Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment

Fabio Muratore[1,2], **Felix Treede**[1,2], **Michael Gienger**[2], **Jan Peters**[1,3]
[1] Institute for Intelligent Autonomous Systems, Technische Universität Darmstadt, Germany
[2] Honda Research Institute Europe, Offenbach am Main, Germany
[3] Max Planck Institute for Intelligent Systems, Tübingen, Germany
Correspondence to muratore@ias.tu-darmstadt.de

Abstract: Exploration-based reinforcement learning on real robot systems is generally time-intensive and can lead to catastrophic robot failures. Therefore, simulation-based policy search appears to be an appealing alternative. Unfortunately, running policy search on a slightly faulty simulator can easily lead to the maximization of the 'Simulation Optimization Bias' (SOB), where the policy exploits modeling errors of the simulator such that the resulting behavior can potentially damage the robot. For this reason, much work in robot reinforcement learning has focused on model-free methods that learn on real-world systems. The resulting lack of safe simulation-based policy learning techniques imposes severe limitations on the application of robot reinforcement learning.
In this paper, we explore how physics simulations can be utilized for a robust policy optimization by perturbing the simulator's parameters and training from model ensembles. We propose a new algorithm called Simulation-based Policy Optimization with Transferability Assessment (SPOTA) that uses a biased estimator of the SOB to formulate a stopping criterion for training. We show that the new simulation-based policy search algorithm is able to learn a control policy exclusively from a randomized simulator that can be applied directly to a different system without using any data from the latter.

Keywords: domain randomization, simulation optimization, direct transfer

1 Introduction

Exploration-based learning of control policies on a physical robot 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 small errors can lead to unstable behavior. 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 here is 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. While already formulated under the name 'optimality gap' by the optimization community in the 1990s [2, 3], the concept of the SOB has neither been transfered to robotics nor Reinforcement Learning (RL) yet.
Deep RL algorithms recently demonstrated super-human performance in playing games [4, 5] and promising results in (simulated) robotic control tasks [6, 7, 8, 9]. However, when transferred to real-world robotic systems, most of these approaches become less attractive due to high sample complexity and a lack of explainability of state-of-the-art deep RL algorithms.

2nd Conference on Robot Learning (CoRL 2018), Zurich, Switzerland.

As a consequence, the research field of 'domain randomization', also called 'perturbed simulations', is gaining interest [10, 11, 12, 13, 14, 15]. 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 an ensemble of models instead of just one nominal model. The randomization of the physics parameters is motivated by the fact that the 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.

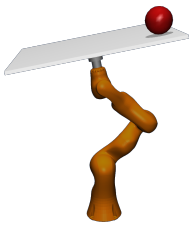


Figure 1: Illustration of the ball-on-plate task

We propose a new policy search meta-algorithm called Simulation-based Policy Optimization with Transferability Assessment (SPOTA) that uses the estimated SOB as a measure of the policy's transferability between domains (i.e., realizations of the randomized simulator). SPOTA enables robots to learn a control policy from a randomized source domain such that it can directly operate in an unknown target domain. While prior methods mostly train one policy for a given number of steps with a fixed number of domains, SPOTA progressively increases the number of domains while training and employs an Upper Confidence bound on the Simulation Optimization Bias (UCSOB) as a stopping criterion. As soon as the UCSOB is lower than a provided threshold of trust, the policy search meta-algorithm terminates. We employ a Monte-Carlo estimator of the SOB in the context of RL and show its expected monotonous decrease with the number of samples.

One of the main benefits of SPOTA is the easily interpretable stopping criterion, which directly follows from formulating the RL problem as a stochastic program. Another benefit of SPOTA is that it can be wrapped around existing policy search algorithms (e.g., Trust Region Policy Optimization (TRPO) [16], Relative Entropy Policy Search [17], or Deep Deterministic Policy Gradient [6]) and thereby augment them with the concept of domain randomization without the need to change the existing policy search algorithm.

We validate the proposed algorithm on the so-called 'ball-on-plate' problem, a balancing task in which the robot has to stabilize a ball at the center of a plate depicted in Figure 1. Using a simulation of the ball-on-plate problem, we compare our approach against the well-studied Linear-Quadratic Regulator (LQR), TRPO [16], and a policy search meta-algorithm called Ensemble Policy Optimization (EPOpt) [11]. Our last contribution is a cross-evaluation between the Vortex [18] and the Bullet [19] physics engine in the scope of domain randomization.

1.1 Related Work

Hereinafter, we review excerpts of the literature regarding the transfer of control policies from simulation to reality, the concept of SOB in stochastic programs, and the application of randomized physics simulations.

Reality Gap. Physics simulations have already been successfully used 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. The mismatch between the simulated and the real world has been addressed by robotics researchers from different viewpoints. Prominent examples are (i) randomly perturbing the observations and actions by adding i.i.d. noise [1, 20], (ii) model generation and selection depending on the short-term state-action history [21], and (iii) learning a transferability function [22].

Simulation Optimization Bias. Hobbs and 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 2 belongs to the class of Stochastic Programs (SPs) for which the assumption required in [2] are guaranteed to hold. The most common approaches to solve convex SPs are Sample Average Approximation (SAA) methods, including: (i) the Multiple Replications Procedure and its derivatives [3, 23] which assess a solution's quality by comparing with sampled alternative solutions, (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 SAA problem's minimizers.

Domain Randomization. There is a large consensus that further increasing the simulator's accuracy 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 that determine the physics simulator, 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 [27, 28, 10, 11, 15, 29, 14, 12, 13, 30, 31]. Wang et al. [27] 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 Neural Network (FNN) policy for under-actuated problem [28], (iii) using a risk-averse objective function [11], (iv) employing recurrent NN policies trained with experience replay [15], (v) optimizing a policy from samples of a model randomly that is chosen from an ensemble which is repeatedly fitted to world data [30]. From the listed approaches [10, 28, 15] were able to cross the reality gap without using samples from the real world. Domain randomization is also applied in computer vision. One example is the work by Tobin et al. [29] 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 and Levine [31] 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.

Adversarial Perturbations. Another approach of learning robust policies in simulation is to apply adversarial disturbances to the training process. Mandlekar 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.

1.2 Problem Statement and Notation

We consider a time-discrete dynamical system given by

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

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 physics parameters $\boldsymbol{\xi} \in \mathbb{R}^{n_\xi}$ (e.g., masses, friction coefficients, or time delays) define the environment a.k.a. the domain. They also parametrize the transition probability density function $\mathcal{P}_{\boldsymbol{\xi}} \colon \mathcal{S}_{\boldsymbol{\xi}} \times \mathcal{A}_{\boldsymbol{\xi}} \times \mathcal{S}_{\boldsymbol{\xi}} \to \mathbb{R}^+$ which describes the system's stochastic dynamics. The initial state $\boldsymbol{s}_0$ is drawn from the distribution $\mu_{0,\boldsymbol{\xi}} \colon \mathcal{S}_{\boldsymbol{\xi}} \to \mathbb{R}^+$. In order to formulate the system from (1) as a Markov Decision Process (MDP), we further define a deterministic reward function $r \colon \mathcal{S}_{\boldsymbol{\xi}} \times \mathcal{A}_{\boldsymbol{\xi}} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$. Finally, a MDP is fully described by the tuple $\mathcal{M}_{\boldsymbol{\xi}} \coloneqq \left\langle \mathcal{S}_{\boldsymbol{\xi}}, \mathcal{A}_{\boldsymbol{\xi}}, \mathcal{P}_{\boldsymbol{\xi}}, \mu_{0,\boldsymbol{\xi}}, r, \gamma \right\rangle$. Simulators can be obtained by implementing a set of physics laws and estimating their associated parameters by system identification. It is important to keep in mind, that even if this procedure yields a very accurate model parameter estimate, simulators are nevertheless just approximations of the real world and are thus always flawed. Here, the physics parameters are drawn from a probability distribution $\boldsymbol{\xi} \sim p\left(\boldsymbol{\xi}; \boldsymbol{\psi}\right)$, parametrized by $\boldsymbol{\psi}$ (e.g., mean, variance). As done in [10, 11, 14, 15], we use this distribution as a prior that ensures the physical plausibility of each parameter. Additionally, using the Gauss-Markov theorem one could also compute the parameters' covariance and hence construct a normal distribution for each physics parameter. Either way, specifying the distribution $p\left(\boldsymbol{\xi}; \boldsymbol{\psi}\right)$ in the current state-of-the-art requires the researcher to make design decisions.

In general, the goal of an 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 $\pi\left(\boldsymbol{a}_t \middle| \boldsymbol{s}_t; \boldsymbol{\theta}\right)$, characterized by its parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, is defined as

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

The resulting state-action pairs are collected in trajectories a.k.a. rollouts $\boldsymbol{\tau} = \{\boldsymbol{s}_t, \boldsymbol{a}_t\}_{t=0}^{T-1}$.

2 Simulation-Based Policy Optimization with Transferability Assessment

We introduce Simulation-based Policy Optimization with Transferability Assessment (SPOTA), a new policy search meta-algorithm that returns a set of policy parameters such that the resulting policy is able to directly transfer from an ensemble of source domains to an unseen target domain. The key novelty is the utilization of an Upper Confidence bound on the Simulation Optimization Bias (UCSOB) as a stopping criterion for the training procedure of an RL agent. The goal of SPOTA is not only to maximize the agent's expected discounted return under the influence of perturbed physics simulations, but also to provide an approximate probabilistic guarantee on the loss in terms of expected discounted return when applying the found policy to a different domain.

We aim to augment the standard RL setting with the concept of domain randomization, i.e. we want to maximize the expectation of the expected discounted return over all (feasible) realizations of the source domain

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\xi}}\left[J(\boldsymbol{\theta}, \boldsymbol{\xi})\right]. \tag{2}$$

This score quantifies how well the policy would perform over an infinitely large ensemble of variations of the nominal world $\mathcal{M}_{\bar{\boldsymbol{\xi}}}$. When training exclusively in simulation, we do not know the true physics model, and thus do not have access to the true $J(\boldsymbol{\theta})$ from (2). Instead, we maximize the estimated expected return from samples obtained by a randomized physics simulator. Thereby we update the policy parameters $\boldsymbol{\theta}$ according to a policy optimization algorithm. The inevitable imperfections of physics simulations will be exploited by any policy search method if it could thereby achieve a 'virtual' improvement, i.e., an increase of $J(\boldsymbol{\theta})$ in simulation. To counter this undesirable behavior, we define a Stochastic Program (SP)

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

where $\boldsymbol{\theta}$ is the decision variable, $\Theta \subseteq \mathbb{R}^{n_\theta}$ is the associated feasible set, $\boldsymbol{\xi}$ is a random variable, and $J(\boldsymbol{\theta}, \boldsymbol{\xi})$ is a real-valued function. The SP above can be approximated by

$$\hat{J}_n(\boldsymbol{\theta}_n^\star) = \max_{\boldsymbol{\theta}\in\Theta} \frac{1}{n} \sum\nolimits_{i=1}^{n} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i) \tag{3}$$

where the expectation is replaced by Monte-Carlo average over the sampled parameters $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$, and $\boldsymbol{\theta}_n^\star$ is the solution to the approximated SP. For the algorithm proposed in this paper, we require the same mild assumptions as in [3]. Framing the RL problem in (2) as an SP, allows for the utilization of the SOB as the convergence criterion for the policy search meta-algorithm introduced in Section 2. The SOB at a solution candidate $\boldsymbol{\theta}^c$ is defined as

$$G(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{\xi}}\left[J(\boldsymbol{\theta}, \boldsymbol{\xi})\right] - \mathbb{E}_{\boldsymbol{\xi}}\left[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})\right] \geq 0, \tag{4}$$

where the first term is the SP's optimal objective function value and the second term is the SP's objective function evaluated at the candidate solution. In order to use the SOB in our approach, we estimate an upper bound

$$\hat{G}_n(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}\in\Theta} \hat{J}_n(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c) \geq G(\boldsymbol{\theta}^c), \tag{5}$$

applying the Monte-Carlo approximation of $J(\boldsymbol{\theta})$ and $J(\boldsymbol{\theta}^c)$ with n i.i.d. samples of the random physics parameters $\boldsymbol{\xi}$. In the Appendix A, we derive the formulation of an upper bound on the SOB and show its monotonous decrease with increasing sample size. Moreover, we show that the approximation $\hat{G}_n(\boldsymbol{\theta}^c)$ can not underestimate the true SOB $G(\boldsymbol{\theta}^c)$, hence it is a safe approximation of the stopping criterion for our algorithm SPOTA. Note that the SOB as defined in (4) always exists unless the solution candidate $\boldsymbol{\theta}^c$ is a global optimum and the difference $G(\boldsymbol{\theta}^c) - \hat{G}_n(\boldsymbol{\theta}^c)$ will only diminish for an infinite number of samples, i.e., $n \to \infty$.

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., an ensemble 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 $\bar{\boldsymbol{\xi}}$. 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 the references' training set but unknown to the candidates. As inputs, we assume a given probability distribution over the physics parameters $p\left(\boldsymbol{\xi}; \boldsymbol{\psi}\right)$, a policy optimization sub-routine `BatchPolOpt` (e.g., TRPO), the batch sizes n_c, n_r, n_τ, n_G, n_J in conjunction with a nondecreasing sequence `NonDecrSeq` (e.g., $n_{k+1} = 2n_k$) for n_c and n_r, the confidence level $(1-\alpha)$, the threshold

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

input : probability distribution $p(\boldsymbol{\xi};\boldsymbol{\psi})$, algorithm `BatchPolOpt`, sequence `NonDecrSeq`, hyper-parameters n_c, n_r, n_τ, n_G, n_J, n_b, α, β

output: policy $\pi\left(\boldsymbol{\theta}^\star_{n_c}\right)$ with an $(1-\alpha)$-level confidence on $\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)$ that is at maximum β

1 Initialize $\pi\left(\boldsymbol{\theta}_{n_c}\right)$ randomly

2 **do** ▷ `next epoch`

3 Sample n_c i.i.d. physics simulators described by $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{n_c}$ from $p(\boldsymbol{\xi};\boldsymbol{\psi})$

4 Solve the approx. SP using $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{n_c}$ and `BatchPolOpt` to obtain $\boldsymbol{\theta}^\star_{n_c}$ ▷ `candidate`

5 **for** $k = 1, \ldots, n_G$ **do**

6 Sample n_r i.i.d. physics simulators described by $\boldsymbol{\xi}^k_1, \ldots, \boldsymbol{\xi}^k_{n_r}$ from $p(\boldsymbol{\xi};\boldsymbol{\psi})$

7 Initialize $\boldsymbol{\theta}^k_{n_r}$ with $\boldsymbol{\theta}^\star_{n_c}$ and reset the exploration strategy

8 Solve the approx. SP using $\boldsymbol{\xi}^k_1, \ldots, \boldsymbol{\xi}^k_{n_r}$ and `BatchPolOpt` to obtain $\boldsymbol{\theta}^{k\star}_{n_r}$ ▷ `reference`

9 **for** $i = 1, \ldots, n_r$ **do**

10 **with** synchronized random seeds ▷ `sync. init. states and obs. noise`

11 Estimate the candidate solution's return $\hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right) \leftarrow 1/n_J \sum_{j=1}^{n_J} \hat{J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right)$

12 Estimate the reference solution's return $\hat{J}_{n_J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right) \leftarrow 1/n_J \sum_{j=1}^{n_J} \hat{J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right)$

13 **end**

14 Compute the difference in return $\hat{G}^k_{n_r,i}\left(\boldsymbol{\theta}^\star_{n_c}\right) \leftarrow \hat{J}_{n_J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right) - \hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right)$

15 **if** $\hat{G}^k_{n_r,i}\left(\boldsymbol{\theta}^\star_{n_c}\right) < 0$ **then** $\hat{G}^k_{n_r,i}\left(\boldsymbol{\theta}^\star_{n_c}\right) \leftarrow 0$ ▷ `outlier rejection`

16 **end**

17 **end**

18 Bootstrap n_b times from $\mathcal{G} = \{\hat{G}^1_{n_r,1}\left(\boldsymbol{\theta}^\star_{n_c}\right), \ldots, \hat{G}^{n_G}_{n_r,n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)\}$ to yield ${}^*\mathcal{G}_1, \ldots, {}^*\mathcal{G}_{n_b}$

19 Compute the sample mean $\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)$ for the original set $\mathcal{G}$

20 Compute the sample means ${}^*\bar{G}_{n_r,1}\left(\boldsymbol{\theta}^\star_{n_c}\right), \ldots, {}^*\bar{G}_{n_r,n_b}\left(\boldsymbol{\theta}^\star_{n_c}\right)$ for the sets ${}^*\mathcal{G}_1, \ldots, {}^*\mathcal{G}_{n_b}$

21 Select the α-th quantile of the bootstrap samples' means and obtain the upper bound for the one-sided $(1-\alpha)$-level confidence interval $\bar{G}^U_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right) \leftarrow 2\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right) - Q_\alpha\left[{}^*\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)\right]$;

22 Set the new sample sizes $n_c \leftarrow$ `NonDecrSeq`(n_c) and $n_r \leftarrow$ `NonDecrSeq`(n_r)

23 **while** $\bar{G}^U_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right) > \beta$

of trust β, and the number of bootstrap samples n_b. SPOTA consists of four blocks: (i) finding a candidate solution, (ii) finding multiple reference solutions, (iii) comparing the candidate against the reference solutions, and (iv) assessing the candidate solution quality.

Candidate Solution. First, a randomly initialized candidate solution is optimized based on an ensemble of n_c source domains (Lines 3 to 4). Practically, the locally optimal policy parameters are optimized on the sample-based approximation (3).

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

Solution Comparison. Each reference solution $\boldsymbol{\theta}^{k\star}_{n_r}$ with $k = 1, \ldots, n_G$ gets evaluated 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 $\hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right)$ and $\hat{J}_{n_J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right)$ are estimated from n_J Monte-Carlo simulations with synchronized random seeds (Lines 10 to 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

$$\hat{G}^k_{n_r,i}\left(\boldsymbol{\theta}^\star_{n_c}\right) = \hat{J}_{n_J}\left(\boldsymbol{\theta}^{k\star}_{n_r}, \boldsymbol{\xi}^k_i\right) - \hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}^k_i\right) \tag{6}$$

may become negative. This issue did not appear in previous work on assessing solution qualities of SPs [3, 23], because they only covered convex problems, where all reference solutions are guaranteed to be global optima. Utilizing the definition of the SOB in (5) 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, all negative samples of the approximated SOB are clipped to zero (Line 15). Alternatives for this method of processing the negative samples are discussed in the Appendix C.

Solution Quality. Next, a $(1-\alpha)$-level confidence interval $\left[0, \bar{G}^U_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)\right]$ for the estimated SOB at $\boldsymbol{\theta}^\star_{n_c}$ is constructed. While the lower confidence bound is fixed to the theoretical minimum, the Upper Confidence bound on the Simulation Optimization Bias (UCSOB) is computed using the statistical bootstrap method [32]. Regarding the statistical bootstrap method, we denote bootstrapped quantities with a left superscript asterisk and optima yielded by solving optimization problems with a right superscript star. There are multiple ways to yield a confidence interval by applying the bootstrap [33]. Here, the 'basic' nonparametric method was chosen, since the aforementioned 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. Applying (6) to each reference solution and realizations yields a set of $n_G n_r$ samples of the approximated SOB $\mathcal{G} = \{\hat{G}^1_{n_r,1}\left(\boldsymbol{\theta}^\star_{n_c}\right), \ldots, \hat{G}^{n_G}_{n_r,n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)\}$. Through uniform random sampling with replacement from $\mathcal{G}$, we generate n_b bootstrap samples ${}^*\mathcal{G}_1, \ldots, {}^*\mathcal{G}_{n_b}$. Thus, for our statistic of interest, the mean approximated SOB $\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)$, the UCSOB becomes

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

where $\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)$ is the mean over all (nonnegative) samples from the empirical distribution, and $Q_\alpha\left[{}^*\bar{G}_{n_r}\left(\boldsymbol{\theta}^\star_{n_c}\right)\right]$ is the α-th quantile of the means calculated for each of the n_b bootstrap samples (Lines 19 to 21). Consequently, the true SOB is covered by the obtained one-sided confidence interval with the approximate probability of $(1-\alpha)$, i.e.,

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

which is analogous to (4) in [23]. Finally, the sample sizes n_r and n_c of the next epoch are set according to the nondecreasing sequence. The procedure stops if the UCSOB 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 one should 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 1.1 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 UCSOB 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 that 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 SOB and therefore lose the main feature of SPOTA.

The hyper-parameters chosen for the experiments in Section 3 as well as further details on the implementation of SPOTA can be found in the Appendix C.

3 Experiments

Within this section we introduce the ball-on-plate task (Figure 1), a balancing task in which the agent has to stabilize a ball at the center of a plate which is attached to a robotic arm. The agent sends task-space acceleration commands to the simulated robot, while the robot's joints are controlled by an inverse kinematics algorithm and low-level PD-controllers. To demonstrate the applicability of SPOTA, we compare it against a LQR and NN policies trained by TRPO as well as EPOpt. We first describe the system's physical modeling as well as the setup. Next, we explain the conducted experiments and finally summarize their results.

3.1 Modeling and Setup Description

The state is defined as $\boldsymbol{s} = [\alpha_p, \beta_p, x_b, y_b, z_b - r_b, \dot{\alpha}_p, \dot{\beta}_p, \dot{x}_b, \dot{y}_b, \dot{z}_b]^\mathsf{T}$, where α_p, β_p are the plate's angles around the x- and y-axis of the inertial frame receptively, x_b, y_b, z_b are the ball's Center of Mass (CoM) position w.r.t. the plate's frame, which is located at the plate's center, and r_b is the ball's radius. Accordingly, the actions are defined as $\boldsymbol{a} = [\ddot{\alpha}_p, \ddot{\beta}_p]^\mathsf{T}$. A picture of the setup showing the reference frames can be found in the Appendix B. We use an exponential reward function, where the exponent is a weighted sum of squared state errors and actions.

All our simulations are set up in the Rcs framework [34]. The robotic arm is mounted on the ground and initialized holding the plate upright with the ball on top. To avoid singularities, the robot's initial pose is set to be not fully stretched out (Figure 1). The ball's initial x-y position on the plate

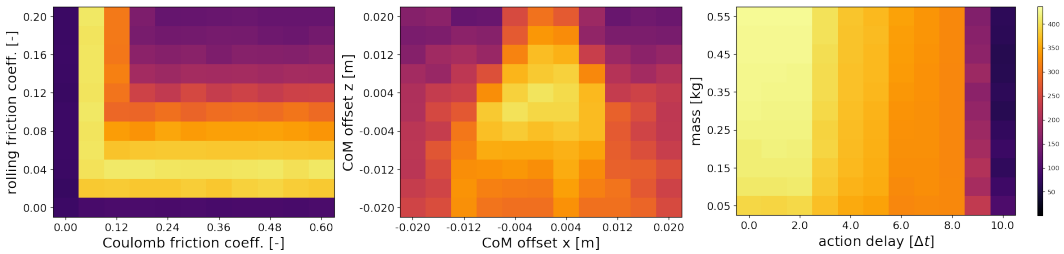


Figure 2: Performance measured in average return of a policy trained with SPOTA, evaluated on grids of various instances of the ball-on-plate task. The UCSOB of this policy was 55.34. Note, that the plotted parameter range exceeds the one experienced during training. The reported values were generated with Vortex by executing 180 rollouts for each grid cell using different initial states.

is drawn from a manifold defined by the space between two concentric circles around the plate's center ensuring that all trajectories start with a similar distance to the goal ($\boldsymbol{s} = \mathbf{0}, \boldsymbol{a} = \mathbf{0}$). As a consequence the variance of the returns is reduced and learning is facilitated. While training, we add zero-mean i.i.d. normal noise with constant covariance to the observations, whereas for testing no noise is injected. The policies' hyper-parameters are documented in the Appendix C.

3.2 Experiments Description

We conducted three experiments on the ball-on-plate setup using (perturbed) physics simulations:

1. evaluating one SPOTA policy on multiple 2D-grids of simulator parameters,
2. comparing LQR, TRPO, EPOpt, and SPOTA policies while varying one simulator parameter,
3. cross-evaluation of policies trained in Vortex and in Bullet then tested in both.

In all experiments, the agent's goal is to stabilize the randomly initialized ball at the plate's center. For the sake of comparability, we measured the performance of each policy using the sum of rewards, called (undiscounted) return. Note that the LQR is only optimal for linear systems and quadratic cost functions, hence we can not expect it to perform best but are nevertheless interested in a well-known baseline from classical control. The motivation of the experiments is to find out which policy is able to transfer to unseen environments. Roughly speaking, the return values $R = \sum_{t=0}^{T-1} r_t$ can be categorized as follows: $R > 350$ excellent performance (fast stabilization of the ball at the center), $R \in [300, 350]$ good performance (ball stabilized in the center at latest on $t = T$), $R \in [200, 300[$ mediocre performance (mostly due to oscillations around the center), $R \in [100, 200[$ bad performance (borderline stable or unstable behavior), $R < 100$ complete failure.

3.3 Results

The following figures summarize the results obtained from the experiments described in the previous section. Additional videos can be found at `https://www.ias.informatik.tu-darmstadt.de/Team/FabioMuratore`.

Experiment 1 conducts a sensitivity analysis of a policy trained using SPOTA w.r.t. changes of the domain. In Figure 2, the performance is plotted as a heat map across grids of configurations generated by varying two simulator parameters simultaneously. It can be seen that SPOTA is able to handle significant changes in sensitive parameters (e.g., CoM offset), as well as every test case for insensitive parameters (e.g., ball mass). Task failures occur for example, when both friction coefficients are too high, i.e., the plate's deflection angle induced by the policy is too small to get the ball rolling. A common cause for complete failure is very slippery environments with high action delay, which leaves little room for corrections computed on the current state feedback.

Experiment 2 provides a comparison of different control policies' robustness against model uncertainties. Figure 3 shows the dependency of the achieved return on varying a set of selected simulator parameters. On the LQR side, there is potentially high performance, but total trust in the dynamics model. This can be observed regarding the action delay (Figure 3 – right) which is assumed to be zero in the LQR model. The TRPO policy trained without domain randomization behaves similarly to the LQR in most cases. In contrast, the SPOTA policy is able to maintain its performance across a wider range of parameter values. Regarding the variation of the ball's rolling friction coefficient (Figure 3 – middle), it can be seen that the risk-averse EPOpt procedure leads to higher robustness for a limited subset of the possible problem instances (e.g., very low rolling friction). This effect can be explained by the fact that EPOpt optimizes the conditional value at risk of the return [11]. On the other side, the risk-neutral approaches outperform the EPOpt in most other cases.

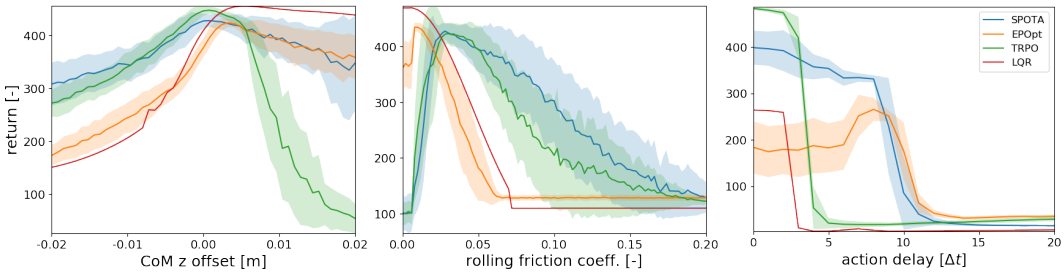


Figure 3: Performance measured in return of SPOTA, EPOpt, TRPO, and LQR policies when varying the ball's CoM offset in z direction (left), the ball's rolling friction coefficient (middle), and the policy's action delay (right).[1]

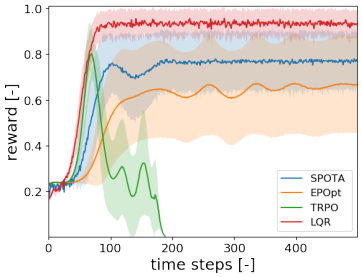


Figure 4: Cross-evaluation of SPOTA, EPOpt TRPO, and LQR policies trained in Vortex and then tested in Bullet. The simulators were set up to maximize the similarity between the physics engines as much as possible. Moreover, the simulator parameters used for evaluating are the same as for determining the LQR and TRPO policy, and equal to the nominal parameters used for the SPOTA as well as EPOpt procedure.[1]

Experiment 3 investigates if the application of domain randomization improves the transferability of a control policy between two different physics engines. The results in Figure 4 confirm two hypotheses. First, learned policies perform worse when tested using a different physics engine. Second, domain randomization alleviates this effect. The LQR baseline does well in both evaluations since the depicted rollouts are based on the nominal parameter values, and the LQR's feedback gains are not learned from samples, i.e., independent of physics engine. Compared to policy trained with SPOTA or EPOpt, the TRPO policy is not able to maintain the level of performance. Note that the definition SOB (4) requires the candidate's and references' simulator parameters to be from the same distribution. Practically, this assumption is violated as soon as one switches the physics engine, since the some parameters, e.g., the friction coefficients, are processed differently.
In conclusion, the results show that, compared to policies which were trained for a single fixed simulator, domain randomization algorithms like SPOTA are better at maintaining their level of performance across a variety of different domains.

4 Conclusion and Future Work

We presented a new policy search meta-algorithm called Simulation-based Policy Optimization with Transferability Assessment (SPOTA) which is able to learn a control policy that directly transfers from a randomized source domain to an unseen target domain. The gist is to frame the training over an ensemble of models as a stochastic program and to use an upper confidence bound on the estimated Simulation Optimization Bias (SOB) as stopping criterion for the training process. Furthermore, the resulting SOB can be interpreted as a measure of the obtained policy's robustness to variations of the source domain. To the best of our knowledge, SPOTA is the only domain randomization approach that provides this quantitative measure for over-fitting to the domains experienced in the training phase. This measure is of high importance, since sample-based optimization is always optimistically biased [2, 3]. We evaluated our method as well as three baselines on a simulation of the introduced ball-on-plate task, a robotic balancing task. The results show that policies trained with SPOTA are able to generalize to unknown target domains, while baselines acquired without domain randomizations fail.
In future work we will test SPOTA on the real-world counterpart of the ball-on-plate task. Apart from that, we plan to investigate modifications to the presented algorithm such as using adaptive probability distributions for sampling the simulator parameters while training. This would allow to sample according to an objective, e.g., maximizing the information gain.

[1] The reported values were generated with Vortex (Figure 3) or Bullet (Figure 4) by executing 180 rollouts for each parameter value, using identical equally-spaced initial states, and the same random seeds for all policies. The solid lines indicate the mean and the shaded areas cover ± 1 standard deviation.

Acknowledgment

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.
Fabio Muratore wants to thank David P. Morton for his quick and helpful answers.

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, Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995, Proceedings*, pages 704–720. Springer, 1995. URL https://doi.org/10.1007/3-540-59496-5_337.

[2] B. F. Hobbs and A. Hepenstal. Is optimization optimistically biased? *Water Resources Research*, 25(2):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.*, 24(1-2):47–56, 1999. URL https://doi.org/10.1016/S0167-6377(98)00054-6.

[4] 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*, 518(7540):529–533, 2015. URL http://dx.doi.org/10.1038/nature14236.

[5] 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*, 550(7676):354, 2017.

[6] 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. URL http://arxiv.org/abs/1509.02971.

[7] S. James and E. Johns. 3d simulation for robot arm control with deep q-learning. *ArXiv e-prints*, 2016. URL http://arxiv.org/abs/1609.03759.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv e-prints*, 2017. URL http://arxiv.org/abs/1707.06347.

[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*, pages 262–270, 2017. URL http://proceedings.mlr.press/v78/rusu17a.html.

[10] 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*, pages 5307–5314, 2015. URL https://doi.org/10.1109/IROS.2015.7354126.

[11] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran. Epopt: Learning robust neural network policies using model ensembles. *ArXiv e-prints*, 1610.01283, 2016. URL http://arxiv.org/abs/1610.01283.

[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*, pages 3932–3939, 2017. URL https://doi.org/10.1109/IROS.2017.8206245.

[13] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. In *ICML, Sydney, NSW, Australia, August 6-11*, pages 2817–2826. PMLR, 2017. URL http://proceedings.mlr.press/v70/pinto17a.html.

[14] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *ArXiv e-prints*, 2017. URL http://arxiv.org/abs/1710.06542.

[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. URL http://arxiv.org/abs/1710.06537.

[16] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *ICML, Lille, France, July 6-11*, volume 37, pages 1889–1897, 2015. URL http://jmlr.org/proceedings/papers/v37/schulman15.html.

[17] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, July 11-15*, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1851.

[18] Vortex physics engine. Online. URL https://www.cm-labs.com/vortex-studio/.

[19] Bullet physics engine. Online. URL bulletphysics.org/wordpress/.

[20] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behaviour*, 6(2):325–368, 1997. URL https://doi.org/10.1177/105971239700600205.

[21] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[22] S. Koos, J. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Trans. Evolutionary Computation*, 17(1):122–145, 2013. URL https://doi.org/10.1109/TEVC.2012.2185849.

[23] G. Bayraksan and D. P. Morton. Assessing solution quality in stochastic programs. *Math. Program.*, 108(2-3):495–514, 2006. URL https://doi.org/10.1007/s10107-006-0720-x.

[24] R. Pasupathy and B. W. Schmeiser. Retrospective-approximation algorithms for the multidimensional stochastic root-finding problem. *ACM Trans. Model. Comput. Simul.*, 19(2):5:1–5:36, 2009. URL http://doi.acm.org/10.1145/1502787.1502788.

[25] S. Kim, R. Pasupathy, and S. G. Henderson. A guide to sample average approximation. In *Handbook of Simulation Optimization*, pages 207–243. Springer, 2015.

[26] F. Bastin, C. Cirillo, and P. L. Toint. Convergence theory for nonconvex stochastic programming with an application to mixed logit. *Math. Program.*, 108(2-3):207–234, 2006. URL https://doi.org/10.1007/s10107-006-0708-6.

[27] J. M. Wang, D. J. Fleet, and A. Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29(4):73:1–73:8, 2010. URL http://doi.acm.org/10.1145/1833351.1778810.

[28] R. Antonova and S. Cruciani. Unlocking the potential of simulators: Design with RL in mind. *ArXiv e-prints*, 2017. URL http://arxiv.org/abs/1706.02501.

[29] 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*, pages 23–30, 2017. URL https://doi.org/10.1109/IROS.2017.8202133.

[30] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *ArXiv e-prints*, 2018. URL http://arxiv.org/abs/1802.10592.

[31] F. Sadeghi and S. Levine. CAD2RL: real single-image flight without a single real image. In *RSS XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16*, 2017. URL http://www.roboticsproceedings.org/rss13/p34.html.

[32] B. Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, pages 1–26, 1979.

[33] T. J. DiCiccio and B. Efron. Bootstrap confidence intervals. *Statistical Science*, pages 189–212, 1996.

[34] Rcs framework. Online. URL https://github.com/HRI-EU/Rcs.

– Appendix –
Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment

A Additional Details on the Simulation Optimization Bias

In this section, we derive an upper bound on the SOB in the context of RL and show its monotonic decrease with increasing number of samples from the random variable. In what follows, we build upon the results of [2, 3] where the SOB appears under the name 'optimality gap'.

A.1 Estimation of the Simulation Optimization Bias

Consider a real-valued function $J(\boldsymbol{\theta}, \boldsymbol{\xi})$ quantifying the expected (discounted) return of a policy, determined by the policy's parameters $\boldsymbol{\theta}$, the domain's parameters $\boldsymbol{\xi}$, and the initial state distribution (omitted for a more concise notation). The optimal solution of the true Stochastic Program (SP) is denoted as $\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})]$. Likewise, the optimal solution of the approximated SP is denoted as $\boldsymbol{\theta}_n^\star = \arg\max_{\boldsymbol{\theta}} \hat{J}_n(\boldsymbol{\theta})$, where $\hat{J}_n(\boldsymbol{\theta}) = 1/n \sum_{i=1}^{n} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i)$ is the Monte-Carlo approximation of $J(\boldsymbol{\theta})$ using n samples of $\boldsymbol{\xi}$. The SOB at the candidate solution $\boldsymbol{\theta}^c$

$$G(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})] \geq 0 \tag{7}$$

expresses the difference between the optimal value of the true SP and its value at the candidate solution. When replacing the arbitrary $\boldsymbol{\theta}^c$ with $\boldsymbol{\theta}_n^\star$, the inequality above reveals that $\boldsymbol{\theta}_n^\star$ is a biased estimator of $\boldsymbol{\theta}^\star$. For i.i.d. samples $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$, we can write

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}, \boldsymbol{\xi})] = \max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_n(\boldsymbol{\theta})\right] \tag{8}$$

Plugging (8) into the first term of (7) yields

$$G(\boldsymbol{\theta}^c) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_n(\boldsymbol{\theta})\right] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})] \leq \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}} \hat{J}_n(\boldsymbol{\theta})\right] - \mathbb{E}_{\boldsymbol{\xi}}[J(\boldsymbol{\theta}^c, \boldsymbol{\xi})] \tag{9}$$

as an upper bound to the SOB. In order to compute this upper bound, we use the law of large numbers for the first term and replace the second expectation in (9) with the sample average

$$G(\boldsymbol{\theta}^c) \leq \max_{\boldsymbol{\theta}} \hat{J}_n(\boldsymbol{\theta}) - \hat{J}_n(\boldsymbol{\theta}^c) = \hat{G}_n(\boldsymbol{\theta}^c)\,, \tag{10}$$

where $\hat{G}_n(\boldsymbol{\theta}^c) \geq 0$ holds. This result is consistent with Theorem 1 and equation (9) in [3] as well as the "type A error" mentioned in [2].

A.2 Decrease of the Estimated Simulation Optimization Bias with Increasing Sample Size

Next, we show that the SOB decreases in expectation when the sample size of the physics parameters $\boldsymbol{\xi}$ is increased. The expectation over $\boldsymbol{\xi}$ of the minuend in (10) estimated from $n+1$ i.i.d. samples is

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_{n+1}\big(\boldsymbol{\theta}_{n+1}^\star\big)\right] &= \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}} \frac{1}{n+1} \sum_{i=1}^{n+1} J(\boldsymbol{\theta}, \boldsymbol{\xi}_i)\right] = \mathbb{E}_{\boldsymbol{\xi}}\left[\max_{\boldsymbol{\theta}} \frac{1}{n+1} \sum_{i=1}^{n+1} \frac{1}{n} \sum_{j=1, j\neq i}^{n+1} J\big(\boldsymbol{\theta}, \boldsymbol{\xi}_j\big)\right] \\
&\leq \mathbb{E}_{\boldsymbol{\xi}}\left[\frac{1}{n+1} \sum_{i=1}^{n+1} \max_{\boldsymbol{\theta}} \frac{1}{n} \sum_{j=1, j\neq i}^{n+1} J\big(\boldsymbol{\theta}, \boldsymbol{\xi}_j\big)\right] = \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{J}_n(\boldsymbol{\theta}_n^\star)\right].
\end{aligned} \tag{11}$$

Taking the expectation over the SOB estimated from $n+1$ samples of $\boldsymbol{\xi}$ and plugging in the upper bound from (11), we obtain the upper bound

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

which shows that the estimator of the SOB in expectation monotonously decreases with increasing sample size. This result is consistent with Theorem 2 in [3].

B Further Modeling and Setup Description

The system's equations of motion can be derived using the Euler-Lagrange formalism. In order to model the dynamical system for the calculation of the LQR's feedback gains, we make the following assumptions: (i) there is no kinetic energy from the bodies rotational motions around their vertical axis, (ii) the ball rolls on the plate without slipping, (iii) there is only linear-viscous friction between the ball and the plate, and (iv) the ball and the plate are symmetric and homogeneous rigid bodies.

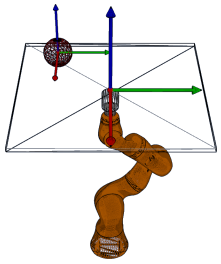


Figure 5: Wireframe display of the ball-on-plate system. The x, y, and z axes are colored in red, green, and blue, respectively.

The system state is defined as $\boldsymbol{s} = [\alpha_p, \beta_p, x_b, y_b, z_b - r_b, \dot{\alpha}_p, \dot{\beta}_p, \dot{x}_b, \dot{y}_b, \dot{z}_b]^\mathsf{T}$, where α_p, β_p are the plate's angles around the x- and y-axis of the inertial frame respectively, x_b, y_b, z_b are the ball's CoM position w.r.t. the plate's frame, which is located at the plate's center, and r_b is the ball's radius. Accordingly, the actions are defined as $\boldsymbol{a} = [\ddot{\alpha}_p, \ddot{\beta}_p]^\mathsf{T}$ in the plate's frame. The obtained model is linearized around the equilibrium $(\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{a} = \boldsymbol{0})$. Finally, the feedback gains of the LQR are computed by solving the discrete-time Riccati equation, using the weight matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ for the state errors and actions, respectively. The same weight matrices are used to define the reward function for the policies trained by RL algorithms.

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

Given a lower bound for the reward $r_{min} \in [0, 1]$, the reward function above yields values in $[0, 1]$ 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, specified in the problems' associated Rcs simulation environment[2]. The ball's initial position on the plate is drawn from a manifold defined by the space between two concentric circles around the plate's center

$$[x_{b,0}, y_{b,0}]^\mathsf{T} = d[\cos(\phi), \sin(\phi)]^\mathsf{T} \text{ with } d \sim \mathcal{U}\left(d\middle|d_L, d_U\right) \text{ and } \phi \sim \mathcal{U}\left(\phi\middle|0, 2\pi\right),$$

ensuring that all trajectories start with a similar distance to the goal $(\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{a} = \boldsymbol{0})$. As a consequence, the variance of the returns is reduced, hence learning is facilitated. To provide a fair comparison, all policies receive the same state observations as inputs. While training, we add zero-mean i.i.d. normal noise with constant covariance to the observations $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(\boldsymbol{\varepsilon}\middle|\boldsymbol{0}, \boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}}\right)$, whereas for testing no noise is injected.

We employed the Vortex [18] and the Bullet [19] physics engine to simulate the rollouts for training and evaluation. As to be expected, the accessible physics parameters for Vortex and Bullet vary. The most significant difference are the physics engines' contact models, especially regarding friction. While Vortex offers provides the option to specify material tables as well as contact materials for specific pairings of materials, Bullet only allows for the specification of a subset of material properties available in Vortex, without the possibility to create custom exceptions. Another difference is between the engine-specific simulation damping factors, which were left unchanged for all experiments. Unfortunately, Vortex is closed source and Bullet is lacking proper documentation, hence an in-depth investigation of the differences is prohibitive. However, our goal is neither to make a qualitative statement nor to provide a detailed comparison of the two physics engines, but rather to evaluate the transferability of the control polices.

C Complementary Implementation Details on SPOTA

As batch policy optimization sub-routine, called `BatchPolOpt` in Algorithm 1, we used the TRPO [16] implementation within the rllab framework[3]. In principle, any batch policy optimization algorithm is applicable for SPOTA.

For each iteration, $\lceil T/\Delta t \rceil n_\tau n$ samples are fed to the optimizer, where T is the maximum runtime of an episode, Δt is the simulation's step size, n_τ is a constant factor, and n is either n_c or n_c depending on which policy is currently optimized. In other words, each optimizer iteration receives as many time steps as n_τ full-length trajectories for n different domains consist of. This implies

[2] https://github.com/HRI-EU/Rcs

[3] https://github.com/rll/rllab

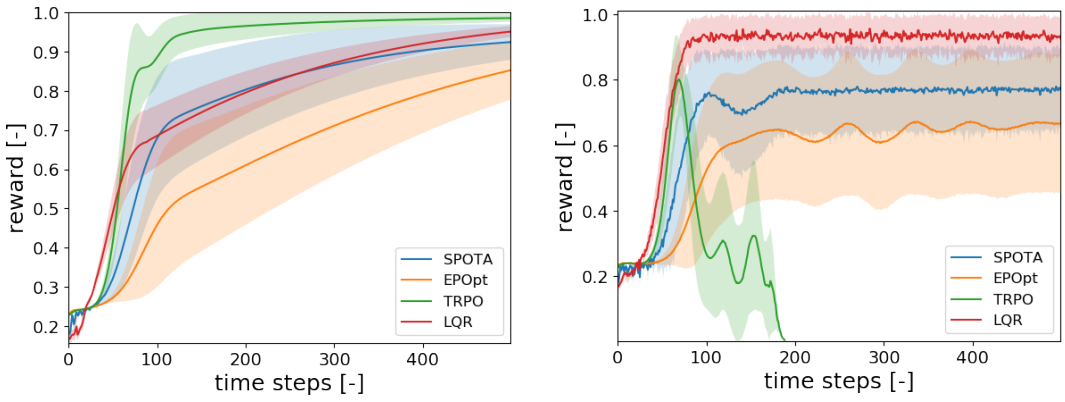
a nondecreasing batch size, which resembles the core idea of retrospective approximation methods [25] where the problem is solved to satisfy error tolerances converging to zero.
As described in Section 2, some samples of the SOB in (6) might be negative. Since we know that a policy's SOB can theoretically not be smaller than zero, we decided to clamp all negative SOB samples to zero. One alternative approach is to recompute negative samples using the other $n_G - 1$ reference solutions. The rationale behind this approach is that one of those reference solutions, although trained on a different domain, could perform better on the current domain than the candidate solution. One could argue that this approach yields better SOB estimates than just clamping them to zero. For simplicity and the reduction of computation time, we decided against this alternative.
The hyper-parameters values used for the experiments are listed in Table 1. Concerning EPOpt, we first used the hyper-parameters extracted from the original paper [11], but did not achieve satisfying results. Therefore, we tuned the number of optimizer iterations n_{iter}, the number of optimizer iterations in which all trajectories are used n_{iter}, the optimizer batch size, and the EPOpt-specific CVaR-parameter ϵ. In order to maintain a fair comparison, we similarly adapted the parameters of the TRPO baseline. Regarding the Coulomb friction, we set the static and the sliding friction coefficients equal.

Table 1: Hyper-parameter values for the experiments in Section 3. 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. Normal distributions, are parametrized with mean and standard deviation, uniform distributions with lower and upper bound.

Parameter	Value	Parameter	Distribution / Value
`BatchPolOpt`	TRPO [16]	ϵ (EPOpt)	0.2
policy architecture	FNN: 16-16 with tan-h nonlinearities	n_{skipiter} (EPOpt)	300
		ball mass [kg]	$\mathcal{N}\left(\xi_1 \middle\vert 0.2, 0.05\right)$
n_{iter} (optimizer)	SPOTA: 200 EPOpt & TRPO: 600	ball radius [m]	$\mathcal{N}\left(\xi_2 \middle\vert 0.08, 0.02\right)$
		ball CoM offset in x-, y-, z-direction [m]	$\mathcal{N}\left(\xi_3, \xi_4, \xi_5 \middle\vert 0, 8\text{e}{-3}\right)$
runtime T	10 s		
step size Δt	0.02 s	Coulomb frict. coeff. [-]	$\mathcal{U}\left(\xi_6 \middle\vert 0, 0.6\right)$
discount factor γ	0.998		
initial n_c (SPOTA)	2	rolling frict. coeff. [-]	$\mathcal{U}\left(\xi_7 \middle\vert 0, 0.01\right)$
initial n_r (SPOTA)	1	slip coeff. [s N^{-1} m^{-1}]	$\mathcal{U}\left(\xi_8 \middle\vert 0, 150\right)$
n_τ	SPOTA: 10 EPOpt & TRPO: 1	action delay [Δt]	$\mathcal{U}\left(\xi_9 \middle\vert 0, 6\right)$
		d_U [m]	0.45 plate width
n (EPOpt & TRPO)	240	d_L [m]	$d_U - 0.01$ m
batch size in steps	$\lceil T/\Delta t \rceil n_\tau n$	`NonDecrSeq`	$n_{k+1} \leftarrow n_0(k+1)$
n_G (SPOTA)	10	min reward $r_{\min}$	1e–4
n_J (SPOTA)	120	confidence level α	0.05
bootstrap samples n_b	1000	threshold of trust β	$0.15 \max_{\boldsymbol{\xi}} \hat{J}_{n_J}\left(\boldsymbol{\theta}^\star_{n_c}, \boldsymbol{\xi}\right)$
weight matrix $\boldsymbol{R}$	$\text{diag}(1\text{e}{-4}, 1\text{e}{-4})$		
weight matrix $\boldsymbol{Q}$	$\text{diag}\left(0.01, 0.01, 1, 1, 1, 1\text{e}{-3}, 1\text{e}{-3}, 0.1, 0.1, 0.1\right)$		
obs. noise cov. $\boldsymbol{\Sigma_\varepsilon}$	$\left(\text{diag}\left(5\text{e}{-4}, 5\text{e}{-4}, 5\text{e}{-4}, 5\text{e}{-4}, 5\text{e}{-4}, 0.01, 0.01, 0.01, 0.01, 0.01\right)\right)^2$		

D Supplementary Results of Experiment 3

In experiment 3, we investigated to what extent SPOTA, EPOpt, TRPO, and LQR policies are able to maintain their performance when evaluated using a different physics engine. Figure 6 displays the full cross-evaluation between Vortex and Bullet, i.e., the extended version of Figure 4. It can be seen from the figures that policies trained using domain randomization generalize better to the other physics engine. In particular the SPOTA policy is able to maintain the performance level of the LQR controller, which's parameters are invariant to the physics engine. Regarding the relatively low performance of the EPOpt policy, we want to add that it is common for this algorithm to yield lower rewards for the nominal domain parameters, but to be robust against variations of these parameters (not depicted in Figure 6). Interestingly, the evaluation in Vortex yields notably better results. One explanation for that could be that, despite using identical physics parameter values, we found that the simulation in Bullet appears to have less friction. Additional videos are provided at `https://www.ias.informatik.tu-darmstadt.de/Team/FabioMuratore`.



(a) Trained in Vortex, tested in Vortex

(b) Trained in Vortex, tested in Bullet

(c) Trained in Bullet, tested in Vortex

(d) Trained in Bullet, tested in Bullet

Figure 6: Cross-evaluation of SPOTA, EPOpt TRPO, and LQR policies trained in Vortex and then tested in Bullet. The simulators were set up to maximize the similarity between the physics engines as much as possible. Moreover, the simulator parameters used for evaluating are the same as used for determining the LQR and TRPO policy, and equal to the nominal parameters for the SPOTA as well as EPOpt procedure.