Robust Policy Transfer via Domain Randomization and Visual-Based Reinforcement Learning

Daniele Amato s334211
Pasqualina Annapaola Ambrosio s345061
Sonia Foco s343043
Marco Galano s338658

Abstract

This report presents a project exploring Reinforcement Learning (RL) for robotic control, moving from foundational algorithms to more advanced techniques. The work begins with testing and benchmarking basic methods, such as REINFORCE and Actor-Critic, within a fixed environment in order to establish reliable baseline performance. Subsequently, a state-of-the-art algorithm, namely Proximal Policy Optimization (PPO), is implemented and successively integrated with a Domain Randomization technique in order to enhance the robustness of the policy across varying conditions, especially to reduce the Reality Gap. The focus then shifts to visual-based RL, where the agent is trained using visual input with three different modalities. The first consists in a purely visual-based learning, training the agent exclusively on images obtained from the simulation. The second adopts a Multi-Modal Learning approach, combining visual input with environment state information, enriching model representation. Finally, the work investigates knowledge transfer between agents through Distillation Learning, offering valuable insights into visual-based RL. The code implementation is available at repository.

1. Introduction

This introduction aims to present the Reinforcement Learning paradigm by providing a simplified overview of its fundamental mathematical concepts and algorithms.

Reinforcement Learning is based on the interaction between an agent and the environment, where the agent learns its behavior by a trial and error approach. Its main goal is to find a policy that maximizes the expected cumulative reward over time. The main elements of Reinforcement Learning are:

- *Agent*: the goal-seeking learner that interacts with the environment to learn an optimal behavior;
- *Environment*: the external system with which the agent interacts by taking actions and receiving rewards;
- *State*: a representation of the current situation of the environment as perceived by the agent;
- *Action*: a decision made by the agent based on the state. The action affects the environment;
- *Policy*: the agent's strategy of behaving given a certain state;
- *Reward signal*: a signal that evaluates the result of an action taken in a given state;

The objective of Reinforcement Learning is to find a policy π^* that maximizes the value function, as expressed by:

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right]$$

where:

- $J(\pi)$ is the expected return if the agent follows policy π;
- $\mathbb{E}_{\tau\sim\pi}[\cdot]$ denotes the expectation over trajectories $\tau \in T$ induced by following policy π;
- $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ is the return or cumulative reward, with r_t that is the reward received at time step t after taking action a_t in state s_t and $\gamma \in [0, 1)$ that is the discount factor that gives more importance to immediate rewards.

The mathematics behind the Reinforcement Learning methods rely on the *Markov Decision Process (MDP)*: a mathematical model used in stochastic functions optimization and to take optimal decisions in situations with uncertainty and sequential decision-making. A MDP schema for Reinforcement Learning is shown in figure 1.

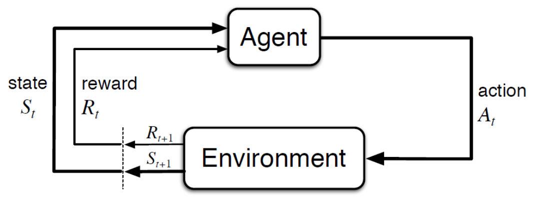

Figure 1. Markov Decision Process schema in RL.

The Markov property on which it relies is that the probability of a future state depends only on the current state. The algorithm selects actions to maximize long-term cumulative rewards, favoring strategic decisions over immediate gains.

In Reinforcement Learning, *value functions* are used to evaluate the quality of these decisions by estimating the expected return given a certain state or state-action pair, under a given policy. Two fundamental value functions are commonly used:

- *State-action value function* $Q^{\pi}(s,a)$: the expected return when starting from state s, performing action a and following policy π. It is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R(\tau)|S = s, A = a]$$

- *State-value function* $V^{\pi}(s)$: the expected return when starting from state s and following policy π. It is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(\tau)|S = s]$$

  and it can be obtained as the weighted average of the action-value function $Q^{\pi}(s,a)$ over the policy π.

These functions form the core of many RL algorithms by estimating and comparing long-term utilities to guide the discovery of an optimal policy.

One of the main challenges in Reinforcement Learning is the *exploitation-exploration* trade-off: agent should exploit acquired knowledge while exploring to discover better actions to select in the future [6]. Another persistent problem is the *sim-to-real problem*: agents trained in simulated environments often perform poorly or fail entirely when deployed in the real world. This discrepancy, namely the *Reality Gap*, stems from differences in dynamics and environmental variables that cannot be accurately modeled in simulation. In this work, this gap is modeled using a *source environment* for training under ideal conditions and a *target environment* with altered parameters to simulate real-world variability, allowing us to evaluate the agent's robustness and generalization when exposed to dynamics that differ from those encountered during training.

2. Related work

Reinforcement Learning has emerged as a powerful paradigm for robotic control, particularly in continuous control tasks simulated by physics engines, such as MuJoCo. Simple policy gradient algorithms like REINFORCE and Actor-Critic [6] have laid the foundation for learning from experience with a trial and error approach. Although they are simple, these methods often suffer from high variance and slow convergence.

To address these challenges, more advanced algorithms such as Proximal Policy Optimization (PPO) [5] have been introduced. PPO is widely used in robotic simulation environments for its ability to enhance policy update stability and performance.

One of the major challenges in applying RL for robotics is the sim-to-real transfer problem. This has been the subject of active debate and investigation in the robotics community, as highlighted in the summary of the R:SS 2020 Workshop [2]. Among the proposed solutions, domain randomization has gained significant attention. This approach introduces stochastic variations in simulation parameters, such as friction coefficients, lighting conditions, object textures or sensor noise, during training. The idea is to expose the policy to a wide distribution of scenarios, improving its robustness and generalizability to real-world conditions. Important works in this area have demonstrated the practical effectiveness of domain randomization for transferring policies from simulation to reality, making it a promising strategy. [4, 7].

Another strategy to reduce the sim-to-real gap is Visual-Based RL, which involves training policies using visual inputs of the robot and its environment. Image-based policies focus on visual patterns that are less dependent on simulator-specific dynamics and therefore more transferable to the real world. In addition, camera-based inputs are typically more accessible and affordable than traditional sensors. This method was first successfully applied on Atari video games, where an agent trained with only raw pixels was able to achieve a performance level comparable to that of a professional human tester [3].

Other approaches combine visual inputs with sensor readings in a *Multi-Modal* context, enhancing the state-based policies with more valuable information about the robot. It has been shown that integrating information from images and proprioception can improve the performance on challenging locomotion tasks compared to solely using one single modality [10].

Furthermore, another approach that can be useful to enhance transferability is *knowledge distillation* [1]. This involves training a *Student* agent with constrained inputs to imitate the behavior of a pre-trained *Teacher* agent that has access to privileged information. In the context of sim-to-real transfer, this approach is particularly interesting since it is possible to train a high-performing teacher in simulation using full-state information and then distill its behavior into a visual-based student that can operate in the real world using only camera inputs, as demonstrated in [9].

3. Methodology

3.1. Gym Hopper environment

All the experiments of this work were run on the Hopper environment from OpenAI Gym, powered by the MuJoCo physics engine. As shown in Figure 2, the Hopper is a onelegged robot model with four body parts: torso, thigh, leg and foot. It moves in two dimensions. The goal is to make the robot learn how to jump in the forward direction without falling, by applying torques on the three hinges connecting the four body parts. The environment is characterized by:

- a continuous *state* space consisting of 11 values ranging in $(-\infty, \infty)$, that represent various parameters such as the height of the hopper, the angles of the joints, the linear velocities and the angular hinge velocities;
- a continuous *action* space consisting of 3 values in the range $[-1, 1]$, that respectively represent the torques applied to the thigh, leg and foot rotors.

Each episode starts with the Hopper standing in a stable and upright position, defined by some specific observation values, and it ends either after a fixed number of *timesteps* or earlier if the robot falls, specifically when the agent violates the healthy state conditions, which are described by a range of certain values of the observation space. At each timestep t, the Hopper takes an action according to the given policy to maximize the returned reward. The reward is given by three elements: a *healthy reward* if the robot is in the healthy state, a *forward reward* which is related to how much it hops forward and a *control cost* that penalizes taking large actions.

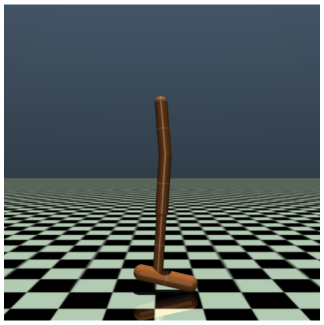

Figure 2. Hopper environment

3.2. REINFORCE (Vanilla Policy Gradient)

The REINFORCE algorithm is a policy gradient learning method that directly optimizes the agent's stochastic policy $\pi(A|S, \theta)$, where θ denotes the policy parameters. In general, the goal of policy gradient methods is to maximize the expected return of the rewards:

$$\max_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$$

The policy parameters θ are updated using gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

where α is the *learning rate*.

The REINFORCE algorithm estimates the gradient of the expected return using Monte Carlo methods by sampling complete episodes. The gradient is then estimated as:

$$\nabla_\theta J(\theta) \approx \hat{g} = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \, R(\tau)$$

Instead of the cumulative reward $R(\tau)$, this approach computes a time-dependent discounted return G_t that accumulates reward starting from time step t. The return is discounted by a factor $\gamma \in [0, 1]$ that controls the importance of immediate rewards. This leads to the update rule:

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

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \ln \pi(A \mid S)$$

In this project's implementation, the policy network is a neural network that returns a probability distribution over actions a_t, given an observed state s_t. The architecture of the network consists of an input layer, a single hidden layer with 64 neurons and a fully connected layer, that outputs the mean of a Gaussian distribution for each action dimension. Both the input and hidden layers are activated by the hyperbolic tangent (*tanh*) activation function. In addition, the standard deviation of the Gaussian is a trainable parameter vector that is activated by the softplus activation function to guarantee positivity.

Given that the gradient estimation in the basic REINFORCE method suffers from high variance, a widely used technique is to subtract a *baseline* $b_s(t)$, which is typically a constant value, from the return G_t in order to reduce the variance while keeping the estimator unbiased. The update rule becomes:

$$\theta_{t+1} = \theta_t + \alpha (G_t - b_s(t)) \nabla \ln \pi_\theta(A \mid S)$$

3.3. Actor-Critic

Actor-Critic is a policy gradient learning method that combines two components: the *Actor*, responsible for selecting actions under a policy $\pi_\theta(A|S)$, and the *Critic*, which evaluates the goodness of these actions by estimating a state-value function $V(s_t)$. The actor updates the policy based on feedback from the critic, while the critic updates the value function parameters.

At each time step, the actor samples an action a_t given the current state s_t according to a policy $\pi_\theta(A|S)$ and, after executing the action, the environment returns a reward r_t and the next state s_{t+1}. The critic then estimates the advantage function $adv(s_t)$ as the difference between the discounted rewards G_t and the state-value function $V(s_t)$:

$$adv(s_t) = G_t - V(s_t)$$

The actor updates the policy parameters θ using the policy gradient:

$$\theta_{t+1} = \theta_t + \alpha \, adv(s_t) \nabla_\theta \log \pi_\theta(A|S)$$

where α is the learning rate of the actor.

In this project, the actor and critic policies are implemented as two different neural networks. The actor network is the same used in the REINFORCE algorithm 3.2, while the critic network differs only for the output size of the last fully connected layer which outputs a single value, representing the estimated value function. During the update phase, the discounted returns G_t and the advantage estimates $adv(s_t)$ are computed. Then the actor updates the policy and the critic updates its parameters ϕ by minimizing the Mean Squared Error between predicted values and discounted returns:

$$\phi^* = \arg\min_\phi \frac{1}{N} \sum_{t=1}^{N} \left(V_\phi(s_t) - G_t\right)^2$$

3.4. PPO

Proximal Policy Optimization (PPO) is a state-of-the-art policy gradient algorithm designed to optimize performance while ensuring stable and conservative policy updates. Among its variants, PPO-Clip is the most widely used due to its simplicity and effectiveness.

The central idea is to limit the extent to which the policy can change between updates, thereby avoiding instability during learning. This is accomplished through a clipped surrogate objective function. Given the probability ratio $r_t(\theta)$ between new and old policies:

$$r_t(\theta) = \frac{\pi_\theta(A|S)}{\pi_{\theta_{old}}(A|S)},$$

the clipped objective is defined as:

$$L^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta) \cdot \text{adv},\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \text{adv}\right)\right]$$

where adv is the advantage estimate, measuring how much better or worse taking an action in a given state results with respect to the expected return of the current state given the current policy, and ϵ is a hyperparameter, namely the *clip range*.

This objective function restricts policy updates in the following way: if the advantage adv is positive, the policy is encouraged to increase the probability of the selected action, but only up to a factor of $(1+\epsilon)$. Conversely, if adv is negative, the policy should reduce the probability of the action, but not below $(1-\epsilon)$. This clipping mechanism prevents excessively large policy updates, which can otherwise degrade performance.

3.5. Domain Randomization

Domain Randomization (*DR*) in Reinforcement Learning is a powerful technique to enforce the model to learn a more robust policy. The Gym MuJoCo Hopper environment offers of a *source* and a *target* environments. These two differ for the weight of the *torso* of the Hopper, which is $\approx 2.474kg$ for the source environment and $\approx 3.534kg$ for the target. The model is trained in the source environment and tested in the target environment, and the weight change of the torso body part represents a shift between the simulations and the real world that cannot be modeled in advance.

To improve the robustness of the model to mass-shifting, a DR technique can be adopted: each body part is expressed as a random variable with a certain distribution. Before the beginning of each episode, the masses of all Hopper's components, except for the torso, are randomly sampled from the chosen probability distribution.

With the DR technique the agent interacts in each episode with different masses, ensuring that the optimized policy is less dependent from components' weights, and consenting the model to have a more robust approach to the target environment.

The proposed probability distributions are:

- Uniform: $\mathcal{U}(a = (1-p)m,\ b = (1+p)m)$
- Normal: $\mathcal{N}(\mu = m,\ \sigma^2 = (p\,m)^2)$
- Log-normal: $\mathcal{L}(\mu = m,\ \sigma^2 = (p\,m)^2)$

Where m is the mass of the component, and p is a hyperparameter.

3.6. Visual Reinforcement Learning

Visual Reinforcement Learning is based on training agent policies using pixel data from images. Since learning directly from high-dimensional images is highly computationally expensive, the goal is to find an accurate but simplified and compact representation that the agent can use to learn a control policy. [8]. Convolutional Neural Networks (CNNs) are typically used for this purpose, as they can effectively capture complex structures and shapes, enabling the agent to recognize relevant elements of the environment, such as the contour and orientation of the hopper.

In this implementation, the agent receives a temporal stack of 8 rendered frames, analogous to images captured by a camera in a real world scenario, and must infer its configuration and dynamics from these pixels. The raw RGB frame delivered by MuJoCo is passed through a preprocessing pipeline before it reaches the policy network:

- *Cropping*: the image is cropped to keep only the central area with the hopper, removing the sky, ground, and side background;
- *Grayscaling*: the brown color is isolated to highlight the hopper's silhouette, then the image is converted to grayscale;

- *Spatial resizing*: the grayscale frame is resized to 84×84 px to balance detail and computational efficiency.

Finally, the image is converted to a floating-point tensor with values normalized to [0, 1] and appended to a rolling buffer that holds the eight most recent frames. The resulting observation tensor with shape 8 × 84 × 84 is passed through a custom CNN inspired by the one used for Atari [3]. The network consists in 3 hidden layers, each one activated by a ReLu activation function, and a final flatten layer:

- *first layer*: 32 filters with kernel size 8 and stride 4, the output is $32 \times 20 \times 20$;
- *second layer*: 64 filters with kernel size 4 and stride 2, the output is $64 \times 9 \times 9$;
- *third layer*: 64 filters with kernel size 3 and stride 1, the output is $64 \times 7 \times 7$;
- *flatten layer*: the output is a tensor of size 3136.

Visual Reinforcement learning was explored by implementing three different approaches.

3.6.1 Visual-only approach

The first strategy focuses on training the PPO algorithm solely on raw pixel observations. To reduce the dimensionality of the visual features extracted by the CNN, a linear transformation is applied in order to map the output vector of size 3136 to a more compact representation of size 512, which is then followed by a ReLU activation function. This vector is provided as input to the PPO algorithm that learns a control policy accordingly.

Training directly from high-dimensional images is computationally expensive and can be sample inefficient. Therefore, although a visual-only approach can be useful for addressing the sim-to-real gap since it does not rely on observation states, further alternative strategies to improve learning efficiency and performance were explored.

3.6.2 Multi-Modal approach

This work then focuses on the combination of observation states and images as input to the PPO algorithm with a Multi-Modal approach. Once the images are processed by the CNN, they are concatenated with the state observation vector of dimension 11. Then a linear transformation is applied to the vector of dimension 3136+11 reducing it to a dimension 512, followed by a ReLU activation function. This representation is provided as input to the PPO algorithm that learns a control policy accordingly.

This strategy can enhance sample efficiency by leveraging the use of multi-modal inputs, but it still requires full access to observation states.

3.6.3 Knowledge Distillation

The final strategy adopted is knowledge distillation, which is a technique used to transmit knowledge from a larger model, the *Teacher*, to a smaller one, the *Student*. The teacher model computes its policy using the PPO algorithm trained on state observations. Once trained, the teacher model is used to generate a dataset of images paired with the corresponding actions that the agent would take. This dataset is then used to perform supervised training on the student model, which consists of two components:

- a *feature extractor*, which processes input images with a CNN and outputs a feature vector. This is implemented with the same CNN used in the other visual approaches and then followed by a linear transformation that outputs a feature vector of dimension 64, instead of 512. This choice was made to limit the model's complexity during supervised training;
- a *Multi-Layer Perceptron* head, which takes the feature vector and predicts the corresponding action. In particular, it consists of two fully connected layers, the first one followed by a ReLu activation function while the second one by a Tanh. The first linear layer keeps the feature dimension at 64, while the second one maps it to 3 dimensions, to match the action components. Finally, the Tanh is used to normalize the output values within the range $[-1,1]$.

The objective of this training is to enable the student to imitate the teacher's actions using only visual inputs. At each epoch, the model iterates over batches of 64 images and, for each batch, it produces action predictions. These predictions are then compared to the teacher's actions using a Mean Squared Error (MSE) loss. Then, the optimizer updates the model parameters to minimize this loss. The training runs for 20 epochs, allowing it to learn effectively without becoming too computationally intensive. After each epoch, the average loss is monitored to ensure that the model is learning effectively.

Finally, the supervised learning approach is used in a reinforcement learning setting. To do this, a new agent is trained on the image-based environment using a PPO algorithm, as in the visual-only approach, but now the agent is initialized with the pre-trained feature extractor obtained through distillation. During training, the parameters of the CNN are frozen, in order to preserve the visual representations learned in the supervised phase, letting the agent focus only on improving its decisions rather than also learning to interpret visual inputs.

This approach trains the teacher with full state data and allows the student to rely only on visual inputs at test time, making the final policy suitable for real-world applications relying on images.

4. Experiments and results

This section presents the conducted experiments and the corresponding evaluation of models' performance.

4.1. REINFORCE and Actor-Critic evaluation

The first experiments focused on REINFORCE and Actor-Critic algorithms. It is important to note that in this phase of the experiments the models were both trained and tested on the source environment, as the scope of this part of the project is to show how these less complex policies work.

For the REINFORCE algorithm, the model was trained both without baseline and with a constant baseline $b = 20$. For each algorithm, the model was trained for 100 000 episodes. The rewards were averaged across four runs for each algorithm due to the high fluctuation of returns that was observed in an initial phase. The moving average of the cumulative rewards with confidence intervals of each episode during training phase is shown in Figure 3. It can

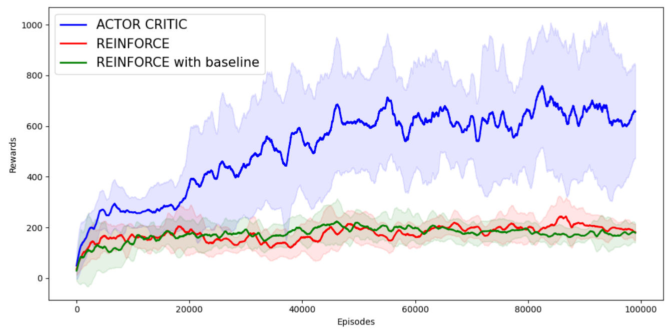

Figure 3. Training rewards of REINFORCE, Actor-Critic.

be seen that REINFORCE and REINFORCE with baseline are almost overlapping. The rewards saturate around 15 000 episodes for a value of approximately 200. In contrast, the rewards given by the Actor-Critic algorithm are significantly higher, even though suffering from greater standard deviation. The rewards saturate around 50 000 episodes for a value of approximately 800.

After the training phase the learned policies were evaluated over 50 episodes. The average rewards and corresponding standard deviations are reported in Table 1. It can be noted that these results are coherent with the expectations from the training rewards.

4.2. PPO evaluation

Subsequently, the project focused on the implementation of a RL pipeline involving a PPO algorithm. The agent and policy implementations for this algorithm come from the library *Stable-Baselines3*. In this phase it was dedicated a particular attention to the *Reality Gap*.

Table 1. Average reward and standard deviation over 50 evaluation episodes.

Algorithm	Mean Reward	std
REINFORCE	158.02	30.77
REINFORCE + Baseline	149.59	16.79
Actor-Critic	791.06	297.89

The first task consisted in performing a Grid Search over three hyperparameters:

- *n_steps (NS):* The number of timesteps for each episode, after which the policy is updated. The tested values were NS $\in \{2048, 4096\}$;
- *learning_rate (LR):* The learning rate of the algorithm. The tested values were LR $\in \{3e-4, 1e-3\}$;
- *clip_range (CR):* How much the new policy can deviate from the old one. The tested values were CR $\in \{0.1, 0.2, 0.3\}$.

This model selection was done on an agent trained and validated in the source environment after a training lasted 1 000 000 timesteps. The results are shown in Table 2.

Table 2. Grid search results over hyperparameters NS, LR, CR.

NS	LR	CR	Mean Reward	std
2048	3e-4	0.1	1267.51	28.82
2048	3e-4	0.2	1708.03	3.34
2048	3e-4	0.3	1715.42	9.29
2048	1e-3	0.1	1705.08	122.22
2048	1e-3	0.2	1537.92	287.56
2048	1e-3	0.3	678.55	118.49
4096	3e-4	0.1	1736.12	2.02
4096	3e-4	0.2	1750.51	25.52
4096	3e-4	0.3	1712.40	18.53
4096	1e-3	0.1	1731.82	64.47
4096	1e-3	0.2	1723.12	2.66
4096	1e-3	0.3	1301.06	295.53

In particular, it can be seen that the algorithm is not much sensitive to the choice of hyperparameters, which was expected due to the high stability of the PPO algorithm. However, the chosen configuration for the following steps was $NS = 4096, LR = 3e-4, CR = 0.1$.

Building upon these results, some evaluation bounds for the following DR phase are established:

- *Lower Bound (LB):* The LB is the performance of the agent trained in the source environment and evaluated

in the target environment. It represents the worst possible score that the agent can achieve without DR. This is an interesting insight into the problem of the Reality Gap.
- *Upper Bound (UB):* The UB is the performance of the agent trained and evaluated in the target environment. It represents the best score that the agent can achieve, but it is unrealistic due to the fact that it assumes that the agent manages the Reality Gap, which is impossible in practice.

The bounds for the DR phase are shown in Table 3.

Table 3. Upper Bound and Lower Bound for DR phase.

Bound	Mean Reward	std
LB	804.28	6.88
UB	1714.23	5.45

4.3. Domain Randomization evaluation

In this phase the work focused on improving the performance of the model trained in the source environment for 1 000 000 timesteps and evaluated in the target environment, i.e. optimizing the LB.

A Grid Search was performed for choosing the best probability distribution among the ones listed in 3.5. For this purpose the agent was trained and validated in the source environment, in order to not overfit the policy on the target environment.

The tested values for the hyperparameter p were $p \in 0.1, 0.4, 0.7, 1$. The results are shown in Table 4.

Table 4. Grid search results over distributions Uniform, Normal, Log-Normal and hyperparameter p.

Distribution	p	Mean Reward	std
uniform	0.1	1720.11	5.87
uniform	0.4	1754.92	8.87
uniform	0.7	1714.11	2.98
uniform	1	1688.43	2.51
normal	0.1	1734.95	3.49
normal	0.4	1723.69	6.85
normal	0.7	1071.50	23.23
normal	1	893.43	23.89
log-normal	0.1	1764.12	4.56
log-normal	0.4	1709.21	8.76
log-normal	0.7	1614.82	7.85
log-normal	1	1681.26	2.10

Based on these results, the chosen distribution was the Log-normal with parameter $p = 0.1$.

$$\mathcal{L}(\mu = m,\, \sigma^2 = (0.1\, m)^2)$$

A final evaluation over 50 episodes of the agent trained in the randomized source environment and tested in the target environment brought the following results:

Mean Reward: 1648.01, *std:* 158.46

4.4. Visual-based approaches

This part of the work focused on training a RL policy with images with different strategies.

4.4.1 Visual-Only Approach evaluation

The trained model is PPO with a *CNNPolicy* and parameter $NS = 1024, LR = 3e-4, CR = 0.1$ over 1 000 000 timesteps in the source environment. A final evaluation over 50 episodes of the agent evaluated in the source environment brought the following results:

Mean Reward: 190.07, *std:* 7.78.

4.4.2 Multi-modal Approach evaluation

The trained model is PPO with a *MultiInputPolicy* with parameter $NS = 1024, LR = 3e-4, CR = 0.1$ over 1 000 000 timesteps in the source environment. A final evaluation over 50 episodes of the agent evaluated on the source environment brought the following results:

Mean Reward: 713.30, *std:* 56.24.

4.4.3 Distillation evaluation

For this experiment, the *Teacher* was trained in advance in the source environment, for 1 000 000 timesteps with the configuration found in Section 4.2, which was $NS = 4096, LR = 3e-4, CR = 0.1$.

Subsequently, from the Teacher model, images and corresponding taken actions are extracted, in order to build the database for the *Student*. The Student is first evaluated after the distillation training phase, obtaining over 50 episodes:

Mean Reward: 513.47, *std:* 159.23.

Later the model is fine-tuned in a reinforcement learning setting. The model obtained over 50 episodes:

Mean Reward: 438.47, *std:* 2.34.

5. Discussions and findings

The experimental results conducted in this work provide interesting insights for each of the illustrated approaches.

The comparison between REINFORCE, Actor-Critic and PPO algorithms shows significant differences in performance. Although the logic behind REINFORCE with its baseline variant is simple to understand, it shows limited effectiveness. Additionally, no appreciable differences can be noted between the basic REINFORCE and the variant with baseline. The Actor-Critic method demonstrates substantial improvement, with a mean reward of 791.06, even though suffering from higher standard deviation, possibly pointing out the potential instability of this foundation algorithm.

These limitations are effectively addressed by PPO algorithm, which completely outperforms the previous two algorithms, providing not only higher mean rewards but also a smaller variability. In fact, the LB performance of PPO surpasses the average reward of Actor-Critic. Furthermore, the UB, obtained in the same conditions of the foundation algorithms, is more than twice the score of Actor-Critic. Moreover, PPO was found to be faster than the previous algorithms in training.

The implementation of a Domain Randomization technique tries to bridge the sim-to-real gap, working on the robustness of the policy. The best performance is obtained with a Log-normal distribution with hyperparameter $p = 0.1$, with mean reward of 1648.01, which can be considered satisfactory compared to the upper bound of 1714.23. This upholds that the introduction of Domain Randomization is an effective way to obtain good performances notwithstanding the real world differences.

In the Visual-Based Reinforcement Learning extension, different behaviors were observed across the adopted strategies. The image-only policy managed to yield modest rewards, as its mean reward was 190.07, similarly to the performance obtained with REINFORCE. This is possibly due to the complexity of the task and the small computational capacity available, making it hard to train a policy on raw pixel data only. On the other hand, the multi-modal approach improved the performance, reaching a mean reward of 713.30, leveraging both visual information and states. Although this result is still lower than the one obtained training on states alone, it shows that combining modalities can act as a valid compromise in scenarios where full state information is partially available or unreliable. However, collecting information about both states and images is computationally expensive and in a real world scenario requires more expensive devices to collect it.

For this reason, the distillation-based strategy appears to be a promising approach for real-world applications, transferring knowledge from a well-performing Teacher to a visual-only Student. It achieves a mean reward of 513.47 with a standard deviation of 159.23 with the policy obtained after the supervised learning, while it brings a mean reward of 438.47 with a standard deviation of 2.34 after the fine-tuning phase, suggesting that the fine-tuned policy is more robust due to the lower variance. Despite not reaching the peak performance of the Multi-Modal approach, the rewards are more than twice that of the images-only model, demonstrating a substantial improvement. Most importantly, unlike the multi-modal setup, this approach does not require access to state information during runtime, as states are only needed to generate the training dataset.

These findings collectively indicate that integrating multiple sources of information and adopting advanced transfer learning techniques significantly enhance the real-world applicability of Reinforcement Learning solutions in robotics.

It is also worth noting that all experiments were carried out on limited computational resources, which suggests a significant margin for potential improvements with more powerful hardware available. This could open up promising directions for future work. One possibility is to apply visual domain randomization during training, introducing stochastic variations in the rendered images, such as changes in colors, lighting conditions or viewpoints of the camera, to improve generalization to real-world conditions. Another strategy is to implement deeper or pre-trained CNNs that might provide more effective feature extraction from visual inputs.

References

[1] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 2015. 2

[2] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Florian Golemo, Mehdi Mozifian, et al. Perspectives on sim2real transfer for robotics: A summary of the r:ss 2020 workshop. 2020. 2

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. 2, 5

[4] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. 2018. 2

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. 2

[6] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. 2

[7] 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. 2017. 2

[8] David Valencia, Henry Williams, Yuning Xing, Trevor Gee, Minas Liarokapis, and Bruce A. MacDonald. Image-based deep reinforcement learning with intrinsically motivated stimuli: On the execution of complex robotic tasks. 2024. 4

[9] Jun Yamada, Marc Rigter, Jack Collins, and Ingmar Posner. Twist: Teacher-student world model distillation for efficient sim-to-real transfer. 2023. 2

[10] Bang You and Huaping Liu. Multimodal information bottleneck for deep reinforcement learning with multiple sensors. 2024. 2