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

1st Francesco Pio Russo
MS Computer Engineering Student
Polytechnic University of Turin
Turin, Italy
s292473@studenti.polito.it

2nd Ekaterina Sharova
MS Computer Engineering Student
Polytechnic University of Turin
Turin, Italy
s270231@studenti.polito.it

3rd Xileny Seijas Portocarrero
MS Computer Engineering Student
Polytechnic University of Turin
Turin, Italy
s258129@studenti.polito.it

***Abstract*—Recent developments in the robotics field have shown tremendous success for deep reinforcement learning in a number of fields. Simulated environments are used to train the various agents due to the drawbacks of collecting real-world data, such as sample inefficiency and the expense of doing so. This helps by offering a potentially limitless stream of data and allays worries about true robot safety.**

The underlying rationale for sim-to-real transfer in deep reinforcement learning is covered in this survey work, along with an overview of the primary techniques currently in use: Domain Randomization, Domain Adaptation, imitation learning, meta-learning, and knowledge distillation. We classify some of the most current works that are most pertinent and highlight the key application scenarios. Finally, we highlight the most encouraging directions and examine the main advantages and disadvantages of the various strategies.

Due to the challenges of learning in the real world and modeling physics into simulations, one of the biggest issues with the Reinforcement Learning paradigm today is its problematic applicability to robots. In this research, we investigate Domain Adaptation using Uniform Domain Randomization, one of the most sophisticated Domain Randomization techniques, to see how it addresses the drawbacks of earlier techniques and to highlight its advantages and disadvantages.

***Index Terms*—Reinforcement learning, Domain Adaptation, Uniform Domain Randomization, Trust Region Policy Optimization, custom Hopper, MuJoCo.**

I. Introduction

Making your model applicable to the actual world is one of the most challenging issues in robotics. We frequently need to train models on a simulator, which potentially delivers a limitless quantity of data, because deep RL algorithms are sample inefficient and data collecting on actual robots is expensive. When using actual robots, however, failure is frequently caused by the reality gap between the simulator and the real world. Physical parameter inconsistencies (such as friction, kp, damping, mass, and density) and, more dangerously, improper physical modeling is what cause the gap (i.e. collision between soft surfaces).

Traditionally, autonomous navigation is solved under separate problems such as state estimation, perception, planning, and control. This approach may lead to higher latency combining individual blocks and system integration issues. On the other hand, recent developments in machine learning, particularly in Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), enable an agent to learn various navigation tasks end-to-end with only a single neural network policy that generates required robot actions directly from sensory input[1]. These methods are promising for solving navigation problems faster computationally since they do not deal with the integration of subsystems that are tuned for their particular goals.

Numerous of the advanced DeepRL calculations, which have impelled recent breakthroughs, posture tall test complexities, therefore often blocking their coordinate application to physical frameworks. The use of RL algorithms in the real world also brings up several issues related to agent and environment safety.

The development of higher fidelity simulators has received a lot of attention, but it demonstrated that dynamics randomization using low fidelity simulations can also be a successful method for creating policies that can be applied immediately[5].

We must enhance the simulator and get it closer to reality in order to narrow the sim2real gap. To achieve it here can help us several strategies, such as System Identification, Domain Adaptation and Domain Randomization.

In many real application of RL (especially robotics), it is impractical to train the agent in the real world, as the real world cannot be sped up. This is especially the case for robotics, since robots are expensive and can wear out from numerous operations. Thus, sim-to-real techniques are essential methods to increase real-world sample efficiency. Among various sim-to-real methods, domain randomization is particularly attractive as it requires little to no real data.

In this work, we apply a data-driven approach and use real world data to adapt simulation randomization such that the behavior of the policies trained in simulation better matches their behavior in the real world. Therefore, starting with some initial distribution of the simulation parameters, we can perform learning in simulation and use real world rollouts of learned policies to gradually change the simulation randomization such that the learned policies transfer better to the real world without requiring the exact replication of the real world scene in simulation.

II. Related work

Since the first system identification studies, there has been an issue with establishing precise representations of the robot and its surroundings that can make it easier to develop robotic controllers in practice. Model-based RL investigated optimal policies utilizing learnt models within the context of reinforcement learning (RL)[12].

Many of the skills shown in simulation have yet to be duplicated in the real world due to the high sample complexity of RL algorithms and other physical restrictions. Additionally, researchers have looked into parallelizing robot training[4]. However, only a limited number of domains have been used to successfully show training strategies directly on physical robots.

A. Domain Adaptation

In order to adjust the data distribution in a simulation to match the real one, a mapping or regularization required by the task model is known as Domain Adaptation (DA). Many DA models are based on adversarial loss or GAN (Generative Adversarial Nets), particularly for image classification or end-to-end image-based RL tasks[1].

The difficulty of moving control rules from simulation to reality may be considered as an example of domain adaptation, in which a model developed in one domain is transferred to another[3]. One of the important assumptions behind these methodologies is that the multiple domains share comparable properties, so that representations and behaviors learnt in one will be beneficial in the other. Learning invariant characteristics has emerged as a viable method for capitalizing on these similarities.

B. Domain Randomization

The purpose of domain randomization is to provide enough simulated variability at training time such that at test time the model is able to generalize to real-world data.

We sometimes need our reinforcement learning agents to be robust to different physics than they are trained with, such as when attempting a sim2real policy transfer. Using domain randomization (DR), we repeatedly randomize the simulation dynamics during training in order to learn a good policy under a wide range of physical parameters[11].

We may build a range of simulated environments with randomized attributes using domain randomization (DR), and we can train a model that functions in each of them. Given that the real system is anticipated to be one sample in the wide range of training variations, it is likely that this model can adapt to the real-world environment. DR may just need a little quantity or no real data, as opposed to DA, which needs a fair number of real data samples to capture the distribution[5].

C. Uniform Domain Randomization

Uniform Domain Randomization uniformly samples various simulation parameters to randomize the environment.

To make the definition more general, let us call the environment that we have full access to (i.e. simulator) source domain and the environment that we would like to transfer the model to target domain (i.e. physical world). Training happens in the source domain.

During policy training, episodes are collected from source domain with randomization applied. Thus the policy is exposed to a variety of environments and learns to generalize.

In the original form of DR each randomization parameter is bounded by an interval, and each parameter is uniformly sampled within the range. The randomization parameters can control appearances of the scene, including but not limited to the followings. A model trained on simulated and randomized images is able to transfer to real non-randomized images[7].

Physical dynamics in the simulator can also be randomized. Studies have showed that a recurrent policy can adapt to different physical dynamics including the partially observable reality. A set of physical dynamics features include but are not limited to: Mass and dimensions of objects, Mass and dimensions of robot bodies, Damping, kp, friction of the joints, Gains for the PID controller (P term), Joint limit, Action delay, Observation noise[6].

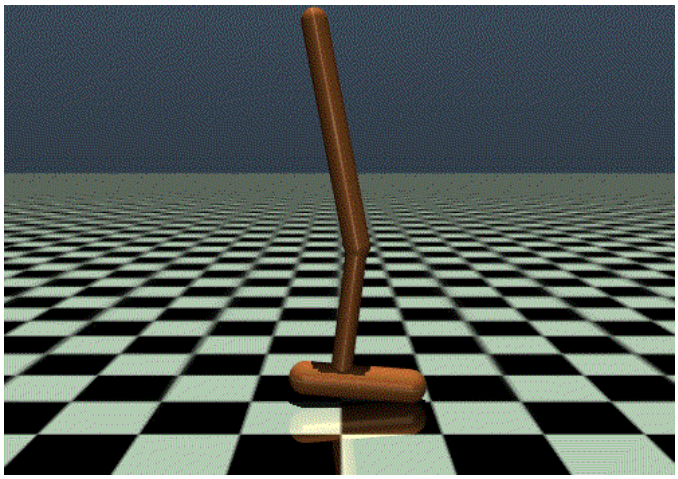

Fig. 0. Screen shot of MuJoCo Hopper.

III. Environment

Multi-Joint dynamics with Contact is referred to as MuJoCo. It is a general-purpose physics engine designed to support research and development in fields like as robotics, biomechanics, graphics and animation, machine learning, and other disciplines that call for quick and precise modeling of articulated structures interacting with their surroundings.

The built-in XML parser and compiler preallocate low-level data structures for the runtime simulation module, which is tailored to maximize speed. The native MJCF scene description language, an XML file format created to be as legible and editable by humans as feasible, is used by the user to define models. It's also possible to load URDF model files. The library features native OpenGL-rendered interactive visualization.

Model-based computations including control synthesis, state estimation, system identification, mechanism design, inverse

dynamics data analysis, and parallel sampling for machine learning applications may all be implemented using MuJoCo. It can also be used as a more conventional simulator, such as in interactive virtual worlds and games.

In this work we trained an RL agent on Hopper - a two-dimensional one-legged figure that consist of four main body parts, the functionality of it based on the method Infinite Horizon Model Predictive Control (IHMPC) for tasks with no specified goal state. This method combines offline trajectory optimization and online Model Predictive Control (MPC), generating robust controllers for complex periodic behavior in domains with unilateral constraints (e.g., contact with the environment). IHMPC can generate robust behavior in domains with discontinuities and unilateral constraints, even at the face of extreme external perturbations. IHMPC involves a limited computational load, and can be executed online on a standard laptop computer. The resulting behavior is extremely robust, allowing the Hopper to recover from virtually any perturbation.

In comparison to traditional control environments, the Hopper environment seeks to expand the number of independent state and control variables. The hopper is a one-legged, two-dimensional creature that has four primary body parts: a torso at the top, a thigh in the center, a leg at the bottom, and a single foot on which the body is supported. By adding tension to the three hinges that link the four body components, it is possible to create hops that travel forward (to the right)[12].

The objective of an Agent is to discover the most effective approach to carry out a specific activity in an Environment. The agent attempts to complete that task by acting in accordance with a strategy, such as moving left, right, or stopping (or Policy). The agent then examines the status of the environment and calculates a goodness score (or reward) depending on the action it just completed. The RL algorithm then makes use of this feedback information to update or modify the current strategy (or policy) in order to achieve greater performance in the future.

The State space and Observation spaces are identical in most simulated situations. There are two types of state/observation and action spaces (i.e. Discrete and Continuous)[12]. The agent's position is all that states and observations consist of in the most basic examples, but in more complicated and higher-dimensional examples, additional information (such as angular velocity, horizontal speed, joint positions, etc.) or complex visual information, such as an RGB matrix of observed pixel values, may be present.

Information about the action and observation spaces is essential for evaluating RL algorithms because many of them were created exclusively for a certain kind of state, observation, and action space. For instance, the PPO algorithm can be employed in environments with either discrete or continuous action spaces, in contrast to the DDPG method, which can only be used to environments with continuous action spaces[12].

IV. Method

A. Soft Actor Critic (SAC)

The Soft Actor Critic (SAC) technique bridges the gap between stochastic policy optimization and DDPG-style methods by optimizing a stochastic policy in an off-policy manner. Although it isn't a direct replacement for TD3, it integrates the clipped double-Q method and gains from target policy smoothing because of the policy's inherent stochasticity in SAC[5].

Entropy regularization is one of SAC's key characteristics. Entropy, a gauge of the policy's randomness, and expected return are trade-offs that the policy is trained to maximize. This is closely related to the exploration-exploitation trade-off: when entropy rises, exploration increases, which speeds up learning later. Additionally, it can stop the strategy from prematurely achieving a suboptimal local optimum[2].

We use state-independent parameter vectors to describe the log standard deviations in VPG, TRPO, and PPO. The log standard deviations are outputs from the neural network in SAC, and as a result, they depend on state in a sophisticated manner. In our experience, SAC with state-independent log std developers did not function.

The SAC objective has a number of advantages. First, the policy is incentivized to explore more widely, while giving up on clearly unpromising avenues. Second, the policy can capture multiple modes of near-optimal behavior. In problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions.

B. Proximal Policy Optimization (PPO)

The Proximal Policy Optimization (PPO), which is much easier to build and tweak and performs on par with or better than state-of-the-art methods, this algorithm uses a policy gradient approach for reinforcement learning. The goal was to create an algorithm that, although utilizing solely first-order optimization, had the data efficiency and dependable performance of TRPO.

When compared to its actual impact, this statement is actually pretty little. The natural policy gradient solves the convergence issue that plagues policy gradient methods. Natural policy gradient is not scalable for large-scale situations in practice since it requires a second-order derivative matrix. For practical purposes, the computing complexity is too great. Research is conducted extensively to approach the second-order method and reduce complexity[8]. PPO follows a somewhat different methodology. It formalizes the constraint as a penalty in the objective function rather than imposing a rigid constraint.

To keep new policies close to existing ones, the PPO of first-order approaches employs a few additional strategies. PPO techniques are far easier to apply and, according to empirical evidence, appear to perform at least as well as TRPO[8,6].

Similar to TRPO, PPO-Penalty approximates a KL-constrained update by penalizing the KL-divergence in the objective function rather than making it a hard constraint.

It also automatically scales the penalty coefficient throughout training.

The objective of PPO-Clip does not contain a KL-divergence term, and there is no constraint at all. It relies on specific clipping in the goal function to eliminate incentives for the new policy to diverge much from the previous one.

The main objective of reinforcement learning is to identify the best possible behavior that maximizes rewards. Essential RL approaches known as policy gradient methods use an estimator of the gradient of the predicted cost to directly optimize the parameterized policy. We should steer clear of parameter modifications that drastically alter the policy in a single step in policy gradient approaches. By limiting our stride length to fall within a ”trust region,” TRPO can increase training stability. TRPO’s implementation is challenging since it requires a KL divergence limit on the amount of the update for each iteration. As a result, PPO, which greatly simplifies TRPO while simulating it, was introduced. Instead of employing a KL divergence constraint, PPO employs a penalty based on the fact that a particular surrogate objective forms a pessimistic bound on the performance of the policy.

C. Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization is a policy gradient method in reinforcement learning that avoids parameter updates that change the policy too much with a KL divergence constraint on the size of the policy update at each iteration. TRPO couples insights from reinforcement learning and optimization theory to develop an algorithm which, under certain assumptions, provides guarantees for monotonic improvement. It is now commonly used as a strong baseline when developing new algorithms[6].

Policy gradient methods (e.g. TRPO) are a class of algorithms that allow us to directly optimize the parameters of a policy by gradient descent. In this section, we formalize the notion of Markov Decision Processes (MDP), action and state spaces, and on-policy vs off-policy approaches. This leads to the REINFORCE algorithm, the simplest instantiation of the policy gradient method.

V. Experiments

Experiments were carried on using simulations provided by the Hopper environment, one of the environments of the MuJoCo physics engine with a simulation timestep of 0.002s. Furthermore, two custom variants were created ad-hoc and to simulate the reality gap, the source domain Hopper, where the policy training occurs, has a torso mass shifted by 1kg with respect to the target domain, that technically represents the real world. Each episode consists of at most 500 control timesteps.

In the Hopper environment, the state space is described by the observation space, which is continuous. Observations are made of positional values of different body parts of the hopper and of their corresponding velocities. The action space is continuous, and an action represents the torques applied between links. In the source environment, the masses of the torso, thigh, leg and foot are respectively 2.53kg, 3.93kg, 2.71kg and 5.09kg. While in the target environment the only variation is in the mass of the torso, which is 1kg heavier.

As a first approach we trained different models using PPO, SAC and TRPO, state-of-the-art reinforcement learning algorithms and evaluated how well they performed on the source environment. Based on the results obtained, we decided to proceed with the training of two agents that used PPO and TRPO algorithms respectively. The agent using PPO was trained with 0.0003 learning rate, while the agent using TRPO was trained with two different learning rates such as 0.001 and 0.0003. The two types of agents were trained using a variation of timesteps, ranging from 50000 to 500000. The agents were evaluated over 50 episodes in simulation and results are shown in Fig.1, Fig.2, Fig.3.

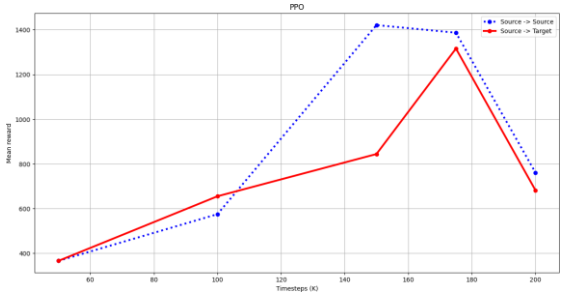

Fig. 2. Mean reward of different agents using PPO algorithm with learning rate 0.0003 trained over different timesteps in the source environment and evaluated in the source and target environments.

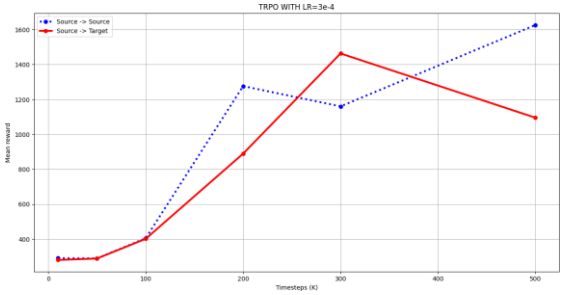

Fig. 3. Mean reward of different agents using TRPO algorithm with learning rate 0.0003 trained over different timesteps in the source environment and evaluated in the source and target environments.

TABLE I
BEST MEAN REWARD OF PPO AND TRPO AGENTS

Agent	Source to Source	Source to Target	Target to Target
PPO (lr=0.0003, ts=175K)	1387.40 +/- 5.54	1316.47 +/- 10.34	715.40 +/- 37.81
TRPO (lr=0.001, ts=200K)	1519.37 +/- 11.66	1440.56 +/- 185.18	**1465.65 +/- 7.18**
TRPO (lr=0.0003, ts= 300K)	1160.93 +/- 243.49	**1463.48 +/- 38.29**	769.00 +/- 210.86

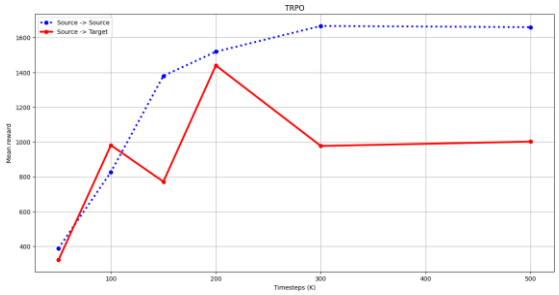

Fig. 4. Mean reward of different agents using TRPO algorithm with learning rate 0.001 trained over different timesteps in the source environment and evaluated in the source and target environments.

We proceed to implement Uniform Domain Randomization to close the gap between source and target domains and make the agent more robust, by using a uniform distribution over the masses of the thigh, leg and foot of the hopper in the source environment. In this way, the agent is forced to maximize its reward training with sampled values that vary at each episode. In Fig.4, we can see the effects of different percentages of mass variation applied to train the agent using the PPO algorithm.

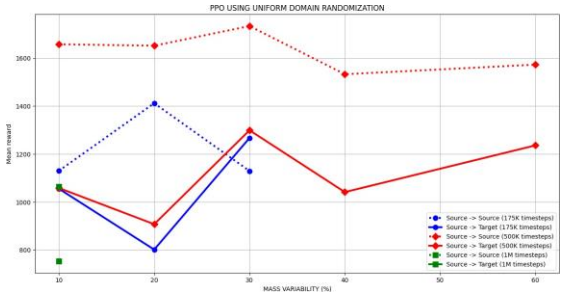

Fig. 5. Mean reward of different agents using PPO algorithm with learning rate 0.0003 and increasing timesteps trained with different mass variability percentages in the source environment and evaluated in the source and target environments.

As we can see, even increasing the timesteps to train the model, with a low mass variability, the agent is not able to achieve good results when evaluated. The same occurs in the case of TRPO, as shown in Fig.5, Fig.6.

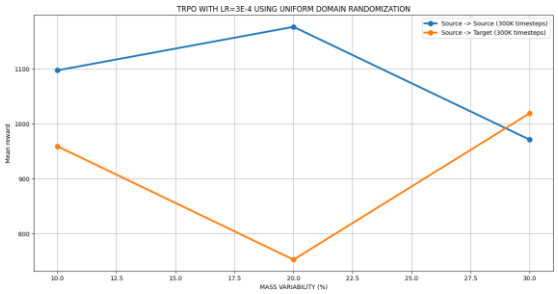

Fig. 6. Mean reward of an agent using TRPO algorithm with learning rate 0.0003 and 300K timesteps trained with different mass variability percentages in the source environment and evaluated in the source and target environments.

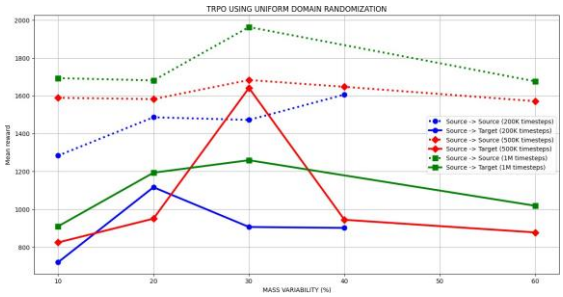

Fig. 7. Mean reward of different agents using TRPO algorithm with learning rate 0.001 and increasing timesteps trained with different mass variability percentages in the source environment and evaluated in the source and target environments.

TABLE II
BEST MEAN REWARD OF PPO AND TRPO AGENTS USING UNIFORM DOMAIN RANDOMIZATION

Agent	Mass Variability	Source to Source	Source to Target
PPO (lr=0.0003, ts=500K)	30%	1734.12 +/- 7.45	1299.25 +/- 79.71
TRPO (lr=0.0003, ts= 300K)	30%	971.38 +/- 35.90	1019.81 +/- 62.84
TRPO (lr=0.001, tr=500K)	30%	1683.84 +/- 10.20	**1641.62 +/- 181.69**

As shown in the results, Uniform Domain Randomization is able to overcome the shift in the mass of the torso of the hopper.

Finally, we attempt to train a control policy able to receive images as observations, allowing the robot to interact with the environment from those images. We implemented a custom convolutional neural network as the underlying policy structure.

Following the approach described in [7], we implemented some of the pre-processing methodologies to our task. The Hopper frames were 500 x 500 pixel images with a 128 colour palette. To reduce the computational cost of processing such images, we converted the images to grey-scale and then resize them to 100 x 100 pixel images. We also choose to stack 4 frames together to get our final input for the model. In Fig.7, transformations applied to the initial frames can be seen.

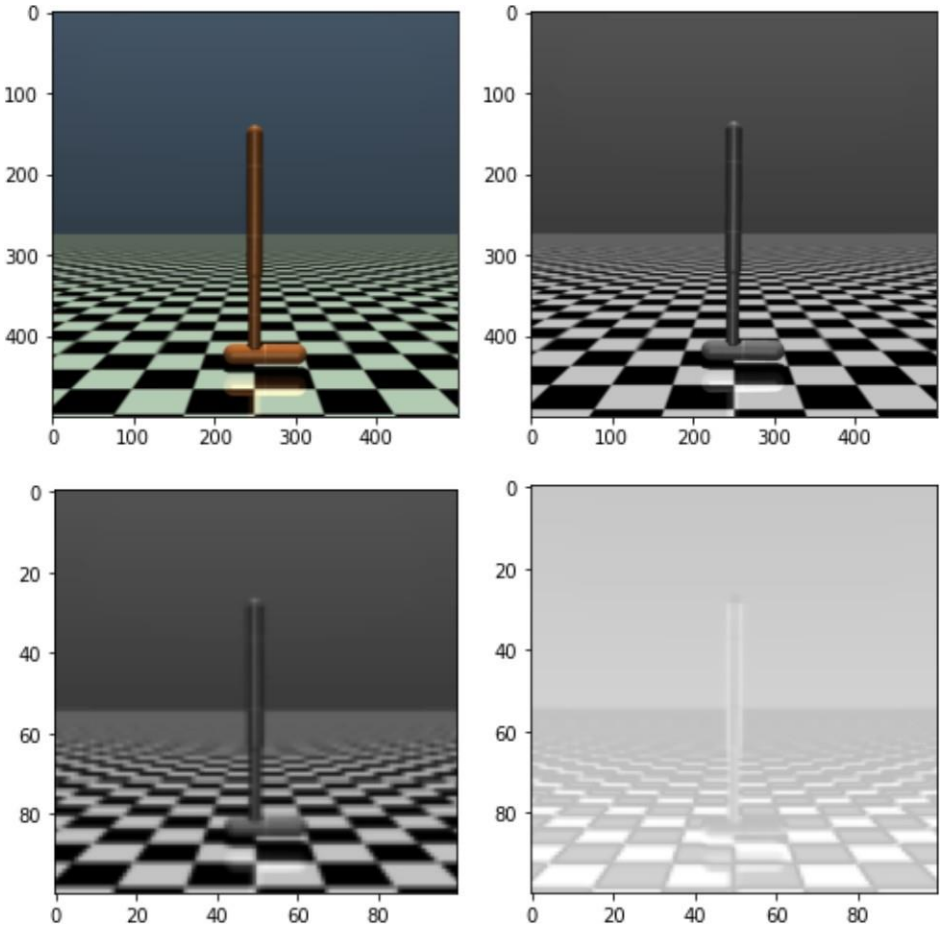

Fig. 8. Pixel observations of the custom Hopper. From left to right, the first image is the pixel transformation of the observation, the second one shows the grey-scale result, the third one is the result of resizing the image to 100 x 100 pixels and the last one shows 4 staked frames of our tranformed pixed observations.

So, the final input to the neural network consists in an 100 x 100 x 4 image. The first hidden layer convolves 32 8 x 8 filters with stride 4 with the input image and applies an hyperbolic tangent. The second hidden layer convolves 64 4 x 4 filters with stride 2, again followed by a hyperbolic tangent. The final hidden layer is used to flatten the input. The output layer is a linear layer followed consists of 256 rectifier units with a single output for each valid action. In these final experiments, we used the TRPO algorithm with minibatches of size 32. In Fig.8, the mean reward of the trained agents using 30% mass variability and different timesteps and learning rates are shown.

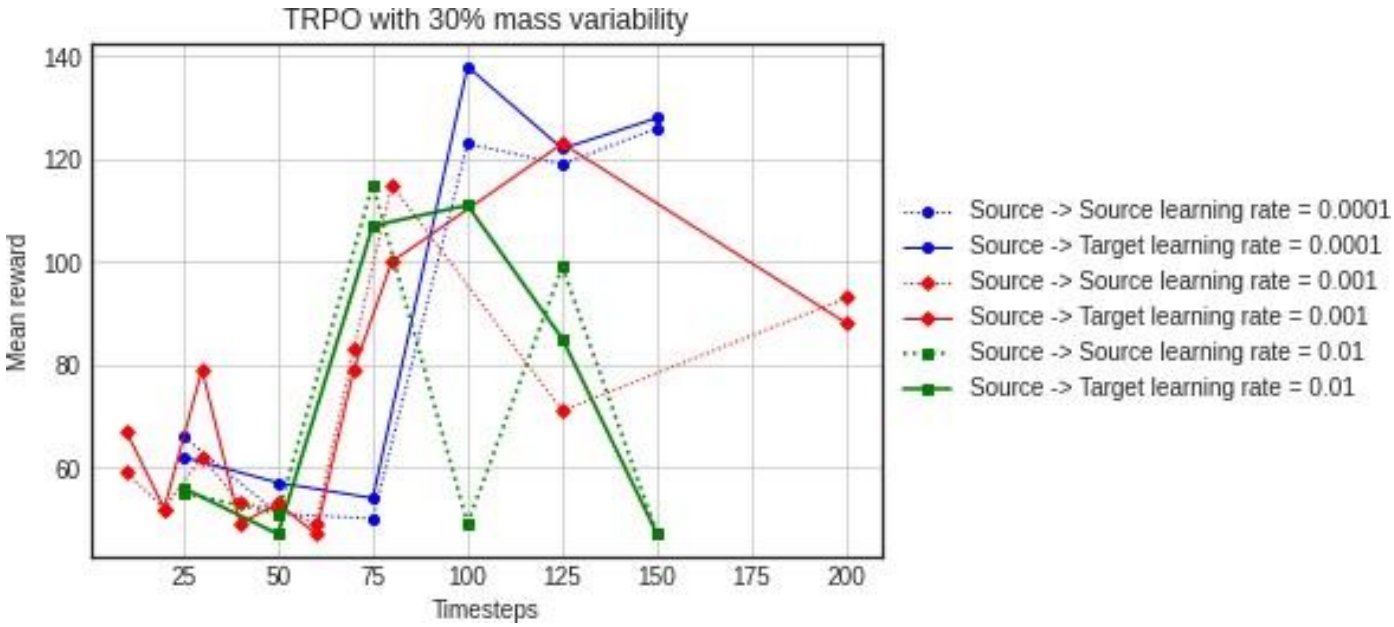

Fig. 9. Mean reward of different agents using TRPO algorithm with different learning rates and 30 percent mass variability trained over different timesteps in the source environment and evaluated in the source and target environments.

TABLE III
BEST MEAN REWARD OF TRPO AGENTS USING UNIFORM DOMAIN RANDOMIZATION TRAINED WITH PIXEL OBSERVATIONS

Agent	Source to Source	Source to Target
TRPO (lr=0.0001, ts=100K)	**122.81 +/- 1.27**	**138.08 +/- 4.50**
TRPO (lr=0.001, ts=125K)	71.48 +/- 38.44	122.79 +/- 23.77
TRPO (lr=0.01, ts=100K)	49.42 +/- 0.97	111.29 +/- 4.55

VI. CONCLUSIONS

In this paper, we described the approaches implemented to achieve Domain Adaption using Uniform Domain Randomization and a state-of-the-art reinforcement learning algorithm such as TRPO. We trained control polices for a custom Hopper in two domains. We were able to show the behaviour of an agent trained with different reinforcement learning algorithms and the effects of applying a uniform distribution to the masses. Finally, we attempt to train a visual-based control policy from raw input images. We intend to improve the results of this task by implementing a pre trained model as underlying policy structure and refine our image pre processing methodologies. Furthermore, since working with images can be computationally challenging, we believe that by increasing the timesteps during the training phase we can achieve higher rewards.

REFERENCES

[1] Richard S. Sutton, Andrew G. Barto "Reinforcement Learning: An introduction (Second Edition)".

[2] Kober, J., Bagnell, J. A., Peters, J. (2013). "Reinforcement learning in robotics: A survey". The International Journal of Robotics Research.

[3] Kormushev, P., Calinon, S., Caldwell, D. G. (2013). "Reinforcement learning in robotics: Applications and real-world challenges".

[4] Peng, X. B., Andrychowicz, M., Zaremba, W., Abbeel, P. (2018, May). "Sim-to-real transfer of robotic control with dynamics randomization"

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

[6] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P. (2015, June). "Trust region policy optimization".

[7] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning." arXiv, Dec. 19, 2013.

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). "Proximal policy optimization algorithms".

[9] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., Fox, D. (2019, May). "Closing the sim-to-real loop: Adapting simulation randomization with real world experience".

[10] Leemon Baird. Residual algorithms: Reinforcement learning with func- tion approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

[11] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: arXiv preprint arXiv:1604.06778 (2016).

[12] Tiboni, G., Arndt, K., Kyrki, V. (2022). "DROPO: Sim-to-Real Transferwith Offline Domain Randomization".