Sim-to-real transfer of Reinforcement Learning policies in robotics

Alberto Gagliardi
Politecnico di Torino
Turin, Italy
s302243@studenti.polito.it

Riccardo De Cesare
Politecnico di Torino
Turin, Italy
s299955@studenti.polito.it

Luca Zaccaria
Politecnico di Torino
Turin, Italy
luca.zaccaria@studenti.polito.it

***Abstract*—The goal of the project is to use state-of-the-art reinforcement learning algorithms to train control policies for a robot in simulation. We experimented the sim-to-real transfer problem, represented by a manually injected discrepancy in a dynamic parameter between two domains, namely source and target. We found a robust control policy using Uniform Domain Randomization (UDR) and tested the policy in different conditions. Finally we trained a control policy based solely on visual inputs to understand the difference in training and performances with respect to the previous approach.**

***Index Terms*—Reinforcement Learning, Robotics, Uniform Domain Randomization, Proximal Policy Optimization**

I. Introduction

Nowadays Reinforcement Learning has become an intriguing reserarch field of Machine Learning. Its final goal is to train an agent (controller) accomplishing a certain task and whose performances are based on a received reward. Starting from basic strategic table games like Go, it is being adopted in the Robotics field, which certainly is more difficult than the previous applications. From a practical point of view the reason why generally Reinforcement Learning algorithms are not trained directly on real robots is due to the fact that collecting a large number of experiments is pretty costly not only in terms of hardware components but also for time and work required. The point towards which most of the RL community is struggling to move is the one where the real-world interactions are extremely limited, so the policy describing our agent should be almost completely obtained from simulation design as discussed in [1]. Another aspect to take care of is the safety, as the robot in the training phase could show unexpected behaviours. The work to carry out in finding a good policy should take into account these constraints [2]. The last key point of this feasibility study concerns the so called sim-to-real gap which can be simply seen as an under-modelling of the real system with the addition of the model parameters uncertainty [1]. To tackle this problem we adopted Domain Randomization (DR) among all the physical parameters, that is a common standard as working path. More specifically we focused our attention on two simple but effective type of distribution upon which basing the randomization: the Uniform Domain Randomization (UDR) and the Normal Domain Randomization (NDR). The last objective aims to a different way of training the agent feeding it not with the physical states but with raw images that will involve a CNN network as base structure for the policy.

II. Background

In Reinforcement Learning (RL) the theoretical framework of Markov Decision Process (MDP) is generally used. A MDP $\mathcal{M}$ can be defined over 2 main spaces:

- Action space $\mathcal{A}$
- State space $\mathcal{S}$

These 2 spaces can be continuous or discrete and have different dimensions. The other main ingredients are:

- Initial state distribution $p(s_0)$
- State transition distribution $p(s_{t+1}|s_t, a_t)$
- Reward distribution $p(r_t|s_t, a_t, s_{t+1})$

The agent interacts with the environment selecting at each (discrete) time-step an action to be performed, a vector $a_t \in \mathcal{A}$, according to a policy $\pi(a|s)$, that is also a probability distribution. The state transition distribution completely define the environment behaviour, while the reward distribution is generally manually defined, leading to the well known problem of reward shaping. We can define the expected discounted return as

$$G_t = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \tag{1}$$

Where $\gamma \in [0, 1]$ is called discount factor and define the present value of future rewards. The general objective of RL is to a find an optimal policy $\pi^*(a|s)$, such that the expected G_t is maximized. A key hypothesis in MDP framework is that the process has the so called Markov property, that is the state must include complete information about all aspects of the past agent-environment interaction that are relevant for the future. A representation of MDP is reported in Fig. 1.

III. Related work

Recalling the sim-to-real gap, the RL community is now trying to take advantage from both simulation and real-world positive aspects. If on one hand we would like to train the agent in a simulated environment to avoid problems concerning safety and time-money waste, on the other hand the simulators could not provide the characteristics that are difficult to model of the real-world, including: non-rigidity physical effects, wear-and-tear or high-fidelity sensors [3].

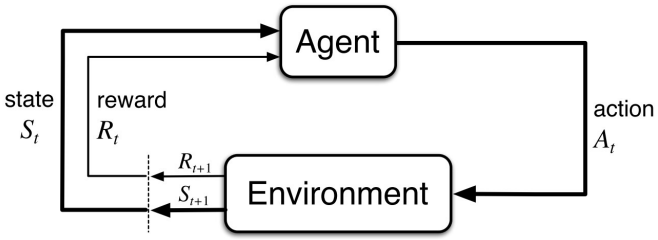

Fig. 1. MDP simplified framework

This issue pushed us to make use of one of the most used and latest finding in RL, namely the Domain Randomization. The idea here is to train our agent in a simulation environment with physical parameters changing at each episode in order to lead to a policy able to tackle a set of possible real characteristics of the target environment (uncertain by definition). The state of the art of Domain Randomization includes several well-suited expedients. DROPO [4] is an off-line method based on the capture of a dataset of real sampled trajectories used as target to update the parameters distribution, it is a quite innovative way of obtaining a good distribution from which selecting our parameters. We are more focused in finding an optimal static distribution to sample our parameters. This is motivated by the fact that an adaptive method will involve the need of acquiring a data collection of real experiments to best fit the online-computed distribution. But in the case one has not the possibility of a real hardware try, a static distribution can be an optimal trade-off to have a basic application of DR. In the last few years this research field has been widely explored pulling out several interesting innovations, like for example the DiDoR method [5], an intriguing approach aimed to obtain a final policy by imitating a certain number of teaching policies for each domain described by a specific set of parameters. Although these kind of approaches are quite recent, Uniform Domain Randomization (UDR) can be a convenient solution in order to get significant results while keeping a low complexity of the overall algorithm. This is a static method where at each episode, a certain bunch of parameters is uniformly selected according to upper and lower bounds for each of them. Our task is then to provide suitable intervals that let policy obtained from the randomized source be able to fit the target environment in terms of final return. Moreover for sake of completeness we tried to establish a comparison with another basic probability distribution, that is the Gaussian one, leading to Normal Domain Randomization (NDR). This choice is also interesting because it aims to give more importance to the mean values which in this case are the nominal values of the three masses.

Another important innovation in the sim2real transfer is the training with images. This method was applied the first time with success in Atari games [6]. In particular, they tested vision-based RL on seven different Atari games: in six of them outperformed all previous approaches and in three of them surpassed a human expert.

In synthesis this implementation allows to train an agent with a network that can see the time evolution of the environment due to a stack of images.

IV. Methods

Among the different possible algorithms available we chose to use Proximal Policy Optimization (PPO) [7]. This choice is based on the fact that this is a policy optimization algorithm, which means that it actually performs optimization on the most intuitive and direct possible objective in the RL framework. We preferred PPO to Trust Region Policy Optimization (TRPO) [8] because PPO is a first order method meant to improve TRPO, which on the other hand is a second order method.

A. PPO introduction

PPO is an on-policy algorithm which main focus is to take the largest possible step of Stochastic Gradiend Ascent (SGA) without causing performance collapse. The possible performance deterioration is due to the fact that SGA is performed on an approximation of the real objective to be optimized, called surrogate objective. The surrogate objective introduced by [7] is taken directly from [8], with the key modification of the constraints to take into consideration.

B. Key equations and notation

We denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, in which $\pi_\theta(a_t|s_t)$ represents the probability to take an action a_t given the state s_t. The probability ratio resumes the policy changes after an update. TRPO surrogate objective is

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}[r_t(\theta) \cdot \hat{A}_t] \tag{2}$$

In which $\hat{A}$ is defined starting from the advantage function

$$A_{\pi_\theta}(s_t, a_t) = Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t). \tag{3}$$

Advantage function (3) is an estimate of how much is convenient to take a certain action, as it compares the Q-function, representing the state-action value, and the value function. To impose the trade-off between bias and variance of the considered estimator a new version of the classical advantage function is considered [9]. In order to reduce the variance of the estimator a normalized version of advantage function is often introduced

$$\hat{A_\pi} = \frac{A_\pi - \mu(A_\pi)}{\sigma(A_\pi)} \tag{4}$$

The main objective proposed by [7] is the following

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[min(r_t(\theta) \cdot \hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t)] \tag{5}$$

The clip function serves as a regularizer, due to the fact that all updates that are over a certain threshold, selected changing the hyper-parameter ϵ, are cut off. The complete objective introduced by [7] uses two other terms

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}[L_t^{CLIP}(\theta) - c_1 \cdot L_t^{VF}(\theta) + c_2 \cdot S[\pi_\theta](s_t)] \tag{6}$$

where c_1 and c_2 are coefficient that respectively ensure value function fitting and act as entropy bonus. In particular c_2 is very useful if we want to encourage a more explorative behaviour. In order to have a reliable version of the PPO algorithm we used the stable-baselines3 library.

C. Uniform Domain Randomization

Uniform distribution is one of the most common way of selecting a certain parameter within its own cloud of uncertainty. This choice is motivated by the fact we do not know the effective value of each parameter and so this kind of probability better described the chance of getting any specific possible environment.

$$\zeta_k \sim \mathcal{U}\ \{\zeta_{k_{min}},\ \zeta_{k_{max}}\} \tag{7}$$

From (7) we get the random parameters that will be used to described the specific environment to train for one single rollout. The randomization will be applied again until we reach the prescribed number of total episodes. The policy obtained by this continuous change in source environment parameters will be evaluated in the target domain as shown in Fig. 2.

D. Normal Domain Randomization

A possible intuitive alternative to UDR approach could be the use of a normal probability distribution, from which we sample our parameters.

$$\zeta_k \sim \mathcal{N}\ \{\mu_k,\ \sigma_k\} \tag{8}$$

As in the previous method we update the parameters at the end of each episode. In this case we properly select the mean and standard deviation values of the corresponding parameter as described in (8). From a practical point of view this option can be justified by thinking that any kind of production standard tries to realize the specific component with values as close as possible to the nominal ones.

E. Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of deep learning network. It consists of multiple layers:

- convolutional layers, apply filters to scan the input image and extract features;
- activation layers, apply a non-linear function to introduce non-linearity into the model;
- pooling layers, downsample the feature maps to reduce their size;

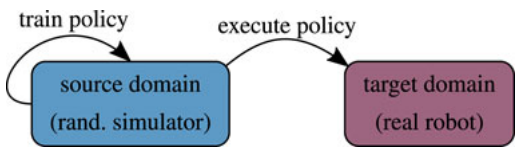

Fig. 2. Sim-to-Real with DR

- fully connected layers, which perform the final linear mapping to obtain the desired outputs.

CNNs have proven to be effective in tasks such as image classification, object detection and they are widely used in computer vision.

F. Vision based reinforced learning

Vision based reinforcement learning leverages the power of computer vision techniques to extract relevant features from raw visual inputs and improve the performance of RL agents. In particular it uses an observation space composed by images and a CNN-based policy $\pi(a|s)$, whose input are raw pixels and as output has an action vector a_t.
Normally the input of the network is composed by a stack of images, this allow the network to get some information about the time evolution of the environment. The images to be fed to the network in simulation are normally taken as snapshots of the simulator rendering. The use of raw images as input can be very useful in the sim2real transfer problem. Getting an observation of the state variables describing the environment is in general a difficult task in the real-world deployment, whilst mounting a camera can result to be much easier. Referring to the MDP framework, using images as input could in general break the Markov property of the environment, indeed in this case we are slightly moving from this theoretical framing of the problem, while trying to improve the actual performances. Using as state observation the actual state variables of the system guarantees the Markov property at least in the simulator. On the other hand using images as state observations it's not guaranteed mathematically to allow a complete reconstruction of the dynamics.

V. Experiments

A. Gym Hopper environment

To experiment with the chosen algorithm we used the Hopper environment by the OpenAI Gym [10]. This environment is based on the physics simulator MuJoCo. The hopper is a two-dimensional one-legged figure that consists of four body parts: torso at the top, thigh in the middle, leg in the bottom and a single foot on which the entire body rests. These masses are connected by hinges, as shown in Fig.3. The goal is to make the hopper move in the forward direction by applying the right control signals to the hinges. With reference to the MDP framework the action vector is $a_t \in R^3$, with $a_{t,i} \in [-1, 1]$. An action represents the torques applied between links expressed in $N \cdot m$. An observation is a vector $s_t \in R^{11}$ and represent physical parameters as angular velocities and different angles between the links. The reward consists of three parts:

- healthy reward: every time-step that the hopper is healthy, it gets a reward of fixed value;
- forward reward: this reward is positive if the hopper hops forward (positive x direction);
- ctrl cost: a cost for penalising the hopper if it takes actions that are too demanding for the hinges.

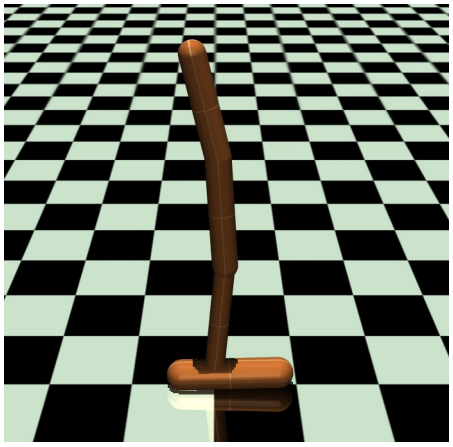

Fig. 3. Hopper environment in MuJoCo physical simulator

The total reward returned is:

$$reward = healthy\ reward + forward\ reward - ctrl\ cost \tag{9}$$

We have 2 different environments, namely source and target in which the hopper changes slightly. They have the same masses relative to thigh, leg and foot, instead torso mass is changing with a shift of $-1\ kg$ in source environment.

B. Agents with standard hyper-parameters

At first we started training the agents with the standard hyper-parameters provided by the library, following the indication by [7]. Indeed standard hyper-parameters in PPO are, according to the authors, meant to generalize as much as possible to different environments and moreover is well known the strong performance dependence of RL algorithm to slight changes of hyper-parameters. We trained two agents with 500k time-steps, respectively in the source and in the target environment. We tested both agents to appreciate the difference in performance, results are reported in table I. At first we couldn't find much difference in performance between the two agents due to the fact that we were only training for 100k time-steps. Training for longer time the difference becomes evident because the policy results to adapt much more to the peculiarities of the environment. As expected we can see from table I, training the agent in source and testing its behaviour in target lead to poor performances with respect to the source-source configuration. Practically this result means that while in the source-source configuration the agent hops many times in the forward direction, in the source-target it can only hop for 2 or 3 jumps before falling down. Hyperlink to videos relative to source-source and source-target qualitative performance.

C. Agents with different hyper-parameters

In order to understand the differences between agents with non-standard hyper-parameters we trained several other policies. Some changes in the hyper-parameters are very critical, leading to performance drops. These choices are only performed to verify the dependence of the training and test behaviours on these hyper-parameters. All the hyper-parameters are tuned independently in order to isolate as much as possible the effects. In the following a list of all the hyper-parameters changes we tried:

- entropy coefficient bonus (called c_2 in (6)) set to 0.01;
- γ, that is the discount factor representing how much the agent seek reward in the close future, set to 0.7;
- learning rate set to 0.003;
- number of steps before optimizing set to 3840;
- value function coefficient (called c_1 in (6)) set to 0.8.

As it is possible to appreciate from Fig. 4, that reports results relative to training rollouts, the average reward at the end is very similar for all the different hyper-parameters used, except for the higher learning rate and for the lower γ. In particular a very high learning rate (one order of magnitude higher than the standard one) leads to an initial increase in performances and a successive drop due to a possible too big step in the policy parameters space, causing an unrecoverable situation. A γ too small as in this case, on the other hand, is an infeasible choice, leading to an agent seeking reward not in the long run, consequently not improving so much in training. Two notable results are the ones given by the increase of the number of steps and the increase of the value function coefficient. The

TABLE I
STANDARD AGENTS RESULTS

	Average reward	Standard deviation
Source-Source	1515	6
Source-Target	1071	168
Target-Target	1604	29

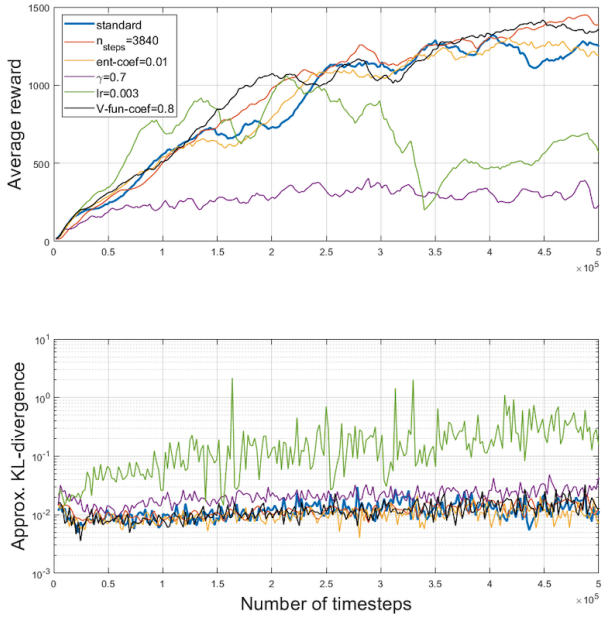

Fig. 4. Average reward and approximate KL-divergence using different hyper-parameters

first result can be theoretically justified due to the fact that in this case we are using a longer roll-out to perform the update, so the approximations (relative to gradient, value function and reward landscape) are closer to the true functions [11]. In Fig. 4 we can observe a similar behaviour in the KL-divergence of all the agents, except for the one with higher learning rate. Being the KL-divergence a measure of how much a policy changes with respect to the previous update, this trend well represent the fast change in the policy parameters space.

D. Training with UDR

Even though the concept of UDR is pretty straightforward to be applied, the main task here is to set a suitable range within the different parameter values can span. For this reason we have to take into account two opposite constraints: the noise level of the introduced randomness needs to match the precision of the parameters estimation. If the noise is too low, the randomization is pointless. On the other side, if the noise level is too high, the learning procedure will fail given that the physical model has its own dynamical behaviour that would be no more feasible in case of a huge change of its main characteristics [12]. After finding these two raw boundary conditions the problem is reduced to tune the medium-level noise measured as a percentage of the nominal values, that becomes a full-fledged hyper-parameter. In our particular case the objective is to randomize just the last three masses of the Hopper environment keeping the first one (torso) unchanged because it is responsible for the difference between the source and the target environments. Using the source-target and target-target previously obtained results, the last test should provide the mean return on the target environment trained with a randomized source in presence of UDR. As shown in Fig. 5, the investigation started from very low percentages of randomization w.r.t. the nominal values of each mass but the results were not so appreciable until 5% was reached. From this point on the randomizing effect begins acting well highlighting a significant improvement w.r.t. to the source-target environment test. With further experiments it can be seen how 7.5% could be considered a good choice as it seems to be closer to the optimal return. Increasing a bit our new hyper-parameter the performances slightly drop but they are still quite relevant. This is not the case when it gets over the 50% where it comes out that physical constraints begin to be violated reaching the worst condition at 75% when the randomized-source trained agent is not even able to match the original source-source outcome.

E. Training with NDR

The complementary case of UDR is represented by NDR. At this step fixing the nominal value of each mass as the mean value through which we randomize the masses, we have to properly tune just the standard deviation of the applied normal distribution. The chosen standard deviation will be then multiplied by the specific nominal parameter in order to make it compliant with the corresponding mean value. We just selected it by a trial and error procedure from which we got a 10% of standard deviation as shown in Fig. 6.

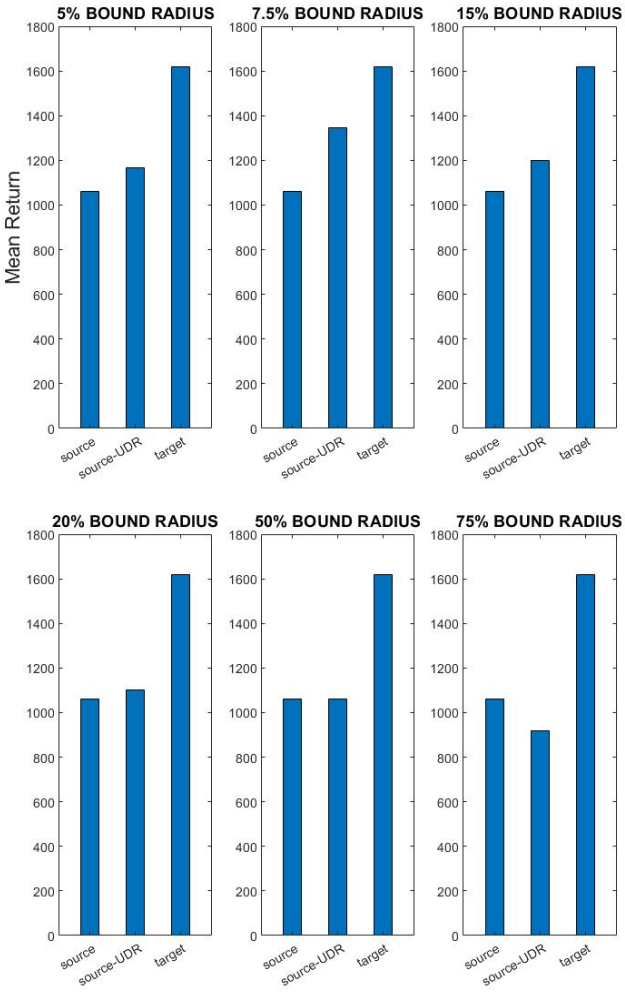

Fig. 5. Comparison of mean returns with different uncertainty intervals. On the x-axis training environments are compared in terms of mean return got from executing the corresponding policy on the target. The mean return is given by the averaged sum of three different runs for the randomized environment (source-UDR). The second and third subplots show the goodness of the expedient of the randomization, while the last one represents the worst-case condition.

Here we can see how both the source-source and source-target tests show a similar final mean return ($\simeq$ 1300), which is quite good because for sure it overcomes the simple source-target result without randomization. With respect to the results in UDR it is slightly lower but still quite good for our purposes. A different aspect that was not highlighted in the previous study is the influence of the standard deviation of the reward. In particular an higher standard deviation is relative to a less deterministic policy. For both cases the obtained values are pretty acceptable and not so high, but for the source-source case we got a clearly lower result w.r.t. the source-target one. This is reasonable from a logical point of view given that the source test environment retains the torso mass of the source training one.

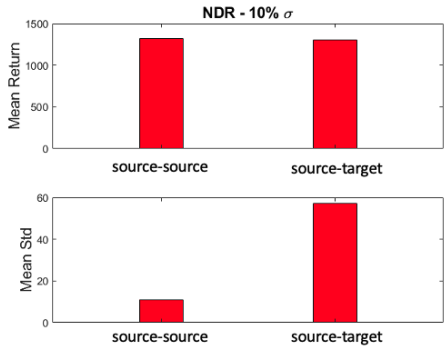

Fig. 6. Mean returns and standard deviations in both source and target test environments with a particular standard deviation adopted for normal distribution. Returns are quite similar and good, while mean standard deviations is smaller in source.

F. Training with raw images

From [6] Vision Based Reinforced Learning was used for the first time with good results in Atari Games.
In this section, we try to apply this method on Hopper environment to improve its performance. To achieve this goal it's necessary to convert the observation space into a structure compatible with one or more images and use a CNN policy in PPO algorithm to extract the features from the image. We performed the following experiments.

- *Single image observation space*

This test is performed with an observation space that is represented by an image in RGB render mode.
As shown in Fig. 7 this implementation doesn't give a higher mean reward and it is very discontinuous. This result could have been predictable because with only one image the network is not capable to grasp the evolution of the environment's dynamics. The CNN used is reported in Fig. 8.

- *Four stacked images*

Train a network with only one image may give poor results because the CNN network is not exposed to the time evolution of the environment dynamics.
In this section, we overcome this limit stacking renders from previous time-steps; indeed, this step often helps to keep information on the velocity of moving objects.
To achieve this goal, firstly we convert the images in greyscale

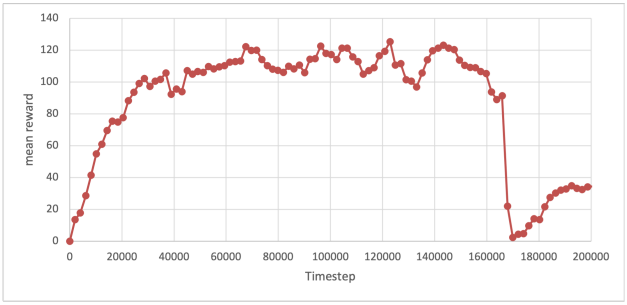

Fig. 7. Average reward with single image observation space

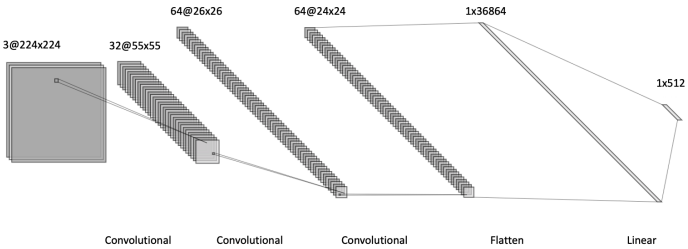

Fig. 8. CNN network employed

render to have a lower computational effort. We stack four observations of the environment state into a single tensor which will be the input of the convolutional neural network.
Fig. 9 shows the mean reward in function of the timesteps. In this plot two models trained with different Learning rates are compared. It is possible to observe that the model with higher Learning Rate (LR=0.0003) reaches an higher mean reward but, with it has a very high standard deviation: this may be a red flag, in particular it could indicate that the policy doesn't converge properly.
On the contrary, in the plot with lower Learning Rate (LR=0.0001) we can observe a more standard trend, but no conclusion can be drawn because 100k time-steps are not enough to have a reliable result.

Referring to the Hopper environment we can conclude that this method is computational heavy and at least for a limited number of time-steps (around 100k) it is unreliable.

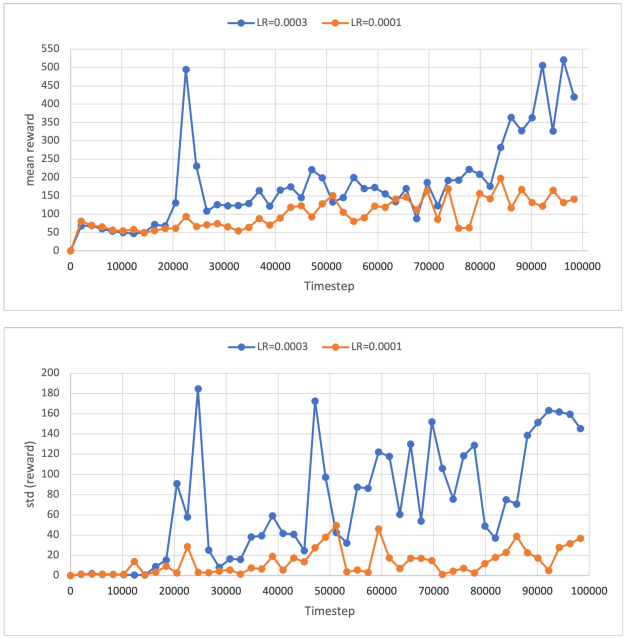

Fig. 9. Mean reward and its standard deviation with a four images stack as observation space and different Learning Rates

TABLE II
SUMMARY TABLE

	Train in source, Test in source		**Train in source, Test in target**	
	Mean Reward	***STD Reward***	***Mean Reward***	***STD Reward***
PPO standard	1515	6	1071	168
UDR 7.5% bnd	1410	52	1470	72
Gaussian 10% std	1312	11	1296	57
One image	125	27	118	33
Four images, LR=0.0003	425	175	449	179

VI. Limitations and Future Works

Normally this type of works in simulation are supported by a real implementation, with all the related problems of real hardware and real-world experiments but also the possibility to collect real data.
In particular a different approach for the Domain Randomization could be translated in an investigation with experimental results. This obviously includes the availability in terms of physical real hardware to test. However this cost will be balanced by a more reliable parameters distribution to deal with, making the overall randomization more effective.
Instead this work present all the limitations of a sim2sim experiment given that the environment is trained and tested in simulation.
Another aspect to improve is represented by the number of time-steps on which the training is carried out. In this work the computational effort that we could reach is very limited and this has a very evident effect in particular for the training with raw images. As shown in the Fig.9 the train is stopped before the mean reward reaches a plateau.
A possible future enhancement may be to create a custom policy for the PPO algorithm and test different CNN networks to compare their results and choose the best. Another possible implementation that could be interesting to test and tune is the recurrent version of PPO, using LSTM memory cells to track the dynamics' evolution of the environment.

VII. Conclusions

This work explored different methods to train the Hopper environment in MuJoCo physical simulator. We reported all the results in Table II. It's possible conclude that Domain Randomization is a very useful technique that is able to reduce the sim2sim transfer gap. In particular the obtained policy is able to generalize much better with respect to an agent with standard training. Another finding is that training with raw images is a very promising novelty but it requires a very high computational effort to let the agent reach good performances. Moreover stacking many frames to be processed at once seems to increase the performances but requires even more computational effort. All code is available at the following link.

References

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
[2] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.
[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.
[4] G. Tiboni, K. Arndt, and V. Kyrki, "Dropo: Sim-to-real transfer with offline domain randomization," 2022.
[5] J. Brosseit, B. Hahner, F. Muratore, M. Gienger, and J. Peters, "Distilled domain randomization," 2021.
[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.
[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.
[8] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015.
[9] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015.
[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[11] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A closer look at deep policy gradients," 2018.
[12] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, 2022.