Modular Robotic Reinforcement Learning Platform for Object Manipulation

Vishnudev Kurumbaparambil, Subashkumar Rajanayagam and Stefan Twieg
Department of Electrical, Mechanical and Industrial Engineering, Anhalt University of Applied Sciences, Bernburger Str. 55, 06366 Köthen, Germany
vishnudev.kurumbaparambil@student.hs-anhalt.de, subashkumar.rajanayagam@hs-anhalt.de, stefan.twieg@hs-anhalt.de

Keywords: Reinforcement Learning, Robot Arm, Object Manipulation, OpenAI Gym, Stable Baselines, Robot Operating System.

Abstract: The field of robotics and autonomous systems has witnessed significant advancements in recent years, with an increasing focus on enhancing the capabilities of robotic agents through RL. This project centres around applying Reinforcement Learning techniques to do object manipulation tasks with a specific focus on making a robot arm reach a target object. Using a combination of Gazebo and Robot Operating System (ROS) environments, the robot arm is trained using custom OpenAI Gym environments to simulate the task. The primary objective involves positioning the end-effector of the robot arm close to a designated object and overcoming challenges such as self-collisions during movements. Various iterations of RL training, including different reward logics, curriculum learning approaches, and fine-tuning parameters, are explored to refine the decision-making capabilities of the agent. The training process of Curriculum Learning involves a phased approach, starting with basic movements and progressing to more complex tasks, demonstrating improved performance. However, challenges such as prolonged training times and uncertainties in arm behaviour persist. The project highlights the complexities inherent in designing effective RL strategies for robotic systems and stresses the need for further research to enhance computational efficiency and reliability in real-world applications.

1 INTRODUCTION

1.1 Motivation

Over the years, machines have played a significant role in reducing human workload. With technological advancements like artificial intelligence, machines have become more efficient and intelligent. Exploring the broad spectrum of applications, robots, especially robotic arms, hold immense potential in transforming various industries. This project aims to leverage Reinforcement Learning (RL) fundamentally based on [1], a machine learning technique facilitating experiential learning for robots, to enhance object manipulation capabilities, unlocking diverse possibilities for applying robotic arms across industries.

1.2 Problem Statement

The effectiveness of robotic systems often relies on their ability to perform tasks with precision [2]. This project aims to teach a robot arm to reach for an object using RL. The primary objective involves positioning the end-effector of the robot arm close to a designated object and overcoming challenges such as self-collisions during movements. The idea is to start with a simpler goal, like reaching the object, and later move to more complex tasks, like grabbing. The model's performance can be initially refined through simulation. In the future, this can be applied to a real robot arm. Continuous adjustments and fine-tuning based on simulation outcomes are crucial for achieving the best results in practical situations. This approach leads to a robot system that can apply knowledge because it already learned how to do it (in a virtual environment/ OpenAI Gym) [3].

2 LITERATURE REVIEW

2.1 Reinforcement Learning

In simple terms, RL is the study of agents and how they learn from their interaction with the environment to fulfil desired goals by trial and error. As shown in Figure 1, based on the action (a_t) and reward (r_t) hypothesis, calculated by comparing the desired goal and observed state (S_t), facilitates learning the desired outcome. The Environment is the world where the agent resides and interacts. After each interaction, the agent gets information about the environment's new State. Based on this new state, it decides its next step. A Reward is a signal that an agent gets from the environment which tells how good or bad the state is and successful the goal is [4]. The goal of the agent is to maximize these rewards called "return". So, using RL methods the agent can learn to achieve its goal [5]. In our case, the robot arm (Agent) will decide on its next best joint angles (Action a_t) to reach a desired end effector location based on Reward r_t, which is calculated using the distance between the previous end effector location (State S_t) and the desired location. This is repeated until a satisfactory level of rewards is achieved which corresponds to a high success rate or accuracy of the robot arm reaching the desired end effector location.

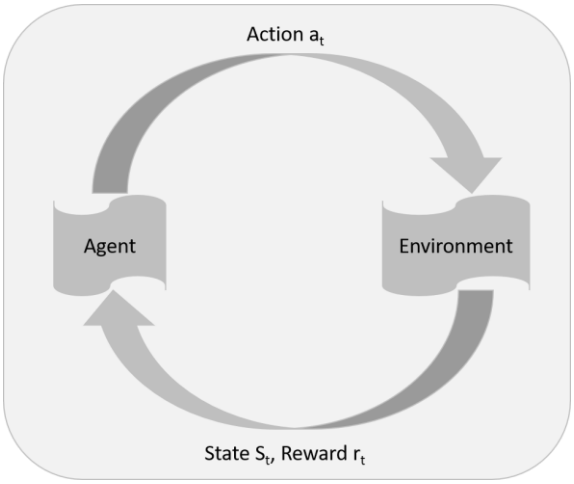

Figure 1: RL Agent-Environment interaction loop.

Mapping the states to the actions are called policies. These policies optimize themselves by choosing actions that maximize the rewards. These policies are categorized as value, policy, and model-based algorithms. In value-based algorithms such as Deep Q-Networks (DQN), policies are derived indirectly by learning the value function [6]. In policy-based algorithms such as Proximal Policy Optimization (PPO), the policies are directly improved [7]. While model-based algorithms such as Dyna-Q focus on learning the model of the environmental dynamics and thereby being able to learn policy functions [8].

2.2 Robot Arm Manipulation

A few research studies have been reviewed. It gave insight into different reinforcement methods. Open-World Object Manipulation using Pre-Trained Vision-Language Models [9]: The researchers utilize pre-trained vision-language models to extract object-related details from the textual instructions and images. By leveraging this information, the robot becomes capable of identifying unknown objects and performing the required task, such as picking up the object. A Survey on Deep RL Algorithms for Robotic Manipulation [10]: This review provides an overview of recent advancements in deep RL for robotic manipulation tasks. Review papers [11-16] provided applications in environmental perception, path planning, behaviour decision and path planning. This provided a good idea of the potential of RL when applied appropriately specific to the domain. In addition, it provided a good overview of the available state-of-the-art RL algorithms.

2.3 Limitation

Implementation based on RL in robotics is mostly implemented on simulation and then transferred to the real hardware via transfer learning. In simulation, the environment can be reset easily, but on real hardware, human intervention is necessary to reset the environment [17]. Utilizing RL algorithms also has the potential to learn much faster and more efficiently in solving real-world problems [18, 19]

However, to address these limitations, it is important to develop a robotics RL platform that provides flexibility and appropriate evaluation criteria, with which integration and experimentation with available state-of-the-art or new novel algorithms with various robots can be done. During the literature review, we only identified platforms that are general to reinforcement learning or specific to certain robots.

3 DEVELOPMENT PLATFORM

After the literature review, it became evident that the ROS is the primary choice for robot control. When it comes to RL, OpenAI Gym [20] is a widely adopted framework. However, OpenAI Gym is designed for general RL and lacks specific features required for robotics applications. Bridging the gap between these two systems posed a significant challenge. The final goal is to combine all these, creating a stable training environment for our RL agent.

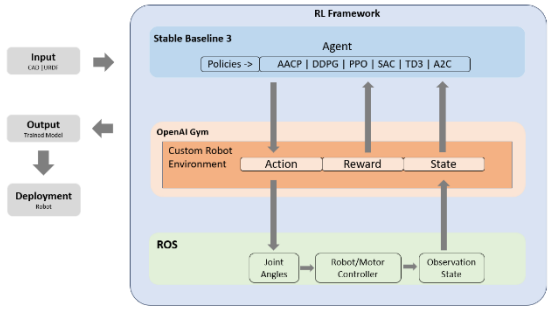

Figure 2: Platform architecture for robotics.

Figure 2, shows the overview and interconnection of the developed robotic RL platform.

3.1 Integrating OpenAI Gym and ROS

The initial step involves the comprehensive analysis of translating the problem statement into Python code while considering the integration of Gym and ROS. The logical step is to develop a custom environment class using OpenAI Gym's APIs for path planning, employing RL to guide the robot arm to desired locations. Within this environment, integrate ROS to manage the arm's initialization, control, and feedback acquisition. The computation of rewards is based on the feedback generated by various ROS nodes, which are continually publishing data. The core framework for the reward system, data gathering from sensors, and state management are encapsulated within the step function of this custom OpenAI Gym environment. Additionally, the procedures to be executed at the end of each episode, such as resetting variables and the robot's state, are delineated in the reset function of the custom OpenAI Gym environment.

Integrating OpenAI Gym and ROS provides a robust solution for robot arm manipulation, combining their strengths to develop and train RL agents for precise and effective movements.

3.2 Designing a Custom OpenAI Gym Environment

OpenAI Gym package provides templates for creating a custom environment which should mirror the real-world application. Understanding the real world provides insights into the factors influencing the model and this can define how the agent can predict the actions that can be taken by the model. Then, we need to calculate rewards based on how the environment changes due to the agent's actions. ROS APIs can be used to read the environment state like the robot's joint positions. The reward calculation is crucial because it tells the agent if it's actions are good or bad. The reward system inherently steers the learning process of the agent, shaping its understanding of the optimal course of action in pursuit of the defined objectives.

To create a custom OpenAI Gym environment, we need to define several key functions required by the class template. These functions include:

1) **__init__**: This function initializes the environment and sets up any necessary parameters.
2) **step**: It defines what happens during each step in the environment. It calculates the next state, reward, and whether the episode is done.
3) **reset**: This function resets the environment to its initial state at the beginning of each episode.
4) **render**: If necessary, this function can be used to visualize the environment.

3.3 Robot Arm Models

During the initial phase, a simple robot arm model was custom-designed using the Unified Robot Description Format (URDF) [21]. This model consisted of three cylinders stacked on top of each other, with two movable joints of type 'revolute'. In this early stage, our primary goal was to control these joints using ROS from Python, and subsequently and subsequently integrating it with OpenAI Gym to utilize RL.

The initial ROS and OpenAI Gym framework was established using this basic three-cylinder model shown on the left side of Figure 3. A CAD model can be converted to URDF format using the "sw_urdf_exporter" plugin in SolidWorks. Our robot arm features five revolute joints for arm movement and two prismatic joints for the end effector, as shown on the right side of Figure 3.

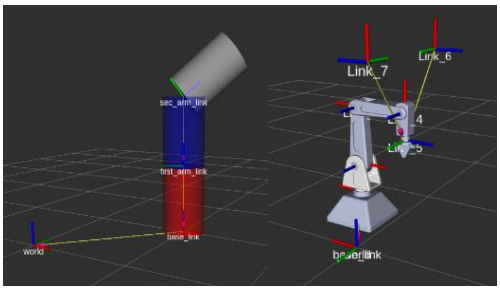

Figure 3: Initial simple cylindrical model (left) and CAD to URDF model (right).

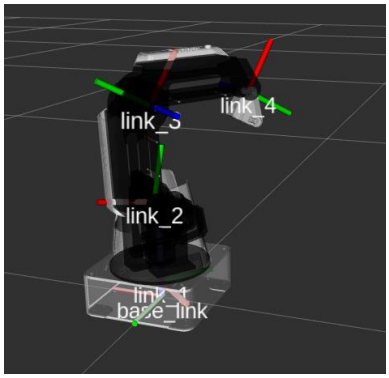

Figure 4: Dobot Magician robot arm.

The next step was get insights into the performance difference between the simulation and the real world. For this purpose, we chose Dobot Magician robotic arm available at our laboratory and its URDF model was obtained from the manufacturer's website as shown in Figure 4. This URDF model would be used for further training and experimentation. The model featured four revolute joints for arm movement, with no end effector. One notable issue encountered with the URDF of the Dobot Magician is its lack of self-collision constraints, that's why we implemented "selfCollision" tags and a contact sensor, working together to detect self-collisions.

4 TRAINING OF THE RL AGENT

During the training process, we aim to refine the RL Agent's performance. This involves adjusting simulations and enhancing the reward logic, all to improve the agent's decision-making capabilities.

4.1 Varying Simulation Parameters

In our simulation setup, the parameters `<max_step_size>`, `<real_time_factor>`, and `<real_time_update_rate>` in the <physics> tag of world file play crucial roles in controlling the dynamics of the simulation. The `<max_step_size>` determines the maximum time step for simulation updates, influencing the trade-off between accuracy and speed. The `<real_time_factor>` sets the ratio of simulation time to real-world time, allowing for control over the simulation's temporal behaviour. Finally, `<real_time_update_rate>` dictates how frequently the simulation updates in real-time, influencing responsiveness and computational load. Fine-tuning these parameters is essential to achieve the desired balance between simulation fidelity and computational efficiency in our experimentation.

The optimal simulation rate parameters are found out to be 0.01 for MSS and 1000 for real_time_update_rate.

4.2 Varying Environment Parameters

4.2.1 Scaling Episode Iteration Steps

For each training episode, the maximum number of tries or episode length can be configured.

Table 1: Training results after varying the Episode Length.

Episode Length	TT (s)	C/E/BF	AS/ AU	ELM	ERM	GR
50	1860	3337/0/103	930/5415	44.3	-901	0
75	1859	4222/0/111	724/4776	51.3	-1440	0
100	2082	5072/0/134	646/4001	48.5	-1750	1

Table 1 consists of TT (Time Taken), Collisions/Errors/Bottle Fell (C/E/BF), Episode Length Mean (ELM), Episode Reward Mean (ERM) and Goal Reached (GR) of the experiment. Based on the obtained results, setting the episode length to no more than 50 yields better outcomes in terms of performance and stability. This is seen in Table 1 at the BF for the Episode Length of 50.

4.2.2 Observation Space

Exploring an alternative approach is to expand the observation space by adding the positions of both the object and the robot arm end effector. While it remains experimental, this adjustment aims to evaluate the impact on the learning process and determine if a more comprehensive observation space enhances the agent's understanding of the environment.

Table 2: Training results with different Observation Space (OS). D – Distance, OP – Object Distance and GP – Gripper Distance.

OS	C/E/BF	AS/ AU	ERM	GR
D	3337/0/103	930/5415	-901	0
D + OP	2398/0/94	934/6375	-818	1
D + OP + GP	2787/0/111	953/5942	-876	2

Table 2 consists of Collisions/Errors/Bottle Fell (C/E/BF), Action Success/ Action Unsuccess (AS/AU), Episode Reward Mean (ERM), and Goal Reached (GR) of this experiment. As can be seen in above Table 2, the goal is reached when the OS is more comprehensive and hence is the better approach.

4.3 Varying Reward Logic

4.3.1 Reward Logic v1

In defining the reward strategy, the central objective is to guide the robot arm toward the object while steering clear of collisions. The proposed approach involves assigning an exponentially increasing penalty for consecutive collisions, focusing on the impact of repeated impacts. Conversely, as the robot arm approaches the object, the reward scales proportionally, underlining the importance of proximity. To provide a stronger incentive, the reward for nearing the object is intentionally set slightly higher. Importantly, a distinct reward is introduced for transitioning from a collision state to a favourable movement state, emphasizing the priority of avoiding collisions over mere distance alterations. The reward system guides the agent to minimize the distance to the object while avoiding collisions effectively. Adjusting these reward components will be crucial for shaping the agent's behaviour for optimal task performance. Table 3 below illustrates the results after training, where the Iteration Per Episode (IPE) is assigned in the first column.

Table 3: Training results of the Reward Logic v1.

IPE	TT (s)	C/E (%)	AS/ AU	ELM	ERM	GR
50k	8953	0/2.45	11.93/5 2.29%	28.8	-414	66
50k	8635	0/2.21	11.57/5 5.86%	20.4	-89600	117

The obtained results fell short of perfection, revealing a notable issue in the potential for rewards to exhibit significant fluctuations, particularly with very high negative values attributed to penalties for self-collisions. The learning process of the agent lacks stability, evident in the mean reward that varies sporadically between negative hundreds and occasional spikes into the thousands.

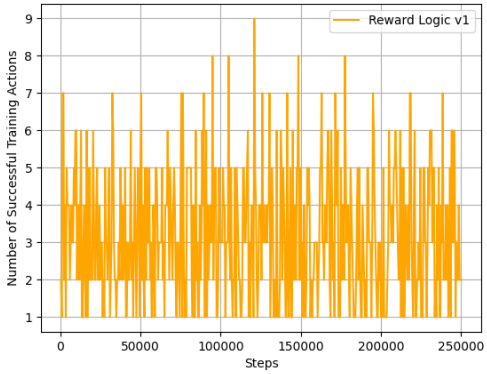

Figure 5: Number of Successful Training Actions with Reward Logic v1.

Furthermore, the number of actions demonstrating improvement has not shown any significant enhancement as seen in Figure 5. Looking at the average reward in Figure 6, we see that it often goes up and down a lot. This is mainly because of the harsh penalty for consecutive collisions. While this penalty is needed, we should adjust the rewards to avoid extremely negative values. This should help stabilize the learning process.

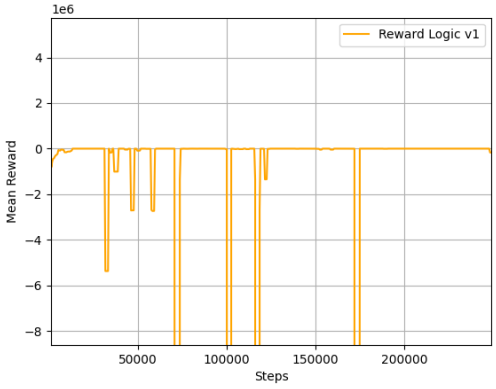

Figure 6: Reward Mean with Reward Logic v1.

4.3.2 Reward Logic v2

The updated logic adopts a human-centric approach, aligning with how a person would be motivated and penalized in reaching the object. This strategy aims to

create a more realistic and balanced learning experience. The key considerations for this version reward system are as follows:

1) Max Steps and Object Reach Reward: The highest reward should occur when the robot reaches the object in the fewest steps, with diminishing rewards as the step count increases. Even if the maximum steps are taken, a positive but reduced reward should encourage continued attempts.
2) Collision Penalties: The penalty for collisions should be set in a way that, while giving higher penalties for discouraging continuous collisions, it does not plunge into extremely negative values. This ensures that the agent perceives the task as challenging but not impossible.
3) Safe Movement Reward: Transitioning from a collision to a safe movement should be positively rewarded. However, this reward should not be too high to discourage frequent switching between collision and safe movements.
4) Distance-Based Rewards and Penalties: Penalties for moving away from the object and rewards for proximity should be balanced to motivate the agent without overly emphasizing proximity.
5) Balanced Reward Values: Reward values for positive and negative actions must be finely tuned to motivate goal achievement while discouraging collisions and moving away from the object. Striking a balance ensures that the agent remains engaged and does not lean towards inactivity due to disproportionately high penalties.

This approach mirrors human motivation and helps stabilize the learning process for the agent. Below Table 4 shows the comparison among versions.

Table 4: Training results with Reward Logic v2.

Version	TT (s)	C/E/BF	AS/AU	ELM	GR
v1	8953	15465/0/1223	5965/26143	28.8	66
v2	10319	17807/0/2004	6907/22086	10.6	129

Upon comparing the plots of the two versions in Figure 7, a significant improvement is observed in the number of successful episodes where the robot arm successfully reached the target.

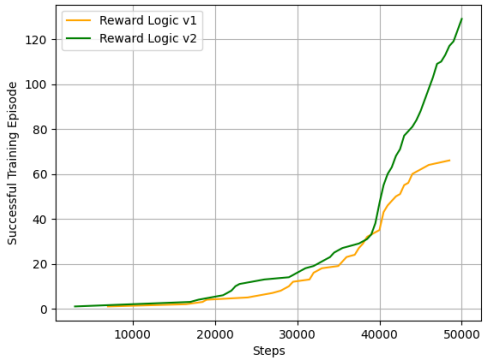

Figure 7: Successful Training Episodes with Reward Logic v2.

Other parameters such as the number of collisions and successful training steps don't show a significant improvement in the observed results as seen in Figure 8 below.

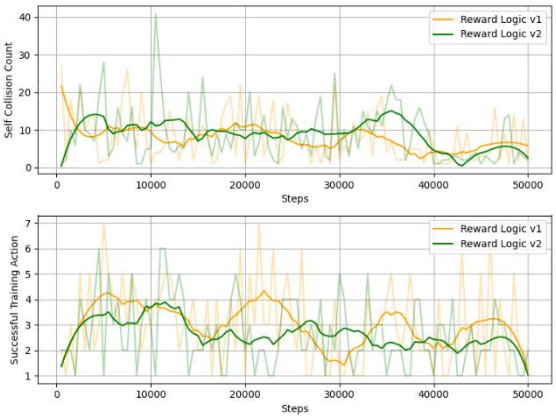

Figure 8: Reward Logic v2 indicating no improvement in Collisions and Successful Training Action.

As seen in Figure 9, the rewards appear to be within an acceptable range, steadily increasing with no irregular negative peaks.

Figure 9: Reward Mean with the Reward Logic v2.

To gain further insights, we conducted testing for 250,000 steps in both versions, and the results are as follows in Table 5.

Table 5: Training results with Reward Logic v2 (250k IPE steps).

Ver	TT (s)	C/E/BF (%)	AS/AU (%)	ELM	GR
v1	44111	20.78/0/3.01	12.99/59.03	-168000	1003
v2	47154	34.69/0/2.91	12.53/46.69	-11.2	497

Surprisingly, the improvement in the number of successful episodes is not as significant as in version 1. Additionally, the number of collisions continues to increase. Analyzing the mean reward plot in Figure 10 reveals prominent negative peaks at multiple points. This problem emerged when the bottle was thrown far away, leading to an excessively high distance calculation and subsequently resulting in a very negative reward based on distance.

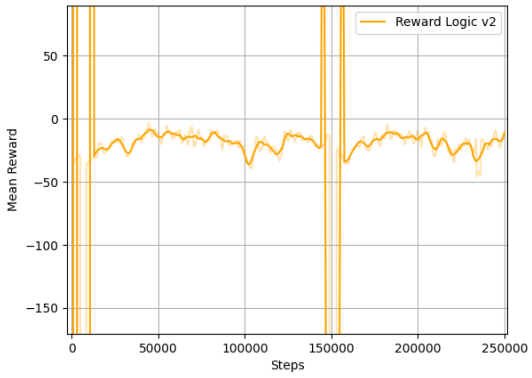

Figure 10: Reward Mean with Reward Logic v2 after running for 250k IPE steps.

4.3.3 Reward Logic v3

To address the excessive negative values of reward in v2, extra safeguards have been incorporated in this version. If collisions persist beyond a certain threshold, the training is terminated with a penalty. Additionally, a check has been implemented to ensure the bottle's proximity, accounting for scenarios of potential displacement due to rapid arm movements.

Upon analyzing Table 6, a decline in performance is seen. The current threshold for successive collisions is set at 5, potentially terminating many training episodes prematurely and limiting the agent's learning. In the next training iteration (version 3.1), the self-collision threshold will be increased to 10. However, there is no apparent improvement, and the results even show a slight negative trend. The mean reward exhibits no occurrence of negative peaks, suggesting the success of safeguard mechanisms.

Table 6: Training results with Reward Logic v3.

Version	TT (s)	C/E/BF	AS/AU	ELM	GR
v1	8953	15465/0/1223	5965/26143	28.8	66
v2	10319	17807/0/2004	6907/22086	10.6	129
v3	9298	17376/0/746	6004/24022	26.6	38
v3.1	10472	19693/0/751	5563/22697	33.5	13

4.3.4 Reward Logic v4

Despite numerous iterations, most versions of the reward logic performed poorly compared to the initial one. To enhance the reward system, reward shaping was considered, in which rewards are based on the quality of action by evaluating deviations from ideal actions. Consequently, penalties are applied, providing the agent with feedback and influencing its behaviour. This strategy aims to guide the agent towards actions that align more closely with the desired outcomes, potentially accelerating the learning process and improving overall task performance. The collision threshold also has been updated to 25. The shaping reward based on the action is implemented specifically whenever the valid distance measured is not improved. The training has been executed for 50,000 steps, and the outcomes are presented in Table 7 below. The successful training episode is also shown in Figure 11.

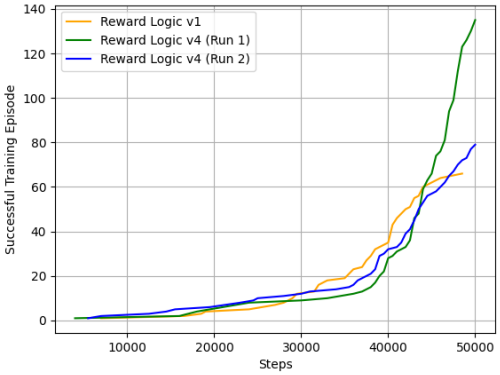

Figure 11: Successful Training Episode with the Reward Logic v4 (50k IPE steps).

There is also a reduction in the number of collisions, and successful actions show a slight improvement as seen in Figure 12.

Table 7: Training results with Reward Logic v4 after executing for 50k steps.

Version	TT (s)	C/E/BF	AS/AU	ELM	GR
v1	8953	15465/0/1223	5965/26143	28.8	66
v2	10319	17807/0/2004	6907/22086	10.6	129
v3	10472	19693/0/751	5563/22697	33.5	13
v4	9301	12640/0/1583	6782/27941	-	135
v4	8799	12967/0/1238	6266/28488	-192	79

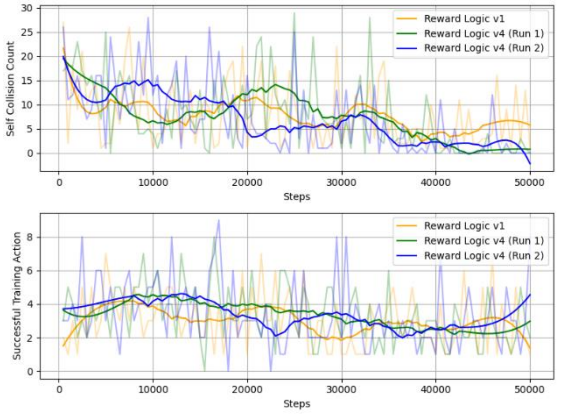

Figure 12: Plots of Self-Collision Count and Successful Training Action with Reward Logic v4.

Table 8: Training results of Training with Reward Logic v4 after running 250k IPE steps. Ver stands for Version.

Ver	TT (s)	C/E/BF (%)	AS/AU (%)	GR
v1	44111	20.78/0.0012/3.0	12.99/59.03	1003
v4	45692	21.92/0/3.25%	14.11/58.07	1069

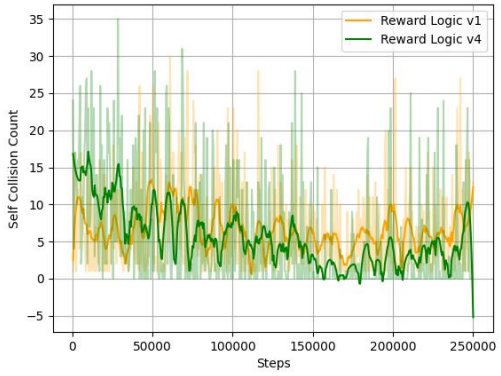

Figure 13: Self-Collision Count with Reward Logic v4 running for 250k IPE steps.

To delve deeper, additional testing was conducted for 250,000 steps and summarized in Table 8. A noticeable improvement in the number of successful episodes is evident, and there is a decreasing trend in the number of collisions toward the end of training as seen in Figure 13. Overall, this version exhibits slightly better performance than the initial reward logic and will be used for further testing.

4.4 Varying Policies

Stable Baselines provides a diverse range of RL policies such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradients (DDPG), Trust Region Policy Optimization (TRPO), and more. The applicability of each policy depends on the action and observation types specific to the environment. The possible policies for the current task are tried. These are Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3).

4.4.1 Advantage Actor-Critic Policy

Up to this point, all training sessions have been conducted using this A2C policy. The results are summarized below in Table 9.

Table 9: Training results with A2C Policy.

TT (s)	C/E/BF (%)	AS/AU (%)	GR
8799 (50k steps)	25.93/0/2.48	12.53/56.98	79
45692 (250k steps)	21.92/0/3.25	14.11/58.07	1069

4.4.2 Policy Comparison

Results obtained with DDPG, PPO and SAC policies highlight a much lower number of successful episodes as can be seen in Table 10. This is also evident in Figure 14 when DDPG is compared with A2C.

Table 10: Training results with PPO Policy.

RL	TT (s)	C/E/BF (%)	AS/AU (%)	GR
DDPG	8995	16.97/0/1.73	11.48/67.74	16
PPO	8697	24.63/0/0.87	9.58/62.80	5
SAC	11884	15.51/0/1.09	11.26/67.44	4

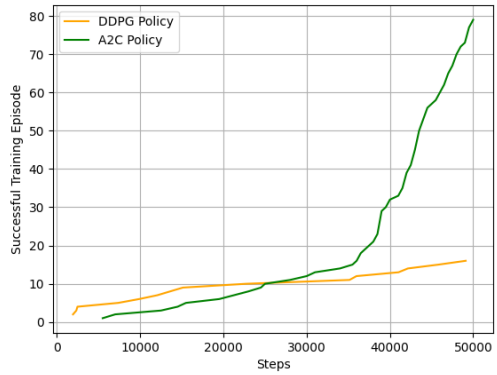

Figure 14: Comparison of Successful Training Episode between A2C and DDPG Policies.

4.4.3 Comparison of TD3 and A2C Policy

The TD3 policy has demonstrated a notable number of successful actions and achieved goals as shown in Table 11.

Table 11: Training results with TD3 Policy.

TT (s)	C/E/BF	AS/AU	GR
10274 (50k steps)	8476/0/1860 16.95/0/3.72%	7083/31565 14.17/63.13%	45
50951 (250k steps)	43062/0/10308 17.22/0/4.12%	45894/144635 18.36/57.86%	785

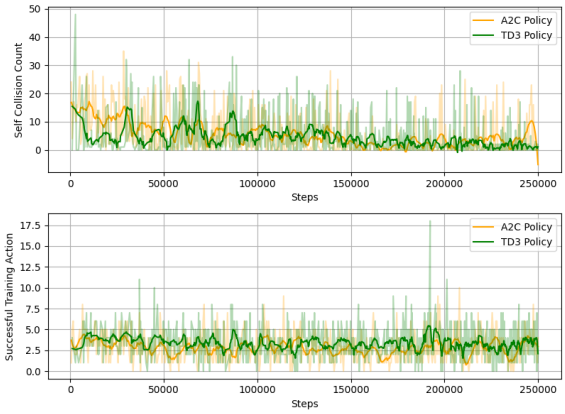

Figure 15: Plot of Self-Collisions Count and Successful Training Actions with A2C and TD3 Policies.

To further assess its performance, it was executed for 250,000 steps, and the results are detailed in Table 11. Upon analyzing the plots, it is evident that the TD3 Policy exhibits fewer collisions compared to the A2C policy. Additionally, the number of successful actions is higher as seen in Figure 15. Although the A2C policy achieves more goals, the TD3 policy shows an exponential increase in successful goals towards the end of training, indicating a potential for superior performance as shown in Figure 16.

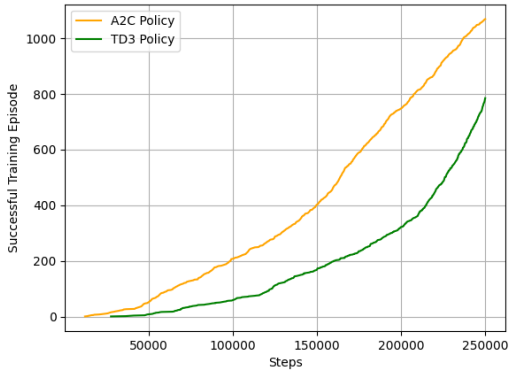

Figure 16: Plot of Successful Training Episodes with A2C and TD3 Policies.

Based on these findings, the decision has been made to proceed with the TD3 policy for subsequent testing.

4.5 Curriculum Learning

Curriculum learning is a training strategy in machine learning where the model is exposed to a series of tasks or goals of increasing complexity. The approach gradually introduces more challenging scenarios, mirroring human learning by starting with simpler concepts before advancing to more intricate ones. Curriculum learning can enhance the overall learning efficiency and generalization capabilities of a model, enabling it to better handle a diverse range of situations. An initial experiment involved breaking down the task into two stages. In the first stage, the robot was trained to reach a specific point instead of the bottle, overlooking the scenario where the bottle might fall. Following this, the model was loaded and retrained with the comprehensive goal of reaching the bottle.

In Figure 17 above, it is evident that the process of retraining the model is functioning effectively. The loaded model retains and builds upon previous training knowledge, resulting in improved performance during subsequent training sessions. Given the objective of having the robot arm reach the bottle, we encounter two primary challenges: collisions between the arm and the risk of the bottle toppling. To systematically address these challenges at different stages, the task is partitioned into three distinct goals, each trained separately:

- The primary objective is to guide the robot arm to move without collisions. Penalties are given exclusively for collisions, while all valid movements receive rewards, with greater weight assigned to getting closer to the fixed point.
- In this stage, additional penalties are introduced for movements that take the robot farther away from the fixed point. There is no condition check for the possibility of the bottle falling.
- The final stage involves the robot attempting to reach the bottle. Extra penalties are imposed if the bottle falls during this stage.

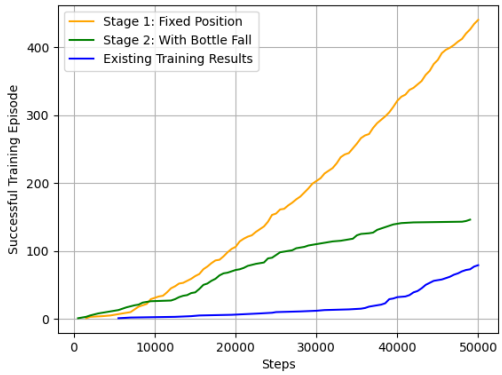

Figure 17: Training the agent in two stages.

Figures 18 and 19 below illustrate the successful episodes and self-collisions. Each stage undergoes training for a duration of 100,000 steps.

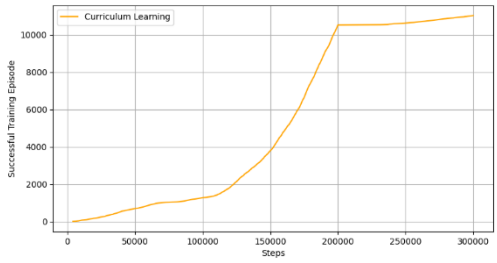

Figure 18: Plot of Successful Training Episodes with Curriculum Learning.

As seen in Figure 19, the total count of self-collisions has considerably decreased. The stage with the highest number of successful episodes is stage 2, where the objective is to move to a fixed point without considering the possibility of the bottle toppling over while penalizing collisions and movements away from the designated point. However, once it moves to stage 3, the rate of successful episodes is reduced significantly. But the total number of successful episodes is still good as seen in Table 12. Table 12 consists of Collisions/Errors/Bottle Fell (C/E/BF), Action Successful/ Action Unsuccessful (AS/AU) and Goal Reached (GR) of the experiment.

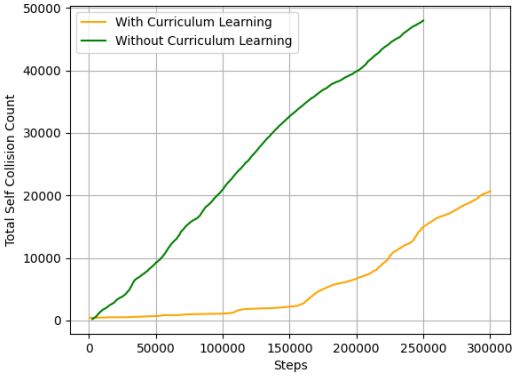

Figure 19: Plot of Total Self-Collision Count with Curriculum Learning.

Table 12: Training results with Curriculum Learning.

Stage	C/E/BF (%)	AS/AU (%)	GR
1	1.03/0/0	10.72/85.32	1275
2	5.42/0/0	26.74/57.69	9253
3	12.23/48.75	16.85/63.33	501

The understanding from this curriculum learning is that the robot can be trained initially for simple goals and then trained on top of this for sophisticated goals.

4.6 Results

In all experiments, total training time emerges as the key factor influencing performance. Regardless of policy and parameters, better results are consistently observed with increased training steps for both A2C and TD3 algorithms. The introduction of curriculum learning indicates potential for enhanced performance. The extended training duration for each stage facilitates better learning by the robot. However, inherent variability in machine learning outcomes presents a challenge to achieving consistent results across trials. The best model obtained through the Curriculum-based approach is evaluated in two scenarios. In the first approach, the final position is fixed, without any condition regarding the bottle getting toppled. In the second approach, the condition of the bottle getting toppled is considered. Two trials are conducted for each approach, and the results are

presented in Tables 13 and 14 below. The observed trend reveals that the robot arm tends to topple the bottle during its attempts to reach it, resulting in a lower success rate. However, when tasked with reaching a fixed point without considering the risk of the bottle falling, the success rate is notably higher.

Table 13: Testing of the best model with a Scenario of fixed position.

Scenario 1: Fixed Position	Count of final states [Episode Counts: 50]		
	BottleFell	End_Success	End_NoSuccess
Trial 1	0	32	18
Trial 2	0	22	28

Table 14: Testing of the best model with a Scenario of reaching a bottle.

Scenario 2: Bottle Position	Count of final states [Episode Counts: 50]		
	BottleFell	End_Success	End_NoSuccess
Trial 1	43	3	4
Trial 2	34	11	5

Moreover, the variation in outcomes between the two trials in both scenarios underscores the dynamic nature of the model's performance across different instances. This highlights the importance of assessing the confidence level of the model for real-life scenarios.

5 LIMITATIONS

Executing the project posed a significant time challenge, with optimal results achieved through extended training steps, but with the trade-off of higher execution time. Training for 50k steps alone took approximately 2.5 hours, while the recommended training times for better performance range from 500k to 3000k steps. Attempts to run for such prolonged durations were also hindered by computational constraints, occasionally resulting in script stalling. The current agent predicts actions within the motion range, occasionally leading to abrupt arm movements, causing objects in its path to topple. An incremental update to actions could mitigate this issue, with the initial action being randomly predicted and subsequent actions featuring incremental changes. This approach aims to prevent sudden, unpredictable movements. Despite extended training durations, the performance of the arm cannot be guaranteed or predicted with absolute confidence. Occasional collisions and abrupt arm movements persist, posing challenges when transferring the learned behaviour to real hardware. Rigorous analysis of results after extended training periods is essential to reaffirm confidence in the model's performance.

6 CONCLUSIONS

This project has explored the application of RL in the domain of object manipulation. As the field of artificial intelligence continues to evolve, the integration of advanced RL techniques in object manipulation will undoubtedly play a pivotal role in shaping the future of automation and robotics.

The successful integration of Gazebo, ROS, and custom OpenAI Gym environments provided an efficient development environment for simulating realistic robotic tasks. The primary goal of positioning the robot arm near the target object involved distance calculation and collision management within the robot arm simulation, utilizing Gazebo plugins and Python scripts. Throughout the training process, we optimized environment parameters and iteratively refined reward logic, transitioning from basic logic to more sophisticated versions, and incorporating reward shaping and tuning approaches. The improvement of the reward system, especially the shift towards a more human-centric approach, highlighted the complexity and challenges inherent in designing effective RL strategies. Implementing curriculum learning, mirroring human learning progression significantly enhanced the model's efficiency. This is a promising approach and better results are expected when executed for more training steps. However, computational limitations were observed, suggesting the need for more powerful devices in future research for enhanced performance. The project deepened our understanding of object manipulation with RL, emphasizing the intricate dynamics in robotic systems.

ACKNOWLEDGMENTS

We acknowledge support by the German Research Foundation (Deutsche Forschungsgemeinschaft DFG) - and the Open Access Publishing Fund of Anhalt University of Applied Sciences.

REFERENCES

[1] R.S. Sutton and A.G. Barto, "Reinforcement Learning: An Introduction," IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 1054-1054, 1998, doi: 10.1109/TNN.1998.712192.

[2] Z. Zhu et al., "High precision and efficiency robotic milling of complex parts: Challenges, approaches and trends," Chinese Journal of Aeronautics, vol. 35, no. 2, pp. 22-46, 2022, doi: 10.1016/j.cja.2020.12.030.

[3] H. Ju et al., "Transferring policy of deep reinforcement learning from simulation to reality for robotics," Nat Mach Intell, vol. 4, pp. 1077-1087, 2022, doi: 10.1038/s42256-022-00573-6.

[4] A. K. Shakya et al., "Reinforcement learning algorithms: A brief survey," Expert Systems with Applications, vol. 231, p. 120495, 2023, doi: 10.1016/j.eswa.2023.120495.

[5] A. Ray et al., "Spinning Up OpenAI," 2018, [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#.

[6] V. Mnih et al., "Human-level control through deep reinforcement learning," 2015.

[7] J. Schulman et al., "Proximal Policy Optimization Algorithms," 2017.

[8] A. Plaat, "Model-Based Reinforcement Learning," 2022.

[9] A. Stone et al., "Open-World Object Manipulation using Pre-trained Vision-Language Models," ArXiv, /abs/2303.00905, 2023.

[10] D. Han et al., "A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation," Sensors, 2023, doi: 10.3390/s23073762.

[11] S. Chen and Y. Li, "Active vision in robotic systems: a survey of recent developments," The International Journal of Robotics Research, vol. 30, no. 11, pp. 1343-1377, 2011, doi: 10.1177/0278364911410755.

[12] R. S. Pol and M. Murugan, "A review on indoor human aware autonomous mobile robot navigation through a dynamic environment: survey of different path planning algorithm and methods," 2015 International Conference on Industrial Instrumentation and Control (ICIC), IEEE, 2015.

[13] M. Foukarakis et al., "Combining finite state machine and decision-making tools for adaptable robot behavior," International conference on universal access in human-computer interaction, Heraklion, Crete, Greece, pp. 625–635, Springer, 2-27 June 2014.

[14] R.-E. Precup and H. Hellendoorn, "A survey on industrial applications of fuzzy control," Computers in Industry, vol. 62, no. 3, pp. 213-226, 2011.

[15] O. Boubaker, "The inverted pendulum benchmark in nonlinear control theory," International Journal of Advanced Robot Systems, vol. 10, no. 233, pp. 1-9, 2013.

[16] L. Sciavicco and B. Siciliano, "Modelling and control of robot manipulators," Springer Science & Business Media, Berlin/Heidelberg, 2012.

[17] A. Sharma et al., "Autonomous Reinforcement Learning: Formalism and Benchmarking," 2021.

[18] H. Nguyen and H. La, "Review of Deep Reinforcement Learning for Robot Manipulation," 2019.

[19] M. Towers et al., "Gymnasium," 2023.

[20] Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System, Noetic," 2018, [Online]. Available: https://www.ros.org.

[21] Ageofrobotics, "urdf_tutorial," 2023. [Online]. Available: https://github.com/ageofrobotics/urdf_tutorial.