}^{k\star}_{n_r}$ with $k = 1, \ldots, n_G$ against the candidate solution $\boldsymbol{\theta}^\star_{n_c}$ for each realization of the random variable $\boldsymbol{\xi}^k_i$ with $i = 1, \ldots, n_r$ on which the reference solution has been trained. In this step, the performances per domain $\hat{J}_{n_J}(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i)$ and $\hat{J}_{n_J}(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i)$ are estimated from n_J Monte-Carlo simulations with synchronized random seeds (Lines 10–13). Thereby, both solutions are evaluated using the same random initial states and observation noise. Due to the potential suboptimality of the reference solutions, the resulting difference in performance

$$\hat{G}^k_{n_r,i}(\boldsymbol{\theta}^\star_{n_c}) = \hat{J}_{n_J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right) - \hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right) \tag{12}$$

may become negative (Line 14). This issue did not appear in previous work on assessing solution qualities of SPs [3, 20], because they only covered convex problems, where

Algorithm 1: Simulation-based Policy Optimization with Transferability Assessment (SPOTA)

input : probability distribution $\nu(\boldsymbol{\xi};\boldsymbol{\phi})$, algorithm `PolOpt`, sequence `NonDecrSeq`, hyper-parameters n_c, n_r, n_G, n_J, n_b, α, β
output: policy $\pi(\boldsymbol{\theta}^\star_{n_c})$ with a $(1-\alpha)$-level confidence on $\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c})$ which is upper bounded by β

1 Initialize $\pi(\boldsymbol{\theta}_{n_c})$ randomly
2 **do**
3 Sample n_c i.i.d. physics simulators described by $\boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_{n_c}$ from $\nu(\boldsymbol{\xi};\boldsymbol{\phi})$
4 Solve the approx. SP using $\boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_{n_c}$ and `PolOpt` to obtain $\boldsymbol{\theta}^\star_{n_c}$ ▷ `candidate solution`
5 **for** $k=1,\dots,n_G$ **do**
6 Sample n_r i.i.d. physics simulators described by $\boldsymbol{\xi}^k_1,\dots,\boldsymbol{\xi}^k_{n_r}$ from $\nu(\boldsymbol{\xi};\boldsymbol{\phi})$
7 Initialize $\boldsymbol{\theta}^k_{n_r}$ with $\boldsymbol{\theta}^\star_{n_c}$ and reset the exploration strategy
8 Solve the approx. SP using $\boldsymbol{\xi}^k_1,\dots,\boldsymbol{\xi}^k_{n_r}$ and `PolOpt` to obtain $\boldsymbol{\theta}^{k\star}_{n_r}$ ▷ `reference solution`
9 **for** $i=1,\dots,n_r$ **do**
10 **with** synchronized random seeds ▷ `sync initial states and observation noise`
11 Estimate the candidate solution's return $\hat{J}_{n_J}(\boldsymbol{\theta}^\star_{n_c},\boldsymbol{\xi}^k_i) \leftarrow 1/n_J \sum_{j=1}^{n_J} \hat{J}(\boldsymbol{\theta}^\star_{n_c},\boldsymbol{\xi}^k_i)$
12 Estimate the reference solution's return $\hat{J}_{n_J}(\boldsymbol{\theta}^{k\star}_{n_r},\boldsymbol{\xi}^k_i) \leftarrow 1/n_J \sum_{j=1}^{n_J} \hat{J}(\boldsymbol{\theta}^{k\star}_{n_r},\boldsymbol{\xi}^k_i)$
13 **end**
14 Compute the difference in return $\hat{G}^k_{n_r,i}(\boldsymbol{\theta}^\star_{n_c}) \leftarrow \hat{J}_{n_J}(\boldsymbol{\theta}^{k\star}_{n_r},\boldsymbol{\xi}^k_i) - \hat{J}_{n_J}(\boldsymbol{\theta}^\star_{n_c},\boldsymbol{\xi}^k_i)$
15 **end**
16 **for** $k=1,\dots,n_G$ **and** $i=1,\dots,n_r$ **do** ▷ `outlier rejection`
17 **if** $\hat{G}^k_{n_r,i}(\boldsymbol{\theta}^\star_{n_c}) < 0$ **then**
18 **for** $k'=1,\dots,n_G,\ k'\neq k$ **do** ▷ `loop over other reference solutions`
19 **if** $\hat{G}^{k'}_{n_r,i}(\boldsymbol{\theta}^\star_{n_c}) > \hat{G}^k_{n_r,i}(\boldsymbol{\theta}^\star_{n_c})$ **then** $\hat{G}^k_{n_r,i}(\boldsymbol{\theta}^\star_{n_c}) \leftarrow \hat{G}^{k'}_{n_r,i}(\boldsymbol{\theta}^\star_{n_c})$; **break** ▷ `replace solution`
20 **end**
21 **end**
22 **end**
23 **end**
24 Bootstrap n_b times from $\mathcal{G} = \{\hat{G}^1_{n_r,1}(\boldsymbol{\theta}^\star_{n_c}),\dots,\hat{G}^{n_G}_{n_r,n_r}(\boldsymbol{\theta}^\star_{n_c})\}$ to yield $\mathcal{G}^B_1,\dots,\mathcal{G}^B_{n_b}$ ▷ `bootstrapping`
25 Compute the sample mean $\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c})$ for the original set $\mathcal{G}$
26 Compute the sample means $\bar{G}^B_{n_r,1}(\boldsymbol{\theta}^\star_{n_c}),\dots,\bar{G}^B_{n_r,n_b}(\boldsymbol{\theta}^\star_{n_c})$ for the sets $\mathcal{G}^B_1,\dots,\mathcal{G}^B_{n_b}$
27 Select the α-th quantile of the bootstrap samples' means and obtain the upper bound for the one-sided $(1-\alpha)$-level confidence interval $\bar{G}^U_{n_r}(\boldsymbol{\theta}^\star_{n_c}) \leftarrow 2\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c}) - Q_\alpha[\bar{G}^B_{n_r}(\boldsymbol{\theta}^\star_{n_c})]$ ▷ `UCBOG`
28 Set the new sample sizes $n_c \leftarrow$ `NonDecrSeq`(n_c) and $n_r \leftarrow$ `NonDecrSeq`(n_r)
29 **while** $\bar{G}^U_{n_r}(\boldsymbol{\theta}^\star_{n_c}) > \beta$

all reference solutions are guaranteed to be global optima. Utilizing the definition of the OG in (7) for SPOTA demands for globally optimal reference solutions. Due to the non-convexity of the introduced RL problem the obtained solutions by the optimizer only are locally optimal. In order to alleviate this dilemma, we perform an outlier rejection routine (Lines 16–22). As a first attempt, all other reference solutions are evaluated for the current domain i. If a solution with higher performance was found, it replaces the current reference solution k for this domain. If all reference solutions are worse than the candidate, the value is clipped to the theoretical minimum (zero).

Solution Quality is assessed by constructing a $(1-\alpha)$-level confidence interval $\left[0, \bar{G}^U_{n_r}(\boldsymbol{\theta}^\star_{n_c})\right]$ for the estimated OG at $\boldsymbol{\theta}^\star_{n_c}$. While the lower bound is fixed to the theoretical minimum, the Upper Confidence Bound on the Optimality Gap (UCBOG) is computed using the statistical bootstrap method [21]. We denote bootstrapped quantities with the superscript B instead of the common asterisk, to avoid a notation conflict with the optimal solutions. There are multiple ways to yield a confidence interval by applying the bootstrap [22]. Here, the 'basic' nonparametric method was chosen, since the aforementioned potential clipping changes the distribution of the samples and hence a method relying on the estimation of population parameters such as the standard error is inappropriate. The solution comparison yields a set of $n_G n_r$ samples of the approximated OG $\mathcal{G} = \{\hat{G}^1_{n_r,1}(\boldsymbol{\theta}^\star_{n_c}),\dots,\hat{G}^{n_G}_{n_r,n_r}(\boldsymbol{\theta}^\star_{n_c})\}$. Through uniform random sampling with replacement from $\mathcal{G}$, we generate n_b bootstrap samples $\mathcal{G}^B_1,\dots,\mathcal{G}^B_{n_b}$. Thus, for our statistic of interest, the mean estimated OG $\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c})$, the UCBOG becomes

$$\bar{G}^U_{n_r}(\boldsymbol{\theta}^\star_{n_c}) = 2\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c}) - Q_\alpha\left[\bar{G}^B_{n_r}(\boldsymbol{\theta}^\star_{n_c})\right], \tag{13}$$

where $\bar{G}_{n_r}(\boldsymbol{\theta}^\star_{n_c})$ is the mean over all (nonnegative) samples from the empirical distribution, and $Q_\alpha[\bar{G}^B_{n_r}(\boldsymbol{\theta}^\star_{n_c})]$ is the α-th quantile of the means calculated for each of the n_b bootstrap samples (Lines 25–27). Consequently, the true OG is lower than the obtained one-sided confidence interval with the approximate probability of $(1-\alpha)$, i.e.,

$$\mathbb{P}\left(G(\boldsymbol{\theta}^\star_{n_c}) \leq \bar{G}^U_{n_r}(\boldsymbol{\theta}^\star_{n_c})\right) \approx 1-\alpha,$$

which is analogous to (4) in [20]. Finally, the sample sizes n_r and n_c of the next epoch are set according to the non-decreasing sequence. The procedure stops if the UCBOG

at $\boldsymbol{\theta}^{\star}_{n_c}$ is less than or equal to the specified threshold of trust β. Fulfilling this condition, the candidate solution at hand does not lose more than β in terms of performance with approximate probability $(1-\alpha)$, when it is applied to a different domain sampled from the same distribution.

Intuitively, the question arises why we do not use all samples for training a single policy and thus most likely yield a more robust result. To answer this question we want to point out that the key difference of SPOTA to the related methods is the assessment of the solution's transferability to different domains. While the approaches reviewed in Section 5 train one policy until convergence (e.g., for a fixed number of steps), SPOTA repeats this process and suggests new policies as long as the UCBOG is above a specified threshold. Thereby, SPOTA only uses $1/(1+n_G n/n_c)$ of the total samples to learn the candidate solution, i.e., the policy which will be deployed. If we would use all samples for training, hence not learn any reference solutions, we would not be able to estimate the OG and therefore lose the main feature of SPOTA.

4 Experiments

We evaluate SPOTA on two sim-to-real tasks pictured in Figure 1, the Ball-Balancer and the Cart-Pole. The policies obtained by SPOTA are compared against Ensemble Policy Optimization (EPOpt), and (plain) Proximal Policy Optimization (PPO) policies. The goal of the conducted experiments is twofold. First, we want to investigate the applicability of the UCBOG as a quantitative measure of a policy's transferability. Second, we aim to show that domain randomization enables the sim-to-real transfer of control policies learned by RL algorithms, while methods which only learn from the nominal domain fail to transfer.

4.1 Modeling and Setup Description

Both platforms can be classified as nonlinear under-actuated balancing problems with continuous state and action spaces. The Ball-Balancer's task is to stabilize the randomly initialized ball at the plate's center. Given measurements and their first derivatives (obtained by temporal filtering) of the motor shaft angles as well as the ball's position relative to the plate's center, the agent controls two servo motors via voltage commands. The rotation of the motor shafts leads, through a kinematic chain, to a change in the plate angles. Finally, the plate's deflection gets the ball rolling. The Ball-Balancer has an 8D state and a 2D action space. Similarly, the Cart-Pole's task is to stabilize a pole in the upright position by controlling the cart. Based on measurements and their first derivatives (obtained by temporal filtering) of the pole's rotation angle as well as the cart position relative to the rail's center, the agent controls the servo motor driving the cart by sending voltage commands. Accelerating the cart makes the pole rotate around an axis perpendicular to the cart's direction. The Cart-Pole has a 4D state and a 1D action space. Details on the dynamics of both systems, the reward functions, as well as listings of the domain parameters is given in Appendix A. The nominal models are based on the domain parameter values provided by the manufacturer.

In this paper, both systems have been modeled using the Lagrange formalism and the resulting differential equations

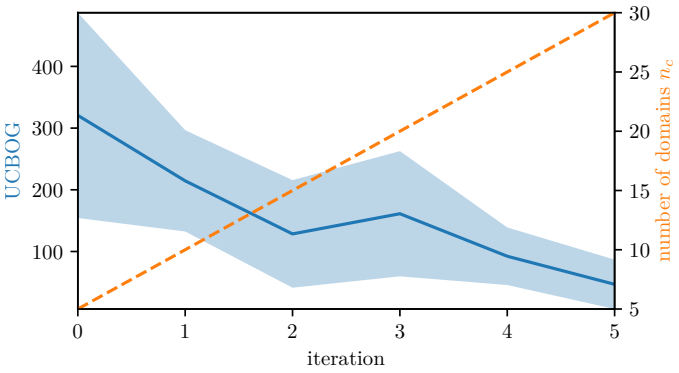


Figure 5: Upper Confidence Bound on the Optimality Gap (UCBOG) and number of candidate solution domain over the iteration count of SPOTA. Every iteration, the number and domains (dashed line) and hence the sample size is increased. The shaded area visualize ± 1 standard deviation across 9 training runs on the simulated Ball-Balancer.

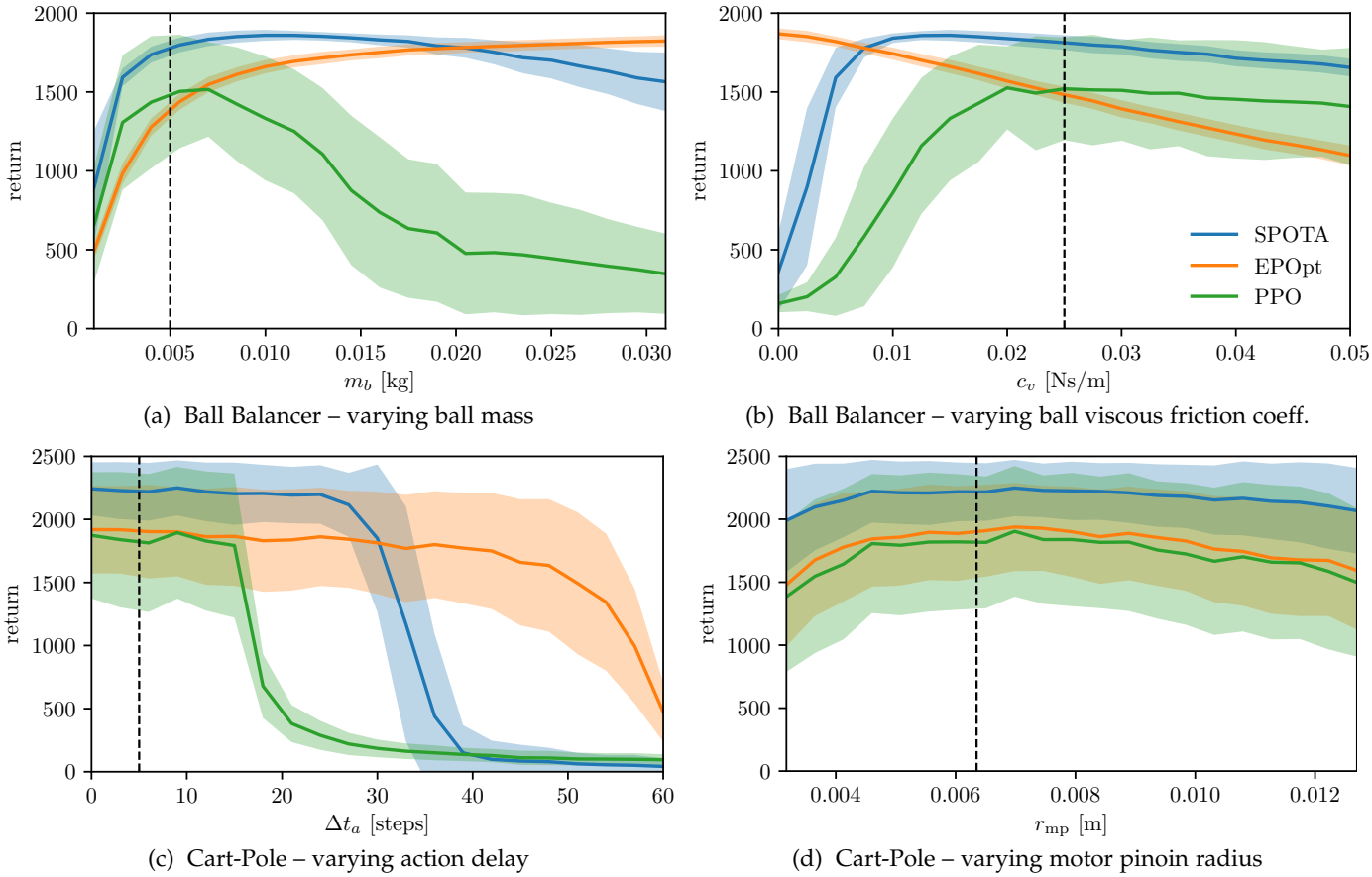
are integrated forward in time to simulate the systems. The associated domain parameters are drawn from a probability distribution $\boldsymbol{\xi} \sim \nu(\boldsymbol{\xi}; \boldsymbol{\phi})$, parameterized by $\boldsymbol{\phi}$ (e.g., mean, variance). Since randomizing a simulator's physics parameters is not possible right away, so we developed custom a framework to combine RL and domain randomization. Essentially, the base environment is modified by wrappers which, e.g., vary the mass or delay the actions.

4.2 Experiments Description

The experiments are split into two parts. At first, we examine the evolution of the UCBOG during training (Section 4.3). Next, we compare the transferability of the obtained policies across different realizations of the domain, i.e., simulator (Section 4.3). Finally, we evaluate the policies on the real-world platforms (Section 4.4).

For the experiments on the real Ball-Balancer, we choose 8 initial ball positions equidistant from the plate's center and place the ball at these points using a PD controller. As soon as a specified accuracy is reached, the evaluated policy is activated for 2000 time steps, i.e., 4 seconds. All experiments on the real Carl-pole start with the cart centered on the rail and the pendulum pole hanging down. After calibration, the pendulum is swung up using an energy-based controller. When the system's state is within a specified threshold, evaluated policy is executed for 4000 time steps, i.e., 8 seconds.

All policies have been trained in simulation with observation noise to mimic the noisy sensors. To focus on the domain parameters' influence, we executed the sim-to-sim experiments without observation noise. The policy update at the core of SPOTA, EPOpt, and PPO is done by the Adam optimizer [23]. In the sim-to-sim experiments, the rewards are computed from the ideal states coming from the simulator, while for the sim-to-real experiments the rewards are calculated from the sensor measurements and their time derivatives. The hyper-parameters chosen for the experiments can be found in the Appendix B.



(a) Ball Balancer – varying ball mass

(b) Ball Balancer – varying ball viscous friction coeff.

(c) Cart-Pole – varying action delay

(d) Cart-Pole – varying motor pinoin radius

Figure 6: Evaluation of the learned control policies on the simulated Ball Balancer (top row) as well as the simulated Cart-Pole (bottom row), varying the ball mass m_b (a), the viscous friction coefficient (b), the action delay Δt_a (c), and the motor pinion radius r_{mp} (d). Every domain parameter configuration has been evaluated on 360 rollouts with different initial states, synchronized across all policies. The dashed lines mark the nominal parameter values. The solid lines represent the means, and shaded areas show ± 1 standard deviation

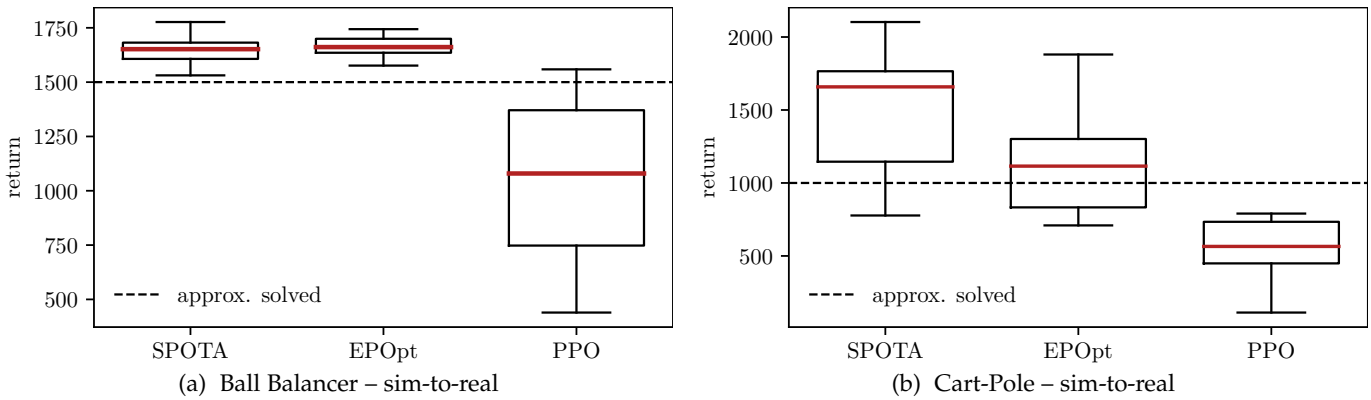
4.3 Sim-to-Sim Results

The UCBOG value (13) at each SPOTA iteration depends on several hyper-parameters such as the current number of domains and reference solutions, or the quality of the current iteration's solution candidate. Figure 5 displays the descent of the UCBOG as well as the growth of the number of domains with the increasing iteration count of SPOTA. As described in Section 2.3.2, the OG and thus the UCBOG only decreases in expectation. Therefore, it can happen that for a specific training run the UCBOG increases from one iteration to the next. Moreover, we observed that the proportion of negative OG estimates (12) increases with progressing iteration count. This observation can be explained by the fact that SPOTA increases the number of domains used during training. Hence the set's empirical mean approximates the domain parameter distribution $\nu(\boldsymbol{\xi}; \boldsymbol{\phi})$ progressively better, i.e., the candidate and reference solution become more similar. Note, that due to the computational complexity of SPOTA we decided to combine results from experiments with different hyper-parameters in Figure 5.

To estimate the robustness w.r.t. model parameter uncertainties, we evaluate policies trained by SPOTA, EPOpt, and PPO under on multiple simulator instances, varying only one domain parameter. The policies' sensitivity to different parameter values is displayed in the Figure 6. From the Figures 6a to 6c we can see that the policies learned using (plain) PPO are less robust to changes in the domain parameter values. In contrast, SPOTA and EPOpt are able to maintain their level of performance across a wider range of parameter values. The Figure 6d shows the case of a domain parameter to which all policies are equally insensitive. We can also see that EPOpt trades off performance in the nominal domains for performance in more challenging domains (e.g., low friction). This behavior is a consequence of its CVaR-based objective function [11]. Moreover, the results show a higher variance for the PPO policy than for the others. From this, we conclude that domain randomization also acts as regularization mechanism. The final UCBOG value of the evaluated SPOTA policies was 46.42 for the Ball-Balancer and 55.14 for the Cart-Pole. Note, that the UCBOG can not be directly observed from the curves in Figure 6, since the UCBOG reflects the gap in performance between the best policy for a specific simulator instance and the candidate policy, whereas the Figure 6 only shows the candidates' performances.

4.4 Sim-to-Real Results

When transferring the policies from simulation to reality without any fine-tuning, we obtain the results reported in



(a) Ball Balancer – sim-to-real (b) Cart-Pole – sim-to-real

Figure 7: Evaluation of the learned control policies on the real Ball Balancer (a) and the real Cart-Pole (b). The results were obtained from 40 rollouts per policy on the Ball Balancer (5 repetitions for 8 different initial ball positions) as well as 10 rollouts (1 initial state) on the Cart-Pole. The dashed lines approximately mark the threshold where the tasks are solved, i.e., the ball is centered in the middle (a), or the pendulum is stabilized upright (b) at the end of the episode.

Figure 7. The approaches using domain randomization are in most cases able to directly transfer from simulation to reality. In contrast, the policies trained on a singular nominal model using failed to transfer in all but 2 trials on the Ball-Balancer as well as in all trials on the Cart-Pole, even though these policies solved the simulated environments.

Regarding the Ball-Balancer, one explanation why the reported PPO policy did not transfer to the real platform could be the value of the viscous friction coefficient (Figure 6b). A possible test for this hypothesis would be to train multiple policies on a range of nominal models with altered viscous friction parameter value, and if these policies still do not transfer, examine the next domain parameter. However, this procedure is redundant and can quickly become prohibitively time-intensive. Concerning the experiments on the Cart-Pole, we observe a larger reality-gap for all policies. We believe that this discrepancy is caused by unmodeled effects between the rack and the cart's pinion (e.g., the heavy wear and tear of the pinon made out of plastic). Moreover, the variance of the returns is significantly higher. This increase can be largely explained by the variance in the initial states caused by the pre-defined swing-up controller.

A video of the SPOTA policy's sim-to-real transfer on both platforms can be found at https://www.ias.informatik.tu-darmstadt.de/Team/FabioMuratore.

4.5 Limitations of the Presented Method

The computation of the UCBOG13, and hence the estimation of the SOB, relies on the relative quality of the candidate and the reference solutions. Therefore, the most notable limitation of the presented method is the optimizer's ability to reliably solve the SP (2). Since we are interested in the general setting of learning a control policy from a black-box simulator, we chose a model-free RL algorithm. These kind of algorithms can not guarantee to find the optimal, or loosely speaking a very good, solution. One way to alleviate this problem is to compute the reference policies from a single domain using methods from control engineering, e.g. a LQR. Note that this solution would require an analytic model of the system and a specific kind of reward function (to preserve comparability), e.g. quadratic functions in case of the LQR.

Another limitation of SPOTA is the increased hyper-parameter space which is a direct consequence from the employed (static) domain randomization. In combination with the fact that SPOTA is solving the underlying RL problem $(1+n_G)n_{iter}$ times, the procedure becomes computationally expensive. One can counter this problem for example by parallelizing the computation of the reference policies as well es the hyper-parameter search. Both are embarrassingly parallel. Moreover, the necessity of finding good domain parameter distribution parameters ϕ could be alleviated by adapting these distributions, as done in [17].

Finally, to estimate the SOB, SPOTA assumes that the target domain is covered by the source domain distribution, which can not be guaranteed if the target is a real-world system. However, in the current state-of-the-art there is no way to estimate a policy's transferability to a domain from an unknown distribution. Due to mentioned assumption, SPOTA's transferability assessment strongly depends on the simulator's ability to model the real world.

5 Related Work

In the following, we review excerpts of the literature regarding the transfer of control policies from simulation to reality, the concept of the optimality gap in Stochastic Programs (SPs), and the application of randomized physics simulations.

5.1 Key publications on the Optimality Gap

Hobbs & Hepenstal [2] proved for linear programs that optimization is optimistically biased, given that there are errors in estimating the objective function coefficients. Furthermore, they demonstrated the "optimistic bias" of a nonlinear program, and mentioned the effect of errors on the parameters of linear constraints. The optimization problem introduced in Section 3 belongs to the class of SPs for which the assumption required in [2] are guaranteed to hold. The most common approaches to solve convex SPs are sample average approximation methods, including: (i) the Multiple Replications Procedure and its derivatives [3, 20] which assess a solution's quality by comparing with sampled alternative solutions, and (ii) Retrospective Approximation [24, 25]

which iteratively improved the solution by lowering the error tolerance. Bastin et al. [26] extended the existing convergence guarantees from convex to non-convex SPs, showing almost sure convergence of the minimizers.

5.2 Prior work on Reality Gap

Physics simulations have already been used successfully in robot learning. Traditionally, simulators are operating on a single nominal model, which makes the direct transfer of policies from simulation to reality highly vulnerable to model uncertainties and biases. Thus, model-based control in most cases relies on fine-tuned dynamics models.
The mismatch between the simulated and the real world has been addressed by robotics researchers from different viewpoints. Prominent examples are:

1) adding i.i.d. noise to the observations and actions in order to mimic real-world sensor and actuator behavior [1],
2) model generation and selection depending on the short-term state-action history [27],
3) learning a transferability function which maps solutions to a score that quantifies how well the simulation matches the reality [28],
4) randomizing the physics simulation's parameters, and
5) applying adversarial perturbations to the system,

where the last two approaches are particularly related.

5.3 Required Randomized Simulators

Simulators can be obtained by implementing a set of physical laws, as well as by using general purpose physics engines like Bullet [29], MuJoCo [30], or Vortex [31]. The associated physics parameters can be estimated by system identification, which involves executing rollouts on the physical platform [32]. Additionally, using the Gauss-Markov theorem one could also compute the parameters' covariance and hence construct a normal distribution for each domain parameter. Alternatively, the system dynamics can be captured using non-parametric methods like Gaussian processes [33]. It is important to keep in mind, that even if the selected procedure yields a very accurate model parameter estimate, simulators are nevertheless just approximations of the real world and are thus always flawed.

As done in [10, 11, 14, 15, 16] we use the domain parameter distribution as a prior which ensures the physical plausibility of each parameter. Note that specifying this distribution in the current state-of-the-art requires the researcher to make design decisions. Chebotar et al. [17] presented a promising method which adapts the domain parameter distribution using real-world data in the loop. The main advantage is that this approach alleviates the need for hand-tuning the distributions of the domain parameters, which is currently a significant part of the hyper-parameter search. However, the initial distribution still demands for design decisions. On the downside, the adaptation requires data from the real robot which is considered significantly more expensive to obtain. Since we aim for performing a sim-to-real transfer without using any real-world data, the introduced method only samples from static probability distributions.

5.4 Background on Domain Randomization

There is a large consensus that further increasing the simulator's accuracy alone will not bridge the reality gap. Instead, the idea of domain randomization has recently gained momentum. The common characteristic of such approaches is the perturbation of the parameters which determine the physics simulator and the state estimation, including but not limited to the system dynamics. While the idea of randomizing the sensors and actuators dates back to at least 1995 [1], the systematic analysis of perturbed simulations in robot RL is a relatively new research direction.

Wang et al. [34] proposed sampling initial states, external disturbances, goals, as well as actuator noise from probability distributions and learned walking policies in simulation. Regarding robot RL, recent domain randomization methods focus on perturbing the parameters defining the system dynamics. Approaches cover: (i) trajectory optimization on finite model-ensembles [10] (ii) learning a feed-forward NN policy for an under-actuated problem [35], (iii) using a risk-averse objective function [11], (iv) employing recurrent NN policies trained with experience replay [15], and (v) optimizing a policy from samples of a model randomly chosen from a set which is repeatedly fitted to real-world data [36]. From the listed approaches [10, 35, 15] were able to cross the reality gap without acquiring samples from the real world.

Moreover, there is a significant amount of work applying domain randomization to computer vision. One example is the work by Tobin et al. [37] where an object detector for robot grasping is trained using multiple variants of the environment and applied to the real world. The approach presented by Pinto et al. [14] combines the concepts of randomized environments and actor-critic training, enabling the direct sim-to-real transfer of the abilities to pick, push, or move objects. Sadeghi & Levine [38] achieved the sim-to-real transfer by learning to fly a drone in visually randomized environments. The resulting deep NN policy was able to map from monocular images to normalized 3D drone velocities. In [39], a deep NN was trained to manipulate tissue using randomized vision data and the full state information. Bousmalis et al. greatly reduced the number of necessary real-world samples for learning a robotic grasping task by combing generative adversarial networks and domain randomization [40].

5.5 Adversarial Perturbations for Domain Randomization

Another approach of learning robust policies in simulation is to apply adversarial disturbances to the training process. Mandleka et al. [12] proposed physically plausible perturbations by randomly deciding when to add a rescaled gradient of the expected return. Pinto et al. [13] introduced the idea of a second agent whose goal is to hinder the first agent from fulfilling its task. Both agents are trained simultaneously and make up a zero-sum game. In general, adversarial approaches may provide a particularly robust policy. However, without any further restrictions, it is always possible create scenarios in which the protagonist agent can never win, i.e., the policy will not learn the task.

6 Conclusion

We proposed a novel measure of the Simulation Optimization Bias (SOB) for quantifying the transferability of an arbitrary policy learned from a randomized source domain to an unknown target domain from the same domain parameter distribution. Based on this measure of the SOB, we developed a policy search meta-algorithm called Simulation-based Policy Optimization with Transferability Assessment (SPOTA). This gist of SPOTA is to iteratively increase the number of domains and thereby the sample size per iteration until an approximate probabilistic guarantee on the optimality gap holds. The required approximation of the optimality gap is obtained by comparing the current candidate policy against multiple reference policies evaluated in the associated reference domains. After training, we can make an approximation of the resulting policy's suboptimality when transferring to a different domain from the same (source) distribution. To verify our approach we conducted two sim-to-real experiments on nonlinear continuous control problems. The results showed that SPOTA policies were able to directly transfer from simulation to reality while the baseline without domain randomization failed.

In the future we will investigate different strategies for sampling the domain parameters to replace the i.i.d. sampling from hand-crafted distributions. One idea is to employ Bayesian optimization for selecting the next set of domain parameters. Thus, the domain randomization could be executed according to an objective, and potentially increase sample-efficiency. Furthermore, we plan to devise a formulation which frames domain randomization and policy search in one optimization problem. This would allow for an joint treatment of finding a policy and matching the simulator to the real world.

Acknowledgments

Fabio Muratore gratefully acknowledges the financial support from Honda Research Institute Europe.
Jan Peters received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 640554.

References

[1] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life,, Granada, Spain, June 4-6*. Springer, 1995, pp. 704–720.

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

[3] W. Mak, D. P. Morton, and R. K. Wood, "Monte carlo bounding techniques for determining solution quality in stochastic programs," *Oper. Res. Lett.*, vol. 24, no. 1-2, pp. 47–56, 1999.

[4] "Quanser platforms," Online. [Online]. Available: https://www.quanser.com/products/

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *ArXiv e-prints*, 2015.

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

[9] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *CoRL, Mountain View, California, USA, November 13-15*, 2017, pp. 262–270.

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

[11] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *ArXiv e-prints*, vol. 1610.01283, 2016.

[12] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *IROS, Vancouver, BC, Canada, September 24-28*, 2017, pp. 3932–3939.

[13] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *ICML, Sydney, NSW, Australia, August 6-11*. PMLR, 2017, pp. 2817–2826.

[14] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *RSS, Pittsburgh, Pennsylvania, USA, June 26-30*, 2018.

[15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *ArXiv e-prints*, 2017.

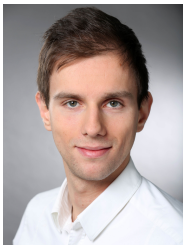
[16] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *RSS, Cambridge, Massachusetts, USA, July 12-16*, 2017.

[17] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. D. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *ArXiv e-prints*, vol. abs/1810.05687, 2018.

[18] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *ArXiv e-prints*, vol. 1808.00177, 2018.

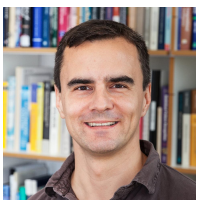
[19] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain randomization for simulation-based policy optimization with transferability assessment," in *CoRL,*

Zürich, Switzerland, 29-31 October, 2018, pp. 700–713.
[20] G. Bayraksan and D. P. Morton, "Assessing solution quality in stochastic programs," *Math. Program.*, vol. 108, no. 2-3, pp. 495–514, 2006.
[21] B. Efron, "Bootstrap methods: another look at the jackknife," *The annals of Statistics*, pp. 1–26, 1979.
[22] T. J. DiCiccio and B. Efron, "Bootstrap confidence intervals," *Statistical Science*, pp. 189–212, 1996.
[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR, San Diego, CA, USA, May 7-9*, 2015.
[24] R. Pasupathy and B. W. Schmeiser, "Retrospective-approximation algorithms for the multidimensional stochastic root-finding problem," *ACM Trans. Model. Comput. Simul.*, vol. 19, no. 2, pp. 5:1–5:36, 2009.
[25] S. Kim, R. Pasupathy, and S. G. Henderson, "A guide to sample average approximation," in *Handbook of Simulation Optimization*. Springer, 2015, pp. 207–243.
[26] F. Bastin, C. Cirillo, and P. L. Toint, "Convergence theory for nonconvex stochastic programming with an application to mixed logit," *Math. Program.*, vol. 108, no. 2-3, pp. 207–234, 2006.
[27] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.
[28] S. Koos, J. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Trans. Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2013.
[29] "Bullet physics engine," Online.
[30] "Mujoco phyiscs engine," Online. [Online]. Available: http://www.mujoco.org/
[31] "Vortex physics engine," Online. [Online]. Available: https://www.cm-labs.com/vortex-studio/
[32] R. Isermann and M. Münchhof, *Identification of Dynamic Systems: An Introduction with Applications*. Springer Science & Business Media, 2010.
[33] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, ser. Adaptive computation and machine learning. MIT Press, 2006.
[34] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers for uncertain inputs and environments," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 73:1–73:8, 2010.
[35] R. Antonova and S. Cruciani, "Unlocking the potential of simulators: Design with RL in mind," *ArXiv e-prints*, 2017.
[36] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *ArXiv e-prints*, 2018.
[37] 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, Vancouver, BC, Canada, September 24-28*, 2017, pp. 23–30.
[38] F. Sadeghi and S. Levine, "CAD2RL: real single-image flight without a single real image," in *RSS, Cambridge, Massachusetts, USA, July 12-16*, 2017.
[39] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *CoRL, Zürich, Switzerland, 29-31 October*, 2018, pp. 734–743.
[40] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *ICRA, Brisbane, Australia, May 21-25*, 2018, pp. 4243–4250.



Fabio Muratore received his M.Sc. degree in Mechanical Engineering from the Technical University of Munich in 2016. Currently, he is perusing his Ph.D. at the Intelligent Autonomous Systems Group at the Computer Science Department of the Technical University of Darmstadt. His research interests center around robotics, artificial intelligence, and learning from physics simulations.

Michael Gienger received the diploma degree in Mechanical Engineering from the Technical University of Munich (TUM) in 1998. From 1998 to 2003, he was research assistant at the Institute of Applied Mechanics of the TUM, addressing issues in design and realization of biped robots. He received his Ph.D. degree with a dissertation on "Design and Realization of a Biped Walking Robot". After this, Michael Gienger joined the Honda Research Institute Europe in 2003. Currently, he works as a Chief Scientist in the field of robotics. His research interests include mechatronics, robotics, control systems, robot learning and human-robot interaction.



Jan Peters is a full professor (W3) for Intelligent Autonomous Systems at the Computer Science Department of the Technical University of Darmstadt, and at the same time a senior research scientist and group leader at the Max-Planck Institute for Intelligent Systems, where he heads the interdepartmental Robot Learning Group. Jan Peters has received the Dick Volz Best 2007 US Ph.D. Thesis Runner-Up Award, the Robotics: Science & Systems - Early Career Spotlight, the INNS Young Investigator Award, and the IEEE Robotics & Automation Society's Early Career Award as well as numerous best paper awards. In 2015, he received an ERC Starting Grant and in 2019, he was appointed as an IEEE Fellow.

APPENDIX A
MODELING DETAILS ON THE PLATFORMS

The Ball-Balancer is modeled as a nonlinear second-order dynamical system

$$\ddot{\boldsymbol{s}} = \begin{bmatrix} \ddot{\theta}_x \\ \ddot{\theta}_y \\ \ddot{x}_b \\ \ddot{y}_b \end{bmatrix} = \begin{bmatrix} (A_m V_x - B_v \dot{\theta}_x)/J_{eq} \\ (A_m V_y - B_v \dot{\theta}_y)/J_{eq} \\ (-c_v \dot{x}_b r_b^2 - J_b r_b \ddot{\alpha} + m_b x_b \dot{\alpha}^2 r_b^2 \\ \quad + c_{kin} m_b g r_b^2 \sin(\theta_x))/\zeta \\ (-c_v \dot{y}_b r_b^2 - J_b r_b \ddot{\beta} + m_b y_b \dot{\beta}^2 r_b^2 \\ \quad + c_{kin} m_b g r_b^2 \sin(\theta_y))/\zeta \end{bmatrix},$$

with the motor shaft angles θ_x and θ_y, the ball positions relative to the place center x_b and y_b, the plate angle β and α around the x and y axis, and the commanded motor voltages $\boldsymbol{a}^\mathsf{T} = [V_x, V_y]$. To model the gears' backlash, we set all voltage values between $V_{thold,-}$ and $V_{thold,+}$ to zero. These threshold values have been determined in separate experiments for both servo motors. The Ball-Balancer's domain parameters as well as the ones derived from them are listed in Table 4. For the Ball-Balancer's we define the reward function as

$$r(\boldsymbol{s}_t, \boldsymbol{a}_t) = \exp\left(c\left(\boldsymbol{s}_t^\mathsf{T} \boldsymbol{Q}_{BB} \boldsymbol{s}_t + \boldsymbol{a}_t^\mathsf{T} \boldsymbol{R}_{BB} \boldsymbol{a}_t\right)\right)$$
$$\text{with } c = \frac{\ln(r_{min})}{\max_{\boldsymbol{s}\in\mathcal{S}_{\tilde{\boldsymbol{\xi}}}, \boldsymbol{a}\in\mathcal{A}_{\tilde{\boldsymbol{\xi}}}} \boldsymbol{s}^\mathsf{T} \boldsymbol{Q}_{BB} \boldsymbol{s} + \boldsymbol{a}^\mathsf{T} \boldsymbol{R}_{BB} \boldsymbol{a}}.$$

Given a lower bound for the reward $r_{min} \in [0, 1]$, the reward function above yields values within $[r_{min}, 1]$ at each time step. We found that the scaling constant $c < 0$ is beneficial for the learning procedure, since it prohibits the reward from going to zero too quickly. The constant's denominator can be easily inferred from the nominal state and action set's boundaries.

The Cart-Pole is modeled as a nonlinear second-order dynamical system given by the solution of

$$\begin{bmatrix} m_p + J_{eq} & m_p l_p \cos(\alpha) \\ m_p l_p \cos(\alpha) & J_p + m_p l_p^2 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} F - m_p l_p \sin(\alpha)\, \dot{\alpha}^2 - B_{eq} \dot{x} \\ -m_p l_p g \sin(\alpha) - B_p \dot{\alpha} \end{bmatrix},$$

where the commanded motor voltage V is encapsulated in

$$F = \frac{\eta_g K_g k_m}{R_m r_{mp}} \left(\frac{\eta_m V - K_g k_m \dot{x}}{r_{mp}}\right).$$

The system's state $\boldsymbol{s}$ is given by the cart's position x and the pole's angle α, which are defined to be zero at the rail's center and hanging down vertically, respectively. The Cart-Pole's domain parameters as well as the parameters derived from them are listed in Table 5. Similar to the Ball-Balancer, the Cart-Pole's reward function is based on an exponentiated quadratic cost

$$r(\boldsymbol{s}_t, a_t) = \exp\left(-\left(\boldsymbol{s}_t^\mathsf{T} \boldsymbol{Q}_{CP} \boldsymbol{s}_t + a_t^\mathsf{T} R_{CP} a_t\right)\right).$$

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

APPENDIX B
PARAMETER VALUES FOR THE EXPERIMENTS

Table 2: Domain parameter values for the illustrative example. Additional (domain-independent) parameters are $m = 1\,\text{kg}$ and $\phi = 0.7$.

Domain	g_i[m/ sec^2]	k_i[N/m]	x_i[m]
Mars	3.71	1000	0.5
Venus	8.87	3000	1.5

Table 3: Hyper-parameter values for the experiments in Section 4. All simulator parameters were randomized such that they stayed physically plausible. We use n as shorthand for n_c or n_r depending on the context.

Hyper-parameter	Value
`PolOpt`	PPO
policy architecture	FNN 16-16 with tan-h
optimizer	Adam
learning rate	1e−4
number of iterations n_{iter}	400
max. steps per episode T	Ball-Balancer: 2000
	Cart-Pole: 2500
step size Δt	0.002 s
temporal discount γ	0.999
λ (andvantage estimation)	0.95
initial n_c	5
initial n_r	1
`NonDecrSeq`	$n_{k+1} \leftarrow \lfloor n_0(k+1) \rfloor$
rollouts per domain parameter n_τ	10
batch size	$\lceil T/\Delta t \rceil n_\tau n$ steps
number of reference solutions n_G	20
rollouts per initial state n_J	Ball-Balancer: 120
	Cart-Pole: 50
confidence parameter α	0.05
threshold of trust β	Ball-Balancer: 50
	Cart-Pole: 60
number of bootstrap replications B	1000
CVaR parameter ϵ (EPOpt)	0.2
r_{min}	10^{-4}
$\boldsymbol{Q}_{BB}$	diag(1, 1, 5e3, 5e3, . . .
	1e−2, 1e−2, 5e−2, 5e−2)
$\boldsymbol{R}_{BB}$	diag(1e−3, 1e−3)
$\boldsymbol{Q}_{CP}$	diag(10, 1e3, 5e−2, 5e−3)
R_{CP}	1e−4
obs. noise std for linear pos	5e−3 m
obs. noise std for linear vel	0.05 m/s
obs. noise std for angular pos	0.5°
obs. noise std for angular vel	2.0 °/s

Table 4: The domain parameter distributions and derived parameters for the Ball-Balancer (Figure 1 left). All parameters were randomized such that they stayed physically plausible. Normal distributions are parameterized with mean and standard deviation, uniform distributions with lower and upper bound. The lines separate the randomized domain parameters from the ones depending on these.

Parameter	Distribution	Unit
gravity constant	$\mathcal{N}\left(g \middle\vert 9.81, 1.962\right)$	[kg]
ball mass	$\mathcal{N}\left(m_b \middle\vert 5\text{e}{-3}, 6\text{e}{-4}\right)$	[kg]
ball radius	$\mathcal{N}\left(r_b \middle\vert 1.96\text{e}{-2}, 3.93\text{e}{-3}\right)$	[m]
plate length	$\mathcal{N}\left(l_p \middle\vert 0.275, 5.5\text{e}{-2}\right)$	[m]
kinematic leverage arm	$\mathcal{N}\left(r_{kin} \middle\vert 2.54\text{e}{-2}, 3.08\text{e}{-3}\right)$	[m]
gear ratio	$\mathcal{N}\left(K_g \middle\vert 70, 14\right)$	[−]
gearbox efficiency	$\mathcal{U}\left(\eta_g \middle\vert 1.0, 0.6\right)$	[−]
motor efficiency	$\mathcal{U}\left(\eta_m \middle\vert 0.89, 0.49\right)$	[−]
load moment of inertia	$\mathcal{N}\left(J_l \middle\vert 5.28\text{e}{-5}, 1.06\text{e}{-5}\right)$	[kg m^2]
motor moment of inertia	$\mathcal{N}\left(J_m \middle\vert 4.61\text{e}{-7}, 9.22\text{e}{-8}\right)$	[kg m^2]
motor torque constant	$\mathcal{N}\left(k_m \middle\vert 7.7\text{e}{-3}, 1.52\text{e}{-3}\right)$	[N m/A]
motor armature resistance	$\mathcal{N}\left(R_m \middle\vert 2.6, 0.52\right)$	[Ω]
motor viscous damping coeff. w.r.t. load	$\mathcal{U}\left(B_{\text{eq}} \middle\vert 0.15, 3.75\text{e}{-3}\right)$	[N m s]
ball/plate viscous friction coeff.	$\mathcal{U}\left(c_v \middle\vert 5.0\text{e}{-2}, 1.25\text{e}{-3}\right)$	[−]
positive voltage threshold x servo	$\mathcal{U}\left(V_{thold,x+} \middle\vert 0.353, 8.84\text{e}{-2}\right)$	[V]
negative voltage threshold x servo	$\mathcal{U}\left(V_{thold,x-} \middle\vert -8.90\text{e}{-2}, -2.22\text{e}{-3}\right)$	[V]
positive voltage threshold y servo	$\mathcal{U}\left(V_{thold,y+} \middle\vert 0.290, 7.25\text{e}{-2}\right)$	[V]
negative voltage threshold y servo	$\mathcal{U}\left(V_{thold,y-} \middle\vert -7.30\text{e}{-2}, -1.83\text{e}{-2}\right)$	[V]
offset x servo	$\mathcal{U}\left(\Delta\theta_x \middle\vert -5, 5\right)$	[deg]
offset y servo	$\mathcal{U}\left(\Delta\theta_y \middle\vert -5, 5\right)$	[deg]
action delay	$\mathcal{U}\left(\Delta t_a \middle\vert 0, 30\right)$	[steps]
kinematic constant	$c_{kin} = 2r_{\text{kin}}/l_p$	[−]
combined motor constant	$A_m = \eta_g K_g \eta_m k_m / R_m$	[N m/V]
combined rotary damping coefficient	$B_v = \eta_g K_g^2 \eta_m k_m^2 / R_m + B_{\text{eq}}$	[N m s]
combined rotor inertia	$J_{eq} = \eta_g K_g^2 J_m + J_l$	[kg m^2]
ball inertia about CoM	$J_b = 2/5 m_b r_b^2$	[kg m^2]
combined ball inertia	$\zeta = m_b r_b^2 + J_b$	[kg m^2]

Table 5: The domain parameter distributions and derived parameters for the Cart-Pole (Figure 1 right). All parameters were randomized such that they stayed physically plausible. Normal distributions are parameterized with mean and standard deviation, uniform distributions with lower and upper bound. The lines separate the randomized domain parameters from the ones depending on these.

Parameter	Distribution	Unit
gravity constant	$\mathcal{N}\left(g \middle\vert 9.81, 1.962\right)$	[kg]
cart mass	$\mathcal{N}\left(m_c \middle\vert 0.38, 0.076\right)$	[kg]
pole mass	$\mathcal{N}\left(m_p \middle\vert 0.127, 2.54\text{e}{-2}\right)$	[kg]
half pole length	$\mathcal{N}\left(l_p \middle\vert 0.089, 1.78\text{e}{-2}\right)$	[m]
rail length	$\mathcal{N}\left(l_r \middle\vert 0.814, 0.163\right)$	[m]
motor pinion radius	$\mathcal{N}\left(r_{mp} \middle\vert 6.35\text{e}{-3}, 1.27\text{e}{-3}\right)$	[m]
gear ratio	$\mathcal{N}\left(K_g \middle\vert 3.71, 0\right)$	[−]
gearbox efficiency	$\mathcal{U}\left(\eta_g \middle\vert 1.0, 0.8\right)$	[−]
motor efficiency	$\mathcal{U}\left(\eta_m \middle\vert 1.0, 0.8\right)$	[−]
motor moment of inertia	$\mathcal{N}\left(J_m \middle\vert 3.9\text{e}{-7}, 0\right)$	[kg m^2]
motor torque constant	$\mathcal{N}\left(k_m \middle\vert 7.67\text{e}{-3}, 1.52\text{e}{-3}\right)$	[N m/A]
motor armature resistance	$\mathcal{N}\left(R_m \middle\vert 2.6, 0.52\right)$	[Ω]
motor viscous damping coeff. w.r.t. load	$\mathcal{U}\left(B_{eq} \middle\vert 5.4, 0\right)$	[N s/m]
pole viscous friction coeff.	$\mathcal{U}\left(B_p \middle\vert 2.4\text{e}{-3}, 0\right)$	[N s]
action delay	$\mathcal{U}\left(\Delta t_a \middle\vert 0, 10\right)$	[steps]
pole rotary inertia about pivot point	$J_p = 1/3 m_p l_p^2$	[kg m^2]
combined linear inertia	$J_{eq} = m_c + (\eta_g K_g^2 J_m)/r_{mp}^2$	[kg]