POLITECNICO DI TORINO
Repository ISTITUZIONALE

DROPO: Sim-to-real transfer with offline domain randomization

Original
DROPO: Sim-to-real transfer with offline domain randomization / Tiboni, Gabriele; Arndt, Karol; Kyrki, Ville. - In: ROBOTICS AND AUTONOMOUS SYSTEMS. - ISSN 0921-8890. - 166:(2023). [10.1016/j.robot.2023.104432]

Availability:
This version is available at: 11583/2982392 since: 2023-09-21T23:57:41Z

Publisher:
Elsevier

Published
DOI:10.1016/j.robot.2023.104432

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

02 May 2024

Robotics and Autonomous Systems 166 (2023) 104432

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

DROPO: Sim-to-real transfer with offline domain randomization

Gabriele Tiboni [a,b], Karol Arndt [b,*], Ville Kyrki [b]

[a] Department of Control and Computer Engineering, Politecnico di Torino, Francesco Ferrucci Street, 112, Torino, 10141, Italy
[b] Intelligent Robotics Group, Department of Electrical Engineering and Automation, Aalto University, Maarintie 8, Espoo, 02150, Finland

ARTICLE INFO

Article history:
Received 20 June 2022
Accepted 14 April 2023
Available online 12 May 2023

Keywords:
Robot learning
Transfer learning
Reinforcement learning
Domain randomization

ABSTRACT

In recent years, domain randomization over dynamics parameters has gained a lot of traction as a method for sim-to-real transfer of reinforcement learning policies in robotic manipulation; however, finding optimal randomization distributions can be difficult. In this paper, we introduce DROPO, a novel method for estimating domain randomization distributions for safe sim-to-real transfer. Unlike prior work, DROPO only requires a limited, precollected offline dataset of trajectories, and explicitly models parameter uncertainty to match real data using a likelihood-based approach. We demonstrate that DROPO is capable of recovering dynamic parameter distributions in simulation and finding a distribution capable of compensating for an unmodeled phenomenon. We also evaluate the method in two zero-shot sim-to-real transfer scenarios, showing successful domain transfer and improved performance over prior methods.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Over the past decade, there have been significant research advances in reinforcement learning (RL), resulting in a variety of success stories, from playing Atari games [1] to defeating the world champion in Go [2]. These successes have sparked a lot of interest in RL among roboticists; however, due to low sample efficiency of most algorithms, the challenge of accessing significant data collections at training time, as well as the need for random exploration resulting in the risk of hardware damage, applications of RL in robotics have been lagging behind.

A promising approach to solve the problem of limited data availability is to use simulated environments for training the RL agents, and to later deploy them on real-world physical systems [3–5]. However, while this solves the problem of fast and safe data collection, inaccuracies in simulation caused by, e.g., uncertainty over physical parameters and unmodeled phenomena, may result in policies that do not transfer well to the real system. Such discrepancy is commonly known as the *reality gap*.

One can aim for more generalizable RL agents by performing the training over multiple variants of the environment, varying in appearance [5–8], dynamics parameters [4,9–13], or both [14,15]. This technique, known as *domain randomization* (DR), has proven to be successful in a range of robotics setups. However, finding well-performing domain randomization distributions over physical parameters can be difficult [16]—the distributions need to be wide enough to encompass the plausible true parameter values in the real world, or make up for phenomena which are specific to the real system but are not modeled in the simulator [17]. At the same time, the DR distribution needs to allow for convergence while training a single policy that performs well across all the randomized environments, especially in the zero-shot transfer scenario. Therefore, there has been an increasing research interest in developing approaches to automate DR [18–22].

In this paper, we introduce Domain Randomization Off-Policy Optimization (DROPO), a novel method for automatically inferring domain randomization distributions. DROPO fits a dynamics parameter distribution to an offline dataset using a likelihood-based metric, which explicitly maximizes the probability of observing real-world data in simulation. The simulator can thus be thought of as a stochastic forward model whose randomness arises from variability in the physical parameters of the scene. Our approach, outlined in Fig. 1, departs the trend of online-adaptive DR—where data from the target domain is collected on-policy during the parameter optimization process, requiring iterative access to the real-world setup [15,18,21,23]—and follows the recently introduced offline setting [20], while using a probabilistic metric to encourage variance in the converged parameter distributions. This allows DROPO to optimize DR distributions using only precollected data (e.g. human demonstrations) without suffering from the distributions collapsing to point estimates due to L2-based loss functions. In contrast to previous claims [19], our findings support that a likelihood-based method can be effortlessly implemented with a non-differentiable black-box simulator and gradient-free optimization techniques.

The contributions of this work are: (1) introducing DROPO, a novel method for offline optimization of domain randomization

* Corresponding author.
E-mail address: karol.arndt@aalto.fi (K. Arndt).

https://doi.org/10.1016/j.robot.2023.104432
0921-8890/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

distributions given safely collected data, (2) an experimental evaluation of DROPO to show convergence to ground truth dynamics distributions, (3) demonstrating that domain randomization distributions obtained with DROPO can be used to successfully train control policies in the presence of unmodeled phenomena, (4) showing that RL policies can be directly transferred to the real world when trained in simulation on DROPO's dynamics distributions. The source code for running DROPO with arbitrary offline datasets is publicly available.[1]

2. Related work

Domain randomization as a method for training generalizable reinforcement learning policies has been widely studied over the past few years, both in the context of computer vision [5–8], dynamics models [4,9–13] and both at the same time [14,15]. A major challenge, however, is that domain randomization requires the randomization distributions for each parameter to be specified when the model is trained. DR essentially trades optimality for robustness: using too wide ranges may have negative impact on the training, making it difficult for the agent to learn a single policy that generalizes to such a wide range of dynamics, while using too narrow ranges would bias the model towards a small set of dynamics values and thus possibly hinder generalization to the real world, as in the case of system identification of point-estimate dynamics [24,25].

To this end, multiple methods for automatically tuning the randomization ranges were proposed, such as minimizing discrepancy between simulated and real trajectories [21] or directly maximizing the expected real-world return [23]; however, as these methods explicitly ask for real-world feedback during the process, they require access to and careful supervision of the physical setup in order to run the policy trained on the intermediate DR distributions.

To relax these requirements, DROID [20] proposed a setting where the DR distribution is optimized using a fixed, offline dataset pre-collected in the target domain, aiming to improve data-efficiency and reduce safety risks coming from rolling out policies trained in unconverged dynamics in contact-rich tasks. This framework reduced the real-world interaction to a single data collection step performed before parameter optimization; additionally, this setting does not make any assumptions about the data collection procedure, allowing it to be collected with another policy, or through human demonstrations. Our method follows the same problem setting, but differs from DROID in that it uses a likelihood-based objective function and optimizes the distribution variance in addition to the mean. We provide a thorough comparison between our method and DROID in Section 4.

A variation of this offline setting, where the real-world data collection policy is assumed to be available, was addressed in BayesSim [19]. Under this assumption, BayesSim trains a neural network to predict dynamics parameter posteriors using trajectories collected in simulation by the given policy under a variety of dynamic conditions. Trajectories collected by the same policy on the real world setup are then passed to the trained network to predict the dynamics parameter distributions in the real world. Similar to our work and DROID, BayesSim does not require access to the real hardware during optimization; however, it still assumes the data collection policy to be available during optimization. In contrast, we propose a maximum likelihood-based framework which does not require the data collection policy to be known, making it more suitable for use with human demonstrations. We provide a detailed comparison between our method and BayesSim with human demonstrations in Section 4.

[1] https://github.com/gabrieletiboni/dropo.

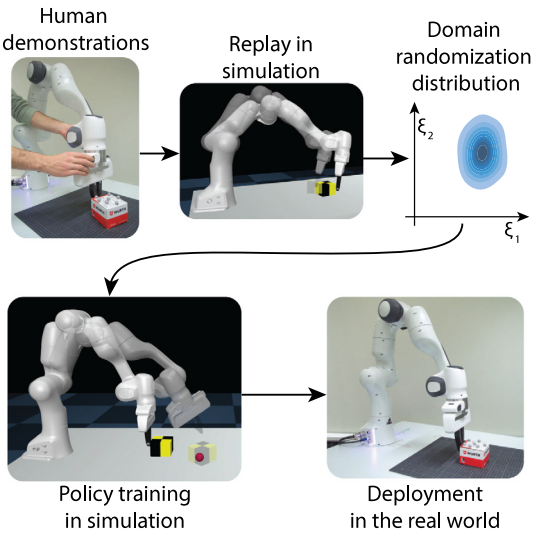

Fig. 1. DROPO uses off-policy data or human demonstrations to learn a domain randomization distribution, which is later used to train a policy that can be directly transferred to the real world.

Other works followed a different approach for applying domain randomization, by keeping a fixed range of the physical parameters and, instead, relying on intelligent sampling techniques to improve generalization: [22] guides training on increasingly harder environment variations, while [26] increases the number of sampled simulated environments until satisfactory transfer behavior is reached.

Finally, approaches based around differentiable physics simulators were also devised [27]. While DROPO could in principle be used with a differentiable simulator, allowing for the use of a gradient-based optimizer, we do not make any assumptions about the differentiability of the simulator in this work.

3. Method

3.1. Background and problem formulation

A Markov decision process (MDP) $\mathcal{M}$ is described by its state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution $p(s_0)$, state transition distribution $p(s_{t+1}|s_t, a_t)$ and the reward distribution $p(r_t|s_t, a_t, s_{t+1})$. Each *episode* begins with an *agent* starting in state $s_0 \sim p(s_0)$. At each step, the agent selects an action a following a policy $\pi(a|s)$. The problem addressed in reinforcement learning is then to find an optimal policy $\pi^*(a|s)$, such that the expected cumulative reward (the *return*) is maximized.

In the domain transfer scenario, we can view the real world as an MDP with state transition probabilities $p_{real}(s_{t+1}|s_t, a_t)$, with the simulation additionally parameterized by the dynamics parameter vector ξ: $p_{sim}(s^{sim}_{t+1}|s_t, a_t, \xi)$. We assume that both MDPs share the state and action spaces, the initial state distribution, and the reward function.

Under this formulation, the problem we address in this work can be stated as follows: given a dataset of real-world state–action trajectories $\mathcal{D} = \{s_0, a_0, s_1, \ldots, s_T\}$, find the distribution $p^*(\xi)$ such that the likelihood of transitioning to the real state s_{t+1} when taking action a_t while in s_t is maximized over the dataset:

$$p^*(\xi) = \arg\max_{p(\xi)} \mathop{\mathbb{E}}_{\substack{s_t, a_t, s_{t+1} \sim \mathcal{D} \\ \xi \sim p(\xi)}} p_{sim}(s_{t+1}|s_t, a_t, \xi). \quad (1)$$

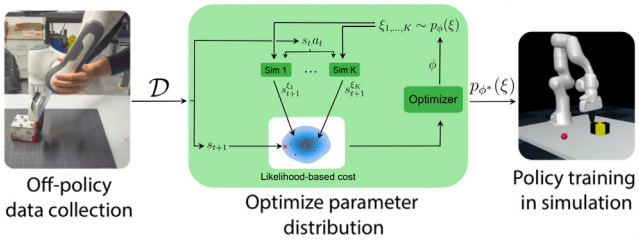

Fig. 2. Overview of the main stages of DROPO. The dynamics parameter distribution is optimized to maximize the probability of observing the collected dataset in simulation.

3.2. Method overview

DROPO consists of three main steps: dataset collection on the real hardware, dynamics distribution fitting, and policy training. An overview of the method is presented in Fig. 2 with a step-by-step walkthrough of dynamics distribution fitting presented in Algorithm 1.

Algorithm 1: DROPO

Result: Parameters ϕ^* of $p_{\phi^*}(\xi)$
1 Initialize ϕ to $(\mu_{init}, \Sigma_{init})$;
2 Initialize empty dataset $\mathcal{D}$;
3 Fill $\mathcal{D}$ with trajectories from the target domain ;
4 **while** *not converged* **do**
5 Sample a set Ξ of K random dynamics $\xi \sim p_\phi(\xi)$;
6 **forall** $s_t, a_{t..t+\lambda-1}, s_{t+\lambda} \in \mathcal{D}$ **do**
7 **forall** $\xi_k \in \Xi$ **do**
8 Set simulator parameters to ξ_k ;
9 Set the simulator state to s_t ;
10 Execute actions $a_{t..t+\lambda-1}$;
11 Observe $s^{\xi_k}_{t+\lambda} \sim p_{sim}(s^{sim}_{t+\lambda} | s_t, a_{t..t+\lambda-1}, \xi_k)$;
12 **end**
13 Compute the mean $\bar{s}^{\phi}_{t+\lambda}$ and covariance $\Sigma^{\phi}_{t+\lambda}$ (Eq. (2));
14 Evaluate the log-likelihood $\mathcal{L}_t$ of $s_{t+\lambda}$ under $\mathcal{N}(\bar{s}^{\phi}_{t+\lambda}, \Sigma^{\phi}_{t+\lambda})$ (Eq. (3));
15 **end**
16 Compute the total log-likelihood $\mathcal{L} = \sum_t \mathcal{L}_t$;
17 Update ϕ towards maximizing $\mathcal{L}$;
18 **end**
19 Train a policy with DR using the converged $p_{\phi^*}(\xi)$;

The data collection step is where the dataset $\mathcal{D}$, used for parameter optimization, is collected on the physical hardware. This can be done either by running any previously trained policy (e.g., a policy for another task) or by manually guiding the robot through kinesthetic teaching. Since DROPO uses offline, off-policy data, there is no need to collect additional data at later stages. This is also the only step where the physical hardware is necessary—all later steps of the method are performed in simulation, up until the final deployment.

The core part of DROPO is the second step, where the domain randomization distribution $p_\phi(\xi)$—parameterized by ϕ—is optimized to maximize the likelihood of real-world data. In particular, the likelihood is computed w.r.t. the *next-state distribution* under ϕ, which can be expressed as the marginal probability density $p_{sim}(s^{sim}_{t+1}|s_t, a_t, \phi) = \int p_{sim}(s^{sim}_{t+1}|s_t, a_t, \xi) p_\phi(\xi) d\xi$. To approximate this integral, the state–action pair (s_t, a_t) in question is executed multiple times through simulators with different dynamics parameters ξ_k sampled from $p_\phi(\xi)$ (steps 7–12 in Algorithm 1). By doing so, the resulting next-state distribution $p_{sim}(s^{sim}_{t+1}|s_t, a_t, \phi)$ may be estimated, which captures uncertainty induced by both the simulator dynamics and the given DR distribution. The parameters ϕ of $p_\phi(\xi)$ are then adjusted to maximize the likelihood of the real s_{t+1} under $p_{sim}(s^{sim}_{t+1}|s_t, a_t, \phi)$.

We can further generalize the algorithm by allowing to step the simulator by more than a single step during the inference process. This mechanism allows to pick a desired evaluation frequency—regardless of the sampling rate of collected data—and to increase the signal-to-noise ratio in the difference between successive states. This results in an additional hyperparameter $\lambda \in \mathbb{N}^+$, which controls how many successive actions are executed in the environment before evaluating the log-likelihood of the real $s_{t+\lambda}$ under $p_{sim}(s^{sim}_{t+\lambda}|s_t, a_{t..t+\lambda-1}, \phi)$. This generalization is reflected in step 10 of Algorithm 1.

Once the parameter optimization process has converged, the resulting distribution $p_{\phi^*}(\xi)$ is used to train a policy in the third step of DROPO. This policy can then be deployed on the setup that the data was collected from in the first step.

3.3. Data collection

The first step of DROPO is to collect the state–action trajectory dataset $\mathcal{D}$, which is going to be used for domain randomization parameter optimization. This dataset can either be collected from human demonstrations, or by running a previously trained, safe policy in the environment. Regardless of the method used, the data should be collected in such a way that allows for identification of the relevant dynamics parameters; for example, it will not be possible for DROPO to infer the friction coefficient of an object that does not move during demonstrations.

In case demonstrations are provided, the true actions are not directly known and need to be inferred based on the measured state sequence using an inverse dynamics model. While learning inverse dynamics may be a challenging problem in itself, in certain environments—such as position- or velocity-controlled robot arms—this inference can be simplified under the assumptions that the robot's on-board controllers are perfect. While this is never the case in real-world systems, we found that DROPO was able to handle this discrepancy.

Once collected, the dataset is preprocessed in order for the states to be directly replicable in simulation. First, data from different sensor modalities needs to be synchronized; this is because data from position measuring devices, such as motion capture systems, is likely to be delayed w.r.t. robot joint position measurements. Additionally, it is necessary to ensure that the data is sampled with the same frequency as the simulated environment timesteps. To achieve this, we fit an Akima spline to all sensor measurements and evaluate it at each environment timestep. We chose the Akima interpolation method over a cubic spline, as it prevents the values from overshooting in-between sampled points.

3.4. Distribution fitting

Once a preprocessed dataset is available, we move on to the core part of DROPO—estimating the dynamics probability distribution $p_\phi(\xi)$.

In our method, the optimized parameter vector ϕ includes both the means and variances of each univariate distribution, assuming uncorrelated parameters. Therefore, with a dynamics parameter vector $\xi \in \mathbb{R}^d$, the parameterization is represented by a $2d$-dimensional vector $\phi \in \mathbb{R}^{2d}$. This is in contrast to DROID [20], where only the means are optimized while the

standard deviations are taken from the CMA-ES optimization distribution instead. This change allows to explicitly learn the variability of each simulated physical parameter to best explain heteroschedasticity and uncertainty in real-world data. DROPO can thus be thought as a random coefficient statistical model, where the optimized coefficients are themselves assumed to be random variables, as opposed to the more standard fixed-effects regression analysis.

The distribution fitting process starts with an arbitrary initial guess $\phi_{init} = (\mu_{init}, \Sigma_{init})$ on the dynamics distribution, roughly informed by looking at the replayed trajectory in sim. The main inference part follows by sampling K dynamics parameters from $p_\phi(\xi)$. Then, for each parameter vector ξ_k, we set the simulator state to the original real state s_t, execute the real action sequence $a_{t..t+\lambda-1}$ and observe the next state $s^{\xi_k}_{t+\lambda}$. This process allows us to estimate the next-state distribution $p_{sim}(s^{sim}_{t+\lambda}|s_t, a_{t..t+\lambda-1}, \phi)$, modeled as a Gaussian distribution. In particular, we infer the corresponding mean and covariance matrix as follows:

$$\begin{aligned} \bar{s}^{\phi}_{t+\lambda} &= \frac{1}{K}\sum_k s^{\xi_k}_{t+\lambda} \\ \Sigma^{\phi}_{t+\lambda} &= \widehat{\mathrm{Cov}}(p_{sim}(s^{sim}_{t+\lambda}|s_t, a_{t..t+\lambda-1}, \phi)) + \mathrm{diag}(\epsilon), \end{aligned} \tag{2}$$

where $\widehat{\mathrm{Cov}}(\cdot)$ is the unbiased sample covariance matrix estimator. Here, ϵ is a hyperparameter used to compensate for observation noise and regularize the likelihood computation in case singular covariance matrices would appear—e.g. due to directions of variance in the state space unexplainable by the simulator dynamics. Tuning this hyperparameter will, effectively, adjust how much next-state variance is to be modeled by variance in dynamics parameters, relative to observation noise. As demonstrated in our experiments, setting this parameter to too low values may result in obtaining an overly wide randomization distribution: in the extreme case of $\epsilon = 0$, DROPO would attempt to model all phenomena that cannot be reproduced in a single simulation by variability in the dynamics, potentially overfitting to noise or unmodeled effects. However, increasing ϵ to excessively large values will eventually lead to over-regularization and point-estimate dynamics, as all unexplained variance would be modeled as observation noise, falling back to pure system identification. To ensure stability while promoting variance in the dynamics parameters, we propose to tune ϵ by increasing it until the Mean Squared Error (MSE) between real and sim trajectories no longer meaningfully decreases—similarly to the Elbow method popularly used in machine learning. We present a thorough analysis of the impact of the ϵ parameter in the Experiments section.

After observing the resulting simulator states, we make the assumption that the true, real-world observation $s_{t+\lambda}$ originates from the next-state distribution induced by dynamics randomization: $s_{t+\lambda} \sim \mathcal{N}(\bar{s}^{\phi}_{t+\lambda}, \Sigma^{\phi}_{t+\lambda})$

The log-likelihood of $s_{t+\lambda}$ under this distribution can be then calculated following the standard Gaussian formula (ignoring the constant term)

$$\mathcal{L}_t = \frac{1}{2}(\log\det \Sigma^{\phi}_{t+\lambda} + (\bar{s}^{\phi}_{t+\lambda} - s_{t+\lambda})^\top \Sigma^{\phi^{-1}}_{t+\lambda}(\bar{s}^{\phi}_{t+\lambda} - s_{t+\lambda})) \tag{3}$$

The log-likelihoods of each individual transition can then be summed to obtain the total log-likelihood of the dataset $\mathcal{D}$ under $p_\phi(\xi)$: $\mathcal{L} = \sum_t \mathcal{L}_t$

We finally adjust the parameters ϕ to maximize the total log-likelihood $\mathcal{L}$. As we do not make any assumptions about the differentiability of the simulator, we optimize $\mathcal{L}$ using a gradient-free optimization method. DROPO can, in principle, work with an arbitrary optimization method; we use CMA-ES [28] throughout the experiments, with initial scale of 1 and optimization parameters normalized linearly (means) or in log scale (variances) to the interval [0, 4]. Note that, in order to further improve stability, we model $p_\phi(\xi)$ as an uncorrelated truncated normal distribution bounded to values within two standard deviations away from the mean, with consecutive resampling when unfeasible parameters are drawn (e.g. negative masses).

3.5. Policy training

The policy $\pi(a|s)$ is trained for the given target task entirely in simulation using the converged domain randomization distribution $p_\phi(\xi)$. This is achieved by sampling new dynamics parameters $\xi \sim p_\phi(\xi)$ at the start of each training episode. The RL policy can then be trained with an arbitrary reinforcement learning algorithm; we use Proximal Policy Optimization (PPO; [29]) and Soft Actor–Critic (SAC; [30]) in our experiments. The trajectories coming from the offline dataset are not used in any way during the policy training process; as such, the demonstrations are not required to be specific to the particular task, as long as they allow the relevant dynamics parameters to be identified.

4. Experiments

The experimental evaluation aims to answer the following questions:

1. is DROPO capable of recovering the original dynamics parameters and randomization distributions in simulated environments?;
2. is DROPO capable of training a policy that can solve the task and transfer when unmodeled phenomena are present in simulation?;
3. how does the value of the ϵ hyperparameter affect the obtained domain randomization distribution?;
4. what is the real-world performance of policies trained with DROPO compared to DROID [20], BayesSim [19], and uniform domain randomization (UDR)?

4.1. Benchmark methods

Throughout our experiments we benchmark UDR similarly to [23], as we assess the average performance of several policies trained on different uniform bounds. More precisely, we train 10 policies with DR on 10 uniform bounds respectively, which are randomly sampled from the search spaces reported in the corresponding tables. This would reflect how manually picking uniform distributions within a reasonable search space affects the final performance.

Furthermore, in contrast to the original paper on DROID [20], we replaced torque measurements with the raw state vectors when implementing the DROID baseline. This modification makes more sense for the pushing and sliding tasks that we use for evaluation, as joint torques do not express enough information about the object being manipulated by the robot, especially for the sliding task—in contrast to the door opening task used in the original DROID paper. The CMA-ES implementation used in DROPO and DROID is provided by Nevergrad [31].

Finally, in order to adapt BayesSim to the offline setting with human demonstrations (where the data collection policy is not available), we repeat the original action sequence during data collection. Besides this modification, we make use of the original open-source repository provided by [19]. In most experiments, we evaluate both neural network (MDNN) and quasi-random Fourier (MDRFF) features and report the better result for brevity. An exception to this is the sim-to-real experiment with the Panda robot, where results with both features are reported in order to compare the generalization capability of both variants. Since

BayesSim uses mixture Gaussian distributions, the distribution inference results reported in the tables and plots in this section are independent Gaussian approximations of the complete results, for the sake of brevity. The complete BayesSim posteriors can be found in Appendix D.2.

4.2. Hopper

In the Hopper environment [32], a one-legged robot is tasked to learn to jump forward as fast as possible without falling down. This environment has been used as a standard control benchmark in a variety of papers [18,29,30,33]. We introduce parameter randomization to this environment by varying the mass of each link. We use this environment as a simple benchmark to test DROPO's capability of identifying the dynamic parameters of the system and recovering the parameter distribution that was used to generate the dataset. We later introduce an unmodeled effect in the form of a misspecified mass (which is not included in the optimization problem), similarly to [18]. Given the simple sim-to-sim scenario, the offline trajectories for running DROPO have been collected by rolling out semi-converged policies trained directly in the target domain.

4.2.1. Point-dynamics system identification

When the offline dataset is collected on a single Hopper environment instance, DROPO is expected to identify the target physical parameters and to converge to point-estimate dynamics rather than distributions. We tested this behavior on the Hopper environment by collecting two trajectories (1000 state transitions corresponding to 8 s of wall time) on the ground truth dynamics shown in Table 1, both with and without noise of variance 10^{-5} injected in the offline observations. The results are reported in the same table, as DROPO achieved convergence to the original dynamics regardless of the presence of noise and effectively decreased the standard deviation of learned parameters to 10^{-5}, set as the lowest bound during the optimization problem.

In the point-estimate dynamics estimation, DROID also converged to the ground-truth parameter values, with a lower standard deviation than DROPO (as CMA-ES does not impose any lower bounds on the variance). Thus, both DROPO and DROID perform virtually the same in this benchmark.

On the other hand, BayesSim overestimates the distribution variance and does not accurately recover the mean parameter value. This is in line with the results reported in the original paper [19], where the resulting posterior distribution is much wider than the ground-truth values used for data generation. While wider distributions may benefit generalization, BayesSim falls short of DROID and DROPO in terms of pure system identification capabilities.

4.2.2. Recovering the dynamics distribution

In this experiment, the dataset $\mathcal{D}$ was collected while varying the target dynamics according to some ground truth distribution $p(\xi)$, which we want to recover. We collected two trajectories on the Hopper environment, varying all masses as $m_i \sim \mathcal{N}(\mu_i, 0.25^2)$ around their original values μ_i every 25 state transitions, except for the torso mass which was kept fixed. Since BayesSim expects the input to be trajectories collected from a single dynamics setting rather than a set of transitions, we had to use a different test procedure for this baseline; namely, we sample multiple dynamics conditions from the true distribution and run BayesSim individually for each of them. The resulting distribution is then obtained by marginalizing the output distribution over the input dynamics distribution.

The results are reported in Fig. 3, which shows DROPO successfully learning the original variance of the randomized masses, in addition to their means. We believe that such explainability is only possible when explicitly optimizing parameters through probabilistic distance metrics, as DROID's L2-based cost function fails at providing wide distribution ranges on the randomized masses. Note how BayesSim, while preserving some variance in the converged distributions, resulted in much less accurate dynamics distribution estimates. Analogously to the previous experiment, a noisy version of the dataset has also been tested, yielding similar results.

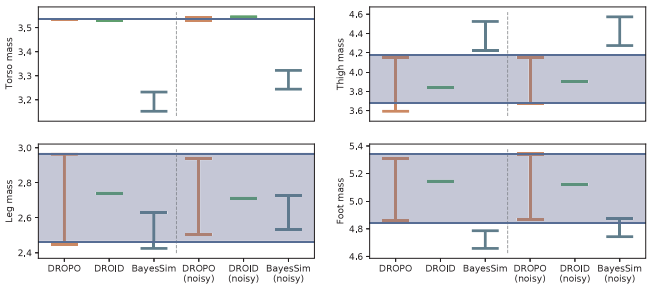

Fig. 3. Dynamics distributions optimized by DROPO, DROID and BayesSim on offline data collected from a distribution of Hopper environments (indicated by the shaded areas). The distributions are displayed with 2σ-wide ranges centered on their means.

4.2.3. Unmodeled phenomena

Finally, we also tested DROPO's ability to compensate for unmodeled phenomena in simulation by assessing the complete transfer to a target simulator from an under-modeled source domain. We introduced such unmodeled effect by misspecifying the Hopper torso mass by 1 kg in the source simulator and excluding it from the optimization space. Hence, each method may only optimize the means and variances of the remaining three masses to match the target transitions, and cannot adjust the incorrect torso mass.

To perform this evaluation, we first ran each method to infer the dynamics distribution given the same noiseless data as in Section 4.2.1. Since there are no ground-truth domain parameter values that could be used as a reference when unmodeled phenomena are present, the domain randomization distributions cannot directly be used to quantitatively compare the estimated dynamics parameters. Instead, we used the obtained distributions for policy training in the environment with a misspecified mass, and evaluated the resulting policies in the original environment where data was collected. We repeated this process 3 times for each method, with different random seeds. The policies were trained with SAC [30].

The results of this evaluation are presented in Fig. 4, together with a policy trained directly in the target environment to provide a notion of the upper bound. We observe that policies trained with DROPO's randomization distributions outperform all baseline methods, almost matching the policy trained in the ground-truth environment in terms of performance. At the same time, the policies obtained with DROID and BayesSim perform much worse. Note how some UDR policies are at times able to generalize well to the target environment, suggesting that the task could be solved by manual trial and error with different uniform bounds.

To provide more insight into these results, we can look at the converged domain randomization distribution reported in Table 1. Note how, even though the target dataset has been collected on point-dynamics parameters, DROPO converges to a distribution over the three randomized masses. This behavior is indeed expected, as due to the unmodeled phenomenon introduced, a single point-estimate parameter vector is not able to

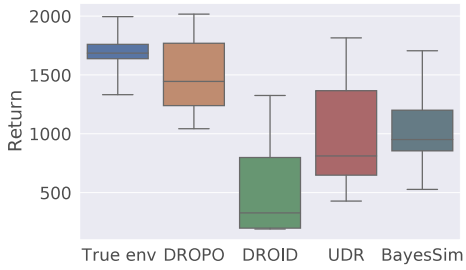

Fig. 4. Sim-to-sim: episode returns in the target Hopper environment, comparing policies trained directly in the target environment, and trained in the environment with a misspecified torso mass using domain randomization distributions inferred by DROID, DROPO and BayesSim.

accurately reproduce all transitions in the target dataset. Therefore, DROPO widens the dynamics distribution to maximize the likelihood of target-domain data to be observed in simulation. In addition, the optimized means are now much different from the ground truth parameters. On the other hand, the results show that—just like in the distribution recovery experiments—DROID is eager to converge to point-estimate parameters, in order to minimize the L2-distance based objective function (see Appendix A for an intuitive explanation of this phenomenon). Such a policy performs well in the training environment, but fails to generalize to the test conditions.

Furthermore, notice how BayesSim produced distributions with means located close the center of the search space with large standard deviations. As such, it was able to learn a successful policy that works over a range of dynamics parameters, including the environment where the unmodeled phenomenon is present. However, excessively large standard deviations result in overly conservative policies that produce a slower forward motion than DROPO, resulting in a lower return. We suspect that this might be caused by the domain shift between the training data collected from the simulation with misspecified mass and the data from the real environment used to infer the randomization distribution. In BayesSim, the inference network is trained on trajectories obtained from the simulator, each collected in different dynamics conditions. As a result, any trajectory that cannot be reproduced by single dynamics conditions in simulation will effectively lie outside of the inference network's training domain. Thus, we claim that any task involving unmodeled phenomena or unmodeled noise is substantially difficult for the BayesSim neural network, as the evaluation domain does not match the training domain anymore. Later, we obtained similar results in the Panda sim-to-real experiments, as described in Section 4.3.2.

To summarize, we observe that a combination of both system identification and domain randomization is essential to solve the task under this unmodeled effect, as neither UDR—pure unoptimized domain randomization—nor DROID—which converged to a point-estimate—result in a successful zero-shot transfer.

4.2.4. The domain variance–observation noise tradeoff

Throughout our experiments, we observed that introducing the ϵ hyperparameter is generally beneficial to obtain more stable and sound results (see Section 3). This turned out to be particularly effective when optimizing the estimated likelihood function in presence of noise or unmodeled effects. However, overly large values of ϵ should be avoided to prevent DROPO from converging to point-estimate dynamics parameters. To study this trade-off, we report a thorough sensitivity analysis on this hyperparameter, showing its impact on the final performance of DROPO for the previously described Hopper experiments and describing our tuning procedure in more detail.

Table 1
Optimized dynamics distributions on Hopper OpenAI Gym environment. The ground truth values indicate the simulator instance used during data collection.

	Masses (kg)		m_1	m_2	m_3	m_4
	Ground truth		3.534	3.927	2.714	5.089
	Search space	min	1.767	1.963	1.357	2.545
		max	7.069	7.854	5.429	10.18
Noiseless	DROPO ϵ=1e−8	μ^*	3.534	3.927	2.714	5.089
		σ^*	1.0e−5	1.1e−5	1.1e−5	1.0e−5
	DROID	μ^*	3.534	3.927	2.714	5.089
		σ^*	1.2e−12	2.1e−12	2.0e−12	1.1e−12
	BayesSim	μ^*	3.350	5.342	3.994	5.560
		σ^*	0.697	0.707	0.686	0.541
Noisy	DROPO ϵ=1e−5	μ^*	3.531	3.931	2.717	5.093
		σ^*	8.1e−3	1.3e−3	1.3e−3	2.4e−4
	DROID	μ^*	3.534	3.927	2.714	5.089
		σ^*	1.2e−12	2.4e−11	2.0e−12	1.2e−12
	BayesSim	μ^*	6.295	4.002	3.486	3.771
		σ^*	0.218	0.257	0.245	0.256
Unmodeled	DROPO ϵ=1e−3	μ^*	2.534	4.707	3.181	4.385
		σ^*	(fixed)	0.667	0.206	0.826
	DROID	μ^*	2.534	4.021	3.591	4.845
		σ^*	(fixed)	6.0e−6	1.7e−6	4.2e−6
	BayesSim	μ^*	2.534	3.493	2.901	4.596
		σ^*	(fixed)	1.576	1.471	1.537

The results of point-dynamics recovery in Hopper with varying ϵ are presented in Fig. 5. We observe that, for a low value of ϵ, DROPO overestimates the standard deviation in the noisy setting, as it attempts to explain the random variation in next-states by variance in dynamics. However, for ϵ values equal to the noise variance (or higher), DROPO converges to very low standard deviations, corresponding to a point-estimate of dynamics parameters and leading to a significant drop in MSE. The MSE is computed w.r.t. the converged distribution ϕ^* obtained by DROPO, as follows: $MSE(\phi^*) = \sum_t \|(\bar{s}_{t+1}^{\phi^*} - s_{t+1})\|^2$. This metric resembles the trajectory discrepancy function adopted by previous works [20,21], and we propose to use it as a reference to reach a stable optimization process and account for observation noise when tuning ϵ. Note how, in the same figure, the MSE drop is not as clear for the noiseless setting: here, ϵ only serves the purpose to regularize the log-likelihood computation, which could be ill-conditioned with too low values (e.g. $\epsilon = 10^{-10}$ in Fig. 5(b)) as close-to-singular covariance matrices may appear when estimating the next-state distribution.[2]

We observe a similar phenomenon in the dynamics distribution recovery scenario (Fig. 6). Small ϵ values result in an excessively large variance in masses in presence of noise, while correctly identifying the ground-truth distribution in the noiseless case. As ϵ grows larger, more of the variety in next-states gets attributed to noise and the estimated dynamics distributions move towards point dynamics, both in the noisy and noiseless case. The best ground-truth distribution recovery is observed when ϵ matches the true noise variance, which also corresponds to the sharp drop in MSE observed in Fig. 6(b)—thus validating the proposed tuning process.

Finally, we report similar observations for the unmodeled phenomenon experiment (Fig. 7), where—as ϵ increases—unexpl-

[2] Adding a small value ϵ to the diagonal of an ill-conditioned covariance matrix has the effect to increase all eigenvalues by ϵ, thus making the likelihood computation more stable.

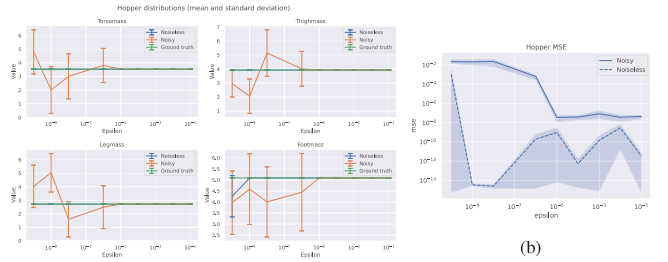

Fig. 5. Epsilon sweep on point-estimate dynamics recovery: (a) shows the converged distributions in the noiseless and noisy ($\sigma^2 = 10^{-5}$) cases, (b) shows how the MSE changes with ϵ (results averaged over 4 random seeds).

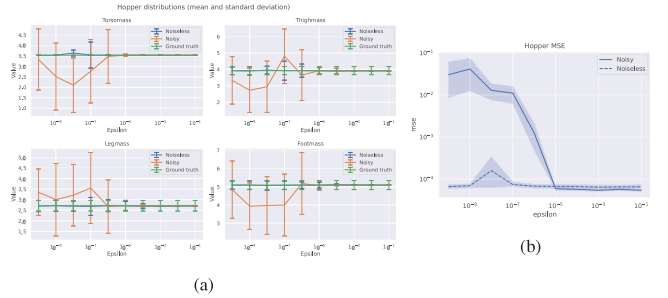

Fig. 6. Epsilon sweep on dynamics distribution recovery: (a) shows the converged distributions in the noiseless and noisy ($\sigma^2 = 10^{-5}$) cases, (b) shows how the MSE changes with ϵ (results averaged over 4 random seeds).

ainable effects in state space due to under-modeled dynamics get attributed to noise, stabilizing the optimization problem. Once again, we tuned ϵ based on the MSE observed in Fig. 7(b), and picked $\epsilon = 10^{-3}$, which is where the MSE values stop to meaningfully improve. As observed in Fig. 7(a), this value closely approximates the mean obtained with higher ϵ values, while preventing the distribution from collapsing due to excessive regularization. To this end, we would like to emphasize that a conservative choice may always be made with DROPO when tuning the value of ϵ: higher values will only shift DROPO towards lower standard deviations (eventually providing similar results to DROID), while allowing a desirably low MSE to be obtained.

Note how the proposed tuning procedure draws similarities to the elbow method popularly used in machine learning, e.g. to find the best number of clusters in clustering analysis or the number of principle components to retain in principal component analysis.

4.3. Robotics experiments

We further evaluate DROPO on two sim-to-real robotics setups: a hockey puck sliding and a box pushing task.

4.3.1. HockeyPuck

In the HockeyPuck setup, the Kuka LWR4+ is equipped with a hockey stick and tasked with hitting a hockey puck such that it stops at the given target location. For this setup, we use an ice hockey puck and a whiteboard as a low-friction surface for the puck to slide on. The simulation environment is built in MuJoCo [34] akin to the real-world setup. Both setups are shown in Fig. 9. The yellow puck and the purple area in Fig. 9(a) show the initial position of the puck and the range of possible goal positions.

In this setup, the actions are whole hitting trajectories; this is achieved by first training a variational autoencoder (VAE) on

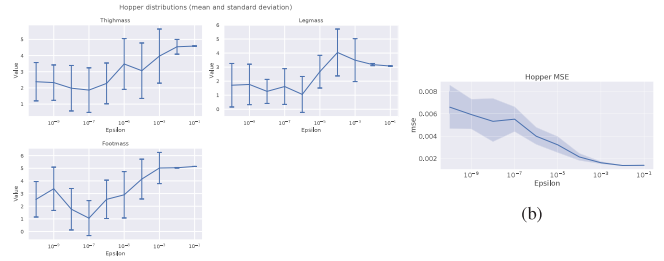

Fig. 7. Epsilon sweep on the unmodeled phenomenon experiment: (a) shows the converged distributions, (b) shows how the MSE changes with ϵ (means and variances averaged over 8 seeds).

a range of task-specific trajectories, and later using the decoder as a trajectory generator, with actions given in the latent space of the VAE, following [6,35]. This simplifies the reinforcement learning problem by turning it into a contextual multi-armed bandit problem.

The demonstrations for optimizing the parameter distribution are obtained by rolling out 5 random trajectories of 750 transitions (or 7.5 s) each, sampled from the latent space of the VAE ($z \sim \mathcal{N}(0, I)$). The joint positions are obtained using the robot's internal sensors, while the positions of the hockey puck are obtained with OptiTrack, a motion capture system, with the frequency of 120 Hz. During the post-processing, the data is synchronized, the positions are resampled to match the simulated timesteps of the simulated environment, and the velocities are obtained by taking the derivative of the spline used to resample the signal.

The state space used for dynamics fitting with DROPO contains the positions and velocities of the robot joints and the hockey puck. The use of a trajectory generator allows us to use original commanded velocities as actions when replaying trajectories in the simulator, and—on this particular setup—removes the need of action inference from demonstrations.

We randomize a total of 18 dynamics parameters: the mass of the puck, the puck-surface friction coefficients, the time constant of the stick-puck contact—regulating the stiffness and damping while keeping the interaction critically damped—and the gains of the position controller used in simulation. Note that, while more parameters may be added to the optimization problem to increase the model's explanatory power, complex correlations between the parameters may arise in addition to a generally higher complexity.

We use the reward function of [36]: $r = -d^2 - log(d + \alpha)$, where d is the distance to the target and α is a constant (we use $\alpha = 10^{-3}$ throughout the experiments).

The results comparing DROPO, DROID and UDR are shown in Fig. 10. We see that, on this setup, both DROPO and DROID significantly outperformed UDR, and that there was no significant difference between the two methods, despite DROID essentially converging to point estimates (see Table 2). This is likely caused by the bandit formulation and the resulting feed-forward nature of the task; once the action is selected, the agent does not receive any feedback from the environment. Under this setup, it is thus optimal for the policy to select the action (or the trajectory) which minimizes the expected final distance from the target position under the dynamics distribution used in training. This closely resembles the optimization criterion used by DROID (minimizing the expected L2 norm between trajectories). Additionally, under this formulation the policy lacks the ability to make up for possible deviations from the trajectory obtained in point-estimate dynamics. In a sequential decision making problem with

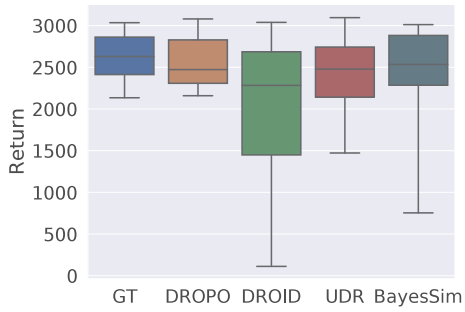

Fig. 8. Sim-to-sim: episode returns in the target simulated environment when trained on domain randomization distributions inferred by DROID, DROPO and BayesSim and on the ground-truth dynamics distribution (GT) in the Panda pushing task.

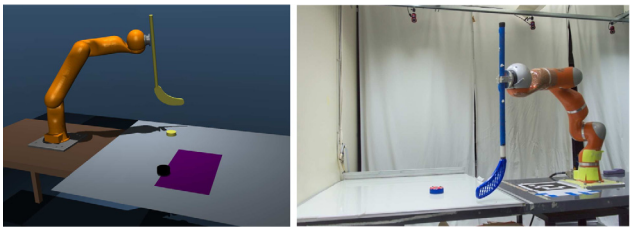

Fig. 9. The hockeypuck setup in simulation (a) and in the real-world (b). The purple area in (a) shows the range of possible goal positions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

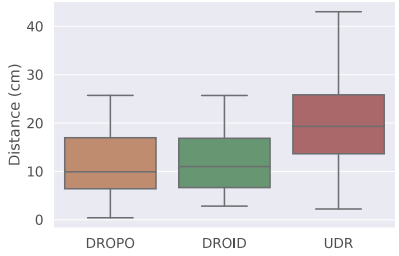

Fig. 10. Sim-to-real performance in terms of the final distance to the target position on the hockeypuck setup (in centimeters; lower is better).

uncertain dynamics, the agent needs to be able to recover from a wider range of states, due to the uncertainty in state transitions that stems from the uncertainty in dynamics. In a bandit problem, however, the system operates in a feedforward manner, and thus is not able to handle such situations at all. As a result, the benefits of having a distribution of dynamics are diminished.

4.3.2. Pushing

We finally evaluated DROPO on a pushing task with the Franka Panda robot. In this setup, shown in Fig. 11, the goal of the robot was to push a box to a fixed target position. To make this task more challenging, the box center of mass was altered by filling it up with heavy steel bolts on one side and bubble wrap on the other, as shown in Fig. 11(c). Therefore, we randomize a total of 5 dynamics parameters to shorten the reality gap in this setup: mass of the box, box-table friction coefficients f_x and f_y, and center of mass offset from the geometric center (com_x, com_y).

Fig. 11. The push setup in simulation (a) and in the real world (b), with the insides of the box shown in (c).

Table 2
Optimized dynamics distributions on 5 trajectories collected on the Hockeypuck environment. The converged mass m, friction coefficients f_x and f_y along the two axes, and timeconst parameter t_{const} are reported.

		m	f_x	f_y	t_{const}
Search space	min	0.08	0.2	0.2	0.001
	max	0.2	0.75	0.75	0.02
DROPO ϵ=1e−2	μ^*	0.111	0.349	0.231	0.019
	σ^*	2.2e−04	3.4e−05	4.7e−04	6.7e−05
DROID	μ^*	0.100	0.334	0.226	0.019
	σ^*	1.5e−07	8.9e−06	9.7e−06	1.7e−05

Due to the offset center of mass, we additionally included the orientations and angular velocity of the box in the state space used for fitting DROPO.

In order to model log-likelihood of orientations with DROPO, we used quaternion representations and evaluated the angle between the quaternion in the real-world dataset and the one in simulation; this angle, α, then replaced the orientation inside the next state s^{ξ}_{t+1}, with α inside the real-world state vector set to 0. Since such a distribution would not be well-modeled with a Gaussian, for each transition we insert two resulting next states (with α and $-\alpha$), resulting in a symmetric distribution of angles centered around 0.

We first tested our method in a sim-to-sim version of the task: synthetic data was collected by a forward-pushing policy as demonstrations for DROPO, DROID and BayesSim, sampling new dynamics parameters from the ground-truth distribution after every rollout, for a total of 50 trajectories of 30 transitions each, amounting to 30 s of wall-time interaction. The resulting DR distributions are presented in Table 3. Like before, the reported BayesSim results are independent Gaussian approximations of the actual results, which can be found in Appendix D.2. We again observe that DROID converged to a point estimate, with standard deviations in the order of 10^{-10}–10^{-11}, while DROPO results in a distribution that is relatively close to the ground truth, albeit not the same. The largest discrepancy can be observed in the inferred friction value along the x direction. Since the trajectory used for this demonstration was pushing the box primarily away from the robot (along the y axis), accurately inferring friction along the lateral axis is challenging, and the variations in box motion resulting from variations in this friction are minor and can be reasonably well modeled by variation in other parameters. Similarly, the mass of the box can also be relatively difficult to identify, given that the inference is performed with a position controller and the mass of the box is much smaller than the mass of the robot.

We then used these DR distributions to train a policy for pushing the box to a goal on the front-left side of the robot. We used the same reward function as for the Hockey experiments, with additional control penalties added to regularize the motions and to deter the simulated robot from going over the velocity and acceleration limits of the physical robot arm. The performance

of each policy is shown in Fig. 8. We observe that, despite the converged distribution being slightly different than the ground truth, the policies trained with DROPO performed close to the ground truth policies, while DROID policies performed noticeably worse, likely due to overfitting to point-estimate dynamics. Interestingly, the UDR baseline outperformed DROID too. This suggests that, for the underlying task, DR is a more effective transfer technique than system identification even when the distribution is not carefully tuned; in other words, wider but less accurate dynamics parameter distributions provide better results than converged point estimate parameter values.

We then evaluated DROPO's ability to transfer to the real-world setup, in comparison to DROID, BayesSim, and UDR. A single task-agnostic trajectory of about 13 s was collected through kinesthetic guidance and preprocessed in a similar way to hockey puck—we used robot's joint encoders to get the joint positions and OptiTrack to track the object.

The converged DR distributions obtained from this data by all benchmarked methods are presented in Table 4, averaged over three seeds. Once again, we point out the smaller standard deviations DROID converged to, compared to DROPO. However, as the real-world dynamics parameters are not known, the final evaluation has to be carried out by means of policy performance on the real system, displayed in Fig. 12 (5 rollouts per policy, per seed).

In this task, DROPO consistently outperformed previous methods and pushed the box as close as 2 cm away from the target location, showing another successful zero-shot transfer. We observe how BayesSim, which resulted in less accurate parameter inference across our experiments, achieved inconsistent performance and obtains similar results to DROID and UDR on average. This result further motivated the need to infer dynamics such as to account for both accurate system identification and domain randomization at the same time, even for a simple pushing task. BayesSim with neural network features (MDNN) predicted the box center of mass to be over 20 cm away from the center in each direction, with a standard deviation of over 50 cm, which results in the center of mass being located way outside of the object. Like in the Hopper experiment with an unmodeled phenomenon, this likely originates from the rather poor generalization to real-world data, which lies outside of the BayesSim network's training distribution—consisting of simulated data only. However, since the environment implementation clips the center of mass values to be within object boundaries, the agent was still able to learn a somewhat sensible policy for some random seeds. BayesSim with random Fourier features (MDRFF) shows slightly better generalization to real world data, producing a more sensible center of mass value, albeit with an excessive standard deviation of over 7 cm in both directions and (as visualized in the detailed results in Appendix D.2) with negative mass being predicted on some occasions. In terms of task performance, the trained policy still performs worse on average than DROPO or DROID.

5. Conclusions

In this paper we introduced DROPO, a method for crossing the reality gap by optimizing domain randomization distributions with limited, offline data. We demonstrated that, unlike previous methods, DROPO is capable of accurately recovering the dynamics parameter distributions used to generate a dataset in simulation. We also showed how DROPO may compensate for a misidentified value of a physical parameter, leading to well-performing reinforcement learning policies in the target domain.

We then moved on to real-world robotics setups—a hockey puck sliding and a box pushing tasks. We demonstrated that a policy trained in simulation with DROPO can be directly transferred to the real world on both setups, while previous methods fail on the box pushing task. We conclude that our method is able to effectively perform domain randomization and system identification at the same time, finding the right balance of parameter uncertainty by adjusting ϵ.

In the current work, we used a variety of data collection strategies to collect the offline dataset—a partially-trained task policy (*Hopper*), random sliding trajectories (*Hockeypuck*) and human demonstrations obtained through kinesthetic teaching (*Pushing*). Future works may investigate how the data collection strategy impacts the accuracy of parameters obtained by offline methods such as DROPO and DROID; in particular, prior work on exploration in meta-learning would make an interesting extension [37,38].

Under the current formulation, DROPO is limited to relatively low-dimensional parametric distributions. This limitation stems from the rather costly non-differentiable optimization procedure; using less restrictive sampling procedures, such as Stein variational gradient descent used in [22], could be a promising way

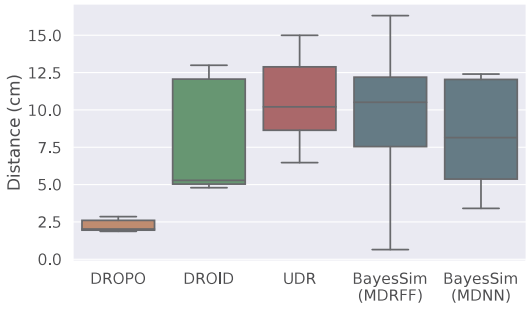

Fig. 12. Sim-to-real performance in terms of the final distance to the target position on the pushing setup (in centimeters; lower is better).

Table 3
Results of the sim-to-sim pushing task experiment with randomized box mass m, table-box friction coefficients (f_x, f_y) and box center of mass in meters (com_x, com_y).

		m	f_x	f_y	com_x	com_y
Search space	min	0.08	0.20	0.20	−0.032	−0.032
	max	2.00	2.00	2.00	0.032	0.032
Ground truth	μ^*	0.7	0.8	0.8	0.01	0.01
	σ^*	0.3	0.2	0.2	0.001	0.001
DROPO $\epsilon=0$	μ^*	0.788	0.603	0.683	0.015	0.009
	σ^*	0.108	0.033	0.18	0.001	0.003
DROID	μ^*	0.774	0.737	0.716	0.014	0.011
	σ^*	3.1e−11	1.3e−10	1.2e−11	7.6e−11	2.4e−11
BayesSim	μ^*	0.703	0.894	0.840	−0.011	0.004
	σ^*	0.269	0.224	0.257	0.309	0.234

Table 4
Optimized dynamics distributions on a single offline trajectory collected on the real pushing environment. The converged box mass m, friction coefficients f_x and f_y, and box center of mass com_x and com_y (in meters) are reported.

		m	f_x	f_y	com_x	com_y
Search space	min	0.08	0.20	0.20	−0.032	−0.032
	max	2.00	2.00	2.00	0.032	0.032
DROPO $\epsilon=1e-4$	μ^*	1.065	0.387	0.921	0.012	−0.032
	σ^*	1.3e−04	3.0e−03	5.3e−04	5.2e−05	7.5e−04
DROID	μ^*	0.826	0.444	0.665	−0.028	−0.028
	σ^*	1.4e−07	2.3e−08	2.6e−08	7.7e−08	6.6e−08
BayesSim MDNN model	μ^*	1.548	3.889	0.964	−0.322	−0.240
	σ^*	2.017	0.805	0.953	0.577	0.632
BayesSim MDRFF model	μ^*	0.796	1.181	1.013	−0.023	−0.040
	σ^*	0.252	0.218	0.316	0.090	0.075

of further improving the accuracy. We showed that DROPO is capable of performing optimization in up to 18-dimensional parameter spaces; while this was enough for simple manipulation tasks, complex scenarios involving interactions between multiple objects may involve many more parameters, which could make DROPO prohibitively expensive. Furthermore, in addition to larger dynamics parameter spaces, we encourage future works to extend our method for likelihood computation in high-dimensional state spaces, e.g. allowing DROPO to work on vision-based policies and plain images collected offline. In this case, the general idea behind DROPO could be used not only to optimize the dynamics parameters of the system, but also to adjust visual domain randomization distributions for image rendering, for instance by adjusting parameters used for procedural generation of textures.

Domain randomization methods require the user not only to specify the dynamics parameter distributions, but also to select what parameters need to be randomized. In our experiments, those parameters were indeed selected manually based on domain knowledge. While this approach works well with simple and well-understood environments, it can present a challenge in situations where expert knowledge of the application domain is limited. Even though randomizing as much as possible may appear to be a good idea on paper, it often leads to a significant increase in the computation cost. Additionally, some parameter combinations may be physically impossible, leading to simulator instabilities.

Aside from that, sim-to-real transfer with domain randomization often requires a significant development effort, requiring an accurate physics engine, object meshes and parametric simulation models to be available. As of now, there is no automated way of designing simulation models that accurately mimic real-world setups; such methods would make the initial software development effort much lower, allowing sim-to-real transfer with domain randomization to be used more widely in the industry, for example in digital twin applications. Recent developments in automated design open up the possibility of automating the construction of simulation models beyond parametric inference. More specifically, the object shapes and the general kinematic structure of the scene could be inferred based on the appearance of the real world system, and have its dynamics parameters tuned based on pre-collected physical observations using a method like DROPO.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Academy of Finland grants 317020 and 328399. We acknowledge the computational resources generously provided by CSC – IT Center for Science, Finland, and by the Aalto Science-IT project.

Appendix A. Least squares for offline DR optimization

In this appendix section we attempt to give a more thorough explanation to why DROID converged to point-estimate dynamics parameters across all our experiments. In particular, we aim to provide an intuition for why we discourage the use of least squares-like methods for trajectory alignment during *offline* optimization of DR distributions. This discussion is further motivated by the results reported in [39], where SimOpt's [21] objective function was used in the offline setting by replaying actions from off-policy data in simulation. Similarly to our results with DROID, this approach resulted in point-estimate dynamics estimates, leading the authors to consider offline SimOpt as a point-estimate system identification method.

Based on the notation used throughout our work, let us consider the optimization problem aiming to minimize the following objective function:

$$J(\xi) = \frac{1}{K}\sum_{i=1}^{K} \|f(s_i, a_i; \xi) - s_{i+1}\|^2 \tag{A.1}$$

with $f(\cdot;\xi)$ being the simulator dynamics—assumed to be deterministic for simplicity—parameterized by parameters ξ.

Note that, since the objective function is defined for a given dynamics parameter vector ξ, minimizing it by acting on the full distribution is not straightforward. For example, DROID [20] uses CMA to minimize the squared residuals and then retains the final inner CMA covariance matrix to ultimately build the DR distribution. Similarly, SimOpt [21] uses a sampling-based gradient-free algorithm based on relative entropy policy search [40] allowing to minimize residuals using only samples $\xi \sim p_\phi(\xi)$.

Regardless of the specific method, since the final goal is the minimization of $J(\xi)$, we can reason about its expected value w.r.t. $p_\phi(\xi)$, which we are ultimately interested in minimizing. Intuitively, when optimizing the objective in (A.1) under no additional assumptions or constraints, it is indeed reasonable to believe that no benefit is gained by ending up with a higher expected value of $J(\xi)$ in exchange for a certain desired variance.

We then claim that optimization problems aiming to optimize the expected value of quantity in (A.1) by acting on the random component itself, inevitably converge to point-estimates. Interestingly enough, this draws similarities to the weighted average of a finite sequence of numbers: no matter how you pick the weights (e.g. the probability distribution), the average will always be greater than or equal to the minimum of the sequence; only when moving all the probability mass to the minimum value itself we get the lowest possible average value.

We attempt to give a concise mathematical demonstration for this in our continuous case, under a few assumptions for the sake of simplicity. Let us bound the search space of the optimization problem to an arbitrary region in the dynamics space. Let $j_{min} = J(\xi_{min})$ be the global minimum of $J(\xi)$ in the region delimited by the support of the current $p_\phi(\xi)$, potentially trimmed by the predefined search space in case of unbounded support. Therefore, note how the random variable $Y = J(\xi) - j_{min}$ is always greater than or equal than zero. Following the non-negativity and linearity properties of expected values, we consequently get:

$$\mathbb{E}[Y] = \mathbb{E}[J(\xi) - j_{min}] \geqslant 0 \tag{A.2}$$

$$\mathbb{E}[J(\xi)] \geqslant j_{min} \tag{A.3}$$

This final result indicates that regardless of how we design the distribution $p_\phi(\xi)$, we cannot do better than simply minimizing the objective function w.r.t. ξ only — indeed, we can only do worse. Therefore, the optimization problem should converge to the point-estimate ξ_{min} even if acting on the full distribution, assuming that our optimizer converges to the global minimum.

We then encourage future works to take these considerations into account when designing optimization of dynamics distributions with fixed offline data. For example, one could add custom constraints to promote variance in the dynamics parameters or, as shown by our work and BayesSim [19], move to probabilistic metrics to infer target parameters.

Finally, it is worth noting that these considerations do not apply to the original SimOpt method (as presented in [21]), as its objective function is conceptually different in the online

Table B.5
The amount of data used for each experiment, broken down into trajectories, state transitions, and the total time.

Experiment	Number of trajectories	Transitions per trajectory	Time per transition	Total transitions	Total time
Hopper	2	500	8 ms	1000	8 s
Hockeypuck	5	750	10 ms	3750	37.5 s
Sim-to-sim push	50	30	20 ms	1500	30 s
Sim-to-real push	1	1286	10 ms	1286	12.86 s

setting: SimOpt optimizes ϕ so that *a policy* trained on such distribution gets minimum squared residuals between its observed sim and real states; i.e., the objective is directly to get a policy that performs the same in both domains, as opposed to simple trajectory alignment. Thus, having policy training and data collection from the real-world system in the loop prevents the optimization from converging to a point estimate, as a policy trained on a point estimate would—in most cases—fail to generalize to the real world, resulting in a larger discrepancy between simulated and real trajectories than a policy trained on a non-degenerate dynamics distribution.

Appendix B. The impact of ϵ

To further analyze the impact of ϵ on DROPO's performance in presence of noise, we performed a secondary analysis where we corrupt the collected Hopper datasets with different levels of noise.

The results for the point-dynamics Hopper dataset described in Section 4.2.1 are depicted in Fig. B.13. In line with the main experimental evaluation, we observed a consistent drop in MSE where ϵ matches the noise level σ^2. In addition, the results in terms of the converged mean (Fig. B.13(b)) are consistent for $\epsilon \geq \sigma^2$, and the standard deviations are very low, corresponding to a correct identification of point dynamics parameters.

We finally performed a similar analysis for the distribution recovery case, which produced similar results: the dynamics distribution recovery is most accurate at ϵ close to the noise variance σ^2, which also corresponds to the sharp drop in MSE. In particular, note how the converged standard deviations of the four masses resemble the ground truth distributions defined in Section 4.2.2, as long as ϵ does not increase to overly large values w.r.t. the current noise level.

Appendix C. Sensitivity to the amount of data

As observed in our experiments section, DROPO succeeded to transfer RL pushing policies to the real world with as little as a single demonstrative trajectory for parameter inference. In this section, we aim to quantify the sensitivity of DROPO to the amount of offline data collected. To this end, we analyzed the converged means of the dynamics distributions with different amount of state transitions in different noise levels, for the Hopper environment in the point-dynamics setting of Section 4.2.1. In particular, we tested DROPO with 10, 30, 100, 300, 1000 and 3000 state transitions sub-sampled from the offline dataset, and kept ϵ fixed at the value of 10^{-5}. We report the confidence interval of the converged mean of each mass of the Hopper in Fig. C.15, obtained from 4 runs for each evaluation.

We observe that, when noise variance is around 10^{-8} or lower, as little as 10 state transitions are enough for DROPO to converge to values very close to the ground truth values. For higher noise levels, this number increases to several hundreds of state transitions. However, note the relatively small scale on the y-axis, where confidence intervals get as wide as 10 grams at most.

For the sake of clarity, we finally combine and report in Table B.5 the amount of data used in all our experiments. Target data has been collected in less than a minute of interaction for all tasks, motivating the adoption of offline DR optimization for critical-to-operate environments. Noticeably more data has been used to infer dynamics for the Hockeypuck environment, due to the nature of the task and the robot arm moving through the air before hitting the hockey puck, with the puck moving only for about a second in each trajectory (thus, only a small subset of

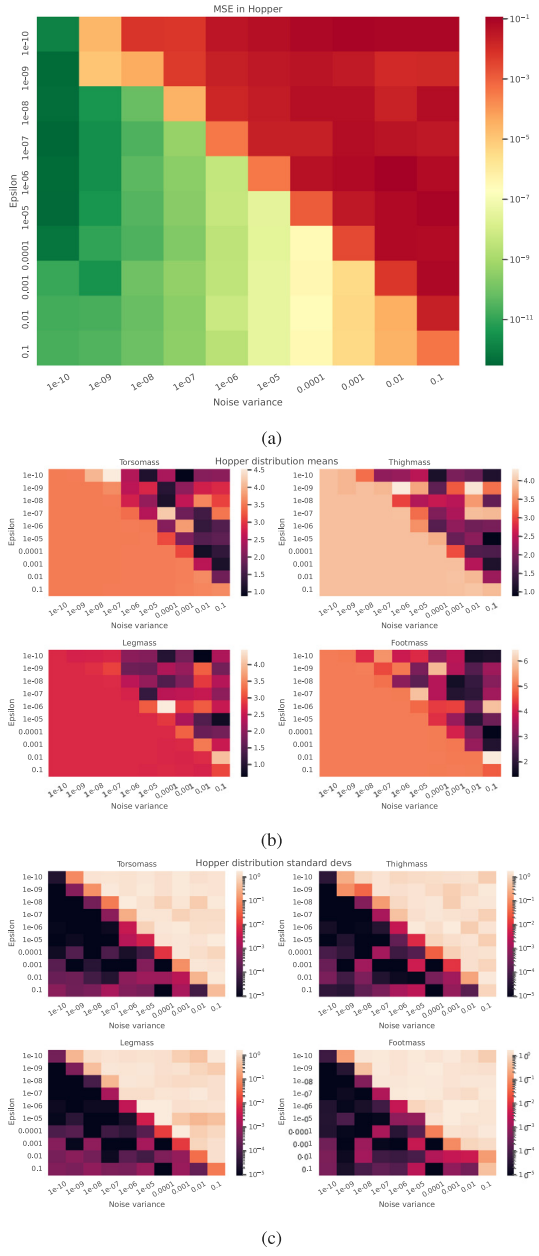

Fig. B.13. Relationship between noise level, epsilon and the resulting (a) MSE and (b, c) distribution, expressed by mean and standard deviation in the Hopper point-dynamics system identification case (results averaged over 4 random seeds).

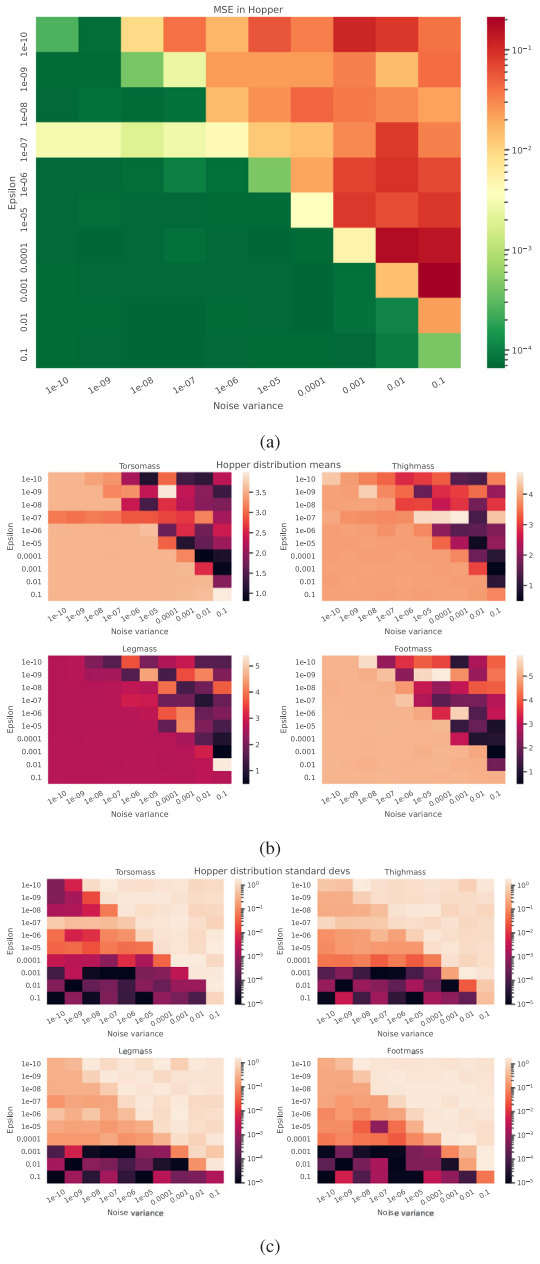

Fig. B.14. Relationship between noise level, epsilon and the resulting (a) MSE and the (b, c) distribution, expressed by mean and standard deviation in the Hopper dynamics distribution recovery case (results averaged over 4 random seeds).

state transitions could be used to infer the dynamics of the hockey puck).

Appendix D. Complete BayesSim results

BayesSim parameterizes the resulting dynamics distributions as a mixture of multivariate Gaussians, while other methods used simple Gaussian distributions. In order to clearly present the inferred distributions in comparison with the other methods in the Experiments section of the paper, we simplified the BayesSim results by only reporting the diagonal uncorrelated approximation of the Gaussian mixture. While this gives a good intuition about the accuracy of the results, it loses some of the

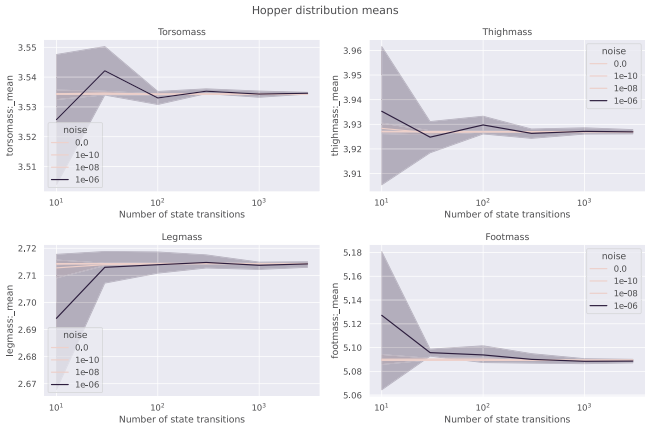

Fig. C.15. Hopper dynamics parameter estimated mean with its 95% confidence interval (estimated by bootstrapping over 4 random seeds).

information; hence, we here provide plots of the full mixture Gaussian distributions.

In this section, we provide the full BayesSim inference results by plotting histograms and joint distributions over all pairs of dynamics parameters. Each color corresponds to an independent training run of the BayesSim inference network.

D.1. Hopper

Here, we present the complete dynamics estimation results in Hopper. The ground-truth values for the point dynamics recovery experiment were reported in Table 1 and for the dynamics distribution recovery in Fig. 3.

The parameters (from param0 to param3) represent the masses of consecutive Hopper links—torso, thigh, leg and foot. In the unmodeled phenomenon experiment (Fig. D.18), the torso mass is excluded as it is fixed to an incorrect value and is not inferred during optimization. In this case, as previously argued, the test environment does not lie within the distribution of environments used for training data generation. As a result, the evaluation data lies outside of the training domain of the BayesSim inference network. This is likely the reason why the network produces a distribution with a large standard deviation centered around the mean of the search bounds.

We observe that in all distribution recovery experiments (shown in Fig. D.16, Fig. D.17, and Fig. D.18), BayesSim correctly returns a mixture model that is effectively a unimodal Gaussian distribution (with the weights of the remaining mixture components very close to zero).

In Fig. D.17, we also observed a significant discrepancy between different repetitions of the experiment, i.e. independent training runs of the BayesSim inference network.

D.2. Panda

Here, we present the full BayesSim results for the Panda pushing task. The estimated dynamics parameters are, in order, the mass of the box, the friction coefficient values along axes x and y, and the x and y position of center of mass of the box.

In the sim-to-sim inference experiment (Fig. D.19) we observe that, despite there being some variance between random seeds, the models generally agree that the center of mass along x and y should be shifted in the positive direction, and that the mass

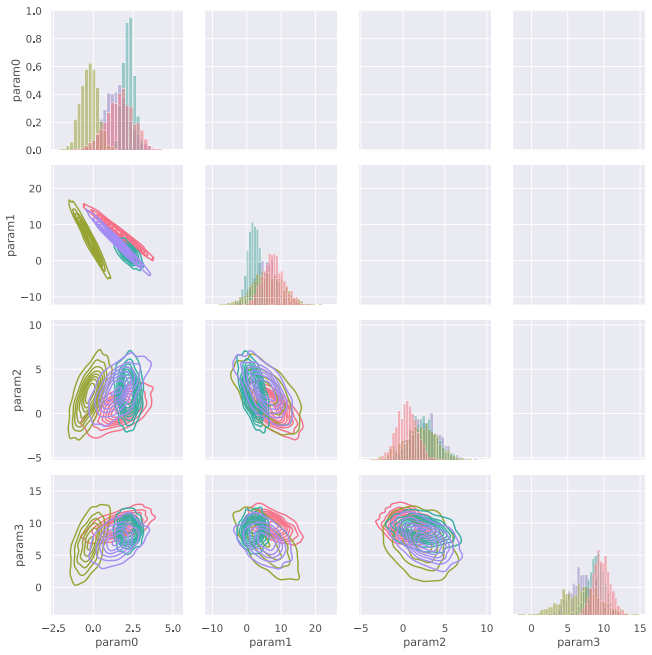

Fig. D.16. Complete dynamics identification results in Hopper with BayesSim. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

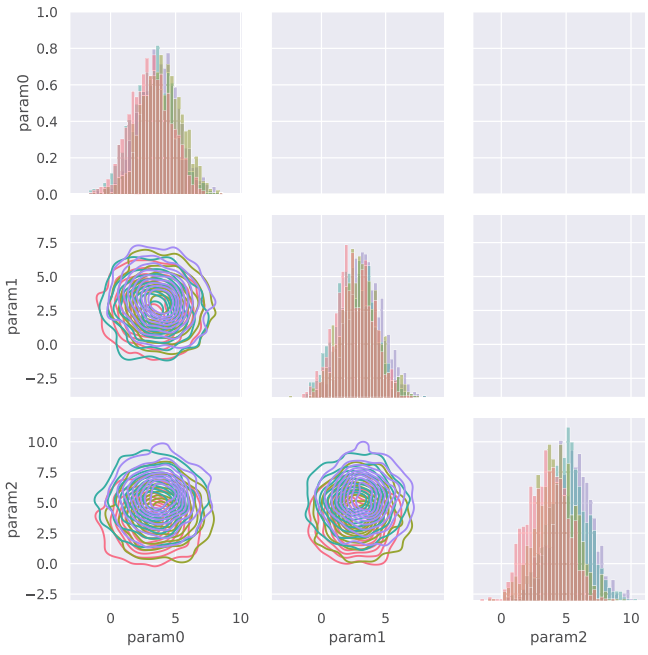

Fig. D.18. Complete dynamics estimation results in Hopper with unmodeled phenomena with BayesSim. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

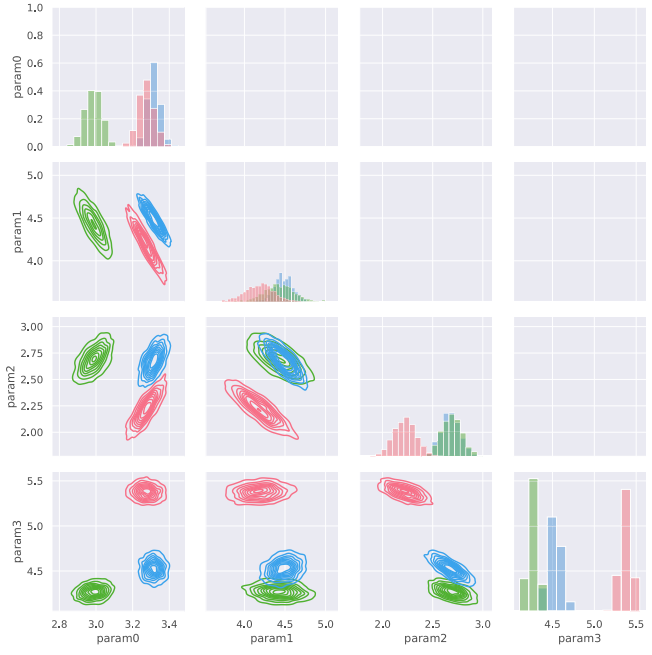

Fig. D.17. Complete dynamics distribution recovery results in Hopper with BayesSim. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

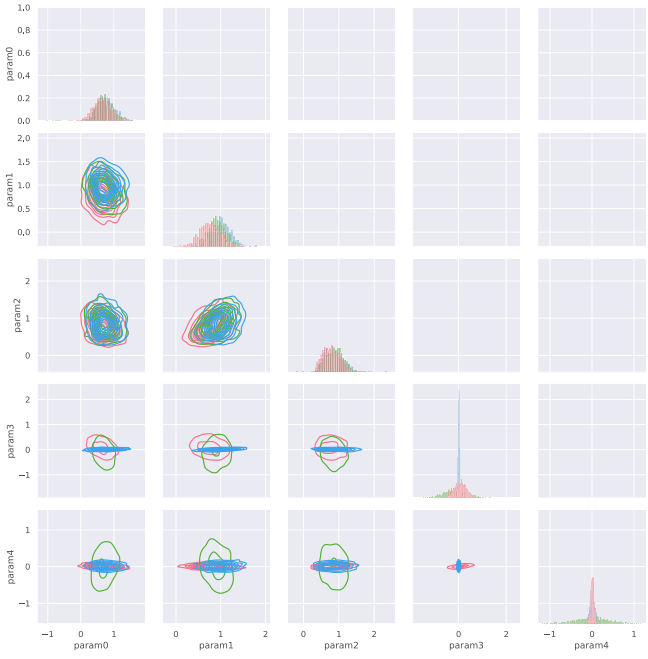

Fig. D.19. Complete dynamics distribution recovery results in Panda with BayesSim. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of the object is rather hard to identify (as expressed by the large standard deviation for param0).

However, when operating on real-world data, the model's performance degrades significantly, especially with the neural network features (MDNN), shown in Fig. D.20. In this case, the model produces very large standard deviations, potentially hindering the training process and often leading to unfeasible physical parameters—e.g. negative masses or box center of mass that lies outside the box boundaries. To address this issue, all values were clipped inside the simulator and kept within the pre-identified search space, which made the policy training possible. Like in the unmodeled Hopper experiment, we suspect that this behavior is the result of the network being evaluated on out-of-distribution

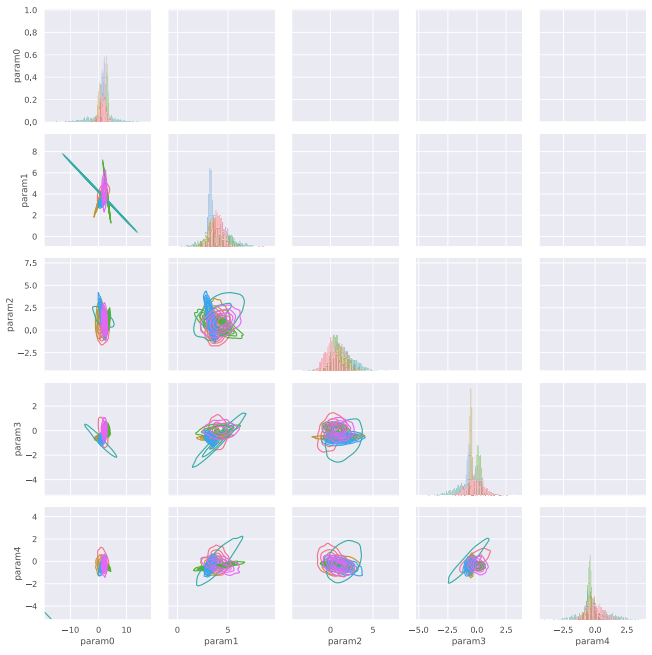

Fig. D.20. Complete real-world dynamics estimation results in Panda with BayesSim (MDNN features). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

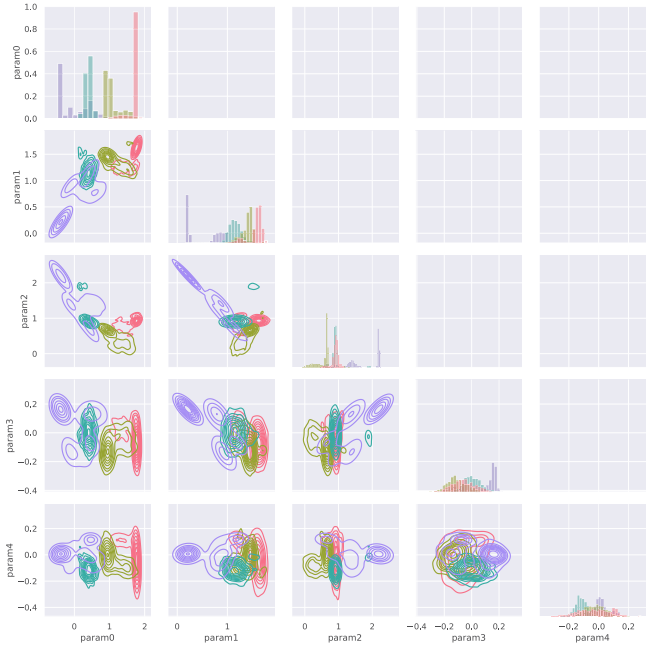

Fig. D.21. Complete real-world dynamics estimation results in Panda with BayesSim (MDRFF features). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

data. This hypothesis appears stronger when looking at the results obtained with the quasi-random Fourier features, where the inferred distributions look more sensible, although in some cases the mixture components still predict the mass to be negative (Fig. D.21). The feature extraction process reduces the dimensionality of the input data, making it easier for the model to generalize to new data, as it is more likely that the noisy, unmodeled real-world trajectory would appear close to data seen during training.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013.
[2] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[3] W. Zhao, J.P. Queralta, T. Westerlund, Sim-to-real transfer in deep reinforcement learning for robotics: a survey, in: IEEE Symposium Series on Computational Intelligence (SSCI), 2020.
[4] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, Sim-to-real: learning agile locomotion for quadruped robots, in: Robotics: Science and Systems, 2018.
[5] F. Sadeghi, S. Levine, CAD2RL: Real single-image flight without a single real image, in: Robotics: Science and Systems, 2017.
[6] A. Hämäläinen, K. Arndt, A. Ghadirzadeh, V. Kyrki, Affordance learning for end-to-end visuomotor robot control, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.
[7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[8] F. Sadeghi, A. Toshev, E. Jang, S. Levine, Sim2Real view invariant visual servoing by recurrent control, in: CVPR, 2018.
[9] X.B. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-real transfer of robotic control with dynamics randomization, in: IEEE International Conference on Robotics and Automation (ICRA), 2018.
[10] K. Arndt, M. Hazara, A. Ghadirzadeh, V. Kyrki, Meta reinforcement learning for sim-to-real domain adaptation, in: IEEE International Conference on Robotics and Automation (ICRA), 2019.
[11] R. Antonova, S. Cruciani, C. Smith, D. Kragic, Reinforcement learning for pivoting task, 2017.
[12] A. Nagabandi, I. Clavera, S. Liu, R.S. Fearing, P. Abbeel, S. Levine, C. Finn, Learning to adapt in dynamic, real-world environments through meta-reinforcement learning, in: International Conference on Learning Representations, ICLR, 2019.
[13] M. Hazara, V. Kyrki, Transferring generalizable motor primitives from simulation to real world, IEEE Robot. Autom. Lett. 4 (2019).
[14] OpenAI, Learning dexterous in-hand manipulation, Int. J. Robot. Res. 39 (2020).
[15] Y. Du, O. Watkins, T. Darrell, P. Abbeel, D. Pathak, Auto-tuned sim-to-real transfer, 2021.
[16] E. Valassakis, Z. Ding, E. Johns, Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.
[17] K. Arndt, A. Ghadirzadeh, M. Hazara, V. Kyrki, Few-shot model-based adaptation in noisy conditions, IEEE Robot. Autom. Lett. (2021).
[18] A. Rajeswaran, S. Ghotra, B. Ravindran, S. Levine, EPOpt: Learning robust neural network policies using model ensembles, 2017.
[19] F. Ramos, R.C. Possas, D. Fox, BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators, 2019.
[20] Y.-Y. Tsai, H. Xu, Z. Ding, C. Zhang, E. Johns, B. Huang, DROID: Minimizing the reality gap using single-shot human demonstration, IEEE Robot. Autom. Lett. 6 (2) (2021).
[21] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, D. Fox, Closing the sim-to-real loop: Adapting simulation randomization with real world experience, in: IEEE International Conference on Robotics and Automation (ICRA), 2019.
[22] B. Mehta, M. Diaz, F. Golemo, C.J. Pal, L. Paull, Active domain randomization, in: Conference on Robot Learning, 2020.
[23] F. Muratore, C. Eilers, M. Gienger, J. Peters, Data-efficient domain randomization with Bayesian optimization, IEEE Robot. Autom. Lett. 6 (2021).
[24] S. Kolev, E. Todorov, Physically consistent state estimation and system identification for contacts, in: 2015 IEEE-RAS 15th International Conference on Humanoid Robots, IEEE, 2015.
[25] A.D. Allevato, E. Schaertl Short, M. Pryor, A.L. Thomaz, Iterative residual tuning for system identification and sim-to-real robot learning, 44 (7) (2020) 1167–1182.
[26] F. Muratore, M. Gienger, J. Peters, Assessing transferability from simulation to reality for reinforcement learning, IEEE Trans. Pattern Anal. Mach. Intell. (2019).

[27] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, J.Z. Kolter, End-to-end differentiable physics for learning and control, in: Advances in Neural Information Processing Systems, 31, 2018.
[28] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017.
[30] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, ICML, 2018.
[31] J. Rapin, O. Teytaud, Nevergrad - A gradient-free optimization platform, 2018.
[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, 2016.
[33] A. Kumar, J. Fu, M. Soh, G. Tucker, S. Levine, Stabilizing off-policy q-learning via bootstrapping error reduction, in: Advances in Neural Information Processing Systems, 2019, pp. 11761–11771.
[34] E. Todorov, T. Erez, Y. Tassa, Mujoco: A physics engine for model-based control, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012.
[35] A. Ghadirzadeh, A. Maki, D. Kragic, M. Björkman, Deep predictive policy training using reinforcement learning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[36] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, J. Mach. Learn. Res. 17 (1) (2016).
[37] K. Arndt, O. Struckmeier, V. Kyrki, Domain curiosity: Learning efficient data collection strategies for domain adaptation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.
[38] J. Zhang, J. Wang, H. Hu, Y. Chen, C. Fan, C. Zhang, Learn to effectively explore in context-based meta-RL, 2020, Arxiv.
[39] B. Mehta, A. Handa, D. Fox, F. Ramos, A user's guide to calibrating robotic simulators, in: Proceedings of the 2020 Conference on Robot Learning, in: Proceedings of Machine Learning Research, 155, PMLR, 2021, pp. 1326–1340.
[40] J. Peters, K. Muelling, Y. Altun, Relative entropy policy search, in: AAAI, 2010.

Gabriele Tiboni was born in Turin, Italy in 1997. He received a B.Sc. degree in Computer Engineering in 2019, and a M.Sc. degree in Data Science & Engineering in 2021, both at Polytechnic University of Turin. He is currently a Ph.D. student in the Visual And Multimodal Applied Learning Laboratory at the Polytechnic University of Turin. His main research interests include machine learning and computer vision applied for learning autonomous systems in the field of robotics.

Karol Arndt obtained a Bachelor degree in Control Engineering from Poznan University of Technology (PUT; Poznan, Poland) in 2016 and a double Masters from PUT and Université Paris-Saclay (Paris, France) in 2017. Since 2018, he is currently a doctoral student at Aalto University, Helsinki, Finland.

His research interests include domain adaptation, simulation to real transfer, and the applications of meta learning and reinforcement learning in robotics in general.

Ville Kyrki received the M.Sc. and Ph.D. degrees in computer science from Lappeenranta University of Technology, Lappeenranta, Finland, in 1999 and 2002, respectively. Since 2012, he is currently an Associate Professor of Intelligent Mobile Machines with Aalto University, Helsinki, Finland. His primary research interests include robotic perception, decision making, and learning. Dr. Kyrki is a Fellow of Academy of Engineering Sciences (Finland) and the founding Chair of the IEEE Finland Section Jt. Chapter of CS, RA, and SMC Societies. He was an Associate Editor for the IEEE Transactions on Robotics in 2014–2017.