POLITECNICO DI TORINO

Master's degree in
Data Science and Engineering

Master Thesis

Towards safe and efficient transfer of robot policies from simulation to real world

Supervisors:
Prof. Barbara Caputo
Prof. Ville Kyrki
M.Sc. Karol Arndt

Candidate
Gabriele Tiboni

October 2021

Towards safe and efficient transfer of robot policies from simulation to real world Master Thesis. Aalto University, Espoo.

The work in this thesis will be submitted to the IEEE Robotics and Automation Letters (RA-L) journal.

Acknowledgements

I've had the pleasure to work in the Intelligent Robotics group at Aalto University for the full duration of my thesis. Each and every single student, Professor, or employee at Aalto University I've met has contributed to my fantastic stay in Helsinki. Despite the severe circumstances and consequences of the pandemic, Professor Ville Kyrki has constantly kept the group an exciting working environment for everyone. I would like to thank Ville for investing his time and resources into the supervision of my work, providing constant support and precious feedbacks along the way. On the other hand, a special thanks goes to my advisor Karol Arndt for an endless number of reasons. Karol has guided me through the journey from day zero to the very last minute, contributing and supporting to this work on each single aspect. He taught me how rigorous yet fun research can be, and pushed me through each tough moment. More importantly, I doubt I'll find another mate as helpful and willing to help as Karol. In addition, I would like to express my gratitude to Professor Barbara Caputo for believing in my capabilities and offering her supervision during my thesis. I'm particularly grateful to all my Italian and Finnish friends as well, who never stopped believing in my work. I would like to thank Teemu, Oliver, Mimi, Simone, Diego and Andrea for the great memories spent together.

Ringrazio con maggiore attenzione gli amici di una vita, che anche in questa circostanza hanno saputo starmi vicino in questi mesi. Un ringraziamento a Nicolò, Alberto V., Alberto S., Andrea, Fabio e Federica per aver superato il limite intraprendendo un pazzo viaggio tra le foreste finlandesi.

Un ringraziamento speciale va senza dubbio alla mia famiglia che, forse per orgoglio, non ringrazio mai abbastanza. Michele, Silvia, Pietro, Benedetta e Paolo sanno come supportarmi senza neanche farmelo notare.

Infine, una note speciale va alla mia accompagnatrice più intima. Davina è stata la mia arma in più in tutte le battaglie vinte in questi mesi. Ti ringrazio di cuore per tutti i tuoi sforzi e il tuo supporto, che sei sempre pronta a darmi con sorprendente piacere. Mi auguro non sia l'ultimo traguardo che tu possa aiutarmi a raggiungere.

Abstract

Reinforcement learning approaches have demonstrated great promise for flexible and efficient robot learning. Yet, current data-driven algorithms require large amounts of data to learn even simple robotic tasks and leave open safety concerns when training on real hardware. Physics simulators offer several advantages in this regard, allowing to train robot policies entirely in simulation before the final deployment onto the real world. However, while this solves the problem of fast and safe data collection, any small inaccuracy or parameter discrepancy in simulation may potentially lead to policies which do not directly transfer to the real system.

Domain randomization has recently gained a lot of traction as a method to overcome the reality gap experienced by transferred policies, encouraging robustness to domain shifts by randomizing physical parameters in the simulated scene. As current applications of this method require tedious manual engineering to find optimal randomization ranges, we introduce a novel algorithm to automatically identify the parameter distributions to train on, based on limited real-world data safely collected through human demonstrations. We show that the optimized distributions are capable of compensating for unmodelled phenomena in simulation. Furthermore, we evaluate our method on two real robots demonstrating a successful domain transfer and improved performance over prior methods.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the overall machine learning world rapidly grows over time and finds applications in all sorts of fields and research areas, the common narrative still seems to be that of a somewhat magic artificial intelligence algorithm that does it all. The data-driven approach generally offers high flexibility when solving a problem, avoiding the need to hard-code complex tasks for all possible different situations. Imagine, for example, manually engineering a computer vision program for recognizing images of cats or explicitly coding an artificial player to play chess professionally. Such tasks clearly require a novel approach to be solved. This is when machine learning comes into play, bringing tools and techniques to extract knowledge from raw experience (or data), with minimal injection of hard-coded rules. Several astonishing success stories in the past decade have shown how learning from data may reach and even outperform human capabilities [1, 2].

It's important to highlight that profoundly different learning frameworks exist for different scenarios, depending on the available data and resources. Indeed, learning from data should still be split into three broad categories: supervised, unsupervised, and reinforcement learning.

- Unsupervised Learning (UL): it deals with the problem of extracting hidden patterns from raw data with no expert signal.
- Supervised Learning (SL): it uses a labelled dataset to learn general rules for mapping input data to the desired output classes or values.
- Reinforcement Learning (RL): rather than providing ground truth labels, a quantitative feedback signal is given for each decision made on the input data, such that the desired outputs can be learned through a trial-and-error process.

While the first two approaches rely on a fixed given dataset to learn the task at hand, reinforcement learning generally has the advantage of allowing data to be arbitrarily collected during the training process and continuously improve the model based on the obtained feedback signal. Intuitively, such process may not only be applied to situations where the optimal decisions (or *actions*) are generally known and want to be learned from experience, but even when optimality needs to be discovered altogether. For example, we may be interested in learning superhuman skills in video games [3], or optimally place chips when manually designing hardware is too complex or time-consuming [4]. It may then seem just a matter of time until RL agents — the entities making decisions autonomously — learn to ace every complex task. However, this is currently not the case. There are mainly three reasons as to why reinforcement learning has yet to gain widespread adoption.

First of all, a feedback signal may be hard to design or not accessible at all. Let's suppose we want to teach a humanoid robot how to do grocery shopping. How would you reward each single decision the robot makes, encouraging it to go all the way from leaving the house to paying for the desired goods and coming back home? How would you automatically penalize unwanted behavior along the way?

Secondly, an environment to collect data from may not be readily available, or even inconvenient and costly to query. This is the case when thinking of robotic tasks to be learned in the real world: while a robot can potentially freely explore to learn optimal behavior, uncontrolled exploration may be highly dangerous for the surrounding environment as well as for the robot itself. On top of this, further costs of wear and tear and human supervision of real hardware have to be taken into account.

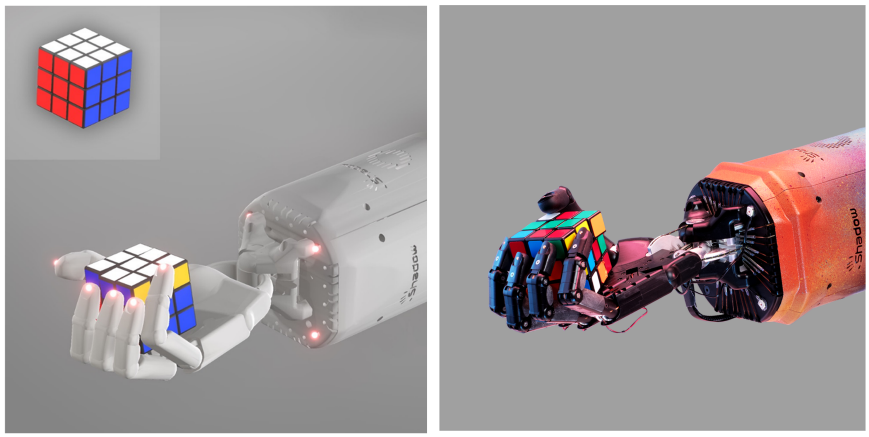
Lastly, current reinforcement learning algorithms have technical limitations. Indeed, in order to converge to an optimal behavior large amounts of data are required, translating to a need for extensive and exhaustive exploration at training time. Again, this is not be acceptable whenever safety or data-efficient concerns are implicitly embedded in the task to be solved.

Overall, as much as robotic tasks can be naturally phrased as reinforcement learning problems, we are faced with many limitations when it comes to learning optimal robot behavior through trial-and-error. This is why the dream of self-taught intelligent robots has been lagging behind. Indeed, despite advances in deep learning and computer vision, we have not yet come across fully autonomous robots delivering services in our homes, or managing logistics, surveillance and other appealing applications.

Robot learning All research efforts that point towards making robots acquire novel skills fall under the umbrella of *Robot Learning*, which generally describes the point of intersection between machine learning and robotics. Although there exists a second narrative to robot learning which tries to avoid data-driven techniques and exploit traditional adaptive control [5], there is a clear surge of interest in learning algorithms for developing smart robots that can operate in highly dynamic environments [6]. This shift is highly motivated by the need for reducing manual engineering when coding complex tasks. In addition, correctly handling all situations that may occur at test time is often not possible when the robot has no knowledge on how to generalize.

Early examples of robot behaviors learned through reinforcement learning already demonstrate the potential flexibility robots can reach: a wheeled mobile robot (OBELIX) has been taught to push office boxes [7]; a Zebra Zero robot arm learned to solve a peg-in-hole insertion task [8]; autonomous robust control for a helicopter has been achieved [9]; a humanoid robot learned to autonomously balance a pole on its hand [10]. However, many efforts were still needed to overcome the aforementioned major challenges in RL for robot learning. To push state-of-the-art forward, researchers in the field have found different ways to deal with the three presented issues. They came out with strategies such as learning from human demonstrations in a supervised learning fashion [10, 11, 12], constraining exploration in the real-world [13, 14, 15], or learning in virtual environments. In particular for robot learning, the latter refers to the use of physics simulators to model the real environment and provide a safe, reliable way to collect data with no harm and minimal costs. The idea of learning in simulated environments is not new, and proved to be effective even in the area of evolutionary robotics (ER) [16, 17]. In contrast with reinforcement learning, ER learns optimal control by iteratively making new generations of controllers as random variations of the best $N-$controllers from the previous iteration.

As physics simulators started to get more sophisticated and well-supported for programming and modeling robotic tasks [18], the field of simulator-based robot learning increasingly attracted researchers, with a substantial number of promising works showing robot behaviors learned entirely in simulation [19, 20, 21] (see Fig. 1.1). Perhaps the most remarkable recent success story in the field is a work by OpenAI [22], where dexterous in-hand manipulation is learned through reinforcement learning in the context of a robotic hand asked to re-position an object arbitrarily on its palm. The authors were able to learn the task solely in simulation, and directly deploy the behavior in the real-world without further adaptation. In general, the idea of training reinforcement learning controllers (or *policies*) in simulation and transferring such knowledge on the real setup is referred to as the *sim-to-real transfer* paradigm. As promising as it may look for inexpert readers, the sim-to-real approach does not yet come without challenges, effectively making works such as [22] incredibly more impressive.



(a) (b)

Figure 1.1: The simulated (a) and real (b) environments in OpenAI's work [19]. A policy for solving the Rubik's cube is trained entirely in simulation and later deployed on the real world.

Domain shift While simulators entirely remove the need for safety constraints and allow for less data-efficient approaches, learning in simulation is notoriously challenging due to inaccuracies and unmodelled effects when modeling the real task [16, 23]. Any discrepancy in behavior between the simulated environment and the real setup will result in a distribution shift between training and test data, leading to a problem of *domain shift* in machine learning terms. Physics simulators inherently make use of approximated forward dynamics models — how the environment reacts given a certain starting state and action — which may be particularly hard to model in the case of in-contact tasks. On top of this, simulators have to deal with discrete-time integration for computing the kinematic properties of each object in the scene, potentially leading to instabilities.

Moreover, any further discrepancy in physical parameters (e.g. masses, friction coefficients, joint dampings), actuator dynamics, noise or sensor delay present in the real world will likely increase the gap between the two domains [24], formally denoted in literature as the *reality gap* (see section 3.3). Finally, real world dynamics may change over time due to different causes such as wear and tear of the robot and temperature/humidity changes, effectively making the transfer an even harder problem and including the need for the robot to learn a robust behavior to shifts in dynamics.

Whenever a shift in data distribution is involved in the process of a learning algorithm, the issue in machine learning is addressed as a *transfer learning* problem

of some type, where knowledge needs to be transferred from one domain to another, on possibly even different tasks. In general, machine learning algorithms rely on the assumption that data collected or provided at training time is observed according to a theoretical unknown probability distribution $P(x)$. Then, we wish to use the sample of data available from $P(x)$ as a training set to infer global knowledge, i.e. minimize the expected value of an objective function of interest over $P(x)$ entirely (empirical risk minimization). Such function is generally called a *loss* function, and may be arbitrarily designed to learn the underlying task, e.g. map input data x to the desired output values y (in supervised learning), or maximize the reward signal in reinforcement learning. When the distribution $P(x)$ is held constant across training and test data, the *probably approximately correct (PAC)* learning framework [25] assures us that generability in minimizing the loss function on the entire distribution can be achieved with an arbitrary probability and up to a given accuracy, given that a certain number of i.i.d. data points is observed from $P(x)$ and a correct parameter search space is used. However, such theoretical guarantees give little to no help in practice, as many assumptions are often dropped. This is certainly the case in sim-to-real transfer scenarios, where real data likely follows a different distribution than the one observed when querying the simulator.

Overcome the reality gap Due to the reality gap, the dream of transferable reinforcement learning policies learned from endless synthetic data has not yet become a reality. Several approaches have however been proposed by researchers all over the world to ultimately cross the gap between the simulated source domain and the real one, demonstrating astonishing advances in recent years.

A traditional approach to bring simulation closer to reality is known as *System Identification* (SI), which broadly refers to all strategies which improve the mathematical model of the physical system to make it more realistic. Such approach has been commonly used to automatically infer physical parameters of the simulator given real-world data [26, 27, 28, 29]. However, as even the most accurate simulator won't be perfect, system identification methods may still lead to policies biased on the source domain.

A more recent alternative to tackle the problem is *Domain Randomization (DR)*, which encourages robustness to distributional shifts by training on an ensemble of simulated environments with varying physical parameters [20, 30, 19, 31, 32], varying appearance [33, 34, 35] or both at the same time [36, 22]. In other words, reinforcement learning policies with DR are learned as to maximize the reward obtained on a training environment which constantly changes over time according to some predefined parameter ranges, in order to learn optimal behavior robust to discrepancies in the forward model or data representation. The concept is rather simple, but has shown to be crucial on a number of robotic tasks for successfully transfer from simulation to reality, and gained substantial popularity

in robotics in recent years. In the context of varying physical parameters — also known as *dynamics randomization* — commonly randomized values are masses and sizes of the objects in the modelled scene, friction coefficients of interacting surfaces, joints damping of the robotic arm, simulator-specific parameters for modeling contacts, gains for closed-loop joint motion control, sensor noise, and action delay.

One open problem when dealing with dynamics randomization is however the lack of knowledge on the exact randomization distributions of parameters the simulator should be sampled from. The authors in [22] highlight the importance of centering the distributions of randomized parameters around the real measured values — when possible — but current domain randomization works in practice rely on tedious manual engineering to come up with feasible dynamics distributions [23]. Intuitively, overly wide parameter ranges will hinder the training process or even make it impossible to find a single policy which learns optimal behavior robust to all variations, while point-estimate values will likely result in policies overfitting on training data and lacking generability. The latter problem is further exacerbated by the tendency of RL agents to exploit potential simulator glitches to their advantage [37, 38, 39], and generally optimistically bias learning on training data.

Objectives and contributions The work in this thesis is motivated by an ongoing effort from researches all over the world in designing affordable and efficient algorithms for sim-to-real transfer using domain randomization [40, 41, 42]. By affordable, we generally refer to low cost of data collection and introduction of safety guarantees, two key goals when designing sim-to-real transfer algorithms in robotics. However, as previously stated, popular applications of domain randomization as a method to overcome the reality gap require back-and-forth tuning of RL policies with constant access to the real robot to pick the best randomization distributions. Such approach is, in general, sub-optimal as much manual work is invested into the process. Several alternatives in recent years have been proposed in the attempt to automatically learn the best parameter ranges for training with DR [43, 44, 36, 42], all of which still require online access to the real robot during the process or do not explicitly promote variance in the final distribution [45]. We believe that iteratively collecting data on the real setup with partially-optimized policies may be prohibitively expensive and should be avoided. Therefore, we aim at studying a more affordable yet efficient way to infer randomization ranges, avoiding policy rollouts on real hardware altogether. The ultimate goal of this work is to move data-driven robot learning research one step closer towards the design of sim-to-real transfer frameworks which are safe, cheap to execute and efficient.

The contributions of the work in this thesis are the following:

(1) introducing *Domain Randomization Off-Policy Optimization* (DROPO), a

novel method for offline optimization of the simulation parameters distribution based on a fixed set of safely collected data through human demonstrations;

(2) presenting an experimental evaluation of DROPO to show convergence to ground truth dynamics distributions in simulation;

(3) showing that the optimized distributions obtained can effectively shorten the reality gap and make up for unmodelled phenomena when policies are trained on such ranges and directly transferred to the real world, with improved performances over prior methods;

We test DROPO on two real robotic tasks: a sliding task with the Kuka LWR4+ robotic arm equipped with a hockey stick and asked to hit a puck onto a target position; a pushing task with the Franka Panda robotic arm asked to push a heavy box with shifted center of mass. We demonstrate that DROPO is capable of using manually collected data to infer randomization ranges for training reinforcement learning policies in simulation which can successfully transfer their behavior to the real world.

Learning from images in an end-to-end fashion is out of the scope of this thesis. Therefore, we deal with the randomization of physical parameters only and tackle robot learning assuming kinematics information of the objects in the scene is known at each point in time.

Structure of the contents The thesis is organized as follows:

- **Chapter 2** covers important mathematical preliminaries, technical terminology and background concepts needed to understand the contents of the thesis.
- **Chapter 3** dives into the details of the sim-to-real transfer challenge, presenting and motivating the existence of the reality gap and formally introducing domain randomization as a method to shorten it.
- **Chapter 4** covers the research efforts in recent years in making affordable and efficient algorithms for robot learning which use some form of domain randomization to transfer from simulation to real world.
- **Chapter 5** describes in detail our novel method DROPO, together with the formulation of the underlying problem to be solved.
- **Chapter 6** provides the experimental evaluation of DROPO, both on toy simulation environments and on two real robotic tasks: sliding a hockey puck and pushing forward a heavy box with shifted center of mass.

- **Chapter 7** gives an overall conclusion to the work in this thesis, highlighting the key takeaways and open directions for further research.

Chapter 2

Background

In this chapter, we present the necessary background tools to understand the contents of the thesis. In particular, the reinforcement learning paradigm is briefly introduced in technical terms, together with the problem of domain shift in machine learning. Finally, a motivation on gradient-free optimization algorithms is given with a short description of Covariance matrix adaptation evolution strategy (CMA-ES), which is later used by DROPO in our experiments. For a more thorough explanation of the theoretical concepts, we provide and encourage reading the external references.

2.1 Reinforcement Learning

The Reinforcement Learning (RL) paradigm exhaustively presented in [46] fills the gap in the world of machine learning where supervised and unsupervised learning fall short. Reinforcement Learning describes a setting where interaction with an underlying environment is needed to learn *what to do*. We are therefore interested in mapping situations (referred to as *states*) to decisions (*actions*), in order to discover the desired behavior and successfully interact with the environment. A key characteristic of RL is that optimal behavior is not generally known in advance or, in other words, the learner is not told which actions to take in each particular moment, as opposed to the more common supervised learning paradigm. Indeed, a labelled training dataset in such interactive setting would be challenging to design, as expert behavior representative of all different situations should be provided. In this regard, instead of extracting rules from a fixed labelled dataset for mapping input data to desired outputs, the goal is to progressively learn the optimal decision for each state by trial and error, penalizing unwanted behavior and rewarding good actions. To achieve this, the learner, which is better known as *agent* in RL terms,

clearly needs to get some kind of feedback signal in response to its decision making. We are then ready to phrase the general problem setting of reinforcement learning, which closely resembles the actual learning setting humans constantly deal with: a goal-seeking agent placed in an unknown environment is given information on the current situation and asked to act accordingly, causing the environment to change its state; at each interaction, the agent is given a scalar reward signal for it to learn behavior such as to maximize the total reward. Note how the problem is inherently interactive and closed-loop, as a single action may not itself result in a correct or improper behavior, but rather a set of actions may be needed to reach the goal.

Thanks to its generality, reinforcement learning interacts with several fields beyond engineering and mathematics, such as psychology and neuroscience. RL is able to bring together statistics, computer science, optimization theory and other branches of mathematics to provide a general yet non-trivial framework for learning optimal decision making from experience. An illustration of the reinforcement learning flow is showed in Fig. 2.1.

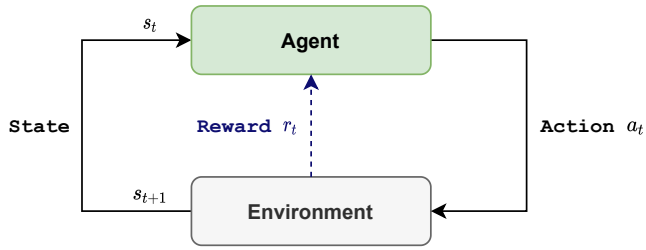


Figure 2.1: Illustration of the reinforcement learning framework. A goal-seeking agent aims at maximizing a numerical reward signal while interacting with the environment. The agent is asked to make a decision a_t given information on the current state s_t, causing the environment to evolve to a new state s_{t+1}. The environment returns a scalar reward as a feedback of the interaction occurred.

A new interesting aspect arises in reinforcement learning as opposed to other kinds of learning, named the *exploration-exploitation trade-off*. Let us introduce it with a simple example. Suppose that our agent is a customer at a Finnish restaurant in town. The customer is seated and asked to choose their desired main course from the menu. They had already visited the same restaurant previously and found out that salmon soup was pretty good: should the customer try out new alternatives this time, running the risk of getting a worse-tasting dish, or stick to the previous favourite choice? As no other information is provided about the environment, it seems like the agent is posed with a dilemma on whether to *explore* the environment to potentially discover new favorable options, or to *exploit* the current knowledge to maximize reward in a greedy fashion. Intuitively, in this toy

scenario, the customer could visit the restaurant repeatedly and consecutively try out all options while keeping track of past choices and rewards (personal taste), assuming a deterministic and stationary environment — the cook never messes up the order and dishes always taste the same. However, in general, as the number of states and available actions grow and other challenges come into play (e.g. sparse rewards, safety concerns), it soon becomes prohibitively unfeasible to collect sufficient interaction samples to fully explore the underlying environment. Therefore, the agent generally needs to find a balance between exploration and exploitation while learning.

The simplified setting in the aforementioned example represents the building block of reinforcement learning problems where no sequential decision making is involved and only a single non-terminal state exists, that of the customer deciding on the menu. This simplified setting of RL is referred to as the *n-Armed Bandit Problem*, with n the number of discrete available choices. In practice, however, action spaces and state spaces may be continuous, the environment may change over time or behave stochastically with respect to each state-action pair, and rewards need to be maximized over consecutive actions (see Section 3.1 for more complex examples in robot learning).

Markov Decision Processes (MDPs) allow for a straightforward mathematical formulation of the RL paradigm. An MDP M is described by its state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution ρ_0, state transition probability function $p(s'|s,a)$ — or simply *dynamics* — and scalar reward function $r(s',s,a)$, assumed to be deterministic for simplicity. Here, the *Markov property* is assumed, indicating that the evolution of the environment only depends on the current state and action chosen, regardless of past history. The reinforcement learning problem associated with M thus involves an agent that receives some initial representation of the environment state $s_0 \in \mathcal{S}$ according to ρ_0, and it is then asked to select an action a_0 from the range of available actions $\mathcal{A}$. From the interaction with the environment, the agent moves forward to the next state $s_1 \sim p(s_1|s_0,a_0)$ and receives a feedback signal $r_0 = r(s_1,s_0,a_0)$. The process is then repeated either until a terminal state is found — in which case an *episode* is said to take place — or potentially forever for tasks where no such notion of individual episodes exists (continuing tasks). Recall that the agent has no knowledge of the environment dynamics $p(\cdot|s_t,a_t)$, as optimal behavior needs to be learned from experience rather than with exact methods (e.g. dynamic programming).

Without losing on generality, we can model the agent's goal as the maximization of the total reward in the long run, considering possibly infinitely many steps in the future weighted by an exponentially decaying discount factor $\gamma \in (0,1]$. This objective function takes the name of *discounted cumulative reward*, or *discounted*

return, defined as the total reward accumulated since a time instant t:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \tag{2.1}$$

We generally allow for $T \to \infty$ or $\gamma = 1$, but never both at the same time to prevent a divergent series. The decision making process carried out by the agent is modelled by means of a mapping from the current state to the probability of selecting each possible action. Such mapping is called a *policy* and is denoted as $\pi(a|s) = \mathbb{P}(a|s)$. Therefore, the policy π effectively encodes the agent's learned behavior by providing a probability distribution of actions for each possible situation. The reinforcement learning problem ultimately aims at optimizing π to achieve maximum expected discounted return since the start, as follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \tag{2.2}$$

Where $r_t = r(s_{t+1}, s_t, a_t)$ and actions are sampled according to π as $a_t \sim \pi(a_t|s_t)$. Note how the expected value is therefore taken over action probabilities, environment dynamics, and initial state distribution ρ_0, to account for all potential stochastic effects.

In practice, some parameterization of π needs to be chosen to perform the optimization problem (2.2). There is no definite answer on which parameterization is best, but there exist two main approaches for designing π: value function methods and direct policy search. Although both are universally used, the two approaches highly differ in their implementation details and algorithms used. For robot learning in particular, it seems that policy search methods are more practical and generally used [47], as they allow straightforward management of continuous action spaces.

2.1.1 Value function methods

Value function methods approach policy learning implicitly. Intuitively, they aim at estimating how favorable a particular state (or state-action pair) is, given past experience. In the case of the indecisive customer toy example, this translates to approaching the problem by trying out all available dishes and keeping an estimate of the reward associated with each choice, and finally design the policy to pick the dish whose value is maximum. In general for environments that involve multiple states and sequential decision making, the value of a state is defined in terms of expected return in the long run, given that as a starting state. Note, indeed, how the policy concept is now implicit: the complexity of the problem is moved in the estimation of expected cumulative rewards, rather than in the optimization of π

explicitly. The value of being in a certain state s and following policy π from there, is formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi [\ G_t \mid s_t = s] \tag{2.3}$$

with $v_\pi(s)$ denoted as the *state-value function* for policy π. Analogously, the *action-value function* $q_\pi(s, a)$ can be defined, which further takes into account the current selected action:

$$q_\pi(s, a) = \mathbb{E}_\pi [\ G_t \mid s_t = s, a_t = a] \tag{2.4}$$

We then wish to find a policy π^* which yields to the optimal state-value function v_*, such that $v_*(s) = v_{\pi^*}(s) \geqslant v_{\pi'}(s)\ \forall s \in \mathcal{S},\ \pi'$. When found, the optimal policy π^* would also result in an optimal action-value function q_*, similarly defined. In practice, such functions can be kept track of in a tabular fashion, explicitly modeling each state entry $s \in \mathcal{S}$. For continuous state and action spaces, however, parameterized function approximators such as neural networks are instead used.

Overall, we're left with the problem of maximizing value functions while exploring the environment. *Monte Carlo* methods do this by sampling estimates of the returns G_t at each time step, when collecting full episodes with a current policy π. Such approach usually excels for episodic tasks and it's expected to yield smaller variance in the estimated returns. On the other hand, *Temporal Difference* (TD) methods attempt to relax the need to wait for exact sampled returns by bootstrapping, i.e. making use of learned value function estimates to immediately start learning from a single transition. For example, TD methods can approximate $G_t \approx r_t + \gamma \hat{v}(s_{t+1})$ and update their belief about $\hat{v}(s)$ accordingly, as soon as r_t is obtained. Algorithms such as SARSA and Q-learning are examples of value-based TD-methods for RL problems with discrete action spaces. The latter has been successfully applied in the field of Deep Reinforcement Learning [48], dealing with the estimation of value functions in high-dimensional state spaces (e.g. images) with convolutional neural networks [X].

A full description of Monte Carlo and TD methods together with their associated algorithms is reported in [46] (chapters 5 and 6 respectively).

2.1.2 Policy Search

Policy search methods follow a different paradigm, performing the optimization problem (2.2) explicitly in the policy parameters space rather than estimating value functions. The more traditional value-based approaches are sometimes impractical in cases of high-dimensional continuous state and action spaces, due to their need to approximate returns for each state-action pair. Furthermore, theoretical guarantees on finding global optima are lost for function approximation cases. Policy search methods guarantee convergence to at least a local minimum [49, 46] and allow straightforward management of continuous actions, making them the

go-to choice for robotics applications. In addition, they allow for the incorporation of prior knowledge into the policy, such as adapting its structure conveniently or initializing the robot behavior with expert demonstrations.

Policy search methods assume a certain parameterization θ of the policy π_θ. Then, they aim at iteratively update the parameters θ towards the optimization of the objective function in (2.2), commonly noted as:

$$J(\theta) \ = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \tag{2.5}$$

How the computation of the policy update is performed differs from algorithm to algorithm. Policy gradient methods aim at estimating the gradient $\nabla_\theta J(\theta)$ to compute updates in a hill-climbing fashion, e.g. through backpropagation in the case of neural network function approximation. While finite-difference can be applied — estimating $\nabla_\theta J(\theta)$ by small perturbations on θ — the so called likelihood ratio methods are most commonly adopted in policy gradient literature. They exploit the REINFORCE [50] trick to estimate $\nabla_\theta J(\theta)$: referring to the generated trajectories — sequences of states and actions rolled out on the environment — as τ, the expected value in the objective function (2.5) can be rephrased in terms of $p_\theta(\tau)$, the probability of each trajectory to be rolled out given the current policy, as:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r(\tau)] = \int r(\tau) p_\theta(\tau) d\tau \tag{2.6}$$

with $r(\tau)$ the trajectory-associated return. Then, the following calculus trick

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \cdot \nabla_\theta \log p_\theta(\tau) \tag{2.7}$$

ultimately allows to state the gradient $\nabla_\theta J(\theta)$ in terms of visited states, actions selected and rewards obtained:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int p_\theta(\tau) \cdot \nabla_\theta \log p_\theta(\tau) \cdot r(\tau) d\tau && (2.8) \\ &= \mathbb{E}_\tau [\nabla_\theta \log p_\theta(\tau) \cdot r(\tau)] && (2.9) \\ &= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot r(\tau) \right] && (2.10) \end{aligned}$$

Eq. 2.10 finally provides a simple way to compute the gradient, e.g. through sample estimates of the expected value in Monte Carlo style (REINFORCE [50]). Note how the formulation resembles the famous cross-entropy loss in supervised learning, as it tries to maximize the probability of each action picked during the episode according to the reward obtained from there. More precisely, negative returns will lead to a negative gradient, encouraging a lower probability when

performing the update step, and viceversa for positive returns. In practice, further tricks are implemented to promote faster training times, such as subtracting a constant (or state-dependent) baseline from the rewards, or boostrapping as in the case of Temporal-Difference methods. In particular, Actor-Critic methods [51, 52] are a kind of policy gradient algorithms which learn both a parameterized policy (actor) and an estimate of the value function (critic), using the latter to approximate future rewards in TD fashion and allow single-step updates.

The Proximal Policy Optimization (PPO) family of reinforcement learning algorithms by OpenAI [53] are a perfect example of popularly used policy gradient methods. As they are fairly easy to implement and tune and still obtain close to state-of-the-art results, they commonly became the default algorithms in reinforcement learning for many applications. See Chapter 6 for examples of policies trained with PPO in our experiments.

2.1.3 Model-based vs. Model-free RL

A further important distinction in reinforcement learning approaches exists, namely the separation of model-based and model-free algorithms. As the main goal of the agent is to progressively improve its reward in the long run under generally unknown dynamics, RL algorithms may iteratively query the environment and explore until termination. However, more complex techniques such as *planning* can be implemented. Planning involves reasoning about the current action, either while learning or acting at test time, by predicting the future environment response. The prediction may come from a given model of the environment, as in the case of dynamic programming and value iteration, or may be estimated by the agent with previous experience. Whenever methods primarily rely on planning, we refer to these as *Model-based*, as they involve a (possibly learned) model of the environment. Otherwise we fall in the *Model-free* RL regime, where the agent solely relies on real samples of the environment and its learned policy to take the current action. All previously mentioned algorithms in the chapter such as SARSA, Q-Learning and Actor-Critic algorithms are model-free. On the other hand, Sutton and Barto also introduce Dyna-Q (see Chapter 8 in [46]), a learning algorithm for planning agents.

Regarding the applications of interest for the work in this thesis, robot learning in the context of sim-to-real transfer has also been referred to as model-based, at times [43, 54]. This is because a (simulated) model of the real environment is queried when learning instead of collecting real samples. However, this statement may be misleading with respect to the initial formulation of model-based algorithms, as policies learned in simulation do not necessarily act in the real world by planning.

2.2 Domain shift

Although the importance of the existence of domain shifts in machine learning models has been briefly motivated in Chapter 1, a more thorough theoretical explanation is needed to introduce the sim-to-real transfer challenge.

At the heart of machine learning models lies the idea of learning general rules starting from a finite sample of observed data, called the *training dataset*. We then wish to use such collected data to extract hidden patterns, or in other words to build accurate mathematical models able to map input data to the desired outputs. The latter may in general refer to labels (*supervised classification tasks*), real values (*supervised regression tasks*) or decisions (*reinforcement learning tasks*). We ultimately aim to deploy the model on previously unseen data (the *test dataset*), in order to map new samples automatically and intelligently. Therefore, generalizability is key when training models, such that learned mappings are not specific to the training dataset — a case known as *overfitting* — but resemble general rules that work on other relatively similar data. Here, the concept of *similar* describes the importance of learning on samples which are representative of test data, i.e. coming from the same statistical distribution. Learning theory has a number of theorems and studies that give theoretical guarantees on this setting, allowing learning algorithms to estimate mappings $f : \mathcal{X} \to \mathcal{Y}$ — from input data $\mathcal{X}$ to output values $\mathcal{Y}$ — whose expected behavior across the whole probability distribution is probably approximately correct (PAC) [25].

Whenever the training set follows a different representation of the data the model will be tested on, we say there exists a *domain shift* at test time. In such setting, theoretical guarantees no longer hold and a new formulation is needed to study the problem. The concept has gained a lot of interest in all machine learning fields as many real-world scenarios often constrain the learner to perform under a distributional shift. Indeed, data at test time may:

- **be highly dynamic**: when the environment significantly changes over time, the training set may soon become outdated. In such cases, a domain shift at test time occurs by means of novel data following slightly different distributions. For instance, building a classifier for testing patients against COVID-19 from data gathered in May 2020, would likely under-perform in half a year from there given the fast spread of the pandemic and the profound change in the marginal distribution of positive cases. In the same way, lifespan predicting models may return inaccurate results as current society evolves to different statistics, or robotic policies may misbehave on changes in dynamics of the robotic arm due to wear and tear and temperature variations. Therefore, we wish to build models that can adapt to changes over time.

- **follow unobserved patterns**: many data distributions considered for real-world tasks are enormously large and impractical to be collected exhaustively. We may then expect that important slices of informative data under the same distribution will be missing from our training data. Say we wish to build an object detector for self-driving cars to be able to recognize their surroundings: if the model is solely trained on the streets of Helsinki, it will most likely generalize poorly to unseen cities with different appearances. A recent work in photorealism enhancement of synthetic images from videogames [55] demonstrated incredible state-of-the-art performances, yet showed limitations of training the model on the German landscape only, ending up in misrendering Californian mountains and other details. Similarly, image recognition systems for photos may fail to adapt to changes in lighting conditions or backgrounds, if the training dataset lacks such informative data.

- **be expensive to collect**: as in the case of robotic manipulation tasks, test data may in practice be prohibitively expensive to collect, or even dangerous. This usually translates to using synthetic datasets instead of real data, implicitly introducing a shift in the data distributions. In addition to dynamics discrepancy between real and simulated data for reinforcement learning applications, even a bias in the appearance of the simulator may cause the learned end-to-end robotic policy to fail in the real world [33].

- **present similarities to previously solved tasks**: a crucial part of machine learning nowadays lies in exploiting previously solved tasks for improving new models, instead of learning them from scratch. The intuition behind this approach stems from data often sharing similarities among distributions, both in structure and meaning. For example, an image classifier trained to recognize pictures of animals may be used as a starting point for distinguishing trucks by cars. Indeed, the model may have learned to extract important global features in the data structure such as shapes and colors, so that new tasks can be solved more efficiently, faster and with limited data needs. Moreover, data may be different in structure but similar in meaning, hence informative to carry out the same task on a different data distribution: a computer vision model trained to recognize photos of dogs, may be deployed on drawings and sketches of dogs, with significant different appearance. Other important applications of transferring knowledge from one domain to another are seen in *Natural Language Processing* (NLP), where language models learned on heterogeneous large online corpora may be applied for speech recognition tasks of radio broadcasting programs [56], despite the significant difference in language style. Likewise, in reinforcement learning we can opt to provide a better prior to the learning algorithm by initializing the value function with that of similar tasks previously solved [57].

Given the above real-world scenarios, the need of successful techniques for dealing with domain shifts is rather important, both when training and test data intrinsically differ and when out-of-distribution data wants to be exploited to improve the task at hand.

2.2.1 Transfer Learning

We define *Transfer Learning* (TL) as the group of techniques in machine learning which involve helping models deal with domain shifts. This may come in the form of exploiting previously solved tasks to improve performance on a new task, or addressing the problem of solving the same task under uncertainties or distributional discrepancies in training and test data. Humans themselves have been proven to exploit previous knowledge obtained in similar tasks to learn faster and more efficiently, according to several psychological studies [58, 59]. Therefore, the motivation to apply such concept in machine learning should now be clear. Intuitively, we wish to design methods that take advantage of data collected from a source domain and task, to learn a target task whose data lies in a possibly different target domain. More technically, we aim to bias the learning process of the target task on reasonably good hypotheses.

It may be hard to give a global and exhaustive definition of the concepts of *domain* and *task* which include all possible frameworks in the machine learning field. Rather, multiple surveys have addressed the study of transfer learning for different paradigms, with slight different notations and definitions. In order to introduce the formulation of transfer learning in our reinforcement learning case of interest, we find helpful to provide insights on the better-established version of the problem in the supervised learning case, first.

Transfer learning in supervised learning The outstanding survey published by Pan and Yang [60] soon became the main point of reference in literature for the definition and categorization of transfer learning settings in supervised learning (SL). The notations and definitions in this paragraph have then been inspired by the same authors. Let us introduce the notions of domains and tasks when dealing with learning algorithms under distributional shifts in SL setting.

- A **domain** $\mathcal{D}$ represents a certain data structure and probability distribution over it. Thus, it is defined by a *feature space* $\mathcal{X}$, indicating the space of input data, and a marginal probability distribution $p(\boldsymbol{x})$, where $\boldsymbol{x} = \{x_1, \dots, x_d\} \in \mathcal{X} \subseteq \mathbb{R}^d$. For instance, in the case of age prediction, the underlying domain considered may have a feature space $\mathcal{X}$ consisting of general characteristics of each individual, and a theoretical underlying probability $p(\boldsymbol{x})$ describing the true distribution of ages in the community or society of interest.

- A **task** $\mathcal{T}$ describes the learning goal for a domain $\mathcal{D} = \{\mathcal{X}, p(\boldsymbol{x})\}$. Therefore, it consists of the label space $\mathcal{Y}$ and a theoretical objective predictive function $f : \mathcal{X} \to \mathcal{Y}$, which is used to predict the label $y \in \mathcal{Y}$ for each test instance $\boldsymbol{x}$. More generally, the objective predictive function may be thought of a probability distribution over the label space, as $p(y|\boldsymbol{x})$. In practice, such function is not known and needs to be learned from a set of observed data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, referred to as the *training dataset*. In the previous example of age prediction, the label space $\mathcal{Y}$ of a task $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ consists of positive integer values, while $f(\cdot)$ describes the learned function mapping each individual to their age.

Whenever a domain shift of some kind is present we can thus model data as coming from different domains, e.g. the set of training samples is assumed to belong to a source domain $\mathcal{D}_S$ while the model is deployed on a different target domain $\mathcal{D}_T$. In particular, note how two domains may differ by data structure, probability distribution over the data, or both at the same time. Finally, the transfer learning problem can be formally defined in its most general formulation:

Definition 2.2.1 (Transfer learning in SL) *Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$ [60].*

The above definition includes the possibility to have either domain discrepancies at training and test time, or differences in the task to be performed. The former case refers to settings where the data structure or probability distribution changes at test time while the objective of interest remains the same, such as when training an image classifier on photos of animals and aiming to recognize drawings or sketches of them. This setting is commonly noted in literature as *Domain adaptation*, and may be addressed by learning better data representations for the target domain [61]. On the other hand, when data of interest presents similarities with datasets of previously solved tasks, we can exploit the source domain to learn a new target task more efficiently, usually by providing a much better initial prior to the learning algorithm.

More technically, the authors in [60] categorize transfer learning into three broad subsettings, depending on the different situations considered: *inductive*, *transductive* or *unsupervised* transfer learning.

Overall, while the use and theory of transfer learning in SL is well established, different tools are needed to deal with the reinforcement learning case to account for the difference in paradigm. For this reason, several other surveys have been published in the attempt to give a clear definition for transfer learning methods in RL [62, 63] and deep RL [64].

Transfer learning in reinforcement learning Unlike supervised learning, reinforcement learning involves several elements in the learning process besides the space of input and output variables, such as the state-transition probability function and reward function. Therefore, transfer learning settings need to take into account all situations where any of these may differ in real applications. The work by Alessandro Lazaric [63] is the most complete yet recent survey on TL for the reinforcement learning framework. The author categorizes all transfer scenarios into three main subsettings, depending on the similarities and differences between the source and target tasks. Here, the concept of tasks differs from the supervised learning case, as it includes the full underlying MDP which models the RL problem at hand. In particular, [63] defines:

- A **task** M as the MDP described by the tuple $(\mathcal{S}, A, T, r)$, where $\mathcal{S}$ and $\mathcal{A}$ are the usual state and action spaces, T is the state-transition probability function — often abbreviated to *transition function* or simply *dynamics* — and r the reward function. The initial state distribution ρ_0 is now assumed to be embedded in T.

- A **domain** D as the joint space $\mathcal{S} \times \mathcal{A}$, now strictly depending on the MDP of the task M considered.

To model transfer learning settings, we can thus consider a space of tasks $\mathscr{M}$ and a probability distribution Ω over the space of tasks. Therefore, we define the *environment* $\mathscr{E} = \{\mathscr{M}, \Omega\}$, which uniquely identifies the setting of the transfer problem considered[1]. In practice, the training process only has access to a finite number of tasks drawn from $\mathscr{M}$ (named the *source tasks*), and we wish to exploit the knowledge retained when solving such tasks to generalize across the *target task* in $\mathscr{M}$. The target task is the final MDP we wish to deploy our learned policy on. Following this naming convention, we can then give a general definition of transfer learning for reinforcement learning, according to [63]:

Definition 2.2.2 (Transfer learning in RL) *Let $\mathscr{K}_S^L$ be the knowledge collected from L source tasks $\{M_i, \dots, M_L\}$ drawn from $\mathscr{M}$ according to Ω. Let $\mathscr{K}_T$ be the knowledge available (if any) from the target task. A transfer learning algorithm is the result of a "transfer of knowledge" phase, defined as*

$$\mathscr{A}_{transfer} : \mathscr{K}_S^L \times \mathscr{K}_T \to \mathscr{K}_{transfer} \tag{2.11}$$

[1] Note how the term *environment* now has a different meaning than in standard RL literature and should not be confused.

where the learner extracts useful knowledge for the transfer, and a "learning" phase, where the extracted knowledge is used to learn the target task:

$$\mathscr{A}_{learn} : \mathscr{K}_{transfer} \times \mathscr{K}_T \to \mathscr{H} \tag{2.12}$$

with $\mathscr{H}$ the space of final hypotheses that can be returned (e.g. a space of policies).

Given the above definition, we can then identify three main subsettings of transfer learning in RL:

(I) *Transfer from a source task to a target task, with fixed domain.* In this setting, the domain $\mathcal{S} \times \mathcal{A}$ is held fixed while the transition function T or reward function r is allowed to vary. The context of a transfer from a single source task $M_S = (\mathcal{S}, \mathcal{A}, T_S, r_S)$ to a target task $M_T = (\mathcal{S}, \mathcal{A}, T_T, r_T)$ is considered. Therefore, the space of tasks is described by $\mathscr{M} = \{M_S, M_T\}$, and Ω simply provides M_S to $\mathscr{A}_{transfer}$ for collecting knowledge in the source task, and M_T to $\mathscr{A}_{learn}$ when dealing with the target task. This can be thought of as the analogous case in RL of inductive transfer learning for SL. To give an example, suppose a physics simulator is used to learn a real robotic task: the two domains may be assumed to be the same but their dynamics likely present discrepancies due to imprecise simulated models. See Section 3.4.1 for an overview of transfer algorithms dealing with such issue.

(II) *Transfer across multiple tasks with fixed domain.* Similarly to the first case, the state and action spaces are assumed to be the same across all source and target tasks. However, a more general context is considered, where $\mathscr{E}$ is defined by a set of tasks $\mathscr{M}$ and a probability distribution Ω over them. In general, we are not restricted to have a single target task $M_T \in \mathscr{M}$, as multiple may exist. To follow up on the previous example, we can think of a parameterized physics simulator used to learn a policy for a real-world task: if the parameters of the simulator vary according to a higher-level distribution, then multiple tasks in simulation can be sampled arbitrarily. Intuitively, transfer algorithms in this setting rely on gaining knowledge of the target task while learning on a finite number of realizations of Ω, similarly to classification and regression. See Section 3.4.2 for the formalization of *domain randomization*, the popular technique in literature used by DROPO, as a particular case of transfer across multiple tasks with fixed domain.

(III) *Transfer across tasks with different domains.* Lastly, we consider the setting were the state space $\mathcal{S}$ and action space $\mathcal{A}$ are also allowed to vary across tasks, in addition to dynamics and rewards. Therefore, in general $\mathscr{M}$ consists of source tasks of the form $(S_S^i, \mathcal{A}_S^i, T_S^i, r_S^i)$ and target tasks $(S_T^i, \mathcal{A}_T^i, T_T^i, r_T^i)$. Again, we can think of a sim-to-real transfer task as an example. This time, we additionally introduce a domain discrepancy when we consider end-to-end

learning from images: the appearance of the simulator may not represent the same look of real world scene. Domain randomization over textures and colors may then be applied as a method to shorten the gap [33].

Many algorithms that aim to tackle the above three settings have been presented in literature. Generally speaking, they can be grouped under the type of knowledge transferred, and the objective pursued. More specifically, algorithms may attempt to transfer data samples from source tasks to increase the amount of data available, provide a set of initial parameters for learner $\mathscr{A}_{learn}$, or finally transfer the full solution representation learned on the source task(s). For each transfer type, different evaluation metrics can ultimately be used to assess the transfer method, depending on our main goal when learning the target task: learning speed, asymptotic performances, jumpstart improvement.

For instance, a work on the Nao humanoid robot has shown benefits when transferring action value functions from previously learned similar tasks as a starting point for learning the new task [57]; Sherstov and Stone [65] introduced random task perturbations in source domains to reduce the set of possible actions and possibly speed up the learning process for the target task; Phillips [66] analyzed worst-case performance loss when a policy is directly transferred to the target task; Lazaric et al. [67] tried to augment training data with source instances which are most likely to occur in the target MDP according to an estimate of the target dynamics;

In conclusion, the sim-to-real setting in robot learning is the transfer of interest for the particular work in this thesis. In such setting, the physics simulator allows for arbitrary generation of source tasks, while knowledge and data from the target task (i.e. the real world setup) is costly to obtain. The sim-to-real transfer challenge is thoroughly described and motivated in Chapter 3.

2.2.2 Transfer types

In addition to the formal definition of transfer learning, it may be helpful to revise the common terms in literature when referring to different types of transfers. In particular, we highlight the popular concepts of *Zero-shot transfer* and *Few-shot transfer*.

Zero-shot transfer generally refers to a machine learning problem in a transfer setting of some kind where no data or knowledge from the target task is available. More specifically, this would translate in having an empty $\mathscr{K}_T$ in equations 2.11 and 2.12, for the reinforcement learning paradigm. In robot learning literature, zero-shot transfer often consists in training policies entirely in simulation and deploying them to the real world with no further adaptation or finetuning. This approach clearly comes with the advantage of avoiding all challenges related to learning on real hardware.

However, real-world applications often demonstrate benefits when some form of knowledge $\mathscr{K}_T$ from the target task is acquired and exploited. Such cases fall in the Few-shot transfer regime, which describes situations where *limited* data can be collected from the real setup. This data can thus be used to either augment transferable knowledge (Eq. 2.11) or to directly impact the learning algorithm (Eq. 2.12). Notice how data from the target task is either way assumed to be minimal, otherwise a transfer learning method would not be needed in the first place.

2.3 Gradient-free optimization

Optimization theory is a crucial building block of machine learning. Indeed, despite all sorts of differences among them, at the heart of each leaning algorithm lies the solution of an optimization problem. The structure of the problem may then vary, based on the loss function designed and the paradigm considered. For instance, standard linear regression algorithms perform maximum-likelihood estimation of the predictors in a closed-form solution, function approximation classification tasks perform minibatch minimization of the loss function through backpropagation in neural networks, or shallow classification tasks with Support Vector Machines aim to minimize the hinge loss function through quadratic programming.

In general, whenever closed-form solutions may not be found for the optimization problem at hand, we're left with two options: *gradient-based optimization*, based on the iteratively update of parameters towards the direction of greater ascent (or descent), or *gradient-free optimization*. While the former often excels at finding local minima for convex problems in high-dimensional and non-linear settings, it may fall short when dealing with noisy and discontinuous functions. Furthermore, the underlying objective function of interest may not be explicitly known, in which case the computation of the gradient can only be approximated through finite-differences.

Gradient-free optimization methods come into play to overcome these challenges. They are often easy to apply and able to find good solutions out of the box. In contrast, they do not necessarily converge to the true global minima. In practice, such methods approach parameter search with some combination of random and statistical components. Popular gradient-free optimization algorithms include Bayesian Optimization, Evolutionary strategies, Simulated Annealing or simple Random Search.

Throughout the work in this thesis, the physics simulator used for robot learning is assumed to be non-differentiable, i.e. its forward dynamics are unknown to the algorithm. Therefore, optimization is performed with the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [68].

2.3.1 CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is part of the broader group of evolutionary algorithms, a class of population-based metaheuristic optimization algorithms. In particular, Evolution Strategies (ES) rely on the idea of evolution to search the parameter space in the optimization problem. They generally adopt a *mutation* and *selection* operator to evolve the population. More specifically, the mutation operator is used to generate offspring from the current population in the attempt to further explore the search space, while the selection operator takes care of selecting a subset of the best individuals in the current population for later mutations. Intuitively, the process is repeated until termination.

CMA-ES models the current population as a realization of a multivariate Gaussian distribution in the parameter space, so that mutation can take place by sampling from the current distribution, given a mean μ and covariance matrix Σ. Selection is performed by shifting the mean of the distribution towards a more favorable neighborhood, by taking a weighted average of the K-best candidates in the current population of size n. In addition, CMA-ES performs a self-adaptation step on the covariance matrix by updating σ and C at each iteration, where $\Sigma = \sigma^2 \cdot C$. This step aims at adapting the "overall" standard deviation σ — also known as the *step size* — and the covariance matrix C in a maximum-likelihood fashion to fit the objective function contour lines (see Fig. 1 in [68]) and promote variance increase in promising directions.

A simple overview of CMA-ES is summarized in Algorithm 1, involving the optimization of a generic fitness function $f(\cdot)$ in an input space of individuals $x \in \mathcal{X}$. For a detailed explanation on how the updates are performed, we encourage reading the in-depth review by Nikolaus Hansen [68].

Algorithm 1: CMA-ES

Result: μ or $argmax(f(x_i))$

Set K, n;
Initialize μ, σ, $C = \mathbf{I}$;
while *not terminate* **do**
    // Mutation;
    for i *in* $\{1, \ldots, n\}$ **do**
        Sample $x_i \sim \mathcal{N}(\mu, \sigma^2 C)$;
        Evalute candidate solution $f_i = \text{fitness}(x_i)$;
    end
    // Selection;
    $\mu \leftarrow$ update mean;
    // Self-adaptation;
    $\sigma \leftarrow$ update step size;
    $C \leftarrow$ update covariance matrix;
end

Chapter 3

The sim-to-real transfer challenge

This chapter presents and justifies the use of physics simulators in solving robotic tasks, as well as their challenges when it comes to transferring learned behaviors to the real world. In particular, a more thorough introduction to reinforcement learning in the context of robotics is given in the first section. Later, a detailed motivation on the existence of the *reality gap* is reported, the main factor hindering a straightforward transfer. Finally, the common approaches for dealing with the sim-to-real transfer paradigm are presented, including *domain randomization*, the technique we build upon in our method and experiments.

3.1 Reinforcement Learning in Robotics

Whenever a real-world task involves autonomous sequential decision making, the problem may be naturally tackled with reinforcement learning. It is indeed straightforward to apply the abstraction of the RL setting presented in Section 2.1 to real-world problems. In particular, the field of robotics opens great possibilities for applications of data-driven algorithms in the attempt of learning flexible and autonomous robots. Suppose we are given a robotic arm and wish to learn a policy to successfully push an object placed on a table to a target position. Here, available information to the agent (the robot) may include the pose of the object (i.e. its position, velocity, orientation, and angular velocity) as well as internal kinematic properties of the arm. More precisely, robotic arms are often modelled as rigid objects consisting of several *joints*, which describe the number of *degrees of freedom* (DoF) of the considered arm. Exploiting knowledge of the joints position and velocity and pose of the object, the agent in question may then have a complete

statistic to make a decision towards the final goal. The space of actions $\mathcal{A}$ may consist in direct torque commands to each joint, or higher-level target joint position commands when the arm motion is guided through closed-loop controllers (e.g. PID controllers). Finally, a reward signal $r(\cdot)$ as feedback to the robot's behavior needs to be designed. Common choices aim to reward a closer distance from the end-effector — the gripper or tool at the end of the arm — to the object, as well as proximity of the object to the target goal. In addition, multiple terms may be added to penalize potentially dangerous contacts with the table, encourage lower energy consumption or avoid instabilities and unfeasible situations (e.g. exceeding joint limits). Analogously, the RL paradigm may be applied in tasks other than robotic manipulation, such as navigation [69, 70] or locomotion [71, 72].

In general, approaching robot learning with data-driven techniques would allow the robot to autonomously learn an optimal control policy, without the need of manually detailing proper behavior. Moreover, self-taught robots may develop generalizable capabilities to perform under unseen situations, a complex problem to tackle when addressing robot learning with traditional direct programming.

These motivations led to a strong interest from researchers in the fields of robotics to apply RL on robotic tasks. A number of great examples exist of problems in robot learning successfully tackled with RL-based methods: a door opening task [45], a t-shirt folding problem [73], optimal helicopter control [74], learning pool strokes [75], a peg-in-the-hole insertion task [43], a drawer opening task [43] or a ball-in-a-cup task [44]. A remarkable recent success story by OpenAI [22] even shows a robotic hand learning successful dexterous in-hand manipulation behavior for object reorientation.

However, all the previously mentioned promising works in the field rely on novel complex techniques to apply RL in robotic tasks. Indeed, while a straightforward mathematical formulation of the problem can be made, a direct implementation of reinforcement learning methods in robotics encounters several challenges in practice. The brilliant survey by Kober et al. [6] highlights the most important problems faced when applying reinforcement learning methods in the field of robotics. We take inspiration by this work and report the three major challenges that roboticists have to deal with:

- **Curse of dimensionality** The underlying MDP that models reinforcement learning problems in robotics usually involves continuous state and action spaces, given the nature of the information the environment provides and the possible commands for interaction. However, it's well proven that even for discrete high-dimensional cases an exponential growth of state-action pairs makes it impractical for RL algorithms to cover the whole space $\mathcal{S} \times \mathcal{A}$. This phenomenon is noted in literature as the *Curse of dimensionality* [76]. When it comes to continuous spaces, algorithms have to involve function approximators to estimate value functions and action probabilities. Therefore,

it gets even harder to collect informative data throughout the whole state-action space to reach optimal control. To give an example, robotic arms often come with 7 degrees of freedom (see Fig. 3.1), which already translate to 14 dimensions in the state vector (due to both joint angles and joint angular velocities) and 7 actuator dimensions. Hence, state vectors may easily consist of 20-30 real-valued dimensions with potential objects and other information involved in the scene. RL strategies need to cope with these high-dimensional continuous spaces assuming that no exhaustive exploration can be performed.

- **Curse of real-world samples** Although the main idea of reinforcement learning relies on learning from experience by trial and error, having the robot directly interact with the environment often turns out to be problematic. Real hardware usually needs substantial human supervision, periodic maintenance and suffers from wear and tear. In addition, robots would aim to learn optimal policies by randomly acting in the real world, i.e. they would inherently need to misbehave in order to learn how to behave. This clearly comes with open safety concerns for the surrounding environment and the robot itself, which may not be acceptable given the high costs of robotic hardware. Moreover, data collection in the real world is by itself time-consuming, as even resetting the environment to its initial state comes with human labor and supervision costs. On top of this, even when experience is gathered in the form of trajectories on the real environment, physical limitations in the sensors and actuator systems in use will result in noisy and delayed observations. Lastly, the real setup is open to changes in dynamics over time due to natural changes in the environment and external factors. For instance, wear and tear of the robot or temperature conditions of the room may affect the environment dynamics, other than small perturbations in the setup over longer periods of time. Indeed, comparing algorithms and reproducing experiments faithfully is by itself challenging. In conclusion, gathering experience is a tedious and expensive process in robotics. RL algorithms need to be data-efficient and cope with noise and delayed sensors in a real-time sequential decision making scenario, besides considering safety constraints.

- **Curse of goal specification** Reinforcement learning comes with the advantage of avoiding explicit programming of the task at hand. Instead, we simply need to provide a reward signal that the agent wishes to maximize. In practice, however, the particular design of the reward function may highly impact the training time of the agent or even its asymptotic performance, given convergence to local optima only. Take a robotic task as an example: if a hockey player robot is tasked to score a goal, we may intuitively provide positive feedback on goal achievement; the robot, however, may never end up achieving the goal while exploring the state-action space. This type of isolated reward signal is known in literature as *sparse reward* signal. More

complex reward functions may be designed to provide intermediate feedback to the agent for each action taken, often requiring significant manual engineering and back-and-forth tuning. Guiding the learning process in such manner is denoted as *reward shaping* [77]. Furthermore, reward computation in the real world may be problematic to compute due to partial information on the environment (e.g. unknown contact forces, distances among untracked objects and noisy observations), and is at times assumed to be unavailable.

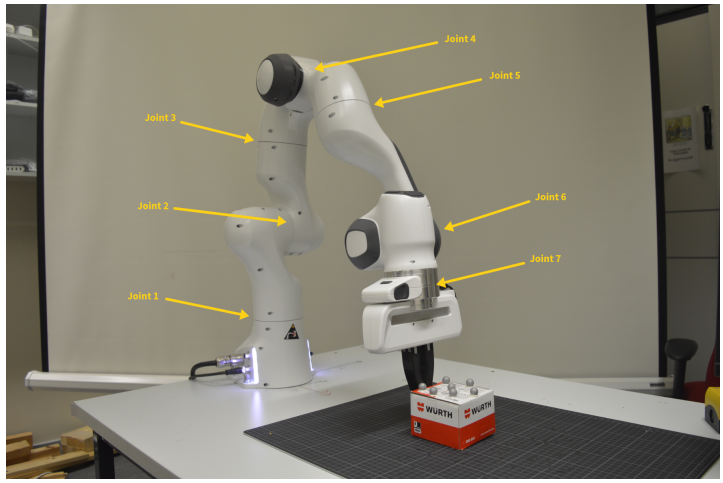


Figure 3.1: Positions of the 7 degrees of freedom (joints) in the Panda Franka robotic arm. Each joint can exert an arbitrary torque allowing the arm to move around and interact with the environment, e.g. push the box on the table.

Multiple works reviewed and analyzed in details the aforementioned issues in reinforcement learning for robotics [6, 78, 79, 80]. Researchers in the field had to find ways to work around these challenges in order to successfully train RL policies for real-world robotic tasks. We propose a categorization of the works in literature addressing the presented issues under three broad groups, depending on the main approach taken:

- **Cautious exploration** This approach heavily relies on making data-efficient algorithms or adding safety constraints when learning optimal control. Often, model-based reinforcement learning methods are used to make the most out of the limited data available and avoid the expensive direct interaction with the environment [81]. Hester and Stone [13] implement a model-based data-efficient method to discourage exploring directions that do not look promising. Schneider J. [82] proposes to exploit uncertainty in the learned

model to intelligently predict expected returns and allow safe control. Similarly, Deisenroth and Rasmussen [83] exploit model uncertainties for efficient long-term planning. The same authors further introduce a collision-avoidance technique in a box-stacking task [14]. Berkenkamp et al. [15] analyze safety-critical systems and impose stability constraints while learning to preserve safety. Dalal et al. [84] introduce a safety layer with predefined constraints that are never violated during training. A detailed description of recent advances in safe reinforcement learning algorithms is finally reported by Brunke et al. [5]

- **Learning from demonstrations** An alternative approach commonly adopted in literature is the use of expert demonstrations to teach the task or provide high-quality prior knowledge. This technique is motivated by the often easy way for humans to provide expert data samples representative of proper behavior. This approach is also popularly known as *imitation learning* or *programming by demonstrations*, and doesn't necessarily have to involve reinforcement learning policy update steps. Argall et al. [11] and Billard et al. [12] provide great surveys on the general framework of learning from demonstrations. To get a clearer idea, a few success stories are reported: Abbeel et al. [85] propose a learning setting which entirely avoids exploration by providing teacher demonstrations, and later deploy a similar method to learn helicopter optimal control [74]. Vecerik et al. [86] demonstrate a fast solution to insertion tasks from raw images with a small number of human demonstrations. Schaal S.[10] provides demonstrations to a pole balancing robot which learns the task in one single trial. Finally, notice how works may occasionaly rely on human demonstrations to extract additional knowledge from the target environment rather than directly learning from it [45]. In particular, the novel method DROPO introduced in this thesis exploits manual guidance of the real robot to infer the environment dynamics, but later uses a model-free reinforcement learning algorithm to learn the task.

- **Learning in simulated environments** Lastly, researchers may overcome costly data collection and safety concerns by learning in simulation. Physics simulators offer great advantages even in terms of faster training times with parallel computation and provide complete information on the configuration of the scene. For example, the reward function may be designed to penalize dangerous contacts or take into account complex noisy-free kinematics information in real-time (orientations and accelerations). A large number of works have implemented techniques to learn RL policies solely in simulation and directly deploy the behavior to the real world. This approach is referred to as the *sim-to-real transfer* paradigm. Antonova et al. [21] demonstrated successful transfer of a pivoting task learned in simulation. OpenAI [19]

solved the Rubik's cube with a robotic hand by learning entirely on synthetic data. Tan et al. [32] adopted simulators to learn locomotion gaits in quadruped robot. As physics simulators became more sophisticated and capable of modeling photorealistic scenes, simulation-based deep reinforcement learning techniques have also been proposed in several fields spanning autonomous driving [87], robotic manipulation [88, 89] and locomotion [71, 72]. We Finally, note that a further adaptation step in the real-world may still be stacked on top of sim-to-real transfer methods to ultimately tune the policy behavior on the target domain, as in the case of [90, 91].

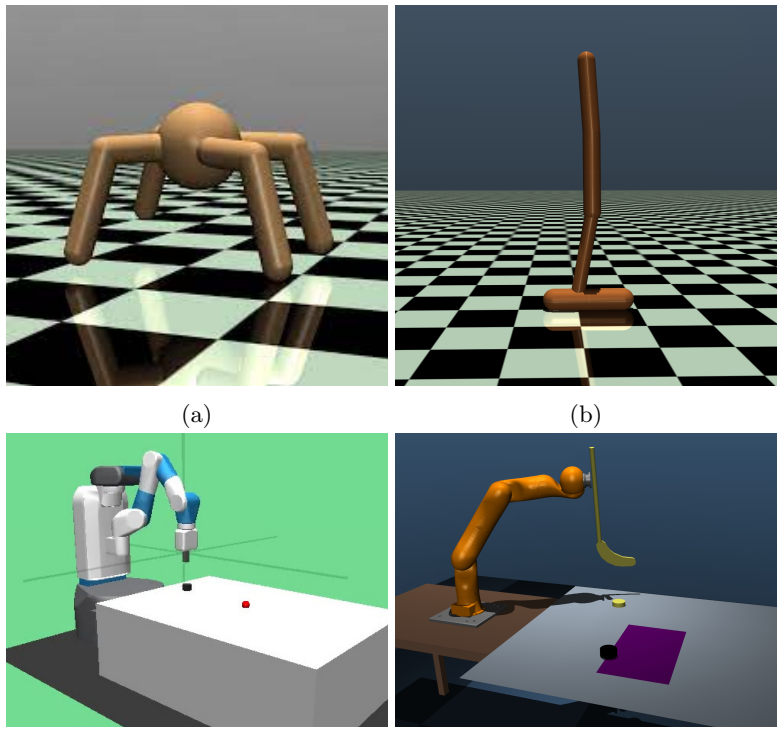
As the work in this thesis deals with the sim-to-real transfer setting, we are mostly concerned with methods that learn in simulation and deploy their knowledge on the real world. Despite the promising motivations, such approaches yet come with several challenges, due to the imprecise fidelity of physics simulators. Indeed, sim-to-real transfer methods experience a transfer learning problem where a shift in distribution from training to test data is present. This shift is caused by multiple limitations of simulators and external factors in the real world, and it's generally known as *model bias* or *reality gap* (See Section 3.3). A recent paper by Höfer et al. [92] reports the current expert perspectives on sim-to-real transfer for robotics and highlights the importance of further research in the area.

3.2 Simulators for robotic tasks

To better understand why transferring policies learned on a physics simulator is notoriously challenging, a brief introduction of existing simulators and their features needs to be given. Collins et al. [93] provide a recent in-depth survey on actively adopted physics simulators for robotics research. The most popular platforms include PyBullet, Gazebo, Nvidia Flex and MuJoCo. The latter, in particular, is the simulator of choice throughout the work in this thesis.

Physics simulators allow to construct virtual scenes that closely resemble the real-world. Fig. 3.2 illustrates examples of simulated scenes for the training of robot policies. Physics engines usually come with a large number of parameters to model the environment at hand, including contact-specific parameters, visual appearances, physical properties (e.g. masses, friction coefficients, size of the objects, joint dampings) and options for forward dynamics solving and numerical integration. Intuitively, such a great flexibility also translates to a more complex process when choosing the correct parameters, especially when real-world behavior needs to be mimicked.

The MuJoCo physics engine [95], which stands for **Mu**lti-**Jo**int dynamics with **Co**ntact, has been widely adopted to solve reinforcement learning tasks in simulation, with impressive successful examples [19, 22].



(a) (b)

(c) (d)

Figure 3.2: Examples of robotic tasks modelled with the MuJoCo physics engine: (a) the Ant, (b) Hopper and (c) FetchSlide Gym environments by OpenAI [94] are shown, together with the hockeypuck environment (d) used in our experiments (see Chapter 6).

MuJoCo, like other physics simulators, rely on approximate solvers for computing the environment responses — formally known as *forward dynamics* — due to the prohibitive computational complexity of frictional contact-rich optimization problems. In particular, notice that contact forces are generated by modelling object interactions as non-linear mass-spring-damper systems. Here, the non-linearity arises due to a penetration-dependent impedance function $d(r)$ which scales stiffness and damping as the objects get closer.

Overall, the procedure followed by MuJoCo for computing forward dynamics consists of two phases:

1. **Solver phase** The first phase involves the computation of all kinematic and dynamic properties in the scene needed to solve Eqs. 1a-b-c in [95]. More

specifically, actuator forces and contact constraints are taken into account to return the joint accelerations of the robotic arm and the accelerations of the other simulated bodies.

2. **Integration phase** Given the computed accelerations, the engine then integrates the values one timestep forward returning positions and velocities, i.e. the next state s_{t+1} in RL terms.

Finally, note that physics simulators also allow an arbitrary accurate reset of the environment whenever needed. Likewise, they can be queried in particular configurations of the scene at any point in time.

3.3 The reality gap

Many works have by now highlighted the existence of the so called *reality gap* [16, 96, 32, 33, 17]. This phenomenon lies at the heart of the sim-to-real transfer challenge and refers to all discrepancies and distributional shifts separating a simulated model from the real world. In 1999 [96], Brooks stated:

> There is a real danger (in fact, a near certainty) that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing and actuation — it is very hard to simulate the actual dynamics of the real world.

Intuitively, training policies on imprecise models may indeed end up degrading performances in the target environment. Whenever we opt for the sim-to-real paradigm we must then face a problem of transfer learning, where the source and target tasks — according to the definition given for RL in Section 2.2.1 — generally differ, either by dynamics T, domain $\mathcal{D}$ or both. Based on previous works, we identified a list of factors that induce the existence of the reality gap:

- **Modeling approximations** As analyzed in the previous section, physics simulators are forced to solve forward dynamics with approximate models. This, however, introduces a gap between simulated dynamics T_S and the target unknown real-world dynamics T_T, especially when modelling contact dynamics. Kober et al. [6] and Tan et al. [32] report that even small inaccuracies may add up over time and make simulators diverge from the real-world, particularly in the case of stability-critical tasks (e.g. a pole balancing task). Finally, numerical integration errors may also contribute to increase the gap.

- **Parameters discrepancy** While simulators provide high flexibility, they often turn out to be tedious and complex to tune. In order for a model of the scene to be fully defined, each parameter in the simulator must be specified including physical properties and solver-specific options. While some

parameters may be actively measured (e.g. masses and sizes) others may be arguably impractical to compute in reality, such as joint damping, friction coefficients, or damping and stiffness parameters for contacts. Moreover, due to discrepancies in forward models, even fitting simulators with exact measured parameter values would not in general guarantee optimal fidelity. In practice, this complexity translates to a time-consuming manual trial and error to get realistically behaving simulators [23].

- **Unmodeled phenomena** Even in terms of phenomena actually modeled, simulators can not get as accurate as the real-world. Real hardware is in practice faced with effects that are hard to model in simulation, such as sensor noise and delay. Indeed, roboticists must always assume that real systems are affected by limitations of digital components and that current information of the environment is noisy and delayed in time by a variable step. In addition, phenomena such as air drag may be missing in the simulation completely. A poor visual appearance of the simulator may also be included in this section as an under-modeled phenomenon, as it introduces a gap in source and target domains. Indeed, end-to-end learning approaches from raw images likely suffer when transferring from visually inaccurate scenes [33].

- **Non-stationarity of real environments** A final factor motivating the existence of the reality gap is the ever-changing nature of real environments. Real robots are subject to wear and tear and may need to be stopped after extensive periods of use, suggesting non-constant behavior across their lifetimes. Moreover, the real world is often open to small perturbations over time such as object deformation and degradation, and may present non-stationary external effects (e.g. temperature and humidity changes). Finally note that the learning process in a constantly changing environment may not even fully converge, as a *tracking* solution would be needed [97].

An overview of the identified challenges is illustrated in Fig. 3.3. To make things worse, numerical errors may also play a significant role in increasing the gap. The aforementioned issues can therefore be grouped as the driving factors of the *sim-to-real transfer challenge*. To advance robotic research in this direction, algorithms need to cope with all of the above difficulties and lead to policies which are robust and generalizable (i.e. policies that do not overfit on training data). Moreover, a careful choice of the physics simulator is also important, as these can affect real-world policy performance differently in robotic manipulation [98].

Kadian et al. [99] recently attempted to quantitatively predict the degree of performance loss after the transfer, and showed that a proper tuning of the simulator may increase the confidence in the prediction. In general, most research in the sim-to-real transfer field has focused on developing intelligent automatic processes

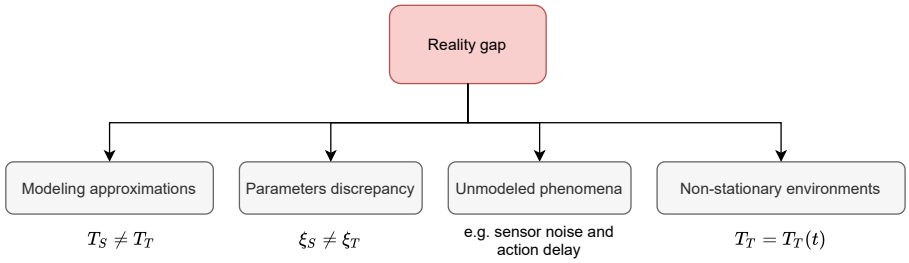


Figure 3.3: Overview on the factors inducing the reality gap. The functions T_S and T_T indicate the transition probability respectively for the source (simulation) and target (real world) tasks. ξ denotes the physical parameters of the scene (e.g. masses, friction coefficients). While all of them eventually contribute to a discrepancy in source and target transition probability functions, we highlight the difference between modeling approximations and lack of modeling (unmodeled phenomena), as well as the issue of parameter uncertainty and non-stationarity of real environments.

for shortening the gap with minimal manual engineering effort (e.g. repetitive tuning).

3.4 Common approaches

A recent work by Valassakis et al. [23] analyses the sim-to-real transfer challenge in detail and presents an objective evaluation of common techniques adopted to cross the reality gap. Such techniques often involve automatic tuning (*System Identification*) or randomization (*Domain Randomization*) of simulated parameters, or explicit injection of noise [33, 31] and external forces [100] while training. The former two approaches make up most of the sim-to-real transfer literature and are later defined.

3.4.1 System identification

Broadly speaking, the *system identification* (SI) field includes all techniques for building accurate mathematical models from experimental data. In the particular case of the sim-to-real transfer paradigm, the term more closely refers to the act of tuning models (i.e. physics simulators) to bring them closer to the real world, therefore bridging the reality gap. While system identification generally includes carefully measuring real parameters and manual trial-and-error as valid approaches,

most literature focuses on applying machine learning methods to perform an automatic tuning of the simulator based on real-world data. However, how the data should be collected and how the parameters of the simulators should be tuned accordingly are open questions.

We may present the system identification approach in robotics in terms of a transfer learning method. Recall the notation in Section 2.2.1 for the RL case. SI deals with the transfer from a single source task — the MDP induced by the physics simulator — to a target task, i.e. the real world. The target knowledge $\mathscr{K}_T$ usually comes in the form of a limited set of target data, used to infer simulation parameters and additional knowledge $\mathscr{K}_{transfer}$. This knowledge is ultimately used to learn the target task, e.g. by simply training a policy with the tuned simulator and directly deploy it in the real environment.

Let us introduce system identification in sim-to-real transfer for robotics with a simple example: a non-linear regression problem. Given real transitions $\{s_t, a_t, s_{t+1}\}_{t=0}^{T-1}$ collected offline, let $p_{sim}(\cdot|s, a;\ \xi)$ be the implicit forward dynamics of a physics simulator parameterized by vector $\xi \in \Xi$, representing the tunable parameters (e.g. masses, friction coefficients, object sizes, joint dampings). System identification may be performed by solving the following optimization problem:

$$\xi^* = \arg\min_{\xi} \sum_{t=1}^{n} \|s_{t+1} - s_{t+1}^{\xi}\|_2^2 \ , \quad s_{t+1}^{\xi} \sim p_{sim}(\cdot|s_t, a_t;\ \xi) \tag{3.1}$$

In practice, however, real data is noisy, physics simulators are often not differentiable and a simple euclidean distance may poorly perform on multi-dimensional state vectors which include heterogeneous information. Therefore, more advanced techniques in literature have been proposed.

Kolev et al [54] attempts to jointly optimize state observations from real sensors and physical parameters similarly to a SLAM problem setting, using a euclidean-like distance differently weighted on each dimension. Hanna and Stone [101] propose to optimize parameters to minimize a probabilistic distance measure between simulated and real observation distributions (e.g. Kullback-Leibler), iteratively collecting data from the real-world.

Other works involve the approximation of complex patterns through the training of neural networks: the authors in [102] and [103] attempt to predict the parameter difference between a source and target model and iteratively change the dynamics accordingly, after a pre-training step on synthetic data only. Golemo et al. [104], on the other hand, trained a recurrent neural network to predict the residual between source and target state observations, and later modified the policy training process to account for the discrepancy learned. Zeng et al. [105] attempted to infer control parameters from visual observations training deep neural networks.

Overall, we can argue that system identification alone may fail to cross the reality gap. As discussed in Section 3.3, multiple challenges need to be addressed

when dealing with the reality gap, and simply tuning a simulator may never fully capture real trends. For example, a particular set of physical parameters may generate data which faithfully represents the real world on a portion of the state-action space, while poorly performing elsewhere. Furthermore, system identification does not address the non-stationarity of real environments.

3.4.2 Domain randomization

An alternative approach for crossing the reality gap which has gained substantial traction in recent years is named *domain randomization* (DR). Put in simple terms, DR aims to achieve a successful transfer by randomizing the simulator parameters and learn a robust control policy over them. Intuitively, the new objective of the underlying reinforcement learning problem now takes into account the distribution of randomized parameters to learn optimal behavior. This formulation usually helps to address all four challenges presented in Section 3.3, as training on a ensemble of models potentially helps against non-stationary environments, unmodeled phenomena such as noise, and discrepancies in forward models. Moreover, the distribution of physical parameters may be tuned to reduce a pure parameter discrepancy with the real world.

Commonly randomized parameters are the masses of each link in the robot's body and object in the scene, friction coefficients of interacting objects, damping acting on the joints, gains of closed-loop controllers (e.g. joint position controllers) and size of the objects. The appearance of the simulator may also be randomized when learning in end-to-end fashion from visual observations [33, 34, 35], but inspecting this setting is out of the scope of this thesis.

Let us formalize the new optimization problem to be solved in the case of domain randomization. Let $p(\xi)$ be the distribution of the randomized simulator parameters ξ. We then wish to learn a policy π_{θ^*}, such that:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\xi \sim p(\xi)} \left[E_{\pi_\theta, \rho_0} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \right] \tag{3.2}$$

In practice, the implementation requires minimal effort: $p(\xi)$ is used to sample new environment dynamics at each training episode and the learning process is kept the same. $p(\xi)$ is then often parametrized as a uniform or a multivariate Gaussian distribution. Several recent works have shown successful sim-to-real transfer of robotic tasks using domain randomization for dynamics [20, 21, 22, 19, 32, 31, 30].

More formally, we may again view domain randomization as in instance of a transfer learning problem (see Section 2.2.1). This time, we investigate a transfer from a distribution of tasks $M \sim \Omega$ induced by $p(\xi)$, to the target real-world task M_T, which we assume belonging to the same space $\mathscr{M}$. Similarly to a supervised learning approach, by gaining some knowledge $\mathscr{K}_S^L$ from a finite number L of

realizations $M_S \sim \Omega$ we aim to learn a policy which generalizes well to the full distribution of tasks, including M_T. Throughout the work in this thesis, we assume that the domain $\mathcal{S} \times \mathcal{A}$ is held fixed across all tasks. Therefore, we only analyze domain randomization as a method to shorten gaps in dynamics rather than in data structures.

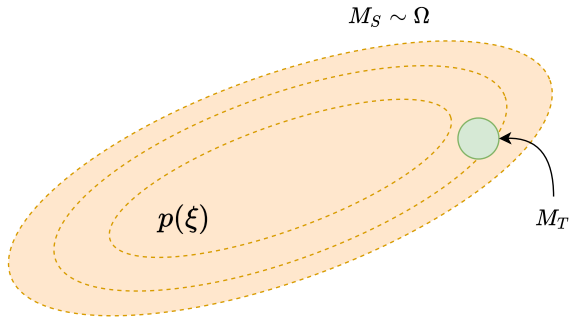


Figure 3.4: A domain randomization distribution $p(\xi)$ over physical parameters induces a distribution Ω over source tasks M_S. The goal with DR is to train a policy to perform well on a finite number of realizations of Ω, hoping to generalize to the unknown target real-world task M_T.

While domain randomization appears to effortlessly lead to robust policies, an important challenge still needs to be addressed to achieve a successful transfer. Namely, the choice of the exact distribution $p(\xi)$ to sample from turns out being a rather tough problem. Intuitively, training optimal policies on overly wide parameter ranges likely hinders the learning process or even make it impossible to find a single policy which solves the task on all variations of the environment. On the other hand, a distribution $p(\xi)$ in the form of a point-estimate would exhibit the same issues of system identification, with policies lacking generalizability. Therefore, how to pick domain randomization distributions remains an open research direction worth pursuing [106].

When manually engineered uniform distributions are picked, the approach is generally referred to as *uniform domain randomization* (UDR). UDR may be the result of prior task-specific knowledge or tedious back-and-forth tuning of $p(\xi)$ until the desired transfer is achieved. However, as the engineering effort associated with this process is relatively high [23], statistical methods and machine learning may be applied to optimize the distribution with minimal human intervention.

In this thesis, we introduce a novel method to automatically infer $p(\xi)$ based on limited and safely-collected real-world data (see Chapter 5 for a full description).

A number of related works which attempt to achieve a similar goal are reported in Chapter 4, highlighting their advantages and shortcomings.

Finally, note that DR may be also phrased as a *meta-learning* problem, as in the case of policies with recurrent structure (e.g. LSTM [107]). In particular, training an LSTM with domain randomization may lead to policies which learn to *adapt* their behavior based on the dynamics of the current environment, rather than finding a single robust behavior across all variations [20].

Chapter 4

Related works

This chapter aims to give an overview of all encountered works in literature that address the issue of intelligently designing domain randomization distributions in the context of environment dynamics. Domain randomization proved successful for several real-world robotic tasks in recent years, both when randomizing physical parameters [21, 20, 32, 30, 31], appearance of the simulator [33, 35, 34], or both [36, 19, 22].

However, as briefly mentioned in Section 3.4.2, the most common way to approach domain randomization is still by manually tuning the distribution of parameters with tedious trial and error, yielding substantial engineering effort. Therefore, we stress on the importance of finding more advanced methods to apply domain randomization for sim-to-real transfer of robotic tasks with minimal human intervention [106].

Early attempts by Rajeswaran et al. [42] demonstrated that adapting the DR distribution $p(\xi)$ over physical parameters ξ is a promising approach for efficient sim-to-real transfer, despite showing experiments in simulation only. The authors proposed to update the dynamics prior distribution in Bayesian fashion given a set of data from the target setup, hence tuning the simulator with a distribution that resembles the uncertainty over the real physical parameters. The method was able to identify target parameters in simulation and even train policies robust to model inaccuracies. Ramos et al. [108] approached the problem similarly, diving deeper into how to approximate the posterior $p(\xi|\mathbf{x})$ given a dataset of target observations $\mathbf{x}$. More specifically, they modeled the distribution as a Gaussian mixture model with diagonal covariance matrices and tested domain randomization with this posterior on a number of simulated OpenAI Gym environments [94]. However, as both previous methods do not show experiments on transfers to real robots, they do not explicitly deal with many of the challenges arising with RL for robot learning, such as data collection on real hardware and sensor noise (See

Section 3.1).

Differently, the approach by Chebotar et al. [43] (SimOpt) gained popularity for its promising framework applied in an actual sim-to-real transfer scenario. The authors proposed a euclidean-like distance as a way to measure discrepancies between simulated and real trajectories. Then, they chose to optimize a multivariate Gaussian distribution over the physical parameters of the simulator by iteratively collecting data from the target environment and minimizing the trajectory-based measure. At each iteration, a new policy is trained in simulation with the current randomization distribution and used to collect real data for the next step. We refer to this approach as *online-guided domain randomization*, as the distribution $p(\xi)$ is optimized with progressive real data collection while training.

Du et al. [36] investigated a similar online framework by jointly learning a policy and simulation parameters in the context of image-based RL, by randomizing both physical parameters and appearance of the simulator. They propose a Search Parameter Model (SPM) which predicts whether the given physical parameters are higher or lower than the true parameters based on observed data. At each iteration, new real-world data is collected and the training process goes on.

Muratore et al. [44] (BayRN) attempted to automatically search the randomization ranges that maximize real-world performances explicitly. The problem is solved through Bayesian optimization (BO), by training a policy with DR on the candidate distribution $p(\xi)$ and evaluating its reward on the target domain.

However, we argue that online-guided approaches still expose the method to significant human intervention in the form of careful supervision of the real setup while running partially converged policies or policies with unoptimized randomizations. Indeed, as these methods include the real environment in the training loop to optimize domain randomization distributions, safety concerns and real constraints such as manually resetting the environment often make the process slow and potentially dangerous. In addition, training cannot be fully parallelized on a remote cluster in this setting as the algorithm needs access to the target environment in between iterations.

The work by Tsai et al. [45] (DROID), published a few months prior to completing this thesis, is the most reminiscent to our proposed method. The authors relax the need of iterative policy rollouts on the real environment while training, assuming that only a fixed set of manually collected data is available. In particular, they show that randomization distributions $p(\xi)$ may be optimized in an offline setting with a single target trajectory by minimizing the L2 norm between real and simulated torques. However, we claim that their objective function does not promote variance in the parameters, hence leading to a point-estimate upon convergence. As DROID works in the same framework as our proposed method, we take their method as a baseline and prove our claims in our experiments (see Chapter 6).

OpenAI's successful work in solving a Rubik's cube with a robotic hand [19]

also presented an automatic method for adjusting uniform parameter ranges. In this case, the authors approached DR as a meta-learning problem, by exploiting memory in recurrent neural network structures to adapt policy behavior rather than learning a single robust solution. In particular, they let parameter ranges grow in size over time to encourage a gradually increasing complexity, until average performances in simulation are no longer greater than some thresholds. Note that the authors also include injection of noise in the observations and explicitly model action delay in their simulated model. We generally do not consider this step as part of the pure domain randomization approach, but several works have shown that it can be stacked on top of DR to further encourage robustness in the transfer [16, 31, 22].

Finally, other works have taken different approaches to the tuning of $p(\xi)$, namely by keeping a fixed distribution and relying on intelligent sampling techniques to improve the transfer. The main assumption of these works is that even when the distribution is held fixed, trained policies with DR have large variance when using standard sampling or are sub-optimal. Therefore, Mehta et al. [40] propose to learn an optimal curriculum of tasks within the given distribution that facilitates training from easier to harder parameters, without requiring real-world data; Muratore et al. [41] attempt to learn the minimum number of sampled tasks needed for the policy to learn robust behavior with respect to other reference environments within the same distribution; Yu et al. [109] train a family of policies in simulation with DR on a given $p(\xi)$, then pick the best performing one on the real environment.

Chapter 5

DROPO

In this chapter we introduce *Domain Randomization Off-Policy Optimization*, abbreviated to DROPO, a novel method to automatically infer the parameter ranges of the domain randomization distribution $p(\xi)$.

While learning in simulation removes some of the challenges of real-world reinforcement learning in robotics (see Section 3.1), sim-to-real transfer algorithms still need to cope with many of the limitations of current simulators in modeling the real-world, i.e. the reality gap. Dulac-Arnold et al. [80] recently identified nine high-level challenges that reinforcement learning algorithms for robotics need to address in order to successfully deploy RL policies in the real-world for good. With the work in this thesis we aim to tackle seven of these challenges, leaving the rest as future directions of research. In particular, with respect to the challenges mentioned in [80], DROPO is able to:

- learn off-line from fixed rollouts of an external behavior policy;
- strongly limit the number of samples needed from the real system for learning an efficient policy in simulation;
- cope with safety-critical environments as learning happens entirely in simulation and target data is collected through hand-coded policies or human demonstrations;
- deal with real tasks that are non-stationary or stochastic, by learning parameter ranges that cover the dynamics of the real environment;
- avoid real-world reward computation as no policy is evaluated on the target setup while training;
- deploy policies that work in real-time on real hardware, as inference and training happens offline;

- work under delayed and noisy sensors, by preprocessing the target dataset and intrinsically assuming noise at inference time;

Learning from high-dimensional state spaces such as visual observations in end-to-end fashion is out of the scope of this work. Therefore, we assume pose estimation of the interacting objects is given or measured through motion-tracking systems.

5.1 Intuition

We introduce *Domain Randomization Off-Policy Optimization* (DROPO), a novel method which aims to learn parameter distributions for DR while heavily limiting human supervision of the real environment. Indeed, throughout our work we assume that the robot is not accessible while training and that only a fixed limited set of data from the real setup can be collected, e.g. through human demonstrations, prior to running the algorithm. For this reason, in contrast to online-guided approaches, we denote our method as *off-policy*: the optimization problem takes place on data collected offline through a different policy than the one to be optimized.

The core idea of DROPO lies in *maximizing the probability of real data to be observed in simulation*, by adjusting the domain distribution $p(\xi)$ accordingly. More specifically, when performing domain randomization we can think of the physics simulator as a model with stochastic behavior induced by $p(\xi)$, as the forward dynamics would now vary depending on the current parameter sample $\xi \sim p(\xi)$. Therefore, we aim to approximate the likelihood of collected data with respect to the stochastic physics simulator and maximize it. After the inference step, a policy is trained entirely in simulation with domain randomization on the optimized distribution $p(\xi)$, and finally deployed in the real-world. An overview of DROPO's logic is illustrated in Fig. 5.1

Note that, by maximizing likelihood, variance in the optimized parameters is promoted when the simulator is not able to capture all real-world transitions with a single point-estimate parameter. For this reason, DROPO intuitively tries to find a domain distribution that covers all collected data points without overly widen the ranges. This conservative aspect is important when training feed-forward neural network policies, as a single robust solution needs to be found.

To further highlight the importance of an off-policy approach to adjust simulators for sim-to-real transfer algorithms, we identified a list of advantages over online-guided methods:

- **Safe data collection**: real robotic tasks often allow humans to provide some demonstrations of the task or come up with hand-coded policies to collect data with no harm for the environment. Collecting data this way is safer than running RL policies trained in simulation;

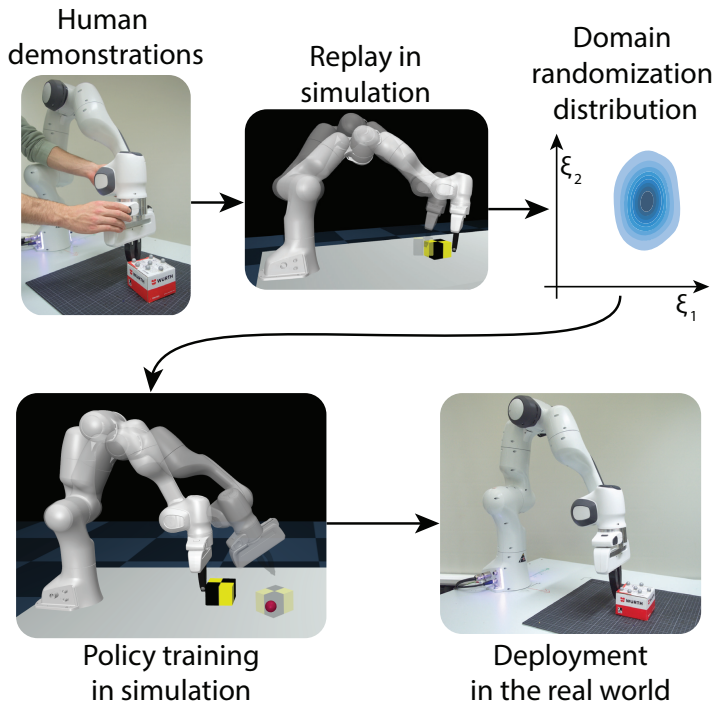


Figure 5.1: DROPO uses off-policy data or human demonstrations to learn a domain randomization distribution, which is later used to train a policy that can be transferred to the real world.

- **Minimal human intervention**: interleaving real-world rollouts to policy updates requires regular human supervision of the target environment. Policies learned in simulation need to be carefully examined before they are deployed on the real robot to avoid unexpected behavior and manual resetting of the environment is often needed in robotic tasks. In general, running policies on real hardware is an expensive task in RL for robotics (see Section 3.1). Finally, online-guided algorithms clearly cannot be run remotely on a cluster and exploit full parallelization during training from start to end.

- **No real-word reward computation**: DROPO does not include the real environment in the optimization loop, nor evaluating policy performance on real hardware during the process. Therefore, no reward computation in the real world is needed;

- **Informative data**: as data is manually collected by humans, the trajectories can be chosen such as to give the most information for the inference of physical parameters. We argue that on-policy approaches do not guarantee that intermediate policies correctly explore the target state-action space for optimal inference.

5.2 The Method

5.2.1 Problem formulation

Recall the a Markov decision process M defines an RL task and is described by its state space $\mathcal{S} \subseteq \mathbb{R}^p$, action space $\mathcal{A} \subseteq \mathbb{R}^q$, initial state distribution ρ_0, dynamics $p(s_{t+1}|s_t, a_t)$ and reward function $r(s_{t+1}, s_t, a_t)$, assumed deterministic for simplicity. Each episode begins with an agent starting in state $s_0 \sim \rho_0$. At each step t, the agent selects an action a_t following a policy $\pi_\theta(a_t|s_t)$. The problem addressed in reinforcement learning is then to find an optimal policy $\pi_{\theta^*}(a|s)$, such that the expected discounted cumulative reward $J(\theta)$ is maximized, with a discount factor $\gamma \in (0,1]$:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r(s_{t+1}, s_t, a_t)\right] \tag{5.1}$$

In the context of the sim-to-real transfer paradigm, we can view the real world as the target task M_T with state transition probabilities $p_{real}(s_{t+1}|s_t, a_t)$. On the other hand, we view the physics simulator as a forward model with stochastic dynamics induced by a distribution $p_\phi(\xi)$ over the physical parameters $\xi \in \Xi \subseteq \mathbb{R}^d$ of the simulator. The parameterization ϕ of the distribution may, for example, carry the mean and covariance matrix of a multivariate Gaussian distribution. Therefore, we describe the simulator dynamics at each transition with the random variable S^ϕ_{t+1} distributed as $p_{sim}(s^\phi_{t+1}|s_t, a_t; \phi)$. Under this setting, the simulator then results in a distribution of source tasks, according to $p_\phi(\xi)$. In particular, each sample ξ from $p_\phi(\xi)$ leads to a specific simulator described by transition probabilities $p_{sim}(s^\xi_{t+1}|s_t, a_t; \xi)$.

We assume that all MDPs, including the target one, share the same state-action space $\mathcal{S} \times \mathcal{A}$, initial state distribution, and reward function.

The problem we address in this work can be stated as follows: given a dataset of state-action trajectories from the target task $\mathcal{D} = \{s_0, a_0, s_1, \ldots, s_{T-1}, a_{T-1}, s_T\}$, find ϕ^* such that the probability of transitioning to the real-world state s_{t+1} in simulation, when taking action a_t while in s_t, is maximized over the dataset:

$$\phi^* = \arg\max_\phi \mathop{\mathbb{E}}_{s_t, a_t, s_{t+1} \sim \mathcal{D}} \; p_{sim}(S^\phi_{t+1} = s_{t+1}|s_t, a_t; \phi) \tag{5.2}$$

An illustration of the problem formulation is shown in Fig. 5.2

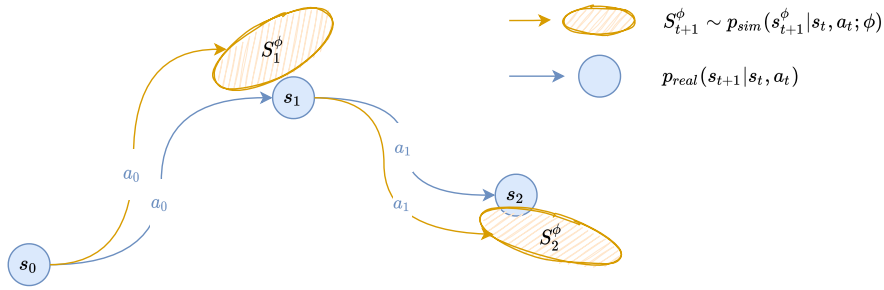


Figure 5.2: In DROPO, the probability of the observed real word data (blue path) is maximized with respect to the distribution of states of the stochastic simulator (orange path), whose random behavior is induced by $p_\phi(\xi)$

5.2.2 Method overview

DROPO consists of three main phases: dataset collection on the real hardware, dynamics distribution fitting, and policy training.

The data collection step, performed on the physical hardware, is where the dataset $\mathcal{D}$, used for parameter optimization, is collected. This can be done either by running previously trained policies (e.g. a policy for another task), by hand-coding an exploration policy, or by manually guiding the robot. 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 phase, where the parameter distribution is inferred. During the optimization process, each state-action pair in $\mathcal{D}$ is executed multiple times by the stochastic simulator to approximate the distribution of the random variable $S^{\phi}_{t+1} \sim p_{sim}(s^{\phi}_{t+1}|s_t, a_t; \phi)$. Recall that since domain randomization is applied, we model the underlying distribution over dynamics parameters as $p_\phi(\xi)$, which induces a stochastic behavior of the simulator when queried. The parameters ϕ of $p_\phi(\xi)$ are then adjusted to maximize the likelihood of s_{t+1} under the random variable S^{ϕ}_{t+1}.

Once the parameter optimization process has converged, the resulting distribution $p_{\phi^*}(\xi)$ is used to train a policy with domain randomization (see Section 3.4.2) in the third step of DROPO. This policy can then be deployed on the setup that the data was collected from in the first phase.

An overview of DROPO is presented in Fig 5.3.

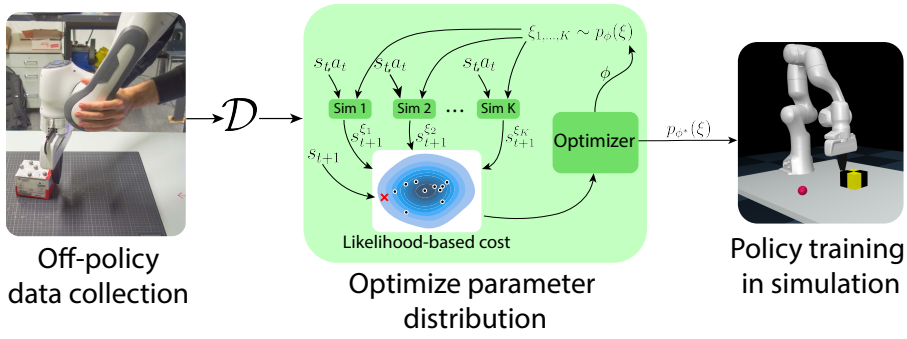


Figure 5.3: Overview of the three main stages of DROPO. A dataset collected on the physical hardware is used to optimize a parameter distribution, which is then used for policy training.

Dataset 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 by running another policy in the environment, or collected from human demonstrations. 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 damping values for a joint that does not move during the demonstrations, or the friction coefficient of an object that does not move during the 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 arm— this inference can be simplified under the assumptions that the robot's onboard controllers are ideal. While this is never the case in real-world systems, we found that DROPO was able to handle this discrepancy. An alternative approach, proposed by DROID [45], suggests to replay the demonstrations on the real robot once collected: this way, although the trajectory followed may be slightly different, the real actions taken can be automatically obtained.

After collection, the dataset needs a few preprocessing steps 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 with respect to 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 later evaluate the spline 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.

Distribution fitting Once a preprocessed trajectory dataset is available, we start the core part of DROPO — estimating the dynamics probability distribution $p_\phi(\xi)$, modeled as a multivariate Gaussian.

In contrast to DROID [45], in our method the ϕ parameter vector includes not only the means of the distribution $p_\phi(\xi)$, but also the variances. This change allows to explicitly learn the variability of each simulated physical parameter to best explain heteroscedasticity and uncertainty in real-world data. As such, DROPO can be thought of a random coefficient statistical model (see Section **??**), where the optimized coefficients are themselves assumed to be random variables, as opposed to the more standard fixed-effects regression analysis. This parameterization allows fitting real data with a stochastic model on the dynamics parameters, such that the probability of observing the dataset in simulation is maximized.

The distribution fitting process starts with an arbitrary initial guess $\phi_{init} = (\mu_{init}, \Sigma_{init})$ on the dynamics parameter distribution, roughly informed by looking at the replayed trajectory in sim.

The main inference part follows by sampling K dynamics parameters $\{\xi_1, \ldots, \xi_K\}$ from $p_\phi(\xi)$. Then, for each parameter vector ξ_i, we set the simulator state to the original state s_t, execute the same action a_t, and observe the next state $s_{t+1}^{\xi_i}$. This process allows us to estimate the distribution of the random variable S_{t+1}^{ϕ}.

In particular, we infer the next-state distribution mean $\bar{s}_{t+1}^{\phi}$ and covariance Σ_{t+1}^{ϕ} as follows:

$$\begin{aligned} \bar{s}_{t+1}^{\phi} &= \frac{1}{K} \sum_{i=1}^{K} s_{t+1}^{\xi_i} \\ \Sigma_{t+1}^{\phi} &= \widehat{\mathrm{Var}}(S_{t+1}^{\phi}) + \mathrm{diag}(\epsilon) \end{aligned} \tag{5.3}$$

Where $\widehat{\mathrm{Var}}(\cdot)$ is the unbiased sample covariance matrix estimator:

$$\widehat{\mathrm{Var}}(S_{t+1}^{\phi}) = \frac{1}{K-1} \sum_{i=1}^{K} \left(s_{t+1}^{\xi_i} - \bar{s}_{t+1}^{\phi} \right) \left(s_{t+1}^{\xi_i} - \bar{s}_{t+1}^{\phi} \right)^T \tag{5.4}$$

Here, ϵ is a newly introduced hyperparameter used to compensate for observation noise and regularize the likelihood computation in case singular sample covariance matrices would appear. Tuning this hyperparameter will, effectively, adjust how much next state variance is to be explained by variance in dynamics parameters ξ, relative to observation noise. As demonstrated in our experiments,

setting this parameter to too low values may result in obtaining a too wide randomization distribution, as, in the extreme case of $\epsilon = 0$, DROPO would attempt to explain all variability in the state transitions by some variance in the dynamics parameters, including unexplainable phenomena and noise. In contrast, setting ϵ to overly large values would lead DROPO to explain most of the variability as homoscedastic noise, leading to narrow dynamics distributions in the maximum-likelihood computation.

Obtaining the mean and the covariance matrix allows us to fit a multivariate Gaussian distribution that models S^{ϕ}_{t+1}. We can then make the assumption that the true, real-world observation s_{t+1} originates from the distribution S^{ϕ}_{t+1} induced by dynamics randomization, as:

$$s_{t+1} \sim \mathcal{N}(\bar{s}^{\phi}_{t+1}, \Sigma^{\phi}_{t+1})$$

The log likelihood of s_{t+1} under this distribution can be then calculated following the standard Gaussian probability density function:

$$\mathcal{L}_{t+1} = -\frac{1}{2}(p \log 2\pi + \log |\Sigma^{\phi}_{t+1}| + (\bar{s}^{\phi}_{t+1} - s_{t+1})^{\top} \cdot (\Sigma^{\phi}_{t+1})^{-1} \cdot (\bar{s}^{\phi}_{t+1} - s_{t+1})) \quad (5.5)$$

Intuitively, we may repeat the process for each transition in the dataset $\mathcal{D}$. The log-likelihoods of each individual transition can then be summed to obtain the total log likelihood of the dataset $\mathcal{D}$ under a stochastic simulator induced by $p_{\phi}(\xi)$:

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t \quad (5.6)$$

We finally adjust the parameters ϕ to maximize the total log likelihood $\mathcal{L}$. As we don't make any assumptions about the differentiability of the simulator, we optimize $\mathcal{L}$ using a gradient-free optimization method. The method can, in principle, work with an arbitrary optimization method; we use CMA-ES [68] throughout the experiments.

Policy training After obtaining $p_{\phi}(\xi)$, we train a policy $\pi_{\theta}(a|s)$ for the given target task with domain randomization on $p_{\phi}(\xi)$. This is achieved by sampling new dynamics parameters $\xi \sim p_{\phi}(\xi)$ at the start of each training episode (see Section 3.4.2). The RL policy can then be trained with an arbitrary reinforcement learning algorithm; we use PPO [53] in our experiments. Remind that no data from the offline-collected dataset is involved in the training process, as learning from demonstrations is out of the scope of this work.

Algorithm 2: DROPO

Result: Parameters ϕ^* of $p_{\phi^*}(\xi)$

Initialize ϕ to $(\mu_{init}, \Sigma_{init})$;
Initialize empty dataset $\mathcal{D}$;
Fill $\mathcal{D}$ with demonstrations from the target task ;
while *not converged* **do**
    Sample K dynamics parameters $\xi_1, \dots, \xi_K$ from $p_\phi(\xi)$;
    forall $s_t, a_t, s_{t+1} \in \mathcal{D}$ **do**
        foreach $\xi_i \in \{\xi_1, \dots, \xi_K\}$ **do**
            Set simulator parameters to ξ_i ;
            Set the simulator state to s_t ;
            Execute a_t ;
            Observe $s_{t+1}^{\xi_i} \sim p_{sim}(s_{t+1}^{\xi_i} | s_t, a_t; \xi_i)$;
        end
        Compute the mean $\bar{s}_{t+1}^{\phi}$ and covariance Σ_{t+1}^{ϕ} (Eq. 5.3);
        Evaluate the log-likelihood $\mathcal{L}_{t+1}$ of s_{t+1} under $\mathcal{N}(\bar{s}_{t+1}^{\phi}, \Sigma_{t+1}^{\phi})$ (Eq. 5.5);
    end
    Compute the total log-likelihood $\mathcal{L} = \sum_t \mathcal{L}_t$;
    Update ϕ towards maximizing $\mathcal{L}$;
end
Train a policy $\pi_\theta(a|s)$ using the converged $p_{\phi^*}(\xi)$;

5.3 Implementation choices

Important considerations need to be made regarding the implementation of algorithm 2, as approximating the likelihood function with implicit forward models can be rather challenging [108]. Therefore, we now report all implementation details needed to get the best-performing version of DROPO.

Parameterization of $p_\phi(\xi)$. The literature hasn't agreed on whether Gaussian or uniform distributions are best suited for domain randomization. Both approaches have been adopted successfully in solving sim-to-real tasks in recent years. Throughout our work, we opted for Gaussian distributions over dynamics parameters for mainly two reasons. Firstly, it seems to be the most common choice among the related works which adapt the distribution to bring simulated observations closer to real data with feed-forward policies [43, 108, 45]. This is likely motivated by the fact that a meaningful *gradient* is more easily provided to the optimization algorithm, whereas the flat nature of uniform distributions would make it harder to give information for the adjustment of ϕ. Secondly, a number of works have claimed uniform distributions to lead to high-variance policies while training [41, 40, 109].

More specifically, we model the distribution $p_\phi(\xi)$ as an uncorrelated multivariate Gaussian distribution with diagonal covariance matrix. Therefore, with a dynamics parameter vector $\xi \in \mathbb{R}^d$, the parameterization is represented by a $2d$-dimensional vector $\phi \in \mathbb{R}^{2d}$, containing mean and variance of each univariate physical parameter. While a full covariance matrix may alternatively be parameterized, we opted for limiting the amount of parameters in the optimization problem and observed that closely related works have not found significant correlations among physical parameters [43].

Finally, we have to deal with the feasible boundaries of some parameters ξ, such as masses and friction coefficients. Due to their nature, such physical parameters must indeed always be positive, which is not guaranteed when sampling from a Gaussian distribution $p_\phi(\xi)$. In our implementation, we adjust the Gaussian probability density function by resampling values which lie farther than two standard deviations from the mean. In addition, we discard unfeasible parameters potentially arising within the boundaries considered and resample them. If physically unfeasible parameters are still present in our current sample after 20 resampling attempts, we manually set these to their closest boundary value.

Likelihood computation. Computing the Gaussian probability density function with sample estimates of the covariance matrix in high dimensional spaces is often challenging. Indeed, note that the set of K state observations $\{s_{t+1}^{\xi_1}, \dots, s_{t+1}^{\xi_K}\}$

is used to infer Σ^{ϕ}_{t+1} which may be singular under certain circumstances. For example, if the state space dimension p is greater than K, then a singular — hence not invertible — covariance matrix is returned, preventing the computation of the likelihood function. In general, singularities may also appear when $p < K$, as the simulator dynamics $p_{sim}(s^{\phi}_{t+1}|s_t, a_t; \phi)$ may not be able to show variability across p-linearly independent directions in the state space. We overcome this issue by adding a small portion of the identity matrix, scaled by ϵ, to the sample covariance matrix (Eq. 5.3). This regularization fixes all potential singularities by forcing all eigenvalues of Σ^{ϕ}_{t+1} to be greater than zero.

Furthermore, note that the resulting covariance matrix Σ^{ϕ}_{t+1} is by definition symmetric and positive-definite, hence we can work with the Cholesky decomposition $\Sigma^{\phi}_{t+1} = LL^T$, where L is a lower-triangular matrix. This allows faster computation and better numerical precision of the log-likelihood function, as the inverse of the covariance matrix can be computed with p^3 operations and its determinant can be calculated with the alternative formulation $\log|\Sigma| = 2\sum_{i=1}^{p} L_{i,i}$.

Hyper-tuning of ϵ. The choice of hyperparameter ϵ highly affects the converged distribution $p_{\phi^*}(\xi)$. Intuitively, ϵ should be kept as small as possible to ensure that variability is captured in terms of dynamics rather than homoschedastic noise. On the other hand, ϵ is needed for a better conditioning of the sample covariance matrix Σ^{ϕ}_{t+1}, as explained in the previous paragraph, and to account for noisy observations. To ensure stability and encourage randomness in the dynamics parameters, we tune ϵ to be the smallest value that leads to a Mean Squared Error (MSE) below a predefined acceptable threshold value τ_{MSE}, which is generally easier to estimate given the task of interest. Here, we refer to the MSE w.r.t. to the target dataset $\mathcal{D}$ as the metric $d_{\mathcal{D}}(\phi^*) = \sum_t \|(\bar{s}^{\phi^*}_{t+1} - s_{t+1})\|^2$. In other words, we tune ϵ such as to bias the likelihood computation towards minimizing the same objective function as that of a classical non-linear regression problem with fixed-coefficients, up until a certain threshold. See Section 6.2.5 for a practical example.

Finally, note how ϵ could be naturally exploited to inject noise on data at training time on top of the domain randomization induced by $p_{\phi}(\xi)$, due to its definition. This approach is however out of the scope of this work and left as a future direction of research.

Inference over multiple timesteps. The algorithm in 2 may be somewhat limited, as likelihood computations are performed for each consecutive real-world transition. In practice, the specific time step separating each transition depends on a number of factors such as stability concerns in simulation when doing numerical integration, or the frequency of real-world sensors. To make the algorithm more flexible, we allow for likelihood computations considering a longer time horizon. We then introduce a hyperparameter $\lambda \in \mathbb{N}^+$, which explicitly describes

the number of timesteps to consider for each likelihood computation. Therefore, at each time instant t we generally aim to infer the distribution of the random variable $S^{\phi}_{t+\lambda}$ distributed as $p_{sim}(s^{\phi}_{t+\lambda}|s_t, a_t, a_{t+1}, \dots, a_{t+\lambda}; \phi)$, by executing multiple actions at once. Note how the next transition now starts from $s_{t+\lambda}$, as no overlapping sequences are considered. This flexibility may also be viewed as a regularization technique to deal with real-world noisy observations. More specifically, a single-timestep transition may not be affected by the current dynamics parameter distribution $p_{\phi}(\xi)$ as much as noise in the real observations, therefore we look further ahead to infer variability actually induced by the physical parameters. We denote this technique as λ-*steps*, and for the sake of notation clarity we omit it from the general DROPO formulation.

Regularization of CMA-ES optimization. Even though gradient-free optimization algorithms allow to optimize functions without explicitly working on their gradients, they often need careful tuning in order to get the best results. In particular for CMA-ES (see Section 2.3.1), the scale of the optimized parameters highly affects the moving covariance matrix estimate and the ability to minimize the objective function efficiently. In this work, the optimized parameters are the components of the vector $\phi \in \mathbb{R}^{2d}$, carrying mean and variance of each dynamics parameter ξ randomized. We choose to work with each mean and variance in the interval [0,4]. Therefore, we initialize each parameter to the middle value 2 and initially set Σ in CMA-ES as the identity matrix. This allows each dimension to be similarly scaled in the input space of the optimization problem, and the best solution to be at most 2 standard deviations away from the initial guess. Then, each candidate solution ϕ is denormalized back to the original space when evaluated, according to predefined search bounds $[L_{\phi_i}, U_{\phi_i}]$. More specifically, means ϕ_{2i} are linearly rescaled as $\phi_{2i}(U_{\phi_{2i}} - L_{\phi_{2i}})/4 + L_{\phi_{2i}}$, while variances ϕ_{2i+1} are optimized in log-space and denormalized as $L_{\phi_{2i+1}} \left(\frac{U_{\phi_{2i+1}}}{L_{\phi_{2i+1}}}\right)^{\phi_{2i+1}/4}$.

Chapter 6

Experiments and Results

Several experiments have been designed and carried out both in simulation and in the real world to validate the claims of DROPO. In particular, we test DROPO's ability to recover ground truth dynamics parameters in simulation both in the form of point-estimates and distributions. Moreover, we report experiments demonstrating that converged dynamics distributions $p_{\phi^*}(\xi)$ can be used to train policies which overcome unmodeled phenomena in simulation and generalize well to the real world.

To demonstrate the above claims in simulation, we designed a simplified setting of the sim-to-real paradigm where the transfer happens solely in simulation, with a predefined target simulator representing the real-world. We denote this setting as *sim-to-sim transfer* and we report the corresponding experiments on the simulated Hopper OpenAI Gym environment [94] in Section 6.2.

The real-world experiments are carried out on two real robotic arms: Kuka LWR4+ and Franka Panda.

Overall, this chapter reports the following five setups and experimental questions:

- **sim-to-sim point-estimate system identification**: is DROPO able to converge to the ground truth physical parameters ξ when a synthetic dataset $\mathcal{D}$ is filled with Hopper transitions generated with such parameters?
- **sim-to-sim distribution recovery**: is DROPO able to converge to the ground truth distribution $p_\phi(\xi)$ when a synthetic dataset $\mathcal{D}$ is filled with Hopper transitions generated with dynamics parameters ξ sampled from such distribution?
- **sim-to-sim unmodeled phenomena**: is DROPO able to identify dynamics parameter distributions that can be used to learn policies which overcome

unmodeled phenomena in simulation?

- **sim-to-real sliding task**: we test the full DROPO framework on a sim-to-real transfer scenario. Here, the KUKA LWR+ robotic arm is equipped with a hockey stick and tasked with hitting a hockey puck such that it stops at the given target location.

- **sim-to-real pushing task**: we test the ability of DROPO to solve sim-to-real transfer tasks as in the previous experiment. This time, the goal is to push a heavy box with shifted center of mass forward to a target location.

6.1 Baselines

Throughout our experiments we compare the results obtained with DROPO with two other approaches, namely our *baselines*.

Firstly, we compare DROPO with uniform domain randomization (UDR), the current standard way to apply domain randomization in many sim-to-real robotic works [32, 20, 110]. UDR refers to all those cases where domain randomization is applied on manually engineered distributions, likely by prolonged trial and error and tedious tuning to get the trained policies successfully transferred to real robots. Therefore, as DROPO aims to automatically infer randomization distributions, we aim to compare its efficiency against unoptimized uniform bounds. More specifically, we follow the same approach taken by authors in [44]: instead of manually tuning domain randomization distributions — which would be hard to scientifically quantify and reproduce throughout the experiments — we randomly sample several uniform bounds within the considered search space of parameters ξ and train a policy with DR on each of these bounds. This way, UDR results should give a fair idea on how different choices of uniform bounds affect the performances on the target domain, and how easy it is to solve the task with random uniform domain randomization.

Secondly, we compare DROPO with the only other — to the best of our knowledge — offline-guided DR optimization method by Tsai et al. [45] (DROID), published a few months prior to the completion of this thesis. In contrast to our method, DROID optimizes only the means of each parameter ξ in $p(\xi)$ and finally obtains a distribution by exploiting the converged covariance matrix of CMA-ES. In this setting, DROID uses an L2-distance based objective function to optimize parameters ξ in simulation. In particular, the authors compare joint torques measured on the real robot versus torques exerted in simulation, when the offline trajectories are reproduced with a joint position controller (e.g. a PID controller). We claim that this formulation may be conveniently changed to a comparison of states, whenever needed. For example, when inferring dynamics parameters in our sliding task, torque measurements would give little to no help in understanding the

hockeypuck mass or friction coefficient with the sliding surface: for the majority of the episode the robotic arm moves unconstrained in the air and only hits the puck in a certain time instant, making torque measurements unstable and unreliable. We therefore implement DROID with L2-distances among state observations rather than torques throughout all our experiments. Nevertheless, we claim that DROID tends to converge to point-estimate dynamics in both cases, as the L2-distance based objective function does not promote variance. We validate our claims across our experiments.

6.2 Sim-to-Sim transfer

This section reports all our findings in the simplified sim-to-sim transfer setting. To model a transfer to unknown tasks, we take a reference simulator as the target environment and collect demonstrations from it as if it were the real world. More specifically, we can think of the target simulator as a distribution of tasks parameterized by ϕ^g, denoted as the *ground truth dynamics parameters*. Intuitively, DROPO can be normally applied with no information on ϕ^g, and then finally evaluated by comparing the converged distribution to the ground truth value or directly measuring policy performance on the target environment.

6.2.1 Experimental setup

All experiments in this section are conducted on the Hopper OpenAI Gym environment [94], shown in Fig. 6.1. Hopper is a great toy environment to test out algorithms for real robotic tasks and has previously been proposed to benchmark RL algorithms and assess the results of sim-to-real transfer algorithms [42].

In this environment, a 3-DoF one-legged robot is tasked to learn to jump forward in a single direction as fast as possible without falling down. Hence, the agent is rewarded based on its current velocity along the horizontal axis. Moreover, the agent is given an additional reward term for *staying alive* — not falling down — and is slightly penalized proportionally to the square of the magnitude of the torques exerted, to promote stability in the control policy. Each episode then starts with the Hopper standing in the up-right position with a low uniform noise on the joint positions and velocities. Finally, if the agent survives longer than 500 timesteps, the episode is automatically stopped.

The reinforcement learning problem associated with the Hopper environment can be viewed the same way as for manipulation tasks: each of the three joints of the one-legged robot can freely move and exert an arbitrary torque to move the robot around, therefore we aim to learn optimal torque control over these joints as to maximize the reward function earlier described.

The state observation vector fully describes the configuration of the simulator at each time instant and includes 11 dimensions, split in: 3 joint positions, 3 joint velocities, x-y-z Cartesian velocities of the torso and y-z Cartesian positions of the torso. In particular, the x Cartesian position of the torso, describing lateral movement on the horizontal axis, is not included as it doesn't affect the configuration of the Hopper.

The physical parameters considered for domain randomization in DROPO are the 4 masses of each link in the robot, namely the *torso* (m_1), *thigh* (m_2), *leg* (m_3) and *foot* (m_4). The upper and lower search bounds for the optimization of the means ϕ_{2i} have been set to double and half the ground truth masses respectively, while standard deviations may range from half the search interval size down to 10^{-5}.

Finally, note that we keep λ fixed to 1 throughout all sim-to-sim experiments for the sake of simplicity, as the default version of DROPO already performs well in simulation. λ-steps are instead used in the sim-to-real experiments. The sample size K is set to 100 to ensure accurate estimation of means and variances in the 11-dimensional state space.

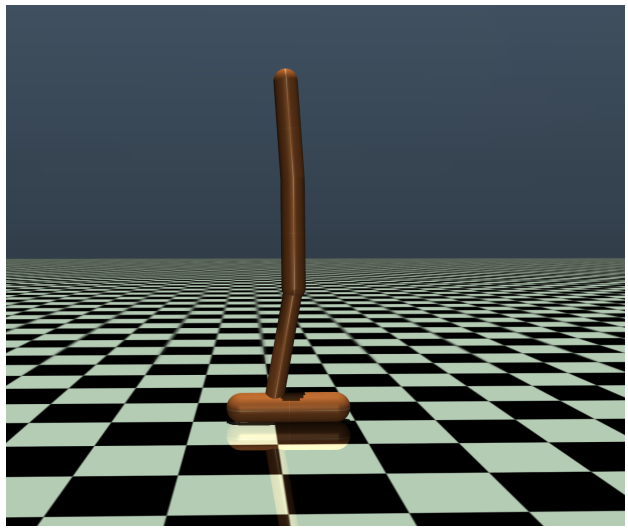


Figure 6.1: The Hopper environment by OpenAI Gym [94]. The one-legged robot has three degrees of freedom and is tasked to learn to jump forward in a single direction as fast as possible without falling down.

6.2.2 Point-dynamics system identification

This experiment aims to answer a first simple research question: is DROPO able to converge to the ground truth physical parameters ξ^g when a synthetic dataset $\mathcal{D}$ is filled with Hopper transitions generated with such parameters?

When the target dataset is collected in simulation on a single environment instance, DROPO is indeed expected to identify the original physical parameters and to converge to point-estimate dynamics rather than distributions. Indeed, a distribution over physical parameters would only lower the likelihood as a single point-estimate may already explain all of the transitions in $\mathcal{D}$.

We tested this behavior on the Hopper environment by collecting a target dataset $\mathcal{D}$ consisting of 1000 transitions — two trajectories worth of data — from the target Hopper environment with a partially-converged exploration policy. Note that even though it is generally unrealistic to have a ready-to-use exploration policy in a sim-to-real transfer case, this approach is handy in simulation as human demonstrations cannot be physically provided. Therefore, we explicitly train a policy for exploration in the target environment prior to running DROPO, and stop the training half-way through convergence to avoid optimal behavior when collecting data. A noisy version of the dataset $\mathcal{D}_{noisy}$ has also been created by injecting zero-mean Gaussian noise of variance 10^{-5} in the observations. The ground truth dynamics of the target task used for data collection are shown in Table 6.1.

The hyperparameter ϵ has been tuned according to our description in Section 5.3, and set to $\epsilon = 10^{-8}$ and $\epsilon = 10^{-5}$ for the clean and noisy versions respectively. We report in Section 6.2.5 a full example of how ϵ can be tuned.

The results are reported in the same table, as DROPO successfully converges to the original ground truth dynamics regardless of the presence of noise. Furthermore, DROPO effectively decreases the standard deviation of learned parameters to 10^{-5}, set as the lowest bound during the optimization problem. In this case, DROID is also able to correctly identify the ground truth parameters.

6.2.3 Distribution recovery

The premise of this experiment is very similar to the previous one; this time, however, the dataset $\mathcal{D}$ was generated using a ground truth dynamics distribution parameterized by ϕ^g, rather than a single value ξ^g, as if the target environment behaved randomly. The goal for DROPO is then to recover the original distribution the dataset was collected from, so that policies can be trained on distributions that actually capture the target dynamics.

The dataset $\mathcal{D}$ consists of 1000 transitions — two trajectories worth of data — collected on the Hopper environment with the same partially-converged exploration policy used in 6.2.2, sampling new dynamics parameter every 10 transitions according to ϕ^g. As a result, the reference dataset $\mathcal{D}$ contains transitions collected

	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
$\mathcal{D}$	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
$\mathcal{D}_{noisy}$	DROPO ϵ=1e−5	μ^*	3.534	3.927	2.714	5.089
		σ^*	1.0e-5	1.0e-5	1.0e-5	1.0e-5
	DROID	μ^*	3.534	3.927	2.714	5.089
		σ^*	1.2e-12	2.4e-11	2.0e-12	1.2e-12

Table 6.1: Converged dynamics distributions on the Hopper environment with a point-dynamics target task. Two trajectories from the target environment have been collected offline with a partially-converged exploration policy. Both DROPO and DROID are able to converge to the original ground truth values, even when noise of variance 1e-05 is injected in the observations ($\mathcal{D}_{noisy}$).

with 100 different observations $\xi \sim p_{\phi^g}(\xi)$. Analogously to the previous experiment, we additionally test DROPO on a noisy version $\mathcal{D}_{noisy}$ of the dataset, with 10^{-5} variance Gaussian noise on the state observations.

Two different ground truth distributions have been tested and reported in Table 6.2 and 6.3. In particular, we designed ϕ^{g_1} to vary the torso mass only, while ϕ^{g_2} randomizes all parameters except for the torso mass. The challenge with these ground truth distributions is to effectively learn variance in the varying parameters, while converging to point-estimates on those who remain fixed while collecting data.

The results are illustrated in Fig 6.2, which shows DROPO successfully learning the original ground truth variance of the randomized masses in both cases. 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 when these vary during data collection. The noisy version of the dataset shows slightly higher individual variances in DROPO's optimized distribution. We explain this trend with DROPO looking for a trade-off between observation noise and variability on the random coefficients. Indeed, the algorithm will end up trying to explain some noise in terms of dynamics variance, if needed to maximize likelihood. Such behavior may be desired for DR policy training, and in line with several sim-to-real works which manually inject observation noise at training time [32, 31, 16].

	Masses (kg)		m_1	m_2	m_3	m_4
	Ground truth	μ^{g_1}	3.534	3.927	2.714	5.089
		σ^{g_1}	0.5	1e-5	1e-5	1e-5
	Search	min	1.767	1.963	1.357	2.545
	space	max	7.069	7.854	5.429	10.179
$\mathcal{D}$	DROPO ϵ=1e−8	μ^*	3.524	3.927	2.714	5.090
		σ^*	0.478	3.1e-4	3.4e-4	1.1e-5
	DROID	μ^*	3.465	3.925	2.715	5.097
		σ^*	4.91e-10	4.59e-10	9.36e-10	2.97e-10
$\mathcal{D}$ noisy	DROPO ϵ=1e−5	μ^*	3.652	3.926	2.719	5.096
		σ^*	2.32e-12	1.19e-11	3.96e-11	1.16e-11
	DROID	μ^*	3.664	3.942	2.722	5.101
		σ^*	6.07E-11	5.99E-11	2.51E-10	4.74E-11

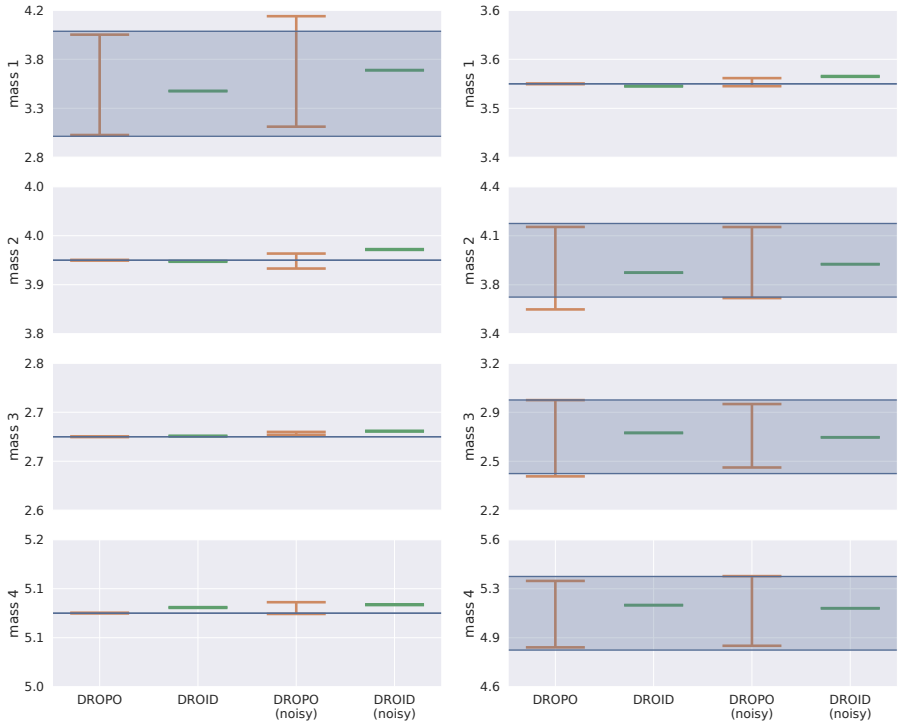
Table 6.2: Distribution recovery results on a target dataset collected in simulation with the ground truth parameters sampled from $\phi^{g_1} = (\mu^{g_1}, \sigma^{g_1})$. In this setting, only the torso mass varies in the target environment.

	Masses (kg)		m_1	m_2	m_3	m_4
	Ground truth	μ^{g_2}	3.534	3.927	2.714	5.089
		σ^{g_2}	1e-5	0.25	0.25	0.25
	Search	min	1.767	1.963	1.357	2.545
	space	max	7.069	7.854	5.429	10.179
$\mathcal{D}$	DROPO ϵ=1e−8	μ^*	3.534	3.873	2.704	5.083
		σ^*	0.0004	0.280	0.259	0.226
	DROID	μ^*	3.531	3.843	2.741	5.145
		σ^*	4.91e-10	4.59e-10	9.36e-10	2.97e-10
$\mathcal{D}$ noisy	DROPO ϵ=1e−5	μ^*	3.537	3.912	2.721	5.105
		σ^*	0.005	0.242	0.216	0.237
	DROID	μ^*	3.544	3.901	2.711	5.124
		σ^*	1.01e-10	2.52e-10	4.04e-10	6.93e-11

Table 6.3: Distribution recovery results on a target dataset collected in simulation with the ground truth parameters sampled from $\phi^{g_2} = (\mu^{g_2}, \sigma^{g_2})$. In this setting, all masses except for the torso mass vary on the target environment.

6.2.4 Unmodeled phenomena

The ability of DROPO to overcome injected unmodeled phenomena in simulation is now tested, by evaluating the policy performance of the complete transfer to a target task. Therefore, we apply all three steps of the DROPO's framework, from



(a) (b)

Figure 6.2: (a) Optimized dynamics distributions by DROPO and DROID on offline data collected with a varying torso mass as $m_1 \sim \mathcal{N}(3.53, 0.5^2)$. (b) Dynamics distribution optimized on a different dataset collected with all masses varying as $m_i \sim \mathcal{N}(\mu_i^g, 0.25^2)$, except for the torso mass. The blue shaded areas report the ground truth dynamics distributions the datasets were collected on.

data collection, to distribution fitting and policy training.

We introduce an unmodeled effect in simulation in the form of a misspecified torso mass that remains fixed in the source simulation. Hence, DROPO only optimizes the means and variances of the other three masses and cannot adjust the incorrect torso mass. In this setting, a target dataset $\mathcal{D}$ is collected on the ground truth parameters shown in Table 6.4, and then used by DROPO to infer the three remaining masses such as to overcome a misspecification of the torso mass by 1kg.

Once a converged dynamics distribution $p_{\phi^*}(\xi)$ is obtained, a policy $\pi_\theta(a|s)$ is trained with domain randomization on $p_{\phi^*}(\xi)$ (see Section 3.4.2). In this experiment, we use Soft-Actor-Critic [111] as the RL learning algorithm. In particular,

we use two simple fully connected network structures with ReLu activation to learn the policy and state-action value function:

- **Policy network structure**: 11 input neurons (the observed state), 2 fully connected layers with 256 hidden neurons, fully connected output layer with 6 dimensions (joint torque means and standard deviations);
- **Value function network structure**: 14 input neurons (the observed state plus the action space dimension), 2 fully connected layers with 256 hidden neurons, fully connected output layer with a single dimension describing the state-action value function;

Furthermore, mini-batches of 256 transitions are used to update the networks' parameters, through the Adam optimizer. A learning rate of 10^{-3} is used.

The results of the converged dynamics distribution by DROPO and DROID are reported in Table 6.4. Note how even though the target dataset $\mathcal{D}$ has been collected on point-dynamics parameters ξ^g, DROPO converges to a distribution over the three randomize masses. This behavior is indeed expected, as due to the unmodeled phenomenon introduced in simulation, a single point-estimate parameter vector is not able to reproduce all transitions in the target dataset. Therefore, DROPO widens the dynamics distribution to maximize likelihood and bring the source misspecified simulator closer to the target one. In addition, note that also the optimized means are now much different than the ground truth parameters used. On the other hand, the results show that DROID's objective function is eager to converge to point-estimate parameters, in order to minimize the L2-distance based object function. While the optimization process of DROID may be stopped early to prevent this from happening, the authors in [45] explicitly mention to optimize until convergence. Nevertheless we claim that, even if we do stop the optimization process early, the distribution obtained would not be representative of the variance in real world dynamics.

Finally, the performances of the policies trained with domain randomization on the distributions obtained by DROPO and DROID are evaluated on the target ground truth environment. The results are illustrated in Fig 6.3, together with the UDR baseline and a ground truth policy directly trained on the target environment. As explained in Section 6.1, we report UDR results as the average performances over several policies trained on randomly picked uniform bounds $p(\xi)$. More precisely, we train 10 policies with domain randomization on 10 bounds randomly sampled from the search space in Table 6.4. Note how some UDR policies are at times able to generalize well to the target environment, hence could technically be solved by manual trial and error with uniform bounds. However, DROPO's ability to automatically infer dynamics distribution is able to train policies which effectively transfer to the target domain much better than point-estimate system identification dynamics (e.g. DROID).

	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	10.179
$\mathcal{D}$	DROPO ϵ=1e−3	μ^*	2.534	4.573	2.603	5.089
		σ^*	(fixed)	0.567	0.487	0.954
	DROID	μ^*	2.534	4.021	3.591	4.845
		σ^*	(fixed)	6.0e-6	1.7e-6	4.2e-6

Table 6.4: Optimize dynamics distributions for DROPO and DROID respectively. This time, the source simulator used during the optimization has a misspecified torso mass by $1kg$, and only the remaining three masses can be adjusted.

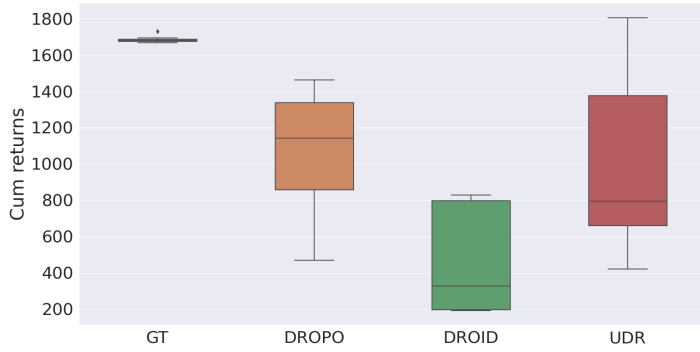


Figure 6.3: Policy performances on the target Hopper environment. DROPO and DROID policies are trained on domain randomization distributions optimized with a misspecified torso mass by 1kg. The UDR baseline shows average results over 10 policies with randomly sampled uniform bounds. Ground truth (GT) shows performances when training on the correct mass values directly in the target environment.

6.2.5 Hyperparameter tuning

The choice of the hyperparameter ϵ highly affects DROPO's converged dynamics distribution, due to its definition (see Section 5.2). Therefore, we report in detail a practical example of the tuning of ϵ. More specifically, we show how ϵ has been adjusted in the case of point-dynamics system identification with a noisy dataset (experiment 6.2.2).

Recall that ϵ acts as a regularization hyperparameter to ensure stability in the covariance matrix estimate Σ_{t+1}^{ϕ} — by keeping all eigenvalues greater than zero — and to deal with uncorrelated sensor noise throughout the observations. Intuitively, ϵ should be kept to the minimum value which regularizes the likelihood computation satisfactorily.

In the case of noisy point-dynamics system identification, the target dataset $\mathcal{D}$ is collected from a specific target simulator described by parameters ξ^g, and then corrupted with 10^{-5} variance Gaussian noise. To tune ϵ for this dataset, we run DROPO several times in parallel with ϵ values varying in log-space. For each converged distribution ϕ^*, we then measure the associated MSE $d_{\mathcal{D}}(\phi^*) = \sum_t \|(\bar{s}_{t+1}^{\phi^*} - s_{t+1})\|^2$.

Fig. 6.4 reports the MSE values and the total variance of the converged distribution $p_{\phi^*}(\xi)$ for each ϵ value on the x-axis. Observe how the MSE drops considerably in the point $\epsilon = 10^{-5}$, suggesting that a much better local minimum is found. Since the MSE does not significantly vary thereafter, we pick the value of 10^{-5} as the final value for the hyperparameter ϵ. It's important to highlight that the total variance also decreases as ϵ increases: indeed, in these case the covariance matrix Σ_{t+1}^{ϕ} already has enough variance on each dimension to explain the target transition s_{t+1}, hence smaller variances in the physical parameters are needed.

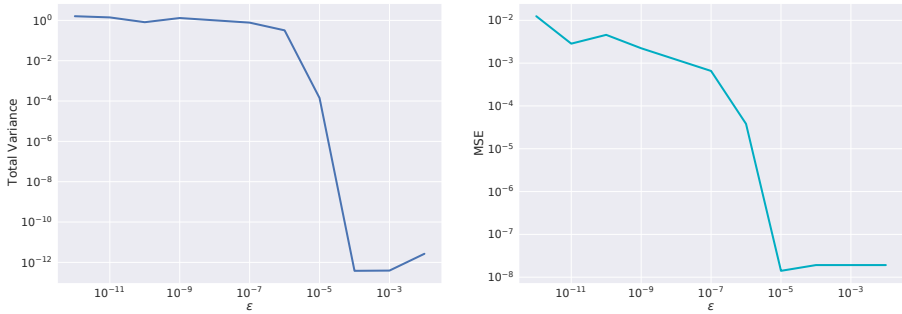


Figure 6.4: Total variance (left) and Mean Squared Error (right) of the dynamics distribution when DROPO is run with different ϵ values on $\mathcal{D}_{noisy}$ (experiment 6.2.2). Hyperparameter ϵ should be kept to the minimum possible value while ensuring stability on the converged means and on the covariance estimation. In the above plots, $\epsilon = 10^{-5}$ leads to a significant drop in MSE, and has thus been selected.

6.2.6 Discussion

The sim-to-sim transfer scenario proved to be a simple yet effective way to assess the claims of DROPO. A successful identification of ground truth point-dynamics and distributions has been achieved consistently, demonstrating the efficient role of a probabilistic-based objective function in simple settings.

In addition, we found that DROPO was able to converge to dynamics distributions that are useful for training with domain randomization, when an unmodeled phenomenon in the source simulator is present. On average, better policy performances were observed with respect to our baselines DROID and UDR. Note, however, that none of these approaches are able to achieve the ground truth policy performance. This suggests that the current DROPO implementation may be further expanded to explicitly account for better transfer performances. For instance, methods such as ADR [40] or SPOTA [41] may be stacked on top of DROPO to improve the training process.

While satisfactory results have been obtained, we highlight the importance of tuning the hyperparameter ϵ when running DROPO. When tuned improperly, DROPO may lead to instabilities due to numerical errors and singular estimated covariance matrices. Therefore, ϵ should always be carefully adjusted to minimize the Mean Squared Error, similarly to the case of a standard fixed-effects regression analysis, up until a predefined threshold. More specifically, we infer the threshold by looking at Fig. 6.4 and identifying acceptable values. Moreover, as the target dataset is collected offline, we may also reproduce the target trajectories in simulation under different parameters ξ and visually examine the agent's behavior. While this approach has not been applied in this case, it turned out crucial for the sim-to-real experiments (Section 6.3). As a result of hyperparameter tuning, note that different ϵ values are reported across the experiment, with the particularly high value of $\epsilon = 10^{-3}$ in the case of optimization under a misspecified simulator.

For the sake of simplicity, λ has been kept to 1 throughout all sim-to-sim experiments, as no significant advantage to using higher values has been empirically found.

Finally, we point out that further experiments may be carried out by analyzing additional physical parameters in the scene. For example, the friction coefficient with the ground surface may also be optimized and randomized during training, together with the damping factor of each joint.

6.3 Sim-to-Real transfer

While some of the works in the field of domain randomization for robotic tasks underplay the importance of assessing their algorithms in sim-to-real transfer scenarios [42, 108], we further enhance the findings in Section 6.3 and apply DROPO

on two real robotic tasks: *sliding* and *pushing*. In particular, we denote:

- the *Hockeypuck* environment as the RL task associated with hitting a puck such that it stops on the desired target location;
- the *PandaPush* environment as the RL task associated with pushing a heavy box with shifted center of mass forward in a straight line;

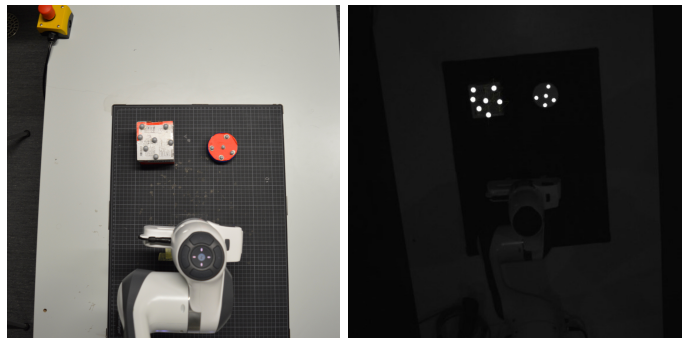
These experiments aim to answer important research questions on the ability of DROPO to solve real world robotic task: (i) is DROPO able to reliably optimize dynamics distribution on data collected on real-world setups affected by sensor noise and delay? is DROPO able to train policies with domain randomization on such distributions that generalize well to the real world?

6.3.1 Experimental setups

The two setups considered involve the 7-DoF Kuka LWR4+ and Franka Panda robotic arms. In both cases, the goal is to learn an optimal policy to control their behavior through closed-loop joint position controllers, rather than directly outputting torques. In order to get information on the configuration of the scene, the agents obtain their current joint positions with the robot's internal sensors, and are given information on the position and orientation of the objects in the scene (e.g. the hockey puck and heavy box) through motion capture systems. More precisely, we adopt OptiTrack to track each object in the scene through the attachment of reflective spherical markers (see Fig 6.5). The system uses a set of twelve cameras carefully positioned to capture the objects from multiple angles. An illustration of the OptiTrack setup used in the lab is shown in Fig 6.6.

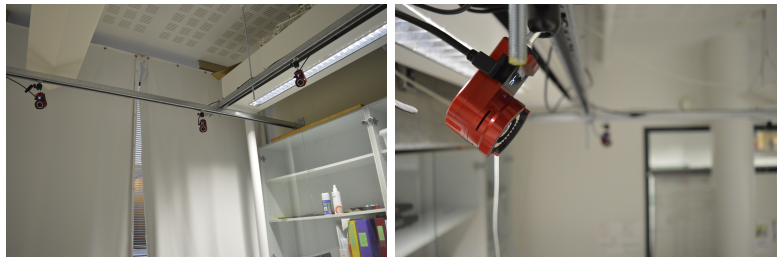
Hockeypuck environment In the HockeyPuck setup, the Kuka LWR4+ is equipped with a hockey stick and tasked with hitting a hockey puck such that it stops at the given target location. For this setup, we use an ice hockey puck and a whiteboard as a low-friction surface for the puck to slide on. The simulation environment is built in MuJoCo [95] to resemble the real-world setup. Both setups are shown in Figure 6.7. The yellow puck and the purple area in Figure 6.7a show the initial position of the puck and the range of possible goal positions.

In this setup, the actions are whole hitting trajectories, with a unified action vector of $17 * 7 = 119$ dimensions — 17 commands along the trajectory for 7 actuated joints. This is achieved by first training a variational autoencoder (VAE) on a range of task-specific trajectories, and later using the decoder as a trajectory generator, with actions given in the 2-dimensional latent space of the VAE, following [112, 113]. This simplifies the reinforcement learning problem by turning it into a contextual multi-armed bandit problem with continuous action space.



(a) (b)

Figure 6.5: Side-by-side comparison of the real view (a) of the PandaPush setup and the view through one of OptiTrack's cameras (b). The reflective markers attached on the heavy box and the hockeypuck are used to track the position of the object at a frequency of 120Hz, with a precision of 1mm.

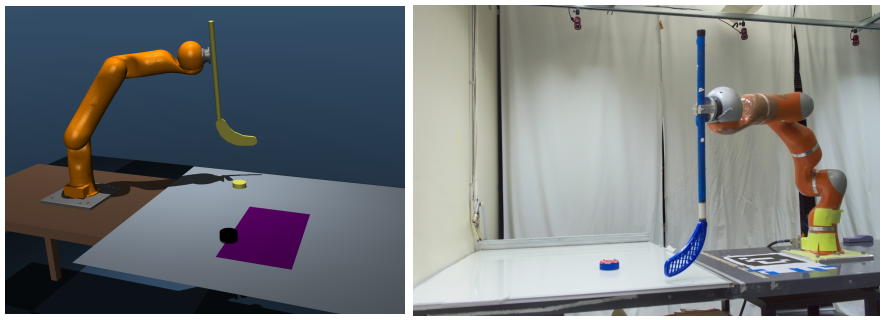


(a) (b)

Figure 6.6: Illustration of the positioning of some of the OptiTrack's cameras around the lab. The cameras are positioned such as to capture the reflective marker configurations from different angles.

The offline demonstrations for optimizing the parameter distribution in DROPO are thus obtained by rolling random trajectories 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 position of the hockey puck is obtained with OptiTrack in real time. During the post-processing, the data collected 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



(a) (b)

Figure 6.7: The hockeypuck setup in simulation (a) and in the real-world (b). The purple area in (a) shows the range of possible goal positions, while the black puck indicates the current goal.

and velocities of the robot joints and the hockey puck. The use of a VAE allows us to use original commanded positions as actions when replaying trajectories in the simulator, and — on this particular setup — removes the need of action inference from demonstrations.

The reward function designed for this environment only includes the distance d between the final position of the puck and the goal position. More precisely, the reward function is computed as $r(d) = -d^2 - \log(d^2 + 0.001)$.

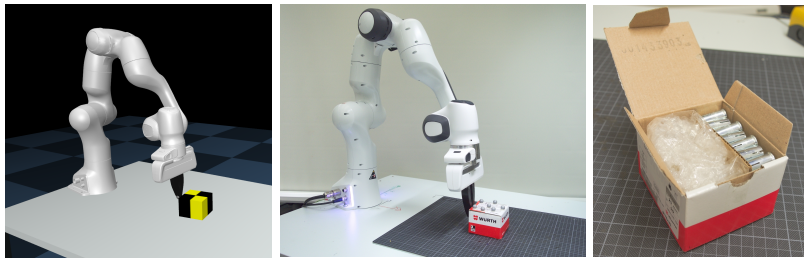
PandaPush environment We further evaluate DROPO on a pushing setup with the 7-DoF Franka Panda robot, shown in Figure 6.8. In this setup the goal of the robot is to push a heavy box forward in a straight line by 24cm. To make things challenging, the box's center of mass is shifted by filling it up with heavy steel bolts on one side and bubble wrap on the other, as shown in Figure 6.8c.

In contrast to the hockeypuck environment, data for the PandaPush setup can be collected by manually moving the robot around, also known as *kinesthetic guidance*. This allows us to directly control the robot's end effector such as to push the box and collect informative data to infer the shifted center of mass. The data is then preprocessed in a similar way to hockeypuck — we use robot's joint sensors to get the joint positions and OptiTrack to track the heavy box. Due to the offset center of mass, we additionally include the orientations and angular velocity of the box in the state space used for fitting DROPO, to allow maximum information of the scene.

The reward function for this environment includes a combination of multiple terms to penalize improper behavior of the robot, besides encouraging a lower distance between the box and the target location. In particular, a penalization term

is added for joint accelerations and velocities, and contacts between the robotic arm and the table. We remind that the whole training process still happens entirely in simulation, as only the final converged policy is deployed on the real hardware.

The PandaPush environment expects actions as joint accelerations, which are then integrated to get target joint velocities and positions. This formulation usually leads to higher stability while training reinforcement learning policies, since a potentially dangerous random action while exploring the environment would not cause abrupt movement in a single time instant. In addition, joint accelerations are easy to constrain under predefined safe bounds by scaling them with a sigmoid function.



(a) (b) (c)

Figure 6.8: The PandaPush setup in simulation (a) and in the real world (b), with the inside of the box shown in (c).

6.3.2 Sliding

We test DROPO's framework on the Hockeypuck environment for solving a sliding task. First, a dataset of offline trajectories is collected with a pretrained VAE, as reported in the setup description 6.3.1. The final dataset $\mathcal{D}$ contains 5 trajectories from the real setup, for a total of 3750 transitions. Later, $\mathcal{D}$ is processed and used by DROPO to optimize the physical parameters of the environment. In this setup, we adjust the following parameters ξ:

- Mass of the puck;
- Friction coefficient of the contact between the puck and the surface (separately for the x and y axes);
- The *timeconst* parameter of MuJoCo acting on the contact between the hockey stick and the hockeypuck. This parameter affects both the stiffness and damping of the contact pair in question, modeled as a non-linear spring-damper system (see Section 3.2);

- The proportional and derivative gains of the closed-loop joint position controller used in simulation;

The aforementioned dynamics parameters make up a total of 18 dimensions in the parameter vector ξ, and 36 in ϕ — means and variances. This way, we are able to test DROPO on a setup with a considerably higher number of optimized parameters, with respect to the sim-to-sim setting (Section 6.2).

Informed by the simulated behavior when reproducing offline trajectories, we applied a slight variation of the DROPO implementation to best work with the Hockeypuck setup. A significant discrepancy between the simulated and real joint position controller, plus a delay in the OptiTrack measurements, made it hard to reset the environment along the collected trajectories and obtain reliable behavior. In particular, resetting the environment while the hockey stick is about to hit the puck may largely increase the gap with the real-world behavior, as the hitting motion is perturbed with noisy puck positions and incomplete information on the internal parameters of the real position controller. To make the optimization problem more stable, for each transition s_{t+1} in $\mathcal{D}$ we reset the environment to the starting configuration (rather than s_t) and execute all intermediate actions $a_0, \dots, a_t$. This way, we make sure we always reset the environment the same way as it was on the target setup. Finally, the sample size parameter K is set to 60.

After the distribution fitting step, a policy is trained on the converged $p_{\phi^*}(\xi)$. We use PPO [53] to train our policies in simulation for all baselines. We adopt a fully connected structure for both the actor and critic networks, with two hidden layers of 128 neurons and hyperbolic tangent activation functions. In contrast to the Soft-Actor-Critic implementation, the critic network here only approximates the state value function $v_{\pi_\theta}(s)$, used to perform TD(0) steps as in Actor-Critic methods (see Section 2.1.2).

During training, random goals in the purple area of the Hockeypuck environment are sampled at each new episode and given as target locations.

			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
$\mathcal{D}$	DROPO $_{e=1e-2}$	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

Table 6.5: Optimized dynamics distributions on dataset $\mathcal{D}$ collected on the Hockeypuck environment. For the sake of clarity, we only report the converged mass m, friction coefficients f_x and f_y along the two axes, and timeconst parameter t_{const}.

The results of the full sim-to-real transfer in the Hockeypuck environment are

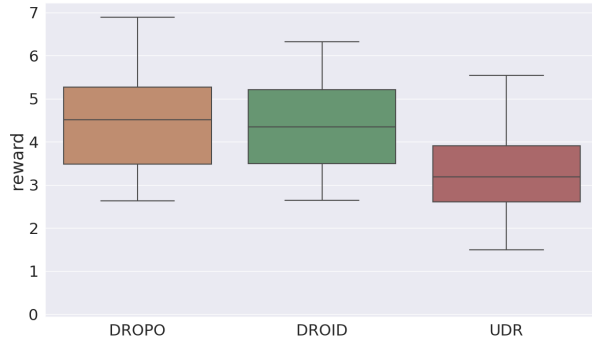


Figure 6.9: Sim-to-real performance on the Hockeypuck environment.

illustrated in Fig. 6.9. The figure reports the performances for DROPO and DROID as the average of the real-word reward obtained over three policies trained on the same converged dynamics distributions, displayed in Table 6.5. The UDR baseline is evaluated as described in Section 6.1, by averaging the performance of 10 policies trained on 10 randomly sampled uniform bounds. We roll-out evaluation trajectories in the real world by selecting 12 points positioned in a grid such as to cover the target purple area. We then use these points as target locations for each of the policies trained and compute the reward with the OptiTrack measurement of the puck position.

Overall, no significant difference in the rewards obtained by DROPO and DROID has been found, while randomly sampled uniform bounds make it hard for the policy to efficiently learn the task. A thorough discussion of the results is later reported in Section 6.3.4.

6.3.3 Pushing

DROPO is further compared to DROID and UDR in solving the pushing task on the Franka Panda robotic arm. Kinesthetic guidance of the robot is used to collect one trajectory from the real setup, where the box with shifted center of mass is moved around for about 10 seconds. The trajectory is then reproduced by the real robot using a joint position controller asked to follow the collected path. This step is needed to obtain the real target actions $a_t \in \mathcal{D}$, which would have to be otherwise inferred. The dataset is then preprocessed to synchronize each observation modality to the same frequency and sampling timesteps. $\mathcal{D}$ finally consists of 1050 transitions. Each state observation in the target dataset now

contains the arm's joint positions and velocities, the box position, orientation, velocity and angular velocity.

The distribution fitting step of DROPO for the pushing environment optimizes the following physical parameters ξ:

- Mass of the box;
- Friction coefficients along the x and y axes of the contact between the box and the ground surface;
- The center of mass of the box, both along the x and y axes;

A total of five dynamics parameters are therefore considered. Clearly, several other parameters are also involved in the simulation to model the real-world scene, such as the box's torsional friction and the friction coefficients regulating the contact between the robot's end-effector and the box. However, to avoid problematic correlations between dynamics and simplify the optimization problem we choose to manually set the remaining parameters involved. These are set to either their default values or with informed guesses by examining the simulator behavior when reproducing $\mathcal{D}$. In particular, an unrealistic tilting of the box in simulation has been solved by manually adjusting these dynamics. We point out that this visual inspection step comes with no additional engineering effort in DROPO's offline framework, as the implementation already requires to reproduce target data in simulation.

			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
$\mathcal{D}$	DROPO e=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

Table 6.6: Optimized dynamics distributions on dataset $\mathcal{D}$ collected on the PandaPush environment by kinesthetic guidance. The converged mass m, friction coefficients f_x and f_y, and box's center of mass com_x and com_y are reported.

Furthermore, note how we take a similar approach as in the Hockeypuck environment to deal with the unknown hidden parameters of the real joint position controller. This time, we simply avoid resetting the robotic arm during DROPO's execution, and run DROPO on the ordered offline transitions. The full state of the box is normally reset according to the current s_t. By doing so, the motion of the arm does not have to be reset to partially-known intermediate states. To further

account for observation noise, we set $\lambda = 10$. The sample size parameter K is set to 50.

As in the sliding task, we use PPO [53] to train policies for all baselines with the same policy structure described in Section 6.3.2. Each policy is now rolled-out on the real hardware five times for evaluation, measuring the performance as the distance of the box to the target location. Three policies on three different executions of DROPO — hence with potentially different dynamics distributions — are trained, and analogously for DROID. UDR is assessed the same way as throughout all our experiments (see Section 6.1). We report the results of one of the three executions of DROPO and DROID in Table 6.6, for the sake of simplicity. The policy performances are illustrated in Fig. 6.10.

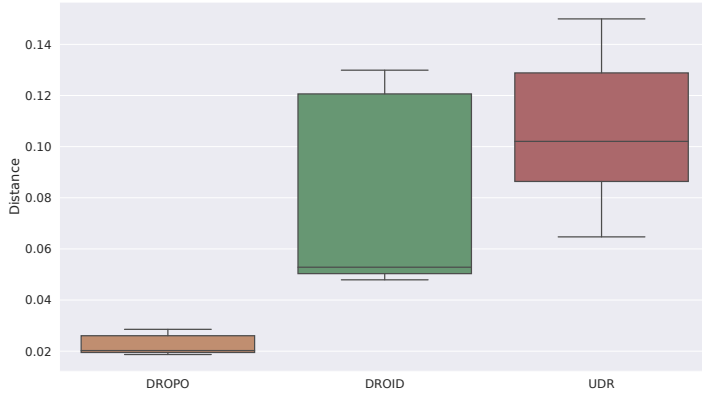


Figure 6.10: Sim-to-real performance on the PandaPush environment in terms of distance between the final box position and target location.

6.3.4 Discussion

The application of DROPO and domain randomization on the sliding and pushing sim-to-real transfer setups raised interesting findings. We thus report our conclusions on the results obtained.

Hockeypuck environment. The variation of DROPO's implementation described in the sliding setup has proven useful to overcome sensor delays and unknown hidden parameters in the joint position controller. Perhaps, this suggests that multiple implementations may be designed starting from the the default implementation of DROPO to best reproduce offline data in simulation, depending on the task.

An important finding on the performance of the Hockeypuck environment is the lack of significant evidence for a better-performing method between DROID and DROPO. We argue that this similarity arises due to the nature of the environment itself, as it likely does not benefit from domain randomization as much as other setups. Indeed, the bandit formulation coming from the trajectory generator constrains the reinforcement learning problem to operate within a well-defined, task-specific action space, where it can only produce valid hitting motions. Because of this, it's hard to think of a way to find a single robust action to overcome distributions over friction coefficients or puck masses. On the other hand, UDR is expectedly unable to solve the task with randomly sampled bounds, for the same reasons. Overall, system identification in the Hockeypuck environment is more important than domain randomization, due to the nature of the task.

However, we found that DROPO was able to adapt to this scenario and behave as a system identification method by converging to substantially narrow distributions, with a best ϵ value of 10^{-2}.

PandaPush environment. In contrast to the Hockeypuck setup, we found domain randomization to be crucial in solving the pushing task. DROPO was able to consistently outperform both our baselines on average. Note that to further inspect the variance of each method, the experiments have also been averaged on multiple executions of DROPO and DROID. Similarly to DROID, the UDR baseline is unable to solve the task with several randomly sampled uniform bounds. This suggests that in order to learn optimal behavior for the PandaPush environment both domain randomization and system identification need to be applied.

We found that DROPO is therefore able to converge to a distribution which can be used to train a policy that transfers well to the real setup, with the box stopping about 2cm away from the target location. In particular, the center of mass of the box has been correctly identified to be shifted by around 3cm from the center and corrected for by the learned policy when pushing the box forward. We point out that other physical parameters turned out to be harder to infer, such as the center of mass on the x-axis — the direction of movement of the box during the offline trajectory. To this regard, DROPO likely does not have enough information from $\mathcal{D}$ to come out with a reliable estimate of such parameter. However, for the same reason, the policy shows to be unaffected by a larger variance in these estimates over multiple DROPO executions. We believe that more complex implementations of DROPO may include the insertion of prior knowledge over the physical parameters such as to bias the optimization of parameters which are not possible to infer from $\mathcal{D}$. Finally, we show that a pushing task may be solved with minimal engineering effort in tuning domain randomization distributions and with simple feed-forward neural networks, in contrast to the prior work in [20] which uses recurrent structures to adapt policies on the target domain.

Chapter 7

Conclusions and future work

In this thesis we introduced DROPO, an offline-guided method for optimizing domain randomization distributions over the physical parameters of a simulator. The main premise of DROPO is to minimize human intervention and manual engineering during the process, by designing a safe and efficient framework for sim-to-real transfer in robotic tasks.

DROPO uses a maximum likelihood-based approach to adjust dynamics parameters such as to best reproduce real-world data. This way, domain randomization can be applied on distributions that resemble uncertainty over real-world physical parameters and cross the reality gap by learning robust policies.

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 the optimized distribution in simulation can be used to compensate for a misspecified value of a parameter, and how such a distribution can be used to train a well-performing reinforcement learning policy for the target domain.

When applied to real-world robotic setups, DROPO has successfully outperformed the recent method DROID [45] and the popular UDR approach. In particular, we tested DROPO on two robotic tasks: a sliding and a pushing task. We demonstrated that a policy trained with DROPO's converged distributions can successfully solve the task on both setups. More precisely, we discussed the nature of the two tasks and the relative importance of domain randomization and system identification. DROPO was able to find a satisfactory balance between the two techniques by converging to dynamics distributions — when these were needed — and adapting the parameters as in standard system identification at the same time.

Future work. In the current work, we used a variety of data collection strategies to collect the offline dataset—a partially-trained policy on the target task

(*Hopper*), random sliding trajectories (*Hockeypuck*) and human demonstrations obtained through kinesthetic teaching (*PandaPush*). We suggest that future work may investigate and evaluate how the data collection strategy impacts the accuracy of parameters obtained with DROPO. In particular, prior work on exploration in meta-learning would make an interesting extension [114, 115].

In order to adjust the value of ϵ, we used a simple procedure that minimizes the Mean Squared Error. While this worked well in our experiments and ensured stability in the likelihood computation, developing a way of learning ϵ would allow the method to automatically determine the amount of uncorrelated noise in the data. This would effectively encourage the method to capture as much variety as possible using dynamics randomization while modelling the rest with noise. Moreover, we believe that injecting noise in the training data according to the learned ϵ value would be a promising approach to stack on top of domain randomization for dynamics, as other works have similarly shown [20, 31, 19, 16].

We further propose to examine and expand DROPO to tasks with high-dimensional state spaces (e.g. images in deep reinforcement learning) or parameter spaces. Indeed, we believe that more complex implementations should be designed to estimate the likelihood function in unstructured high-dimensional spaces, with limited sample-size K. Moreover, our experiments are limited to a maximum of 18 randomized parameters. We encourage to test DROPO on tasks which include up to 100 optimized parameters.

Finally, we used the gradient-free optimization algorithm CMA-ES throughout our experiments. While convenient, the gradient-free optimization process adds a significant cost compared to gradient-based optimizers, which are capable of descending on the optimization cost without evaluating multiple parameter values (in this case, mean and variance of dynamics parameters). While this was not studied in this thesis, recent work on differentiable physics simulation [116] might allow for noticeable performance improvements over gradient-free optimization with MuJoCo.

Bibliography

[1] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. *Agent57: Outperforming the Atari Human Benchmark*. 2020. arXiv: 2003.13350 [cs.LG] (cit. on p. 1).

[2] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961 (cit. on p. 1).

[3] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. arXiv: 1912.06680 [cs.LG] (cit. on p. 2).

[4] Azalia Mirhoseini et al. *Chip Placement with Deep Reinforcement Learning*. 2020. arXiv: 2004.10746 [cs.LG] (cit. on p. 2).

[5] Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. *Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning*. 2021. arXiv: 2108.06266 [cs.RO] (cit. on pp. 3, 30).

[6] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. eprint: https://doi.org/10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721 (cit. on pp. 3, 27, 29, 33).

[7] Sridhar Mahadevan and Jonathan Connell. "Automatic programming of behavior-based robots using reinforcement learning". In: *Artificial Intelligence* 55.2 (1992), pp. 311–365. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(92)90058-6. URL: https://www.sciencedirect.com/science/article/pii/0004370292900586 (cit. on p. 3).

[8] V. Gullapalli, J.A. Franklin, and H. Benbrahim. "Acquiring robot skills via reinforcement learning". In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 13–24. DOI: 10.1109/37.257890 (cit. on p. 3).

[9] J.A. Bagnell and J.G. Schneider. "Autonomous helicopter control using reinforcement learning policy search methods". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 1615–1620 vol.2. DOI: 10.1109/ROBOT.2001.932842 (cit. on p. 3).

[10] Stefan Schaal. "Learning from Demonstration". In: *Advances in Neural Information Processing Systems*. Ed. by M. C. Mozer, M. Jordan, and T. Petsche. Vol. 9. MIT Press, 1997. URL: https://proceedings.neurips.cc/paper/1996/file/68d13cf26c4b4f4f932e3eff990093ba-Paper.pdf (cit. on pp. 3, 30).

[11] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2008.10.024. URL: https://www.sciencedirect.com/science/article/pii/S0921889008001772 (cit. on pp. 3, 30).

[12] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. "Robot Programming by Demonstration". In: Jan. 2008, pp. 1371–1394. DOI: 10.1007/978-3-540-30301-5_60 (cit. on pp. 3, 30).

[13] Todd Hester and Peter Stone. "TEXPLORE: Real-time sample-efficient reinforcement learning for robots". In: *Machine Learning* 90 (Mar. 2013). DOI: 10.1007/s10994-012-5322-7 (cit. on pp. 3, 29).

[14] Marc Deisenroth, Carl Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: June 2011. DOI: 10.15607/RSS.2011.VII.008 (cit. on pp. 3, 30).

[15] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. *Safe Model-based Reinforcement Learning with Stability Guarantees*. 2017. arXiv: 1705.08551 [stat.ML] (cit. on pp. 3, 30).

[16] Nick Jakobi, Phil Husbands, and Inman Harvey. ""Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics,"". In: vol. 929. Jan. 1995, pp. 704–720 (cit. on pp. 3, 4, 33, 42, 60, 77).

[17] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers". In: July 2010, pp. 119–126. DOI: 10.1145/1830483.1830505 (cit. on pp. 3, 33).

[18] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. "A Review of Physics Simulators for Robotic Applications". In: *IEEE Access* 9 (2021), pp. 51416–51431. DOI: 10.1109/ACCESS.2021.3068769 (cit. on p. 3).

[19] OpenAI et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv:1910.07113 [cs, stat]* (Oct. 15, 2019). arXiv: 1910.07113. URL: http://arxiv.org/abs/1910.07113 (visited on 08/02/2021) (cit. on pp. 3–5, 30, 31, 37, 40, 41, 77).

[20] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (May 2018), pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528. arXiv: 1710.06537. URL: http://arxiv.org/abs/1710.06537 (visited on 01/21/2021) (cit. on pp. 3, 5, 37, 39, 40, 56, 75, 77).

[21] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. *Reinforcement Learning for Pivoting Task*. 2017. arXiv: 1703.00472 [cs.RO] (cit. on pp. 3, 30, 37, 40).

[22] OpenAI et al. "Learning Dexterous In-Hand Manipulation". In: *arXiv:1808.00177 [cs, stat]* (Jan. 18, 2019). arXiv: 1808.00177. URL: http://arxiv.org/abs/1808.00177 (visited on 01/21/2021) (cit. on pp. 3, 5, 6, 27, 31, 37, 40, 42).

[23] Eugene Valassakis, Zihan Ding, and Edward Johns. "Crossing The Gap: A Deep Dive into Zero-Shot Sim-to-Real Transfer for Dynamics". In: (Aug. 15, 2020). arXiv: 2008.06686. URL: http://arxiv.org/abs/2008.06686 (visited on 01/21/2021) (cit. on pp. 4, 6, 34, 35, 38).

[24] M. NEUNERT, Thiago Boaventura, and J. BUCHLI. "WHY OFF-THE-SHELF PHYSICS SIMULATORS FAIL IN EVALUATING FEEDBACK CONTROLLER PERFORMANCE - A CASE STUDY FOR QUADRUPEDAL ROBOTS: Proceedings of the 19th International Conference on CLAWAR 2016". In: Oct. 2016, pp. 464–472. ISBN: 978-981-314-912-0. DOI: 10.1142/9789813149137_0055 (cit. on p. 4).

[25] L. G. Valiant. "A Theory of the Learnable". In: *Commun. ACM* 27.11 (Nov. 1984), pp. 1134–1142. ISSN: 0001-0782. DOI: 10.1145/1968.1972. URL: https://doi.org/10.1145/1968.1972 (cit. on pp. 5, 16).

[26] Shaojun Zhu, Andrew Kimmel, Kostas Bekris, and Abdeslam Boularias. "Model Identification via Physics Engines for Improved Policy Search". In: (Oct. 2017) (cit. on p. 5).

[27] Svetoslav Kolev and Emanuel Todorov. "Physically consistent state estimation and system identification for contacts". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids). Seoul, South Korea: IEEE, Nov. 2015, pp. 1036–1043. ISBN: 978-1-4799-6885-5. DOI: 10.1109/HUMANOIDS.2015.7363481. URL: http://ieeexplore.ieee.org/document/7363481/ (visited on 05/11/2021) (cit. on p. 5).

[28] Jie Tan, Zhaoming Xie, Byron Boots, and C. Karen Liu. "Simulation-based design of dynamic controllers for humanoid balancing". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 2729–2736. DOI: 10.1109/IROS.2016.7759424 (cit. on p. 5).

[29] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. "Using inaccurate models in reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning - ICML '06*. the 23rd international conference. Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 1–8. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143845. URL: http://portal.acm.org/citation.cfm?doid=1143844.1143845 (visited on 09/26/2021) (cit. on p. 5).

[30] Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. "Meta Reinforcement Learning for Sim-to-real Domain Adaptation". In: *arXiv:1909.12906 [cs]* (Sept. 16, 2019). arXiv: 1909.12906. URL: http://arxiv.org/abs/1909.12906 (visited on 09/29/2021) (cit. on pp. 5, 37, 40).

[31] Zihan Ding, Ya-Yen Tsai, Wang Wei Lee, and Bidan Huang. "Sim-to-Real Transfer for Robotic Manipulation with Tactile Sensory". In: *arXiv:2103.00410 [cs]* (Feb. 28, 2021). arXiv: 2103.00410. URL: http://arxiv.org/abs/2103.00410 (visited on 06/01/2021) (cit. on pp. 5, 35, 37, 40, 42, 60, 77).

[32] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". In: *arXiv:1804.10332 [cs]* (May 16, 2018). arXiv: 1804.10332. URL: http://arxiv.org/abs/1804.10332 (visited on 08/02/2021) (cit. on pp. 5, 31, 33, 37, 40, 56, 60).

[33] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *CoRR* abs/1703.06907 (2017). arXiv: 1703.06907. URL: http://arxiv.org/abs/1703.06907 (cit. on pp. 5, 17, 22, 33–35, 37, 40).

[34] Fereshteh Sadeghi and Sergey Levine. *CAD2RL: Real Single-Image Flight without a Single Real Image*. 2017. arXiv: 1611.04201 [cs.LG] (cit. on pp. 5, 37, 40).

[35] Stephen James, Andrew J. Davison, and Edward Johns. *Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task*. 2017. arXiv: 1707.02267 [cs.RO] (cit. on pp. 5, 37, 40).

[36] Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. "Auto-Tuned Sim-to-Real Transfer". In: *arXiv:2104.07662 [cs]* (May 20, 2021). arXiv: 2104.07662. URL: http://arxiv.org/abs/2104.07662 (visited on 08/03/2021) (cit. on pp. 5, 6, 40, 41).

[37] Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. "On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning". In: (Feb. 26, 2021). arXiv: 2102.13651. URL: http://arxiv.org/abs/2102.13651 (visited on 09/26/2021) (cit. on p. 6).

[38] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. "Emergent Tool Use From Multi-Agent Autocurricula". In: *arXiv:1909.07528 [cs, stat]* (Feb. 10, 2020). arXiv: 1909.07528. URL: http://arxiv.org/abs/1909.07528 (visited on 09/26/2021) (cit. on p. 6).

[39] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. *Augmenting Automated Game Testing with Deep Reinforcement Learning*. 2021. arXiv: 2103.15819 [cs.LG] (cit. on p. 6).

[40] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. "Active Domain Randomization". In: *arXiv:1904.04762 [cs]* (July 10, 2019). arXiv: 1904.04762. URL: http://arxiv.org/abs/1904.04762 (cit. on pp. 6, 42, 52, 66).

[41] Fabio Muratore, Michael Gienger, and Jan Peters. "Assessing Transferability from Simulation to Reality for Reinforcement Learning". In: *arXiv:1907.04685 [cs]* (Oct. 21, 2019). arXiv: 1907.04685. URL: http://arxiv.org/abs/1907.04685 (cit. on pp. 6, 42, 52, 66).

[42] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. "EPOpt: Learning Robust Neural Network Policies Using Model Ensembles". In: *arXiv:1610.01283 [cs]* (Mar. 3, 2017). arXiv: 1610.01283. URL: http://arxiv.org/abs/1610.01283 (visited on 01/21/2021) (cit. on pp. 6, 40, 57, 66).

[43] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience". In: *arXiv:1810.05687 [cs]* (Mar. 5, 2019). arXiv: 1810.05687. URL: http://arxiv.org/abs/1810.05687 (cit. on pp. 6, 15, 27, 41, 52).

[44] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. "Data-efficient Domain Randomization with Bayesian Optimization". In: *arXiv:2003.02471 [cs, stat]* (Jan. 5, 2021). arXiv: 2003.02471. URL: http://arxiv.org/abs/2003.02471 (visited on 01/21/2021) (cit. on pp. 6, 27, 41, 56).

[45] Ya-Yen Tsai, Hui Xu, Zihan Ding, Chong Zhang, Edward Johns, and Bidan Huang. "DROID: Minimizing the Reality Gap using Single-Shot Human Demonstration". In: *arXiv:2102.11003 [cs]* (Feb. 23, 2021). arXiv: 2102.11003. URL: http://arxiv.org/abs/2102.11003 (cit. on pp. 6, 27, 30, 41, 48, 49, 52, 56, 63, 76).

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0-262-19398-1. URL: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html (cit. on pp. 9, 13, 15).

[47] Marc Peter Deisenroth. "A Survey on Policy Search for Robotics". In: *Foundations and Trends in Robotics* 2.1 (2011), pp. 1–142. ISSN: 1935-8253, 1935-8261. DOI: 10.1561/2300000021. URL: http://www.nowpublishers.com/articles/foundations-and-trends-in-robotics/ROB-021 (visited on 05/13/2021) (cit. on p. 12).

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: (), p. 9 (cit. on p. 13).

[49] J. Peters. "Policy gradient methods". In: *Scholarpedia* 5.11 (2010). revision #137199, p. 3698. DOI: 10.4249/scholarpedia.3698 (cit. on p. 13).

[50] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: (), p. 28 (cit. on p. 14).

[51] Vijay Konda and John Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf (cit. on p. 15).

[52] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2016) (cit. on p. 15).

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (cit. on pp. 15, 50, 71, 74).

[54] Svetoslav Kolev and Emanuel Todorov. "Physically consistent state estimation and system identification for contacts". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 1036–1043. DOI: 10.1109/HUMANOIDS.2015.7363481 (cit. on pp. 15, 36).

[55] Stephan R. Richter, Hassan Abu Alhaija, and Vladlen Koltun. "Enhancing Photorealism Enhancement". In: *CoRR* abs/2105.04619 (2021). arXiv: 2105.04619. URL: https://arxiv.org/abs/2105.04619 (cit. on p. 17).

[56] Freha Mezzoudj, David Langlois, Denis Jouvet, and Abdelkader Benyettou. "Textual Data Selection for Language Modelling in the Scope of Automatic Speech Recognition". In: *Procedia Computer Science* 128 (2018), pp. 55–64. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.03.008. URL: https://www.sciencedirect.com/science/article/pii/S1877050918302217 (cit. on p. 17).

[57] Samuel Barrett, Matt E. Taylor, and Peter Stone. "Transfer Learning for Reinforcement Learning on a Physical Robot". In: *Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA)*. Toronto, Canada, May 2010 (cit. on pp. 17, 22).

[58] R. S. Woodworth and E. L. Thorndike. *The influence of improvement in one mental function upon the efficiency of other functions. (I)*. Jan. 1901. DOI: 10.1037/h0074898. URL: https://doi.org/10.1037/h0074898 (cit. on p. 18).

[59] David Perkins and Gavriel Salomon. "Transfer Of Learning". In: 11 (July 1999) (cit. on p. 18).

[60] S.J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359 (cit. on pp. 18, 19).

[61] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. "Domain-Adversarial Training of Neural Networks". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 2096–2030. ISSN: 1532-4435 (cit. on p. 19).

[62] Matthew E. Taylor and Peter Stone. "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *J. Mach. Learn. Res.* 10 (Dec. 2009), pp. 1633–1685. ISSN: 1532-4435 (cit. on p. 19).

[63] Alessandro Lazaric. "Transfer in Reinforcement Learning: A Framework and a Survey". In: *Reinforcement Learning*. 2012 (cit. on pp. 19, 20).

[64] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. "Transfer Learning in Deep Reinforcement Learning: A Survey". In: *CoRR* abs/2009.07888 (2020). arXiv: 2009.07888. URL: https://arxiv.org/abs/2009.07888 (cit. on p. 19).

[65] Alexander A. Sherstov and Peter Stone. "Improving Action Selection in MDP's via Knowledge Transfer". In: *Proceedings of the Twentieth National Conference on Artificial Intelligence*. July 2005 (cit. on p. 22).

[66] Caitlin Phillips. *Knowledge Transfer in Markov Decision Processes*. 2006. URL: https://www.cs.mcgill.ca/~martin/usrs/phillips.pdf (cit. on p. 22).

[67] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Transfer of Samples in Batch Reinforcement Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 544–551. ISBN: 9781605582054. DOI: 10.1145/1390156.1390225. URL: https://doi.org/10.1145/1390156.1390225 (cit. on p. 22).

[68] Nikolaus Hansen. "The CMA Evolution Strategy: A Comparing Review". In: vol. 192. June 2007, pp. 75–102. ISBN: 978-3-540-29006-3. DOI: 10.1007/3-540-32494-1_4 (cit. on pp. 23, 24, 50).

[69] Kai Zhu and Tao Zhang. "Deep reinforcement learning based mobile robot navigation: A review". In: *Tsinghua Science and Technology* 26.5 (2021), pp. 674–691. DOI: 10.26599/TST.2021.9010012 (cit. on p. 27).

[70] Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj, and Mahbube Ardani. *Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments*. 2020. arXiv: 2005.13857 [cs.RO] (cit. on p. 27).

[71] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. "Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning". In: *ACM Trans. Graph.* 35.4 (July 2016). ISSN: 0730-0301. DOI: 10.1145/2897824.2925881. URL: https://doi.org/10.1145/2897824.2925881 (cit. on pp. 27, 31).

[72] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. "DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning". In: *ACM Trans. Graph.* 36 (2017), 41:1–41:13 (cit. on pp. 27, 31).

[73] Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara. "Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation". In: *Robotics and Autonomous Systems* 112 (2019), pp. 72–83. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2018.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0921889018303245 (cit. on p. 27).

[74] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. "Apprenticeship Learning for Helicopter Control". In: *Commun. ACM* 52.7 (July 2009), pp. 97–105. ISSN: 0001-0782. DOI: 10.1145/1538788.1538812. URL: https://doi.org/10.1145/1538788.1538812 (cit. on pp. 27, 30).

[75] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. "Skill learning and task outcome prediction for manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. clmc. Shanghai, China, May 9-13, 2011. URL: http://www-clmc.usc.edu/publications/P/pastor-ICRA2011.pdf (cit. on p. 27).

[76] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN: 9780486428093 (cit. on p. 27).

[77] Adam Daniel Laud. "Theory and Application of Reward Shaping in Reinforcement Learning". AAI3130966. PhD thesis. USA, 2004 (cit. on p. 29).

[78] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. "Reinforcement Learning in Robotics: Applications and Real-World Challenges". In: *Robotics* 2.3 (2013), pp. 122–148. ISSN: 2218-6581. DOI: 10.3390/robotics2030122. URL: https://www.mdpi.com/2218-6581/2/3/122 (cit. on p. 29).

[79] Tengteng Zhang and Hongwei Mo. "Reinforcement learning for robot research: A comprehensive review and open issues". In: *International Journal of Advanced Robotic Systems* 18.3 (2021), p. 17298814211007305. DOI: 10.1177/17298814211007305. eprint: https://doi.org/10.1177/17298814211007305. URL: https://doi.org/10.1177/17298814211007305 (cit. on p. 29).

[80] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. *Challenges of Real-World Reinforcement Learning*. 2019. arXiv: 1904.12901 [cs.LG] (cit. on pp. 29, 43).

[81] Athanasios Polydoros and Lazaros Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics". In: *Journal of Intelligent Robotic Systems* 86 (May 2017), pp. 153–. DOI: 10.1007/s10846-017-0468-y (cit. on p. 29).

[82] Jeff G. Schneider. "Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning". In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. NIPS'96. Denver, Colorado: MIT Press, 1996, pp. 1047–1053 (cit. on p. 29).

[83] Marc Deisenroth and Carl Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search." In: Jan. 2011, pp. 465–472 (cit. on p. 30).

[84] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. *Safe Exploration in Continuous Action Spaces*. 2018. arXiv: 1801.08757 [cs.AI] (cit. on p. 30).

[85] Pieter Abbeel and Andrew Y. Ng. "Exploration and Apprenticeship Learning in Reinforcement Learning". In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: Association for Computing Machinery, 2005, pp. 1–8. ISBN: 1595931805. DOI: 10.1145/1102351.1102352. URL: https://doi.org/10.1145/1102351.1102352 (cit. on p. 30).

[86] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. *A Practical Approach to Insertion with Variable Socket Position Using Deep Reinforcement Learning*. 2018. arXiv: 1810.01531 [cs.RO] (cit. on p. 30).

[87] Błażej Osiński, Adam Jakubowski, Piotr Miłoś, Paweł Zięcina, Christopher Galias, Silviu Homoceanu, and Henryk Michalewski. *Simulation-based reinforcement learning for real-world autonomous driving*. 2020. arXiv: 1911.12905 [cs.LG] (cit. on p. 31).

[88] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. *Asymmetric Actor Critic for Image-Based Robot Learning*. 2017. arXiv: 1710.06542 [cs.RO] (cit. on p. 31).

[89] Joshua Tobin et al. *Domain Randomization and Generative Models for Robotic Grasping*. 2018. arXiv: 1710.06425 [cs.RO] (cit. on p. 31).

[90] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. arXiv: 1703.03400 [cs.LG] (cit. on p. 31).

[91] Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. *Sim-to-Real Robot Learning from Pixels with Progressive Nets*. 2018. arXiv: 1610.04286 [cs.RO] (cit. on p. 31).

[92] Sebastian Höfer et al. *Perspectives on Sim2Real Transfer for Robotics: A Summary of the R:SS 2020 Workshop*. 2020. arXiv: 2012.03806 [cs.RO] (cit. on p. 31).

[93] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. "A Review of Physics Simulators for Robotic Applications". In: *IEEE Access* 9 (2021), pp. 51416–51431. DOI: 10.1109/ACCESS.2021.3068769 (cit. on p. 31).

[94] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG] (cit. on pp. 32, 40, 55, 57, 58).

[95] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109 (cit. on pp. 31, 32, 67).

[96] Rodney Brooks. "Artificial Life and Real Robots". In: (June 1998) (cit. on p. 33).

[97] Richard Sutton, Anna Koop, and David Silver. "On the Role of Tracking in Stationary Environments". In: *Computing* (Jan. 2007). DOI: 10.1145/1273496.1273606 (cit. on p. 34).

[98] Jack Collins, David Howard, and Jürgen Leitner. *Quantifying the Reality Gap in Robotic Manipulation Tasks*. 2018. arXiv: 1811.01484 [cs.RO] (cit. on p. 34).

[99] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. "Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?" In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6670–6677. ISSN: 2377-3774. DOI: 10.1109/lra.2020.3013848. URL: http://dx.doi.org/10.1109/LRA.2020.3013848 (cit. on p. 34).

[100] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. *Robust Adversarial Reinforcement Learning*. 2017. arXiv: 1703.02702 [cs.LG] (cit. on p. 35).

[101] Josiah P. Hanna and Peter Stone. "Grounded Action Transformation for Robot Learning in Simulation". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 4931–4932 (cit. on p. 36).

[102] Adam Allevato, Elaine Schaertl Short, Mitch Pryor, and Andrea Thomaz. "Iterative residual tuning for system identification and sim-to-real robot learning". In: *Autonomous Robots* 44 (Sept. 2020). DOI: 10.1007/s10514-020-09925-w (cit. on p. 36).

[103] Adam Allevato, Elaine Schaertl Short, Mitch Pryor, and Andrea L. Thomaz. *TuneNet: One-Shot Residual Tuning for System Identification and Sim-to-Real Robot Task Transfer*. 2020. arXiv: 1907.11200 [cs.RO] (cit. on p. 36).

[104] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. "Sim-to-Real Transfer with Neural-Augmented Robot Simulation". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto. Vol. 87. Proceedings of Machine Learning Research. PMLR, 29–31 Oct 2018, pp. 817–828. URL: https://proceedings.mlr.press/v87/golemo18a.html (cit. on p. 36).

[105] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. *TossingBot: Learning to Throw Arbitrary Objects with Residual Physics*. 2020. arXiv: 1903.11239 [cs.RO] (cit. on p. 36).

[106] Quan Vuong, Sharad Vikram, Hao Su, Sicun Gao, and Henrik I. Christensen. *How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?* 2019. arXiv: 1903.11774 [cs.LG] (cit. on pp. 38, 40).

[107] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 39).

[108] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. *BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators*. 2019. arXiv: 1906.01728 [cs.RO] (cit. on pp. 40, 52, 66).

[109] Wenhao Yu, C. Karen Liu, and Greg Turk. *Policy Transfer with Strategy Optimization*. 2018. arXiv: 1810.05751 [cs.LG] (cit. on pp. 42, 52).

[110] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. "Adapting Deep Visuomotor Representations with Weak Pairwise Constraints". In: *arXiv:1511.07111 [cs]* (May 25, 2017). arXiv: 1511.07111. URL: http://arxiv.org/abs/1511.07111 (visited on 01/27/2021) (cit. on p. 56).

[111] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In: *ICML*. 2018. URL: http://dblp.uni-trier.de/db/conf/icml/icml2018.html#HaarnojaZAL18 (cit. on p. 62).

[112] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. "Deep Predictive Policy Training using Reinforcement Learning". In: *IROS*. 2017 (cit. on p. 67).

[113] Aleksi Hämäläinen, Karol Arndt, Ali Ghadirzadeh, and Ville Kyrki. "Affordance Learning for End-to-End Visuomotor Robot Control". In: *IROS*. 2019 (cit. on p. 67).

[114] Karol Arndt, Oliver Struckmeier, and Ville Kyrki. "Domain Curiosity: Learning Efficient Data Collection Strategies for Domain Adaptation". In: (2021) (cit. on p. 77).

[115] Jin Zhang, Jianhao Wang, Hao Hu, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. "Learn to Effectively Explore in Context-Based Meta-RL". In: *Arxiv*. 2020. arXiv: 2006.08170 [cs.AI] (cit. on p. 77).

[116] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. "End-to-End Differentiable Physics for Learning and Control". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/842424a1d0595b76ec4fa03c46e8d755-Paper.pdf (cit. on p. 77).