Project Robot Learning

1st Rialti Jacopo
LM Ing. Informatica
Politecnico di Torino
Torino, Italy
s346357@studenti.polito.it

2nd Giunti Alberto
LM Ing. Informatica
Politecnico di Torino
Torino, Italy
s336374@studenti.polito.it

3rd Gjinaj Stiven
LM Ing. Informatica
Politecnico di Torino
Torino, Italy
s333805@studenti.polito.it

***Abstract*—This paper investigates the application of reinforcement learning algorithms in robotic manipulation tasks and trade-offs between specialized and adaptable policies in robotic manipulation tasks, specifically focusing on the Pusher environment from the Gym suite. We compare the performance of Soft Actor-Critic and Proximal Policy Optimization algorithms in training a robotic arm to push objects toward target positions. The study explores various environmental modifications, including the introduction of fixed and randomized obstacles, dynamic goal positions, and density variations.**

***Index Terms*—sim-to-real transfer, reinforcement learning, domain randomization, robotics**

I. Introduction

Reinforcement learning in robotics presents unique challenges due to the continuous nature of state and action spaces, the complexity of physical interactions, and the need for robust policies that can generalize across different environmental conditions. While significant progress has been made in training robots in simulated environments, the transfer of learned policies to real-world applications remains a critical challenge in the field.

This study focuses on the Pusher environment, a benchmark robotic manipulation task where an agent must learn to control a multi-joint robotic arm to push objects toward specified targets. We investigate several key aspects of robotic learning: the comparative performance of modern RL algorithms (SAC and PPO), the impact of environmental variations through obstacle placement and goal positioning, and the effects of physical parameter modifications.

II. Pusher Environment and Analysis

The Pusher environment, part of the Gym suite, is a robotic task where a robotic arm learns to push an object toward a target position. Training policies in this environment require working with a continuous state and action space, making it a useful benchmark for reinforcement learning algorithms. [9]

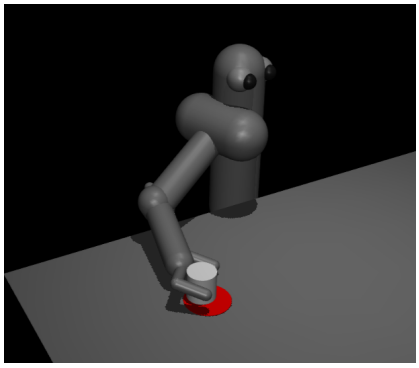

Fig. 1. Screenshot Pusher default.

In the simulation, the environment models the movement of the robotic arm with specific parameters:

A. State Space

The state space of the Pusher environment is continuous, consisting of 20 variables. These variables represent the robotic arm's joint positions, velocities, object positions, and target positions. They describe the current state and movement of the system, which are essential for decision-making during training.

TABLE I
State Space Variables of the Pusher Environment

Obs.	Name	Unit	Range
0-6	Joint positions	rad	$(-\infty, \infty)$
7-13	Joint velocities	rad/s	$(-\infty, \infty)$
14-16	Object position	m	$(-\infty, \infty)$
17-19	Target position	m	$(-\infty, \infty)$

B. Action Space

The action space is continuous and consists of seven variables, each representing the torque applied to a specific joint of the robotic arm. These torques directly control the movement of the arm, allowing it to push the object towards the target.

TABLE II
ACTION SPACE VARIABLES OF THE PUSHER ENVIRONMENT

Num	Action	Joint	Min	Max	Unit
0	Shoulder pan	r_shoulder_pan	-2	2	torque
1	Shoulder lift	r_shoulder_lift	-2	2	torque
2	Upper arm roll	r_upper_arm_roll	-2	2	torque
3	Elbow flex	r_elbow_flex	-2	2	torque
4	Forearm roll	r_forearm_roll	-2	2	torque
5	Wrist flex	r_wrist_flex	-2	2	torque
6	Wrist roll	r_wrist_roll	-2	2	torque

C. Environment Dynamics and Behavior

The arm's movements are controlled by joint torques, with dynamics influenced by factors such as friction and inertia. The object is moved based on the applied forces, and the goal is to push it to the target position.

To introduce variability during training, the object's initial position is randomized within a predefined range. To ensure a meaningful interaction with the target, the object is repositioned until it is at least 0.17 units away from the goal. This prevents trivial cases where the object starts too close to the goal, encouraging the agent to learn a more generalizable pushing strategy.

Episodes conclude upon reaching the maximum of 1000 timesteps. The environment resets after each episode to maintain consistency during training.

The total reward is defined as:

$$reward = reward\_dist + reward\_ctrl + reward\_near. \quad (1)$$

reward_near: Measures the distance between the pusher's fingertip (unattached end) and the object. A more negative value is assigned when the fingertip is farther from the target.

$$reward\_near = -|vec\_1| \quad (2)$$

where **vec_1** represents the vector from the pusher's fingertip to the object.

reward_dist: Measures the distance between the object and its goal. A more negative value is assigned when the object is farther from the goal.

$$reward\_dist = -|vec\_2| \quad (3)$$

where **vec_2** represents the vector from the object to the goal.

reward_ctrl: A penalty for large control actions, measured as the negative squared Euclidean norm of the action.

$$reward\_ctrl = -\sum a^2 \quad (4)$$

The final reward function is:

$$reward = reward\_dist + 0.1 * reward\_ctrl + 0.5 * reward\_near. \quad (5)$$

The `info` dictionary contains the individual reward components.

III. SAC vs PPO

Soft Actor-Critic and Proximal Policy Optimization are two widely used reinforcement learning algorithms with distinct characteristics. SAC is an off-policy reinforcement learning algorithm that optimizes both a policy network and a Q-function network. It incorporates entropy maximization to encourage exploration, which helps in learning robust policies. The algorithm uses a soft Q-learning approach, which allows it to handle high-dimensional action spaces effectively. PPO, on the other hand, is an on-policy method that improves the stability of policy updates by using a clipped objective function. It restricts policy updates within a certain range to prevent drastic changes, making learning more stable and reducing variance.

The Pusher environment was trained using PPO for 200,000 timesteps and SAC for 100,000 timesteps. The performance was measured in terms of mean reward and standard deviation of rewards in 1000 episodes of test.

TABLE III
PERFORMANCE OF SAC AND PPO MODELS

Model	Mean Reward	Standard Deviation	Timesteps
SAC	-27.2975	5.9190	100,000
PPO	-27.9453	4.8330	200,000

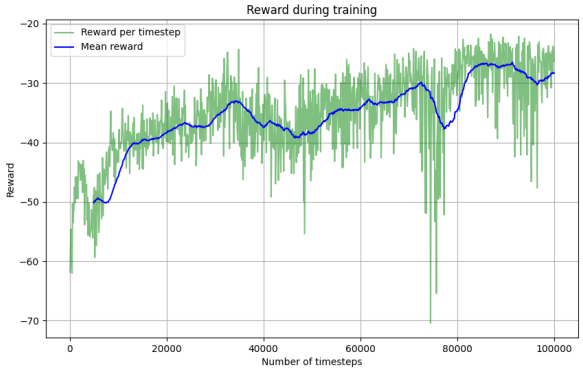

Fig. 2. Reward progression over SAC training episodes in base environment.

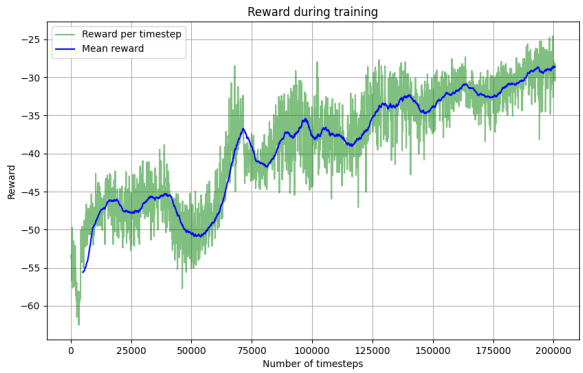

Fig. 3. Reward progression over PPO training episodes in base environment.

SAC achieved a higher mean reward compared to PPO. A less negative mean reward suggests that SAC performed better in achieving the task objectives. PPO showed lower variability in rewards compared to SAC. Lower standard deviation suggests that PPO learned a more stable policy with consistent performance across different episodes. SAC's higher standard deviation implies a greater level of variability in performance, which may be due to its exploration-heavy nature from entropy regularization. SAC, being an off-policy algorithm, generally requires fewer samples to learn effective policies. In this case, SAC performed lightly better then PPO in mean reward, even with fewer training timesteps. SAC encourages exploration through entropy regularization, which may explain the higher variability in its rewards. PPO's stability, on the other hand, resulted in more consistent performance. [10]

In terms of training time, training with SAC, took more time than with PPO. We ran the PPO task on 200000 timesteps and it took approximately half of the time of the training with SAC.

IV. Custom Environment with Fixed Goal

We have created a custom environment that includes three fixed obstacles, which remain in the same position throughout the training phase. These obstacles are modeled as cylindrical objects and serve to challenge the agent by penalizing any collisions between the robotic arm's tips and the obstacles. The obstacles have the same shape as the object the agent is required to push toward the target, making collision avoidance a critical factor in learning an effective policy.

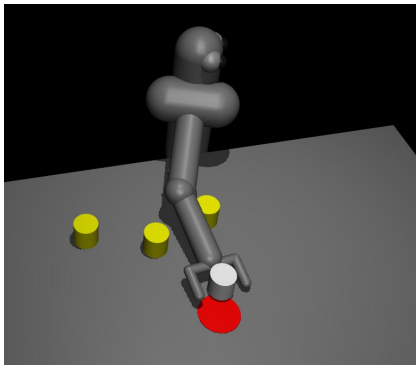

Fig. 4. Screenshot Custom Pusher.

The environment is designed to penalize the agent for each collision with the obstacles. Specifically, a penalty of -0.2 is applied when the agent's object makes contact with any of the obstacles. This penalty is intended to encourage the agent to avoid obstacles, ensuring that it focuses on pushing the object toward the goal while avoiding unnecessary collisions.

The updated reward function is now defined as:

$$reward = reward\_dist + 0.1 * reward\_ctrl + 0.5 * reward\_near + reward\_collision$$

where `reward_collision` is a penalty of -0.2 applied whenever the object collides with any of the three obstacles.

The fixed obstacles are located at the following positions in the environment:

TABLE IV
Obstacle Positions in the Custom Environment

Obstacle	Position (x, y, z)
Obstacle 1	(0.1, -0.05, -0.275)
Obstacle 2	(0.3, -0.07, -0.275)
Obstacle 3	(0.4, 0.14, -0.275)

A. Training with Fixed Obstacles and Fixed Goal

The agent was trained using the *SAC* algorithm for a total of 500,000 timesteps. In our environment, SAC was able to learn how to avoid obstacles while pushing the object toward the goal by balancing exploration with efficient control.

The performance of the agent during training was tracked by monitoring the reward over each episode. The following plot shows how the agent's reward evolves over time, reflecting the improvement in its policy as it learns to avoid obstacles and reach the goal more efficiently.

Fig. 5. Reward progression over training episodes in the custom environment.

The plot illustrates the agent's learning process over 500,000 timesteps. Initially, the agent has little to no understanding of the environment and must learn to use its robotic arm effectively. This includes learning basic control over the arm's movements and gradually improving its ability to navigate the environment. In the beginning, the agent experiences penalties due to collisions with obstacles and inefficient movements, as it is still figuring out how to interact with both the arm and the environment.

As training progresses, the agent learns to avoid obstacles and use the arm more effectively, pushing the object toward the goal more efficiently. The reduction in collisions leads to fewer penalties and an increase in the overall reward, demonstrating the agent's improving strategy over time. By penalizing the agent for collisions and inefficient movements, we guide the agent toward learning an effective policy that minimizes effort and focuses on achieving the goal, all while mastering the manipulation of the arm and avoiding obstacles.

B. Training for Real-World Adaptation with Forearm Collision Penalty

To further adapt the agent for real-world scenarios, we modified the environment and training process by extending collision detection to include the forearm, applying the same penalty that was previously used for the tips. In this version of the environment, the agent is trained not only to avoid the general obstacles but also to minimize collisions with its forearm during the task, setting `conaffinity` and `contype` true. This adjustment was made to simulate the physical constraints and challenges that a real-world robotic arm would encounter when interacting with objects and obstacles, ensuring that it learns to perform tasks in a more controlled and efficient manner, similar to how a real-world robot would avoid damaging itself or its surroundings.

The agent was trained using the *SAC* algorithm, as in the previous setup, with the added forearm collision penalty for 500,000 timesteps. The performance of the agent was evaluated by observing its ability to push the object toward the goal while minimizing collisions with both obstacles and the forearm.

The following plot shows the reward progression over time:

Fig. 6. Reward progression over training episodes with forearm collision penalty.

As shown in the plot, the agent's reward progression over time is very similar to the previous experiment. This is because the agent simply learned to adjust its movements in a way that avoided collisions with its forearm, without significantly affecting the overall reward structure. Despite the added penalty for forearm collisions, the agent still managed to push the object toward the goal effectively by adapting its control strategy. As a result, the reward values remain largely unchanged, as the forearm penalty did not drastically interfere with the agent's ability to complete the task, but rather encouraged more efficient and controlled movements. The agent learned to perform the task with minimal collision impact, reflecting its ability to adapt to the environment's constraints without major performance loss.

C. Training with Randomized Obstacles and Fixed Goal

Building upon the previous model, we created a new version where, at each episode reset, the environment is initialized with randomly placed obstacles within a defined range. This modification enhances the agent's adaptability, making it more robust in dynamic environments where obstacle positions may change over time.

To ensure meaningful obstacle placements, their positions are randomized within a region spanning the x and y coordinates of the goal and the object. This guarantees that obstacles are positioned in relevant areas where they influence the agent's decisions. Additionally, a minimum distance constraint is enforced between obstacles to prevent excessive clustering, ensuring a more realistic distribution.

The agent's observations now include the updated obstacle positions, allowing it to adapt its strategy dynamically. Since the z-coordinate remains fixed, the agent must develop a generalized approach to efficiently navigate the environment and push the object toward the goal while avoiding collisions.

Initially, we observed that the reward values with randomized obstacles were higher compared to the model trained with fixed obstacles. This happened because the model with fixed obstacles was overly dependent on the specific positions of the

obstacles. On the other hand, the randomized obstacles created a more varied environment, which encouraged the agent to develop more flexible strategies.

However, we also noticed that the fixed obstacles in the original setup were positioned in a way that made it difficult for the agent to avoid them, leading to lower reward values. To address this, we retrained a new model with fixed obstacles but increased the spacing between them. This adjustment ensured that the object to be pushed toward the goal would not necessarily collide with the obstacles, providing the agent with more space to maneuver and reducing the dependency on obstacle placement. This change resulted in better reward values and allowed for a more accurate comparison between the different models.

TABLE V
NEW OBSTACLE POSITIONS IN THE CUSTOM ENVIRONMENT

Obstacle	Position (x, y, z)
Obstacle 1	(-0.1, -0.05, -0.275)
Obstacle 2	(0.3, 0.05, -0.275)
Obstacle 3	(0.5, 0.14, -0.275)

The following two plots show the reward progression for both the fixed and randomized obstacle models:

Fig. 7. Reward progression with the newly spaced fixed obstacles.

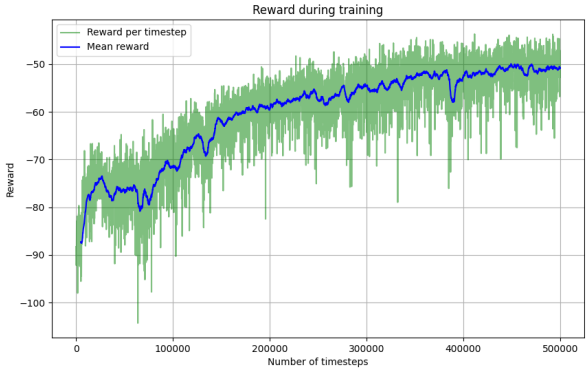

Fig. 8. Reward progression with randomized obstacles.

From the plots, we can observe that the model trained with fixed obstacles achieved a final reward value of approximately -40, while the model with randomized obstacles reached around -50. Additionally, the model with randomized obstacles exhibited greater variance in its reward progression, reflecting the increased difficulty and unpredictability of the environment. Despite this higher variance, the reward progression for the randomized obstacle model was more linear, suggesting that the agent was steadily improving as it adapted to the dynamic nature of the environment.

D. Analysis on Various Environments

We compared the performance of models trained with fixed and randomized obstacles, including the forearm collision penalty, in different environments. The results show how the environment influences the agent's performance, as summarized in the table below:

TABLE VI
COMPARISON OF REWARD FOR DIFFERENT MODELS.

Model	Env	Mean Rew.	Std Rew.
Fixed Obstacles	Fixed Obstacles	-39.19	3.59
Fixed Obstacles	Random Obstacles	-70.22	19.85
Random Obstacles	Fixed Obstacles	-52.45	10.27
Random Obstacles	Random Obstacles	-51.56	8.45

From the table, we can observe the following trends:

- **Model: fixed obstacles — Tested in: fixed obstacles** The model performed well, achieving a mean reward of -39.19 with a low variance of 3.59. This indicates that the agent was able to efficiently navigate the environment and avoid collisions with the fixed obstacles.
- **Model: fixed obstacles — Tested in: randomized obstacles** The model struggled in the randomized obstacle environment, achieving a much lower mean reward of -70.22 with a high variance of 19.85. This suggests that the agent was unable to adapt to the changing obstacle positions, leading to inconsistent performance.
- **Model: randomized obstacles — Tested in: fixed obstacles** The model achieved a mean reward of -52.45 with a variance of 10.27. While its performance was worse than the model trained specifically for fixed obstacles, it still demonstrated reasonable adaptation to the simpler environment.
- **Model: randomized obstacles — Tested in: randomized obstacles** The model obtained a mean reward of -51.56 with a variance of 8.45. This suggests that training in a dynamic environment helped the agent develop more flexible obstacle-avoidance strategies, leading to improved generalization compared to the fixed-obstacle model when tested in a randomized setting.

These results emphasize the trade-off between specialization and generalization in reinforcement learning. Training in a static environment yields better performance in similar conditions but results in poor adaptability to dynamic scenarios. Conversely, training in more variable environments enhances

flexibility and robustness at the cost of slightly reduced performance in simpler settings. This finding underscores the importance of designing training environments that balance complexity and generalization, particularly for real-world applications where conditions are rarely static.

V. Training with Randomized Goal Position

A. Training with Fixed Obstacles and Randomized Goal

In this section, we introduce a model that builds upon the fixed obstacles setup, but with the additional complexity of randomizing the goal position at each episode. This approach aims to make the agent more adaptable to dynamic environments by requiring it to push the object to a randomly chosen goal within a defined range, rather than a fixed target. The goal position is randomized within a limited x-range of $[-0.05, 0.05]$ and a y-range of $[-0.1, 0.1]$. We opted for a small range for the goal randomization to avoid drastically altering the model. A larger range would have required significantly more training time to assess whether the agent was learning effectively. Additionally, larger randomizations could have led to situations where the agent's performance was influenced more by the goal's position rather than its actual behavior, potentially hindering the training process.

This setup introduces an additional layer of complexity, as the agent can no longer rely on a fixed goal position. Instead, it must learn to generalize its behavior, ensuring that it can successfully move the object to various target locations while avoiding the obstacles.

The following plot illustrates the reward progression over time for this new setup:

Fig. 9. Reward progression with fixed obstacles and randomized goal position.

As seen in the plot, the agent achieves a maximum reward of approximately -40, with a slightly higher variance compared to the model with a fixed goal. This variance arises from the added uncertainty introduced by the random goal positions, which makes the task more challenging for the agent. The agent is forced to adapt its strategy at each episode, as the goal location changes, requiring it to generalize its movements and learn more flexible policies.

From a reinforcement learning perspective, the goal randomization leads to a non-stationary environment, as the target changes continuously. This non-stationarity forces the agent to continuously adapt its policy to reach the new goal, as opposed to memorizing a fixed strategy that would only work for a specific target location. Consequently, the agent must develop more robust and flexible policies that can generalize across different goal configurations, rather than overfitting to a specific, fixed environment.

B. Training with Randomized Obstacles and Randomized Goal

In this setup, we trained a model where both the obstacles and the goal position were randomized for each episode. The obstacles were placed using the same randomization criteria as in the previous setup, ensuring they were positioned within a defined range while maintaining a minimum distance between them to prevent excessive clustering. Similarly, the goal's position varied at each episode within a predefined region. This setup aimed to create a more dynamic and challenging environment for the agent, where not only the obstacles but also the goal location could change, requiring the agent to adapt its strategy continuously.

The following plot shows the reward progression for the model trained with both randomized obstacles and randomized goal positions:

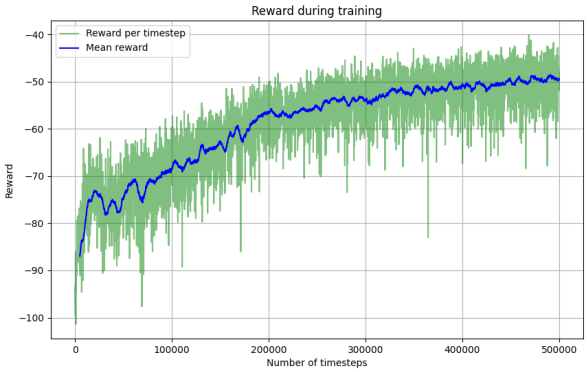

Fig. 10. Reward progression for model with randomized obstacles and goal positions.

As seen in the plot, the agent's reward increases as it learns to navigate and push the object effectively, even in the presence of both dynamically placed obstacles and goals.

The behavior of this model was very similar to the one trained with only randomized obstacles, with the main difference being the additional complexity introduced by the randomization of the goal position. As expected, the model had to adjust its navigation strategy to account for both the varying obstacles and the changing goal location. However, the performance did not drastically differ from the model trained with just randomized obstacles, with a reward progression that followed a comparable pattern. This suggests that while the randomization of the goal position introduces additional

variability, it does not significantly alter the fundamental strategy the agent learns for pushing the object towards the target. This is particularly due to the chosen ranges for goal randomization, which ensure that the goal remains within a reasonable distance from the object's starting position. As a result, the agent is not required to learn drastically different behaviors across episodes but rather refine its existing strategy to adapt to slight variations. This further indicates that the model had already developed a robust approach to obstacle avoidance, and the introduction of a random goal position did not add enough complexity to drastically change its learning behavior.

C. Analysis of Different Environments

We analyzed the performance of models trained with randomized obstacles and different goal positioning (fixed or randomized) when tested in various environments. This comparison provides insights into how well each model generalizes across different conditions.

TABLE VII
COMPARISON OF REWARD FOR DIFFERENT MODELS IN VARIOUS ENVIRONMENTS.

Train Obst.	Train Goal	Test Obst.	Test Goal	Mean	Std
Random	Fixed	Random	Fixed	-51.56	8.45
Random	Fixed	Random	Random	-58.34	7.80
Random	Random	Random	Fixed	-48.75	7.26
Random	Random	Random	Random	-50.13	9.78

From the results in Table VII, we can derive the following key insights:

- **Model: fixed goal — Tested in: fixed goal** The model achieved a mean reward of -51.56 with a standard deviation of 8.45. This indicates that the model adapted to navigating dynamic obstacles but relied on the goal being in a fixed location
- **Model: fixed goal — Tested in: randomized goal** When the same model was tested in an environment where both obstacles and the goal were randomized, its performance dropped to a mean reward of -58.34 with a slightly lower variance of 7.80. This suggests that introducing goal randomness added an additional layer of complexity that the model was not fully prepared for.
- **Model: randomized goal — Tested in: fixed goal** The model performed slightly better than the previous one in this setting, achieving a mean reward of -48.75 with lower variance of 7.26. This implies that training with a moving goal helped the model develop a more flexible strategy, making it more adaptable even when the goal remained fixed.
- **Model: randomized goal — Tested in: randomized goal** When tested in the fully randomized environment, the fully randomized model achieved a mean reward of -50.13 with the highest variance of 9.78 among the evaluated setups. This suggests that while it developed a general strategy for navigating unpredictable conditions, the added complexity of both obstacle and goal variability made the task more challenging, leading to slightly worse overall performance.

These results highlight a key trade-off between specialization and adaptability:

- Models trained with a fixed goal perform well in environments where the goal remains constant but struggle when the goal position varies.
- Models trained with a randomized goal tend to generalize better to different test environments, particularly when transitioning from a dynamic goal to a fixed goal.
- The highest variance is observed in the most unpredictable environment (random obstacles and random goals), confirming that increasing randomness increases task difficulty.

Overall, training in environments with both obstacle and goal randomness leads to more adaptable policies, but at the cost of slightly lower performance compared to models trained in more structured settings.

VI. Analysis of Density Variations

In this section, we analyze the impact of varying the density of both the objects in the environment and the robotic arm on the model's performance. Specifically, we test the model trained with randomized obstacles and a fixed goal under different density conditions.

A. Results of Density Variations

Our experiments involved modifying the density of objects and the robotic forearm by several orders of magnitude. The key observations are as follows:

- **Object Density:** Increasing the mass of objects up to 10^5 times the original values had no significant impact on performance. The mean reward and standard deviation remained nearly unchanged, indicating that the model could handle such variations without notable difficulty. This could be attributed to the fact that the default Pusher environment operates with relatively low baseline densities, and the physics engine may not fully capture the effects of high-mass interactions.
- **Forearm Density:** The default density of the robotic forearm is set to 300. We tested different density values and observed that minor changes (e.g., increasing to 310 and 400) did not significantly alter performance. However, when increasing the density to 1000, a slight degradation in the mean reward was noted. A substantial impact was observed at a density of 3000, where the mean reward dropped to approximately -64, indicating a noticeable reduction in the agent's ability to complete the task.

B. Performance Across Different Densities

Table VIII and Table IX summarize the results obtained for different object and forearm densities.

TABLE VIII
PERFORMANCE VARIATION WITH DIFFERENT OBJECT DENSITIES.

Object Density	Mean Reward	Std Reward
10^{-4}	-51.56	8.45
10^{-1}	-51.19	6.49
10^{1}	-50.70	6.27

TABLE IX
PERFORMANCE VARIATION WITH DIFFERENT FOREARM DENSITIES.

Forearm Density	Mean Reward	Std Reward
300 (Default)	-51.56	8.45
310	-51.38	7.43
400	-51.61	6.05
1000	-54.81	6.57
3000	-64.98	3.53

The results indicate that varying object density up to an order of magnitude of 10^5 does not significantly impact task performance. This may be due to the nature of the Pusher environment, where interactions are primarily limited to controlled pushing motions rather than high-inertia movements. The physics engine may also compensate for mass variations, preventing significant changes in dynamics.

On the other hand, increasing the density of the robotic forearm does influence performance, but only when the increase is substantial (e.g., density 3000). This suggests that beyond a certain threshold, increased mass affects the agent's ability to generate effective control actions. Given that such extreme density variations are unrealistic in real-world robotic applications, we decided not to pursue further investigations in this direction.

VII. CONCLUSIONS

Our research provides several important insights into the application of reinforcement learning in robotic manipulation tasks. We highlighted that SAC achieves superior performance compared to PPO in the Pusher environment, requiring fewer training timesteps while maintaining better mean rewards. This suggests that off-policy algorithms with entropy regularization may be more suitable for continuous control tasks in robotics. The investigation of environmental variations revealed that training with randomized obstacles and goals leads to more robust and adaptable policies, albeit with a slight performance trade-off compared to specialized policies trained in fixed environments. This finding has important implications for sim-to-real transfer, suggesting that incorporating environmental variability during training can enhance policy generalization. Also, these findings contribute to the broader understanding of reinforcement learning in robotics and provide practical insights for developing more robust and adaptable robotic manipulation systems.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Second Edition)*.

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

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

[4] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, M. Mozian, ... and M. White, "Perspectives on sim2real transfer for robotics: A summary of the R: SS 2020 workshop," 2020.

[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," arXiv, Mar. 20, 2017.

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

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

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

[9] Farama Foundation, "Pusher - Gymnasium Documentation," Available: https://gymnasium.farama.org/environments/mujoco/pusher/.

[10] Y. Liu, K. L. Man, T. R. Payne, and Y. Yue, "Evaluating and Selecting Deep Reinforcement Learning Models for Optimal Dynamic Pricing: A Systematic Comparison of PPO, DDPG, and SAC," Jan. 2024, doi: 10.1145/3640824.3640871.

[11] Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations https://stable-baselines3.readthedocs.io/en/master/

[12] MuJoco Documentation https://mujoco.readthedocs.io/en/stable/overview.html