Tampere University

Tarek Ibrahim

GUIDED DOMAIN RANDOMIZATION WITH META REINFORCEMENT LEARNING

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Tapio Elomaa
Prof. Ville Kyrki
Oliver Struckmeier
June 2022

ABSTRACT

Tarek Ibrahim: Guided Domain Randomization with Meta Reinforcement Learning
Master of Science Thesis
Tampere University
Master's Programme in Information Technology - Robotics and Artificial Intelligence
June 2022

Reinforcement learning policies often need to be trained in simulations of the real environments, since training directly on the real agents can either be not feasible or expensive. When transferring those trained policies to the real agents, they often either fail completely or degrade significantly in performance, due to modelling deficiencies of the used simulators and/or mismatch of parameter values between these two domains. Domain randomization methods used in literature have attempted to solve this problem by training policies in environments whose parameters (or a subset of them) are sampled from ranges during training, usually resulting in robust policies. Despite these policies performing better upon transfer, robustness may still come at some cost of performance when compared to policies that could adapt to the current context. Therefore, this thesis proposes Meta Guided Domain Randomization methods where domain randomization techniques are combined with meta reinforcement learning algorithms instead of standard reinforcement learning ones, in order to train adaptive policies. This thesis also presents an attempt to analyze and improve the Active Domain Randomization algorithm - one of the popular guided domain randomization methods from the literature. The improved Active Domain Randomization algorithm is then compared to its proposed meta guided domain randomization counterpart in sim-to-sim experiments on one of MuJoCo® environments, and is shown to offer an improvement in average performance over the entire space of the sampled environments' parameters. Thesis code and materials are available from `https://github.com/Tarek-Ibrahim/msc-thesis`, and custom environments from `https://github.com/Tarek-Ibrahim/gym-custom`.

Keywords: Domain Randomization, Reinforcement Learning, Meta-Learning, Sim2Sim Transfer, Sim2Real Transfer, Transfer Learning

PREFACE

I would like to express my deepest gratitude and appreciation to everyone who contributed to the thesis either directly or indirectly, especially my supervisors Prof. Tapio Elomaa and Prof. Ville Kyrki for offering experience and vision in critical times, and my advisors Oliver Struckmeier and Karol Arndt for being incredibly generous with their time and subject knowledge, and providing invaluable technical and moral support every step along the way. Despite the thesis changing directions multiple times, and encountering numerous obstacles and uncertainties, they were always there to offer guidance and engage in discussions which provided immense help and profound insight.

Tampere, 1st June 2022

Tarek Ibrahim

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

$\mathcal{A}$	Action Space
N	Number of Randomized Parameters
r_{thr}	Environment-specific reward threshold
$\mathcal{S}$	State Space
α	Temperature (in a maximum-entropy reinforcement learning framework)
β	Agent's [meta] learning rate
δ	Adaptation step size of model-agnostic meta-learning algorithm
γ	Discount Rate
ν	Update step size of automatic domain randomization algorithm
τ	Trajectory
Ξ	Randomization/Configuration Space
$\mathop{\mathbb{E}}_{y}[X]$	Expectation function (of random variable X over y)
$U(a, b)$	Uniform distribution with lower limit a and upper limit b
A2C	Advantage Actor Critic
Active DR	Active Domain Randomization
Auto DR	Automatic Domain Randomization
DDPG	Deep Deterministic Policy Gradient
DR	Domain Randomization
DRL	Deep Reinforcement Learning
GDR	Guided Domain Randomization
MAML	Model-Agnostic Meta-Learning
MDP	Markov Decision Process
MES	Main Engine Strength
Meta-GDR	Meta Guided Domain Randomization
Meta-RL	Meta Reinforcement Learning
Meta-UDR	Meta Uniform Domain Randomization
PG	Policy Gradient

POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RARL	Robust Adversarial Reinforcement Learning
RL	Reinforcement Learning
SAC	Soft Actor Critic
SGD	Stochastic Gradient Decent
SVPG	Stein Variational Policy Gradient
TRPO	Trust Region Policy Optimization
UDR	Uniform Domain Randomization

1. INTRODUCTION

1.1 Motivation

Reinforcement learning (RL) is a main machine learning paradigm, where the goal is to find the sequence of actions (in the form of a plan or a policy) for an agent to execute in order to behave optimally with respect to a given task. Agents are often physical, such as robots, autonomous vehicles, engines, and so on, or sometimes virtual, such as video game characters or stock market bots. More precisely, the agent is the controller component of those entities [1]. A reinforcement learning task in this definition can be seen as being broadly comprised of two elements: the environment, which represents the subset of the world outside the agent with which it interacts, and a feedback signal, which is referred to as the reward function [2], [3]. It is primarily with respect to this reward function that the optimality is defined [2]. In particular, it refers to maximizing the expected total rewards - an evaluation metric known as performance [2]. Therefore, the ultimate objective is to learn a policy that maximizes the performance of the real, physical agents.

Training on physical agents directly is often either infeasible or expensive as it requires a large amount of interactions with the environment [4]. For this reason, training in simulations of the real environments then transferring the trained policy onto the real agents is a common practice in reinforcement learning-based controllers [4], [5]. The deficiencies of the used simulators in perfectly modelling the real agents and environments which they are supposed to represent, and the potential mismatch between the parameters of those two domains (i.e. simulation and reality), leads to the transferred policies significantly degrading in performance upon transfer, or even sometimes failing completely [6], [4]. This gap is known as the *reality gap* and the associated transfer problem is known as sim-to-real (Sim2Real) transfer.

To train for successful transfer, simulators can again be used in what is known as sim-to-sim (Sim2Sim) transfer. The parameters which are subject to being mismatched between the domains involved in the transfer process can be internal agent parameters or external environment parameters. Agent parameters can mismatch due to calibration issues, components malfunctioning, wear-and-tear, or parameters being perturbed or changing during operation, but it can also be that a single policy is desired to be used on different agents with differing parameters, for example to reduce maintenance costs [7]. Environment parameters external to the agent can also be mismatched due to them changing or being incorrectly identified.

Domain randomization (DR) which relies on training the agent in environment simulations whose parameters are sampled from ranges rather than being fixed, is one of the most prominent approaches utilized in literature to solve this problem of domain transfer [7], [8], [5], [9]. Compared to sampling uniformly from the parameter ranges, DR methods which utilize a more directed or guided sampling strategy are shown to improve the performance of the resulting policies [8], [10]. Nonetheless, DR methods usually tend to produce policies that are robust to changes in those ranges [4], and this robustness may still come at some cost of performance when compared to an adaptive one which is able to adapt to the current context. For robust policies resulting from uniform sampling, the superiority of adaptive policies has been demonstrated by examples from meta reinforcement learning (Meta-RL) [11], [6] - an approach for learning policies that are able to leverage prior experience from training on a set of tasks and quickly adapt it to new, different tasks from the same distribution [12].

The main objective of this thesis is, therefore, to study if the edge in performance which guided domain randomization methods provide over uniform sampling could be further increased by combining them with meta reinforcement learning to train adaptive policies rather than robust ones for domain transfer.

1.2 Notation & Terminology

The lines between the agent, the policy and environment definitions are not always clear [2]. In the analogy to control theory, the agent can be seen as the controller, the policy as the control law and the environment as everything else (actuators, system/plant, sensors, etc) [1]. Other times, the agent can be used to refer to the whole system along with its controller and actuators (e.g. the whole robot), and the environment to refer to the subset of the external world within which the system operates and interacts. For clarity, this thesis adopts the terminology of using the agent as an alias for the policy (and thus using them interchangeably), and the environment to refer to everything outside of the agent or policy.

Building on this definition, the term *domain* is used synonymously with the term *environment*, in the context of domain transfer [4]. In the same context, randomization (i.e. of domain parameters) is used to mean sampling from a given range, regardless of the specific sampling strategy used (i.e. whether it is random or not), in order to be consistent with its use in the literature, and even though *randomization* is better used to refer only to random (uniform) sampling. Furthermore, both domain randomization methods Active Domain Randomization [8] and Automatic Domain Randomization [10] used in this thesis were abbreviated as ADR in the papers where they were introduced. To avoid confusion, these two methods are differentiated in this thesis instead as Active DR and Auto DR, respectively.

A list of the symbols and abbreviations used throughout this thesis is provided in the List of Symbols and Abbreviations.

1.3 Thesis Outline

This thesis is structured as follows. Preliminaries about reinforcement learning, meta-learning and domain randomization are laid out in chapter 2, followed by a detailed dive into the specific methods used from each of those topics and a review of the related work from the literature in chapter 4. The problems addressed in this thesis and the main contributions are then introduced in chapter 5. Chapter 6 presents the proposed method, Meta Guided Domain Randomization (Meta-GDR), after which the experiments performed and their results are shown and analyzed in chapter 7. Finally, conclusions and ideas for future work are discussed in chapters 8 and 9, respectively.

2. BACKGROUND

2.1 Reinforcement Learning

As previewed in section 1.1, reinforcement learning is an approach to solve a particular type of optimization problems characterized by sequential decision making. In this formalism, the goal is to obtain an action selection mechanism that optimizes a given task. This interaction is visualized in figure 2.1.

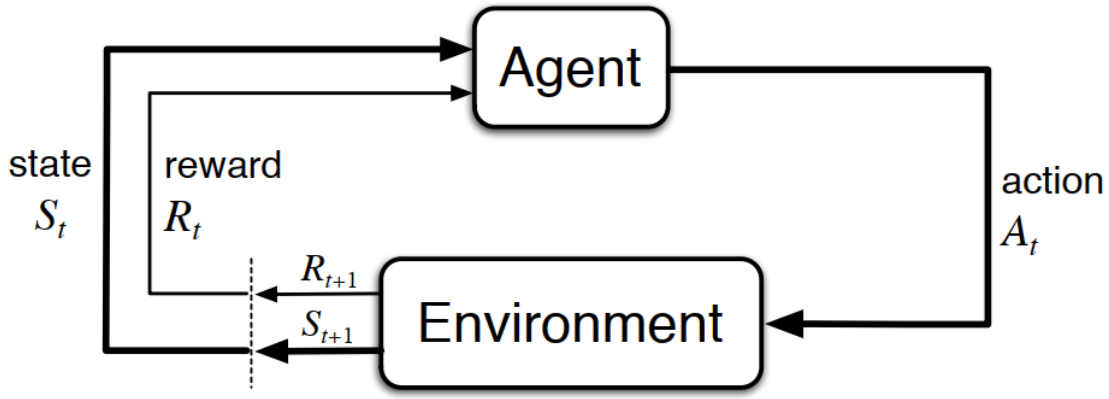


Figure 2.1. *The agent–environment interaction in a Markov decision process (from [2]).*

This action selection mechanism can either be obtained directly as a *policy*, or by relying on a representation of the environment transitions to form a *plan* or obtain a policy [2]. The agent then uses this mechanism to act in the environment it is in, in such a way that optimizes its expected *utility* which is given in terms of a defined feedback or reward signal [13].

Thus, an RL task ($\mathcal{T}$) can be defined as a Markov Decision Process (MDP or $\mathcal{M}$), which consists of an environment component ($\mathcal{E}$) and a feedback or reward component ($\mathcal{R}$) [3]:

$$\mathcal{T} or \mathcal{M} = (\mathcal{E}, \mathcal{R}) \tag{2.1}$$

Based on this definition, environment-based RL tasks are defined as RL tasks that share the same reward component and are differentiated from each other based on changes to the environment component. Similarly, reward-based RL tasks are defined as RL tasks sharing the

same environment component and distinguished from one another based on changes to the reward component.

The environment is defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition function (aka dynamics function) $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$, and an initial state function $\mathcal{P}_0 : \mapsto \mathcal{S}$ [2], [3].

$$\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{P}_0) \tag{2.2}$$

The state space $\mathcal{S}$ is the set of all possible states $s \in \mathbb{R}^{d_s}$ that the environment can be in, or representations of those states (known as observations) in case the desired states are not directly obtainable, and in which case the problem becomes a Partially Observable Markov Decision Process (POMDP) [2]. Similarly, the action space $\mathcal{A}$ is the set of all possible actions $a \in \mathbb{R}^{d_a}$ available for the agent to take. Generally speaking, not all actions are available in every state. Both the state space or action space can either be discrete or continuous. The state transition function $\mathcal{P}$, which describes the dynamics of the MDP, as well as the initial state function $\mathcal{P}_0$, are usually formulated as probabilities [2]:

$$\begin{aligned} p(s'|s,a) &\doteq P(S_{t+1} = s'|S_t = s, A_t = a) \\ p(s_0) &\doteq P(S_0 = s) \end{aligned}$$

Where the variables S, A and R are used to denote the associated random variables. While these functions can be deterministic in case the environment dynamics do not change with time and are fully known to the agent, it is often the case that environment dynamics are changing and/or not known from the agent's perspective.

Finally, the feedback signal is generated by what is known as the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, which can also be deterministic or probabilistic following the same rationale as the initial state and transition functions. In the general, probabilistic case, the reward is usually expressed as the expectation of the associated random variable R [2]:

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

Where $t = \{0, 1, 2 \dots T\}$ is used to denote the timesteps which define the sequence of the sequential decision making process. If the problem ends by reaching a terminal state, it is characterized as a finite-horizon MDP and the RL task as episodic, where T has a finite value [2]. On the other hand, if the problem does not have terminal states, it is characterized as being an infinite-horizon MDP and the RL task as being continuous, where T is ∞ [2].

Having described the components of the underlying MDP problem, it is next described how RL solves this problem. The goal of RL is to learn a sequential action selection mechanism or

strategy $\pi : \mathcal{S} \mapsto \mathcal{A}$ (representing a plan or policy) which optimizes a desired learning objective $J(\pi)$. If the function π can be parameterized by a set of parameters θ, it can be denoted as π_θ [2]. Similar to other functions, π_θ may also be deterministic or probabilistic (stochastic). In the latter, more general case, π_θ becomes a probability:

$$\pi_\theta(a_t|s_t) \doteq P(A_t = a|S_t = s)$$

from which actions are sampled as $a_t \sim \pi_\theta(a_t|s_t)$. In Deep Reinforcement Learning (DRL), policies are represented with neural networks and thus the parameters θ represent the network weights. Since the goal of RL is to optimize the expected, cumulative, long-term rewards, it usually defines the learning objective [2]:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\sum_{i=0}^{\infty} \gamma^i r(s_{t+i+1}, a_{t+i+1})]$$

where $\tau = (s_t, a_t, s_{t+1}, a_{t+1}, \ldots, s_T, a_T)$ represents the agent's trajectory under the policy π_θ, and $\gamma \in [0, 1]$ represents a discounting factor known as the discount rate [2]. The trajectory, also referred to as a *rollout*, is made up of sequential *transitions* $(s_t, a_t, r_{t+1}, s_{t+1})$. The discount rate represents the value of rewards far into the future with respect to more immediate ones, and is introduced so that infinite-horizon MDPs will have a finite-valued sum [2]. Finite-horizon MDPs are a special case where $\gamma = 1$ indicating that rewards at each timestep have equal weight [2]. The summation term is also known as the return G_t [2]:

$$G_t \doteq \sum_{i=0}^{\infty} \gamma^i r(s_{t+i+1}, a_{t+i+1})$$

Hence, the goal of DRL can be formulated as learning the optimal parameters θ^* which parameterize the policy π_{θ^*} that optimizes a given RL task. With a slight abuse of notation, $J(\theta)$ is taken to denote $J(\pi_\theta)$:

$$\theta^* = \mathop{\mathrm{argmax}}_{\theta} J(\theta) \tag{2.3}$$

RL differs from the other paradigms of machine learning in two major ways. While these approaches share the goal of solving optimization problems, RL is aimed as a solution for a particular subset characterized by sequential decision making where the system is not making isolated decisions (e.g. as in classification and regression problems) and where the decisions made at any given timestep can affect the future inputs or decisions [3]. The other major difference is that in RL the whole data is not already available and fed to the policy during training, but has to be actively collected by it [2]. This means that the agent, beside *exploiting* its current

knowledge to maximize its expected returns, has to exert efforts towards *exploring* the environment it is in, in order to collect more data to make more informed decisions that lead to a better optimization of its objective [2]. Therefore, at each timestep, the policy can choose to take an exploratory action which may sacrifice immediate rewards in hopes of obtaining better knowledge about the environment, or an exploitative action which aims to maximize its expected returns based on its current, possibly incomplete knowledge, in what is known as the *exploration-exploitation dilemma* [2]. Thus, the optimal policy in dynamic environments (i.e. environments that have non-stationary elements) usually is one which maintains some degree of exploration throughout the agent's lifetime.

In RL, available methods and algorithms can be categorized in two broad ways. First, if the algorithm relies on building a representation of the environment transitions (i.e. a dynamics *model*) which it then uses to formulate an optimal plan or learn a policy, it can be classified as a model-based algorithm [2]. On the other hand, if the algorithm learns a policy directly (i.e. without an intermediate model) it can be classified as a model-free algorithm [2]. Second, if the policy used for selecting actions (called behaviour policy) is the same as the policy being learned and optimized (called target policy), then the algorithm can be classified as an *on-policy* algorithm [2]. Otherwise, if the optimal target policy is learned from suboptimal (although continuously exploratory) behaviour, the algorithm can be classified as an *off-policy* algorithm [2]. This is not to say that on-policy algorithms do not perform exploration, as they often rely on a measure of entropy and/or on injected action noise and/or on controlling the policy distribution's variance in order to maintain exploration [14].

Examples of on-policy algorithms include Advantage Actor Critic (A2C) [15], Trust Region Policy Optimization (TRPO) [16] and Proximal Policy Optimization (PPO) [17]. Some popular off-policy algorithms are Deep Deterministic Policy Gradient (DDPG) [18] and Soft Actor Critic (SAC) [19]. In terms of sample efficiency, model-based algorithms are more efficient as they rely on an internal representation of the environment which limits interactions needed with the actual environment, as are off-policy algorithms as they can reuse transitions in training (through experience replay) since the order of transitions effectively does not matter [20], [2]. Figure 2.2 illustrates the anatomy of an RL algorithm in light of the above.

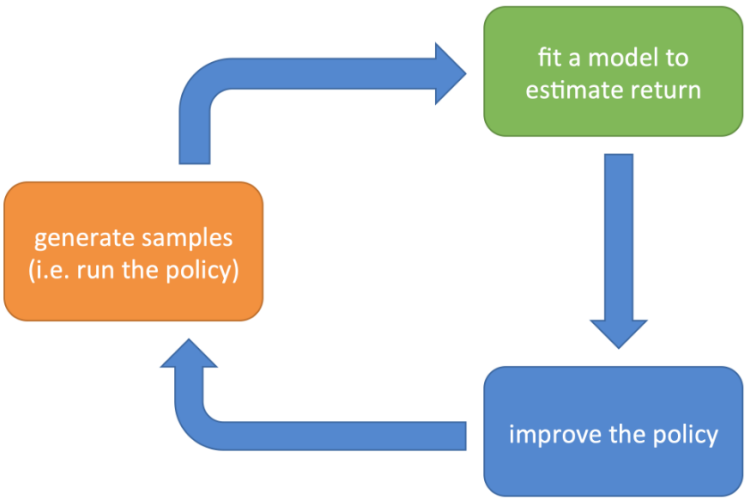


Figure 2.2. *The anatomy of RL algorithms (from [3]).*

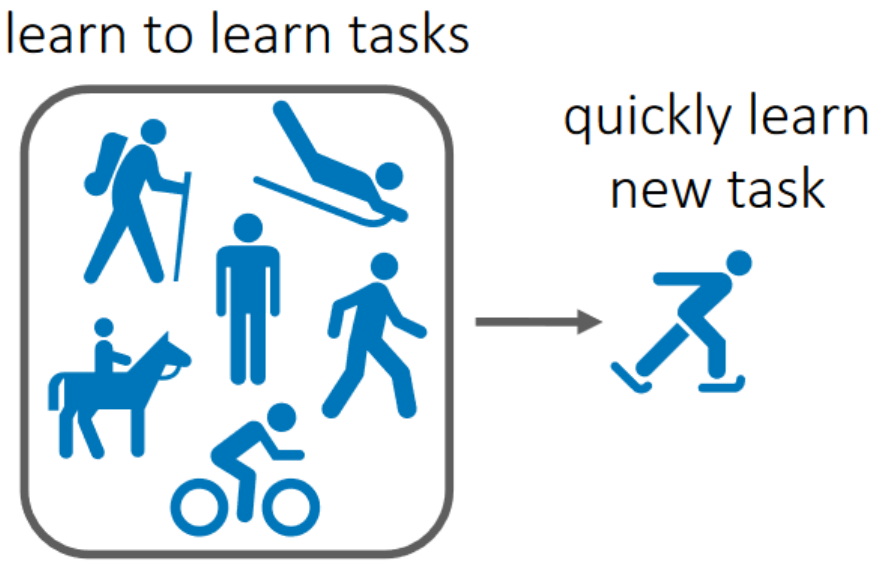


Figure 2.3. *Meta-learning illustration (from [21]).*

2.2 Meta-Learning

Also known as *"learning how to learn"* [22], meta-learning is a machine learning topic and a transfer learning method which has been gaining huge ground recently, both for its applicability to all machine learning paradigms and for being one of the major drivers of the research towards general artificial intelligence. Simply stated, the central idea behind meta-learning is in

leveraging prior experience from training on a set of tasks in order to learn new, different tasks more quickly and/or proficiently [12], as shown in figure 2.3. This can be contrasted to traditional machine learning where the model usually has to be retrained from scratch for every change in the given task, while also usually forgetting the previously seen tasks in the process. Within the context of reinforcement learning, the use of meta-learning is known as meta reinforcement learning (Meta-RL).

Algorithmically, meta-learning is comprised of two general phases: meta-training and meta-testing, each of which has its own training and testing phases [12]. In the meta-training phase, the algorithm optimizes the encapsulation of a certain knowledge related to the shared structure among the tasks (called meta-training tasks) into some representation θ (called meta-representation or meta-parameters if it is in parameter form), such that when presented a new, potentially different task (i) in the testing phase, the algorithm adapts the meta-representation (into θ_i') to optimize the task's objective. The output of the meta-training phase is the optimal meta-representation θ^* which is the input to the meta-testing phase. In the meta-testing phase, a new task is presented for which the algorithm adapts its optimal meta-representation in the training phase, to then optimize in the testing phase [23].

The critical assumption in meta-learning is that all the tasks involved share some underlying structure [12]. This assumption is often stated as a requirement that the testing tasks have to be drawn from the same distribution as the training tasks [12]. In practice, one way in which this translates to in Meta-RL is a limitation on the algorithm used for optimizing the meta-representation to be an on-policy algorithm.

In the taxonomy proposed in [23], meta-learning has three main aspects: the meta-representation, meta-optimizer and meta-objective. The meta-representation relates to the shared knowledge being learned from the meta-training tasks and can take many forms such as be the optimizer, network architecture, a hyperparameter, initial conditions (priors), loss or reward function, or an exploration policy. The meta-optimizer relates to *how* the meta-representation is being learned from the meta-training tasks. Methods that were used in the literature for this purpose are gradient-based optimization, reinforcement learning and evolutionary computing algorithms. Finally, the meta-objective relates to the goal behind carrying out the meta-learning, for instance improving sample efficiency, generalization, learning speed or asymptotic performance.

To obtain a formulation of the meta-learning problem, equation 2.3 can be generalized to other machine learning paradigms outside of RL by considering $J(\theta)$ to be a general objective function and restating the equation as:

$$\theta^* = \underset{\theta}{\operatorname{argopt}} J(\theta) \tag{2.4}$$

If $J(.)$ is a loss function $L(.)$, then optimization translates to a minimization procedure. Furthermore, if the subscript i is taken to denote entities related to task $\mathcal{T}_i$, and D^{tr},D^{ts} to denote

training and testing datasets, respectively, the meta-optimization can be formulated as [23]:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} [L_i(\theta_i^{'*}, D_i^{ts})] \tag{2.5}$$

This represents the outer loop of a meta-learning algorithm. Hence, the inner loop is responsible for adaptation (i.e. obtaining $\theta_i^{'*}$) [23]. The method by which $\theta_i^{'*}$ is obtained divides the meta-learning algorithms into three main categories of methods: black-box, optimization-based and non-parametric [12], [24]. The non-parametric family is the only one which has not yet been extended to RL and which replaces D_i^{ts} with labels y_i^{ts} and the parameters $\theta_i^{'*}$ with $\hat{y}_i^{ts} = f_\theta(D_i^{tr}, x^{ts})$, hence its name [24]. In the previous, f_θ is a similarity measure and x refers to the input samples [24]. Black-box methods are named in this way since they use a neural network or a black-box structure, f_θ, to obtain $\theta_i^{'*}$ [12]. More concretely, the inner loop for black-box methods can be stated as $\theta_i^{'*} = f_\theta(D_i^{tr})$ [12]. Lastly, optimization-based methods involve a bi-level optimization which means that the inner loop is also an optimization step [25]:

$$\theta_i^{'*} = \underset{\theta}{\operatorname{argmin}} L_i(\theta, D^{tr})$$
$$\theta_i^{'} \leftarrow \theta - \delta \nabla_\theta L_i(\theta, D_i^{tr})$$

2.3 Domain Randomization

Domain Randomization (DR) is one of the most popular approaches studied and used in literature for the purpose of addressing the problem of domain transfer. The problem of domain transfer can be defined as the problem of successfully transferring a policy trained in one domain (called a source domain) to an agent operating in a different domain (called the target domain) [4]. The source and target domains can be either the real environment or a simulation of it, with the most popular cases being transferring between simulation and reality (Sim2Real), and simulation and simulation (Sim2Sim). In this terminology, *domain* is an alias for the *environment* (as stated in section 1.2), and the measure of transfer success is with respect to performance (i.e. the expectation of the cumulative rewards). While traditional RL trains a policy on an environment with fixed parameters, in DR a subset of N domain parameters are sampled from their predefined ranges (i.e. randomized) during training in the source domain [4]. The working principle being that these ranges will likely cover the parameter values of the target domain [4], as illustrated in figure 2.4 for sim2real case as an example. Additionally, if the policy sees enough variations during training in the source domain, it will end up regarding the target domain as just another variation when transferred (i.e. its generalization capability to values within and outside those ranges improves) [4]. As a result, the chances of a successful transfer increase and the reality gap is narrowed.

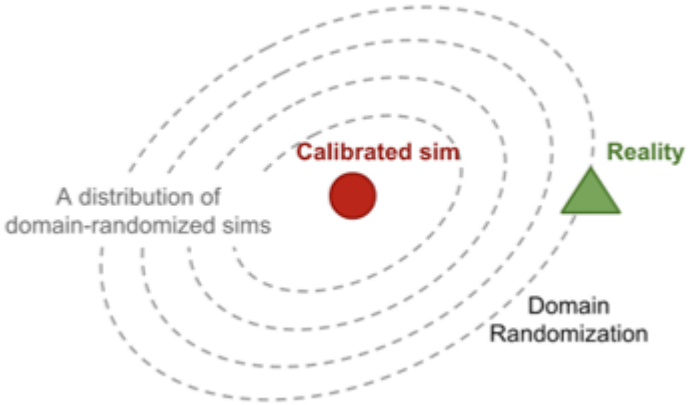


Figure 2.4. *Conceptual illustration of the domain randomization approach for sim2real transfer (from [4]).*

For a mathematical formulation of the problem, the source environment e_ξ is assumed to be parameterized by a set of parameters $\xi = \{\xi_i\}_{i=1}^{N} \subseteq \lambda$ that are accessible for randomization [4]. For clarity, λ is included in the definition to represent the set of parameters that fully parameterize the environment. Each combination of the randomized parameter values ξ, known as a *configuration*, defines a different *environment instance*, and the space containing all possible configurations is known as *the configuration space* Ξ (i.e. $\xi \in \Xi \subset \mathbb{R}^N$) [8]. In general, DR can be formulated, building on equation 2.3, as an optimization problem [26]:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \underset{\xi \sim \Xi}{\mathbb{E}}[J(\theta)] \tag{2.6}$$

In the problem of domain randomization, there are three main questions that follow:

1. How to choose the parameters to randomize?
2. How to define their ranges? and
3. How to sample from the collective configuration space made up by their ranges?

Most methods have so far focused on investigating the best sampling strategy, assuming the parameters and their ranges are known in advance. The first method introduced and the most vanilla involve sampling parameters uniformly from their ranges $\xi_i \sim U(\xi_i^{low}, \xi_i^{high})$. This method is known as Uniform Domain Randomization (UDR) [7] and is explained in further detail in section 4.1. One issue with uniform sampling is the underlying assumption that the values within the randomized parameters' ranges contribute equally to the performance of the resulting policy over all environment instances within the randomization space. This is not generally the case, as some configurations can be more challenging for the policy to learn than others, and thus will need to be sampled with a larger frequency during training [8]. Another issue stems from another implicit assumption of this method that the order in which the policy is presented with the different environment instances during training is irrelevant to its final performance over

the configuration space. However, the available literature behind the topic of curriculum learning suggests otherwise [27]. Not only does it show that optimizing the order of sampling improves performance, but also that, in some cases, disregarding the order may cause the trained policy to fail completely [28].

Guided Domain Randomization (GDR) is a term which is used to refer to the family of methods that employ a sampling strategy which is more directed than sampling uniformly from the configuration space [4]. Most notably, two GDR methods are explored in this thesis and discussed in more detail in section 4.2: Active Domain Randomization (Active DR) and Automatic Domain Randomization (Auto DR). Active DR formulates the DR (aka sampling) problem as its own optimization problem, leading to a bilevel optimization, whereas Auto DR trains a memory-augmented policy in an environment whose randomization parameters have gradually expanding ranges, leading to emergent meta-learning.

3. RELATED WORK

With access to some data from the target domain, *Domain Adaptation* (DA) methods represent a form of transfer learning where a policy trained in a different source domain can be updated to adapt to that target domain [29],[30]. In *Progressive Neural Networks*, the policy is trained mostly in simulation and then for a shorter time on the real agent with few added parameters to encourage fast transfer [31]. In [32], an inverse dynamics model learned from training over the real agent is used in simulation in combination with the source domain policy to train a policy for the target domain.

With little to no access to real data, *Domain Randomization* (DR) methods can train a policy almost entirely or entirely in simulation over different variations of the environment so that it becomes robust to a wide range of variations which the target domain is expected to be a part of [4],[10]. Since DR operates on environment-based RL tasks, these variations can be related to the dynamics function or other parameters of the environment. Randomizing dynamics has been studied in multiple works [5], [33],[34],[35].

Regarding environment parameter randomization, most research has assumed knowing what parameters to randomize and the bounds on their ranges, since this usually can be reasonably estimated based on domain knowledge, and have focused instead on how to sample from these ranges. Uniform Domain Randomization (UDR) [7], introduced the original, most basic form of DR in which the randomized parameters are uniformly sampled from their ranges.

Guided Domain Randomization (GDR) is an umbrella term for the DR algorithms that employ additional factors to make the sampling more guided or directed [4]. If the real data is accessible to a certain extent, they can be used for guidance, either as feedback from the target task to assist in learning which domain parameters to sample [36] or to assist in searching for the configuration ξ that leads to optimal ξ-conditioned policy [37], or as trajectories that can be used to find the configurations which minimize a given measure of discrepancy between simulation and real trajectories [38].

If access to real-world data is not possible, guidance can also come from within the simulator [4]. For example, active domain randomization (Active DR) [8] relies on the discrepancy between transitions in the randomized environment instances and a reference environment in simulation to determine which configuration regions to sample from next. In automatic domain randomization (Auto DR) [10], the agent policy is initially trained on a reference environment

instance until performance exceeds a certain threshold, after which parameter ranges are expanded gradually as the agent continues to exceed the predefined performance thresholds at the boundaries. In [9], robust policies result from training an adversary along with the policy to apply disrupting forces.

Current DR methods utilize standard RL algorithms to train the agents, usually leading to *robust* policies. In contrast to this *robustness*, *adaptiveness* is the central benefit of an RL topic that has enjoyed recent success, namely Meta-RL. A comprehensive survey of meta-learning including Meta-RL is provided by [23], where the different methods are categorized according to their meta-representations, meta-optimizers and meta-objectives. In the setting of environment-based RL tasks, model-based Meta-RL methods [6], [39] and model-assisted model-free methods [40] have been used in combination with uniform/random sampling from the given task distribution. On the other hand, model-free Meta-RL algorithms have been used in the setting of reward-based RL tasks. One such example is the model-agnostic meta-learning algorithm (MAML) [11] which encapsulates the knowledge from training on a set of meta-training tasks into priors on its policy's parameters, which can be quickly adapted to optimize performance on the given downstream task.

Even though in the original MAML paper the tasks are sampled uniformly, several task acquisition mechanisms have been proposed for the setting of reward-based RL tasks [41], [42], [43], [44]. Most notably, [41] combines MAML with Active DR to learn a curriculum of reward-based RL tasks based on discrepancies between pre-adaptation and post-adaptation trajectories. This thesis proposes the novel use of MAML in the setting of environment-based RL tasks combined with a guided task sampling strategy.

4. METHODS

4.1 Uniform Domain Randomization

The most basic parameter sampling strategy is to sample configurations uniformly from the configuration space. This method was introduced in [7] and reproduced in algorithm 1.

Algorithm 1 Uniform Domain Randomization

Inputs:
Ξ : Randomization/Configuration Space, S: Simulator, $\beta > 0$: learning rate
Initialize:
π_θ : agent policy
for each episode **do**
  for each $i = 1 \dots N$ **do** ▷ Sample parameters
    $\xi_i \sim U(\xi_i^{low}, \xi_i^{high})$
  end for
  for each ξ_i **do** ▷ Generate rollouts
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_\theta(.; E_i)$
    $D \leftarrow D \cup \tau_i$
  end for ▷ Gradient Updates
  Update $\theta \leftarrow \theta - \beta \nabla_\theta J(\pi_\theta)$
end for

As was explained in section 2.3, this method involves the implicit assumptions that the different configurations in the configuration space have equal weights in affecting the average performance of the policy over the entire space, and that the order in which the policy encounters each of the environment configurations is inconsequential to its average performance over that space.

4.2 Guided Domain Randomization

Unlike randomly sampling from the configuration space, any sampling strategy which is *directed* in some manner belong to a family of algorithms known as Guided Domain Randomization

(GDR) [4]. The subsequent subsections explains two of the GDR algorithms studied in this thesis, namely Active Domain Randomization Active DR and Automatic Domain Randomization Auto DR.

4.2.1 Active Domain Randomization

The primary GDR method studied in this thesis is Active Domain Randomization, first presented in [8] and the schematic of which is reproduced in figure 4.1.

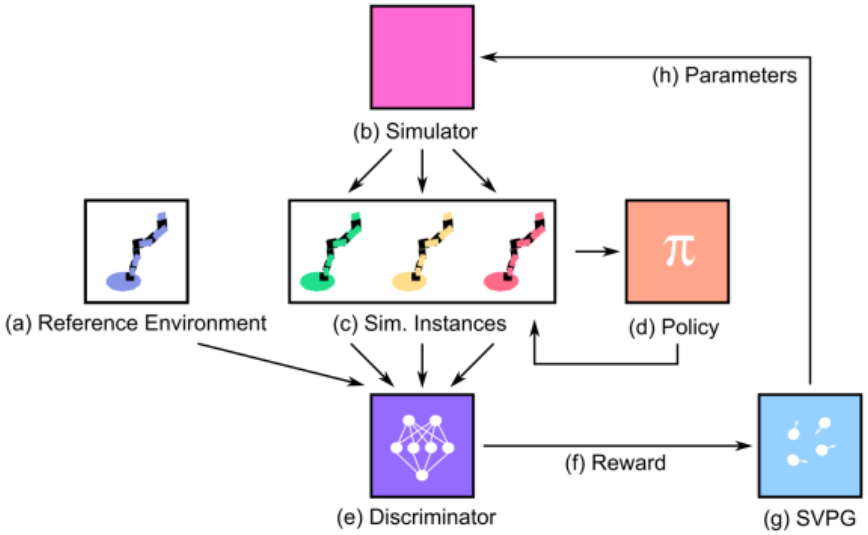


Figure 4.1. *Active Domain Randomization Method Schematic (from [8]).*

This method formulates sampling from the domain parameters as its own RL optimization problem. For comparison, UDR is only composed of components (b), (c) and (d), while (h) is uniformly sampled from the configuration space. In Active DR, however, the standard RL agent optimization is encapsulated within an outer optimization representing the DR process. To optimize for the parameters sampled in each iteration, a DR policy, (g), acts to choose the parameters that maximize the rewards, (f), calculated by a DR reward function, (e). The working principle being that the DR reward function determines the regions of the configuration space the agent policy is currently finding difficult (i.e. currently performing poorly on) and outputs rewards that is aimed at driving the DR policy towards sampling those regions.

In the DR optimization, the states were chosen to be the configurations ξ and the actions to be the changes to those configurations $\Delta\xi$, such that the next sampled configuration $\xi' = \xi + \Delta\xi$. The DR policy, μ_ϕ, which represents the sampling strategy of the Active DR method, was originally chosen to be the Stein Variational Policy Gradient (SVPG) algorithm. Since this is essentially just another RL policy, it is co-trained along with the agent's policy, π_θ, in what the original paper view as the Active DR algorithm actively updating its sampling strategy during

the training process. SVPG, an algorithm first introduced in [45], is essentially an ensemble of PG policies (referred to as particles) communicating via kernel updates to optimize the trade-off between exploration and exploitation. The underlying PG policy of the SVPG algorithm was originally chosen to be A2C in the Active DR paper. The actions are obtained from this policy as $\Delta\xi \sim \mu_\phi$.

The DR reward function was originally chosen to be based on a discriminator network, D_ψ, which is trained to differentiate between rollouts of the agent's policy, π_θ, in the randomized environment instances and the reference environment, (a). The reference environment is an environment with a fixed, known configuration, ξ_{ref}, which represents calibrated values of the randomized parameters, and which the agent policy does not train on. In its essence, the discriminator network is a binary classifier which classifies a given batch of environment transitions as belonging to the randomized environment instances or the reference environment. The core idea being that as the agent improves on the difficult randomized environment instances, the discriminator will find it increasingly difficult to differentiate between the agent's rollouts in them and in the reference environment, thus leading the discriminator to start giving lower rewards in favor of other instances that start to be difficult for the agent. In other words, the easiness to discriminate between those two types of rollouts is used as a proxy for measuring the difficulty which the agent policy is experiencing in performing in the randomized instances.

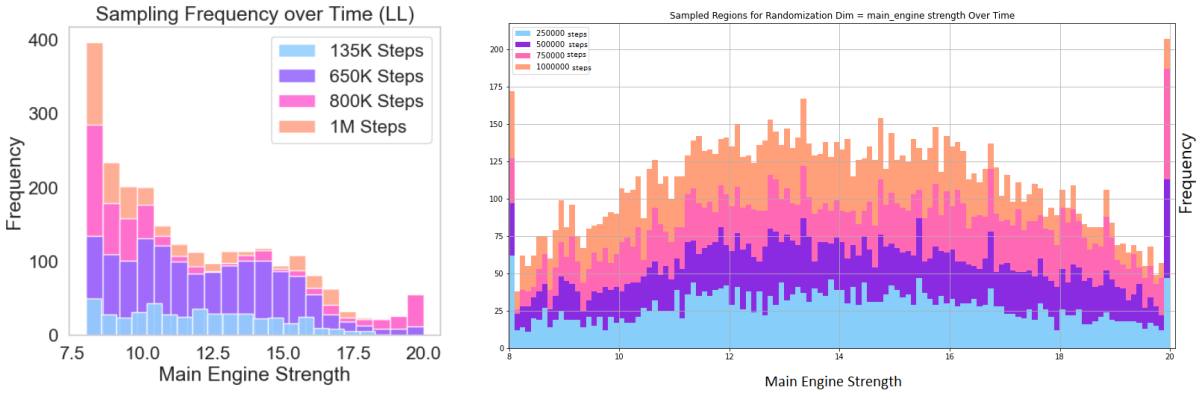
From the above, the DR optimization problem solved by Active DR can be characterized as an N-dimensional optimization problem with non-stationary rewards. The Active DR algorithm shown in algorithm 2 is adapted from [8] with slight modifications. Mainly, the original algorithm encompasses all the parts not colored in blue, the $DR\_reward\_function(.,.)$ being a function of the discriminator network, D_ψ, and the DR policy being the A2C-based SVPG particles.

The active sampling strategy resulting from this algorithm can then be understood to be distinguished from uniform sampling (i.e. UDR) in not only its assignment of different sampling frequencies to different configurations in the configuration space depending on the challenge they pose to the agent's policy, but also in the priority it gives in sampling to regions that are constituting instantaneous difficulty for the agent at each instant of the training process.

Algorithm 2 Active Domain Randomization

Inputs:
Ξ : Randomization/Configuration Space, S: Simulator, ξ_{ref} : reference parameters, $\beta > 0$: learning rate, r_{thr} : reward threshold
Initialize:
π_θ : agent policy, μ_ϕ : DR policy, $E_{ref} \leftarrow S(\xi_{ref})$: reference environment, D_ψ : Discriminator
Pretrain $\pi_\theta(.; E_{ref})$ until rewards $\geq r_{thr}$
for each episode **do**
    for each worker/particle μ_ϕ **do** ▷ Rollout DR workers/particles
        rollout $\xi_i \sim \mu_\phi(.)$
    end for
    for each ξ_i **do** ▷ Generate rollouts
        $E_i \leftarrow S(\xi_i)$
        rollout $\tau_i \sim \pi_\theta(.; E_i)$, $\tau_{ref} \sim \pi_\theta(.; E_{ref})$
        $D \leftarrow D \cup \tau_i$
        $D_{ref} \leftarrow D_{ref} \cup \tau_{ref}$
    end for
    for each $\tau_i \in D$, $\tau_{ref} \in D_{ref}$ **do** ▷ Calculate DR rewards for each env
        $r_i = DR\_reward\_function(\tau_i, \tau_{ref})$
    end for ▷ Gradient Updates
    Update $\theta \leftarrow \theta - \beta \nabla_\theta J(\pi_\theta)$
    Update μ_ϕ
    Update D_ψ with τ_i and τ_{ref} using SGD
end for

Reproducing the results reported in the original Active DR paper was attempted in order to serve as a baseline in the analysis of the proposed methods which try to improve on GDR methods by building on them (i.e. combining them with meta-RL). However, these attempts could not reproduce the reported results. Specifically, for the *LunarLander-v2* environment (explained in section 7.1), the sampled regions for the randomized environment parameter, Main Engine Strength, is reported to have a greater frequency at the lower (presumably more difficult) values, as shown in figure 4.2a, while the actual sampled regions obtained with the original implementation were near-uniform, as shown in figure 4.2b. Accordingly, this resulted in performance curves that were arbitrarily better or worse than UDR.



(a) *Reported Results (from [8]).* ***(b)*** *Obtained Results.*

Figure 4.2. *LunarLander-v2 Main Engine Strength Sampled Regions*

Failure to reproduce the original results signals that any associated claims of Active DR being superior to UDR reported by the paper are accordingly unreliable. Beside having fixed some technical issues with the code of the original implementation and re-tuned some of the hyper-parameters to better fit the nature of the optimization problem it is solving (refer to appendix A for hyperparameters and reasoning behind their choices), this opened an avenue to analyze and potentially improve the Active DR method as a whole. The in-depth analysis of the Active DR method is stated as one of the thesis objectives in section 5.4 and carried out in section 7.2.

4.2.2 Automatic Domain Randomization

Automatic Domain Randomization, proposed in [10], is a relatively simple GDR method which proved effective in assisting a real robotic hand in solving a Rubik's cube under various perturbations, as reproduced in figure 4.3.

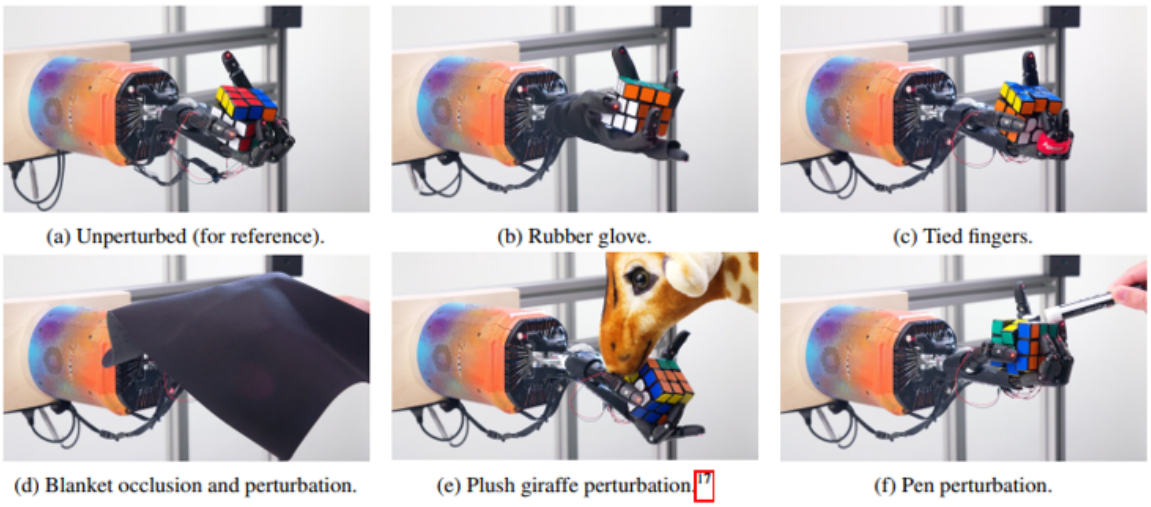


Figure 4.3. *Demonstration of the robustness of the trained policy on a myriad of example perturbations that were applied to the real robot hand while it solved the Rubik's cube (from [10]).*

In this method, the policy starts training in the environment with its default (i.e. calibrated) configuration. This means that the upper and lower bounds on the range of each randomization parameter start out as identical to one another and to the default/calibrated value of that parameter. Once the policy has exceeded a certain threshold representing a predefined number of consecutive successes, the upper and lower bounds on the randomized parameters start to gradually expand as the policy continue to exceed the higher threshold as it trains at the expanding boundaries. This process continues until the boundaries expand to cover the maximum predefined ranges of the configuration space, unless at some point the policy drops below a lower threshold representing a different, lower number of consecutive successes, then the bounds temporarily tighten again. Algorithm 3 presents this process in detail.

Algorithm 3 Automatic Domain Randomization

Inputs:
S: Simulator, ϕ_{ref} : reference (initial) parameters, $\beta > 0$: agent learning rate, r_{thr} : reward threshold, ν : Auto DR update step size, D : agent rollout buffer, $\{B_i^L, B_i^H\}_{i=1}^N$: Performance data buffers, m : Performance buffer length, t_L, t_H : decrease and increase thresholds, $\phi_i^{L,min}, \phi_i^{H,max}$: min and max boundary values
Initialize:
π_θ : agent policy, $\phi \leftarrow \phi_{ref}$
for each episode **do**
  $\xi \sim P_\phi$
  $P_b \sim U(0,1)$
  if $P_b < 0.5$ **then** ▷ Perform boundary sampling
    $x \sim U(0,1), i \sim U\{1,\ldots,N\}$
    if $x < 0.5$ **then** ▷ Select lower bound
      $B_i \leftarrow B_i^L, \xi_i \leftarrow \phi_i^L$
    else ▷ Select higher bound
      $B_i \leftarrow B_i^H, \xi_i \leftarrow \phi_i^H$
    end if
  end if
  $E_i \leftarrow S(\xi)$
  rollout $\tau_i \sim \pi_\theta(.; E_i)$ ▷ Generate rollouts
  $D \leftarrow D \cup \tau_i$
  if $P_b < 0.5$ **then**
    $p \leftarrow EvaluatePerformance(D)$
    $B_i \leftarrow B_i \cup \{p\}$
    if $Length(B_i) \geq m$ **then**
      $\bar{p} \leftarrow Average(B_i)$
      Clear(B_i)
      if $\bar{p} \geq t_H$ **then**
        Expand bound on ϕ_i by ν
      else if $\bar{p} \leq t_L$ **then**
        Tighten bound on ϕ_i by ν
      end if
    end if
  end if
  Update $\theta \leftarrow \theta - \beta \nabla_\theta J(\pi_\theta)$ ▷ Gradient Updates
end for

More concretely, the configuration ξ is sampled from a distribution P_ϕ parameterized by pa-

rameters ϕ. In the simplest case, P_ϕ can be chosen to be a uniform distribution (i.e. $P_\phi = \prod_{i=1}^{N} U(\phi_i^L, \phi_i^H)$), in which case the distribution parameters are the upper and lower bounds on the configuration parameters (i.e. $\phi^L = \xi^{low}$ and $\phi^H = \xi^{high}$). Initially, $\phi^L = \phi^H = \phi_{ref}$ until the boundaries start expanding, then they are maximally limited by $\phi_i^{L,min}$ and $\phi_i^{H,max}$, respectively (i.e. $\phi^{L,min} \leq \phi^L \leq \phi_{ref}$ and $\phi_{ref} \leq \phi^H \leq \phi^{H,max}$).

With a 50% probability, the agent rolls out the policy in the sampled configuration and stores the resulting trajectory in a buffer for the policy optimization step.

Otherwise, one of the randomization parameters is chosen at random and fixed to one of its boundary values, in what is called *boundary sampling*. The policy is then evaluated in the resulting environment instance and its performance p (representing the number of consecutive "successes") appended to the performance buffer B_i associated with the boundary-sampled parameter ξ_i and the type of boundary (i.e. whether it was the upper or lower boundary). If the number of entries in the performance buffer reach or exceed the buffer length m, they are then averaged and compared to the increase and decrease thresholds, t_H and t_L, respectively. These thresholds represent a certain number of consecutive "successes" that the average performance have to surpass or go under in order for the bound associated with that performance buffer to expand or tighten, respectively. The policy rollouts also in this case are stored for the policy optimization step.

This method was primarily included in the study in order to represent another GDR method to which the improved Active DR method can be compared and help contextualize its performance in a better way. However, there are a few limitations and adjustments which were made to this method that are worth mentioning in the following.

Firstly, this method was developed for and tested on the specific, custom example of solving the Rubik's cube by a robotic hand, so its effectiveness on MuJoCo environments or benchmark RL tasks remains unclear. Therefore, in order to adapt this method for use in these contexts, some modifications were applied to it. The number of consecutive successes p was redefined to be the number of workers/rollouts that exceed a defined reward threshold for that environment r_{thr}. Consequently, the increase and decrease thresholds, t_H and t_L, became thresholds on the number of workers/rollouts that exceed r_{thr}.

Secondly, the method relied on a recurrent policy in its RL algorithm to enable domain transfer via emergent meta-learning. The idea being that if a memory-augmented policy was exposed to a large set of variations, it will memorize only the general structure to those variations and this general knowledge can then adapted to the current context at test time, hence giving rise to a similar behaviour as meta-learning. In the experiments presented in this thesis, this was replaced with a non-recurrent policy, such that meta-learning becomes possible only through "true" meta-learning as Auto DR algorithm is combined with a Meta-RL algorithm, as is presented in chapter 6.

Finally, some hyperparameters were changed from what was reported in the original paper in order to fit the nature of the used environments, as reflected in appendix A.

This method is distinguished from uniform sampling by UDR in that it applies some notion of a curriculum in learning, as it gradually expands the bounds of the randomized parameters' sampling ranges as the agent succeeds on them. Implicit in this process is also the potentially unequal time the policy spends at each configuration as the bounds expand, since the algorithm keeps sampling the same configuration until the policy exceeds the thresholds for success on it.

4.3 Model-Agnostic Meta-Learning

Model-Agnostic Meta-Learning (MAML) is a seminal meta-learning and model-free, optimization-based Meta-RL method which was introduced in [11]. In this method, the meta-representation is priors on policy parameters θ learned from training over the meta-training tasks, so that they can be adapted quickly with one or more gradient steps to optimize the given downstream task $\mathcal{T}_i$ (i.e. $\theta \overset{adapt}{\rightarrow} \theta_i^*$), as reproduced in figure 4.4.

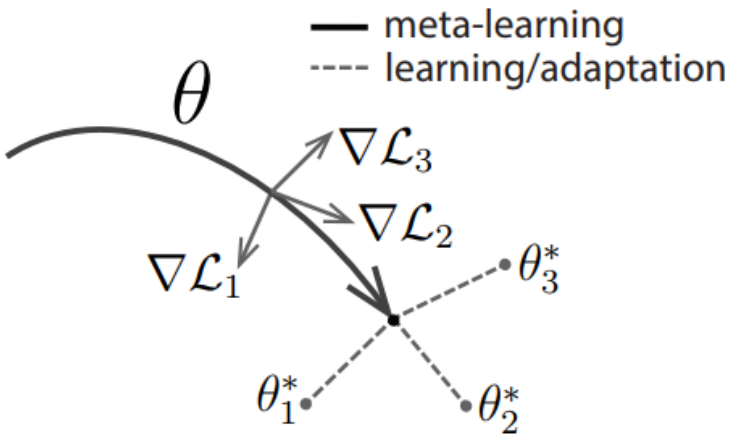


Figure 4.4. *Diagram of MAML (from [11]).*

Although MAML has been traditionally used with reward-based RL tasks, there is no theoretical limitation to use it with environment-based RL tasks instead, like it is done in this thesis, or even with both simultaneously, as long as sufficient shared structure is preserved between the tasks, as explained in section 2.2.

5. PROBLEM FORMULATION

5.1 Problem Statement

This thesis addresses the problem of improving the performance of the learned policies for domain transfer. More formally it can be stated that: given an RL task, a policy is trained in the source domain to behave optimally upon transfer to the target domain, given the parameters that are likely to mismatch between the two domains and their ranges.

To achieve this goal, guided domain randomization is used in combination with a meta-learning approach to extract useful priors from the intelligently-sampled environment configurations during training, which can be adapted to the current context in order to optimize the average performance over the entire configuration space.

5.2 Evaluation Metrics

From the optimization objective formulated in equation 2.6, the primary objective is the average performance of the agent's policy over the entire configuration space and thus it is chosen as the main evaluation metric in this thesis. This metric is computed according to equation 5.1.

$$\mathop{\mathbb{E}}_{\xi\sim\Xi}[\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\sum_{i=0}^{\infty}\gamma^i r(s_{t+i+1}, a_{t+i+1})]] \tag{5.1}$$

However, since the focus is not on improving the absolute performance of the policy in any given fixed environment instance, this metric can be better understood as *generalization*.

5.3 Assumptions & Limitations

The proposed methods in this thesis inherit the assumptions and limitations of their parent methods and the environments they are trained in:

1. MuJoCo® [46] environments are in general fully-observable with deterministic dynamics. However, the dynamics are unknown from the agent's perspective, the initial state function is probabilistic and the environments can be considered as partially-observable (i.e. observability is not a strict limitation) since states are a matter of definition and as neural

networks transform input states to different representations internally in any case.

2. Agent rewards are stationary, dense, and deterministic functions of the resulting state s'. Moreover, since the RL tasks are environment-based, the reward-based tasks are fixed to a single task per environment (i.e. generalization to multiple reward-based tasks is not considered within the scope of the thesis).
3. Domain randomization assumes that the source domain is parametrizable, at least in part, to a set of parameters which can be accessed for randomization during training [4]. In addition, the randomized parameters and bounds on their ranges are assumed to be known, leaving the focus to primarily be on the sampling strategy.
4. Domain randomization also presents an implicit assumption that the nature of the randomizations are such that they represent sufficiently limited perturbations or variations to the randomized domain parameters, preserving a notion of the domain's identity.
5. The central assumption in meta-learning is that the target task shall be drawn from the same distribution as the source tasks. This effectively means that the source and target tasks need to share some structure [12].

5.4 Research Questions & Main Contributions

Experiments presented in chapter 7 study the following research questions in order:

1. What are the issues with the Active DR algorithm? Can they be effectively analyzed and improved? Will an improved algorithm lead to improved policies compared to UDR?
    - Hypothesis: The working principle behind Active DR based on sampling configurations from the configuration space with a frequency proportional to how challenging the sampled configuration is for the trained policy makes intuitive sense as to why it can potentially improve on uniform sampling.
    - Contributions:
        - Analysis and improvement of the Active DR method.
        - Evaluating whether an improved Active DR improves over UDR.
2. Guided domain randomization so far has been used with standard RL algorithms to train a robust policy. Will combining it with a Meta-RL algorithm instead, to train an adaptable policy, yield better performance over the configuration space?
    - Hypothesis: Adaptive policies (learned by Meta-RL algorithms) can potentially provide some improvement over robust policies (learned by standard RL algorithms) in terms of performance over the configuration space, since the meta-parameters of the Meta-RL algorithms can be seen as comparable to the robust policies' parameters, with an additional fine-tuning step to adapt to the current context which can

yield an improved performance.

- Contributions: Replacing standard RL algorithms in GDR methods with Meta-RL algorithms (to learn adaptive policies instead of robust ones) and evaluating if they lead to improved domain transfer in comparison.

6. META GUIDED DOMAIN RANDOMIZATION

By combining MAML algorithm, explained in section 4.3, with the GDR algorithms discussed in section 4.2, the Meta-GDR methods are proposed. Algorithm 4 shows the combination of MAML with Active DR, while the combination of MAML with Auto DR is shown in algorithm 5. For completeness, the combination of MAML with UDR used in the experiments for comparison is given by algorithm 6. Although the default sampling strategy used traditionally with Meta-RL is uniform sampling, Meta-UDR is used here to denote the combination of model-free Meta-RL with UDR and to contrast with using the term Meta-GDR, as model-free Meta-RL has not yet been reported as being used in DR problems or with environment-based tasks.

As opposed to MAML which adapts to different reward-based RL tasks, the Meta-GDR and Meta-UDR algorithms adapt to different environment-based tasks which, in this thesis, amounts to the different environment instances E_i constructed from each sampled configuration ξ. Since the DR algorithms start by training the agent in a rollout of configurations (i.e. a sequence of environment instances), the meta-training tasks of the Meta-GDR and Meta-UDR algorithms are taken to be these environment instances, with the caveat that the sequential nature of these instances (i.e. the dependence of them on the previous ones), which is only relevant in the case of Active DR, is likely not accounted for by the underlying MAML implementation as it is designed to treat the meta-training tasks sampled each iteration as a batch instead with no particular accounting for their order.

By training over the batch of meta-training tasks, the policy encapsulates knowledge of the shared structure among the different environment instances as priors on its parameters, which are then adapted to optimize performance for the current environment instance. From this perspective, meta-parameters can be seen as comparable to the parameters of the robust policies that result from DR algorithms, and the advantage of Meta-GDR over GDR is then hypothesized to come from the additional adaptation step taken to further improve performance on any given configuration within the configuration space.

Algorithm 4 MAML with Active Domain Randomization

Inputs:
Ξ : Randomization/Configuration Space, S: Simulator, ξ_{ref} : reference parameters, $\delta > 0$: adaptation step size, $\beta > 0$: meta step size, r_{thr} : reward threshold
Initialize:
π_θ : agent policy, μ_ϕ : DR policy, $E_{ref} \leftarrow S(\xi_{ref})$: reference environment, D_ψ : Discriminator
Pretrain $\pi_\theta(.; E_{ref})$ until rewards $\geq r_{thr}$
for each episode **do**
  for each worker/particle μ_ϕ **do** ▷ Rollout DR workers/particles
    rollout $\xi_i \sim \mu_\phi(.)$
  end for
  for each ξ_i **do** ▷ Generate pre-adaptation rollouts in rand envs
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_\theta(.; E_i)$
    $D \leftarrow D \cup \tau_i$
  end for
  Evaluate $\nabla_\theta L(\pi_\theta)$ using D and L
  Compute adapted parameters with gradient decent: $\theta'_i = \theta - \delta \nabla_\theta L(\pi_\theta)$
  for each ξ_i **do** ▷ Generate post-adaptation rollouts in rand envs
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_{\theta'}(.; E_i)$, $\tau_{ref} \sim \pi_{\theta'}(.; E_{ref})$
    $D' \leftarrow D' \cup \tau_i$
    $D_{ref} \leftarrow D_{ref} \cup \tau_{ref}$
  end for
  for each $\tau_i \in D'$, $\tau_{ref} \in D_{ref}$ **do** ▷ Calculate DR rewards for each env
    $r_i = DR\_reward\_function(\tau_i, \tau_{ref})$
  end for ▷ Gradient Updates
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_i L(\pi_{\theta'_i})$ using each D'_i and L
  Update μ_ϕ
  Update D_ψ with τ_i and τ_{ref} using SGD
end for

Algorithm 5 MAML with Automatic Domain Randomization

Inputs:
S: Simulator, ϕ_{ref} : reference (initial) parameters, $\delta > 0$: adaptation step size, $\beta > 0$: meta step size, r_{thr} : reward threshold, ν : Auto DR update step size, D, D' : agent rollout buffers, $\{B_i^L, B_i^H\}_{i=1}^N$: Performance data buffers, m : Performance buffer length, t_L, t_H : decrease and increase thresholds, $\phi_i^{L,min}, \phi_i^{H,max}$: min and max boundary values
Initialize:
π_θ : agent policy, $\phi \leftarrow \phi_{ref}$
for each episode **do**
  $\xi \sim P_\phi$
  $P_b \sim U(0,1)$
  if $P_b < 0.5$ **then** ▷ Perform boundary sampling
    $x \sim U(0,1)$, $i \sim U\{1,\ldots,N\}$
    if $x < 0.5$ **then** ▷ Select lower bound
      $B_i \leftarrow B_i^L$, $\xi_i \leftarrow \phi_i^L$
    else ▷ Select higher bound
      $B_i \leftarrow B_i^H$, $\xi_i \leftarrow \phi_i^H$
    end if
  end if
  $E_i \leftarrow S(\xi)$
  rollout $\tau_i \sim \pi_\theta(.; E_i)$ ▷ Generate pre-adaptation rollouts in rand envs
  $D \leftarrow D \cup \tau_i$
  Evaluate $\nabla_\theta L(\pi_\theta)$ using D and L
  Compute adapted parameters with gradient decent: $\theta_i' = \theta - \delta \nabla_\theta L(\pi_\theta)$
  rollout $\tau_i \sim \pi_{\theta'}(.; E_i)$ ▷ Generate post-adaptation rollouts in rand envs
  $D' \leftarrow D' \cup \tau_i$
  if $P_b < 0.5$ **then**
    $p \leftarrow EvaluatePerformance(D')$
    $B_i \leftarrow B_i \cup \{p\}$
    if $Length(B_i) \geq m$ **then**
      $\bar{p} \leftarrow Average(B_i)$
      Clear(B_i)
      if $\bar{p} \geq t_H$ **then**
        Expand bound on ϕ_i with ν
      else if $\bar{p} \leq t_L$ **then**
        Tighten bound on ϕ_i with ν
      end if
    end if
  end if
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_i L(\pi_{\theta_i'})$ using each D_i' and L ▷ Gradient Updates
end for

Algorithm 6 MAML with Uniform Domain Randomization

Inputs:
Ξ : Randomization/Configuration Space, S: Simulator, $\delta > 0$:
adaptation step size, $\beta > 0$: meta step size
Initialize:
π_θ : agent policy
for each episode **do**
  for each $i = 1 \dots N$ **do** ▷ Sample parameters
    $\xi_i \sim U(\xi_i^{low}, \xi_i^{high})$
  end for
  for each ξ_i **do** ▷ Generate pre-adaptation rollouts in rand envs
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_\theta(.; E_i)$
    $D \leftarrow D \cup \tau_i$
  end for
  Evaluate $\nabla_\theta L(\pi_\theta)$ using D and L
  Compute adapted parameters with gradient decent: $\theta_i' = \theta - \delta \nabla_\theta L(\pi_\theta)$
  for each ξ_i **do** ▷ Generate post-adaptation rollouts in rand envs
    $E_i \leftarrow S(\xi_i)$
    rollout $\tau_i \sim \pi_{\theta'}(.; E_i)$, $\tau_{ref} \sim \pi_{\theta'}(.; E_{ref})$
    $D' \leftarrow D' \cup \tau_i$
  end for ▷ Gradient Updates
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_i L(\pi_{\theta_i'})$ using each D_i' and L
end for

7. EXPERIMENTS

7.1 Experimental Plan & Tools

Experiments in this thesis are sim2sim transfer experiments conducted in one of the environments of the MuJoCo® physics engine [46]. Python programming language was used in development with PyTorch as the main machine learning framework.

There are two main experiments presented in this chapter, as illustrated in figure 7.1, corresponding to each of the research questions studied in the thesis and discussed in section 5.4. In the first set of experiments, the Active DR algorithm is analyzed and improvements made to it are compared to a working version of the original implementation. The *best* Active DR algorithm resulting from this first set of experiments is then used in the second set, which primarily compares Active DR to the proposed meta-Active DR algorithm as a representative of the Meta-GDR family.

Because the original Auto DR algorithm relied on emergent meta-learning, it cannot be determined for certain if the resulting policy was entirely robust or adaptive. Therefore, the proposed meta-Auto DR algorithm which relies instead on "true" meta learning cannot be compared to the original algorithm to study the differences between robust and adaptive policies. Instead, the meta-Auto DR algorithm is used in the experiments of this chapter to help contextualize the performance of the Active DR and meta-Active DR algorithms.

In all of these experiments, the environment used was *LunarLander-v2* with the randomization parameter being the main engine strength.

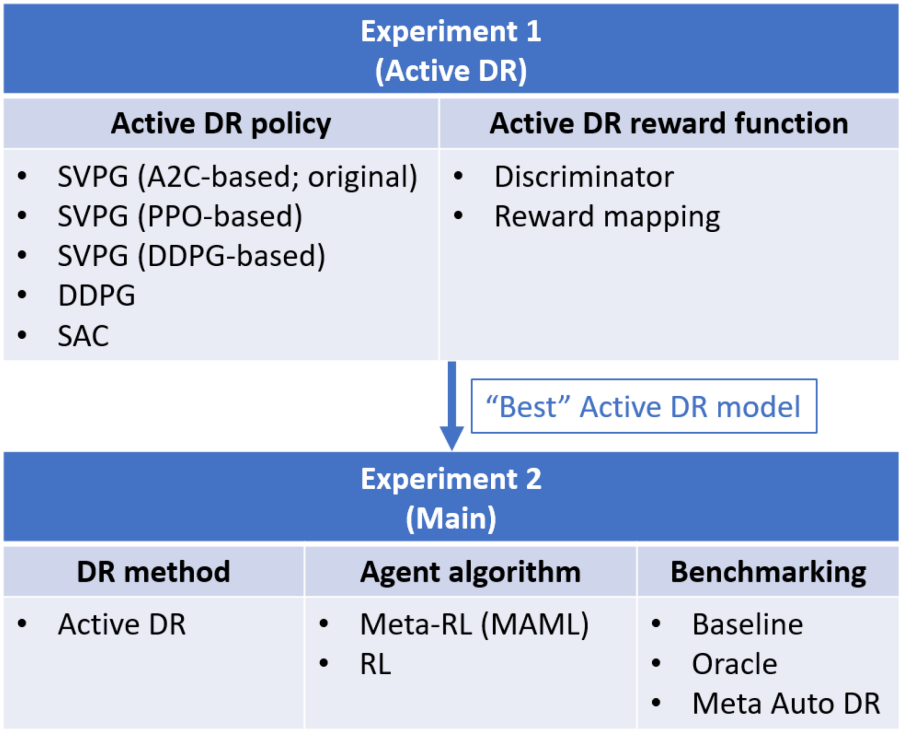


Figure 7.1. *The Experimental Plan.*

In figure 7.1, the baseline used represents the base RL algorithm trained using the default values of the environment parameters without any randomizations during training, while the oracle represents the base RL algorithm trained at each value of the randomization space. Additionally, noise is not injected to actions of the baseline or oracle agents during training, unlike with other algorithms.

In *LunarLander-v2* shown in figure 7.2, the agent controls two engines (main and side) to fulfill the task of landing between the two flags using as little fuel as possible.

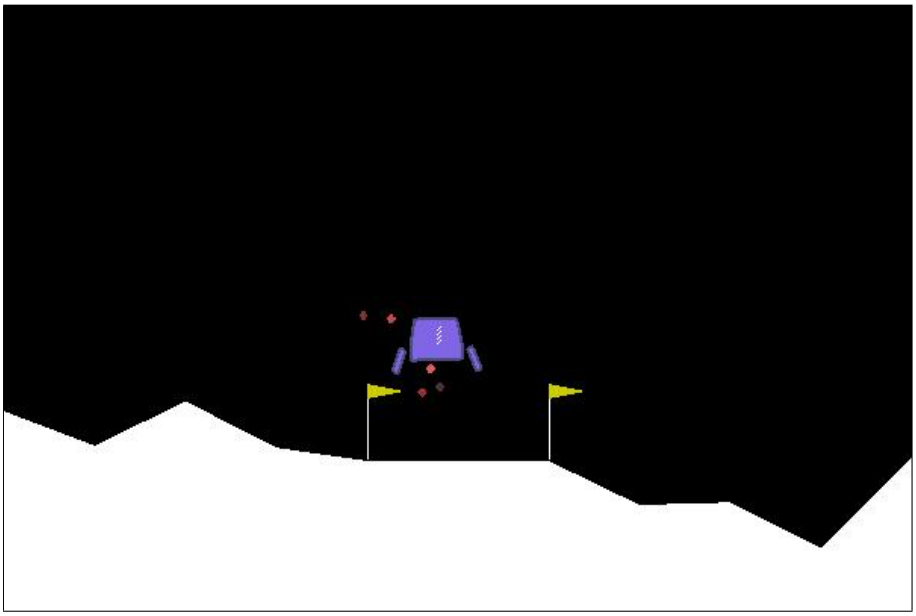


Figure 7.2. *LunarLander-v2 Environment.*

The policy with average performance across 3 seeds in training is used for testing where it is run for every randomization parameter value in the given range for 100 episodes and averaged. The full list of hyperparameters is presented in appendix A.

7.2 Analysis of Active Domain Randomization

7.2.1 Experimental Setup

As mentioned in section 7.1, the analysis of the Active DR algorithm is chosen to take place in the *LunarLander-v2* environment, as it is one of the environments which was used in the original paper, as well as using improvements made on top of the original code. Accordingly, the randomization parameter is chosen to be the main engine strength (MES) which means that the environment-based RL tasks are based on changes to the action space $\mathcal{A}$ of the environment component $\mathcal{E}$ defined in equation 2.2. Since the algorithm as a whole involves multiple components, and three of which are being co-trained, the algorithm is considered somewhat complex and it may be challenging to analyze due to complicated coupling and interplay among these components. So, before analyzing the algorithm as a whole, the DR optimization is broken into its constituent components (aka units), namely the DR policy and the DR reward function, for which unit tests are designed and applied. The purpose of these tests is to provide inputs to each of the units in isolation and compare the output with the expected outcome based on how its working principle is described by the original paper. In case of discrepancies between the actual and expected outputs, hypotheses can be developed as to the reasons why they occur, and, subsequently, improvements or alternatives can be suggested and analyzed.

Policy

In isolation, the DR policy is a standard RL policy used to solve a standard RL optimization problem described in subsection 4.2.1 as *an N-dimensional optimization problem with non-stationary rewards*. While in this thesis the experiments only consider the case where $N = 1$, the non-stationarity of the rewards comes from the fact that what is perceived as challenging by the agent during training is constantly changing. In subsection 4.2.1 it is also explained that the requirement for this policy is to direct sampling towards the regions that produce environment instances which are *currently* perceived by the agent as the most challenging. Despite the underlying 1-dimensional optimization problem seeming simple, this means that this policy's convergence will need to be near-instantaneous, all the while maintaining continuous exploration. An additional layer of complexity arises if the reward distribution over the configuration space is multi-modal. This requires that the policy is able to converge to multiple regions and do so quickly as they shift or change.

To simplify the unit tests, the reward distribution is confined to the unimodal case. Hence, the test cases designed are based on a reward function which assigns linearly-decaying rewards from either sides of a single chosen value in the randomization range, making the expected sampling pattern resemble a "peak" or a "hill".

Although the policy used in the original paper is an A2C-based SVPG policy, any RL algorithm can, in principle, be used in solving the underlying optimization problem given enough samples to train on. The rationale behind choosing SVPG in [8] was explained that it operates in a maximum-entropy RL framework which balances exploration and exploitation. In this vein, several alternatives are investigated including DDPG-based SVPG, PPO-based SVPG, DDPG and SAC algorithms.

Reward Function

The DR reward function providing rewards for the DR policy aims at outputting high rewards for the regions of the parameter sampling space that are *currently* posing the greatest difficulty for the agent to solve, and vice versa [8]. In this sense, the level of difficulty is an alias for the agent's performance in those environment instances. However, as detailed in subsection 4.2.1, discrepancies between the agent's rollouts in the randomized and reference environments is used in the original paper as a proxy for inferring difficulty. The idea being that if the transitions in the rollouts appear different according to a discriminator network, then it indicates that the agent is finding the randomized instances difficult, and vice versa. According to the original paper, as the agent improves on the troublesome environments, it becomes harder for the discriminator to distinguish between the agent's rollouts in them and the reference environment (i.e. they begin to appear similar), which is then reflected in a lower reward for those regions communicated to the DR policy.

To verify the correct operation of the discriminator in relation to how it is intended to behave, thought experiments are discussed to explore the general validity behind its working principle and unit tests are conducted to check its validity specific to the given use case. The unit tests are designed to first investigate if the discriminator can successfully discriminate between trajectories from two different environment instances. Then, it is checked that the discriminator's ability to discriminate between rollouts from a randomized environment instance and the reference environment drop with time as the agent trains and improves on the randomized environment. These unit test cases are listed in table 7.1. Lastly, comments about the general appropriateness of using a discriminator network in the Active DR framework are mentioned and alternatives are suggested.

Agent	Discriminator	Environments
Trained	Fixed (at random initialization)	Fixed (MES$_{ref}$ = 16, MES$_{rand}$ = 10)
Fixed (at random initialization)	Trained	Fixed (MES$_{ref}$ = 16, MES$_{rand}$ = 10)
Fixed (trained on MES = 20)	Trained	Fixed (MES$_{ref}$ = 20, MES$_{rand}$ = 8)
Trained	Trained	Fixed (MES$_{ref}$ = 20, MES$_{rand}$ = 8)

Table 7.1. *Unit test cases for use of discriminator network in Active DR algorithm.*

7.2.2 Results Evaluation & Analysis

Policy

To summarize the requirements on the DR policy mentioned in the previous subsection, they are:

- Fast convergence around maximum reward regions, and
- Maintaining continuous exploration to track those regions as they move with time.

The unit test is designed to check the fulfilment of the first requirement. The maximum reward location is chosen as fixed at the center of the randomization parameter's range. The rest of the rewards are chosen to be inversely proportional to the distance of the sampled value from the peak (maximum reward) location. This is illustrated in equation 7.1, and indicated by the red dotted line in the figures 7.3 – 7.7. This choice was made to resemble the actual scenario encountered by the Active DR algorithm. The rest of the hyperparameters follow the values presented in appendix A.

$$r = reward\_scale \times \frac{1}{|max\_reward\_location - sampled\_value|} \tag{7.1}$$

The results for the for the original A2C-based SVPG policy is visualized in figure 7.3.

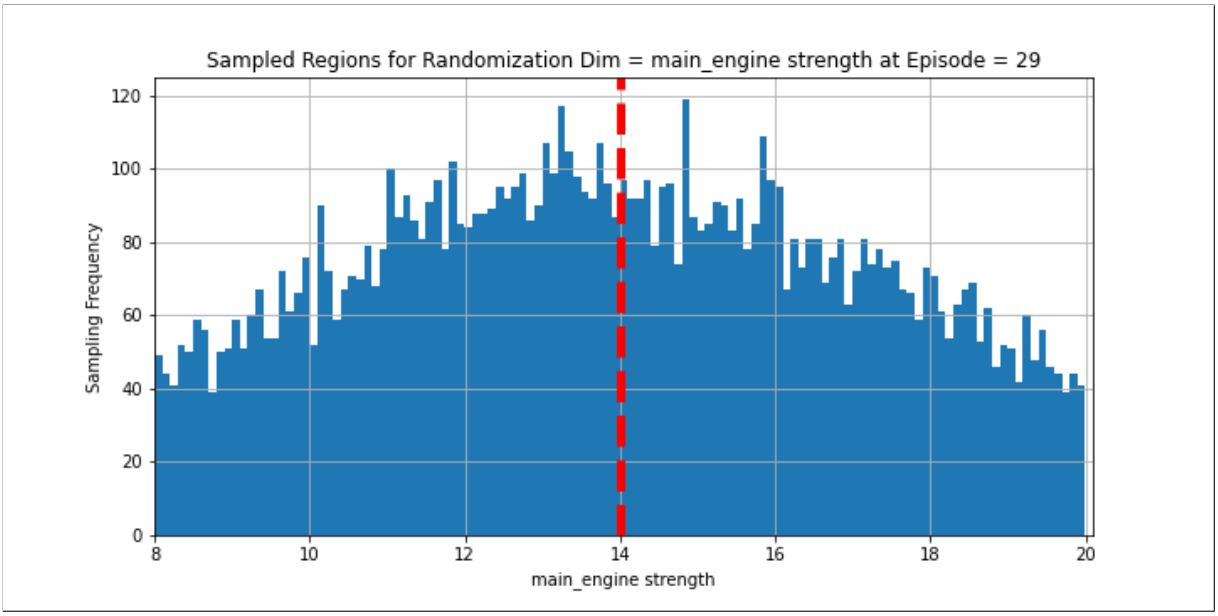


Figure 7.3. *Convergence of the A2C-based SVPG Policy on the Unit Test Case.*

From the figure the convergence appears to be poor since the algorithm still ends up sampling lower reward regions with high frequency. **This renders the A2C-based SVPG policy unsuitable for the actual use case as a part of the whole Active DR algorithm**. Similar results were obtained from evaluating a PPO-based SVPG policy, as is shown in figure 7.4.

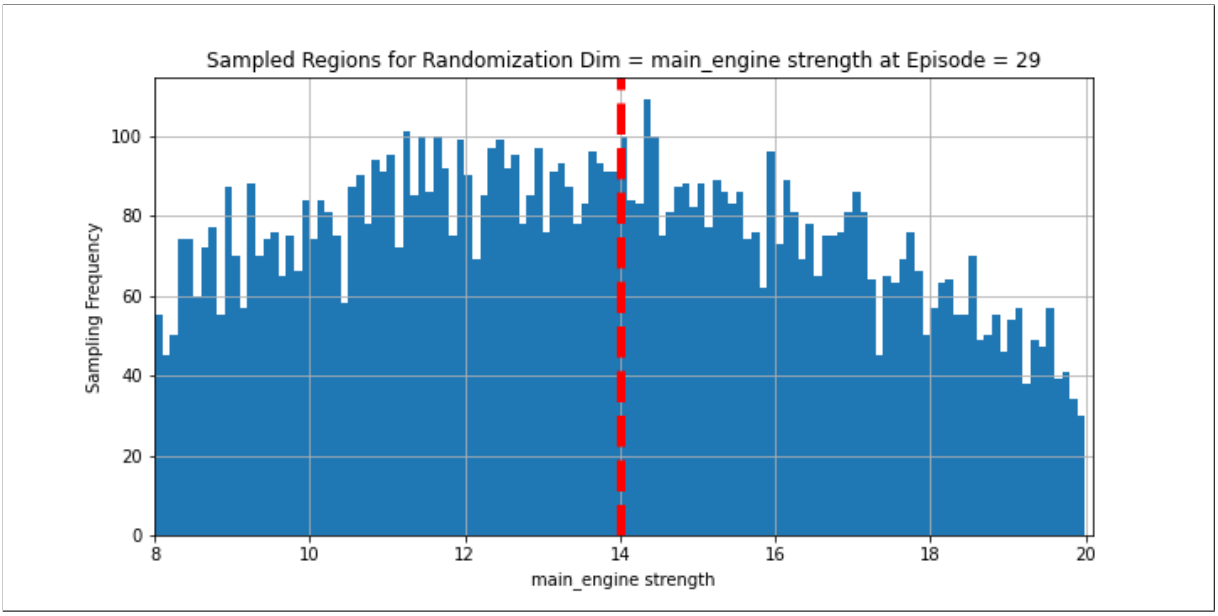


Figure 7.4. *Convergence of the PPO-based SVPG Policy on the Unit Test Case.*

Since SVPG policies based on on-policy RL algorithms turned out to be unsuitable, off-policy algorithms were analyzed next starting with a DDPG-based SVPG policy visualized in figure 7.5.

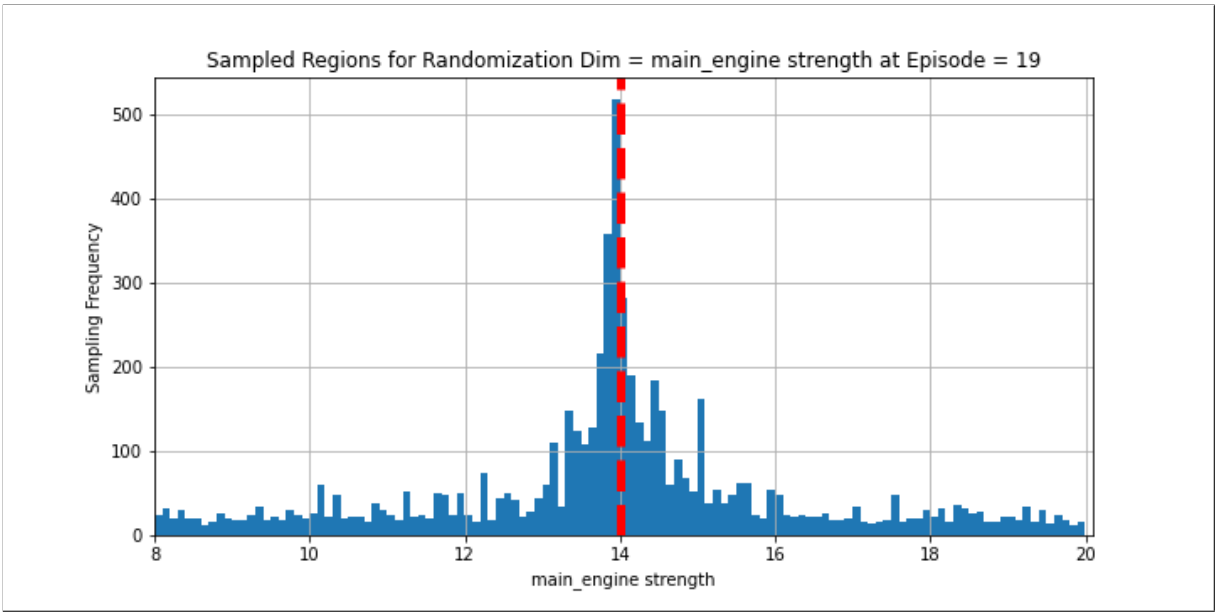


Figure 7.5. *Convergence of the DDPG-based SVPG Policy on the Unit Test Case.*

From the figure it can be seen that the lower reward regions are sampled with much lower frequency. Next, non-SVPG, off-policy algorithms were analyzed, namely DDPG and SAC algorithms, and the results are shown in figures 7.6 and 7.7, respectively.

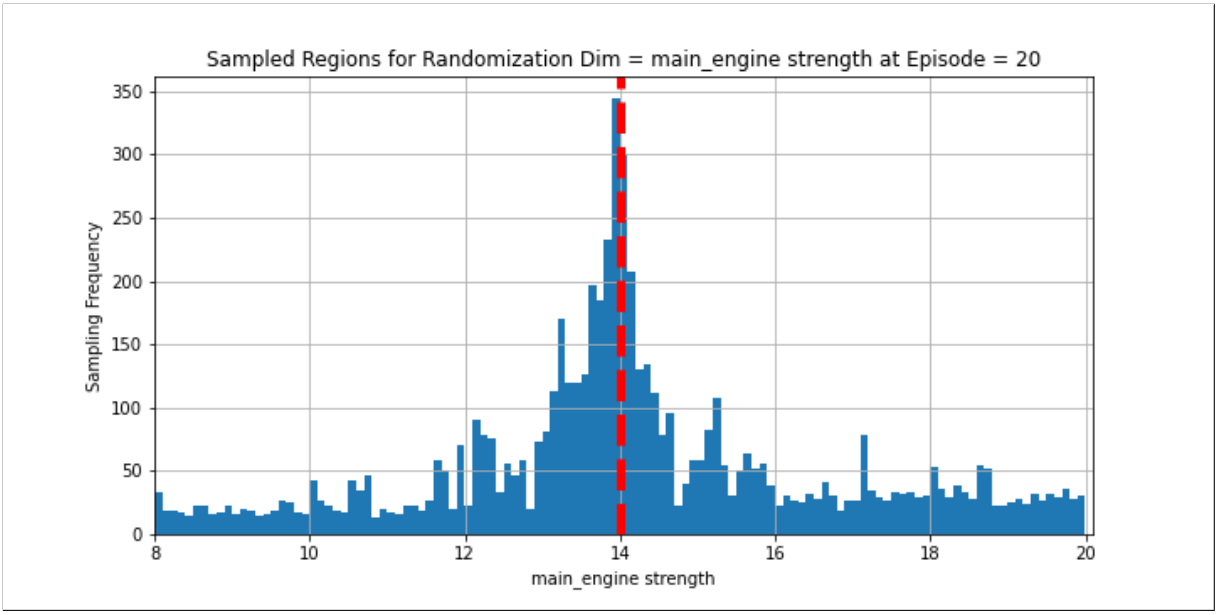


Figure 7.6. *Convergence of the DDPG Policy on the Unit Test Case.*

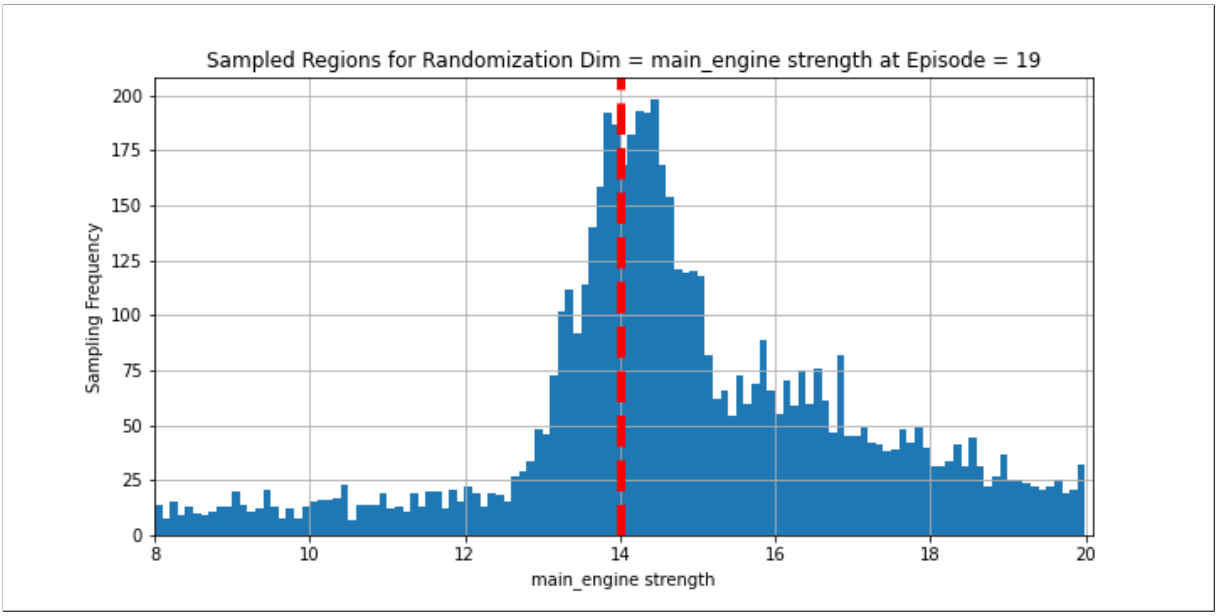


Figure 7.7. *Convergence of the SAC Policy on the Unit Test Case.*

The results demonstrate a faster convergence compared to the on-policy SVPG policies. In addition, these policies also maintain the benefit of sampling the lower reward regions with lower frequencies.

To formulate an explanation for these results, few points should be first noted. In the original SVPG paper and in the Active DR paper, the SVPG method is only used with on-policy RL algorithms. In those on-policy based SVPG methods, each policy worker trains only on its own samples. According to the SVPG paper, the number of these samples per policy worker has to be large enough for training [45]. This requirement was possible to fulfill when the SVPG method was used to solve standard RL agent optimization problems in the paper where it was originally introduced. However, this is not possible with its novel use within the Active DR algorithm to solve the DR optimization problem. This is because the DR optimization problem exists within a bilevel optimization which requires more frequent updates to the location of the maximum reward regions to properly reflect the agent's changing performance during training. The result of this is that the number of training samples has to be much smaller for this case; possibly ending up smaller than is required for proper training of the DR policy.

Thus, one reason for the superiority of the off-policy algorithms compared to the on-policy ones in solving this problem may be due to the fact that the off-policy workers train on a shared pool of samples (from a centralized replay buffer) which grants each particle a larger number of samples to train on every iteration. This is an inherent advantage of the off-policy algorithms over the on-policy ones. For the same reason it can be seen why non-SVPG on-policy algorithms will also be unsatisfactory in this particular setting.

A hyperparameter of particular importance in the case of using off-policy DR policies within Active DR is the size of the replay buffer. This buffer has to contain samples *only* from the

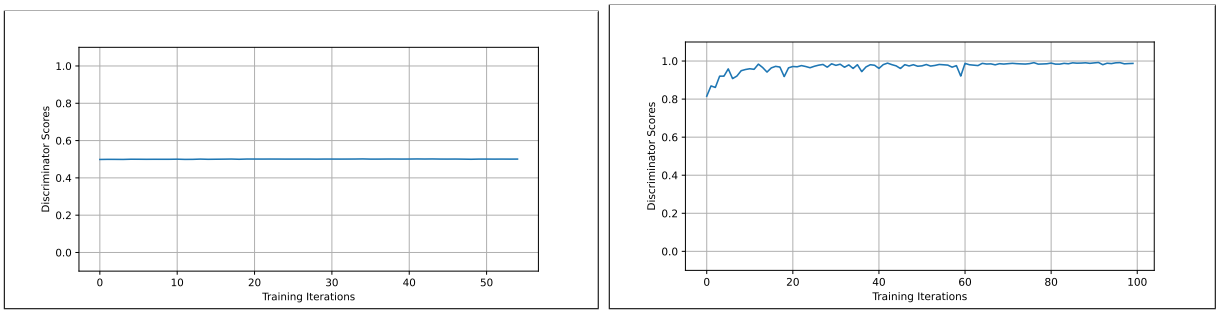
current training iteration so that the policy is only informed by what the agent is *currently* finding challenging. Another important hyperparameter for the SVPG (and SAC) algorithms is the temperature (α). This hyperparameter is responsible for controlling the trade-off between exploration and exploitation in the maximum-entropy RL framework. Although the original SVPG paper mentions that the value of 10 was empirically found to be the best, it might not be suitable for all of the optimization problems that are solved by the DR policy. Therefore, a search for the most suitable value has to be conducted as a part of a hyperparameter search.

As the second requirement for the DR policy builds on the first one by extending the fast convergence to a moving region, passing the unit test described above is a prerequisite for the fulfillment of the second requirement. Hence, only the SAC, DDPG and DDPG-based SVPG policies are considered as suitable candidates to proceed with. Although a unit test case can be designed for the second requirement, the amount and frequency by which the reward distribution shifts during training is expected to differ between the unit test case and the actual use case scenarios when using the full Active DR algorithm in different environments and with different randomized parameters and ranges. As a result, this will call for a different tuning of some hyperparameters such as temperature and so on. Therefore, the testing of this requirement is left to be observed from testing the different proposed alternatives within the full Active DR algorithm in the actual use cases.

Reward Function

The four unit test cases listed in table 7.1 are designed in such a way that the first three tests check that the discriminator network is working correctly as a binary classifier, and the final test to check if it carries out its intended purpose within the Active DR algorithm according to the working principle described in the previous section. These are visualized through the discriminator scores over time. The discriminator gets as input transitions from the randomized environment and outputs a score between 0.5 & 1. A score of 1.0 corresponds to perfectly classifying the input transitions as originating from the randomized environment, while a score of 0.5 indicates the inability of the discriminator to distinguish between the transitions of the two environments.

In the first test, the agent is trained on a fixed randomized environment while the discriminator network starts from its random initialization and is left untrained. Thus, the expected outcome is for the discriminator to remain unable to distinguish between the two environments. This is observed in figure 7.8a which represents the actual outcome of the test. Conversely, figure 7.8b represents the outcome of the second test case where the agent is left untrained at its random initialization while the discriminator is trained. The outcome of this test case also matches the expectation of the discriminator improving over time at distinguishing between the trajectories of the two environments.

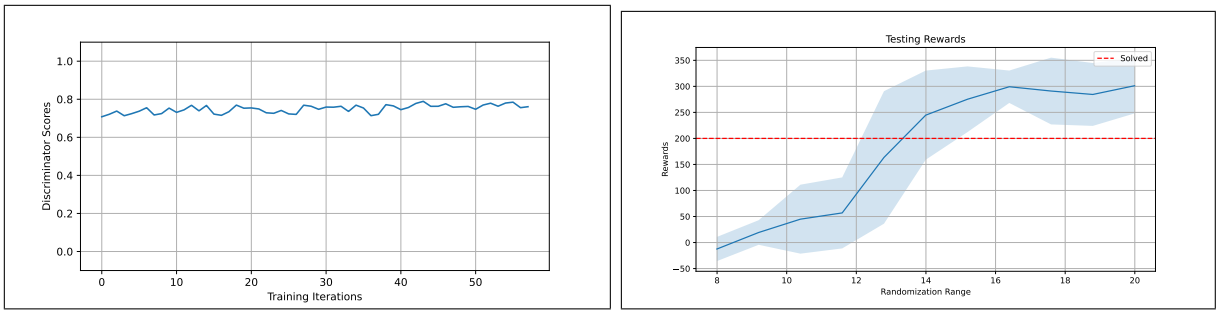


(a) *Discriminator Fixed at Random Initialization and Agent Trained.*

(b) *Agent Fixed at Random Initialization and Discriminator Trained.*

Figure 7.8. *Discriminator Scores Over Time for Fixed Reference and Randomized Environments.*

The third unit test represents a case similar to the second test where the discriminator is trained and the agent is fixed. Although the agent in this case has been pretrained on the reference environment separately, the expected outcome remains the same, i.e. that the discriminator's scores should be observed to increase with time. However, the actual outcome depicted in figure 7.9a suggests that the discriminator's ability at distinguishing between transitions from the two environments does not improve over time as it trains. One explanation for this can be that the environment transitions may appear similar despite the agent's performance being different on both environments as shown in figure 7.9b.



(a) *Discriminator Scores Over Time.*

(b) *Agent's Rewards Over Randomization Range.*

Figure 7.9. *Agent Pretrained on The Reference Environment and Discriminator Trained.*

At this point, it can be concluded that **discrepancy between transitions is not a good proxy for measuring difference in performance** and thus that the discriminator network is not always able to fulfill the intended aim behind using it in the context of the Active DR algorithm.

For completeness, a final test case is performed on the same fixed environments while both the agent and discriminator are trained. The aim of this test is to investigate the validity of the hypothesis made by the original paper which states that the discriminator scores for a certain environment instance will drop as the agent trains and improves on it. This hypothesis also relates to the aforementioned point about the incorrespondence between the differences in

transitions and differences in performance. Figure 7.10 shows the result obtained for this test case which also does not match the expected outcome.

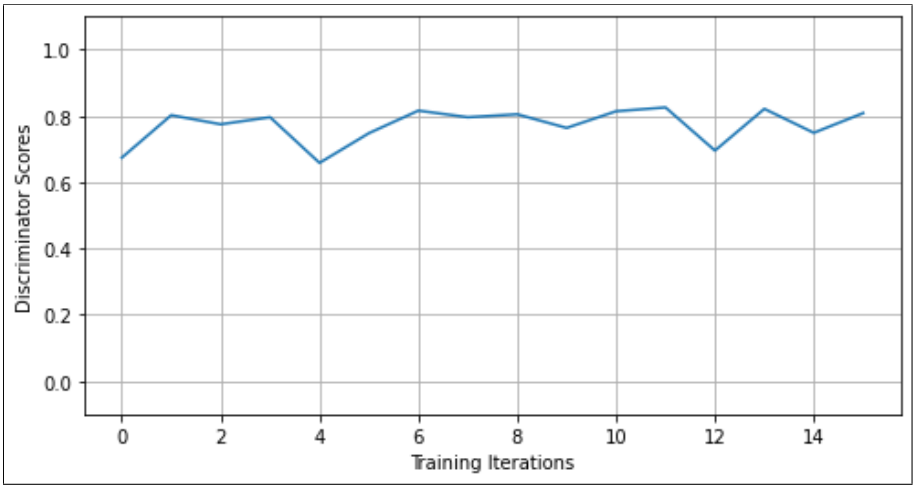
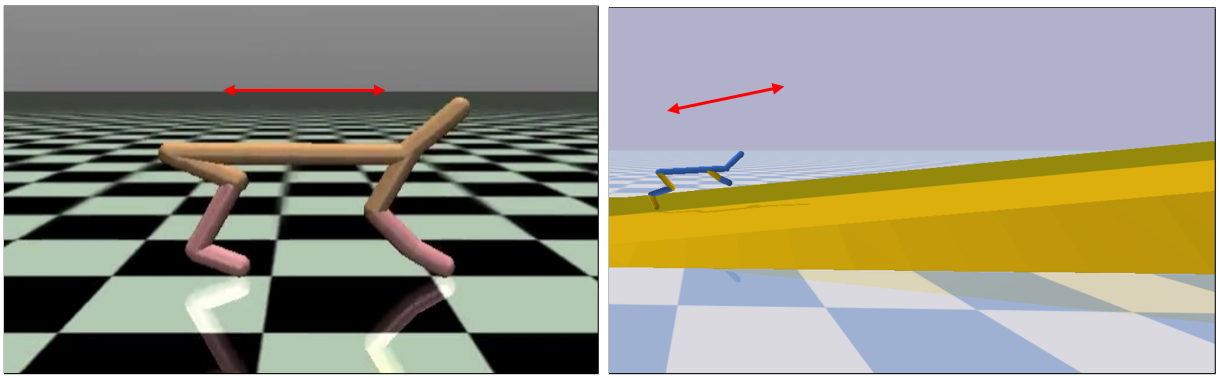


Figure 7.10. *Discriminator Scores Over Time For Trained Agent And Discriminator.*

The general applicability of that hypothesis can be also challenged through a simple thought experiment as an example. This thought experiment takes place in a different MuJoCo environment, namely *HalfCheetah-v2*, where the agent actuates a number of joints to move forward. If the randomized parameter is taken to be the inclination of the ground, transitions between different environment instances will always be fundamentally different, at least in part, regardless of the agent performance on these instances. This is demonstrated in figure 7.11 where the red arrow indicates the direction of typical transitions in each of the environments shown.



(a) *Default Environment With No Ground Inclination* ***(b)*** *Environment With Ground Inclination (from [47]).*

Figure 7.11. *Different Instances of the HalfCheetah-v2 Environment.*

Instead of using a discriminator network as a reward function for the DR optimization problem of the Active DR algorithm, a much simpler function and a more direct measure of performance is the difference between the agent's rewards in the randomized and reference instances. This results in a simple deterministic mapping of the agent rewards to DR rewards. This also has the additional benefit of greatly with several trained components.

The Active DR algorithm in 2 is thus augmented by removing the parts in red relating to the discriminator and using the proposed mapping function for the $DR\_reward\_function(.,.)$. Additionally, the agent starts by pretraining on the reference environment until it surpasses the environment's reward threshold before the algorithm starts actively sampling from the configuration space, as indicated by the line in blue in the algorithm. Those changes are also inherited by algorithm 4.

The use of a deterministic mapping function as alternatives for the discriminator network is evaluated within the full Active DR algorithm in the following part.

Algorithm

As a final step in the set of experiments of this section, the full Active DR algorithm is evaluated on the *LunarLander-v2* environment and the different alternatives for the DR policy and reward function are compared and analyzed. Additionally, the UDR algorithm is included in the results for benchmarking.

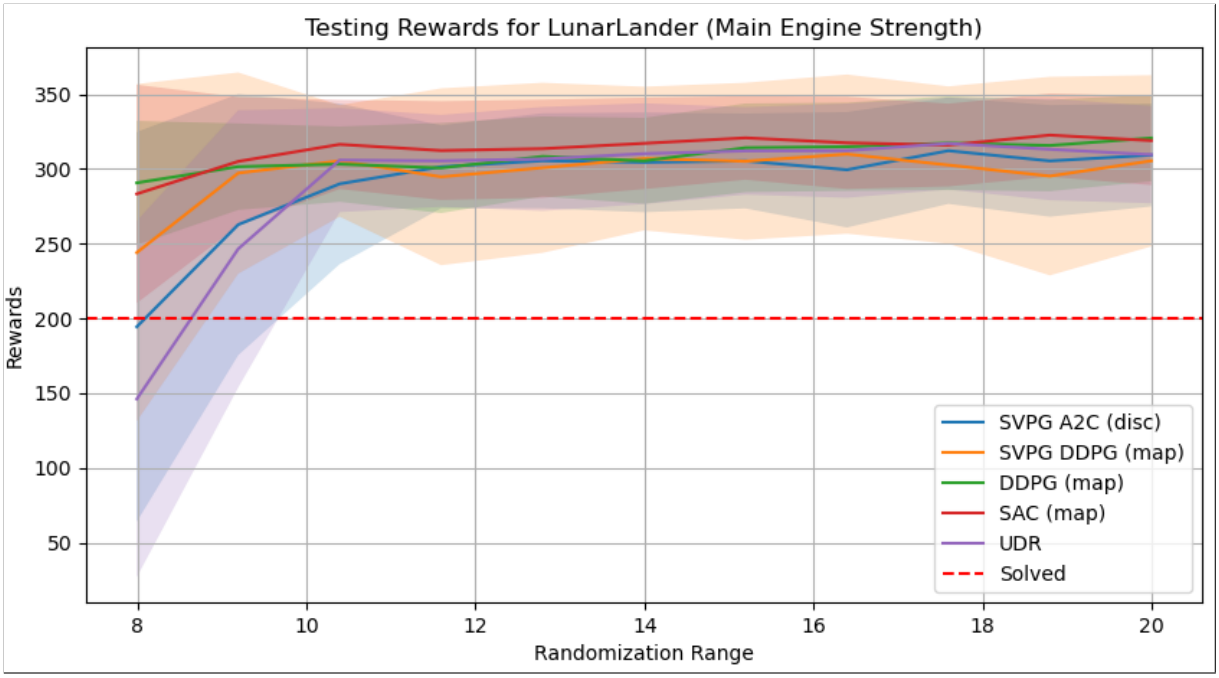


Figure 7.12. *Comparison of The Original Active DR Algorithm to The Proposed Versions.*

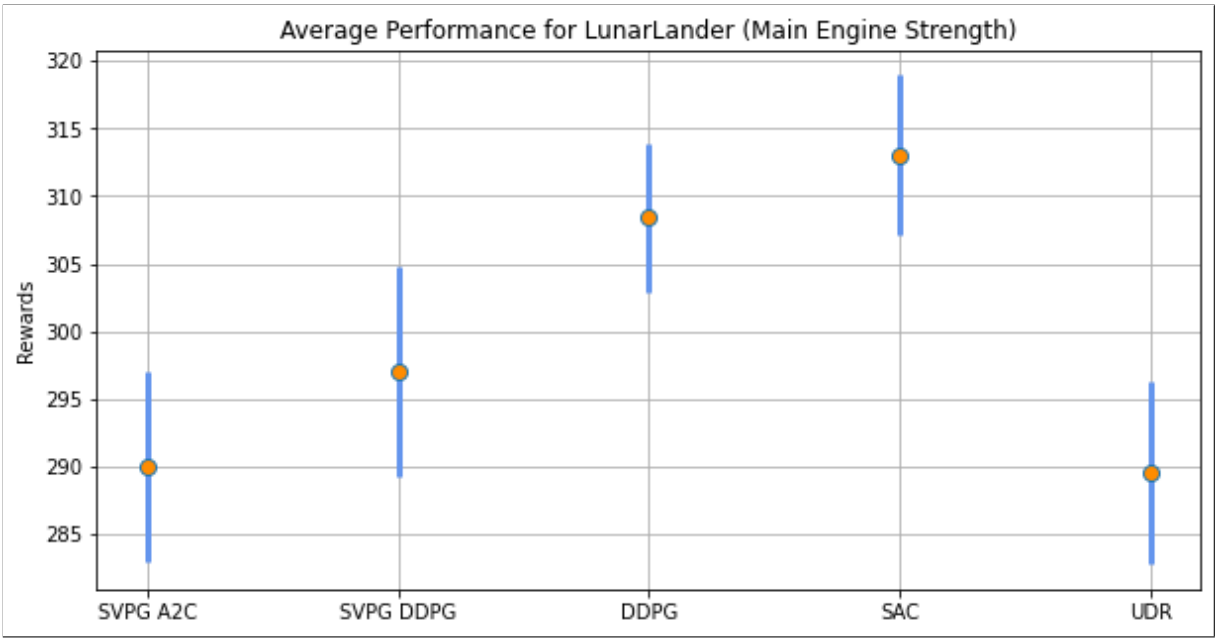


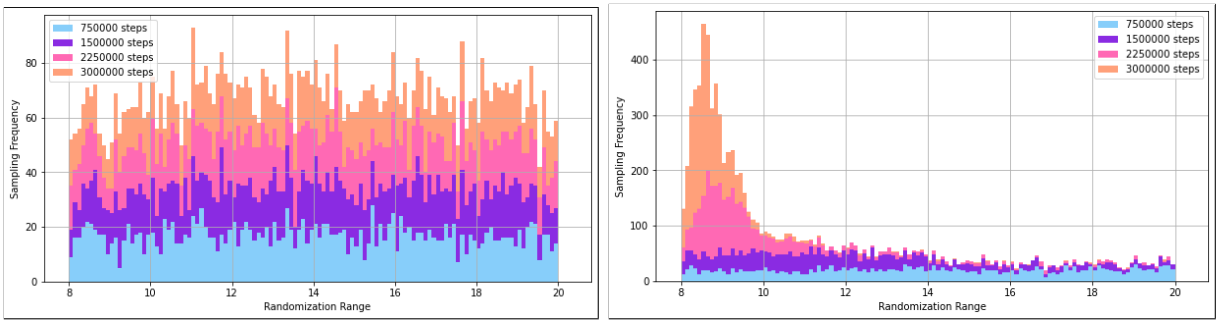
Figure 7.13. *Average Performance and Standard Deviation of The Active DR Alternatives.*

From figure 7.12 the benefits of the proposed improvements are most apparent around the lowest main engine strength values in the range (and hence the most difficult for the agent to solve). Moreover, from figure 7.13 they also seem to be better than the original Active DR algorithm in terms of the average performance over the configuration space – the chosen evaluation metric discussed in section 5.2.

In analyzing the statistical significance of the results at the most challenging value of the randomization parameter (MES = 8), a Mann–Whitney U test [48] showed that the difference between UDR and the original Active DR algorithm with A2C-based SVPG policy and a discriminator network was not very significant (U = 4248.0, n1 = n2 = 100, P > 0.03 two-tailed), whereas an extremely significant difference was observed between UDR and the improved Active DR alternatives using deterministic reward mapping: DDPG-based SVPG (U = 2872.0, n1 = n2 = 100, P < 0.0000002 two-tailed), DDPG (U = 1927.0, n1 = n2 = 100, P < 10^{-13} two-tailed), and SAC (U = 2018.0, n1 = n2 = 100, P < 2×10^{-13} two-tailed). From the test and the figures, the superiority of the Active DR algorithm over the UDR algorithm can also now be confirmed empirically. Additionally, the test also showed significant differences between the original Active DR algorithm with A2C-based SVPG policy and a discriminator network and the proposed alternatives: DDPG-based SVPG (U = 3669.0, n1 = n2 = 100, P < 0.0006 two-tailed), DDPG (U = 3082.0, n1 = n2 = 100, P < 0.000002 two-tailed), and SAC (U = 3007.0, n1 = n2 = 100, P < 0.0000006 two-tailed).

In figure 7.14, the sampled regions of the A2C-based SVPG policy with a discriminator network is compared to that of the SAC policy with a reward mapping function, as an example. It is noticed that the suggested improvements yield a sampling strategy which assigns higher sampling frequencies to values of the randomized parameter constituting the most difficult environment

instances, causing the agent to dedicate more time to them in training. In contrast the sampling pattern of the original A2C-based SVPG policy with a discriminator is almost uniform. This explains the lack of significance between the results of the original A2C-based SVPG policy and UDR observed in the Mann–Whitney U test. Additionally, and of comparable importance to the sampling frequency or pattern, is the *order* of sampling the randomization values. Figure 7.14b demonstrate that the most challenging environments are almost exclusively what the agent has been exposed to towards the final part of its training. On the other hand, if these environments were sampled with higher frequency only early on in the training, the agent's policy may run the risk of *forgetting* the tasks corresponding to the lower values.



(a) *A2C-based SVPG Algorithm With Discriminator.* ***(b)*** *SAC Algorithm With Reward Mapping.*

Figure 7.14. *Sampled Regions of the LunarLander-v2 Main Engine Strength.*

7.3 Guided Domain Randomization With Meta Reinforcement Learning

7.3.1 Experimental Setup

In the set of experiments presented in this section, the proposed meta-Active DR algorithm is analyzed and evaluated against its Active DR counterpart in the same *LunarLander-v2* environment.

The Active DR algorithm chosen for this set of experiments is the one based on the SAC algorithm with the difference between the agent's rewards in the randomized and reference instances as the $DR\_reward\_function(.,.)$. To benchmark the results, a *baseline* policy trained on the default value of the randomized parameter is included along with an *oracle* which presents the asymptotic performance of the agent's RL algorithm on the given task at each value in the randomization parameter's range. Furthermore, the experiments include the proposed meta-Auto DR algorithm to help contextualize the performances of the Active DR and meta-Active DR algorithms.

7.3.2 Results Evaluation & Analysis

Figures 7.15 and 7.16 depict how the proposed algorithm 4 which combines MAML with Active DR compares to the Active DR algorithm.

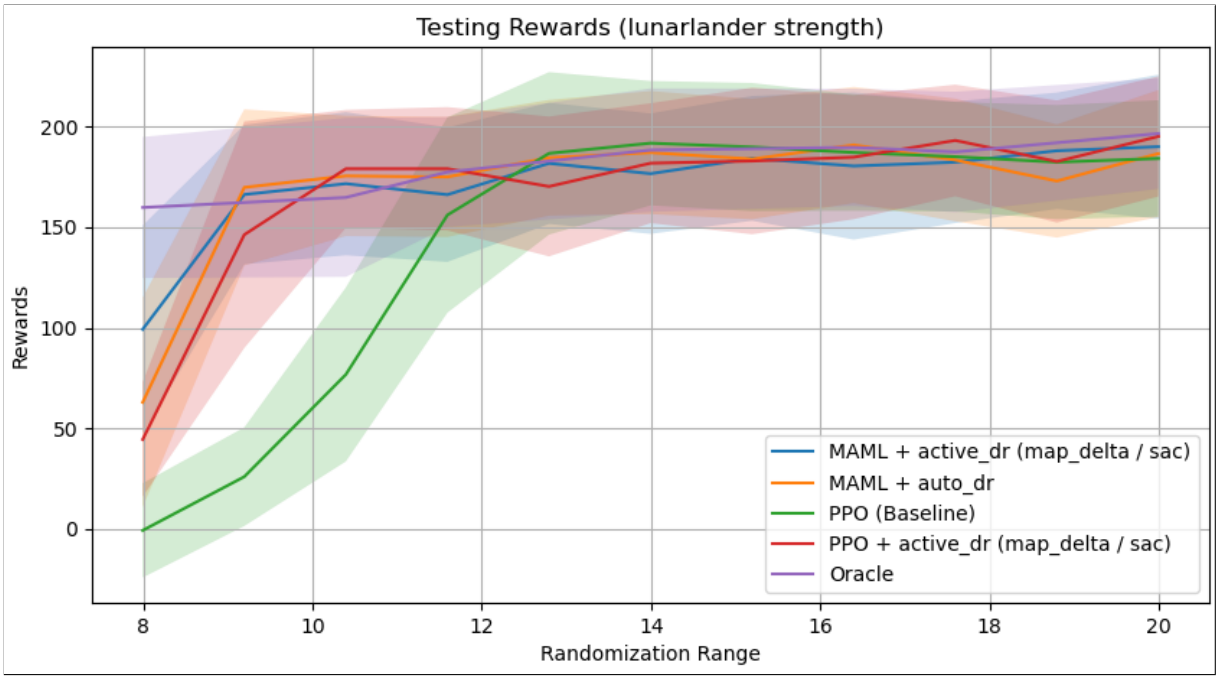


Figure 7.15. *Comparison of The Active DR Algorithm to The Proposed Meta-Active DR.*

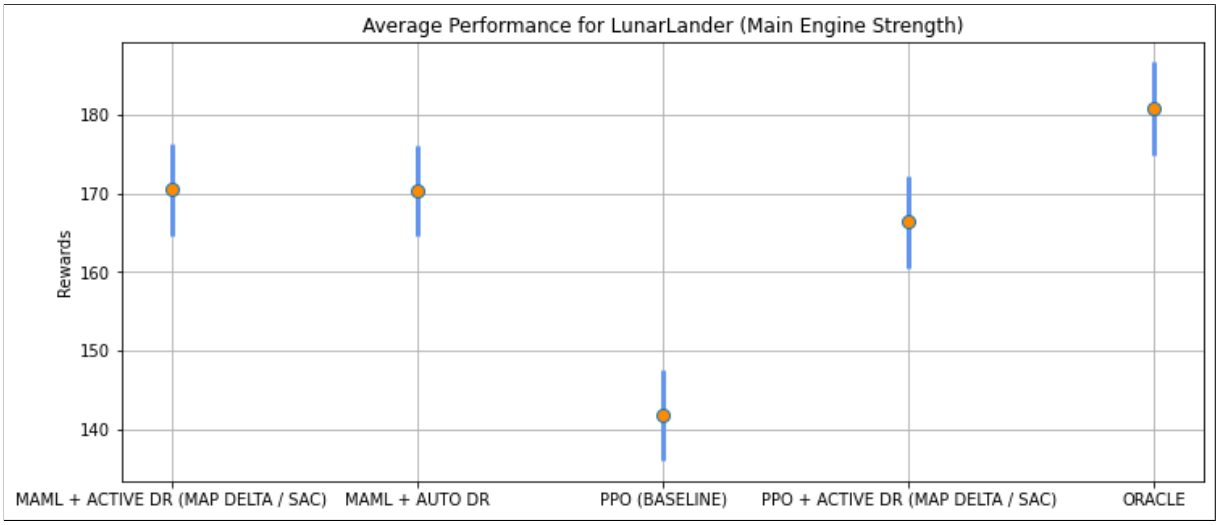


Figure 7.16. *Comparison of The Average Performance of The Active DR Algorithms.*

From the figures, the proposed meta-Active DR algorithm is shown to exhibit an improvement over its Active DR counterpart in the *LunarLander-v2* environment, not only in terms of performance in the more challenging regions of lower main engine strength values (at MES=8 the difference was shown by Mann–Whitney U test as highly significant, U = 2719.0, n1 = n2 = 100, P < 0.00000002 two-tailed), but also in terms of the average performance over the entire range. This suggests that the proposed adaptive policies learned by Meta-RL algorithms offer a promising direction to improve over the performance of the robust policies for environment-based RL tasks.

8. CONCLUSIONS

This thesis presented a study of RL agents trained with domain randomization for the purpose of domain transfer. In particular, the objective was formulated as improving the average performance of these agents over their entire configuration space.

Initially, an analysis of a popular domain randomization method, namely active domain randomization, was carried out. In this analysis, it was demonstrated that the original choices of the DR policy and reward function were unsuitable for the Active DR algorithm to achieve its intended outcomes in the use cases presented. More concretely, it was shown that the A2C-based SVPG policy, originally chosen as the DR policy, exhibited poor convergence qualities which was explained to be due to the small training pool available for the policy. To address this issue, off-policy DR policies were proposed and tested as alternatives, and were shown to yield better convergence on the same test cases. For the DR reward function, chosen originally as a discriminator network, it was demonstrated that the discrepancy between transitions measured by the network was not a good proxy for measuring difference in performance. For a more direct measure, a simple, deterministic reward mapping function was proposed. The proposed improvements to the Active DR algorithm were compared to the original implementation and observed to result in a better average performance in the *LunarLander-v2* environment where the randomization parameter is the main engine strength.

Finally, the problem of improving the average performance of the guided domain randomization methods was analyzed. To this end, algorithms were proposed to replace the standard RL algorithms within those GDR methods with Meta-RL algorithms, resulting in Meta-GDR methods that learn adaptive policies instead of robust ones. The used GDR methods were Active DR and Auto DR, and the used Meta-RL algorithm was MAML. It was then shown that the meta-Active DR method (as a representative of the Meta-GDR family) leads to an improvement of the average performance over its Active DR counterpart in the experiments conducted.

9. FUTURE DIRECTIONS

The experiments in chapter 7 presented the case where the used environment had one randomized parameter. Expanding the study to cases where the number of randomization parameters $N > 1$ is straightforward from an implementation point of view and would provide insight on the any possible interplay between the different randomized parameters as well as highlight the strengths and weaknesses of the different methods involved in the comparisons.

In order for the results to be more conclusive and reliable, conducting the experiments presented in this thesis for a number of other different environments with different randomization parameters becomes essential.

Another possibility to expand the study is to include more evaluation metrics beside generalization (or average performance over the entire configuration space), such as data or sample efficiency, resources consumption, complexity, and so on. A deeper analysis of the advantages, limitations and domains of applicability of adaptive and robust policies could also be another insightful dimension to extend the study in.

The experiments and analysis in this thesis were also limited to sim2sim transfer situations and a natural continuation would be to carry out sim2real transfer experiments, as was previously explained in section 1.1.

To get a more complete picture and arrive at the best solution for domain transfer problems, including more domain randomization methods, such as RARL [9], becomes essential. Additionally, emergent meta-learning from combining domain randomization with memory-augmented RL was studied in [10] and [5]. This approach could also be compared to Meta-GDR and memory-augmented versions of Meta-GDR (e.g. by using a recurrent policy network in the underlying Meta-RL algorithm).

Finally, the Meta-RL algorithm used in this thesis, namely MAML, is a model-free Meta-RL algorithm. Thus, another low-hanging fruit is including model-based Meta-RL in the comparisons, such as GrBAL [6] and MB-MPO [40], as this is the type of Meta-RL which has been solely used in literature so far with UDR in the setting of environment-based RL tasks. Comparing model-based Meta-RL to model-free Meta-RL in this setting not only allows putting the novelty of using model-free Meta-RL in this setting into perspective, but also the novelty of using Meta-RL with GDR instead of UDR.

REFERENCES

[1] Bertsekas, D. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[2] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[3] Hausman, K. *Reinforcement Learning: Review*. Accessed: 2022-31-05. 2021. URL: https://cs330.stanford.edu/slides/cs330_2021_rl_review.pdf.

[4] Weng, L. Domain Randomization for Sim2Real Transfer. *lilianweng.github.io* (2019). URL: https://lilianweng.github.io/posts/2019-05-05-domain-randomization/.

[5] Peng, X. B., Andrychowicz, M., Zaremba, W. and Abbeel, P. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 1–8.

[6] Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S. and Finn, C. Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyztsoC5Y7.

[7] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), pp. 23–30.

[8] Mehta, B., Diaz, M., Golemo, F., Pal, C. J. and Paull, L. Active Domain Randomization. *CoRL*. Ed. by L. P. Kaelbling, D. Kragic and K. Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1162–1176. URL: https://proceedings.mlr.press/v100/mehta20a.html.

[9] Pinto, L., Davidson, J., Sukthankar, R. and Gupta, A. K. Robust Adversarial Reinforcement Learning. *ICML*. 2017.

[10] OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N. A., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W. and Zhang, L. M. Solving Rubik's Cube with a Robot Hand. *ArXiv* abs/1910.07113 (2019).

[11] Finn, C., Abbeel, P. and Levine, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *ICML*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1126–1135. URL: https://proceedings.mlr.press/v70/finn17a.html.

[12] Finn, C. *The Meta-Learning Problem and Black-Box Meta-Learning*. Accessed: 2022-04-06. 2021. URL: https://cs330.stanford.edu/slides/cs330_metalearning_bbox_2021.pdf.

[13] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.

[14] Weng, L. Policy Gradient Algorithms. *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

[15] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. and Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *ICML*. 2016.

[16] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P. Trust Region Policy Optimization. *ICML*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[18] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. Continuous control with deep reinforcement learning. *ICLR*. Ed. by Y. Bengio and Y. LeCun. 2016. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[19] Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *ICML*. 2018.

[20] Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P. and Ba, J. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057* (2019).

[21] Hausman, K. *Lifelong Learning*. Accessed: 2022-31-05. 2021. URL: https://cs330.stanford.edu/slides/cs330_2021_lifelong_learning.pdf.

[22] Thrun, S. and Pratt, L. Learning to learn: Introduction and overview. *Learning to learn*. Springer, 1998, pp. 3–17.

[23] Hospedales, T. M., Antoniou, A., Micaelli, P. and Storkey, A. J. Meta-Learning in Neural Networks: A Survey. *IEEE transactions on pattern analysis and machine intelligence* PP (2021).

[24] Finn, C. *Few-shot learning via metric learning*. Accessed: 2022-04-06. 2021. URL: https://cs330.stanford.edu/slides/cs330_nonparametric_2021.pdf.

[25] Finn, C. *Optimization-based meta-learning*. Accessed: 2022-04-06. 2021. URL: https://cs330.stanford.edu/slides/cs330_optbased_metalearning_2021.pdf.

[26] Vuong, Q. H., Vikram, S., Su, H., Gao, S. and Christensen, H. I. How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?: *ArXiv* abs/1903.11774 (2019).

[27] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E. and Stone, P. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *J. Mach. Learn. Res.* 21 (2020), 181:1–181:50.

[28] Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J. and Stanley, K. O. Enhanced POET: Open-Ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions. *ICML*. 2020.

[29] Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., Saenko, K. and Darrell, T. Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. *WAFR*. 2016.

[30] Gupta, A., Devin, C., Liu, Y., Abbeel, P. and Levine, S. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. *5th International Conference on Learning Representations, (ICLR) 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Hyq4yhile.

[31] Rusu, A. A., Vecerík, M., Rothörl, T., Heess, N., Pascanu, R. and Hadsell, R. Sim-to-Real Robot Learning from Pixels with Progressive Nets. *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 262–270. URL: http://proceedings.mlr.press/v78/rusu17a.html.

[32] Christiano, P. F., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P. and Zaremba, W. Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model. *ArXiv* abs/1610.03518 (2016).

[33] Antonova, R., Cruciani, S., Smith, C. and Kragic, D. Reinforcement Learning for Pivoting Task. *ArXiv* abs/1703.00472 (2017).

[34] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S. and Vanhoucke, V. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*. Ed. by H. Kress-Gazit, S. S. Srinivasa, T. Howard and N. Atanasov. 2018. DOI: 10.15607/RSS.2018.XIV.010. URL: http://www.roboticsproceedings.org/rss14/p10.html.

[35] Yu, W., Tan, J., Liu, C. K. and Turk, G. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*. Ed. by N. M. Amato, S. S. Srinivasa, N. Ayanian and S. Kuindersma. 2017. DOI: 10.15607/RSS.2017.XIII.048. URL: http://www.roboticsproceedings.org/rss13/p48.html.

[36] Ruiz, N., Schulter, S. and Chandraker, M. Learning To Simulate. *7th International Conference on Learning Representations, (ICLR) 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJgkx2Aqt7.

[37] Yu, W., Liu, C. K. and Turk, G. Policy Transfer with Strategy Optimization. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=H1g6osRcFQ.

[38] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N. D. and Fox, D. Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 8973–8979.

[39] Nagabandi, A., Finn, C. and Levine, S. Deep Online Learning Via Meta-Learning: Continual Adaptation for Model-Based RL. *7th International Conference on Learning Representations, (ICLR) 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyxAfnA5tm.

[40] Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T. and Abbeel, P. Model-Based Reinforcement Learning via Meta-Policy Optimization. *2nd Annual Conference on Robot Learning, CoRL 2018, Zurich, Switzerland, 29-31 October 2018, Proceedings*. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 617–629. URL: http://proceedings.mlr.press/v87/clavera18a.html.

[41] Mehta, B., Deleu, T., Raparthy, S. C., Pal, C. and Paull, L. Curriculum in Gradient-Based Meta-Reinforcement Learning. *ArXiv* abs/2002.07956 (2020).

[42] Eysenbach, B., Gupta, A., Ibarz, J. and Levine, S. Diversity is All You Need: Learning Skills without a Reward Function. *7th International Conference on Learning Representations, (ICLR) 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=SJx63jRqFm.

[43] Sharma, A., Gu, S., Levine, S., Kumar, V. and Hausman, K. Dynamics-Aware Unsupervised Discovery of Skills. *8th International Conference on Learning Representations, (ICLR) 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=HJgLZR4KvH.

[44] Zhang, Y., Abbeel, P. and Pinto, L. Automatic Curriculum Learning through Value Disagreement. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan and H.-T. Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/566f0ea4f6c2e947f36795c8f58
Abstract.html.

[45] Liu, Y., Ramachandran, P., Liu, Q. and Peng, J. Stein Variational Policy Gradient. *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. Ed. by G. Elidan, K. Kersting and A. T. Ihler. AUAI Press, 2017. URL: http://auai.org/uai2017/proceedings/papers/239.pdf.

[46] Todorov, E., Erez, T. and Tassa, Y. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[47] Dohare, S., Mahmood, A. R. and Sutton, R. S. Continual Backprop: Stochastic Gradient Descent with Persistent Randomness. *arXiv preprint arXiv:2108.06325* (2021).

[48] Mann, H. B. and Whitney, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60. ISSN: 00034851. URL: http://www.jstor.org/stable/2236101 (visited on 08/15/2022).

APPENDIX A: HYPERPARAMETERS

A general note with the following tables is that with the PPO algorithms, a linear baseline was used instead of a critic network, following what was done in [11]. Additionally, in structures which use policy/actor and value/critic networks, the learning rate given in the tables represent that of the actor network, while the learning rate for the corresponding value network is set to be $10\times$ the actor's learning rate. It is also noteworthy to mention that both the agent rollout batch size for on-policy algorithms and the number of particles of the Active DR method are the same as the number of workers, as each rollout and each particle is collected or represented by each worker. Finally, regarding Active DR, the *DR reward function* parameters are exclusively related to the discriminator network, with the exception of the reward scale. Also, the temperature hyperparameter refers to any temperature, be it the one used by the SVPG algorithms or the one used by the SAC algorithm.

A.1 Environments

The used environment and the associated threshold is presented in table A.1.

Environment	Timesteps (T_{env})	Randomization Dimensions (N)	Reward Threshold (r_{thr})
LunarLander-v2	1000	1 (Main Engine Strength)	200

Table A.1. *Environment Parameters*

A.2 Analysis of Active Domain Randomization Experiment

The hyperparameters used in this experiment are listed in table A.2.

Parameter	Value
General	
training timesteps	3×10^6
evaluation episodes	5
number of workers	10
Shared	
number of layers	2
discount rate (γ)	0.99
optimizer	Adam
PPO clipping parameter (ϵ)	0.2
SAC and DDPG target smoothing coefficient	0.005
Agent (DDPG)	
batch size	100
hidden dimensions	400, 300
learning rate (β)	1×10^{-3}
DR reward function	
reward scale	0.01
learning rate	3×10^{-3}
hidden dimension	128
batch size	320
non-linearity	Tanh
last layer activation	Sigmoid
loss function	Binary cross-entropy
DR policy	
rollout timesteps (T_{dr})	30
dr horizon (H_{dr})	30
batch size	256
hidden dimension	128
epochs	30
temperature (α)	0.001
max step length ($\Delta\xi_{max}$)	0.5
replay buffer size	DR rollout timesteps $\times$ no. of workers

Table A.2. *Analysis of Active Domain Randomization Experiment Hyperparameters*

A.3 Guided Domain Randomization with Meta Reinforcement Learning Experiment

A.3.1 Common Hyperparameters

Table A.3 lists the hyperparameters used that are common to all the algorithms of this experiment.

Parameter	Value
General	
training episodes	350
evaluation episodes	5
number of workers	10
Shared	
discount rate (γ)	0.99
optimizer	Adam
number of layers	2
learning rate (β) (except discriminator)	3×10^{-4}
SAC target smoothing coefficient	0.005
DR rollout timesteps (T_{dr})	30
Agent (PPO)	
hidden dimension	256
MAML adaptation step size (δ)	0.001
MAML number of gradient steps	1
PPO clipping parameter (ϵ)	0.2
non-linearity	ReLU

Table A.3. *Common Hyperparameters*

A.3.2 Active Domain Randomization

The hyperparameters of the Active DR algorithm are listed in table A.4.

Parameter	Value
DR Policy (SAC)	
hidden dimensions size	128
horizon (H_{dr})	30
batch size	256
epochs	30
temperature (α)	0.001
max step length ($\Delta\xi_{max}$)	0.5
replay buffer size	DR rollout timesteps $\times$ no. of workers

Table A.4. *Active Domain Randomization (Active DR) Hyperparameters*

A.3.3 Automatic Domain Randomization

Table A.5 presents the hyperparameters used for the Auto DR algorithm.

Parameter	Value
step size (ν)	0.2
increase threshold (t_H)	6
decrease threshold (t_L)	2
performance queue length (m)	20

Table A.5. *Automatic Domain Randomization (Auto DR) Hyperparameters*

Both the increase and decrease thresholds in the table have been slightly modified in terms of what they represent compared to their original definition in [10]. In this thesis, these parameters represent the number of rollouts in a training episode that exceed the environment's given reward threshold (as explained in subsection 4.2.2).