Domain Randomization Techniques for Reinforcement Learning: Bridging the Reality Gap

Andrea Delli
Politecnico di Torino
s331998
s331998@studenti.polito.it

Giorgia Modi
Politecnico di Torino
s330519
s330519@studenti.polito.it

Ivan Necerini
Politecnico di Torino
s345147
s345147@studenti.polito.it

***Abstract*—Reinforcement learning (RL) has shown great promise in addressing complex control tasks; however, transferring policies from simulation to real-world applications remains challenging due to the sim-to-real gap. This study investigates the effectiveness of domain randomization techniques to enhance policy robustness and generalization across diverse environments. Focusing on Hopper and Walker2d environments, we explore Uniform Domain Randomization (UDR) and Truncated Normal Domain Randomization (TNR) to evaluate their impact on sim-to-real transfer.**

Our experiments reveal that UDR consistently outperforms TNR, achieving higher average rewards and exhibiting superior generalization to unseen dynamics. Notably, single mass randomization experiments underscore the sensitivity of specific body segments, such as leg masses, to overall performance, suggesting that more targeted randomization approaches could further improve policy effectiveness. Additionally, increasing the domain gap demonstrated UDR's ability to maintain, and in some cases exceed, the performance of policies trained directly in target environments, as observed in the Walker2d scenario.

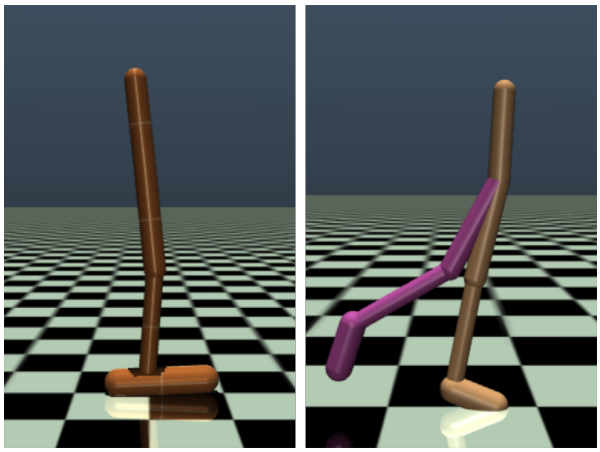

Fig. 1. OpenAI Gym's Hopper (left) and Walker2d (right) environments

I. Introduction

Reinforcement learning (RL) has emerged as a powerful framework for solving complex control and decision-making tasks. However, the gap between simulation-based training and real-world deployment, commonly referred to as the "reality gap" (or "sim-to-real gap"), poses significant challenges. Simulated environments, despite their efficiency and safety for training, often fail to capture the full range of dynamics present in the physical world, leading to suboptimal performance during real-world deployment.

Domain randomization has been proposed as an effective strategy to address this gap. By systematically varying environmental dynamics during training, this technique exposes RL agents to a broader set of scenarios, enhancing their robustness and generalization capabilities. This study focuses on evaluating the impact of domain randomization techniques in mitigating the sim-to-real gap through a controlled sim-to-sim transfer setup. Two reinforcement learning environments, Hopper and Walker2D, are used as testbeds, representing challenges of varying complexity.

In this work, we systematically explore the effects of Uniform Domain Randomization (UDR) and Truncated Normal Domain Randomization (TNR) on policy robustness and generalization.

Key contributions include:

1) A comparative analysis of UDR and TNR across multiple scenarios, including single-mass and multi-mass randomization setups.
2) Insights into the sensitivity of specific dynamics parameters, such as leg and foot masses, on the effectiveness of domain randomization.
3) The evaluation of increasing source-to-target domain gaps, providing a rigorous test of policy robustness.

Our findings highlight the central role of carefully designed domain randomization strategies in reducing the sim-to-real gap, offering practical insights to improve RL-based approaches in robotics.

II. Related Works

Reinforcement Learning (RL) has emerged as a powerful framework for teaching agents to make decisions in complex environments. The foundational work by Sutton and Barto [1] provides a comprehensive introduction to RL, covering its theoretical underpinnings and practical applications. While RL has been successfully applied in many domains, its extension to robotics presents unique challenges [2]. Robots operating in real-world environments must contend with high-dimensional

state spaces, noisy sensor inputs, and dynamically changing conditions [3].

One significant challenge in deploying RL policies learned in simulation to real-world robots is the so-called *sim-to-real gap* (or *reality gap*) [4]. This discrepancy arises because simulation environments, while useful for fast and safe training, cannot perfectly replicate the complexities of real-world physics and dynamics. To address this, researchers have explored the *sim-to-real transfer* paradigm, which aims to bridge the gap between simulated and real-world environments [5].

A widely adopted strategy for sim-to-real transfer is *domain randomization*, as introduced by Tobin et al. [6]. Domain randomization involves training RL agents on a diverse set of randomized environments, forcing the policies to generalize across a range of dynamics parameters such as masses, friction coefficients, and visual textures. This approach has been shown to produce robust policies that can effectively transfer to target domains without requiring extensive fine-tuning.

Building on these concepts, Peng et al. [5] demonstrated the effectiveness of dynamics randomization for robotic control tasks, achieving successful transfer of policies trained in simulation to real-world robots. Other works have extended this idea by combining domain randomization with advanced RL algorithms like Proximal Policy Optimization (PPO) [7] and Soft Actor-Critic (SAC) [8], which provide stable and efficient training frameworks.

In this project, the sim-to-real transfer challenge is explored through a simplified *sim-to-sim transfer* scenario. Specifically, a controlled discrepancy is manually introduced between the source and target domains to simulate the reality gap. This approach allows for a systematic evaluation of how domain randomization can mitigate the negative effects of such discrepancies. By focusing on dynamics parameters such as the link masses in the Hopper environment, the project implements different Domain Randomization techniques to enhance the robustness of learned policies.

The emphasis on sim-to-sim transfer with manually injected discrepancies highlights an important aspect of the broader sim-to-real paradigm. It allows researchers to precisely quantify the performance degradation caused by domain shifts and evaluate the effectiveness of domain randomization techniques.

Overall, Domain randomization is a key strategy for addressing sim-to-real transfer challenges. This project explores how controlled variations in domain parameters enhance the robustness of learned policies, improving reinforcement learning applications in robotics.

III. Methods

This project focuses on implementing a reinforcement learning (RL) pipeline to train control policies for robotic environments using state-of-the-art algorithms, specifically Proximal Policy Optimization (PPO). The environments considered include the Hopper and Walker2d from OpenAI Gym [9], both simulated using the MuJoCo physics engine [10]. These experiments simulate the sim-to-real transfer problem through a sim-to-sim transfer setup, where discrepancies between source and target domains are introduced manually. Below, we detail the methods and tools employed.

A. Reinforcement Learning Pipeline

To train the RL agent, we used the third-party library `stable-baselines3` [11], which provides an easy-to-use implementation of modern RL algorithms. The PPO algorithm was selected for this project due to its robustness and efficiency in optimizing control policies.

Proximal Policy Optimization (PPO) is a policy-gradient method designed to achieve reliable training by avoiding large updates to the policy. The algorithm introduces a clipped surrogate objective to restrict the step size during updates, ensuring that the new policy remains close to the old one. PPO is particularly suitable for continuous control tasks such as the Hopper environment or the Walker2d environment.

B. Hopper and Walker2d Environments

The Hopper and Walker2d environments, both provided by OpenAI Gym, model robots tasked with learning locomotion skills through reinforcement learning. The Hopper environment[1] simulates a one-legged robot with the goal of hopping forward without falling, maximizing horizontal speed. The Walker2d environment[2] simulates a bipedal robot learning to walk efficiently. Both environments leverage the MuJoCo physics engine to simulate realistic dynamics. The environment offers a Python interface, enabling the manipulation of physical parameters and access to observation and action spaces. Below, we summarize the main characteristics and modifications made for this task.

The Hopper environment has an 11-dimensional continuous state space, which includes the vertical position of the torso, joint angles of the thigh, leg, and foot, and the torso's angular orientation, along with the horizontal and vertical velocities of the torso and angular velocities of the joints. The action space is a 3-dimensional continuous vector representing the torques applied to the thigh, leg, and foot joints, each bounded between -1 and 1. To simulate the sim-to-real transfer problem, two custom domains were created: a source domain with a reduced torso mass of 1 kg and a target domain representing the "real-world" scenario with the default Hopper model. The reward function in the Hopper environment consists of a forward reward for horizontal motion, a control cost penalizing large torque values, and a healthy reward for maintaining a valid configuration. Episodes terminate if the hopper falls or deviates from a healthy state, as defined by constraints on its height, angles, and velocities.

In the Walker2d environment, the state space is 17-dimensional, consisting of the z-coordinate of the torso (representing its height), its angular orientation, joint angles for both legs, thighs, feet, and velocities corresponding to these position variables. The action space is an 6-dimensional continuous vector, representing the torques applied to each joint. Similar

[1]https://www.gymlibrary.dev/environments/mujoco/hopper/

[2]https://www.gymlibrary.dev/environments/mujoco/walker2d/

to the Hopper environment, a custom domain strategy is used to evaluate sim-to-sim transfer, with the source domain having a reduced torso mass by 1 kg, and the target domain using the default Walker2d model. The reward function in the Walker2d environment encourages forward progress in the positive x-direction, penalizes excessive torque usage, and provides a healthy reward for keeping the robot upright within valid configuration bounds. Episodes terminate when the robot falls, when its torso's angular orientation exceeds 1 radian, or when the episode reaches a maximum time limit of 1,000 steps.

C. Domain Randomization

To enhance the generalization capability of the trained policy, domain randomization was implemented. This involved randomizing the dynamics parameters of the Hopper environment during training, such as masses and friction coefficients.

We applied two types of domain randomization methods to the masses of the *source* environments: uniform randomization and truncated normal randomization, in order to help the policy generalize to unseen conditions by simulating variability in the environment's physical parameters.

a) Uniform Randomization: The mass m_i of each body segment, excluding the torso, was sampled from a uniform distribution, where all values within a specified range are equally likely:

$$m_i \sim \mathcal{U}(m_i^{\text{original}} - 1, m_i^{\text{original}} + 1). \tag{1}$$

b) Truncated Normal Randomization: The masses were also sampled from a truncated normal distribution, which is derived from a normal distribution where the values are confined to a specific range. The masses are sampled from this distribution as follows:

$$m_i \sim \text{TruncNorm}(m_i^{\text{original}}, 1, m_i^{\text{original}} - 1, m_i^{\text{original}} + 1), \tag{2}$$

where m_i^{original} is used as the mean of the distribution, 1 is the variance, and the range $m_i^{\text{original}} \pm 1$ confines the sampled values to within 1 unit of the original mass. This ensures that most sampled values are close to the original mass, while still allowing for some variability.

IV. Experiments

In this project, we conducted several experiments to evaluate the impact of different design choices on policy performance. This section presents the results of these experiments, starting with initial tests performed on the Hopper environment. These experiments serve as a foundation for later comparisons with experiments conducted on the Walker2d environment.

A. Learning Rate Schedules

In this experiment, we tested three types of learning rate (LR) schedules: constant, linear, and exponential. These schedules control how the learning rate evolves during training, potentially impacting the policy's performance. Each schedule has an initial learning rate value of $v_{\text{initial}} = 0.0003$.

a) Constant Schedule: The learning rate remains fixed at a value of v_{initial} throughout training.

b) Linear Schedule: In the linear schedule, the learning rate decreases linearly from its initial value to zero over the course of training:

$$\text{lr}(p) = p \cdot v_{\text{initial}}, \tag{3}$$

where p represents the progress remaining, a value that decreases linearly from 1 to 0 during training.

c) Exponential Schedule: The exponential schedule decreases the learning rate exponentially as training progresses:

$$\text{lr}(p) = \text{lr}_{\text{initial}} \cdot p^r, \tag{4}$$

where p is the progress remaining (decreasing from 1 to 0 during training) and r is the decay rate, fixed to 0.1 in our case.

In Figure 2, the learning rate schedules—constant, linear, and exponential—are illustrated.

We tested all three schedules on the source Hopper environment with 2,000,000 timesteps for each training. The results, reported in Table I, show that the exponential learning rate schedule outperformed the others. This can be attributed to its gradual reduction of the learning rate, which balances exploration and exploitation effectively. Early in training, a higher learning rate encourages exploration, while the slower decay as p approaches 0 allows for fine-tuning the policy in later stages.
This experiment result led us to apply the exponential learning rate scheduler for all subsequent trainings.

TABLE I
Comparison of Learning Rate Schedules for Hopper Models

LR Schedule	Test Reward (avg $\pm$ std)
Constant	1202.06 $\pm$ 324.22
Linear	1194.65 $\pm$ 237.26
Exponential	1263.31 $\pm$ 215.91

B. Grid Search for Hyperparameters Tuning

To optimize the performance of our reinforcement learning model, we conducted a grid search over the following hyperparameters:

- `n_epochs` $\in \{5, 10, 20\}$
- `clip_range` $\in \{0.1, 0.2, 0.3\}$
- `gae_lambda` $\in \{0.9, 0.95, 0.99\}$
- `gamma` $\in \{0.95, 0.99, 0.999\}$
- `batch_size` $\in \{32, 64, 128\}$

For each combination, we trained the model on the source domain for 100,000 timesteps and evaluated the mean reward on 100 testing episodes.

The results revealed significant variability in performance across parameter configurations. The best combination, which achieved a mean reward of 1461.24, was `n_epochs` = 20, `clip_range` = 0.3, `gae_lambda` = 0.99, `gamma` = 0.999, and `batch_size` = 128.

We subsequently trained this best model for 2 million timesteps to confirm its effectiveness. However, for comparison, we also trained a model using the standard PPO

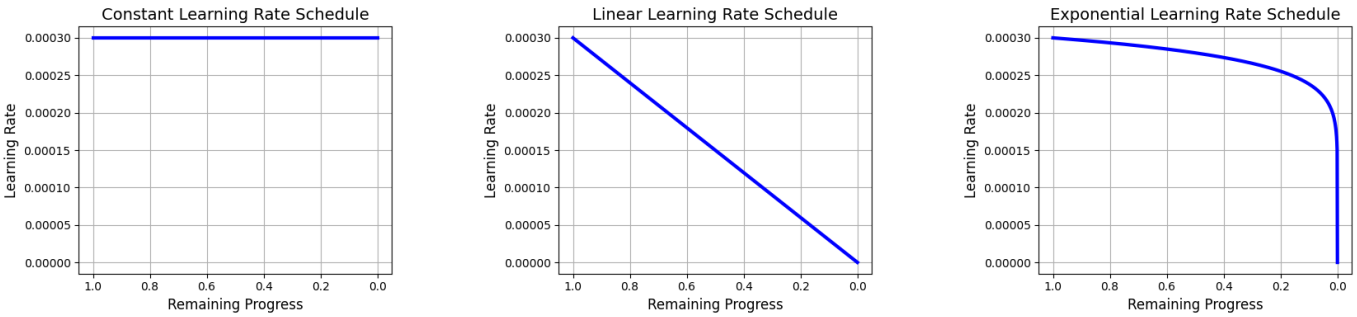

Fig. 2. Comparison between different Learning rate schedules - constant, linear, and exponential.

hyperparameters: `n_steps` = 2048, `batch_size` = 64, `n_epochs` = 10, `gamma` = 0.99, `gae_lambda` = 0.95, `clip_range` = 0.2, and `clip_range_vf` = 1.

Surprisingly, despite the exhaustive hyperparameter grid search, the PPO baseline model, which utilized the standard hyperparameters, performed better in terms of overall reward. This is probably due to higher stochasticity during training caused by the relatively low number of training timesteps used during the grid search experiments. This result led us to the decision to adopt the standard PPO hyperparameters for the remainder of our study.

C. Baseline Models Evaluation on Hopper Environment

To establish a baseline for our study, we trained two models (one on source domain and one on target domain) using the default PPO hyperparameters, as previously described, with an exponential learning rate schedule. All models were trained for 2 million training timesteps and evaluated on both the source and target domains. Figures 3 and 4 illustrate the learning curves of the source and target models, respectively.

- **Source** $\rightarrow$ **Source**: A model trained and tested on the source domain.
- **Source** $\rightarrow$ **Target**: A model trained on the source domain and tested on the target domain.
- **Target** $\rightarrow$ **Target**: A model trained and tested on the target domain.

The evaluation results for these models are presented in Table II. Each test was conducted over 1000 episodes, and the mean reward with standard deviation is reported.

TABLE II
BASELINE PERFORMANCE OF HOPPER MODELS ACROSS SOURCE AND TARGET DOMAINS

Model Setup	**Test Reward (avg $\pm$ std)**
Source $\rightarrow$ Source	1471.47 $\pm$ 311.72
Source $\rightarrow$ Target	1495.05 $\pm$ 253.62
Target $\rightarrow$ Target	1696.48 $\pm$ 93.64

The Target $\rightarrow$ Target model achieves the highest rewards with low variance, indicating strong consistency within the source domain. Similarly, the Source $\rightarrow$ Source model performs well, as expected, since it is trained and tested within the same environment. Surprisingly, the Source $\rightarrow$ Target model performs comparably to the Source $\rightarrow$ Source model, despite the domain gap. This unexpected robustness suggests that the policy trained in the source domain was able to generalize well to the target domain. A possible explanation lies in the simpler dynamics of the source environment; for example, the source domain's configuration, featuring a mass of less than 1 kg, may have resulted in a smoother optimization landscape during training. These conditions appear to have helped the source-trained policy generalize effectively, even to a domain with differing dynamics.

Typically, a domain gap leads to performance degradation, because policies trained in the source domain adapt to its specific physical parameters, such as mass or friction, and may struggle with unseen dynamics in the target domain. However, in this case, the inherent robustness of the source-trained policy appears to have mitigated these challenges. While training directly on the target environment might seem like an ideal solution for maximizing performance, this approach is often impractical in real-world scenarios due to high costs, safety concerns, and the time-intensive nature of hardware training. Simulation, on the other hand, enables faster and safer experimentation while avoiding wear and tear on physical systems. This underscores the importance of sim-to-real research, where the goal is to train policies in simulation that generalize effectively to real-world systems. These baseline results establish a foundation for evaluating advanced techniques, such as domain randomization, in subsequent sections.

D. Domain Randomization on Hopper environment

Despite the promising performance observed in the Source $\rightarrow$ Target configuration of the baseline models, we explored the application of domain randomization to evaluate its impact on policy robustness. How previously discussed in section III-C, two strategies were tested: Uniform Domain Randomization (UDR) and Truncated Normal Domain Randomization (TNR). In both cases, randomization was applied to the link masses of the Hopper, with the torso mass excluded from randomization to maintain the sim-to-real gap. For each episode, the randomized masses were sampled either from uniform distributions (UDR) or truncated normal distributions (TNR), and training was conducted on the source domain using the same PPO algorithm and hyperparameters as in the baseline experiments.

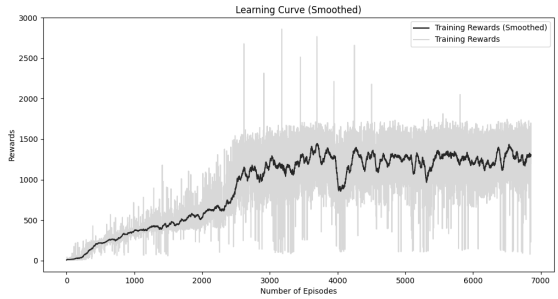

Fig. 3. Learning curve on the source domain.

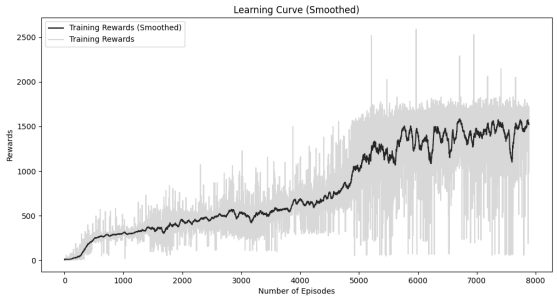

Fig. 4. Learning curve on the target domain.

The results of these experiments, summarized in Table III, reveal notable differences between the two strategies. UDR led to a significant improvement in performance for both Source $\rightarrow$ Source and Source $\rightarrow$ Target configurations, with rewards closely aligned between the two domains. This indicates that the policy trained with UDR is robust to variations in dynamics, successfully generalizing to the target domain despite the domain gap. Interestingly, UDR even outperformed the Target $\rightarrow$ Target baseline reported in Table II, further highlighting its effectiveness. Moreover, UDR significantly reduced the variance, dropping from approximately 300 (baseline models) to less than 10, making the model notably more stable.

In contrast, TNR produced inferior results, with lower rewards and higher variance compared to UDR in both configurations. In addiction, the performance gap between Source $\rightarrow$ Source and Source $\rightarrow$ Target increases. This suggests that TNR may introduce instability during training, potentially due to the properties of the truncated normal distribution, which can lead to inconsistent dynamics encountered across episodes.

TABLE III
COMPARISON OF UNIFORM AND GAUSSIAN DOMAIN RANDOMIZATION IN THE HOPPER ENVIRONMENT

Test reward (avg ± std)	**UDR**	**TDR**
Source $\rightarrow$ Source	1719.87 $\pm$ 8.32	1286.19 $\pm$ 366.47
Source $\rightarrow$ Target	1721.65 $\pm$ 9.58	1351.57 $\pm$ 133.70

These findings demonstrate the benefits of UDR in improving policy robustness by encouraging the agent to solve the task for a range of multiple environments at the same time, such that its learned behavior may be robust to dynamics variations. However, the effectiveness of domain randomization relies on careful design of the randomization ranges. Poorly chosen ranges could either fail to capture meaningful variations or introduce excessive noise, potentially hindering performance. While UDR showed strong generalization in this controlled setting, applying it effectively to real-world scenarios would require ensuring that the simulated variations adequately reflect real-world dynamics.

E. Single Mass Domain Randomization on Hopper Environment

To further investigate the impact of domain randomization on policy robustness, we conducted an additional experiment to isolate the contributions of individual masses. Specifically, we applied domain randomization singularly to each link mass (thigh, leg, and foot) while keeping the others fixed. This experiment aimed to identify which masses most significantly influence the results observed in the prior domain randomization experiments. The outcomes are summarized in Table IV.

Under UDR, all three single mass randomizations produced consistent performance between Source $\rightarrow$ Source and Source $\rightarrow$ Target configurations, showing good generalization. Among these, the leg mass randomization had the most significant impact, yielding the best results with quite low variance. Conversely, foot mass randomization led to significantly lower performance, suggesting that variability only in this parameter introduces dynamics that are particularly difficult for the policy to handle effectively.

TNR showed surprising improvements with single mass randomization compared to the full DR experiment. Notably, the Source $\rightarrow$ Source and Source $\rightarrow$ Target performances aligned well, which was an issue in the previous experiment with all masses randomized simultaneously. Surprisingly, all three single mass randomizations produced robust performance with TNR, outperforming the results obtained in the previous experiment. Also in this case, the leg mass randomization achieved the best performance. This highlights that randomizing a single mass at a time may reduce the instability introduced by simultaneous randomization of multiple dynamics parameters.

Nevertheless, considering the results from the all-masses randomization in the previous experiment, we will continue using Uniform Domain Randomization going forward, rather than Truncated Normal Domain Randomization.

F. Increase Source-Target Gap in Hopper Environment

To further challenge the model's ability to generalize across environments, we increased the domain gap by modifying the source environment's torso mass to be $-2\,\mathrm{kg}$, compared to the target domain configuration. This adjustment amplifies the

TABLE IV
IMPACT OF THIGH, LEG, AND FOOT RANDOMIZATION USING UNIFORM AND NORMAL DISTRIBUTIONS IN HOPPER ENVIRONMENT.

Test reward (avg ± std)	**Uniform**			**Normal**		
	Thigh	**Leg**	**Foot**	**Thigh**	**Leg**	**Foot**
Source → Source	1459.02 ± 101.61	1721.99 ± 104.94	1320.71 ± 268.69	1725.21 ± 17.97	1791.63 ± 246.64	1768.91 ± 53.58
Source → Target	1466.23 ± 112.99	1723.13 ± 103.06	1321.97 ± 271.90	1725.12 ± 17.60	1786.45 ± 261.22	1769.61 ± 52.63

discrepancy between the source and target domains, providing a more rigorous test of policy robustness.

As in previous experiments, we employed Uniform Domain Randomization for all link masses (thigh, leg, and foot). The goal was to evaluate the impact of this increased domain gap on baseline performance and assess the effectiveness of UDR in improving generalization.

The results, summarized in Table V, reveal that the larger domain gap leads to a general decrease in performance across all configurations compared to the original setup (Table II and Table III). However, models trained with UDR show noticeable improvements in generalization, with rewards increasing slightly and approaching those of the Target → Target configuration, though not yet reaching comparable values. Despite the increased difficulty in achieving better results due to the larger gap, UDR proves to be highly effective, improving performance significantly and highlighting its value in training more robust policies under challenging domain conditions.

TABLE V
ANALYSIS OF HOPPER PERFORMANCE UNDER INCREASED SOURCE-TO-TARGET GAP.

Model Setup	**Test Reward (avg ± std)**
Source → Source (no DR)	1190.83 ± 154.54
Source → Target (no DR)	1190.58 ± 160.12
Source → Source (UDR)	1412.48 ± 249.39
Source → Target (UDR)	1416.21 ± 256.35
Target → Target	1696.48 ± 93.64

G. Baseline Models Evaluation on Walker2d Environment

Following the methodology applied to the Hopper environment, we conducted a baseline evaluation on the Walker2d environment. The Walker2d environment is similar to Hopper but features a more complex structure with doubled body segments, making it a suitable extension for our study on sim-to-real transfer.

In the source environment of Walker2d, similar to Hopper, a domain gap is introduced by reducing the torso mass by 1kg compared to the target environment. Using the same PPO algorithm and hyperparameters as described in the Hopper experiments (Section IV-C), we trained and tested models in three configurations: Source → Source, Source → Target, and Target → Target.

The results are summarized in Table VI. Both the Source → Source and Target → Target configurations show strong performance, as policies are fully adapted to their respective domains. In contrast, the Source → Target configuration exhibits slightly lower rewards due to the domain gap, reflecting the challenges of cross-domain generalization.

Additionally, the variance across all three configurations is notably high, indicating that the policies' performance can be unstable, particularly when faced with domain gaps. The results of these experiments are largely consistent with those observed in the Hopper environment. However, in this case, the Source → Target configuration achieves relatively lower rewards in comparison, whereas in the Hopper environment, we found that Source → Target was more aligned with Source → Source performance.

TABLE VI
BASELINE PERFORMANCE OF WALKER2D MODELS ACROSS SOURCE AND TARGET DOMAINS.

Configuration	**Test Reward (avg ± std)**
Source → Source	2376.83 ± 765.40
Source → Target	2039.69 ± 940.58
Target → Target	2293.77 ± 453.39

H. Domain randomization on Walker2d environment

To further evaluate the robustness of policies, we applied domain randomization to the Walker2d environment, following the same methodology as in the Hopper experiments (Section IV-D and Section IV-E).

1) All Mass Randomization: In this setup, all body segment masses were randomized simultaneously, except the torso mass. The results, presented in Table VII, indicate that UDR consistently outperformed TNR in both Source → Source and Source → Target configurations. The higher average rewards and lower variance observed with UDR highlight its effectiveness in training policies that generalize across domains. This result is in line with the findings from the Hopper experiments, where UDR similarly improved performance and stability across varying domain configurations.

TABLE VII
COMPARISON OF UNIFORM AND GAUSSIAN DOMAIN RANDOMIZATION IN THE WALKER2D ENVIRONMENT

Configuration	**UDR (avg ± std)**	**TNR (avg ± std)**
Source → Source	2854.67 ± 583.67	1605.18 ± 761.54
Source → Target	2861.97 ± 576.78	1589.14 ± 749.07

2) Single Mass Randomization: Building on insights from the Hopper experiments, we investigated the impact of randomizing individual body segments (thigh, leg, and foot) in the Walker2D environment. To ensure consistency and symmetry in the randomization process, we implemented a unified sampling approach: when randomizing a specific segment, such as the thigh, the same random value was applied to both the left

TABLE VIII
IMPACT OF THIGH, LEG, AND FOOT RANDOMIZATION USING UNIFORM AND NORMAL DISTRIBUTIONS IN WALKER2D ENVIRONMENT.

Test reward (avg ± std)	**Uniform**			**Normal**		
	Thigh	**Leg**	**Foot**	**Thigh**	**Leg**	**Foot**
Source → Source	3360.91 ± 785.10	4228.56 ± 845.09	3919.82 ± 975.15	3109.39 ± 1116.03	1221.81 ± 725.5	2656.49 ± 779.30
Source → Target	3345.01 ± 812.06	4138.72 ± 826.53	3939.74 ± 956.36	3201.08 ± 1176.75	1260.49 ± 740.97	2689.84 ± 772.25

and right counterparts, preserving the symmetry of the robot's dynamics.

The results, summarized in Table VIII, show that single mass domain randomization is significantly more effective than all-masses domain randomization, achieving much better results for all three individual masses and both types of domain randomization, except for TNR with the leg mass, which showed worse performance. Overall, the behavior is similar to that observed in the Hopper environment, but in this case, single mass randomization delivers far superior results, even surpassing the previous performance (including Target → Target) reported in Table VI and Table VII. This improvement may be due to the fact that randomizing each mass individually (in pairs, in fact) provides better benefits compared to randomizing all masses simultaneously, especially in a more complex environment like Walker2d. For UDR, the best domain randomization performance is achieved by randomizing only the two legs, while for TNR, the best performance is obtained by randomizing both thighs.

I. Increase Source-to-Target Gap on Walker2d

Building on the methodology applied in the Hopper environment, where the domain gap was increased by reducing the torso mass by 2kg instead of 1kg, the Walker2d environment was modified to create an even larger domain gap by tripling the mass of the feet in the source environment compared to the target environment. The results are reported in Table IX.

TABLE IX
ANALYSIS OF WALKER2D PERFORMANCE UNDER INCREASED SOURCE-TO-TARGET GAP.

Model setup	**Test Reward (avg ± std)**
Source → Source (no DR)	2957.91 ± 894.13
Source → Target (no DR)	2894.83 ± 896.59
Source → Source (UDR)	4158.03 ± 757.54
Source → Target (UDR)	4140.05 ± 784.25
Source → Source (TNR)	2314.29 ± 1770.93
Source → Target (TNR)	2353.83 ± 1781.62
Target → Target	2293.77 ± 453.39

We hypothesized that this increased domain gap would result in a noticeable degradation in performance, particularly for models trained without domain randomization. Surprisingly, the results obtained in this experiment outperform the baseline results (Table VI), contrary to our expectations. Both the Source → Source and Source → Target configurations far exceed the baseline results, even surpassing the Target → Target performance, essentially eliminating any domain gap. Despite this, adding UDR significantly improves performance, yielding the best results across all experiments conducted. In contrast, using TNR reduces performance compared to no domain randomization, with extremely high variance. These results do not align with those obtained in the dual-experiment conducted in the Hopper environment, where we observed a general degradation of performance.

V. CONCLUSION

This study investigated the effectiveness of domain randomization techniques in addressing the sim-to-real transfer challenge, with a focus on the Hopper and Walker2d environments. Our experiments demonstrated that Uniform Domain Randomization (UDR) consistently outperforms Truncated Normal Domain Randomization (TNR) across all tested scenarios. UDR not only achieved higher average rewards but also exhibited lower variance, indicating enhanced robustness and stability.

Experiments with single-mass randomization showed that certain dynamics parameters, particularly leg mass, have a significant impact on policy performance. In many cases, single-mass randomization even outperformed all-mass randomization. One area we haven't explored yet, but could be worth investigating, is the combination of multiple masses, such as pairs like thigh and leg, or foot and leg. This approach could provide deeper insights into how different body parts interact and further improve policy performance.

Increasing the domain gap amplified the challenges of generalization, yet policies trained with UDR continued to exhibit strong performance, often surpassing baseline configurations trained directly on the target environment. This underscores the adaptability of UDR to handle a wide range of dynamic differences.

In conclusion, domain randomization, particularly UDR, proves to be a powerful approach to bridging the sim-to-real gap. By exposing reinforcement learning agents to diverse dynamics during training, this method encourages robust policy generalization across varying conditions. These results provide a solid foundation for advancing RL applications.

Future work could involve exploring different types of domain randomization and testing them across a wider range of parameters. This could help identify new strategies to make RL policies more reliable and adaptable in a wider range of real-world situations.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Second Edition)*. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] 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.

[3] 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.

[4] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, M. Mozifian *et al.*, "Perspectives on sim2real transfer for robotics: A summary of the r:ss 2020 workshop," in *Robotics: Science and Systems (RSS) Workshop*, 2020. [Online]. Available: https://arxiv.org/abs/2012.03806

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[6] 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," *CoRR*, vol. abs/1703.06907, 2017. [Online]. Available: http://arxiv.org/abs/1703.06907

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html