Reinforcement Learning using Domain Randomization and Vision-Based Feature Extraction

Anonymous AML submission

Paper ID *****

Abstract

This paper explores the sim-to-real paradigm in Reinforcement Learning by simulating a sim-to-sim scenario, with a focus on domain randomization and feature extraction from images to overcome the reality gap. Domain randomization is a technique used to increase the robustness of an agent to changes in the environment by randomizing certain aspects of the simulation. The study investigates the impact of different hyperparameters and distributions on the effectiveness of domain randomization. Additionally, the study examines the use of features extracted from images using a deep neural network as observations in the environment.

1. Introduction

Reinforcement learning (RL) is a promising approach for training intelligent agents to make decisions in complex environments. In order to avoid the negative effects that real-world training of robots entails, such as costly product maintenance and inefficient training time, the simulation-to-reality (sim-to-real) concept has arose where the idea is to transfer the learned policy in simulation to the real world. However, when it comes to training RL-agents in a sim-to-real scenario, it can be difficult to ensure that the agent will generalize to real-world situations. To address this challenge, researchers have begun to explore techniques such as domain randomization (DR) and training an agent solely on image-based input. These techniques allow the RL-agent to be trained in a variety of simulated environments, thereby increasing its robustness to real-world variations.

In this article, we simulate a sim-to-world task in a sim-to-sim scenario, utilizing the Hopper environment provided by Mujoco [4]. We delve into the use of domain randomization and vision-based feature extraction for training an RL-agent using a state-of-the-art algorithm named Soft Actor-Critic (SAC). First we examine the performance of the Hopper in the different domains by tuning hyperparameters. Then we utilize DR in order to overcome the reality gap. Finally we extract features from images to utilize as observations of the environment in order to train the Hopper.

The outline of the article is presented here. In section 2 a background to the subject is presented. The related work is described in section 3 while section 4 is constituted of our methodology. This is followed in section 5 by the experiments performed. The final part, section 6, holds the conclusion.

2. Background

In order to have a clearer idea on the embodiment of RL (RL) and its surrounded properties, we define in this section all the important concepts and terminologies that ease the understanding of the paper in hand.

2.1. Reinforcement Learning

Reinforcement Learning is an important field in the Machine Learning (ML) realm that bases its study on the dynamism of a certain object, that we refer to as agent, within certain properties that construct an **environment**. The original motivation behind this technique relies on a **decision-making** process to maximise a **reward signal** to the agent. It is widely implemented in the robotic field for its accurate representation and its diverse real-world data, it comes to the aid of robots with the self-learning approach instead of being explicitly programmed to behave a certain way.

The term **agent** represents, in RL, the decision-maker inside the environment. It takes the **current state** of the environment as an input, forms a decision based on it, and outputs an **action** for it to be performed. Generally, agents deploy a diverse level of complexity that ranges from a basic table-based technique to deep neural networks. The role of the agent relies on his ability to recognize the state of a specified environment and to learn the connection between the current state and executing the proceeding action, in order to make logical decisions and maximise the reward signal.

The environment, on the other hand, is the actual system that the agent interacts with. It is defined by four pillars:

- **The state space**: set of all possible states that the environment can be in, and the agent must represent the state of the environment in order to make decisions.
- **The action space**: set of possible actions that the agent can take in each state.
- **The reward function**: reward signal that the agent receives after taking an action in a state.
- **The transition dynamics**: in RL, it represents the agent interacting with its environment in discrete time steps which leads into a change in the state of the environment based on the performed action.

A policy in RL refers to the mapping from states to actions to determine the behavior of the agent in the environment. The policy is updated over time as the agent experiences more and more of the environment.

A mathematical representation of a basic scenario in the RL domain explains that at each time-step t the agent receives the actual state s_t and reward r_t. Consequently, it selects an action a_t from the action space which is received by the environment in hand. The environment shifts to a new state s_{t+1} and then the reward r_{t+1} is calculated through the transition (s_t,a_t,s_{t+1}). We can also represent a policy $\mathbf{P} : \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$, where $P(a, s) = Prob(a_t = a|s_t = s)$, used to maximise the reward function.

2.2. Simulation-to-Reality Paradigm

Physics simulators are well-renowned applications in RL that are used to simulate virtually physical phenomenons and help train them on RL policies because they offer a wide scale of state space and a secure environment for training and testing the agent.

In this paper, the team members are working on *the Hopper Environment*, a Mujoco-based environment. This simulator allows an agent to learn a policy without damaging a real robot and thus reduce the cost of experimentation. The goal is to make hops that move in the forward direction in order to maximise the reward function which is a combination of *the healthy reward* (every time step that the hopper is healthy) , *forward reward* (reward of hopping forward) and *control cost* (a cost for penalising the hopper if it takes actions that are too large) where: *total_reward* = *healthy_reward* + *forward_reward* - *control_cost.*

The *sim-to-real paradigm* is the transfer that occurs between what the simulation realm and the real world. However, this transfer's accuracy is deeply questioned because the simulator used may not match exactly the reality which results in something called the **reality gap**. This reality gap may directly result in poor performances in the physical environment (in the case of robots) or even partial failure of the transfer to the real world. This concern may be efficiently remediated by Domain Randomization (DR).

2.3. Domain Randomization

To solve the issue of **reality gap**, DR is considered as an efficient technique that helps the simulation-to-reality transfer be more compatible and less error prone.

DR is a technique that improves the robustness of the policy currently used, its main objective is the perturbation of the environment to reduce the mismatch with reality and enhance the entropy of the learned policy within a variety of random environments with respect to the real world. By training in diverse environments, the agent will be less sensitive to specific details of the environment.

In this project we intend to use **Uniform DR**, a specific branch of DR where the same level of randomness will be applied to all aspects of the environment (Hopper Environment) however Uniform DR may encounter some limitations:

- **How random should it be?** In fact, it is important to select the most convenient level of randomness that shows promising performance in reducing the gap between reality and simulation. The level of randomness has a crucial responsibility on the learning process and trying to prevail the important features to be well represented in the environment instead of neglecting them.
- **How do we choose the Randomization Distribution?** Selecting the most suitable distribution for the randomization can be challenging and depends on careful consideration of the environmental parameters. It can be complex to balance the need for diversity with the need for similarity to the real-world environment.
- **How real should it get?** Uniform DR can result in unrealistic environments that may not accurately reflect the real-world environment. This can result in the agent learning unrealistic or incorrect behaviors.

2.4. Vision-based RL

Vision Based RL relies on inserting raw images as state observation to the agent rather than a low-dimensional configuration vector. This gives the agent ability to learn flexibility and generality and take advantage of visual data. However, vision-based RL also presents some challenges, such as high-dimensional input spaces and the need for deep learning methods to handle large amounts of data, which eventually leads to the need for more resources to handle the high-dimensional visual data and train the agent.

3. Related Work

To reduce the bridge gap between the theoretical aspect of RL and its practical adaptation, we discuss some state-of-arts in the field of robotics and gaming that have paved the way for a better understanding of the concepts especially related to DR.

3.1. Deep Neural Networks with Domain Randomization

The work in [5] proposes a method for improving the transfer of deep neural networks related to an object detector from the simulation realm to the real world. It addresses a common solution to the reality gap dilemma through the domain randomization approach.

The method addresses the idea of interpreting directly the trained method learned from simulation only where a detector has to detect objects with different shapes, colors and textures and through a motion planner to grasp the targeted object.

Researchers have been able to reach promising results in the real world even in the presence of significant differences between the simulation and reality, such as changes in the textures, lighting, and other properties of the environment.

Then an ablation of the study is conducted where more randomization has been added to the simulated environment in order trying to improve the performance of the network in the real world.

The results show that the network trained with domain randomization outperforms both the network trained only in the simulation and the network trained in the real world.

This research paper was critically important to understand the concept domain randomization presents as major breakthrough to reduce the reality gap problem that is involved in a virtual simulation and enhance the performance in the real world of a physical topic.

3.2. Playing Atari with Deep Reinforcement Learning

This paper [3] was a landmark in the field of reinforcement learning and deep learning, demonstrating that it was possible to train neural networks to perform complex tasks such as playing video games and paving the way for further research in this area. We take inspiration from this paper to solidify our approach towards dealing with a physical phenomenon like *the hopper* in the Vision based section.

The main approach that was taken in this research was the idea of experience replay memory, a method of storing and reusing previous images or sequences of images from different games to ease the training of deep networks.

A method we would have liked to interpret in our own paper due to the lack of resources which signaled great obstacles with respect to training the deep network and especially in the computer vision part.

4. Methodology

This section will bring light to the methodology executed during the project. The steps taken, the algorithm chosen (SAC), and the strategy for uniform DR and vision-based control policies will be introduced.

4.1. Steps

The first step taken towards the development of this project was to get acquainted with the background of related work on RL frameworks in robotics and the sim-to-real transfer challenge. The materials provided were read and analyzed.

The next step was implementing and testing an algorithm for RL. The goal was to train RL agents in the gym Hopper environment. Between TRPO, SAC, and PPO, the chosen algorithm was SAC. The reason why is explained below. Two RL agents were trained, on the source and target domains respectively, and then tested and the average reward over 50 test episodes was reported. The agents were tested on source-source, target-target, and source-target.

Afterwards, uniform DR on the mass of each link of the Hopper was implemented. Here, the idea was to train the agent in a range of multiple environments such that the learned behavior may be robust to changes in the environment.

At last, in the last step, a vision-based RL technique was implemented. In this step, an RL training pipeline was created that used raw images as state observations to the agent. The agent was trained in raw images by using extracted features as observations and tested on source-source and source-target.

The implementation details are discussed below.

4.2. RL Algorithm - SAC

The algorithm chosen to train the RL agent was SAC. SAC (Soft Actor-Critic) is a deep RL and actor-critic algorithm. This means that it puts together two different types of learning algorithms, ”Actor” and ”Critic”, for a learning task. The actor is responsible for learning the policy, which maps states to actions, and tries to find the optimal policy that maximizes reward. The critic, on the other hand, is expected to evaluate the quality of the policy, by estimating the expected return for a given state or state-action pair. Thus, the critic provides feedback to the actor on how to improve the policy and the actor applies this feedback, improving the policy. Over time, they both converge to an optimal solution, reaching a policy that maximizes the expected reward.

Besides this, the SAC algorithm also uses a soft version of the Q-function and entropy-regularized RL to ensure stability and improve exploration. A soft version of the Q-function is a variant of the last, and instead of providing a deterministic estimate of the expected return, it calculates a probabilistic distribution over the possible returns. Additionally, entropy-regularized RL is a variant of RL that adds an entropy term to the function the RL is trying to optimize. This term encourages the algorithm to explore more and decreases the chances of getting stuck in a local optimum.

With SAC, agents were trained in the source environment and target environment and then tested in source-source, source-target, and target-target. The experiments carried out with the hyperparameters and the results achieved are explained in the fourth section.

4.2.1 SAC vs PPO

According to [2], PPO methods empirically seem to perform at least as successfully as TRPO. Based on that information, the group decided to not go for TRPO and chose between SAC and PPO.

One of the biggest differences between SAC and PPO is the fact that while SAC uses off-policy training, PPO uses on-policy [1]. This means that while the on-policy uses observations from the current policy exploration, the off-policy can use observations from previous explorations. On-policy algorithms tend to be more stable but data-hungry while off-policy is the other way around. There are other differences, like the entropy mechanisms and which spaces the algorithms work (PPO works in discrete and continuous while SAC works in continuous alone). Nonetheless, SAC is considered more data-efficient and that was the reason why it was chosen over PPO.

4.3. Uniform DR

In this context, uniform DR is applied by manually designing a uniform distribution over the properties of the Hopper object in the source environment and doing training with values that vary at each episode.

In this step, the group chose three different distributions, and each time the environment is reset, it samples from those distributions.

The distribution values and the results are demonstrated in the fourth section.

4.4. Vision-based Control Policies

For this part, a pre-trained AlexNet model was used as the underlying policy structure. A feature extractor class (subclass of gym.Wrapper) was created to extract the features from the model. While instantiating a new object of the class, the feature extractor is defined (in this case the AlexNet model), and the observation space and the environment are defined as well. The reset method resets the original environment and renders the environment as an RGB image with width and height matching the AlexNet images. The image is preprocessed and it consists of stacking the pixel observations along the first axis and converting it into a tensor. After preprocessing the images, it uses the feature extractor to get the features, converts them to a NumPy array, and returns them. The step method gets 4 observations rendered as images of the same size as AlexNet images and calculates the total reward. Additionally, it preprocesses the 4 observations and uses the feature extractor to take the features, transforms them into a NumPy array, and returns the sum of the observation features, the total reward, the done, and info variables from the initial environment.

The feature extractor class is then used to create a new environment. It will then be used by SAC to train the agent.

The results of this section can be found in the Experiments section.

Body part	Source	Target
Torso	2.53	3.53
Thigh	3.93	3.93
Leg	2.71	2.71
Foot	5.09	5.09

Table 1. Weights of the different body parts for the source and target domain, respectively. All weights are measured in kg.

5. Experiments

In this section the performed experiments are described and discussed. The first part investigates the upper and lower bound of the reward for the sim-to-sim task. Both training and evaluating the model in the target domain constitutes the upper bound while the lower bound is obtained by evaluating the source-trained Hopper in the target environment.

The second part implements DR to overcome the reality gap and increase the reward when transferring the knowledge to the target domain. By uniformly perturbing the different weights of the Hopper the idea is that the source domain will be just another randomly perturbed environment.

Lastly, we examine the idea of vision-based learning, where the observation of the state is inputted as features extracted from AlexNet.

The Hopper is constituted of four body parts: *torso*, *thigh*, *leg* and *foot*. The weights of these body parts, measured in kg, for the source and target domain, respectively, are displayed in table 1.

The 1kg shift in the torso between source and target domains portraits the reality gap, inferring the difficulties in creating a virtual environment consistent with the real world.

5.1. Experiment 1 - Lower and upper bound

In order to reach the lower and upper bound, several experiments were made with different hyperparameter values. There are several parameters in SAC, however, the ones tested were α (learning rate), γ (discount factor) and β (learning starts).

The parameter α (default value = 0.0003) corresponds to the learning rate of the Adam optimizer and the same

Hyperparameters	Source-Source	Source-Target	Target-Target
Default	216.31± 4.26	212.94± 3.23	267.84± 1.03
α = 0.0006	220.01± 1.39	225.60± 2.61	231.74± 2.71
γ = 0.8	173.32± 2.90	178.86± 3.79	178.86± 3.79
β = 200	279.15± 1.25	280.11± 1.36	366.08± 1.84
α = 0.0001	236.03± 1.66	245.41± 2.02	294.16± 1.99
α = 0.0001, β = 200	317.01± 0.88	323.64± 2.90	309.39± 1.93

Table 2. Mean reward with standard deviation obtained with different parameters, respectively, when evaluated for 50 epochs in the source and target domain and training steps = 10,000.

Hyperparameters	Source-Source	Source-Target	Target-Target
α = 0.001, β = 200	1677.62± 27.31	1023.58± 28.47	1672.00± 10.67
α = 0.001, β = 200	1649.00± 7.24	1587.70± 118.29	1639.78± 11.78

Table 3. Mean reward with standard deviation obtained with different parameters, respectively, when evaluated for 50 epochs in the source and target domain and training steps = 300,000.

learning rate is used in all networks. The discount factor γ, default value = 0.99, is a value between 0 and 1 that determines the importance of future rewards in comparison to the current reward. The closer the value is to 1, the more it prioritizes future rewards. The β parameter (default value = 100) corresponds to how many steps of the model to collect transitions before learning starts.

At first, the number of timesteps used for the training was 10,000, and it took between 4-6 minutes to train the model with google colab and GPU. Even though the results are not completely reliable for such a short training period, the idea is to indicate which parameters should be given more concern. The results are seen in table 2. The tables shown in this section consider that the hyperparameters are set to default except the ones indicated in the "Hyperparameters" column.

After increasing the training steps to 30,000, these results were achieved, seen in table 3. Figure 1 displays the variation of reward in the last training result in the source domain.

However, with this number of training steps, there were

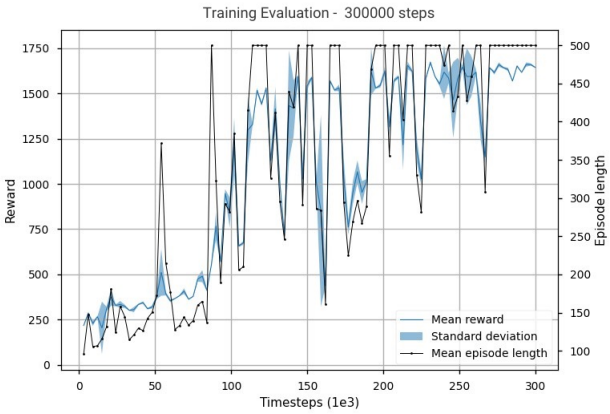

Figure 1. Mean reward, standard deviation, and mean episode length in training in the source environment.

Hyperparameters	Source-Source	Source-Target
α = 1e-5, β = 200	309.59± 22.321	324.49± 17.36
α = 1e-2, β = 300, γ = 0.95	366.34± 24.05	377.30± 1.82
α = 2e-3, β = 300, γ = 0.95	573.37± 21.15	625.70± 44.52

Table 4. Mean reward with standard deviation obtained with different parameters, respectively, when evaluated for 50 epochs in the source and target domain and training steps = 300,000.

Hyperparameters	Source-Source	Source-Target	Target-Target
α = 0.001, β = 200	1659.68± 3.96	1120.18± 201.53	1370.54± 46.47

Table 5. Mean reward with standard deviation obtained with different parameters, respectively, when evaluated for 50 epochs in the source and target domain and training steps = 500,000.

also experiments with other parameters that didn't succeed in having good results, displayed in table 4. Target-target wasn't tested because looking at the results from the agent trained in the source environment, and the lack of computational resources, it was decided to keep trying with different parameters.

Since the results obtained in table 3 were the best ones, we used the same parameters but ran with 500000 training steps. The result is in table 5. The results didn't improve even though the number of steps was significantly increased. However, in order to form a proper conclusion, more experiments should be made.

5.2. Experiment 2 - Domain randomization

In order to tackle the reality gap, DR is used by perturbing the mass of each body part except for the torso. We denote the set of all body parts, $\{\text{thigh}, \text{leg}, \text{foot}\}$, as $\mathbb{B}$. The perturbations are performed by sampling a weight-shift term ϵ_b, where $b \in \mathbb{B}$, for each of the body parts, respectively. The new weights are then updated according to $w_b' = w_b + \epsilon_b, \forall b \in \mathbb{B}$, where w_b represents the weight of the body parts displayed in table 1. Sampling is done every time the agent is reset, that is, when a new episode begins. Three different distributions, all uniform, are investigated in order to find the more prominent one to fight the curses of the reality gap.

5.2.1 Distributions

In the first scenario, $\epsilon_b \sim U[-1, 1)$, (hereafter denoted *distribution 1*) meaning that each body part can have a weight increase or decrease of at most 1kg. This interval is chosen with the aim to keep the weights of the body parts centered around their source and target value but also enable weight shifts similar to the amplitude of the torso. Thus, a smooth transaction from source to target domain is pursued.

In the second case, $\epsilon_b \sim U[0, 2)$, (hereafter denoted *distribution 2*) disallowing any decreasing weights but instead enabling a larger added weight. Since we know that the weight of the torso will increase with one kilogram, the idea is to familiar the agent with similar increments, even though on other body parts.

The third and last distribution to be studied utilizes the proportional mass shift ρ of the torso between the source and target domain. Each body part can have an equally proportional increment or decrement as the torso increment. Formally,

$$\rho = \frac{m_{t,t}}{m_{t,s}} - 1 \tag{1}$$

represents the proportional shift of the torso mass between the domains, where $m_{t,t}$ and $m_{t,s}$ is the weight of the torso in the target and source domain, respectively. The weight shift term is then sampled from

$$\epsilon_b \sim U\Big[m_b(1-\rho), m_b(1+\rho)\Big), \forall b \in \mathbb{B}, \tag{2}$$

where m_b is the mass of the corresponding body part in the source domain (hereafter denoted *distribution 3*), stated in table 1. The idea is to examine the effect of similar proportional perturbations as the torso undergoes from source to target domain, individually for the remaining body parts, rather than a static distribution. This since a static distribution could have more significant impact on an originally smaller weight (e.g. w_{leg}) than a larger one (e.g. w_{foot}).

Distribution	Source	Target
Distribution 1	1767.77 ± 5.62	1835.67 ± 96.52
Distribution 2	1624.72 ± 188.95	1667.84 ± 92.01
Distribution 3	1724.55 ± 13.34	1186.86 ± 190.35

Table 6. Mean reward with standard deviation obtained for each of the distributions, respectively, when evaluated for 50 epochs in the source and target domain, after 500,000 training steps.

5.2.2 Results

For each case, the agent is trained for 500,000 time-steps. The performance is then evaluated in both the source and the target domain for 50 epochs, respectively. The results are displayed in table 6.

At a first glance the results indicates that distribution 1 best overcomes the reality gap, followed by distribution 2 and 3, in descending order. We obtain the highest mean reward in both source and target domain by training an agent using the first distribution. The rewards for distribution 1 and 2 actually both outperforms the upper bound discovered in experiment 1. However, one should keep in mind that the performance is stochastic and there are large variations between episodes, portrayed by the standard deviations. That is, one should not assume it is more profitable to train the agent using these distributions than training in the target environment itself. For more confident results, the experiments should have been run multiple times for longer training periods. Even though, the results can be discussed to some extent.

The fact that using distribution 1 and 2 yield so high rewards can be seen as a indication that they are profound in overcoming the reality gap. Distribution 1 keeps the weights centered around their *true weights* (target weights) and uses only small perturbations. Apparently it seems to create an enough smooth transaction between the two domains to achieve high rewards when tested. Even if distribution 2 does not keep the weights centered around their target weights, this distribution enforces an environment where the weights are always increased, a property known between source and target. This could be the reason for the high reward. The rewards between the two domains for distribution 1 and 2, respectively, are relatively similar and hard to conduct any information from. This difference is significantly larger for distribution 3.

Training the Hopper using distribution 3 does not seem as a good approach to overcome the reality gap, at least when observing the rewards for the target domain. Distribution 3 allows for the largest weight variations which could give too much discrepancy between runs for the agent to learn from. However, the results in the source domain are high. This could infer that even keeping one out of four

Parameters	Source	Target
Default Distribution	39.02 ± 0.95	39.40 ± 0.90
Distribution 1	38.92 ± 0.77	38.94 ± 0.88
Default Distribution Hyper-Parameter 1	2.37 ± 0.02	3.08 ± 0.02

Table 7. Mean reward with standard deviation obtained for a uniform distribution between -1 and 1 denoted as Distribution 1,a learning rate of 0.001 with a starting point of 200 time steps denoted as Hyper-Parameter 1, respectively, when evaluated for 50 epochs in the source and target domain.

weights between training and testing can have a major impact on the results.

Once again, it needs to be stated that these results varies between runs and thus are not statistically significant.

5.3. Experiment 3 - Vision-based input

Vision-based RL relies on images as an input of the observation space instead of the original conventional vector that presents the movement of the components of the hopper.

As it is explained in the *Methodology section*, a new feature extractor environment was created with AlexNet used as the model to extract the features from the images which, in their turns, are preprocessed before extracting their features. A sequence of 4 images are used to extract the features and calculate, with each iteration its total reward signal.

Since distribution 1, that presents a Uniform distribution $\epsilon_b \sim U[-1, 1)$ (where each body can have a variation of its weight by an increase or a decrease by 1kg to keep the weights of a hopper centered), has shown the best performance in the DR process, it was also adapted in the vision-based RL section.

We will assign the name **Hyper-parameter 1** (denoted as HP1) to adopt α (learning rate) of 0.001 and the β (learning start) at 200 time steps.

5.3.1 Results

The experimentation was evaluated on 50 epochs on both source and target domain with 50,000 time-steps used and this is mainly due to how computational expensive it was to run this section. The results are shown in table 7. The comparison between using an vision based RL technique and an observation based technique showcase a significant drop in the rewards. This is mainly due the change in the input type from pixel-based observations (RGB images) and normal observation spaces, meaning that the action is taken now based on a state that comes from an image and not a normal space. This has resulted, not only with a huge gap in the rewards but also some computational complexity with the GPU.

We estimate that this problem may come from the fact of using images instead of physical positions increase the cardinality of inputs and thus decreases the total reward values. We may also raise the point regarding the use of a pretrained only AlexNet to extract the features of the observations.

We can also interpret a mere and direct fluctuation of the result with or without distributions around the 40 benchmark and this is due to the heavy load input images within 1000 feature extracted from AlexNet have on the decision process within the forwarding phase of the network.

We can also distinguish a significant decrease of the values while using hyper-parameters and so we do advise sticking with the default parameters that are set within the SAC policy which refer to a 0.003 learning rate.

6. Conclusion

After conducting a limited but deliberate search for the most optimal configuration to obtain high rewards in both the source and target domains, a set of hyper-parameters was found. The learning rate α should be set to 1e-3, the discount factor γ to 0.99 and the learning should start after 200 iterations. These parameters were utilized in the DR context. In order to bridge the reality gap in the sim-to-sim scenario between the source and target domains, three different, uniform distributions were examined. The most prominent distribution found seemed to be a uniform distribution between -1 and 1, for the given problem. However, it is important to emphasize the variations between runs and that the experiment should be performed multiple times and with a longer training period to get more confident results. Lastly, it was also discovered that using features extracted by a deep neural network from a sequence of 4 images to use as an observation in the environment was not sufficient in order to train the agent, yielding only a reward of approximately 40.

Authors:

- Karl Wennerström, s306756@studenti.polito.it;
- Leonor Gomes, s306877@studenti.polito.it;
- Peter AL Hachem, s293885@studenti.polito.it.

References

[1] AWS. AWS DeepRacer Training Algorithms - AWS DeepRacer, 2023. 4

[2] Stable Baselines3. PPO — Stable Baselines3 1.8.0a4 documentation, 2022. 4

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, Dec. 2013. arXiv:1312.5602 [cs]. 3

[4] MuJoCo. MuJoCo — Advanced Physics Simulation, 2021. 1

[5] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, Mar. 2017. arXiv:1703.06907 [cs]. 3