

---

---

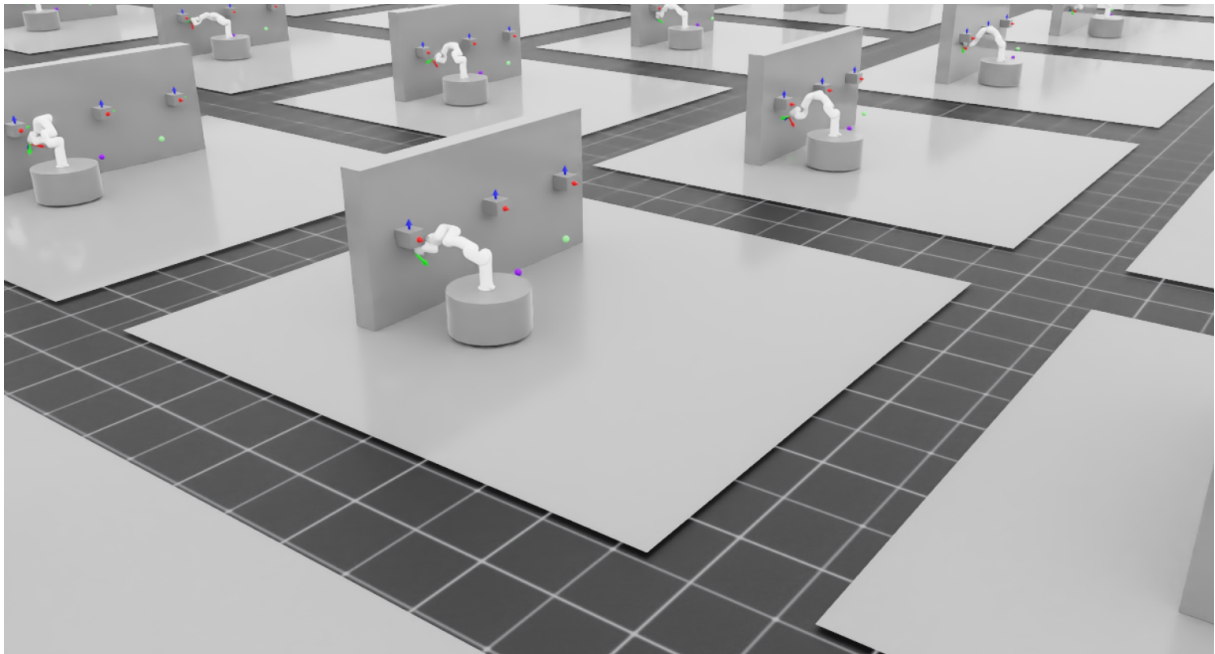
# Master's Thesis

## SKYWALKER

Autonomous Control of a Free-Floating Space Manipulator in  
Simulated Microgravity Using Reinforcement Learning

---

---



Master's Thesis Report Project written by

Dadi Hrannar Davidsson

Bruno Miguelez De Salas

Glenn Gadensgaard Svendsen

Aalborg University  
The Technological Faculty for IT and Design



# AALBORG UNIVERSITY

## STUDENT REPORT

The Technological Faculty for IT and  
Design

Niels Jernes Vej 10, 9220 Aalborg Øst  
<http://www.aau.dk>

**Title:**

Master's thesis - SKYWALKER

**Theme:**

Technological Project Work

**Project Period:**

ROB10 P10 Master's Thesis

**Participant(s):**

Dadi Hrannar Davidsson  
Bruno Miguelez De Salas  
Glenn Gadensgaard Svendsen

**Supervisor(s):**

Simon Bøgh  
Anton Bjørndahl Mortensen

**Copies:** 1

**Page Numbers:** 79

**Date of Completion:**

June 3, 2025

**Abstract:**

This project explores the use of Proximal Policy Optimization (PPO) to enable autonomous control of a robotic manipulator mounted on a free-floating base in a simulated microgravity environment. A simplified setup of ESA's Orbital Robotics Laboratory (ORL) was developed in Isaac Lab, including a robotic arm, fixed grasping points, and a frictionless floor.

The system was trained using curriculum learning and evaluated through structured acceptance tests covering Point-to-Point motion, grasping, base relocation, and multi-step traversal.

Results show that PPO can produce smooth and accurate behaviors without predefined trajectories in early tasks. However, the final traversal task was not solved, highlighting challenges in long-horizon planning. Still, the project demonstrates a functional RL-based control pipeline, a validated simulation framework, and lays the foundation for future deployment on physical platforms. This work provides a proof-of-concept for using deep reinforcement learning to enable autonomous manipulation and movement in simulated space-like conditions.

# Preface

This thesis was completed as the final project in the Master's programme in Robotics at Aalborg University during the spring semester of 2025. The work was carried out in collaboration with the ESA Academy Experiments Programme and supervised by Associate Professor Simon Bøgh and PhD student Anton Bjørndahl Mortensen.

## Acknowledgment

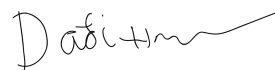
We would like to thank our supervisors, Simon and Anton, for their guidance and support throughout this project. We also wish to thank the ESA Academy Experiments Programme for providing technical feedback and valuable context for the research. Finally, we are grateful to our families, peers, and friends for their continued encouragement during the thesis process.

Aalborg University, June 3, 2025



---

Glenn Gadensgaard Svendsen  
gsvend20@student.aau.dk



---

Dadi Hrannar Davidsson  
ddavid20@student.aau.dk



---

Bruno Miguelez De Salas  
bmigue23@student.aau.dk

# Contents

<b>Acronyms</b>	<b>2</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Initial problem Statement . . . . .	5
<b>2 Problem analysis</b>	<b>6</b>
2.1 Manipulators in Space Sector . . . . .	6
2.2 Orbital Robotics Laboratory (ORL) . . . . .	7
2.3 RL on manipulators . . . . .	9
2.4 Problem Analysis Summary . . . . .	14
<b>3 Problem Formulation</b>	<b>15</b>
3.1 Final problem statement . . . . .	15
3.2 Delimitations . . . . .	15
3.3 Objectives . . . . .	16
3.4 Requirements . . . . .	17
3.5 Acceptance test . . . . .	18
<b>4 Concept design</b>	<b>24</b>
4.1 Physical Setup . . . . .	24
4.2 ROS2 Communication Interface . . . . .	25
4.3 Simulation Setup . . . . .	26
4.4 Weights and Biases Output Visualization . . . . .	32
<b>5 Implementation</b>	<b>33</b>
5.1 Action Space Definition . . . . .	34
5.2 Observation Space Definition . . . . .	34
5.3 Termination Conditions . . . . .	36
5.4 Reward Definition . . . . .	37
5.5 Curriculum Learning . . . . .	46
5.6 Neural Network Architecture in PPO . . . . .	48
5.7 Training Parameters . . . . .	50
<b>6 Testing</b>	<b>52</b>
6.1 Test 1: Point-to-Point Motion Control (0-3500 timesteps) . . . . .	53
6.2 Test 2: Point-to-Point Grasp at Fixed mounting Points (0-8000 timesteps) . . . . .	55
6.3 Test 3: Base Movement After Grasp (5000-15000 timesteps) . . . . .	56
6.4 Test 4: Multi-Point Grasp-Move Sequence (15000-60000 timesteps) . . . . .	58
6.5 Training Metrics . . . . .	62
6.6 Test conclusion . . . . .	65
<b>7 Discussion</b>	<b>67</b>
<b>8 Conclusion</b>	<b>69</b>





## Preface

---

8.1 Future Work . . . . .	70
<b>A Appendix</b>	<b>72</b>
A.1 links . . . . .	72
A.2 Full requirements . . . . .	72
<b>Bibliography</b>	<b>76</b>

# Acronyms

<b>AAU</b>	Aalborg University. 4, 5, 6, 8, 9, 14, 16, 24, 27, 71
<b>DDPG</b>	Deep Deterministic Policy Gradient. 10, 11, 12, 13
<b>DOF</b>	Degree Of Freedom. 10, 11, 12, 18, 24
<b>DRL</b>	Deep Reinforcement Learning. 9, 10, 31, 48
<b>ELU</b>	Exponential Linear Unit. 49
<b>ERA</b>	European Robotic Arm. 6
<b>ESA</b>	European Space Agency. 3, 4, 5, 6, 7, 14, 15, 17, 70, 72
<b>ISS</b>	International Space Station. 3, 6
<b>LSS</b>	Large Space Structure. 3, 4
<b>MIRROR</b>	Multi-arm Installation Robot for Readyng ORUS and Reflector. 3, 4, 5, 7, 9, 14, 15, 27, 71
<b>ORL</b>	Orbital Robotics Laboratory. , 4, 5, 7, 8, 16, 24, 27, 29, 71
<b>PPO</b>	Proximal Policy Optimization. , 10, 11, 12, 13, 14, 15, 16, 30, 31, 32, 34, 36, 37, 44, 48, 49, 50, 52, 66, 67, 69, 70
<b>RL</b>	Reinforcement Learning. , 4, 5, 6, 9, 13, 14, 15, 16, 19, 24, 25, 26, 28, 29, 30, 32, 34, 35, 36, 37, 46, 48, 50, 67, 68, 69, 70, 71, 73, 74
<b>ROS2</b>	Robot Operating System. , 24, 25, 71
<b>SAC</b>	Soft Actor Critic. 10, 12, 13, 68
<b>TQC</b>	Truncated Quantile Critics. 10, 12, 13
<b>URDF</b>	Universal Robot Description Format. 25, 27
<b>USD</b>	Universal Scene Description. 25, 27, 28, 34, 71

# 1 Introduction

With advancements in technology over the past decades, space exploration has been growing steadily. The number of space missions is increasing, and they are becoming more ambitious over time. As a result, there is a growing need for Large Space Structure (LSS) to be built in space. This makes assembly and maintenance tasks more difficult and dangerous for human astronauts to carry out on their own. This has resulted in an increase in research and investment in robotic manipulators for assisting astronauts with constructing LSS.

A good example of a LSS with an increasing number of required tasks is the International Space Station (ISS) as seen in Figure 1.1. The ISS is a space platform designed for scientific research and space exploration, built through a collaboration between NASA, European Space Agency (ESA), and other international partners. It hosts astronauts from various countries who work together on experiments in fields such as biology, medicine, physics, and chemistry.[1]



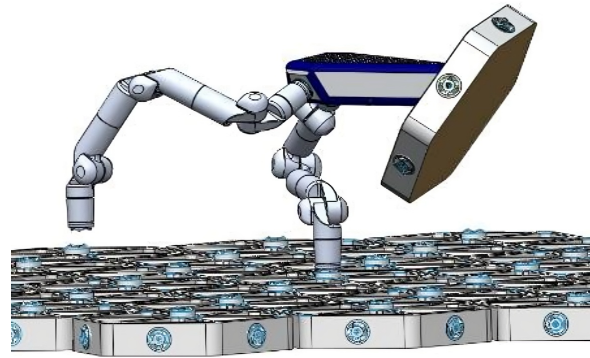
**Figure 1.1:** Visualization of a LSS placed in orbit around the Earth[2].

To support further research operations on the ISS and the construction of future space structures, ESA is supporting the Multi-arm Installation Robot for Readying ORUS and Reflector (MIRROR) project which can be seen in Figure 1.2.

The idea behind the MIRROR Project is to launch non-aerodynamic structures packed inside an aerodynamic rocket. Once the rocket reaches its target location in space, the MIRROR manipulator will then assemble the structure. To move the items to the wanted position, the MIRROR will pick up the item with one of the connectors, and use the others to move around, as visualized in Figure 1.3.



**Figure 1.2:** The MIRROR robot using the HOT-DOCK system to grab two structures.



**Figure 1.3:** Visualization of the MIRROR system walking with an item across a surface[3]

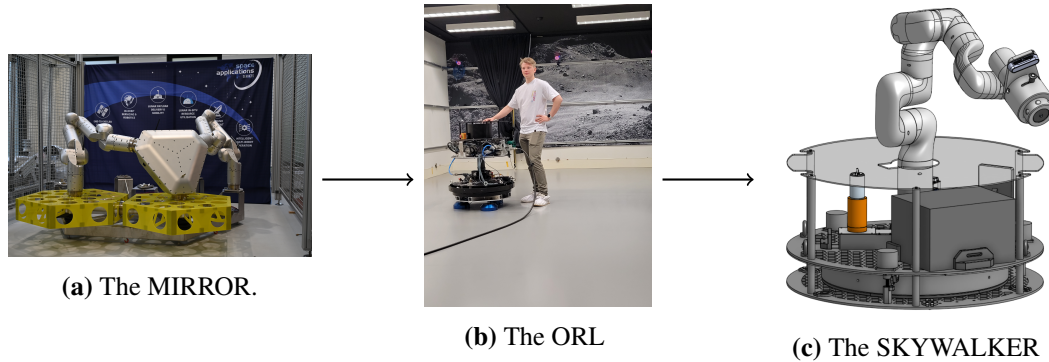
Although MIRROR’s impedance control framework provides robust motion management and contact safety, current capabilities rely heavily on predefined scripts or teleoperation, lacking the autonomous decision-making needed for dynamic adaptation to uncertainties encountered in space environments. RL presents a promising methodology to bridge this autonomy gap, enabling robotic manipulators like MIRROR to autonomously respond to unpredictable scenarios, enhancing both flexibility and efficiency in orbital assembly tasks.

However, applying RL to a space-bound manipulator presents unique challenges. First, ground-based simulations cannot fully replicate zero-gravity conditions, so policies trained on Earth may not translate accurately to orbital operations. Second, safety constraints in space are far stricter, which drastically reduces the feasibility of traditional trial-and-error strategies often used in RL. Finally, model discrepancies between simulation and real hardware can cause the learned policy to fail once deployed: even minor inaccuracies in microgravity or contact-dynamics modelling can lead to unpredictable behaviour on the actual system.

To further develop these initiatives and allow university students to have the opportunity to contribute to ongoing space research ESA has created the ESA Academy’s experiment programme. This project is in collaboration with the ESA Academy’s experiment programme[4] and focuses on exploring the use of RL to control the future robotic manipulators used to construct LSS. To test and validate the use of RL on a physical setup without sending it to space, ESA’s facilities incorporate the Orbital Robotics Laboratory (ORL), a microgravity simulation laboratory included in Figure 1.4b. The setup in the laboratory enables testing of simplified systems under replicated microgravity conditions in a 2D plane. This is achieved using a rigid platform mounted on air bearings, allowing it to levitate a few millimeters above the floor using pressurized air, thereby simulating the absence of gravity in the plane of motion.

Though the final goal of the project is to test the system in ESA’s ORL, a simplified proof-of-concept has been developed and tested at AAU. This local version, known as the SKYWALKER system, is shown in Figure 1.4c. The SKYWALKER system allows for safer and more accessible testing without the risk of damaging the sensitive equipment in the ORL. Figure 1.4 illustrates the step-by-step simplification of the original MIRROR concept, starting from the full-scale mission, through the setup at ESA’s microgravity lab, and ending with the reduced

SKYWALKER system.



**Figure 1.4:** Simplification steps from the MIRROR Project to the ESA ORL and finally the SKYWALKER.

By validating this simplified setup first, the project builds a solid foundation to scale up to more complex environments. This staged approach leads directly into the initial problem statement, where the main challenge of the project is formally introduced.

## 1.1 Initial problem Statement

*”How can Reinforcement Learning be utilized to autonomously control a free-floating system that uses a manipulator to dock and drag itself to navigate on a frictionless floor?”*

This project will be working on a proof-of-concept in using RL to control a manipulator system in space.

The following chapters will start by describing and analyzing the space industry together with the MIRROR project and the current work using RL in space-related projects. This will be followed up with a short section describing ESA’s ORL and a review of the test system at AAU.

After the analysis, the project’s main objectives and system requirements are introduced, along with a series of acceptance tests designed to validate the system’s performance. Based on these requirements, the design and implementation of the proposed solution are then presented.

Finally, the report concludes with a discussion of the test results and a critical evaluation of the proof-of-concept system, reflecting on both its current capabilities and its potential for future development.

## 2 Problem analysis

In this chapter, the analysis to create the solution previously introduced is made. This analysis will cover state-of-the-art space manipulators and their control algorithms, the current setup at ESA and AAU , and how RL is used in the space industry.

### 2.1 Manipulators in Space Sector

Large-scale on-orbit assembly has become increasingly important due to ambitious space missions, including the construction of telescopes, satellite constellations, and deep-space habitats. Such structures often demand versatile and precise robotic manipulators capable of handling a wide range of tasks in dynamic and uncertain environments. Over the last couple of decades, robotic manipulators have played essential roles in space, mainly on the ISS. Prominent examples include Canadarm[5], Canadarm2[6], the European Robotic Arm (ERA)[7], and Dextre[8]. These systems have successfully performed various tasks, including assembly, maintenance, payload handling, and astronaut assistance[9, 10, 11]. The Canadarm and Canadarm2 notably facilitated the ISS's construction and routine maintenance operations by relocating payloads and assisting astronauts during extravehicular activities as seen in Figure 2.1. These manipulators typically operate under direct remote control by astronauts within the ISS, providing precise control but limiting their autonomy[5].



**Figure 2.1:** Astronaut assisted by Canadarm2[6].

The ERA, mounted on the Russian segment of the ISS, extends these capabilities with advanced autonomous control features. It can handle payloads and perform station maintenance semi-automatically or through real-time teleoperation, thus representing a step towards greater autonomy[12, 7]. Additionally, ESA has explored teleoperation techniques through projects such as Analog-1[13], aimed at enhancing remote control capabilities under latency and limited feedback conditions. These efforts aim to refine teleoperation strategies, addressing the constraints inherent in space operations to enable greater precision and safety during remote interactions[13, 9].



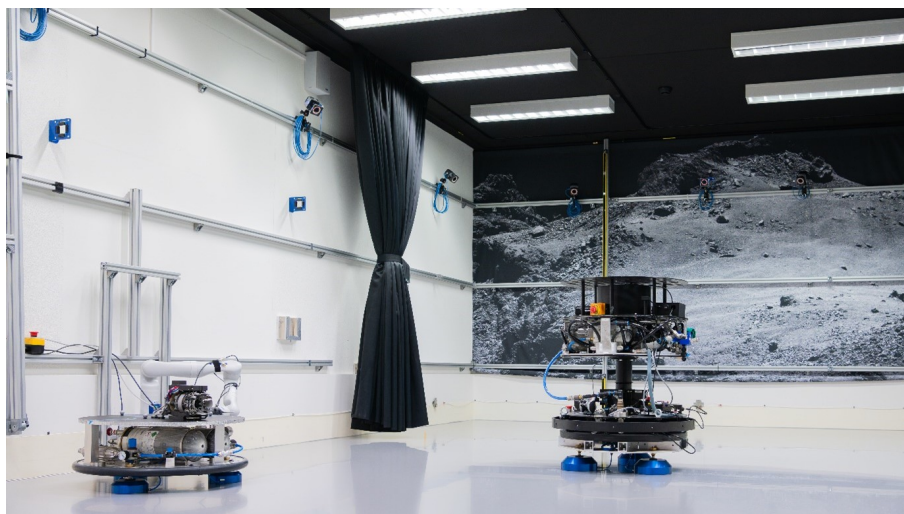
While existing manipulators excel in diverse, general-purpose operations, assembling specific structures such as large telescope mirrors or precise satellite panels introduces unique challenges. ESA's MIRROR project addresses this niche by developing a modular, multi-arm robotic architecture explicitly designed for precise interactions with delicate, flat surfaces. The MIRROR project has demonstrated that a modular robotic architecture is well-suited for assembling large structures in space as well as servicing satellites and moving cargo around in later space missions. The MIRROR system features a relocatable multi-arm manipulator equipped with HOTDOCK standard interconnects, torque-controlled joints, and impedance-based controllers to enable safe interactions with its environment [14, 15].

Once the role of manipulators in space and the relevance of the MIRROR project within this context is understood, the next step is to explore how such systems can be effectively tested on Earth, at the ORL, without requiring deployment in space.

## 2.2 Orbital Robotics Laboratory (ORL)

To test systems in microgravity environments, ESA has designed the ORL, visible in Figure 2.2, for physical testing of developed systems. In the ORL, a 5x9 m epoxy-covered floor is used as the groundwork for the microgravity simulation.[16] This specialized floor simulates microgravity in a horizontal plane, and the low friction is achieved using a stable air gap between the floor and the floating platforms. To locate items on the floor, 44 infrared cameras are placed to cover the entire floor, which can estimate the precise pose of objects using a VICON motion tracking system.

These systems together form an environment that enables test interaction in weightlessness and control systems for thrusters.[17] Currently, two different floating platforms are available for use on the floor in the ORL, the REACSA and the MANTIS systems.



**Figure 2.2:** Image of the ORL with the MANTIS platform on the left side and the REACSA platform on the right.[18]



### **REACSA platform**

The REACSA platform, shown on the right side of Figure 2.2, is the largest air-bearing platform available in the laboratory with a weight of 200kg. It is designed to emulate space dynamics in a planar microgravity-like environment by levitating using compressed air on a floor similar to the one in the ORL. The platform measures 0.7m in diameter and has a height of 1m, being capable of supporting payloads of up to 50kg in its standard configuration and 96kg in a modified configuration where the upper modules are removed. It can be operated either using onboard compressed air tanks, which allow for around 15 minutes of autonomy, or through a tethered setup that enables continuous operation by supplying air externally.

Control of the platform is achieved through eight independently actuated air thrusters that provide planar translation and rotation, complemented by a reaction wheel for precise yaw control. This allows the platform to follow commanded trajectories and perform realistic orbital maneuvers such as attitude adjustments or docking simulations. REACSA was designed for testing control algorithms in scenarios relevant to on-orbit servicing, debris removal, and cooperative multi-agent tasks under realistic dynamical constraints [19].

### **MANTIS platform**

The MANTIS platform, shown on the left side of Figure 2.2, is the smallest air-bearing platform available in the ORL. It has a circular base with a diameter of 0.7m and stands approximately 0.3m tall, weighing around 35kg. Despite its compact size, the platform can support payloads of up to 100kg, making it well-suited for testing lightweight robotic arms and hardware prototypes.

Unlike REACSA, the MANTIS platform does not include onboard actuators for motion control, there are no air thrusters or reaction wheels integrated into the system. As a result, all control must be executed by the system mounted on top of the platform, such as a robotic manipulator or an external control module. This lack of built-in actuation simplifies the platform design but shifts full control authority to the payload, making it ideal for evaluating autonomous robotic systems that require direct control over their own movement. MANTIS can also operate untethered using compressed air tanks for up to 15 minutes, or in a tethered configuration for continuous operation.

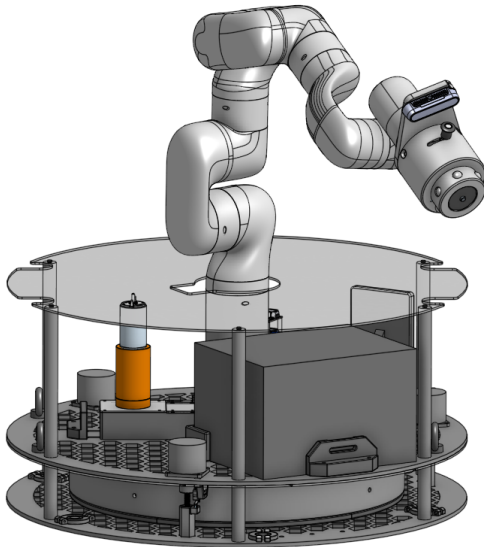
When comparing both platforms, the MANTIS Platform has a higher payload while meeting all the necessary requirements for the project with a simpler and lighter setup, making it the preferred platform for testing. Although the REACSA Platform in its modified configuration also has a payload of 96kg it requires it running without air tanks requiring to be tethered for all tests performed [17].

### **2.2.1 SKYWALKER Platform**

To enable preliminary testing at AAU prior to the main campaign at the ORL, a group of students developed a physical prototype named the SKYWALKER platform as seen in Figure 2.4. It is designed as a simplified version of the ORL platform, it reduces friction using caster ball bearings that roll on a metal plate in place of air bearings. A braking mechanism is included to simulate the deactivation of air bearings on the MANTIS platform by locking the base in place.



The platform is equipped with a battery for untethered operations, an onboard PC, and a driver for the robotic arm, which is mounted on top. To replicate the docking behavior of the MIRROR system, the arm includes a custom-made end effector and corresponding mounting points. A momentum wheel has also been integrated to provide active stabilization during motion, helping to compensate for disturbances and maintain control under dynamic conditions.



**Figure 2.3:** CAD model of the SKYWALKER system developed at AAU .



**Figure 2.4:** Physical setup of the SKYWALKER system developed at AAU .

## 2.3 RL on manipulators

RL is a machine learning paradigm in which an agent learns to make decisions by interacting with an environment to maximize cumulative rewards, making it particularly suitable for sequential decision-making tasks like robotic manipulation.

A powerful extension of RL is Deep Reinforcement Learning (DRL), which uses deep neural networks to approximate value functions or policies. This enables learning in high-dimensional, continuous spaces that are common in robotics.

DRL algorithms typically fall into three main categories: value-based, policy-based, and actor-critic methods [20]. Value-based methods estimate the expected return of taking certain actions in given states and select the action with the highest value. These methods work well in discrete action spaces but struggle with high-dimensional, continuous control problems common in robotics. Policy-based methods take a different approach by directly learning a policy that maps states to actions. While they can handle continuous actions more naturally, they may suffer from high variance and slow convergence. Actor-critic methods combine the strengths of both approaches by using both an actor (the policy) and a critic (the value function). The critic provides feedback to improve the actor's decisions, leading to more stable and sample-efficient

learning [21].

Due to their ability to handle continuous action spaces and maintain training stability, actor-critic architectures are especially suited for robotic manipulation tasks, which involve precise and dynamic control.

This section will review four DRL algorithms that are particularly relevant to robotic manipulation tasks. These algorithms are DDPG, PPO, SAC, and TQC. Although Proximal Policy Optimization (PPO) is technically a policy-gradient method, it incorporates an actor-critic structure internally to enhance training stability and performance, making it functionally similar to actor-critic algorithms in practice and thus being one of the considered options.

### 2.3.1 Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic, model-free algorithm specifically designed for continuous action spaces, making it highly suitable for robotic manipulation tasks. DDPG extends the deterministic policy gradient method to deep neural networks, enabling stable learning in high-dimensional continuous control spaces. However, DDPG can exhibit issues such as overestimation bias and instability during training due to its deterministic nature and sensitivity to hyperparameter tuning.[22]

In [23], a DDPG-based method is implemented for trajectory control of a 6-DOF industrial manipulator. The authors use simulation environments to validate the effectiveness of DDPG in achieving smooth trajectory tracking and joint control, showcasing its ability to handle the nonlinear dynamics typical of industrial arms. This method also demonstrates robustness under external disturbances, which is key for practical deployment.

A similar study in [24] explores the continuous control of robot manipulators using DDPG. Their results highlight improved convergence in policy learning and reduced position errors when compared to traditional PID controllers. The inclusion of reward shaping techniques enhances learning speed and control precision, reaffirming DDPG's suitability for fine-grained manipulation tasks.

The incorporation of parameter noise into DDPG is explored in [25]. This variant addresses the exploration challenges inherent in deterministic policies by injecting noise directly into the actor network's parameters. The approach enables more structured and efficient exploration compared to standard action noise, resulting in improved learning stability and generalization across similar tasks.

A broader perspective is presented in the mini-review [26], which evaluates various DRL methods for manipulation with a strong emphasis on DDPG. The paper discusses practical considerations such as sample efficiency, policy stability, and the importance of hyperparameter tuning in real-world robotic applications. It also highlights the increasing trend of combining DDPG with imitation learning or demonstration-based pre-training to accelerate convergence and reduce unsafe exploration.

Collectively, these works demonstrate DDPG's effectiveness in robotic manipulation across various setups, including high-DOF manipulators, redundant kinematics, and dynamic task conditions. Modifications like parameter noise and reward shaping further enhance its robustness, making it a strong candidate for manipulation tasks in both simulation and real-world environments.

But some limitation of the DDPG is presented in the papers. DDPG is prone to instability during the training phase. This is primarily due to sensitivity to learning rates, reward shaping and noise. DDPG tends to overestimate the Q-values, leading to less optimal policies. This is further intensified when the parameter or action noise is not added within acceptable levels, especially in high DOF tasks.

### 2.3.2 Proximal Policy Optimization (PPO)

PPO employs a policy-gradient approach, directly optimizing actions rather than estimating intermediate values. PPO efficiently manages both discrete and continuous action spaces, making it suitable for precise continuous control tasks like robotic manipulators. Additionally, PPO simplifies hyperparameter tuning, leading to more stable training compared to other methods.

PPO has been adopted to address several challenges in motion planning and autonomous grasping. For instance, in [27], PPO was used to optimize joint velocity targets for a free-floating space manipulator tasked with synchronized motion guidance during in-orbit servicing. By integrating PPO with a feedback linearization controller, the method allowed trajectory generation that adapts dynamically to target motion. Similarly, PPO was utilized in [28] to train a manipulator to grasp irregular-shaped rocks in simulated lunar environments. The use of PCA for object pose estimation helped reduce the complexity of the observation space, while experiments with full camera input showcased PPO's flexibility in high-dimensional state spaces.

Other research, such as [29], integrated PPO with a feature extraction network to autonomously determine grasp parameters (e.g., height, width) for extraterrestrial sample collection. This system proved robust against environmental variability, including changes in object size, color, and friction. In [30], PPO was tested for tracking control and vibration suppression in flexible manipulators under partially observable conditions, highlighting PPO's adaptability to system uncertainty.

PPO has shown great promise for energy-efficient and adaptive motion planning[31]. A digital twin with a reduced-order model and deep neural network compensation used alongside PPO to improve trajectory optimization for speed and energy efficiency. Another paper, [32], introduced an Improved PPO variant for obstacle avoidance in joint space, validated through a Sim-to-Sim-to-Real pipeline using both Gazebo and physical robots. Here, the model was trained in a simplified environment to learn the movement, trained in a more realistic simulated environment for fine-tuning, and at the end tested on a physical robot. Moreover, research on visual-based PPO learning for manipulators [33] emphasized the importance of punishment strategies and visual feature extraction in improving convergence speed and final task performance.

While PPO is generally more stable than earlier methods like DDPG, it can struggle with sample inefficiency in high-dimensional problems and may require careful reward shaping. However, its balance of performance and robustness makes it highly applicable for both space and Earth-bound manipulation tasks.

### 2.3.3 Soft Actor Critic (SAC)

Soft Actor Critic (SAC) combines the advantages of value-based and policy-based methods through an actor-critic architecture. SAC employs entropy regularization, promoting exploration and preventing premature convergence. SAC is particularly advantageous for tasks involving high-dimensional continuous control, like robotic manipulators, balancing exploration and exploitation effectively.

In the context of space manipulation, SAC has demonstrated promise in tasks involving high latency and limited feedback. This can be seen in [34], where a SAC-based controller was employed for teleoperated manipulators under varying communication delays, showing that SAC can effectively mitigate the negative impact of latency on control accuracy. Similarly, in [35], a SAC algorithm was used for collision-free trajectory generation during grasping of moving and rotating objects in space, proving its robustness to dynamic conditions.

Industrial applications also benefit from SAC's stability and efficiency. A hybrid approach that combines SAC with attention mechanisms and neural networks was used to enhance adaptability in dynamic environments [36]. This method allowed the manipulator to learn spatial-temporal relationships and adjust its actions more effectively in real time. In [37], SAC was paired with a modified imitation learning method to track time-varying trajectories using a 3-DOF robotic arm. This combination allowed for more nuanced behavior learning from expert demonstrations.

For more complex environments, such as multi-arm systems or those with periodically moving obstacles, SAC has been integrated with a hindsight replay method to increase sample efficiency and learn more robust policies [38]. Furthermore, domain randomization has been applied to SAC-based systems for tasks of free-floating targets, enabling reliable training that generalizes to real-world space scenarios [39].

Despite its strengths, SAC is not without drawbacks. It often requires substantial tuning and computational resources, especially when combined with other architectures. Moreover, its stochastic nature, while useful for exploration, can lead to suboptimal deterministic performance if not handled carefully.

### 2.3.4 Truncated Quantile Critics (TQC)

Truncated Quantile Critics (TQC) builds upon DQN and SAC, addressing overestimation biases by learning distributions of returns using quantile regression.[40] TQC employs multiple critic networks, each modelling return distributions through minimal quantiles, thus significantly re-

ducing overestimation biases and improving training stability.

TQC has been applied to both simulated and real-world robotic tasks with promising results[40]. TQC demonstrates better sample efficiency and final performance than SAC, albeit at the cost of higher memory usage and computational load due to the multiple critic networks involved.

In robotic manipulation, the use of TQC to imitate human-like motion using a converging reward space, which is gradually increased in complexity[41]. This includes applications where TQC was used to place irregular objects without causing surface damage, ensuring smooth motion close to environmental constraints[42].

Furthermore, [43] applies an entropy-guided version of TQC with dynamic discount factors to enhance adaptability in tasks with limited training data. Similarly, [44] introduces shared control templates to simplify the state-action space, reducing the learning burden while achieving reliable sim-to-real transfer for tasks such as pouring or clamping.

In some tests, TQC shows success in sim-to-real robotic grasping using photorealistic rendering, curriculum learning, and using a strong generalization[45].

While TQC is effective in reducing overestimation and improving training stability, its drawbacks include increased computational requirements and a relatively nascent ecosystem with fewer benchmarks and tools compared to more established methods like PPO or SAC.

### 2.3.5 RL Decisions

PPO was selected for implementation based on a combination of practical and technical considerations.

Compared to DDPG, which is known for instability due to Q-value overestimation and sensitivity to hyperparameters, PPO offers greater robustness during training. Unlike SAC and TQC, which show excellent performance but require more intensive tuning and computational resources, PPO is easier to configure and integrates more readily into available simulation frameworks such as Isaac Lab and SKRL.

Furthermore, PPO's clipped objective function helps ensure stable learning in environments with sparse or delayed rewards, a common condition in early phases of curriculum learning. Since SKYWALKER involves free-floating base motion, grasping, and multi-stage docking tasks, this robustness is critical to avoid training collapse mid-way through long curriculum phases.

Given its reproducibility, support in modern RL libraries, and balanced performance across multiple criteria, PPO was deemed the most practical and reliable algorithm for this proof-of-concept implementation.



### 2.4 Problem Analysis Summary

This section summarizes the key decisions and reasoning developed during the Problem Analysis.

This Problem Analysis began by addressing the need for more autonomous and adaptable control in uncertain environments, motivating a shift from traditional control methods to RL for systems like MIRROR. Due to the risks and constraints of testing in real microgravity, a step-wise simplification was chosen to validate the RL approach safely, starting with a testable, friction-reduced platform at AAU before moving to ESA 's microgravity lab. Among the RL algorithms reviewed, PPO was selected for its balance between performance and stability, making it suitable for safe and effective training in constrained, continuous control environments.

## 3 Problem Formulation

This chapter presents the final problem statement derived from the problem analysis conducted in Chapter 2. Following the problem statement, this chapter outlines the objectives defined to address it, along with the project's requirements. Lastly, acceptance tests are introduced to verify compliance with these requirements.

### 3.1 Final problem statement

Characterized by the information presented in the problem analysis Chapter 2, a more refined problem statement is formulated based on the conclusions drawn from the investigation of current space manipulators and RL algorithms. In the analysis it was determined that PPO would be the ideal RL algorithm for this project, as the algorithm is considered to provide the best balance of training stability, performance in continuous control spaces, and robustness to environmental uncertainty which are all key factors for safe and effective robotic operation in space-like conditions.

Given these findings, the final problem statement is formulated as:

*"How can a PPO Reinforcement Learning algorithm be used to enable autonomous control and navigation for the SKYWALKER system in a simulated microgravity environment. "*

This problem statement serves as the foundation for the objectives and requirements of the proposed solution.

### 3.2 Delimitations

While the broader goal is to enable autonomous assembly using complex multi-arm systems like ESA 's MIRROR platform, this project is focused on delivering a proof-of-concept implementation under constrained conditions to validate the feasibility of using RL for space-relevant manipulation tasks.

The following delimitations define the project scope:

- **Single-arm configuration:** The implementation uses a single robotic arm mounted on a mobile base rather than a multi-arm setup. This simplification allows focused experimentation on locomotion and grasping behavior without the additional complexity of multi-arm coordination.



- **Simulated environments:** All training and validation are conducted in simulation using Isaac Lab, with later testing on frictionless physical platforms (e.g., the SKYWALKER or ORL systems) as a proxy for true microgravity.
- **Simplified task scope:** Rather than performing full-scale orbital assembly, the robot is tasked with navigating between predefined grapppling points, simulating core behaviours relevant to space manipulation and mobility.

These delimitations are made to keep the project focused on validating the technical potential of using PPO for autonomous robotic control in space-like environments, while laying the groundwork for future expansion.

### 3.3 Objectives

The objectives of this project are the following:

ID	Title	Objective Statement
<b>Scientific Objectives</b>		
SCI-001	RL -Based Navigation	Ensure that the robot can autonomously reach a given location and decide how to move efficiently while aware of its surroundings using RL .
SCI-002	RL Control	Demonstrate RL can be effectively used as a controller for handling complex force and position control
SCI-003	Microgravity Dynamics	Investigate difficulties and differences associated with moving in a microgravity environment using an RL controller
SCI-004	Environment Adaptation	Validate that RL controllers can seamlessly transfer from simulation to real-world test beds (AAU and ORL) while retaining performance.
<b>Technical Objectives</b>		
TEC-001	SKYWALKER platform as Validation	Use the SKYWALKER low-friction platform to validate an RL controllers simulation to the real world before testing on the high-cost microgravity platform in ORL.

**Table 3.1:** The project objectives for the validation of the SKYWALKER system



## 3.4 Requirements

This section will cover the requirements needed to be fulfilled to meet the final problem statement and objectives. These requirements are derived from Chapter 2 and categorized into Functional and Design. These requirements are based on a more extensive requirement list made for the ESA Academy Experiment Programme collaboration, which can be viewed in the Appendix A.2.

Functional requirements	
F 1	The system shall be able to reposition its base to a goal position.
Type	Notes
F 1.1 Grasping	The system shall be able to grasp mounting points within 30 seconds.
F 1.2 Stopping	The system shall be able to stop the base in order to reach the next mounting point.
F 1.3 Safety	The system must not unlock the base if it is not grasping a mounting point.
F 1.4 Traversal	The system shall reach the goal 90% of the time.

**Table 3.3:** Functional requirements.

These requirements are particularly made concerning the movement of the full system, and with the interaction of the manipulator with the grasping point. Since the project is made in collaboration with ESA Academy Experiment programme, the constraints in the requirements in Table 3.3 are influenced by comparable systems from ESA and adapted to fit the constraints and goals of the experiment.

This can be seen by the 30-second limits as a practical upper limit, due to the less stringent time performance restrictions in comparison to terrestrial applications. For the reduced or frictionless environment, precision is seen as more valuable than time, where the higher limits provide a practical limit for the performance without sacrificing stability or control. For the traversal, a 90% success rate criteria for the tests reflects typical expectations for semi-autonomous systems in structured environments, as well as success rates observed in ESA testing[12]. A success rate criteria of at least 90% is seen as acceptable in accounting for cumulative errors while still maintaining a reliability level for a proof-of-concept system.

To ensure the full system is able to complete the functional requirements, the physical and structural capabilities of the robot must be sufficient. In Table 3.5, the requirements for the design are presented. These requirements ensure that the system's general setup and the manipulator's kinematic setup are sufficient to complete tasks necessary for repositioning in a delimited space environment.

Design requirements	
D 1	The system shall be able to reach all mounting points.
Type	Notes
D 1.1 Reach	The manipulator length shall exceed the radius of the base so it can reach the mounting points.
D 1.2 DOF	The robot shall have sufficient DOF to reach all mounting points

**Table 3.5:** Design requirements.

To ensure the requirements defined in Table 3.3 and Table 3.5 can be verified and are meeting the criteria defined, acceptance tests can be designed.

## 3.5 Acceptance test

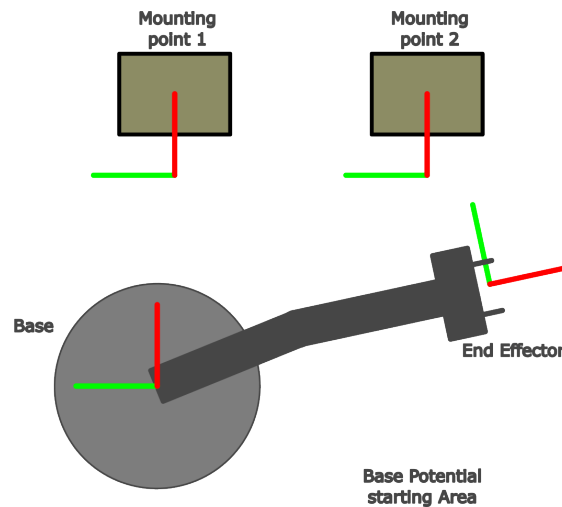
This section will introduce the acceptance tests to validate the requirements presented in Section 3.4. The purpose of these tests is to demonstrate that the developed system can perform the expected tasks in a controlled and repeatable way and provide a direction for the design and implementation chapters. Since our solution targets autonomous robotic control in space-like conditions, all tests are conducted in simulation to ensure safe validation before moving to physical testing.

To provide a structural and clear progression, the acceptance tests are divided into four distinct parts in increasing technical complexity. This progression is to ensure the more basic capabilities are certified before introducing more advanced scenarios.

This section begins with a brief overview of what will be needed in the simulated environment, including the objects present and their placement, followed by a detailed description of each acceptance test and its evaluation criteria.

### 3.5.1 General simulation setup

To test the full system in a simulated environment, a general setup is made. As illustrated in Figure 3.1, the general setup includes the robot and several grasping points aligned in a single row. Each component in this setup, including the base, the manipulator, and the grasping points, has its own coordinate frame, enabling precise positioning and evaluation of the model.



**Figure 3.1:** Illustration of the general setup for the simulated tests. The environment includes two mounting points used for grasping and moving the robot. Coordinate frames are visual for the base, end-effector, and the grasping points used for control strategies and target positioning.

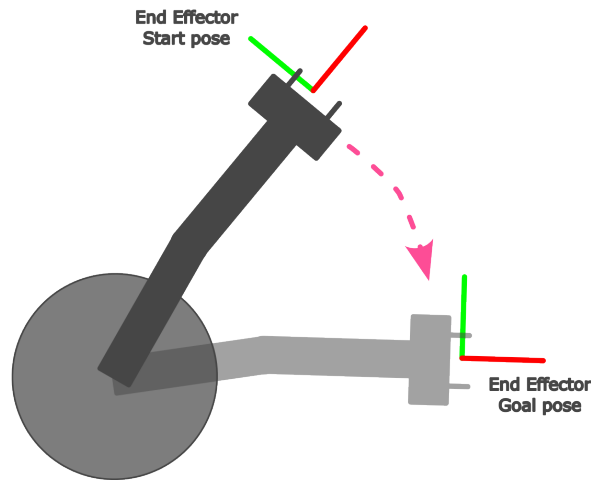
For each test, the robot is spawned at a certain position and tasked with reaching defined goals. Depending on the task, the goal will be defined for the base or the end-effector, but always within reach to ensure feasibility and consistency. Each test will be concluded once the system reaches the goal pose within the given threshold and locks the relevant connectors, or when a timeout is triggered.

Since the focus of these tests is on manipulation and control using RL and less on perception and sensor input to register the surroundings, all relevant information will be available to the models through the simulation. Furthermore, the test environment will be made with the dimensions of the SKYWALKER in mind, as the model is to be applied to the physical model.

## 3.5.2 Point-to-Point motion to learn arm control

**Fulfills Requirement: D1.2, D1.2**

Before training the model to perform grasping and manipulation using the grasping points, it must first learn to move the robotic arm accurately to target positions in free space. This test evaluates the system's ability to execute Point-to-Point motions, moving the end-effector from an initial position to a predefined target pose without obstructions.



**Figure 3.2:** Setup for the Point-to-Point test. The end-effector starts at the initial position and must reach the target pose, indicated by the pink striped arrow.

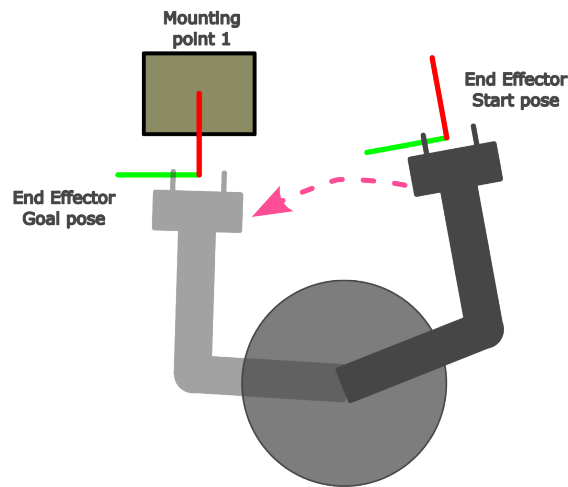
During each trial, the robot is placed in the configuration shown in Figure 3.2, with the goal pose set in the robots workplace.

A test repetition is considered a failure if the end-effector does not reach the target pose within 30 seconds.

### 3.5.3 Grasping Task

#### **Fulfills Requirement: F1.1**

Following the successful execution of Point-to-Point arm motion, the next stage is to train the model to perform grasping. This test assesses the model's ability to move the end-effector to a fixed mounting point and establish a stable grasp, as illustrated in Figure 3.3. The mounting point will be spawned in the robot's feasible area where it will need to reach it and then grasp it.



**Figure 3.3:** Setup for the grasping test. The end-effector starts from an initial position and must reach and grasp the target mounting point. The intended motion is indicated by the pink striped arrow.

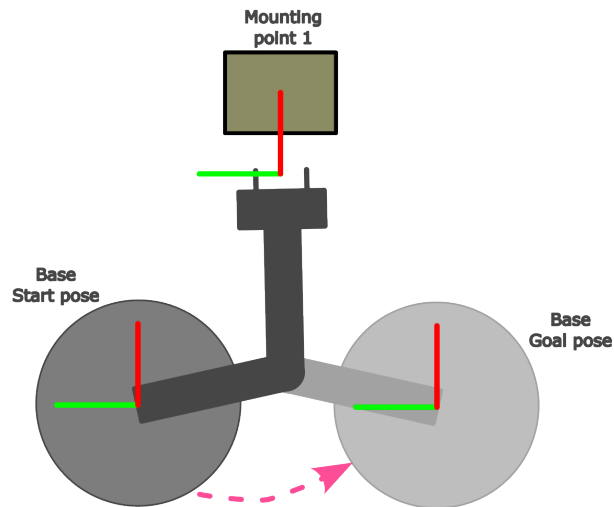
A test repetition is considered a failure if the model does not complete the grasp within 30 seconds.

### 3.5.4 Grasp and move the base to a pose

#### **Fulfills Requirement: F1.2, F1.3, F1.4**

After the model can grasp reliably, the model must learn to move the base while maintaining a stable grasp to a mounting point. This test examines whether the model's ability to grasp the mounting point and move the base to a predefined target pose, as illustrated in Figure 3.4.

The test begins with the robot in its initial base position. The robot must first execute a grasp on the predefined grasping point and then transport the base of the robot to a designated goal pose within the workspace as visualized in Figure 3.4.



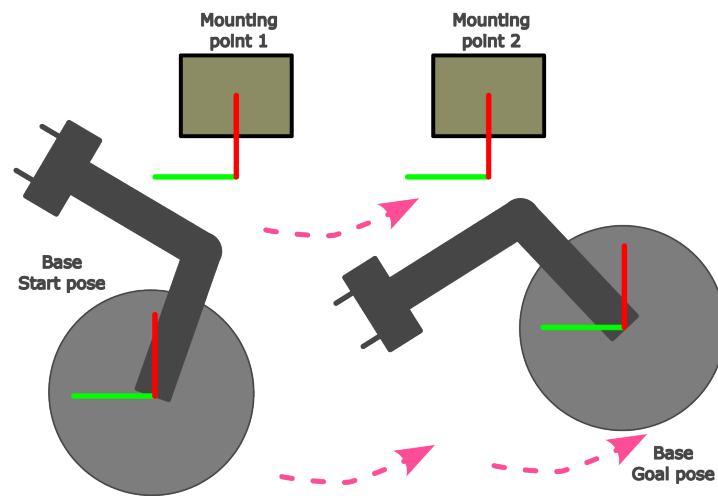
**Figure 3.4:** Illustration of setup for the Grasp and move the base Test. This shows the robot grabbing the first mounting point and dragging the base towards a goal position for the base.

If the model fails to maintain a stable grasp, does not reach the target pose within 30 seconds, the test fails.

### 3.5.5 Move the full setup from Point-to-Point with 2 grasp points

#### **Fulfills Requirement: F1.1, F1.2, F1.3, F1.4**

This test requires the system to be able to move between multiple points, requiring the system to be able to grasp the correct mounting point, move the base enough to grasp the next mounting point, and then move to the goal position as visualized in Figure 3.5. This test integrates all prior skills and ensures that the model can perform sequential grasp-move actions over extended distances.



**Figure 3.5:** Illustration of setup for the multiple grasps Test. This shows the robot starting at the starting pose, 2 mounting points, and the wanted movements by 3 pink striped arrows and a robot placed at the wanted goal pose.

If the model fails to reach the goal position within 30 seconds or test is seen a failure.

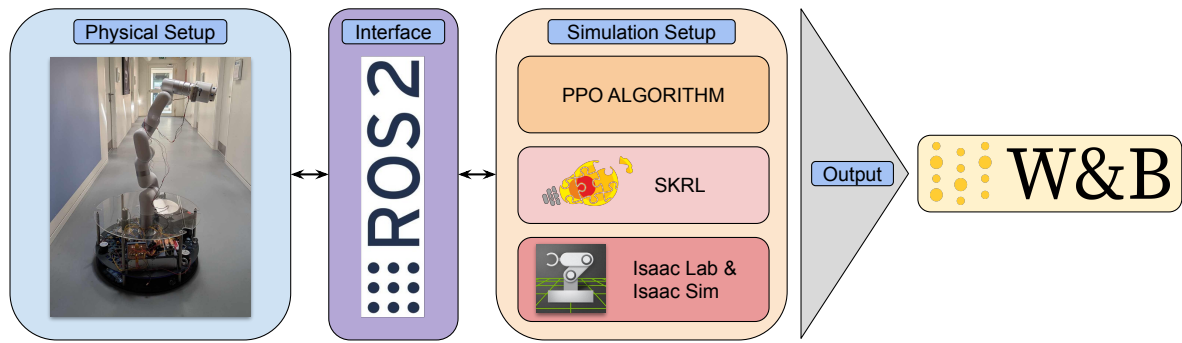
These acceptance tests provide a clear framework for verifying if the developed system fulfils the functional requirements under realistic, simulated conditions.

In the following chapter, the design of the framework to enable these capabilities will be presented.

## 4 Concept design

This chapter presents the overall system design, as summarized in the flowchart shown in Figure 4.1. The design begins with the Physical Setup, which includes the robotic arm and custom hardware models used in the project. This setup interfaces with the Simulation Setup via ROS2, enabling the transition from real-world configuration to virtual training and testing.

Within the simulation framework, several key components are detailed: the simulation platform used to model the robot and its environment, the RL algorithms applied for training, and the middleware libraries that enable communication between the simulation and learning modules. Finally, the training process is monitored and evaluated through an output visualization layer, which provides real-time feedback and performance tracking throughout the training lifecycle.



**Figure 4.1:** Diagram Showing the overall Pipeline for the project.

### 4.1 Physical Setup

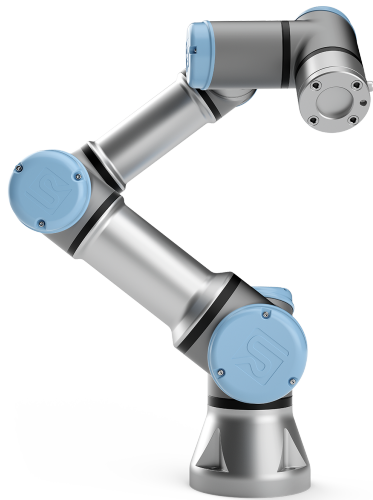
The physical setup includes a robotic arm mounted on a platform. As mentioned earlier, the MANTIS platform will be used for testing at the ORL. To carry out initial tests at the AAU facilities, a custom platform was built with all the components needed to run the algorithm on the robotic arm. This platform uses ball caster wheels on a metal plate to minimize friction and better replicate the conditions of the ORL. The custom platform is also designed to be easily mounted on the MANTIS platform during the main testing campaign.

#### 4.1.1 Robot arms

With the aforementioned design requirements and available robots, two robotic arms become viable for this project. Those robotic arms are the Universal Robots UR3e[46] and the UFactory xArm7[47]. When comparing the two robots, it becomes evident which arm to use. The xArm7 shown in Figure 4.2b has a greater reach, higher DOF, and increased payload capacity, and it



weighs less. The UR3e shown in Figure 4.2a has better documentation online, and it already has some examples implemented in Isaac Sim; however, the technical advantages of the xArm7 outweigh those factors. Therefore, the xArm7 is selected as the robot arm for this project.



(a) UR3e Manipulator[46]



(b) xArm7 Manipulator[47]

**Figure 4.2:** Comparison between both available arms for the project.

As the xArm7 does not have an implementation in Isaac Sim, it will need to be added to the simulated space in the correct format. To add xArm7 to Isaac Sim, it needs to be converted from a Universal Robot Description Format (URDF) file to a Universal Scene Description (USD) file; thankfully, Isaac Sim has a built-in converter for that. The URDF file used for this project is the official URDF model made by UFactory[48].

## 4.2 ROS2 Communication Interface

Robot Operating System (ROS2) is a middleware framework that facilitates communication between different components in a robotic system. It acts as the communication bridge between the physical setup and Isaac Lab in our pipeline. ROS2 enables the transfer of real-time data between the physical robot and the simulation by sending observations from the physical setup to Isaac Lab and actions generated by the model in Isaac Lab back to the physical robot. This communication is crucial for enabling RL in the real world, where the agent must be able to interact with both the simulated and physical environments seamlessly.

ROS2's publish-subscribe model allows for a flexible and efficient transfer of messages, ensuring that the simulation and real-world robot remain synchronized. It also provides tools for handling sensor data, controlling actuators, and managing complex robotic workflows, making it an essential part of the system for both training and deploying the learned policies.

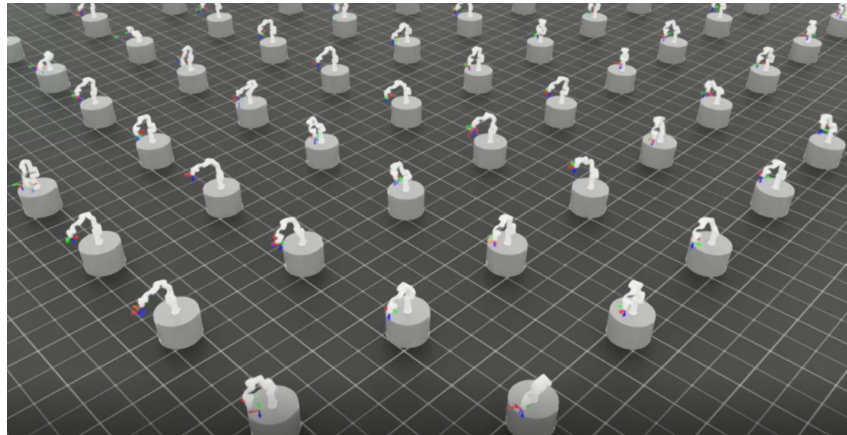
## 4.3 Simulation Setup

This section provides a more detailed explanation of the software and the different components that make up the environment required to complete the simulation setup and begin training. It covers the simulation environment, the simulation platforms, the RL algorithm, and the necessary libraries for the simulation setup to run as a whole.

### 4.3.1 Isaac Sim/Lab

This project will utilize Isaac Sim, a robotics simulation platform developed by NVIDIA.[49] Isaac Sim is built on NVIDIA's Omniverse platform and is designed for realistic, physically accurate simulations of robotic systems. It allows for detailed modelling of physical interactions, sensor inputs, actuator behaviours, and environmental conditions. This ensures that the robotic system developed within this project can be effectively tested and validated in a virtual environment closely reflecting real-world scenarios.

Having a realistic simulation aids greatly when there is a need for sim-to-real transferal. RL algorithms are very dependent on the environment they are trained in, so having realistic parameters lessens the performance degradation of the system. To implement RL training within Isaac Sim, Isaac Lab is used[50]. Isaac Lab is a modular framework designed explicitly for RL in robotic applications, as seen in Figure 4.3.



**Figure 4.3:** Isaac Lab Multiple Environments Visualization.

Its modularity enables easy customization and rapid experimentation with different robot configurations, task scenarios, sensor setups, and environmental conditions. Isaac Lab is particularly suitable for this project due to its seamless integration with Isaac Sim, allowing policies trained in simulation to be directly applicable to real-world hardware. Additionally, Isaac Lab efficiently leverages NVIDIA GPUs to run multiple simulation instances simultaneously, drastically reducing training times. Compared to alternative RL frameworks, Isaac Lab's direct integration with Isaac Sim and its GPU optimization capabilities make it an ideal choice for developing robust, high-performance robotic behaviours for complex space manipulation tasks.

### 4.3.2 Simulation Environment Design

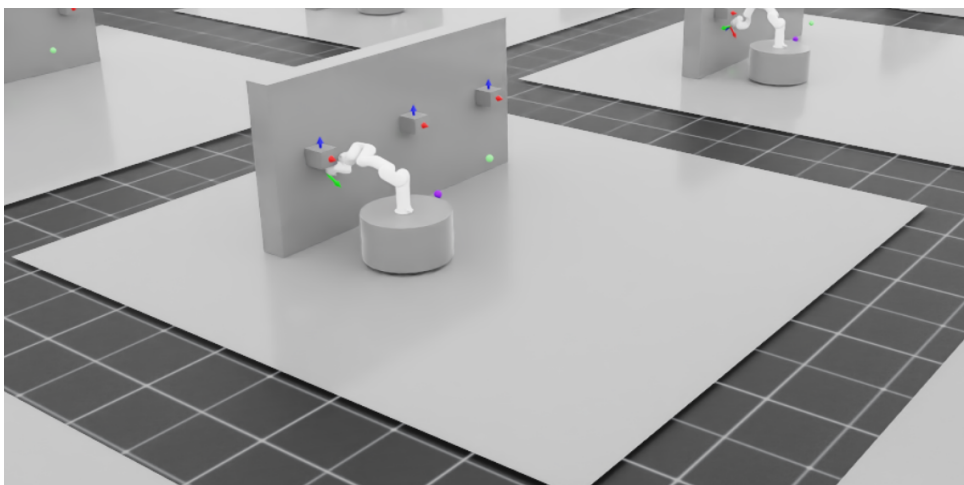
With the simulation platform and robotic arm selected, the next step is to set up a simplified version of the SKYWALKER system in the simulation using Isaac Lab. The goal is to recreate the physical setup closely enough to validate the concept while keeping the simulation efficient in terms of time and complexity.

#### Floor

The simulated environment consists of a flat, frictionless floor that allows the robot to move freely in a two-dimensional plane. In the real-world testbed at AAU, friction is present due to the robot's contact with the surface. However, in the simulation, the aim is to minimize friction to better mimic a zero-gravity effect on the ground plane. This will be more similar to the setup in the ORL. However, this could be modified at any time as a parameter if necessary.

#### Wall with Mounting Points

In addition to the floor, the simulation includes a wall with mounting points that the robot uses to reach its goal, as visualized in Figure 4.4. The mounting points are represented as cubes attached to the wall and are a simplification of the MIRROR project using the HOTDOCK grappling points.

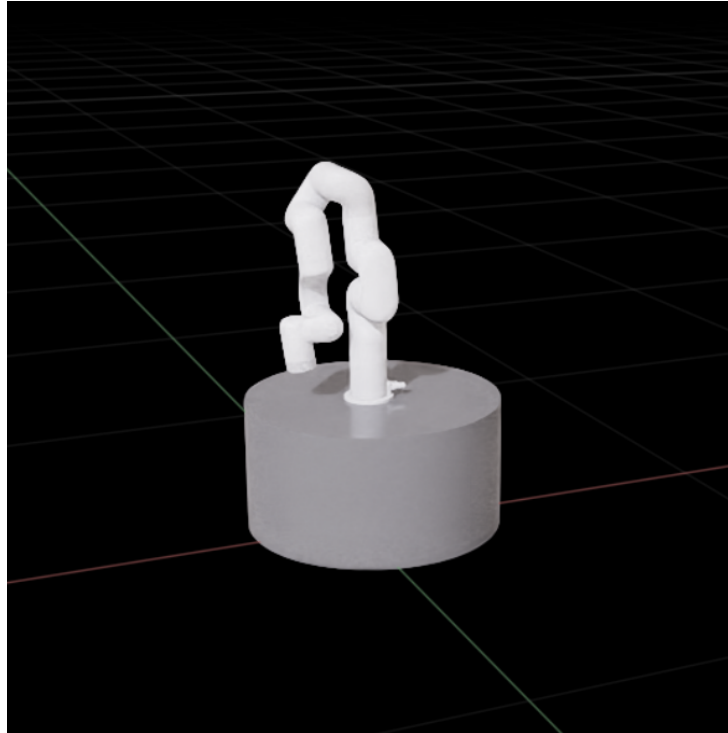


**Figure 4.4:** Environment Setup including the wall, floor, and mounting points.

#### SKYWALKER System

The robot itself will be made up of the xArm7 arm obtained from the URDF file provided by the manufacturer, which is then converted into a USD file by Isaac Sim to make it suitable for the physics engine. The URDF is commonly used to describe the robot's structure in a simple, text-based format, focusing on joints, links, and physical properties, while the USD format is a more versatile, hierarchical format that supports complex 3D scenes, including detailed geometry, lighting, and materials, and is better suited for high-performance simulations and visualizations. This arm will be mounted on top of a cylinder-shaped platform with the same dimensions and weight as the SKYWALKER platform, in order to simplify the transition from the simulation to the real robot. Both the arm and the cylinder will then be saved as a single

USD file, which will be used in Isaac Lab for training. A visualization of this USD model can be seen in Figure 4.5.



**Figure 4.5:** USD file visualization containing the arm and base.

### Gripping Mechanism

The arm USD file includes all the joints of the robot, excluding the gripper. This is where a major simplification is made. In the physical setup, a custom grappling mechanism is designed, as shown in Figure 4.6. Replicating this system in the simulation would require significant time and development, which is not practical at this early stage. Since the main objective is to demonstrate that the robotic arm can learn to crawl using RL, it is not necessary to model the grappling hardware in detail.



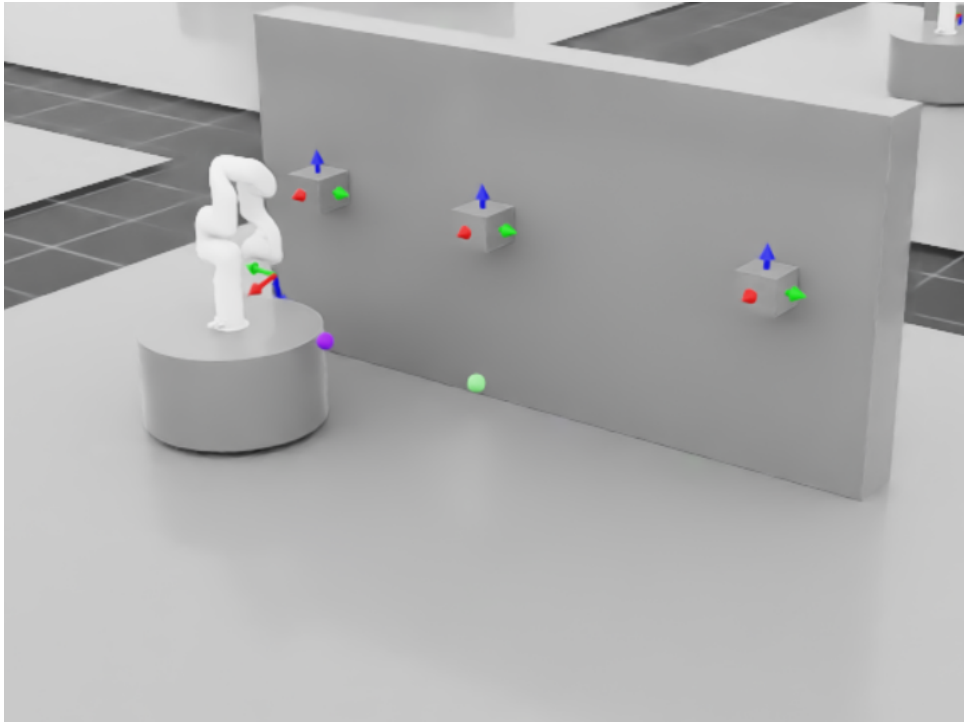
**Figure 4.6:** Custom-designed gripper for the physical setup.

Instead, the simulation uses a surface gripper extension available in Isaac Sim. Although it is not commonly used in Isaac Lab, it suits this application despite some disadvantages, the main one being that the surface gripper extension only runs on the CPU. This means that the GPU will not be utilized for training, which results in longer training times. However, after verifying the difference in training times, the impact was not significant enough to discard this option. The surface gripper works by creating a fixed joint between two objects when they are within a certain distance, and the contact force is below a defined threshold. This simplifies the gripping action into a binary condition, which the gripper activates when the robot is close enough to the target. In the real system, the gripping will not be handled by the RL algorithm. Instead, a separate system will detect the grapppling point, activate the gripper when alignment is correct, and then return control to the RL algorithm once the grip is secured.

A second surface gripper is also used in the simulation to replicate the braking effect of the SKYWALKER platform or the deactivation of the air bearings in the MANTIS platform used at the ORL. This simulates the robot locking its base to the ground, allowing it to reposition its arm before unlocking and continuing to move.

With these components in place, a frictionless floor, wall-mounted grappling points, and surface grippers for gripping and locking, the simulation environment is ready for the first proof-of-concept experiments. An image of the whole environment can be seen in Figure 4.7.





**Figure 4.7:** Simulation Environment in Isaac Lab ready for training.

### 4.3.3 SKRL Library

The SKRL block refers to the integration of Isaac Lab with the SKRL library, which is designed to facilitate RL in robot simulations. SKRL acts as a bridge between Isaac Lab and the PPO algorithm, by providing the necessary functions and utilities to train and evaluate models in a simulated environment. This includes managing the interaction between the environment and the algorithm, ensuring that observations from the environment are passed to the algorithm and actions generated by the trained model are sent back to the simulation for further execution [51].

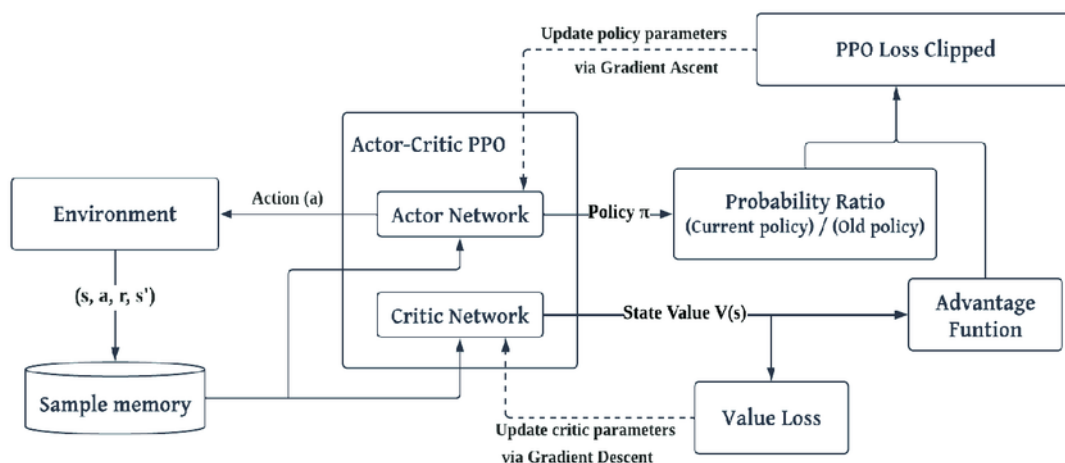
### 4.3.4 PPO Algorithm

As shown in the previous diagram in Figure 4.1, the PPO block communicates with Isaac Lab through SKRL, intending to transform the observations obtained from Isaac Lab into the actions that will be performed. It is also responsible for updating and improving the policy with the rewards defined to make the system learn its way through the environment. PPO's versatility and efficiency make it a great fit for integration with SKRL and Isaac Lab, because the combination of PPO's stable learning and Isaac Sim's realistic physics ensures that the policies learned can be transferred to real-world robotic systems with minimal fine-tuning.

## PPO Learning Workflow

PPO can be used as a DRL algorithm to optimize a policy by using a neural network-based model while ensuring that the policy updates remain stable. In the context of PPO, a policy refers to a model that maps observations taken from an environment to the actions that the agent should take. These observations can include various data, such as the position and velocity of objects, visual inputs, or sensor readings. Meanwhile, actions are an agent's decisions or movements, such as navigating to a specific location or manipulating an object.

The PPO algorithm operates through a cycle of interacting with the environment, updating the policy, and evaluating the results as visualized in Figure 4.8. After performing an action, the agent receives a reward (or penalty) that indicates how well the action contributed to achieving the goal. This makes the design of reward functions a critical task, as they influence the agent's ability to learn effectively. The episode continues until a termination condition is met, such as reaching a goal, failing, or completing a task within a given time limit. This helps the agent know when it has successfully achieved its task or when it needs to start over.



**Figure 4.8:** PPO Flow Chart.[52]

To guide the agent's decision-making process, PPO employs a neural network, called the policy network, which maps observations to actions. This is crucial because in complex environments, especially those with high-dimensional input data like images or sensor readings, the state space is vast, and traditional rule-based methods are impractical. The policy network is also referred to as the actor, as it is responsible for selecting actions. However, the policy network alone is not enough for effective learning. In addition, PPO uses a second neural network called the value network, which estimates how good a particular state is in terms of expected future rewards. This value network is also known as the critic.

When the agent takes an action and receives a reward, the value network helps predict the total expected return from that point onward, given the current observation. This prediction is then compared to the actual reward, creating the advantage function, which helps the algorithm determine if the action led to a better or worse outcome than expected. By using the advantage,



PPO can adjust its policy to improve future decision-making. The advantage function helps find a balance between exploration and exploitation, a common challenge for RL algorithms. This is achieved as follows: when an action yields a higher than expected reward, the advantage function increases, promoting the agent to take similar actions in the future, which promotes exploitation. On the other hand, if an action leads to a worse outcome than expected, the advantage will be negative, and the agent will explore different actions in search of a better strategy.

To ensure stable learning, PPO uses a function that controls how much the policy can change at each update. Without such constraints, large policy shifts could lead to instability or performance collapse. There are two common methods used to enforce this stability constraint:

- **Clipped Surrogate Objective:** Limits the change in the policy by restricting the probability ratio between the new and old policies. If this ratio exceeds a predefined threshold, the objective is clipped, preventing excessively large updates that could degrade performance.
- **KL-Divergence Penalty:** Adds a penalty term to the loss function that measures how far the new policy diverges from the old one. The penalty strength can be adaptively tuned if the divergence is too high, the penalty increases to suppress further changes; if it is low, the penalty is reduced to allow more flexibility.

This approach allows PPO to make safe incremental improvements to the policy using stochastic gradient ascent, ensuring that updates are stable and efficient. The clipping mechanism, in particular, helps prevent overly drastic changes to the policy, making PPO more stable and reliable in dynamic or complex training environments.[53]

### 4.4 Weights and Biases Output Visualization

Weights and Biases, commonly known as WANDB is a tool used for tracking, visualizing, and analyzing the performance of machine learning models, particularly in the context of RL. During the training process, WANDB helps monitor various metrics such as rewards, loss functions, and the progress of the model over time. In RL tasks, WANDB is crucial for understanding how the algorithm is performing and for identifying patterns in the training process.

In this pipeline, WANDB is integrated with Isaac Lab to log and visualize key training metrics in real-time. This helps to track the agent's progress, evaluate its performance, and make necessary adjustments to the model. It also enables easy comparison of different training runs, allowing for an informed approach to optimization.

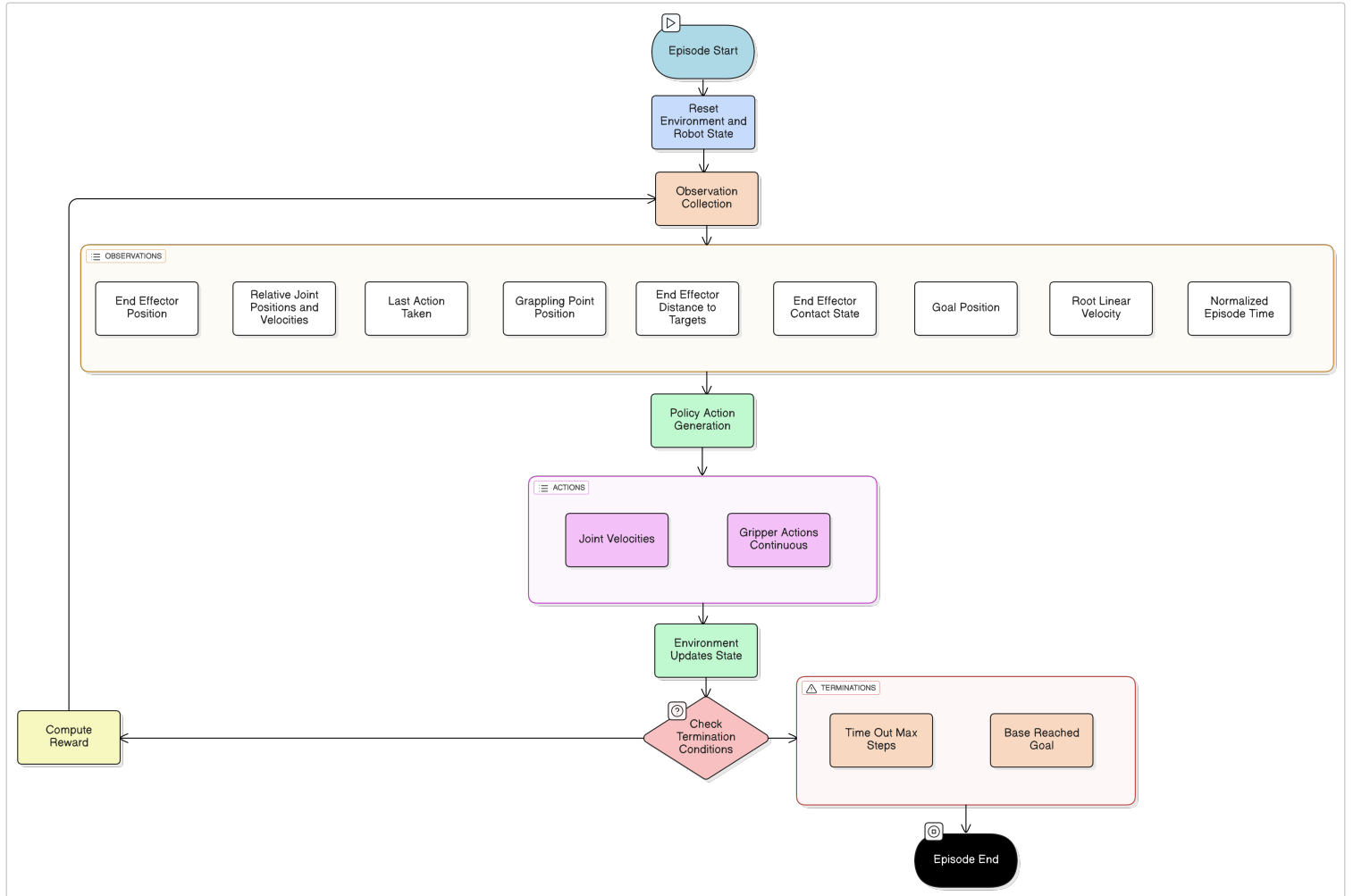


## 5 Implementation

This chapter presents the implementation details of the SKYWALKER reinforcement learning pipeline in Isaac Lab. The goal of this implementation was to fulfill the acceptance tests described in Section 3.5 and meet the requirements outlined in Chapter 3.

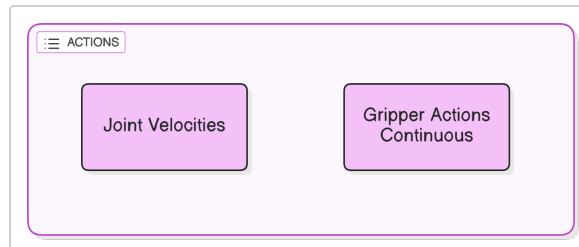
It covers the definition of the action space, observation space, reward structure, and termination conditions used during training. In addition, the simulation environment configuration and the key hyperparameters for training are described.

A visual summary of the overall system architecture is provided in Figure 5.1. The full implementation and codebase can be accessed via the GitHub repository linked in Section A.1.



**Figure 5.1:** Overall architecture of the Isaac Lab implementation.

## 5.1 Action Space Definition



**Figure 5.2:** A block diagram showing the action space of the system.

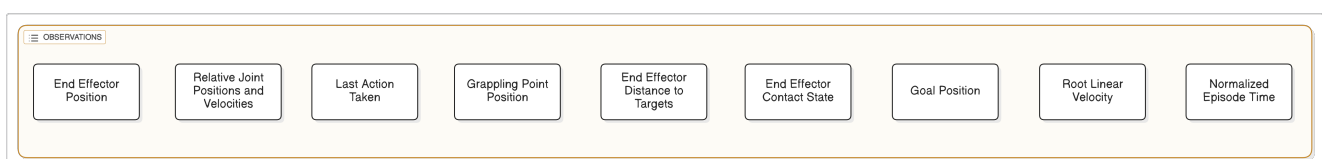
This section will explain the content of the Actions block seen in Figure 5.2, zoomed version of the actions block in Figure 5.1 As described in Chapter 4, the USD model of the SKYWALKER robot used in simulation includes both the xArm7 robotic arm and its supporting platform. The robot has seven actuated joints and two surface grippers used for attaching to the environment. These components define the action space used by the PPO algorithm during training.

In RL , the action space defines the possible outputs of the neural network at each time step, which the agent uses to interact with its environment. In this setup, the PPO algorithm must control a total of 9 actions:

- 7 continuous actions corresponding to the joint velocities of the xArm7 arm.
- 2 discrete actions corresponding to the activation state of the two surface grippers.

However, Isaac Lab’s RL interface only supports continuous action spaces. This creates a challenge for implementing binary actions, such as the surface gripper states (activated or not). To overcome this, continuous values in the range  $[0, 1]$  are produced by the policy, and clipping is used to convert them into discrete values during execution. For example, a gripper action value above 0.5 may be interpreted as activated, while a value below is deactivated. This technique allows the PPO policy to learn binary behaviors within a continuous action space.

## 5.2 Observation Space Definition



**Figure 5.3:** OBlock diagram for the observation space for the RL system.

This section will explain the content of the Observations block seen in Figure 5.3 zoomed version of the actions block in Figure 5.1 In RL, the observation space provides the necessary input for the policy to understand the current state of the environment and decide what action to take. This input is a combination of several observation terms that capture both the internal state of the robot and relevant information from the task environment. These observations are updated at every simulation step and passed to the neural network policy, which learns to map them to optimal actions over time.

The observations used in the SKYWALKER simulation setup are defined in a configuration class and consist of the following:

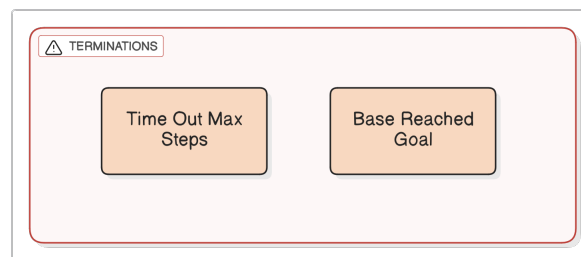
- **End-effector position:** This observation provides spatial information about the robotic arm's end-effector position relative to its base, which is essential for the policy to navigate and reach targets in 3D space. Since it encodes the  $(x, y, z)$  coordinates, it is naturally represented as a 3-dimensional vector, reflecting the three degrees of spatial freedom.
- **Relative joint positions and velocities:** These observations capture the angular positions and velocities of each joint in the robotic arm relative to their rest positions. Since there are 7 joints, each contributing one position and one velocity value, the position and velocity vectors are a total of 42 values. Combined, these form a 14-dimensional observation vector, which fully describes the current configuration and motion of the arm.
- **Last action taken:** Including the previous control action gives the policy temporal context, helping to stabilize learning and smooth out control commands. The size of this vector corresponds to the dimension in the action space.
- **Mounting point position:** This provides the position of the mounting point relative to the robot base, crucial for planning motion towards it. Like the end-effector position, it is represented as a 3-dimensional vector because it describes a point in 3D space.
- **End-effector distance to target objects:** These are scalar Euclidean distances computed between the end-effector and each target object. By explicitly providing these distances, the policy receives spatial feedback directly tied to the task goal, which simplifies the learning problem. Each distance is a scalar because it measures a single length, but the observation includes one such scalar per target object.
- **End Effector contact state:** A binary value indicating whether the surface gripper at the end-effector is currently engaged with an object. This single scalar flag helps the policy decide when to activate or release the gripper.
- **Goal position:** This is the relative position of the target goal with respect to the robot base, used to guide the robot's movement and reward signals. Because the goal exists in 3D space, this observation is again a 3-dimensional vector.
- **Root linear velocity (x, y):** The translational velocity of the robot's base in the horizontal plane is important for balance and navigation. Since it is restricted to planar motion, it is represented as a 2-dimensional vector capturing velocities along the  $x$  and  $y$  axes.
- **Base contact state:** A binary scalar flag that indicates whether the robot's base gripper is locked or free-floating. This single-value observation informs the policy when to engage the

base lock mechanism.

- **Normalized episode time (time fraction):** A scalar value between 0 and 1 representing how far the current episode has progressed. Providing this temporal context enables the policy to adapt its strategy dynamically throughout the episode.

Each of these terms contributes essential information for enabling the PPO policy to learn effective and context-aware behavior by ensuring that the policy has the necessary feedback to adapt to different stages of the manipulation task.

### 5.3 Termination Conditions



**Figure 5.4:** Block diagram of the Terminations used.

This section will explain the content of the terminations block seen in Figure 5.4 zoomed version of the actions block in Figure 5.1 In RL, termination conditions define when an episode should end. These conditions are critical for both learning efficiency and task specification, as they signal to the algorithm when a meaningful conclusion to an attempt has been reached, either due to success, failure, or time constraints.

For the SKYWALKER robot simulation, the termination logic is encapsulated in the configuration class `TerminationsCfg`. The following termination terms are used to control the episode lifecycle during training:

- **Time-out:** This termination occurs when the maximum number of simulation steps per episode is reached. It ensures that episodes have a bounded length, which helps stabilize training and prevents infinite loops in failure scenarios.
- **Goal Reached:** An episode is also terminated if the base of the robot is detected to be around the target goal position within a certain tolerance and its base surface gripper is closed.

These termination conditions serve two main purposes: they reduce the amount of unnecessary exploration once a task is completed, and they prevent the accumulation of rewards in unproductive scenarios. The reward function can then be designed to provide a terminal reward upon successful completion, allowing the agent to better associate its actions with positive outcomes.

## 5.4 Reward Definition

In RL, rewards serve as the primary signal that guides the agent's learning process. At each time step, the agent receives a reward based on its actions and the resulting state, which reflects how well it is performing the task. The goal of the agent is to maximize the cumulative reward over time.

In PPO, rewards directly influence both the policy and the value function. The value function estimates the expected cumulative reward from a given state, helping the algorithm evaluate how good a state is. The policy function uses this information to improve action selection by increasing the likelihood of actions that lead to higher expected rewards.

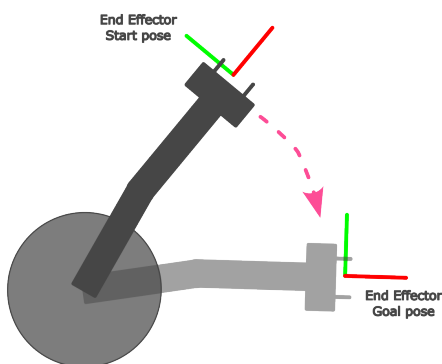
The model will be trained using a series of tasks designed to progressively increase in complexity. These tasks are structured to ensure the robot acquires foundational skills before moving on to more challenging scenarios. The training tasks are as follows:

**Point-to-Point movements:** Establishing basic motion control capabilities as visualized in Figure 5.5.

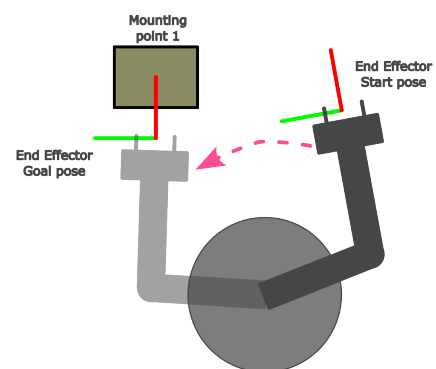
**Grasping mounting points:** Developing precision grasping skills as visualized in Figure 5.6.

**Moving the robot base:** Introducing the possibility to move the base in microgravity as visualized in Figure 5.7.

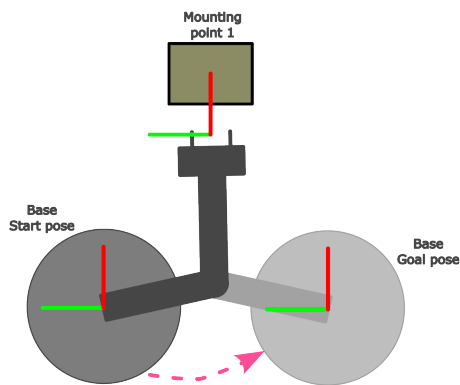
**Transitioning between multiple grasp points to reach a final goal position:** Integrating navigation, grasping, and manipulation skills for complex task completion as visualized in Figure 5.8.



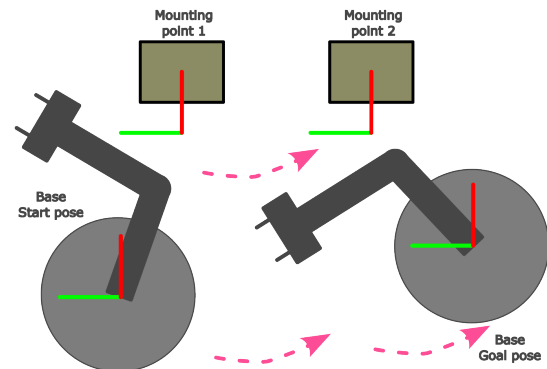
**Figure 5.5:** Task 1 - Point-to-Point movements.



**Figure 5.6:** Task 2 - Grasping mounting points.



**Figure 5.7:** Task 3 - Moving the robot base.



**Figure 5.8:** Task 4 - Transitioning between multiple grasp points to reach a final goal position.

This sequence ensures a logical progression from basic motor skills to advanced, coordinated task execution, aligning closely with the requirements defined for system acceptance.

The rewards described in this section will be ordered as used for these tasks and are all designed to return values between  $-1$  and  $1$ . However, these are not the final values used by the policy. Each reward is scaled by a weight that reflects its importance, and these weights change as the task becomes more complex. This adjustment process follows a concept known as curriculum learning, which helps guide the robot through increasingly difficult challenges. The details of the curriculum learning strategy and the specific reward weights are provided in Section 5.5.

## 5.4.1 Point-to-Point Movement

To start, the system must learn how to move the manipulator. It is given a Cartesian point within its reachable workspace and must move its end effector to that target. The episode begins with the manipulator in its home position, after which it must reach the designated goal point.

To successfully complete this task, the following reward components are implemented:

**End effector to Mounting Point Reward:** This reward function combines two equations, an exponentially decaying potential based on the distance between the end effector and the goal position, and a linearly decaying reward both measured in the robot's base frame. The resulting sum is a function in which the closer the end effector is to the goal, the higher the reward, with a maximum value of  $1$ . The reason for combining these two functions is to benefit from the strong initial reward signal provided by the linear component, which encourages early progress. The exponential component becomes more influential as the end effector nears the goal, enhancing reward precision in the final approach. By scaling each term to contribute a maximum of  $0.5$ , their sum forms a smooth reward curve that peaks at  $1.0$  when the end effector reaches the target. The combined equation and plots can be seen in Equation 5.1 and Figure 5.9 respectively.

$$r_{\text{goal}} = \frac{1}{2} \exp\left(-\frac{\|\mathbf{x}_{\text{ee}} - \mathbf{x}_{\text{goal}}\|}{\sigma}\right) + \frac{1}{2} \cdot \max\left(0, 1 - \frac{\|\mathbf{x}_{\text{ee}} - \mathbf{x}_{\text{goal}}\|}{d_{\text{max}}}\right) \quad (5.1)$$

Where:

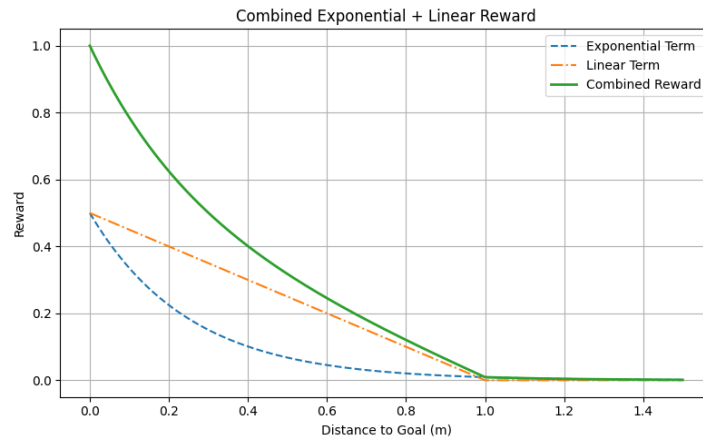
$\sigma$  is the smoothing factor for the exponential decay

$d_{\text{max}}$  is the maximum effective distance for the linear term

$r_{\text{goal}}$  is the total reward value based on distance

$\mathbf{x}_{\text{ee}}$  is the current position of the end effector

$\mathbf{x}_{\text{goal}}$  is the goal position for the end effector



**Figure 5.9:** End-effector Distance to Goal Reward function with  $\sigma = 0.25$ .

**Joint Velocity Minimization:** This reward term, as seen in Equation 5.2, penalizes high joint velocities in the robot, encouraging smoother and more controlled movements.

$$r_{\text{vel}} = -\sum_{i=1}^n \dot{q}_i^2 \quad (5.2)$$

Where:

$\dot{q}_i$  is the velocity of joint  $i$

$n$  is the total number of joints

$r_{\text{vel}}$  is the joint velocity penalty

**Action Rate Penalty:** This reward, shown in Equation 5.3, penalizes large changes in actions between time steps, promoting smoother and more stable control.

$$r_{\text{action}} = -\sum_{i=1}^m (a_i - a_i^{\text{prev}})^2 \quad (5.3)$$

Where:

$a_i$  is the current action for dimension  $i$

$a_i^{\text{prev}}$  is the action from the previous time step

$m$  is the dimensionality of the action space

$r_{\text{action}}$  is the action smoothness penalty

**Self-Collision Penalty:** This term, defined in Equation 5.4, penalizes the robot when multiple self-collisions occur within a single step. It helps prevent unsafe or physically implausible configurations.

$$r_{\text{collision}} = \begin{cases} -1 & \text{if } c \geq c_{\min} \\ 0 & \text{otherwise} \end{cases} \quad (5.4)$$

Where:

$c$  is the number of self-collisions

$c_{\min}$  is the threshold for penalization

$r_{\text{collision}}$  is the self-collision penalty

**Time Step Penalty:** This constant penalty, shown in Equation 5.5, encourages faster task completion by penalizing long episode durations.

$$r_{\text{time}} = -1 \quad (5.5)$$

## 5.4.2 Grasping mounting points

The next task is to teach the system to grasp mounting points. All in all, it is similar to the last test, however, it adds a new action where the system needs to learn how to grasp mounting points. Currently, the system knows how to move to arbitrary goal points in its environment, but now it will be taught how to approach specified mounting points. This entails that the system learns how to reach the designated mounting points, as well as how to finish the task by grabbing and holding it until the episode ends.

In addition to the previous reward components, the grasping task introduces the following rewards:

**Grasp Detection:** This binary reward function, shown in Equation 5.6, evaluates whether the robot is likely grasping a target object. It returns a reward of 1 only when the gripper is closed and the end-effector is within a small distance threshold from the mounting point. This estimation needs to be checked due to the surface gripper being a closed source extension in Isaac Sim, which prevents access to information about the object it is gripping.

$$r_{\text{grasp}} = \begin{cases} 1 & \text{if gripper is closed} \wedge \|\mathbf{x}_{\text{ee}} - \mathbf{x}_{\text{mp}}\| < \tau \\ 0 & \text{otherwise} \end{cases} \quad (5.6)$$



Where:

$\mathbf{x}_{ee}$  is the position of the end-effector

$\mathbf{x}_{mp}$  is the position of the mounting point

$\tau$  is the grasping distance threshold (e.g. 0.05m)

$r_{grasp}$  is the grasp detection reward

**Cylinder Self-Grasp Penalty:** This penalty function, shown in Equation 5.7, discourages the robot from accidentally gripping its own base-mounted cylindrical structure. It applies a negative reward when the secondary gripper is closed while the end-effector is within a small threshold distance from the surface of the cylinder. This helps prevent physically infeasible or undesired grasps that interfere with locomotion or stability.

$$r_{cyl} = \begin{cases} \text{penalty} & \text{if gripper is closed} \wedge \text{dist}_{\text{surf}} < \tau \\ 0 & \text{otherwise} \end{cases} \quad (5.7)$$

Where:

penalty is a negative constant (e.g.,  $-1.0$ ) applied when the condition is met

$\text{dist}_{\text{surf}}$  is the shortest distance from the end-effector to the surface of the cylinder

$\tau$  is the threshold distance for self-collision (e.g., 0.02m)

$r_{cyl}$  is the self-grasp penalty reward component

### 5.4.3 Base movement

This task adds another movement option to the system's navigation capabilities. Here, the system must learn how to move its base from one goal position to another. Specifically, the manipulator will be tasked with moving the base, which rests on a frictionless floor, by interacting with mounting points. The manipulator holds onto a mounting point, enabling it to reposition the base. Upon reaching the goal, the base must lock onto the floor to be ready for the following task.

In addition to the previously established reward components for basic movements, the base movement task introduces these additional rewards:

**Move While Holding Reward:** This reward, defined in Equation 5.8, provides an incentive to move the robot's base toward the goal location, but only while holding the target mounting point. The reward function for the distance is the same as the exponential one defined in Equation 5.1, with the difference that the  $\sigma$  is 1.5, making the plot increase more gradually, as can be seen in Figure 5.10

$$r_{\text{carry}} = \gamma_{\text{grasp}} \cdot \exp\left(-\frac{\|\mathbf{x}_{\text{base}} - \mathbf{x}_{\text{goal}}\|}{\sigma}\right) \quad (5.8)$$

Where:

$\gamma_{\text{grasp}}$  is 1 if the mounting point is currently being held, 0 otherwise

## CHAPTER 5. IMPLEMENTATION

$\sigma$  is the shaping factor

$\mathbf{x}_{\text{base}}, \mathbf{x}_{\text{goal}}$  are positions of the robot base and the goal respectively

$r_{\text{carry}}$  is the conditional goal proximity reward



**Figure 5.10:** Base Distance to Goal Reward function with  $\sigma = 1.5$ .

**Base Docking Reward:** This reward, defined in Equation 5.9, encourages the second gripper to close when near a designated target, for example, a docking point to reach the mounting point or its final goal position. A smooth exponential term provides shaping at all distances, while a bonus is added for successful docking (i.e., closing within a threshold radius). Conversely, a penalty is applied if the gripper closes while too far away, guiding precise control behavior.

$$r_{\text{dock}} = \exp\left(-\frac{\|\mathbf{x}_{\text{base}} - \mathbf{x}_{\text{target}}\|}{\sigma}\right) + b \cdot \gamma_{\text{closed}} \cdot \gamma_{\text{near}} + p \cdot \gamma_{\text{closed}} \cdot (1 - \gamma_{\text{near}}) \quad (5.9)$$

Where:

$\sigma$  controls the width of the approach reward

$b$  is the close-distance bonus,  $p$  is the far-distance penalty

$\gamma_{\text{closed}}, \gamma_{\text{near}}$  are binary flags indicating gripper state and proximity to the target

$r_{\text{dock}}$  is the gripper docking reward

**Simultaneous Gripper Open Penalty:** As shown in Equation 5.10, this penalty discourages leaving both grippers open at the same time for too long. After a short grace period (defined in steps), the agent receives a fixed penalty if both grippers remain open simultaneously, which should be avoided, as due to the frictionless base, the robot can begin sliding forever. The grace period is there so that the robot can spawn in naturally.

$$r_{\text{simul}} = \begin{cases} -1 & \text{if both grippers are open for more than } T_{\text{grace}} \\ 0 & \text{otherwise} \end{cases} \quad (5.10)$$

Where:

$T_{\text{grace}}$  is the allowed number of steps before penalizing

$r_{\text{simul}}$  is the penalty for prolonged inaction of both grippers

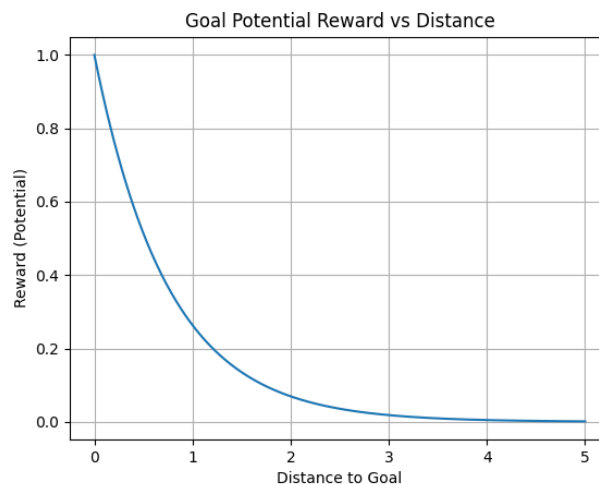
### 5.4.4 Traversal task

All previous tasks lead to this final task. In this scenario, the system spawns at a starting location and must navigate to a goal position by traversing two mounting points. The final goal position is represented by a docking point that the robot can only reach by grabbing the second mounting point. For this task, some of the rewards applied are the same as the ones stated before for previous tasks, but this time applied to the following grappling points:

- End Effector to Grappling Point Reward
- Grasp Detection Reward
- Carry While Holding Reward
- Base Docking Reward

However, there are still rewards specifically added to enhance the behaviour of the robot for this final task:

**Base distance to goal:** This reward follows the same equation as the exponential distance to goal reward in the point to point task Equation 5.1 and the one used in Equation 5.8. Similarly to Equation 5.8 computes the distance between the base and the overall goal, with the difference being that the value for  $\sigma$  is 0.75 so leaving it in between of the two previous plots, meaning that the reward is not as significant in the beginning but increases quickly when it gets to the middle of the task, the plot can be seen in Figure 5.11.



**Figure 5.11:** Base Distance to Goal Reward function with  $\sigma = 0.75$ .

**Hold Far Mounting Point Penalty:** This reward, shown in Equation 5.11, penalizes the robot for holding the mounting point that is farther from the goal during the base positioning phase. It also provides a bonus for releasing the mounting point within a designated "drop zone" near

## CHAPTER 5. IMPLEMENTATION

the target. A short warm-up period at the start of each episode disables both the penalty and bonus to avoid interfering with early exploration.

This reward addresses one of the most challenging parts of the task: learning the transition from grabbing one a mounting point and switching to the next one based on their relevance to the overall goal and the position of the base. Initially, the robot learns to grasp a mounting point to receive a reward, but at some point, it must learn that releasing the currently held (farther) point and moving toward the closer one is more beneficial. By combining this penalty with positive rewards for grasping the closer mounting point and approaching the final goal, the robot is guided toward discovering the importance of switching behavior at the right time.

$$r_{\text{far}} = -\lambda \cdot \gamma_{\text{wrong}} \cdot \sigma(d_{\text{zone}}) + \beta \cdot \gamma_{\text{released}} \cdot \sigma(d_{\text{zone}}) \quad (5.11)$$

Where:

$\lambda$  is the penalty scale

$\beta$  is the release bonus

$\gamma_{\text{wrong}}$  is 1 if the robot is holding the mounting point farther from the goal, 0 otherwise

$\gamma_{\text{released}}$  is 1 if neither mounting point is being held, 0 otherwise

$\sigma(d_{\text{zone}})$  is a smooth step (sigmoid) based on distance to the drop zone

$d_{\text{zone}}$  is the distance between the robot base and the designated drop zone

$r_{\text{far}}$  is the combined penalty/bonus reward

The smooth zone weight is activated only after a warm-up period of  $T_{\text{warmup}}$  steps (e.g., 150). This helps PPO agents detect meaningful gradients during early training.

**Wall Proximity Penalty:** This reward, defined in Equation 5.12, penalizes the robot when its end-effector moves too close to the wall and grasps it. This penalty is introduced to address an unintended behavior caused by the simplification from the physical gripper to the surface gripper, which allows it to grip and pull directly against the wall. This is undesirable, as the robot is intended to move only by interacting with designated mounting points. To discourage this, the reward function checks if the gripper is closed and the end-effector is within a certain distance from the wall. If so, a penalty is applied to prevent the robot from learning to use the wall as a support for movement. As previously explained, this estimation is required due to the surface gripper not being able to export what object it is grabbing.

$$r_{\text{wall}} = \begin{cases} -1 & \text{if gripper is closed} \wedge 0 < y_{\text{wall}} - y_{\text{ee}} < \tau \\ 0 & \text{otherwise} \end{cases} \quad (5.12)$$

Where:

$y_{\text{wall}}$  is the front Y face of the wall (e.g., center + 0.25 m)

$y_{\text{ee}}$  is the end-effector's Y coordinate

$\tau$  is the proximity threshold (e.g., 0.25 m)

$r_{\text{wall}}$  is the wall proximity penalty

**Mount Affinity Reward:** As shown in Equation 5.13, this reward encourages the robot to

move toward the mounting point only when it is not already holding it. It is combined with the distance-based reward from Equation 5.1, so the robot only receives this reward when it is not grasping the target. This helps avoid getting a high reward just by staying close to a point it is already holding and encourages letting go when getting close to the next mounting point. It as well turns the reward off when the robot has reached the docking point as to make sure that the model is not getting rewards for the mounting point it used to reach the docking point and can focus in reaching the next one.

$$r_{\text{affinity}} = r_{\text{goal}} \cdot (1 - \gamma_{\text{grasp}}) \cdot \gamma_{\text{dock}} \quad (5.13)$$

Where:

$r_{\text{goal}}$  is the reward function explained in Equation 5.1

$\gamma_{\text{grasp}}$  is 1 if the mounting point is currently grasped, 0 otherwise

$\gamma_{\text{dock}}$  is 1 if the robot is in the docking zone, 0 otherwise

$r_{\text{affinity}}$  is the shaping reward encouraging proximity to unheld targets

**Let Go Bonus:** This event-based reward gives a one-time bonus when the robot opens its gripper near a mounting point. A cooldown mechanism prevents the reward from triggering repeatedly, allowing it only once every set number of steps. This reward is designed to encourage the release of the first mounting point, as being rewarded for grasping it can make it difficult for the robot to transition to the next one.

$$r_{\text{open}} = \rho \cdot \gamma_{\text{open\_edge}} \cdot \gamma_{\text{near}} \cdot \gamma_{\text{cooldown}} \quad (5.14)$$

Where:

$\rho$  is the reward value for a valid open action

$\gamma_{\text{open\_edge}}$  is 1 if the gripper just opened (rising edge), 0 otherwise

$\gamma_{\text{near}}$  is 1 if the robot is within a threshold distance of the marker, 0 otherwise

$\gamma_{\text{cooldown}}$  is 1 if cooldown is inactive, 0 otherwise

$r_{\text{open}}$  is the reward for opening near the target

**Sequential Docking Reward:** This reward is structured in two parts. First, it encourages grasping the target object within a specified docking zone. Second, once grasped, it rewards the robot for moving its base toward the final goal position. This shaping function ensures that the robot learns both where and when to grasp, and to move purposefully toward the goal after grasping.

$$r_{\text{dock\_goal}} = \underbrace{(r_{\text{base}} + \alpha \cdot \gamma_{\text{closed}} \cdot \gamma_{\text{near}} - \beta \cdot \gamma_{\text{closed}} \cdot (1 - \gamma_{\text{near}})) \cdot \gamma_{\text{dock}}}_{\text{Docking Reward}} + \underbrace{\delta \cdot \exp\left(-\frac{d_{\text{goal}}}{\sigma_g}\right) \cdot \gamma_{\text{closed}}}_{\text{Goal Movement Reward}} \quad (5.15)$$

Where:  $\alpha$  is the bonus for grasping near the object

$\beta$  is the penalty for grasping far from the object

$\delta$  is the scaling factor for goal movement reward  
 $\gamma_{\text{closed}}$  is 1 if the gripper is closed, 0 otherwise  
 $\gamma_{\text{near}}$  is 1 if the end effector is near the object, 0 otherwise  
 $\gamma_{\text{dock}}$  is 1 if the robot is in the docking zone, 0 otherwise  
 $r_{\text{base}} = \exp\left(-\frac{d_{\text{ee-cube}}}{\sigma_b}\right)$  is the base reward for proximity  
 $d_{\text{goal}}$  is the base distance to the goal  
 $\sigma_g$  and  $\sigma_b$  are smoothing factors for exponential rewards  
 $r_{\text{dock\_goal}}$  is the total reward for this behavior

Taking each of the rewards from this section and putting them together creates the overall reward structure seen in Equation 5.16. This however is not the final reward used to create the model as a curriculum is applied to it as well which will be discussed in the next section.

$$\begin{aligned}
 R_{\text{overall}} = & \underbrace{r_{\text{goal}} + r_{\text{vel}} + r_{\text{action}} + r_{\text{collision}} + r_{\text{time}}}_{\text{Point-to-Point Movement}} \\
 & + \underbrace{r_{\text{grasp}} + r_{\text{cyl}}}_{\text{Grasping Mounting Points}} \\
 & + \underbrace{r_{\text{carry}} + r_{\text{dock}} + r_{\text{simul}}}_{\text{Base Movement}} \\
 & + \underbrace{r_{\text{base\_goal}} + r_{\text{far}} + r_{\text{wall}} + r_{\text{affinity}} + r_{\text{open}} + r_{\text{dock\_goal}}}_{\text{Traversal Task}}
 \end{aligned} \tag{5.16}$$

## 5.5 Curriculum Learning

As mentioned previously in the reward section, the weights for each reward were not specified. This is due to these weights not being fixed, instead, they follow a common RL strategy called curriculum learning. In this approach, the importance of each reward evolves as the task progresses, which is especially useful for mastering complex tasks.

An abrupt, step-wise change in the reward signal can destabilize training; for this reason, an intermediate "fade" phase is introduced. During this phase, the most relevant rewards are progressively ramped up (or down), rather than toggled on or off at a single time-step. The numbers shown in the "Fade" column of Table 5.1 therefore represent mid-curriculum values; they are not meant to capture every incremental update, but they illustrate the overall trend as showing each change is redundant.

## CHAPTER 5. IMPLEMENTATION

Reward Name	Equation	P2P	Grasp	Base	Fade	Trav.
Mount Affinity 1	$r_{\text{affinity1}} = \phi(d) \cdot (1 - \gamma_{\text{grasp}})$	1	1	1	0.2	0.2
Grasp Detection 1	$r_{\text{grasp1}} = \begin{cases} 1 & \text{if MP1 grasped} \\ 0 & \text{otherwise} \end{cases}$	1	1	1	0.4	0.4
Move-While-Holding 1	$r_{\text{drag1}} = \gamma_{\text{grasp}} \cdot \exp\left(-\frac{\ \mathbf{x}_{\text{base}} - \mathbf{x}_{\text{goal}}\ }{\sigma}\right)$	0	1	1	0.4	0
Base Docking	$r_{\text{dock}} = \begin{cases} 1 & \text{if docked} \\ 0 & \text{otherwise} \end{cases}$	0	0	2	1	0
Let Go Bonus	$r_{\text{open}} = \rho \cdot \gamma_{\text{open\_edge}} \cdot \gamma_{\text{near}} \cdot \gamma_{\text{cooldown}}$	4	4	5	5	5
Sequential Docking Reward	$r_{\text{dock\_goal}} = (\dots) + \delta \cdot \exp(-d_{\text{goal}}/\sigma_g) \cdot \gamma_{\text{closed}}$	0	0	0	1	2
Mount Affinity 2	$r_{\text{affinity2}} = \phi(d) \cdot (1 - \gamma_{\text{grasp}})$	0	0	2	4	6
Grasp Detection 2	$r_{\text{grasp2}} = \begin{cases} 1 & \text{if MP2 grasped} \\ 0 & \text{otherwise} \end{cases}$	0	0	0	7	12
Move-While-Holding 2	$r_{\text{drag2}} = \gamma_{\text{grasp}} \cdot \exp\left(-\frac{\ \mathbf{x}_{\text{base}} - \mathbf{x}_{\text{goal}}\ }{\sigma}\right)$	0	0	0	5	20
Hold Far MP Penalty	$r_{\text{far}} = \begin{cases} -1 & \text{if holding far MP} \\ +1 & \text{if holding near MP} \\ 0 & \text{otherwise} \end{cases}$	0	0	0	6	2
Goal Affinity	$r_{\text{goal}} = \exp\left(-\frac{\ \mathbf{x}_{\text{base}} - \mathbf{x}_{\text{goal}}\ }{\sigma}\right)$	0	0	0	10	20
Wall Proximity Penalty	$r_{\text{wall}} = \begin{cases} -1 & \text{if near wall} \\ 0 & \text{otherwise} \end{cases}$	1	1	1	1	6
Cylinder Self-Grasp Penalty	$r_{\text{cyl}} = \begin{cases} -1 & \text{if self-grasp} \\ 0 & \text{otherwise} \end{cases}$	2	2	2	2	1.5
Simultaneous Grip Penalty	$r_{\text{simul}} = \begin{cases} -1 & \text{if both grippers open} \\ 0 & \text{otherwise} \end{cases}$	2	2	2	2	2
Joint Velocity Penalty	$r_{\text{vel}} = -\sum_i \dot{q}_i^2$	0.001	0.001	0.001	0.001	0.001
Action Rate Penalty	$r_{\text{action}} = -\sum_i (a_i - a_i^{\text{prev}})^2$	0.001	0.001	0.001	0.001	0.001
Self-Collision Penalty	$r_{\text{collision}} = \begin{cases} -1 & \text{if collision} \\ 0 & \text{otherwise} \end{cases}$	0.2	0.2	0.2	0.2	0.2
Time-Step Penalty	$r_{\text{time}} = -1$	0.01	0.01	0.01	0.01	0.01

**Table 5.1:** Reward components, their analytical expressions, and curriculum weights. “Trav.” = Traversal stage. ”P2P” = Point-to-Point. The last two rows are newly added shaping terms.

By taking each of the weighted rewards from Table 5.1 and adding them to the overall reward equation (Equation 5.16) an extended reward equation can be created as seen in Equation 5.17. In this equation each sub-task of equation is shown as it is introduced in the curriculum.

The total reward  $R_{\text{total}}$  for each task is a weighted sum of the individual reward components relevant to that task. Denoting the weights as  $w_i$  and the rewards as  $r_i$ , the total reward for each task is:

$$\begin{aligned}
R_{\text{total}} = & \underbrace{w_{\text{goal}} \cdot r_{\text{goal}} + w_{\text{vel}} \cdot r_{\text{vel}} + w_{\text{action}} \cdot r_{\text{action}} + w_{\text{collision}} \cdot r_{\text{collision}} + w_{\text{time}} \cdot r_{\text{time}}}_{\text{Point-to-Point Movement}} \\
& + \underbrace{w_{\text{affinity1}} \cdot r_{\text{affinity1}} + w_{\text{grasp1}} \cdot r_{\text{grasp1}} + w_{\text{cyl}} \cdot r_{\text{cyl}}}_{\text{Grasping Mounting-Point 1}} \\
& + \underbrace{w_{\text{drag1}} \cdot r_{\text{drag1}} + w_{\text{dock}} \cdot r_{\text{dock}} + w_{\text{simul}} \cdot r_{\text{simul}}}_{\text{Base Movement}} \\
& + \underbrace{w_{\text{far}} \cdot r_{\text{far}} + w_{\text{wall}} \cdot r_{\text{wall}} + w_{\text{open}} \cdot r_{\text{open}}}_{\text{Transition to MP 2}} \\
& + \underbrace{w_{\text{affinity2}} \cdot r_{\text{affinity2}} + w_{\text{grasp2}} \cdot r_{\text{grasp2}} + w_{\text{goal\_move}} \cdot r_{\text{goal\_move}} + w_{\text{drag2}} \cdot r_{\text{drag2}} + w_{\text{dock\_goal}} \cdot r_{\text{dock\_goal}}}_{\text{MP 2 \& Final Docking}}
\end{aligned} \tag{5.17}$$

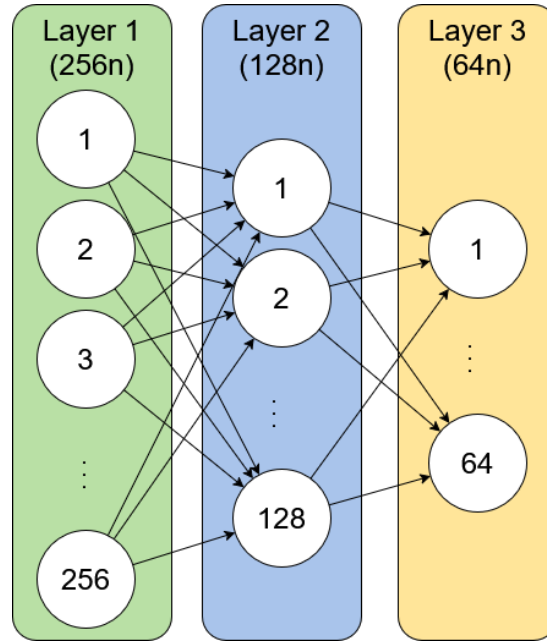
## 5.6 Neural Network Architecture in PPO

In traditional RL, the policy and value functions are often represented using simple tables or mathematical functions. However, this approach doesn't work well for complex environments with large or continuous state and action spaces. This is where DRL comes in. In DRL, neural networks are used as function approximators. Instead of trying to store or calculate values for every possible state or action, the neural network learns to generalize from data. This allows the agent to operate in environments where the state and action spaces are too large to handle otherwise. Neural networks play a key role when using PPO for DRL, being used to learn both the policy and the value function. These networks are defined in the configuration file using the SKRL library.

**Policy Network:** This network is responsible for outputting actions given the current state. It uses a Gaussian policy, which means it outputs a distribution over possible actions, rather than a single action. This allows the agent to explore the environment by sampling from this distribution. The output of this network is the action vector.

**Value Network:** This network estimates the expected return (or value) of a given state. The output is a single scalar value representing the state's value.





**Figure 5.12:** Visualization of a 3-layered fully connected neural network. This version has 256, 128, and 64 nodes in the layers, respectively.

**Network Architecture:** Both neural networks follow a narrowing architecture as visualized in Figure 5.12. Each successive layer has fewer neurons than the previous one: 256 in the first layer, 128 in the middle layer, and 64 in the third one. This design allows the network first to capture a wide range of patterns and feature interactions, then gradually focus on the most relevant information, compressing it into a compact representation suitable for decision-making or value estimation. By narrowing the layers, the model reduces the risk of overfitting and improves training efficiency, while still maintaining enough capacity to handle the complexity of the task.

**Activation Function:** The activation function, as seen in Equation 5.18, used in both neural networks is the Exponential Linear Unit (ELU) function, defined as:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad \text{with } \alpha > 0 \quad (5.18)$$

Unlike simpler functions, ELU allows negative outputs, which leads to more balanced, zero-centered activations. This property helps improve the stability and convergence of gradient-based learning methods. PPO relies on stable and consistent gradient updates for both the policy and value networks. ELU’s smoothness and ability to preserve gradient information help reduce training instabilities and contribute to better performance and faster convergence. This makes ELU a well-suited choice for the neural networks used in this PPO-based implementation.

## 5.7 Training Parameters

In RL, training parameters encompass both algorithm-specific hyperparameters and settings related to the training environment or hardware infrastructure. These parameters are defined before the learning process begins, and they directly influence how the agent explores its environment, updates its policy, and balances immediate versus long-term rewards. Their tuning is a critical part of achieving stable and effective learning.

Below are the key parameters used in our RL setup and their roles in shaping the training process:

### 5.7.1 PPO Hyperparameters

**Discount Factor** ( $\gamma = 0.99$ ): Balances the importance of future rewards. A value close to 1 encourages long-term planning, while a lower value makes the agent prioritize immediate rewards.

**GAE Lambda** ( $\lambda = 0.95$ ): Controls how short-term and long-term information are blended when estimating the advantage function. A lower  $\lambda$  means the agent relies more on immediate, short-term feedback to estimate how good an action was, leading to lower variance but higher bias. A higher  $\lambda$  incorporates more future information into the advantage estimate, which reduces bias but increases variance.

**Learning Rate** ( $1 \times 10^{-4}$ ): Controls the size of the steps taken when updating the model's parameters during training. It does not directly influence how rewards are processed but determines how quickly or cautiously the neural network learns from the computed loss. A high learning rate can lead to unstable training and divergence, while a very low learning rate results in slow progress.

**KL-based Learning Rate Scheduler** ( $threshold = 0.04$ ): Monitors the Kullback–Leibler (KL) divergence, which measures how much the updated policy deviates from the old policy after an update. If this divergence exceeds the threshold, the scheduler reduces the learning rate to prevent overly large policy updates that could destabilize training. On the other hand, if the divergence is small, it increases the learning rate to speed up learning.

**Rollouts** (24): Defines the number of full episodes the agent collects before each policy update. These episodes are used as training data to compute gradients and update the policy. Collecting more rollouts means the agent updates its policy with more experience, improving stability and reducing variance in the updates. However, it also increases the time and memory needed before each update step. To clarify, when running with multiple environments in parallel, these rollouts are collected across all environments, not per environment. For example, with 12 environments, 24 rollouts could be collected in just 2 episode rounds.

**Mini-Batches** (8): After collecting rollouts, the gathered data, including states, actions, rewards, and advantages from all episodes, is first flattened into a single dataset and then randomly



shuffled. This mixing breaks correlations between consecutive time steps, which helps improve generalization and stabilize training. The shuffled dataset is then split into smaller parts called mini-batches. Instead of updating the model using all the data at once, the model is updated multiple times using these smaller, randomized chunks, which enhances learning efficiency and stability.

**Learning Epochs (8):** Specifies how many times the model goes through the entire collected and shuffled rollout data during a single update. Each epoch processes all mini-batches once, updating the model after each mini-batch. Multiple epochs help the agent learn better from the data, but too many can cause overfitting, making the agent less able to generalize to new situations.

**Ratio Clip (0.2):** This parameter limits how much the policy's probability ratios between the new and old policies can change during each update. By clipping these ratios, it prevents excessively large policy updates that could destabilize training. This clipping acts as a hard constraint on policy changes, complementing the adaptive learning rate adjustments driven by the KL divergence scheduler.

**Value Clip (0.2):** Similar to ratio clipping but applied to the value function. It restricts the amount by which the predicted value estimates can change during an update. This helps maintain stable learning of the value function by avoiding large, sudden shifts that can harm training stability.

### 5.7.2 Isaac Lab Parameters

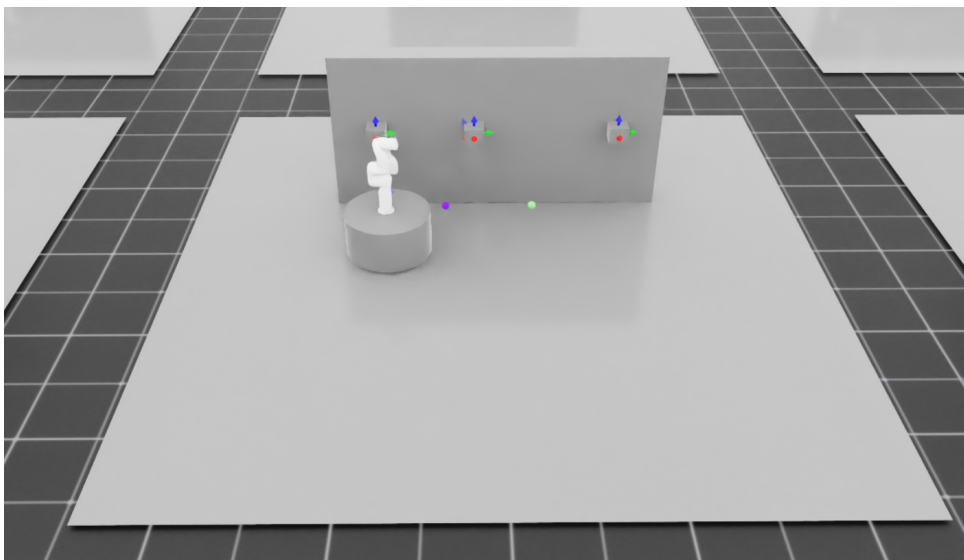
**Episode length (7 seconds):** Maximum number of steps an agent can take in a single episode before the environment resets. A longer episode allows the agent to interact with the environment for a longer duration, making it possible to learn more complex behaviors. In contrast, shorter episodes lead to quicker resets and more frequent learning cycles but may limit the agent's ability to learn long-term strategies.

**Total Timesteps (60.000):** Total number of environment steps the training process will execute across all environments. It defines the overall duration of training.

**Number of Environments (528):** The total number of environments running in parallel during training. More environments allow the agent to collect experience faster, which can speed up learning. However, using many environments also increases the computational load. In Isaac Lab, environments run fully in parallel, which means that one environment can reset while others continue running, making the training process more efficient.

## 6 Testing

This chapter presents the results of the acceptance tests used to evaluate the robotic system against the requirements defined in Section 3.4, following the procedures described in Section 3.5. The tests were carried out in the environment introduced in the Concept Design Chapter 4, with additional visual goal markers: the small purple circle indicates the intermediate target for phase 3 also refereed to as the docking point, while the green circle represents the overall goal for the full task, these can be seen in Figure 6.1.



**Figure 6.1:** Simulation Environment in Isaac Lab ready for training.

Each test begins with a summary of its objective, followed by key observations based on the WANDB output. The tests are structured to evaluate both isolated capabilities and full system performance through increasingly complex scenarios. The results section is organized to first present relevant reward curves for each training phase, followed by an analysis of the final model performance, including total reward plots and key policy statistics and hyperparameters.

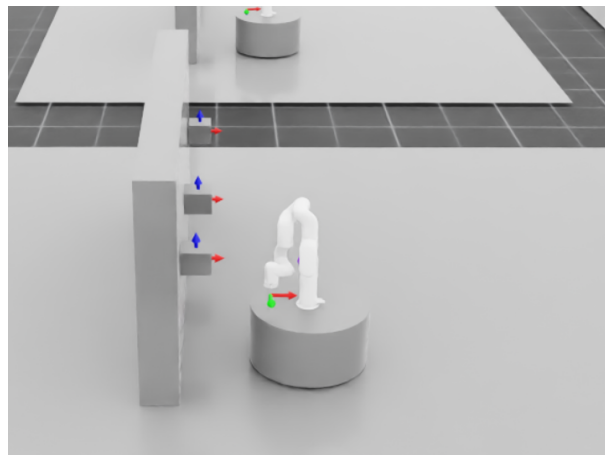
Although each acceptance test has a defined start and end, the PPO training process benefits from smooth transitions between phases. For this reason, some reward components are introduced slightly before they become critical and may continue to be present throughout training. This leads to overlapping reward signals between phases, helping the agent adapt more effectively to new behaviors without abrupt changes. A link to a video of the agent running after training can be seen in Section A.1. Additionally, a complete collection of the training graphs and logs analyzed in this report, exported from WANDB, is available via the link in Section A.1

## 6.1 Test 1: Point-to-Point Motion Control (0-3500 timesteps)

This test evaluates the robot's ability to control the manipulator and move the end-effector to a designated goal pose, as described in Section 3.5.2.

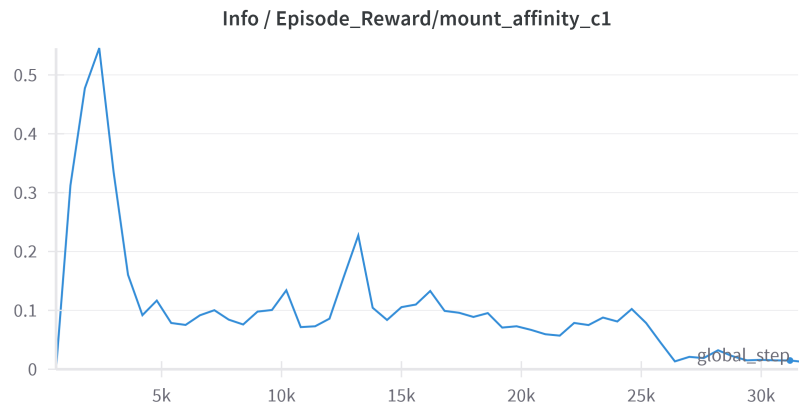
The test is considered successful if the manipulator reaches the goal pose within a 30-second time frame.

At the beginning of the first phase, the robot starts in an initial position, as visualized in Figure 6.2, with the base unlocked. From this configuration, the model quickly learns to dock the base to the floor and begin moving the manipulator around. The main challenge for the agent is that it starts with almost no information about the model or the goal.



**Figure 6.2:** Robot in its initial simulation configuration with an unlocked base.

As shown in Figure 6.3, the agent quickly learns to move the end-effector towards the target, maintaining it at the goal for a significant portion of the time. This is reflected in the graph, which rises during the initial phase and peaks around 3,000 timesteps, just as the second task begins. After this point, the reward decreases as the robot starts to grasp the first mounting point, which deactivates the mount affinity reward.



**Figure 6.3:** Reward for getting closer to mounting point 1.

Additionally, the agent learns to minimize joint velocities (see Figure 6.4) and reduce the frequency of action changes (see Figure 6.5), both of which contribute to improved stability and reduced penalties. This is the case for phase one, as can be seen; however, as other phases get introduced, the action rate becomes more unstable.

Over the initial successive iterations, a noticeable decrease in the joint velocities and action rate, with the increase in rewards for getting closer to the given goal pose, as visualised in Figure 6.3, indicates the model's efficiency in reaching the target position.



**Figure 6.4: Joint velocity Penalty:** Penalty for having high velocities at the joints.



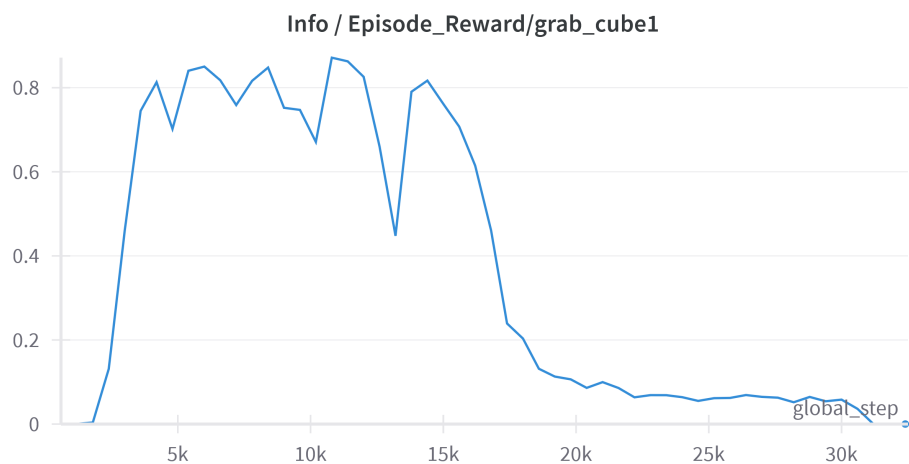
**Figure 6.5: Action rate Penalty:** Penalty for changing between actions too fast.

Out of 528 agents tested in phase 1, all successfully reached their goal pose under 30 seconds, suggesting strong generalization in the learned motion control policy. This test is thus deemed a success.

## 6.2 Test 2: Point-to-Point Grasp at Fixed mounting Points (0-8000 timesteps)

This test evaluates the agent's ability to grasp a predefined mounting point and maintain a stable connection, as described in Section 3.5.3. The task builds upon the reaching capability learned in Test 1. Here, the robot must guide the end-effector to mounting point 1 and perform a grasp.

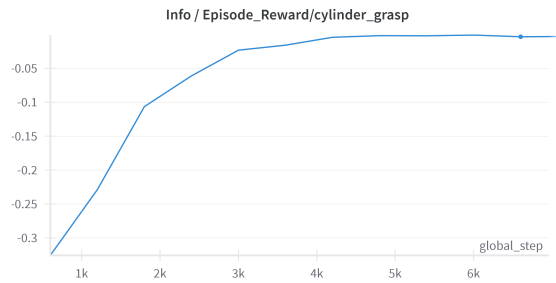
As seen in Figure 6.6, the agent quickly learns how to grasp mounting point 1 after phase 1. The graph shows the reward near instantly going to maximum as it reaches the mounting point, showing that all agents have succeeded.



**Figure 6.6: Grasp detection 1:** Reward for grabbing mounting point 1.

As soon as the agent learns that the end-effector should grasp the mounting point, there is a visible decrease in penalty for the model grasping itself, as seen in Figure 6.7. Meaning that it completely stops grasping its own cylinder. It can also be seen in Figure 6.8 that as the gripper starts grasping the mounting point 1, the simultaneous opening penalty instantly goes from -0.35 to -0.05, signifying that the agent learns to keep at least one gripper closed. Although it does gradually drop afterwards but that is the result of the addition of rewards from the new phases getting introduced.



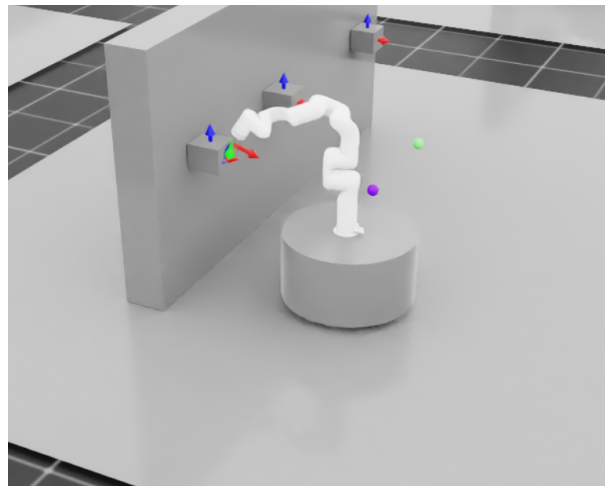


**Figure 6.7: Cylinder Self-Grasp Penalty:** Penalty for grasping the models own cylinder.



**Figure 6.8: Simultaneous Grip Penalty:** Penalty for simultaneous grasping the same mounting points in a row.

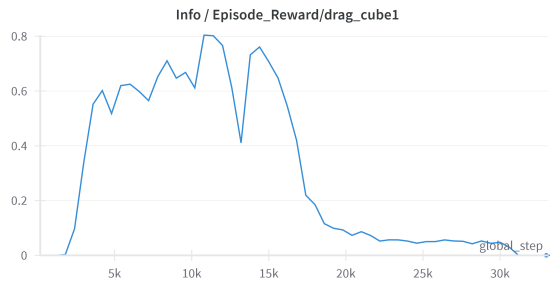
As stated, all agents learn to grasp the cube within 30 seconds, so the test is deemed a success. Figure 6.9 shows the robot grasping mounting point 1.



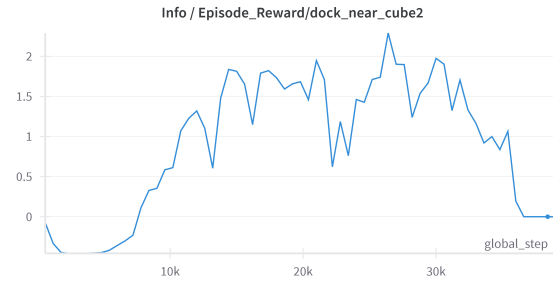
**Figure 6.9:** The robot has reached and docked to mounting point 1.

### 6.3 Test 3: Base Movement After Grasp (5000-15000 timesteps)

This test evaluates the robot's ability to maintain a stable grasp while relocating its base to a new position, as described in Section 3.5.4. Here, the robot is required to move the base from the initial position, being the final position of test 2 shown in Figure 6.9, to a new pose using the secured grasp by the end-effector on mounting point 1. The new pose is the docking point shown as the purple circle in Figure 6.1. A successful base movement is defined by detaching the base from the initial pose and re-attaching the base to the new pose while maintaining a stable grasp to the mounting point.

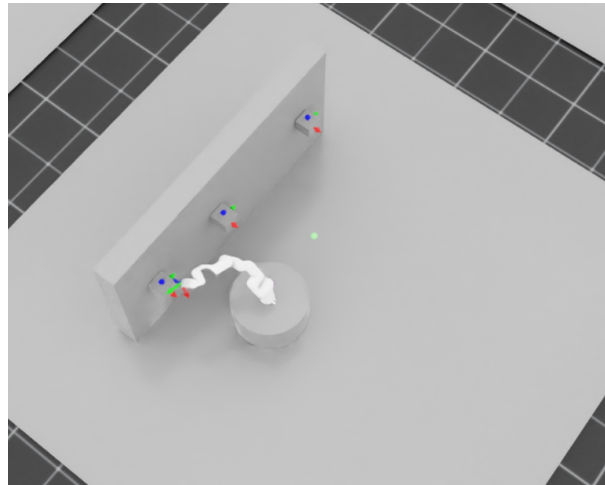


**Figure 6.10: Move-While-Holding 1:** Reward for moving the base of the system towards the docking point while holding the first mounting point.



**Figure 6.11: Base Docking:** Reward showing that the base is docked at the docking point.

As seen in Figure 6.10, the agent starts to move the base towards the docking point when it grasps the first mounting point. Looking at the Base Docking graph in Figure 6.11, the agent initially starts docking too far away from the docking point, giving it penalties. However, as the agent starts to move towards the docking point, the penalty turns into a reward, maximizing around 15000 time steps, showcasing that the agents have learned how to dock at the docking point, completing phase 3. Figure 6.12 shows the agent docked at the docking point.



**Figure 6.12:** The robot has reached and docked at the docking point on the floor.

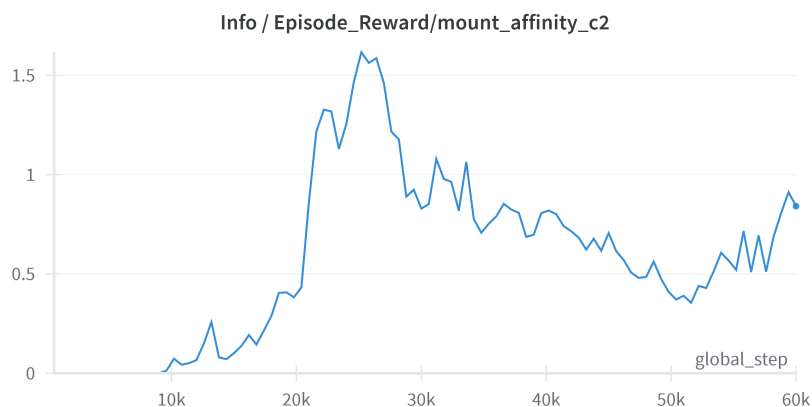
At the end of phase 3, where the rewards for phase 4 are slowly being introduced, a steep decrease in rewards for moving the base to the docking point is visible at around 18000 time steps in Figure 6.10. Since the rewards are changed to prepare the agent to move towards the new mounting point, the model is getting fewer rewards for the first mounting point.

## 6.4 Test 4: Multi-Point Grasp-Move Sequence (15000-60000 timesteps)

This test evaluates the robot’s ability to autonomously release and re-engage different grasping points during navigation, as described in Section 3.5.5. The robot is required to perform a two-phase grasp-move-grasp-move sequence, simulating transitions between support structures. Success is defined as completing both grasp-and-move phases and docking the base within a distance threshold to the overall goal without violating tolerances or losing stability.

At the beginning of this phase, the robot is holding the first mounting point, with its base docked at the intermediate goal position, as seen in the end image for phase 3. From this configuration, it is already within reach of the second mounting point. The primary challenge in this stage is to enable the robot to learn that releasing the first mounting point, while keeping the base stationary, is crucial for progressing to subsequent targets. To support this behavior, a sequence of rewards is introduced progressively.

The first reward encourages proximity to mounting point two, similar to the mechanism used in the initial task. This is implemented via the Mount Affinity 2 reward seen in Figure 6.13, which increases as the end effector approaches the second mounting point. It is gradually introduced starting around 8,000 timesteps, with its weight increasing over time. Initially, its value remains low because the robot is still gripping the first mounting point, preventing it from approaching the second one.



**Figure 6.13: Mounting Affinity 2:** Reward for getting closer to mounting point 2.

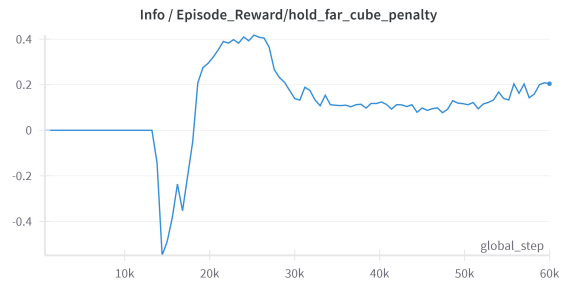
To facilitate the release of the first mounting point, additional rewards are phased in. Around 8,000 timesteps, the Let Go Bonus reward becomes active Figure 6.14, offering a one-time reward when the gripper opens near the first mounting point. Once introduced, this reward rapidly gains value, consequently, the reward for grasping the first mounting point starts to decrease, and Mount Affinity 2 continues to rise steadily. To ensure more consistent release behavior, a new reward is added around 15,000 timesteps: Hold Far Mounting Point Penalty Figure 6.15. This penalizes holding the first mounting point for too long if the robot is docked

## CHAPTER 6. TESTING

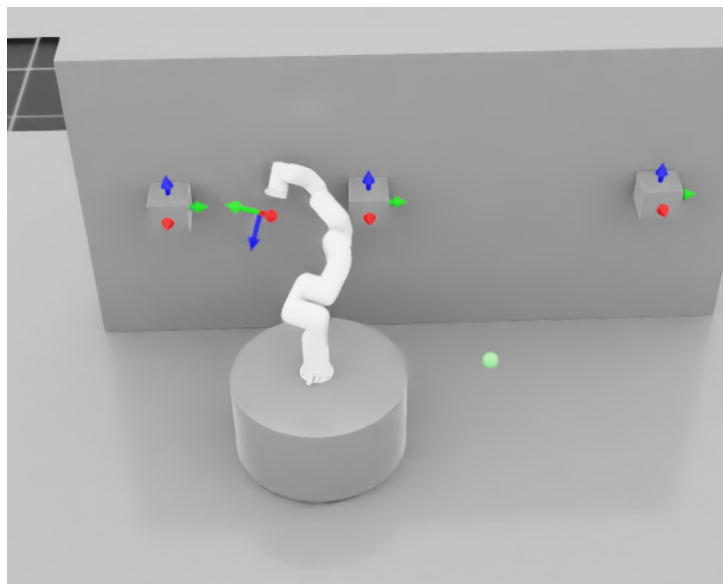
at the docking point and provides a bonus for releasing it. Together, these mechanisms help the model begin to consistently release and move toward the second mounting point, as reflected by a steady increase in the Mount Affinity 2 reward (Figure 6.13) and seen in Figure 6.16.



**Figure 6.14: Let Go Bonus:** Reward for releasing mounting point 1.

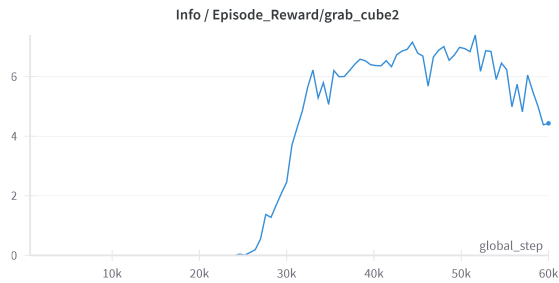


**Figure 6.15: Hold Far MP Penalty:** Penalty for holding on mounting point 1, and reward for releasing it.

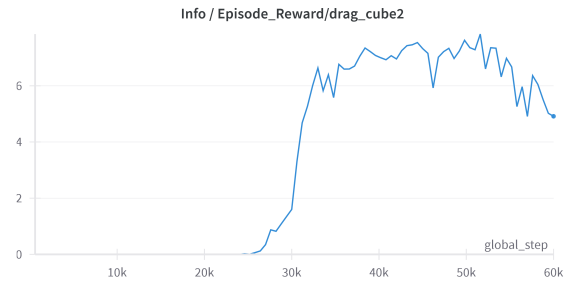


**Figure 6.16:** Mounting Point 1 successfully released.

Around 30,000 timesteps, the robot triggers the Grab Mounting Point 2 reward for the first time, a reward that was active from the start but had not yet been fulfilled. Simultaneously, the Move While Holding Mounting Point 2 reward is activated, encouraging the robot to begin base movement while grasping the second mounting point.

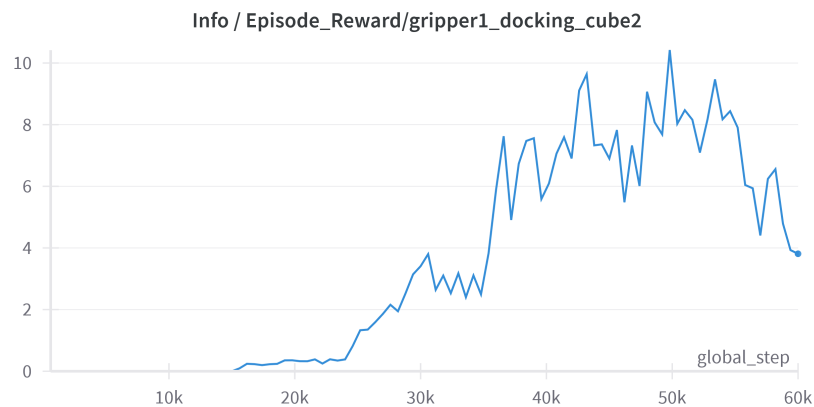


**Figure 6.17: Grasp Detection 2:** Reward for grabbing mounting point 2.

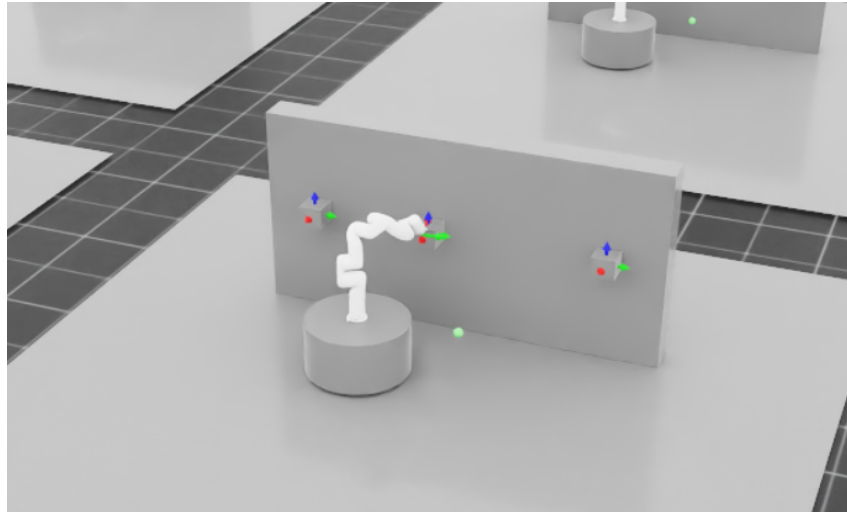


**Figure 6.18: Move-While-Holding 2:** Reward for moving the base while holding mounting point 2.

To reinforce this behavior further, the Sequential Docking reward, seen in Figure 6.19, combines incentives for both grasping cube two and progressing toward the goal. Although these rewards begin increasing significantly after the 30000 timestep mark—indicating that the robot is successfully grasping the second mounting point, it still struggles to move the base. This can be observed in Figure 6.20, where the arm is clearly attached to the mounting point, yet the base remains stationary in the same position as at the end of phase 3. The apparent rise in reward values is mainly due to most rewards including components that grant positive feedback when the end effector is gripping the cube. Coupled with the gradual increase in reward weights, this leads to the upward trends observed in the graphs. Toward the end of training, the curves become more unstable as the focus shifts from merely grasping the second mounting point to actively docking the base at the goal position.



**Figure 6.19: Sequential Docking Reward:** Reward for moving between goal positions.

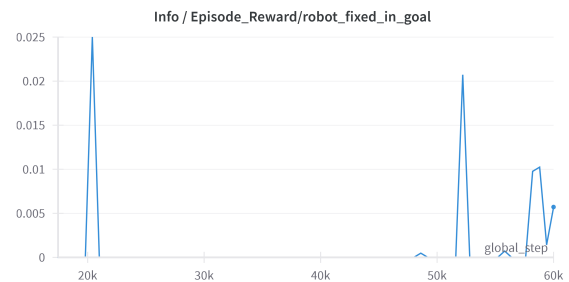


**Figure 6.20:** Managed to grab the second mounting point and start dragging the base.

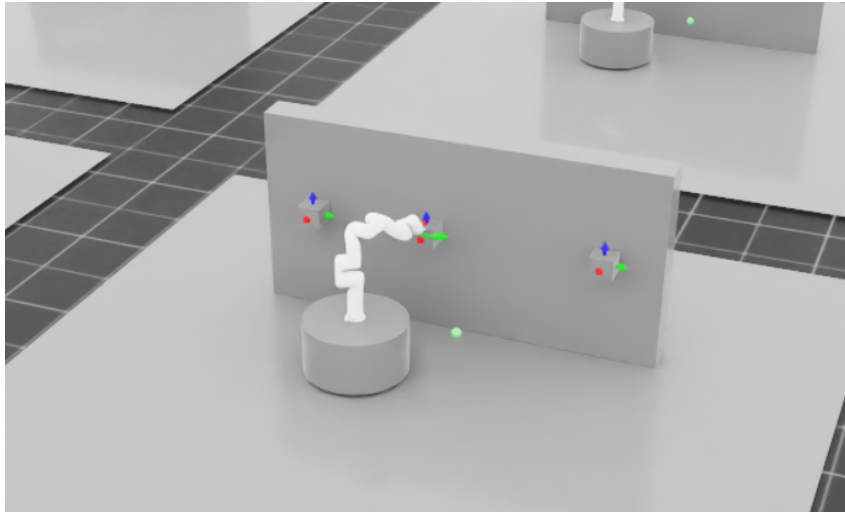
Finally, the increasing weight of the Base Distance to Goal reward, seen in Figure 6.21, tries to provide the final push for goal-reaching behavior. As training progresses, this reward becomes more influential, contributing to some successful completions as evidenced by rising values in the Robot Fixed Goal reward, seen in Figure 6.22. However, these instances where it reaches the goal are sparse and occur either accidentally too early, or too late in the training to represent a meaningful value for the policy to learn this behavior and do it consistently, as can be seen in the graph by sudden peak values with no continuation. The final position for a sample environment after the conclusion of the task at 7 seconds can be seen in Figure 6.23, which as stated by the previously mentioned Fixed Goal Reward is still a distance away from the goal and mostly in the same position as at the end of phase 3.



**Figure 6.21: Goal Affinity:** Reward for getting close to the final goal.



**Figure 6.22: Robot Fixed in Goal Reward:** Marks the completion of the task and the robot having reached the goal.



**Figure 6.23:** Sample final position of one of the environments in the simulation.

## 6.5 Training Metrics

This section presents and analyzes key training metrics observed during the learning process of the model. Since these metrics have not been introduced previously, a brief explanation of each is provided, followed by a detailed discussion of their behavior and shape throughout training.

**Learning Rate**, seen in Figure 6.24, determines how quickly the model updates its parameters in response to feedback. The learning drops sharply within the first 2,000 timesteps. This immediate decay reflects the need for early stability after rapid initial exploration. Following this, the learning rate exhibits a damped oscillation, likely a result of the learning rate scheduler adjusting to balance stability with continued policy improvement.



**Figure 6.24:** Learning Rate.

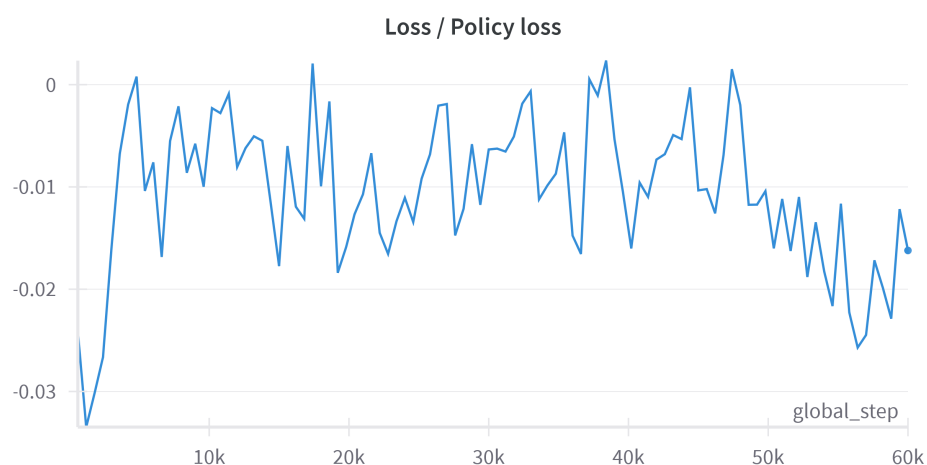


**Value Loss**, seen in Figure 6.25, measures how accurately the value function predicts future returns. Initially, the value loss drops close to zero, indicating that the model rapidly learns to predict rewards in the early, simpler stages of the environment. However, as the environment progresses and new phases are introduced, particularly those requiring more complex sequences of behavior, the loss begins to increase again. This upward trend continues until timestep 27,000 where it starts to better learn these more complex behaviors, making the value loss gradually decrease, returning to near-zero by the end of training.



**Figure 6.25:** Value loss.

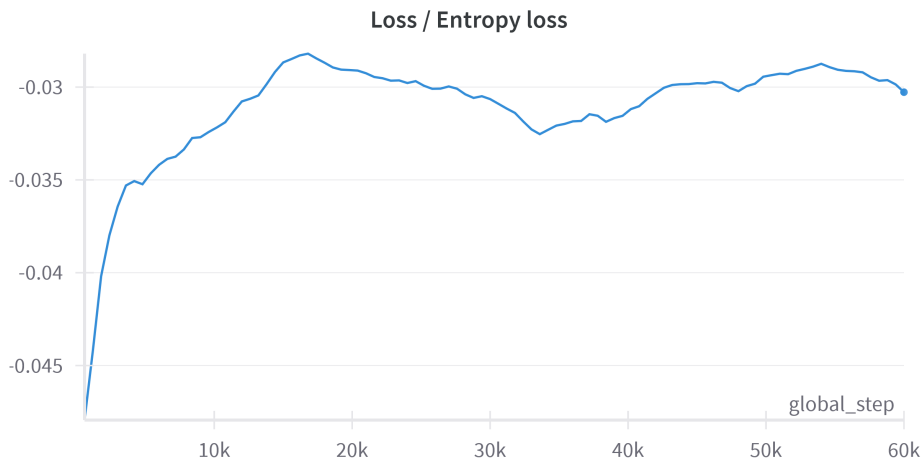
**Policy Loss**, seen in Figure 6.26, reflects the magnitude and direction of updates to the action policy. A sharp initial drop occurs as the model makes significant early improvements. Following this, the policy loss stabilizes and oscillates, representing ongoing smaller updates to the policy until it reaches the end of the training.



**Figure 6.26:** Policy loss.

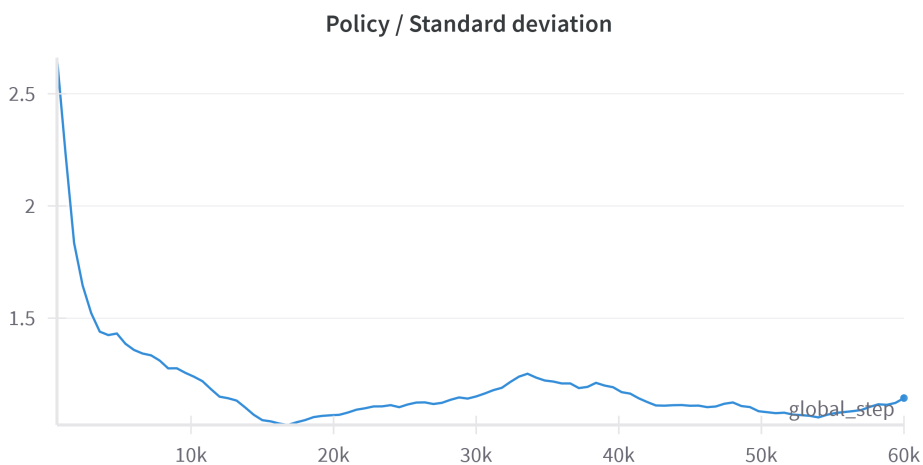
**Entropy Loss**, seen in Figure 6.27, is a proxy for how deterministic or exploratory the policy

is. The entropy loss increases early in the training, reflecting a reduction in exploration as the policy begins to converge toward more confident decisions. From there, entropy loss continues to rise slightly. The gradual increase suggests that while exploration is being reduced, the model retains some diversity in action selection to avoid premature convergence.



**Figure 6.27:** Entropy loss.

**Standard Deviation**, seen in Figure 6.28, represents the spread of the action distribution and serves as an indicator of the model's exploration behavior. It experiences a steep decline during the initial 15,000 timesteps, reflecting the model's growing confidence as it begins to learn effective actions. Following this sharp drop, the standard deviation continues to decrease more gradually. This trend, which inversely mirrors that of the entropy, highlights the model's progressive shift from broad exploration to more focused, confident decision-making.



**Figure 6.28:** Standard deviation.

**Total Reward Mean**, seen in Figure 6.29, represents the average of the sum of all individual rewards, each scaled by its respective weight. This metric is crucial for verifying that no single

reward dominates the learning process and that the overall reward structure remains balanced. Specially in curriculum learning related tasks, where reward weights change across different phases of the task, this visualization ensures that these transitions are smooth and constructive. As shown in the graph, the mean total reward steadily increases throughout training, indicating consistent learning progress and effective reward shaping over time.



**Figure 6.29:** Total Reward Mean.

## 6.6 Test conclusion

Each acceptance test was evaluated with a binary success criterion: The episode is considered passed if the required behaviour is completed within the 30s rollout horizon. Applying this rule, three of the four tests were successful as seen in Table 6.1.

Accept. Test No.	Test name	Passed
1	Control arm	✓
2	Grasp point	✓
3	Control base	✓
4	Grasp-move sequence	✗

**Table 6.1:** Summary of simulation test results

- **AT-1 – Point-to-Point motion (verifies requirements D 1.1, D 1.2).** All 528 evaluation roll-outs reached the designated pose within the time limit, confirmed by the sustained rise in Mount-Affinity reward and the reduction of joint-velocity penalties (Figure 6.3).
- **AT-2 – Fixed-point grasp (verifies requirements F 1.1).** Every trial achieved and maintained a grasp on the mounting point. The “Grasp detection 1” gets maximised (Figure 6.6), indicating consistent grasping at the first mounting point.

- **AT-3 – Base relocation (verifies requirements F 1.2, F 1.3).** In all evaluations, the agent detached its base, moved to the docking point, and re-docked the base while the arm remained latched, as reflected in the reward spike of Figure 6.10.
- **AT-4 – Two-step grasp–move sequence (verifies requirement F 1.4).** Although the agent consistently released Grasp 1 and secured Grasp 2, it did not reposition the base at the final goal with sufficient reliability; the Robot Fixed in Goal Reward (Figure 6.22) remains near zero. AT-4 is therefore classified as failed.

The results demonstrate that PPO can learn and combine the manipulation primitives required for single-arm free-floating “crawling”. Completion of phase 4 (AT-4) remains an open task and forms the principal limitation of the present proof-of-concept study.

## 7 Discussion

In this chapter, the results are discussed in terms of what contributed to the successes in earlier phases and what may have led to the shortcomings of the final phase.

As seen from the graphs in the Testing Chapter 6, the model learned effective behaviors for Phases 1 through 3, aligning with their intended objectives. In Phase 1, the robot quickly learned to reach the first mounting point, followed by successfully grasping it in Phase 2. This demonstrates that the reward structure for these initial phases is well-designed and that the robot was able to internalize the desired behaviors in a timely manner. Phase 3 further supports this, showing that one of the project’s core objectives, the ability to reposition the base on a frictionless floor, is indeed feasible using RL. Together, these results validate the early curriculum design and confirm that fundamental sub-skills required for the overall task can be learned effectively.

However, the agent was unable to complete Phase 4. This phase spans the longest training duration, from timestep 15000 to 60000, and while the robot was able to reach and successfully grasp the second mounting point, it consistently failed to relocate its base to the final goal. This means that requirement F 1.4, which specifies a 90% success rate for the full two-step traversal, was unmet.

Initially, this failure was hypothesized to stem from the model being exposed to too many simultaneous reward terms, making it unclear that progressing toward the final goal was the most rewarding behavior. To test this, a curriculum variation was introduced: from timestep 40000 onwards, the weights of all irrelevant reward terms were gradually faded to zero, leaving only those necessary for final goal progression. This unfortunately did not end with the robot reaching the goal, as it still stayed rooted to the docking point.

This suggests that the PPO policy, by that stage, may have converged to a suboptimal local minimum and was unable to adapt effectively to the final task. This may reflect a limitation of PPO when handling complex, multi-stage tasks with sparse terminal rewards and delayed dependencies. However, this hypothesis cannot be fully confirmed due to the limited number of successful Phase 4 trials.

The reason for the limited phase 4 trials stems from PPO being a stochastic algorithm; most runs did not reach Phase 4 within the available training steps and were terminated early if they failed to complete Phases 1 to 3 in a timely manner. Reaching the point where the robot reliably grasped the second mounting point typically occurred between 35000 and 45000 steps, and required approximately three hours of training per run. In total, only 8 of the 176 training runs reached a state where the robot had securely grasped the second mounting point and could attempt to move toward the goal.

In light of these results, it remains plausible that a PPO-based policy could learn the full task given more training time or better reward shaping. However, due to project time constraints, further experimentation was not feasible. It is recommended that future work explore alternative



## CHAPTER 7. DISCUSSION

---

RL algorithms that are more sample-efficient and robust to complex multi-phase tasks, such as SAC.

Although the robot never reached the final goal in a definitive and repeatable manner, this does not imply that the project as a whole was unsuccessful. The results from Phase 3 clearly demonstrate that the robot is capable of repositioning itself on a frictionless surface using RL. Additionally, Phase 4 confirms that the robot can grasp the second mounting point.

Taken together, these findings indicate that the individual components of the task, grasping, movement, and partial sequencing, are achievable with the current approach. Therefore, this project should be viewed as a successful proof-of-concept and a foundation for continued development. With additional training time, more refined and generalized reward shaping, or the use of more advanced RL algorithms, completing the full multi-step task remains a feasible and promising goal for future work.

## 8 Conclusion

This project investigated the use of Proximal Policy Optimization (PPO) reinforcement learning to enable autonomous control of the SKYWALKER system, a free-floating robot equipped with a manipulator, in a simulated microgravity environment. Using a combination of a curriculum-based reward strategy and a realistic simulation in Isaac Lab, the project aimed to assess whether a single-arm robot could navigate by docking and dragging itself between fixed mounting points in a frictionless environment.

### Final problem statement

*How can the PPO Reinforcement Learning algorithm be used to enable autonomous control and navigation for the SKYWALKER system in a simulated microgravity environment?*

This work demonstrated that PPO can indeed be used to control the SKYWALKER system to a meaningful degree. The trained agent successfully:

- Executed Point-to-Point arm movements,
- Grasped at fixed mounting points,
- Locked and unlocked its base appropriately,
- Repositioned itself using its arm while maintaining a secure grasp.

These behaviors emerged reliably through Phases 1 to 3 of the curriculum. However, the final task, requiring a full traversal between multiple mounting points, was not completed. While the agent occasionally initiated the correct behavior sequence, it never succeeded in executing the full grasp–release–regrasp cycle within the allowed limits. As such, the final phase remains an open challenge.

### Evaluation of Project Objectives

#### Scientific Objectives

- **SCI-001 (RL -Based Navigation):** *Achieved.* The agent was able to autonomously move its base across the floor using grasp-release strategies learned entirely through RL .
- **SCI-002 (RL Control):** *Achieved.* The PPO policy managed continuous control of the manipulator’s 7 joints and discrete gripper activation using hybridized continuous inputs.

- **SCI-003 (Microgravity Dynamics):** *Partially Achieved.* The project successfully simulated microgravity using frictionless floors. However, direct simulation of full 6D microgravity dynamics (e.g., full free-floating base in 3D) was beyond scope.
- **SCI-004 (Environment Adaptation):** *Pending.* Sim-to-real transfer was not executed within this project, but all systems were built with that future goal in mind, and the simulation was tailored to match SKYWALKER's real geometry and dynamics.

### Technical Objective

- **TEC-001 (SKYWALKER Platform Validation):** *Achieved.* The SKYWALKER system proved suitable for pre-testing RL policies before deploying to more complex platforms such as ESA's MANTIS.

## Key Contributions

- A realistic Isaac Lab simulation environment for the SKYWALKER robot.
- A custom curriculum structure enabling progressive learning of complex microgravity behavior.
- A validated PPO-based control policy that generalizes across multiple grasp-move tasks.

## 8.1 Future Work

This section outlines a progressive roadmap for advancing the project, moving from immediate improvements to more complex, long-term developments. Each step builds on the previous one by addressing specific limitations and pushing the system closer toward a fully autonomous and generalizable solution for the model implementation in the simulation environment and in the physical robot as well.

**Improving the Current Simulation:** The first step would involve refining the current simulation setup. Although the project has demonstrated that the task is solvable with RL, the implementation could benefit from more in-depth hyperparameter tuning as well as more generalized reward shaping to improve consistency and convergence towards reaching the final goal. Once a stable version is achieved, the next goal would be to introduce greater variability in the simulation, for example, by randomizing the positions of the grappling points and the spawn position of the robot arm and wall. To support this, the reward function could be redesigned to reflect a more generalized objective rather than being tightly coupled to the curriculum structure. By doing so, the agent would be encouraged to learn a higher-level strategy that can adapt across scenarios. Once generalization improves, training could focus on optimizing the speed and efficiency with which the task is completed.



**Initial Sim-to-Real Deployment:** In parallel with simulation improvements, a critical milestone would be to establish the first Sim-to-Real transfer. This would involve integrating ROS2 to enable communication between the simulation-trained policy and the physical robot. A hybrid control strategy is proposed: the RL policy would control movements between grapppling points, while traditional control would handle precise positioning and gripping. Observations would be collected through the robot's onboard camera and passed to the model. When a grappling action is triggered, control would temporarily switch to a traditional controller to adjust the gripper and execute the action. Once the gripper confirms closure, control would return to the RL policy. From the model's perspective, this would be perceived as a short delay in the action sequence. This setup would enable early testing on the SKYWALKER system at AAU while further refinements are done in the simulation environment for a final test at the ORL.

**Further Improvements in Sim-to-Real Implementation:** After validating the hybrid setup, the next phase would involve narrowing the simulation to reality gap. This can be achieved by replacing the default surface gripper with a custom-designed one imported as a USD asset. Doing so would allow the simulation to run headless and on a GPU, accelerating training and improving scalability. Moreover, with realistic gripping behavior in simulation, the RL policy could be fully trained without needing to switch to traditional control during deployment. Another important step would be adding a simulated camera and incorporating visual data into training. This could involve a first version with a vision-based ArUco markers recognition system to guide behavior, and slowly transitioning into training the policy directly on visual inputs as observations.

**Robustness and Generalization Testing:** Once a reliable training pipeline is established, more in-depth tests could be conducted to evaluate the model's robustness under different conditions. For example, changing the robot arm or altering floor friction would help determine how well the policy generalizes to unseen hardware and environments. These tests are key for real-world deployment, especially if the system is required to be adapted to other platforms or tasks.

**Multi-Arm and Complex Task Scaling:** A longer-term objective would be to scale the task to involve two manipulators operating simultaneously. In this setup, the arms would grapple onto mounting points located on separate layers, while the robot's base remains constrained to a frictionless plane, simulating a constant-height movement of the body of a robot similar to the one in the MIRROR Project. It would allow for testing coordination between multiple arms and represent a more realistic step toward the final mission. This final objective would be to apply the learned control strategy to a real system inspired by MIRROR, capable of assembling large space structures. While this concept holds great potential, there is still significant development needed before such an implementation can be achieved.

# A Appendix

## A.1 links

### **Link for Github:**

[https://github.com/Gsvend20/P10\\_MT\\_RL\\_SKYWALKER/tree/IsaacLab/scripts/SKYWALKER](https://github.com/Gsvend20/P10_MT_RL_SKYWALKER/tree/IsaacLab/scripts/SKYWALKER)

### **Link for Google Drive:**

[https://drive.google.com/drive/folders/1DE\\_JgdEmhXWLgGFHSP7r7FuvS3KErRcE?usp=sharing](https://drive.google.com/drive/folders/1DE_JgdEmhXWLgGFHSP7r7FuvS3KErRcE?usp=sharing)

### **Link for WANDB:**

[https://wandb.ai/bruno10mds-aalborg-universitet/IsaacLab-scripts\\_reinforcement\\_learning\\_skr?nw=nwuserbruno10mds](https://wandb.ai/bruno10mds-aalborg-universitet/IsaacLab-scripts_reinforcement_learning_skr?nw=nwuserbruno10mds)

### **Link for Youtube test videos:**

<https://www.youtube.com/playlist?list=PLq4R5gRcRPVgDwr7GdKrnrcwfPasr-Xkb>

## A.2 Full requirements

This section introduces the requirements necessary to develop a potential solution. These requirements follow ESA's specified format, categorizing them into functional, performance, interface, and operational requirements. Each requirement consists of a main (head) requirement, followed by a set of sub-requirements detailing how the main requirement is met. Each sub-requirement is assigned an ID, indicating the type and corresponding head requirement. Additionally, each sub-requirement includes a description, a justification, and a reference to the relevant section from which it is derived.

### A.2.1 Functional Requirements

The following functional requirements define specific behaviors and functions the system needs to meet its intended objectives.

## APPENDIX A. APPENDIX

ID	Description	Justification	Verification	Derived From
F.1	The system shall allow autonomous repositioning by pulling (dragging) itself	Enable autonomous repositioning.	Demonstration	N/A
F.1.1	The system shall provide a pulling force of at least TBD N to overcome base inertia.	Ensures sufficient force in low friction environments.	Test	F.1
F.1.2	The system shall lock the base when not actively pulling.	Prevents unwanted drift and ensures stability.	Inspection	F.1
F.2	The system shall be able to navigate between defined goal poses.	Establishes navigation capability.	Demonstration	N/A
F.3	The system shall be able to locate grappling points using visualization ----.	Essential for accurate manipulation tasks.	Demonstration	N/A
F.4	The system shall be able to reach grappling points.	Essential for positioning tasks.	Demonstration	N/A
F.5	The system shall be able to grasp grappling points.	Necessary for secure attachment.	Demonstration	N/A
F.5.1	The gripper shall apply a minimum grip force of TBD N.	Secure grasp for pulling.	Test	F.5
F.5.2	The system shall detect a successful grasp via force or torque feedback.	Ensures grasp stability.	Test	F.5
F.6	The system shall have the ability to stop its base to reach other grappling points.	Safety feature.	Demonstration	N/A
F.6.1	The system shall halt the base within TBD seconds of a stop command and then execute the stopping mechanism.	Ensures safety during operation.	Test	F.6
F.7	The base shall only be free-floating if the end-effector is grasping to a grappling points.	Continuous safe control.	Demonstration	N/A
F.7.1	The RL policy shall ensure the system is never left floating.	Essential for safe operation.	Demonstration	F.7
F.8	The RL model may work on multiple different robotic arms.	Ensures modular support.	Demonstration	N/A
F.9	An emergency stop signal shall override all ongoing motions to stop the system.	Provides immediate override.	Test	F.6

## A.2.2 Performance Requirements

The following requirements defines elements to maintain a high performance in the system. These requirements are based on testsable metrics and can be validated in later chapters to ensure high performance.

ID	Description	Justification	Verification	Derived From
P.1	Robot success rate at reaching goal positions shall be above TBD%.	Ensures reliability.	Test	F.2
P.2	The RL policy shall achieve at least TBD% success in reaching designated goals within the defined thresholds.	Establishes minimum reliability.	Test	F.2
P.3	Grasping positional accuracy shall be within euclidean distance of TBD mm.	Precise grasping.	Test	F.4
P.4	The system shall have a grappling point detection positional accuracy within $\pm TBD$ mm.	Alignment accuracy.	Test	F.3
P.5	The system shall be able to locate grappling points within TBD seconds.	Timely response for grasping.	Test	F.3

## A.2.3 Interface Requirements

ID	Description	Justification	Verification	Derived From
I.1	Sensor data, including arm joint states and camera images, shall be transmitted via the ROS2 protocol.	Ensures real-time monitoring and control.	Inspection	I.1
I.2	Emergency stop interface shall be accessible physically and remotely.	Safety-critical requirement.	Demonstration	N/A
I.3	The system shall broadcast state messages over the network.	Keeps operators informed.	Inspection	N/A

## A.2.4 Operational Requirements

ID	Description	Justification	Verification	Derived From
----	-------------	---------------	--------------	--------------

## APPENDIX A. APPENDIX

O.1	The system shall be fully autonomous during normal operation.	Reduces operator workload.	Demonstration	N/A
O.2	The system shall support offline reinforcement learning.	Enhances policy improvement.	Inspection	N/A
O.2.1	The system shall log sensor data and metrics during training sessions.	Facilitates analysis and policy refinement.	Inspection	O.2
O.3	The system shall autonomously monitor for anomalies in physical setup/internal systems.	Protects hardware integrity.	Demonstration	N/A

### A.2.5 Design Requirements

The design requirements is to ensure the developed system is within given thresholds and implemented in the earlier processes.

ID	Description	Justification	Verification	Derived From
D.1	The system weight shall not exceed TBD kg.	Ensures portability.	Inspection	N/A
D.2	The robotic arm shall have a minimum length of TBD meters.	Workspace and handling constraints.	Inspection	N/A
D.3	The robotic arm shall have sufficient degrees of freedom to reach all grappling points.	Ensures full workspace coverage.	Demonstration	F.4

# Bibliography

- [1] NASA. *International Space Station Overview*. Accessed: 2025-05-12. National Aeronautics and Space Administration. 2024. URL: <https://www.nasa.gov/reference/international-space-station/>.
- [2] NASA. *International Space Station Assembly Elements*. 2025. URL: <https://www.nasa.gov/international-space-station/international-space-station-assembly-elements/>. (accessed: 16.05.2025).
- [3] Mathieu Deremetz et al. "Preliminary Test Results of a Relocatable Robotic Demonstrator for In-space Telescope Servicing and Assembly". In: Oct. 2024. URL: [https://www.researchgate.net/publication/384733741\\_Preliminary\\_Test\\_Results\\_of\\_a\\_Relocatable\\_Robotic\\_Demonstrator\\_for\\_In-space\\_Telescope\\_Servicing\\_and\\_Assembly](https://www.researchgate.net/publication/384733741_Preliminary_Test_Results_of_a_Relocatable_Robotic_Demonstrator_for_In-space_Telescope_Servicing_and_Assembly). (accessed: 01.05.2025).
- [4] ESA - Academy Experiments programme. *ESA Academy Experiments programme*. URL: [https://www.esa.int/Education/ESA\\_Academy\\_Experiments\\_programme](https://www.esa.int/Education/ESA_Academy_Experiments_programme). (accessed: 01.06.2025).
- [5] National Air and Space Museum. *Arm, Canadarm Remote Manipulator System*. URL: [https://airandspace.si.edu/collection-objects/arm-canadarm-remote-manipulator-system/nasm\\_A20130168000](https://airandspace.si.edu/collection-objects/arm-canadarm-remote-manipulator-system/nasm_A20130168000). (accessed: 21.03.2025).
- [6] Canadian Space Agency. *About Canadarm2*. URL: <https://www.asc-csa.gc.ca/eng/iss/canadarm2/about.asp>. (accessed: 21.03.2025).
- [7] ESA. *European Robotic Arm*. URL: [https://www.esa.int/Science\\_Exploration/Human\\_and\\_Robotic\\_Exploration/International\\_Space\\_Station/European\\_Robotic\\_Arm](https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_Space_Station/European_Robotic_Arm). (accessed: 21.03.2025).
- [8] Canadian Space Agency. *About Dextre*. URL: <https://www.asc-csa.gc.ca/eng/iss/dextre/about.asp>. (accessed: 21.03.2025).
- [9] Peter Kazanzides. *Teleoperation and Visualization Interfaces for Remote Intervention in Space*. URL: <https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.747917/full>. (accessed: 01.03.2025).
- [10] NASA - Vanessa Lloyd. "Astronauts, Robots and the History of Fixing and Building Things in Space". In: (2020). URL: <https://www.nasa.gov/technology/astronauts-robots-and-the-history-of-fixing-and-building-things-in-space/>. (accessed: 01.03.2025).
- [11] Manu H. Nair et al. "Robotic technologies for in-orbit assembly of a large aperture space telescope: A review". In: *Advances in Space Research* 74.10 (2024), pp. 5118–5141. ISSN: 0273-1177. DOI: <https://doi.org/10.1016/j.asr.2024.08.055>. URL: <https://www.sciencedirect.com/science/article/pii/S0273117724008792>. (accessed: 20.03.2025).
- [12] R. Boumans. *The European Robotic Arm for the International Space Station*. URL: <https://www-sciencedirect-com.zorac.aub.aau.dk/science/article/pii/S0921889097000547>. (accessed: 21.03.2025).

## BIBLIOGRAPHY

---

- [13] ESA - Human Spaceflight. *Telerobotics: Ground-based Rover's Touch Shared with Astronaut in Space*. URL: <https://www.eoportal.org/other-space-activities/telerobotics-1#telerobotics-ground-based-rovers-touch-shared-with-astronaut-in-space>. (accessed: 21.03.2025).
- [14] Mathieu Deremetz et al. "Concept of Operations and Preliminary Flight Model Design of a Modular Multi-Arm Robot using Standard Interconnects for On-Orbit Large Assembly". In: Oct. 2021. URL: [https://www.researchgate.net/publication/354986227\\_Concept\\_of\\_Operations\\_and\\_Preliminary\\_Flight\\_Model\\_Design\\_of\\_a\\_Modular\\_Multi-Arm\\_Robot\\_using\\_Standard\\_Interconnects\\_for\\_On-Orbit\\_Large\\_Assembly](https://www.researchgate.net/publication/354986227_Concept_of_Operations_and_Preliminary_Flight_Model_Design_of_a_Modular_Multi-Arm_Robot_using_Standard_Interconnects_for_On-Orbit_Large_Assembly). (accessed: 01.04.2025).
- [15] Mathieu Deremetz et al. "MIRROR -A Modular and Relocatable Multi-arm Robot Demonstrator for On-orbit Large Telescope Assembly". In: 2023. URL: [https://www.researchgate.net/publication/374786749\\_MIRROR\\_-\\_A\\_Modular\\_and\\_Relocatable\\_Multi-arm\\_Robot\\_Demonstrator\\_for\\_On-orbit\\_Large\\_Telescope\\_Assembly](https://www.researchgate.net/publication/374786749_MIRROR_-_A_Modular_and_Relocatable_Multi-arm_Robot_Demonstrator_for_On-orbit_Large_Telescope_Assembly). (accessed: 01.04.2025).
- [16] ESA - Academy Experiments programme. *Orbital Robotics Lab*. URL: [https://www.esa.int/Education/ESA\\_Academy\\_Experiments\\_programme/Orbital\\_Robotics\\_Lab](https://www.esa.int/Education/ESA_Academy_Experiments_programme/Orbital_Robotics_Lab). (accessed: 21.03.2025).
- [17] Marti Vilella Ramisa. *ORBITAL ROBOTICS LAB USER MANUAL FOR THE ACADEMY EXPERIMENTS PROGRAMME*. URL: [https://esamultimedia.esa.int/docs/edu/Orbital\\_Robotics\\_Lab\\_User\\_Manual\\_for\\_the\\_Academy\\_Experiments\\_programme.pdf](https://esamultimedia.esa.int/docs/edu/Orbital_Robotics_Lab_User_Manual_for_the_Academy_Experiments_programme.pdf). (accessed: 21.03.2025).
- [18] European Space Agency. *Automation and Robotics Laboratories*. Accessed: 2025-05-19. 2024. URL: <https://technology.esa.int/lab/automation-and-robotics-laboratories>.
- [19] Erick Alvarado-Rodríguez et al. "REACSA: Actuated Floating Platform for Orbital Robotic Concept Testing and Control Software Development". In: *ResearchGate* (2023). Accessed: 2025-06-02. URL: <https://www.researchgate.net/publication/374631799>.
- [20] Yuki Mizuno. *A Taxonomy of RL Algorithms*. Medium post. <https://medium.com/@ym1942/a-taxonomy-of-rl-algorithms-341fd1b4c659>. 2023.
- [21] Jiayi Yuan et al. *A Taxonomy of Reinforcement Learning Algorithms*. <https://arxiv.org/abs/2411.18892>. 2024. arXiv: 2411.18892 [cs.LG].
- [22] Timothy P Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. (accessed: 01.04.2025).
- [23] Yuanyuan Liu et al. *Manipulator Control Method Based on Deep Reinforcement Learning*. 2020. DOI: 10.1109/CAC51589.2020.9327713. (accessed: 21.03.2025).
- [24] Mohammadamin Barekatain, Amin Noori, and Hamid Reza Karimi. *Continuous Control of a Robot Manipulator Using Deep Deterministic Policy Gradient*. DOI: 10.1109/ICEE55373.2022.9703155. (accessed: 21.03.2025).
- [25] Hui Liu, Wenhao Wu, and Zhibin Li. *A Manipulator Control Method Based on Deep Deterministic Policy Gradient with Parameter Noise*. URL: <https://www.researchgate.net/publication/356453295>. (accessed: 21.03.2025).
- [26] Mohammad Javad Shafiee et al. *Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review*. 2021. URL: <https://arxiv.org/pdf/2102.04148>. (accessed: 21.03.2025).

## BIBLIOGRAPHY

---

- [27] Rui Zhang et al. “Redundant space manipulator autonomous guidance for in-orbit servicing via deep reinforcement learning”. In: *Aerospace Science and Technology* 122 (2022), p. 107506. (accessed: 05.04.2025).
- [28] Xuesong Shi et al. “Reinforcement learning for robotic rock grasp learning in off-earth space environments”. In: *IEEE Transactions on Instrumentation and Measurement* (2021). (accessed: 05.04.2025).
- [29] Yanhui Yin et al. “Autonomous collision avoidance sample grasping method for extraterrestrial exploration”. In: *Acta Astronautica* (2022). (accessed: 05.04.2025).
- [30] Jun Zhang, Ming Wang, and Hongbo Wang. “A proximal policy optimization based deep reinforcement learning framework for tracking control of a flexible robotic manipulator”. In: *Mathematics* (2022). (accessed: 05.04.2025).
- [31] Jinwei Liu et al. “Manipulator trajectory optimization using reinforcement learning on a reduced-order dynamic model with deep neural network compensation”. In: *Machines* (2023). (accessed: 05.04.2025).
- [32] Weiyu Shang et al. “IPPO: Obstacle avoidance for robotic manipulators in joint space via improved proximal policy optimization”. In: *arXiv preprint arXiv:2210.00803* (2022). (accessed: 05.04.2025).
- [33] Jianan Tang et al. “Research on Manipulator Control Based on Improved Proximal Policy Optimization Algorithm”. In: (2023). (accessed: 05.04.2025).
- [34] Bo Xia et al. “Trajectory Planning for Teleoperated Space Manipulators Using Deep Reinforcement Learning”. In: *arXiv preprint arXiv:2408.05460* (2024). (accessed: 05.04.2025).
- [35] Y. Wang, L. Zhang, and H. Liu. “Space Manipulator Collision Avoidance Using a Deep Reinforcement Learning Control”. In: (2023). DOI: 10.3390/aerospace10090778. (accessed: 05.04.2025).
- [36] Xinghong Kuang et al. “Robotic Manipulator in Dynamic Environment with SAC Combining Attention Mechanism and LSTM”. In: (2024). DOI: 10.3390/electronics13101969. (accessed: 05.04.2025).
- [37] Yujie Liu, Wei Zhang, and Ming Chen. “Trajectory Tracking Control for Robotic Manipulator Based on Soft Actor–Critic and Generative Adversarial Imitation Learning”. In: *Biomimetics* (2024). DOI: 10.3390/biomimetics9120779. (accessed: 05.04.2025).
- [38] J. Kim, S. Lee, and H. Park. “Deep Reinforcement Learning-Based Path Planning for Multi-Arm Manipulators with Periodically Moving Obstacles”. In: (2021). DOI: 10.3390/app11062587. (accessed: 05.04.2025).
- [39] Bahador Beigomi and Zheng H. Zhu. “Towards Real-World Efficiency: Domain Randomization in Reinforcement Learning for Pre-Capture of Free-Floating Moving Targets by Autonomous Robots”. In: (2024). (accessed: 05.04.2025).
- [40] Scott Fujimoto, David Meger, and Doina Precup. “Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics”. In: (2020). URL: <https://arxiv.org/abs/2005.04269>.





## BIBLIOGRAPHY

---

- [41] Jiawei Chen et al. “High Path Converging Reward Space: Imitating Human Behavior in Robot Arm Control”. In: (). URL: [https://kdbk-aub.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi\\_ieee\\_primary\\_10003926&context=PC&vid=45KBDK\\_AUB:AUB&lang=da&search\\_scope=MyInst\\_and\\_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,TQC%20manipulators&offset=0](https://kdbk-aub.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi_ieee_primary_10003926&context=PC&vid=45KBDK_AUB:AUB&lang=da&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,TQC%20manipulators&offset=0). (accessed: 10.04.2025).
- [42] Yongqiang Wu et al. “Constrained Object Placement Using Reinforcement Learning”. In: (2024). URL: <https://arxiv.org/html/2404.10632v1>. (accessed: 10.04.2025).
- [43] X. Zhou et al. “Entropy-Guided Distributional Reinforcement Learning with Controlling Uncertainty in Robotic Tasks”. In: (2024). URL: <https://www.mdpi.com/2076-3417/15/5/2773>. (accessed: 10.04.2025).
- [44] Nikolay Koptev et al. “Guiding real-world reinforcement learning for in-contact manipulation tasks with Shared Control Templates”. In: (2024). URL: <https://link.springer.com/article/10.1007/>. (accessed: 10.04.2025).
- [45] Weiyu Yuan et al. “Deep Reinforcement Learning for Robotic Grasping from Octrees”. In: (2023). URL: [https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK\\_AUB/n4l1aj/alma9921563622805762](https://kdbk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/n4l1aj/alma9921563622805762). (accessed: 10.04.2025).
- [46] Universal Robots. *UR3e*. URL: <https://www.universal-robots.com/products/ur3e/>. (accessed: 01.05.2025).
- [47] UFACTORY. *UFACTORY X-ARM 7*. URL: <https://www.ufactory.us/product/ufactory-xarm-7>. (accessed: 01.05.2025).
- [48] xArm-Developer. *xarm\_ros*. URL: [https://github.com/xArm-Developer/xarm\\_ros](https://github.com/xArm-Developer/xarm_ros). (accessed: 01.05.2025).
- [49] NVIDIA *Isaac Sim*. URL: <https://developer.nvidia.com/isaac/sim>. (accessed: 21.03.2025).
- [50] *Welcome to Isaac Lab!* URL: <https://isaac-sim.github.io/IsaacLab/main/index.html>. (accessed: 21.03.2025).
- [51] Antonio Serrano-Muñoz et al. “skrl: Modular and Flexible Library for Reinforcement Learning”. In: *Journal of Machine Learning Research* 24.254 (2023), pp. 1–9. URL: <http://jmlr.org/papers/v24/23-0112.html>.
- [52] Michael Kudlaty. *Solving Gymnasium’s Car Racing with Reinforcement Learning*. Accessed: June 2, 2025. Sept. 2024. URL: <https://www.findingtheta.com/blog/solving-gymnasiums-car-racing-with-reinforcement-learning>.
- [53] John Schulman et al. *Proximal Policy Optimization Algorithms*. URL: <https://arxiv.org/pdf/1707.06347>. (accessed: 01.06.2025).