Sim-to-Real Transfer in Robotic Reinforcement Learning: Methods and Implementations

Project link on Github: https://github.com/Adelapt98/mldl

Mohammmad Saghali
Politectnico di Torino
s328134@studenti.polito.it

Pouria Mohammadalipourahari
Politectnico di Torino
s327015@studenti.polito.it

Adel Aboutalebi Pirnaeimi
Politectnico di Torino
s324930@studenti.polito.it

***Abstract*—This project explored the application of reinforcement learning (RL) to robotic systems, focusing on the sim-to-real transfer problem. The objective was to develop control policies in a simulated environment that can be effectively transferred to real-world tasks. Initially, we implemented basic RL algorithms, specifically REINFORCE and Actor-Critic Policy Gradient, to train an RL agent. Subsequently, advanced RL algorithms such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) were utilized to establish performance baselines. A significant part of the project was dedicated to implementing Uniform Domain Randomization, a technique designed to enhance the robustness of policies against variations in environmental dynamics, such as changes in mass coefficients. By randomizing these parameters during training, the RL agent learned to perform reliably across a range of conditions, thereby improving its adaptability to real-world scenarios. In the final phase, we extended the project by proposing and implementing novel methods to further refine the sim-to-real transfer process. This included advanced domain randomization techniques, curriculum learning methods, and exploring the adaptability of policies in dynamic environments.**

I. Introduction

Reinforcement learning (RL) is a machine learning paradigm where an agent learns to make decisions through interactions with an environment, optimizing its actions to achieve specific goals [1]. In the field of robotics, RL has been pivotal due to its potential to enable autonomous decision-making in complex and dynamic scenarios. Applications of RL in robotics are diverse, including robot control tasks such as locomotion, manipulation, and navigation, as well as task planning, adaptive behavior, skill acquisition, and coordination among multi-robot systems. One of the most promising and challenging applications of RL in robotics is sim-to-real transfer, where policies trained in simulation are deployed in real-world environments [2].

The sim-to-real transfer problem, or Sim2Real transfer, represents a critical challenge in robotics. This problem involves effectively transferring knowledge or learned policies from simulated environments to the real world. Simulated environments are favored for training due to their controlled, cost-effective, and safe nature. However, a significant issue arises when the policies learned in these simulations fail to perform as expected in real-world scenarios, a discrepancy known as the reality gap. The reality gap is caused by differences in dynamics, sensor noise, actuator limitations, and environmental uncertainties that are difficult to model accurately in simulations, leading to performance degradation or failure when deploying simulated policies on physical robots. To address this challenge, researchers have developed various techniques aimed at improving the robustness and generalization of learned policies. Domain randomization introduces random variations during simulation training to make policies more robust to real-world conditions. [3]–[6].

In this project addressing the sim-to-real problem, we use a custom hopper environment for simulation. After training in this environment, we test in another custom hopper environment with slight modifications from the training setup. We simulate the sim-to-real transfer task in a sim-to-sim scenario by manually injecting discrepancies between the source (training) and target (test) domains. We employ domain randomization of dynamics parameters (masses), a popular strategy to develop robust policies that transfer well to the target domain. Additionally, we use an Adaptive Domain Randomization method, DROID, and implement a curriculum learning method, AutoDR, for domain randomization to enhance sim-to-real transfer in our simplified scenario.

II. Related Work

Domain adaptation has been widely studied to address the challenge of transferring knowledge from simulated environments to real-world applications. It aims to reduce the domain shift by aligning feature distributions. By adapting models trained on simulated data to perform well in real-world settings, domain adaptation techniques aim to improve the robustness and applicability of AI systems across different domains and environmental conditions [4], [7].

Meta-learning, or learning to learn, enhances the adaptability of reinforcement learning models to new environments by enabling fast adaptation to new tasks with minimal adjustments. Rather than learning all the characteristics of the world, the key idea is to learn how to adapt to a changing world in simulation. Meta-learning approaches have shown promise in sim-to-real transfer by allowing policies to quickly adjust to real-world dynamics after being trained in simulation [3], [6].

III. Methodology

In this section, we will provide a comprehensive overview of the methodologies employed in our study for sim-to-real reinforcement learning task. We will begin by describing our simulation and target environments, highlighting the scenarios in which our algorithms will be applied. Following this, we will delve into the specific reinforcement learning methods utilized in our research, including basic RL agents, advanced RL training pipelines, and domain randomization techniques. We will conclude by discussing the project extensions and the innovative approaches we developed to enhance the sim-to-real transfer.

A. Basic Policy Gradient RL Algorithms

We train the agent using two basic policy gradient reinforcement learning algorithms to develop a simple control policy for the Hopper environment. Both REINFORCE and the vanilla version of the actor-critic method are on-policy algorithms, which means they learn from the actions taken by the current policy rather than from a replay buffer of past experiences.

Policy gradient methods aim to directly optimize the policy by adjusting its parameters to maximize the expected cumulative reward. The core idea is to compute the gradient of the expected reward with respect to the policy parameters and update the policy in the direction of this gradient. On-policy methods, such as REINFORCE and actor-critic, use data generated by the current policy to update that same policy. The policy gradient theorem provides the foundation for these methods [8]. It states that the gradient of the expected cumulative reward $J(\theta)$ with respect to the policy parameters θ can be expressed as shown in Equation 1:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) G \right] \tag{1}$$

where $\pi_\theta(a \mid s)$ is the policy, and G is the return.

These algorithms, REINFORCE and actor-critic, are particularly well-suited for learning in the Hopper environment due to their ability to handle continuous action spaces and their effectiveness in optimizing policies directly from the environment's state observations.

*1) **REINFORCE**:* The REINFORCE algorithm is a straightforward implementation of the policy gradient method. It updates the policy parameters by increasing the probability of actions that lead to higher rewards. The update rule for REINFORCE is shown in Equation 2:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t \tag{2}$$

where α is the learning rate, $\pi_\theta(a_t \mid s_t)$ is the policy, and G_t is the cumulative reward from time step t.

*2) **REINFORCE with Baseline**:* To reduce the high variance in the gradient estimates, a baseline $b(s)$ can be subtracted from the return G_t. This adjustment does not change the expected value of the gradient but can significantly reduce its variance, leading to more stable learning. The update rule with a baseline becomes as shown in Equation 3:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) (G_t - b(s_t)) \tag{3}$$

Common choices for the baseline include the average return or a value function estimate.

Using a baseline in REINFORCE is crucial in practice as it helps in reducing the variance of gradient estimates, which can lead to more stable and efficient learning processes.

*3) **Actor-critic**:* The actor-critic method combines the benefits of policy gradient methods and value-based methods. It consists of two components:

- **Actor**: Updates the policy parameters in the direction suggested by the critic.
- **Critic**: Evaluates the actions taken by the actor using a value function.

The update rule for the actor in the actor-critic method is similar to that of REINFORCE but uses the advantage function $A(s_t, a_t)$ instead of the return. The update rule is shown in Equation 4:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t) \tag{4}$$

The advantage function $A(s_t, a_t)$ is typically computed as the difference between the observed return G_t and the value function $V(s_t)$, as shown in Equation 5:

$$A(s_t, a_t) = G_t - V(s_t) \tag{5}$$

The critic is updated by minimizing the temporal difference error, as shown in Equation 6:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{6}$$

where r_t is the reward at time step t and γ is the discount factor.

By combining these two components, the actor-critic method provides more stable updates and faster convergence compared to REINFORCE.

B. Advanced RL Training Pipelines

In this part, we outline the advanced reinforcement learning training pipelines used to enhance the control policies for the Hopper environment. We utilize Proximal Policy Optimization to train the agent more effectively. PPO is an advanced, on-policy algorithm that strikes a balance between performance and computational complexity, making it suitable for complex environments like the Hopper.

*1) **Proximal Policy Optimization (PPO)**:* PPO is a powerful policy gradient method that enhances the stability and performance of basic policy gradient algorithms. It achieves this by using a clipped objective function to restrict the magnitude of policy updates, preventing large, destabilizing updates. The PPO algorithm alternates between sampling data through interaction with the environment and optimizing a surrogate objective function using stochastic gradient ascent.

The clipped objective function in PPO is defined as shown in Equation 7:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \right) \right] \tag{7}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, $\hat{A}_t$ is the estimated advantage at time step t, and ϵ is a hyperparameter that controls the clipping range.

By limiting the extent of policy updates, PPO ensures a more stable and reliable learning process. This makes it particularly effective for environments where maintaining stability is crucial, such as the Hopper.

PPO enhances stability and reliability compared to REINFORCE and Actor-Critic by using a clipped surrogate objective that prevents large policy updates, ensuring smoother training. PPO also improves sample efficiency with multiple epochs of mini-batch updates, making better use of data. Unlike REINFORCE, which suffers from high variance and inefficient sample use, and Actor-Critic methods, which can be unstable without careful tuning, PPO balances simplicity and performance effectively. It includes value estimation error and entropy terms to encourage exploration.

For our implementation of PPO, we utilized the stable-baselines3 library, which provided a reliable and efficient framework for our reinforcement learning task.

C. Uniform Domain Randomization

Domain randomization involves varying the simulation parameters to expose the RL agents to a wide range of scenarios, thereby enhancing their generalization capabilities. Uniform domain randomization ensures that the parameters are varied uniformly across their ranges. Our implementation of this technique significantly improves the robustness of the RL agents, enabling better sim-to-real transfer [5].

In our experiments, we implemented Uniform Domain Randomization (UDR) for the link masses of the Hopper robot. In this setting, UDR refers to manually designing a uniform distribution over the three remaining masses in the source environment (considering that the torso mass is fixed at -1 kg w.r.t. the target one) and performing training with values that vary at each episode, sampled appropriately from the chosen distributions. By varying the link masses uniformly across specified ranges, we exposed the RL agent to diverse dynamics during training.

The underlying idea is to force the agent to maximize its reward and solve the task for a range of multiple environments simultaneously. This approach allows the agent to learn a more generalized policy, which is crucial for handling discrepancies between the source (simulation) and target (real-world) environments.

By generalizing the source environment through uniform domain randomization, we achieve more robust training. The variation in training scenarios ensures that the agent does not overfit to a narrow set of conditions but instead develops strategies that are effective across a broader spectrum of situations. This leads to better performance and stability when the policy is transferred to the real-world environment, making the agent more resilient to the variations it might encounter.

D. DROID: Adaptive Domain Randomization

The DROID methodology enhances reinforcement learning training through three main phases: human demonstration, parameter identification and optimization, and policy learning with optimized domain randomization (DR). This approach aims to improve data efficiency and policy robustness, leading to better performance in real-world tasks [9].

1) Human Demonstration: In the first phase, a human provides a single-shot demonstration of a safe trajectory for the robot to interact with the environment. The feedback from this demonstration is used to identify and update the parameter distribution in the simulation. Unlike random interaction approaches, DROID relies on human guidance to ensure safe trajectory identification for understanding system dynamics.

2) Parameter Identification and Optimization: The second phase focuses on correctly identifying the parameter distribution that minimizes the discrepancy between the simulated and real systems. Instead of finding specific parameter values, DROID determines distributions that consider noise and uncertainties in the real world. Training the RL model within this optimized distribution is key to minimizing the reality gap.

3) Policy Learning with Optimized DR: In the final phase, the optimized parameter distribution from the previous phase is used to train the RL model for policy learning. By aligning simulated trajectories with real-world trajectories, DROID enables efficient policy transfer and generalization in robotic tasks. This approach limits random exploration in the real world, reducing risks and saving time during real robot experiments.

By automating the optimization of environment parameter distribution and combining domain randomization with system identification, DROID significantly enhances policy transfer and learning efficiency in robotic applications.

E. ADR: Automatic Domain Randomization

Automatic Domain Randomization (ADR) enhances reinforcement learning by training models on a maximally diverse distribution of randomized environments. This method follows a curriculum learning approach, starting with zero randomization and gradually increasing the randomization level during training. This progressive approach allows the model to adapt to increasingly complex environments over time [10].

1) Training on Diverse Environment Distribution: ADR trains models on a diverse range of randomized environments by gradually expanding the randomization ranges. This approach maximizes the distribution over environments, exposing the model to a wide variety of scenarios.

2) Curriculum Learning with ADR: Starting with zero randomization, ADR increases the randomization level incrementally. This curriculum learning method helps the model adapt progressively, enhancing its ability to handle complex and varied environments.

3) Comparison with Manual Domain Randomization: ADR automates the adaptation process to changes in hardware and simulation setups, providing broader training distributions and improved transfer capabilities compared to manual domain randomization.

4) Sim2Real Transfer: Policies trained with ADR show strong sim2real transfer capabilities at test time. The models can adjust their behavior based on specific environments, even

those not encountered during training, enhancing robustness and generalization across different tasks.

By incorporating ADR, we leverage its ability to train models on diverse environments, facilitate sim2real transfer, and enable adaptive behavior in real-world scenarios, thereby improving the overall robustness and generalization of our solution.

IV. Experimental Results and Discussion

In this section, we present and discuss the results of the various RL algorithms and methods described in the methodology section. First, we explain the simulation and target environments used to train and test the agent. We then compare the performances of these algorithms on the Hopper environment and analyze the effectiveness of each approach. For better performance comparison, we use lower and upper bound baselines as reference frameworks.

A. The Gym Hopper environment

In this study, we utilize the Gym Hopper environment to facilitate our research on sim-to-real reinforcement learning tasks. The Hopper is a two-dimensional, one-legged figure composed of four main body parts: the torso at the top, the thigh in the middle, the leg at the bottom, and a single foot on which the entire body rests. The objective of the Hopper is to perform hops that propel it forward (in the right direction) by applying torques to the three hinges connecting the four body parts.

1) Simulation Environment: Our simulation environment, referred to as the source domain, is designed based on the Gym Hopper environment. The action space of the Hopper consists of three continuous values, each representing the torque applied at one of the three hinge joints. The observation space comprises positional values of the various body parts of the Hopper, followed by the velocities of these parts (their derivatives), with all positions ordered before all velocities. This setup provides a comprehensive state representation that the reinforcement learning agent uses to make informed decisions.

2) Target Environment: The target environment, or real-world scenario, is also modeled after the Gym Hopper environment but with a deliberate discrepancy to study the sim-to-real transfer. Specifically, the torso mass in the target environment is fixed at -1 kg relative to the simulation environment. This manually injected discrepancy between the source (training) and target (test) domains presents a challenge for the RL agent, as it must adapt to this difference to perform effectively in the real world.

The primary goal of our study is to ensure that the RL agent, trained in the simulation environment, can successfully transfer its learned policies and perform well in the target environment despite the introduced discrepancies. This sim-to-real transfer is crucial for developing robust RL algorithms capable of functioning effectively in real-world applications. By systematically addressing these challenges, our research aims to bridge the gap between simulated training and real-world performance.

We evaluated the performance of different RL algorithms, including REINFORCE, REINFORCE with Baseline, Actor-Critic, PPO, Uniform Domain Randomization, DROID, and Automatic Domain Randomization. The performance was measured based on the average return over 50 test episodes for each algorithm in the Hopper environment.

B. Lower/Upper Bound Baselines

Establishing lower and upper bound baselines provides a reference framework for evaluating the performance of RL algorithms. Lower bound baselines represent minimal performance expectations, while upper bound baselines indicate the best possible performance under ideal conditions. Our study includes a comparative analysis of these baselines with advanced RL methods to highlight the effectiveness of our approaches.

- Source $\rightarrow$ Target (Lower Bound): In this configuration, the agent is trained on the source domain and tested on the target domain. We generally expect lower performance here compared to the Target $\rightarrow$ Target configuration due to the domain shift. The discrepancies between the source (training) and target (test) domains lead to a performance drop, highlighting the challenges of sim-to-real transfer.
- Target $\rightarrow$ Target (Upper Bound): This configuration involves training and testing the agent directly on the target domain. It represents the upper bound of performance, as the agent is perfectly aligned with the test environment. While this configuration provides the best performance, it is often impractical in a sim-to-real setting due to the difficulty and expense of extensive training in the real world.

C. Results

1) Performance of Basic RL Algorithms: We assessed the performance of three fundamental RL algorithms: REINFORCE, REINFORCE with Baseline, and Actor-Critic. Each algorithm underwent training in the source environment and subsequent testing in the target environment. Performance was evaluated based on the average return over 50 test episodes per algorithm. REINFORCE achieved a mean reward of 296 on the target environment, while REINFORCE with Baseline achieved a mean reward of 320, with a baseline of 20. Actor-Critic, utilizing a modified learning rate reduced from the default 1e-3 to 1e-4, achieved the highest mean reward of 381 on the target environment.

Figure 2 demonstrates that REINFORCE achieves the lowest mean reward of 296 among the three methods due to its basic approach, resulting in high variance in gradient estimates that lead to instability and slower convergence. Conversely, REINFORCE with Baseline shows improved performance with a mean reward of 320. The inclusion of a baseline reduces variance, facilitating more stable and efficient learning updates. Actor-Critic surpasses both methods with a mean

reward of 381. Its dual structure, where the critic evaluates actions chosen by the actor, yields more informed and precise policy updates. This integration significantly enhances learning efficiency and stability, emphasizing the advantages of incorporating variance reduction and evaluative mechanisms in reinforcement learning algorithms.

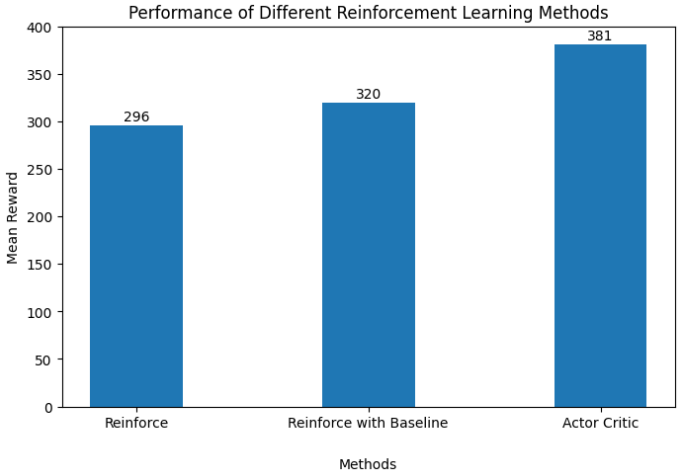

Fig. 1: The figure illustrates the performance measured through the average return over 50 test episodes for each basic algorithm.

2) Performance of Advanced RL Algorithms and Methods: PPO is a powerful policy gradient method that enhances the stability and performance of basic policy gradient algorithms. When trained on the source environment and tested on the target environment, PPO achieves a mean reward of 1242, significantly outperforming the basic RL algorithms discussed earlier.

For better evaluation and comparison of the methods used in the following sections, we establish upper and lower bound baselines for the PPO algorithm. When the agent is trained and tested in the source-target configuration, it represents the lower bound due to disparities between environments. Conversely, the agent achieves the upper bound when trained and tested in the target-target configuration, indicating minimal disparities, as illustrated in Table I.

PPO	Scenario	Mean Reward
PPO	source - source	1490
	source - target (Lower bound)	1242
	target - target (Upper bound)	1626

TABLE I: Mean Reward for different PPO scenarios

By adding UDR to PPO, Initially, we considered a fixed weight variation range for each body part (with ranges between -0.5 to 0.5 and -1 to 1). Through trial and error, we concluded that each body part should have a different variation range proportional to its weight. Specifically, a body part with a lower weight should have a smaller variation range, and conversely, a body part with a higher weight should have a larger variation range. The UDR algorithm was trained over 300,000 episodes and achieved a mean reward of 1527. The variation ranges used were [-0.3, 0.3] for the thight, [-0.2, 0.2] for the leg, and [-0.5, 0.5] for the foot. These configurations were found to optimize the algorithm's performance, resulting in a mean reward of 1527.

In the ADR algorithm, the initial ranges for domain randomization were set to zero, indicating the absence of domain randomization at the start. We defined two mean reward thresholds: a low threshold of 50 and a high threshold of 200. Additionally, we established an update step amount of 0.5 and an evaluation window of 100 episodes. Whenever the evaluation window reaches 100 episodes, the update step mechanism adjusts the ranges. Specifically, if the reward exceeds the high threshold, the range for each body part is increased by an update step size of 0.01. Conversely, if the reward falls below the low threshold, the range is decreased by the same update step size of 0.01. If the reward falls between the thresholds, no adjustments are made. This adaptive approach incrementally increases the difficulty for the model as it trains, thereby exposing it to progressively more challenging conditions over time. We used ADR with PPO and despite the anticipated potential of this algorithm, the results did not significantly surpass those achieved with the UDR algorithm. The ADR algorithm achieved a mean reward of 1530.

In the implementation of the DROID algorithm, we utilized our Upper bound model (a PPO model trained on the target environment) to simulate human demonstration. This approach was adopted because our project is sim-to-sim rather than sim-to-real, and we do not have access to a real-world environment.

To generate the necessary data, we executed the target environment 100 times, recording the actions, observations, and rewards. We then applied these same actions to the training (source) environment. Using this data, we employed a random optimization function to determine the suitable body masses for the source environment, as stated in TABLE II.

Subsequently, we trained the model using these obtained masses within the PPO algorithm. The results were highly promising, yielding the best mean reward achieved in our experiments, with a result of 1580.

Body Part	Value(KG)
Torso	2.534
Thigh	2.556
Leg	3.661
Foot	5.002

TABLE II: Masses of the Hopper's body parts in DROID

The following paragraph provides a comparison of various reinforcement learning methods based on their mean reward performance. As shown in Figure 2, the comparison of advanced reinforcement learning methods highlights clear performance differences. The PPO Lower Bound (source to target) serves as the baseline with a mean reward of 1242, while the PPO Upper Bound (target to target) achieves the highest mean reward of 1626, representing optimal performance.

Among the domain randomization techniques, UDR shows a significant improvement with a mean reward of 1527. AutoDR

slightly surpasses UDR with 1530, and DRIOD achieves the highest among these techniques with 1580, nearing the upper bound's performance.

In summary, advanced domain randomization methods like UDR, AutoDR, and DRIOD significantly enhance reinforcement learning performance, with DRIOD demonstrating the greatest effectiveness.

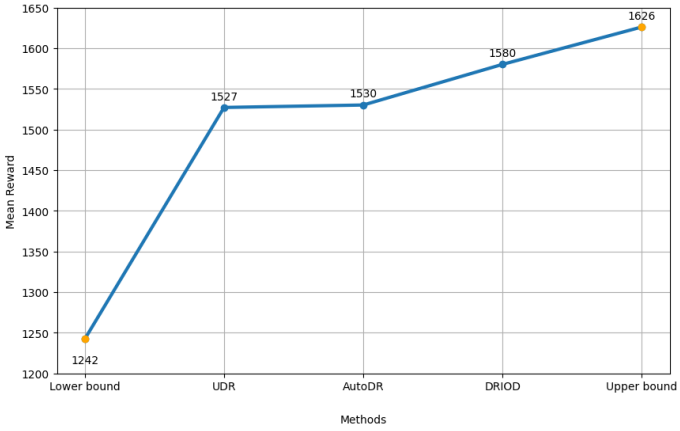

Fig. 2: The figure illustrates the performance comparison of advanced reinforcement learning methods in terms of mean reward.

V. Conclusion

This study focused on enhancing sim-to-real transfer in robotic reinforcement learning (RL) by evaluating various RL algorithms and domain randomization techniques. We implemented basic policy gradient methods (REINFORCE, REINFORCE with Baseline, and Actor-Critic) and advanced methods like PPO with different domain randomization approaches.

Key findings include that basic policy gradient methods, while foundational, are limited by high variance and slower learning rates. PPO significantly improves stability and performance compared to basic methods. UDR enhances robustness by exposing the agent to diverse dynamics during training. ADR shows slight improvements over UDR, leveraging a curriculum learning approach. Finally, DROID achieves the highest performance by combining human demonstrations, parameter optimization, and policy learning with domain randomization.

The results highlight the importance of domain randomization and adaptive learning strategies in bridging the sim-to-real gap. DROID, in particular, demonstrates significant potential for robust real-world performance.

Future work should focus on applying these methodologies to real-world robotic applications, further refining and combining techniques to develop reliable and adaptable robotic systems capable of operating in dynamic environments. This research advances the field of robotic RL, offering valuable insights for improving sim-to-real transfer.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[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] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *CoRR*, vol. abs/2111.00956, 2021.

[4] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017.

[5] 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.

[6] S. Höfer, K. E. Bekris, A. Handa, J. C. G. Higuera, F. Golemo, M. Mozifian, C. G. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. K. Liu, J. Peters, S. Song, P. Welinder, and M. White, "Perspectives on sim2real transfer for robotics: A summary of the R: SS 2020 workshop," *CoRR*, vol. abs/2012.03806, 2020.

[7] L. Weng, "Domain randomization for sim2real transfer," *lilianweng.github.io*, 2019.

[8] L. Weng, "Policy gradient algorithms," *lilianweng.github.io*, 2018.

[9] Y. Tsai, H. Xu, Z. Ding, C. Zhang, E. Johns, and B. Huang, "DROID: minimizing the reality gap using single-shot human demonstration," *CoRR*, vol. abs/2102.11003, 2021.

[10] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019.