POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Development of an Autonomous Mobile Manipulation Robot for Industrial and Agricultural Environments

Tesi di Laurea Magistrale in
Computer Science Engineering

Author: **Simone Giampà**

Student ID: 980857
Advisor: Prof. Matteo Matteucci
Co-advisor: Dr. Gianluca Bardaro
Academic Year: 2023-24

"Any sufficiently advanced technology is indistinguishable from magic."

– Arthur C. Clarke

Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

Keywords: here, the keywords, of your thesis

Abstract in lingua italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

Parole chiave: qui, vanno, le parole chiave, della tesi

Contents

Abstract **iii**

Abstract in lingua italiana **v**

Contents **vii**

Introduction **1**

1 State of the Art and Literature Review **5**
1.1 Robotic Manipulator Control Approaches 5
1.1.1 Mobile Manipulation Tasks 7
1.2 Traditional Control Approaches 8
1.3 Deep Reinforcement Learning: a Data-Driven Approach 11
1.3.1 Challenges in Data-Driven approaches 14
1.4 Whole-body mobile manipulator control 16
1.4.1 MPC+IK for articulated object manipulation 16
1.4.2 Deep Reinforcement Learning for high DoF control 18
1.4.3 Mobile manipulation with Imitation Learning 24
1.4.4 Comparison of Model-Based and Data-Driven approaches 28
1.5 Addressing the Simulation-to-Reality Gap 29
1.6 Object Detection and Grasping 30
1.6.1 Grasping Soft Objects 32

2 Robotic Platform for Mobile Manipulation **35**
2.1 Mobile Robot Platform 35
2.2 Robotic Arm Manipulator 37
2.3 Sensors and Perception 40
2.3.1 3D LiDAR 40
2.3.2 RGB-D Stereo Camera Sensor 40

2.3.3 Intel Realsense Calibration . . . 41
2.4 Soft Gripper Actuator . . . 43
2.5 3D Printed Mounts Design . . . 45
2.5.1 MountV1 . . . 47
2.5.2 MountV2 . . . 48
2.5.3 GPS Antenna Support . . . 51
2.5.4 3D printer maintenance and configuration . . . 52
2.6 Batteries and Power Management . . . 53
2.7 Mobile Manipulation Setup . . . 54

3 Software Architecture and Simulation Environments 59
3.1 ROS2 Control Interface for Igus Rebel Arm . . . 59
3.1.1 Joint Trajectory Controller . . . 62
3.2 MoveIt2 and RViz2 Simulation Environment . . . 63
3.3 Robotic Arm Visual Servoing . . . 67
3.4 Collision Avoidance with Octomap . . . 68
3.5 Soft Gripper Pneumatic Pump Actuation . . . 70
3.6 Ignition Gazebo Simulation Environment . . . 71
3.7 Autonomous Navigation with NAV2 . . . 72
3.7.1 Nav2 Parameters Tuning . . . 73
3.7.2 Optimal DDS configuration . . . 78
3.8 Parking Algorithm for Mobile Robot . . . 81
3.9 ArUco Marker Detection and Pose Estimation . . . 86
3.9.1 Multi-ArUco Plane Estimation Algorithm . . . 86
3.10 Object Detection with YOLOv8 . . . 91
3.11 Pose Estimation with Object Detection and Depth Perception . . . 97
3.11.1 Algorithm for Object's Center Estimation . . . 97
3.11.2 Algorithm for Grasp Pose Estimation . . . 99
3.12 ROS2 Actions Client-Server Architecture for High-Level Tasks . . . 101

4 Experimental Setups and Demonstrations 105
4.1 ArUco Follower Demo . . . 106
4.2 Button Presser Demo . . . 107
4.2.1 Buttons Setup Box and End Effector Setups . . . 108
4.2.2 End Effector Positioning and Linear Trajectories . . . 109
4.2.3 Mobile Button Presser Demo . . . 111
4.2.4 Mobile Button Presser Demo Implementation and Architecture . . . 113
4.3 Object Picking Demo . . . 116

4.3.1 Plants, Colored Balls, and Apples Setup 117
4.3.2 DemoV1 with Manual User Input 120
4.3.3 DemoV2 with Object Detection Neural Network 122
4.3.4 Mobile Fruit Picking Demos 123
4.3.5 Mobile Fruit Picking Demo Implementation and Architecture 126

5 Conclusions and Future Work **133**

Bibliography **135**

List of Figures **139**

List of Tables **145**

List of Algorithms **147**

Acknowledgements **149**

Introduction

Robots were originally designed to perform repetitive tasks and/or dangerous tasks for humans in extreme environments. With continuous developments in mechanics, sensing technology, intelligent control, and other modern technologies, robots have improved autonomy capabilities and are more dexterous.

Articulated robots, also called robotic arms or manipulator arms, are among the most common robots used today. In some contexts, a robotic arm may also refer to a part of a more complex robot. A robotic arm can be described as a chain of links that are moved by joints that are actuated by motors.

The majority of robotics applications focus either on navigation aspects of mobile platforms (e.g. industrial transportation systems, guide robots), or the manipulation of goods with robotic arms (e.g., bin-picking applications). Nonetheless, few applications consider mobile manipulation combining both robotic tasks. There is a lack of real applications of mobile manipulation systems due to the complexity and uncertainty introduced by combining both manipulation and navigation.

Traditionally, such mobile manipulation operations have been solved using analytical planning and control methods. These methods require explicit programming of the skills which can be very costly and error-prone, particularly in problems where decision-making is complex and the environment dynamic and partially known. The performance of these models depends on how well the real world fits the assumptions made by the model. Well-known planning and control methods have been widely used for mobile manipulation behaviors, for example using the Nav2 navigation framework, SLAM algorithms for localization and navigation, and MoveIt for arm and object manipulation.

The main challenge in mobile manipulation is to combine the navigation and manipulation tasks in a single system. The navigation task requires the robot to move from one place to another, while the manipulation task requires the robot to interact with objects in the environment. The robot must be able to plan and execute both tasks in a coordinated manner. This requires the robot to have a good understanding of the environment, including the location of objects and obstacles, and the ability to plan and execute complex

manipulation tasks.

This Thesis project aims to develop a mobile manipulation system that performs manipulation tasks in a dynamic environment. The system is based on two robots: a mobile robot and a robotic arm. The mobile robot is equipped with a LiDAR sensor for navigation and the robotic arm is equipped with a camera for object detection and perception. The system can perform mobile manipulation tasks in both agricultural and industrial environments. The entire project revolves around the development of software components that will enable the robots to perform high-level tasks autonomously, without human intervention, and minimal human supervision. The focus of the project is the complete autonomy of the system, including navigation, object detection, manipulation, and task planning.

The project is based on already available robotic platforms and hardware, and the focus will be on the development of software components. This project includes the development of algorithms, software packages, and libraries that will be used to control the mobile manipulation system and perform high-level tasks.

The final objective of the project is the development and realization of two demonstrations that will showcase the capabilities of the mobile manipulation system. The first demonstration is a system that can interact with a control panel in industrial environments, specifically pressing buttons on a box knowing only the relative positioning of the buttons and the ArUco markers on the box. The second demonstration is a system that can interact with a plant in an agricultural environment, specifically picking up fruits from an artificial tree.

This Thesis is structured as follows:

- Chapter 1: **State of the Art and Literature Review**. This chapter provides an overview of the state-of-the-art and a review of the relevant literature on mobile manipulation.
- Chapter 2: **Robotic Platform for Mobile Manipulation**. This chapter describes the robotic platform used in the project, including all the hardware components and sensors.
- Chapter 3: **Software Architecture and Simulation environments**. This chapter describes the software architecture of the system, all the algorithms, software libraries, and the simulation environments used for testing and development of the system.
- Chapter 4: **Experimental Setup and Demonstrations**. This chapter describes

the experimental setups used to test the system and demonstrate the capabilities of the entire system.

- Chapter 5: **Results and Future Work**. This chapter presents the results of the experiments and discusses the limitations of the system and future work.

1 | State of the Art and Literature Review

This chapter will present the state-of-the-art and literature review of the topics related to this project. The topics are: Robotic Manipulator Control, Deep Reinforcement Learning in Robotic Manipulation and Mobile Manipulation, Autonomous navigation, Object Detection and Grasping.

Particular focus is on the part regarding Mobile Manipulation since it is the main topic of this thesis project. In particular, the potential challenges as well as possible benefits and disadvantages of using each method will be discussed.

1.1. Robotic Manipulator Control Approaches

Currently, the control sequence of a robotic manipulator is mainly achieved by solving inverse kinematic equations to position the end effector with respect to the fixed frame of reference. Robots can be controlled in an open loop or with exteroceptive feedback. The **open-loop control** does not have external sensors or environment sensing capability but heavily relies on highly structured environments that are very sensitively calibrated. Under this strategy, the robot arm follows a series of positions stored in memory and goes through them at various times in their programming sequence. In more advanced robotic systems, **exteroceptive feedback control** (closed loop system) is employed, through the use of monitoring sensors, force sensors, even vision or depth sensors, that continually monitor the robot's axes or end-effector, and associated components for position and velocity. The feedback is then compared to information stored to update the actuator command to achieve the desired robot behavior. Either auxiliary computers or embedded microprocessors are needed to interface with these additional sensors and to perform the required computations. These two traditional control scenarios are both heavily dependent on hardware-based solutions [12].

Other control strategies may include **robotic embodiement** for **imitation learning**.

Robotic embodiment, in the context of imitation learning, is a control strategy that is based on the idea that the robotic system emulates the human body movement, to learn a task quickly, instead of relying on specific ad-hoc training and programming, as suggested in [10]. This article presents an approach to the autoprogramming of robotic assembly tasks with minimal human assistance. The approach integrates "robotic learning of assembly tasks from observation" and "robotic embodiment of learned assembly tasks in the form of skills". The aim of these skills is to let robots execute difficult tasks that involve inherent uncertainties and variations and are most useful in smart manufacturing in industrial scenarios. The robotic embodiment is associated with a dramatic reduction in the human effort required for automating robotic assembly, as well as task training.

With the advancements in modern technologies in artificial intelligence, such as deep learning, and recent developments in robotics and mechanics, both the research and industrial communities have been seeking more software-based control solutions using low-cost sensors, which have fewer requirements for the operating environment and calibration. The key is to make minimal but effective hardware choices and focus on robust algorithms and software. Instead of hard-coding directions to coordinate all the joints, the control policy could be obtained by learning and then be updated accordingly. **Deep Reinforcement Learning (DRL)** is among the most promising algorithms for this purpose because it ideally suits complex robotic manipulation and control tasks in dynamic and unstructured environments, or when the task is too complex to be explicitly programmed. A reinforcement learning approach might be trained on a dataset of experiential data, such as input data from a robotic arm experiment, with different sequences of movements, or input data from simulation models. Either type of dynamically generated experiential data can be collected and used to train a Deep Neural Network (DNN) by iteratively updating specific policy parameters of a control policy network [12].

Robotic control approaches can be broadly categorized into **model-based approaches**, such as the ones using a Model Predictive Controller (MPC) and Inverse Kinematics (IK) computation, and **model-agnostic approaches**, often characterized as **data-driven methods**, including Deep Reinforcement Learning (DRL) and other machine learning techniques.

- **Model-based approaches** rely on explicit models of the robot's dynamics or kinematics to formulate control strategies. MPC optimizes control inputs over a prediction horizon based on the system's dynamics and constraints, while IK determines joint configurations to achieve desired end-effector poses.
- **Model-agnostic approaches** learn control policies directly from data through

interactions with the environment. These data-driven methods leverage neural networks to map observations to actions, allowing robots to adapt to complex and dynamic scenarios without requiring an explicit model.

The main differences lie in the reliance on explicit models in model-based methods, providing **transparency and interpretability**, **versus** the model-free nature of data-driven methods, offering **flexibility and adaptability** to diverse and evolving environments. Integrating these approaches can harness the strengths of both paradigms, combining the precision of model-based control with the adaptability of data-driven learning for enhanced robotic control capabilities in multiple scenarios and tasks.

An issue raised by the real-world application is the safety of the system while sharing the workspace with human workers. Identifying and, more importantly, certifying methods to collaborate with humans in the workspace in a safe way are key points for bringing autonomous mobile robots to real industrial applications.

The following paragraphs will describe the available methods used for robotic manipulator control.

1.1.1. Mobile Manipulation Tasks

Mobile manipulators that combine base mobility with the dexterity of an articulated manipulator have gained popularity in numerous applications ranging from manufacturing and infrastructure inspection to domestic service. Deployments span a range of interaction tasks with the operational environment from minimal interaction tasks, such as inspection, to complex interaction tasks such as logistics resupply and assembly. This flexibility, offered by the redundancy, needs to be carefully orchestrated to realize enhanced performance. Thus, advanced decision-support methodologies and frameworks are crucial for successful mobile manipulation in (semi-) autonomous and teleoperation contexts [29]. Given the enormous scope of the literature, I restrict my attention to decision-support frameworks specifically in the context of wheeled mobile manipulation.

As a quick aside, a disambiguation is necessary between the often interchangeably used **"motion planning"** and **"path planning"**. Although path planning only generates a path within the configuration space, motion planning generates time-indexed motion trajectories. Instead path-following only requires spatial feasibility (e.g., obstacle avoidance), while motion planning requires compatibility with spatiotemporal constraints (engendered in the dynamics of both robot and environment). It is also noteworthy that ultimately any path planning effort requires a final time parameterization into a motion planning exercise before deployment [29].

The combined controllable degrees of freedom within the kinematic chain (from both mobile base and the articulated manipulator) presents the mobile manipulator design architecture the opportunity to address very complex tasks. However, resolving the redundancy (internal/external) is crucial to realizing this potential. As the complexity of the overall mobile manipulation process increases, a **two-stage hierarchical approach** is often pursued:

1. task planning/breakdown into a series of tractable motion planning subtasks and their sequencing
2. motion planning of the high degree-of-freedom mobile manipulator within each sequenced task

It is noteworthy that the two steps (task planning and motion planning) are closely coupled and should be solved concurrently but are addressed separately from a computational tractability perspective [29].

1.2. Traditional Control Approaches

However, a breakdown along the lines of mobile manipulator subsystems (mobile base versus manipulator versus gripper or combinations) or along the nature of the manipulation task (transportation versus grasping) feature prominently in the literature. The task-level and motion-level planning frameworks can be viewed as a form of "artificially constrained" motion planning within a higher dimensional space.

The first applications for mobile manipulators were in the field of logistics, where the mobile base was used for transportation and the manipulator for grasping and placing objects. Traditional control approaches rely on a series of heuristics to solve the problem of navigation, grasping, and placing objects. The mobile base is controlled by a multitude of algorithms, such as SLAM, AMCL and DWA for navigation, while the manipulator is controlled by lower-level motor controllers or trajectory planners. The integration between the mobile base and the manipulator is often done by a **switching layer** that determines the currently pertinent control objectives. Traditional methods often do not handle multiple and different control objectives at the same time, so the robot divides one high-level task into multiple sub-tasks executed in a sequence. This approach requires a lot of engineering effort to coordinate the arm and the base movements, and often fails in complex tasks where the decision-making process is hard.

Mobile Manipulator for Autonomous Localization, Grasping and Precise Placement of Construction Materials in a Semi-Structured Environment A starting

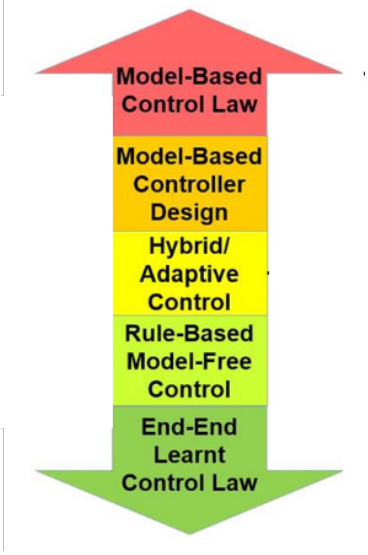


Figure 1.1: The continuum in the literature in regards to control methodology ranging from model-based to end-end data-driven control [29]

point for mobile manipulation is the work [27], presenting the winning mobile manipulation system for the *Mohamed Bin Zayed International Robotics Challenge (MBZIRC)* held in 2020. The proposed system is comprised of a mobile wheeled base performing localization and navigation in a semi-structured environment, and a 5-DoF manipulator for grasping and precise placement of bricks in a carrier. This work was among the first to demonstrate the potential of mobile manipulation in a practical scenario.

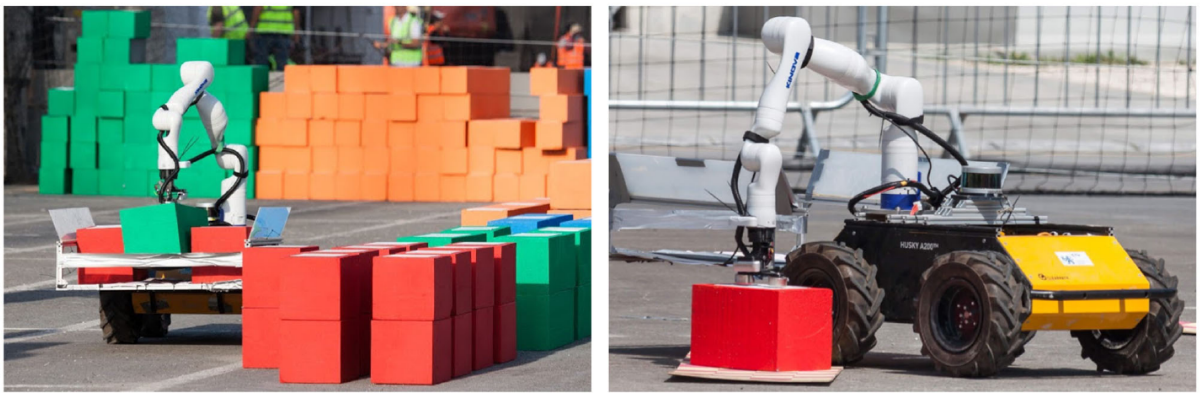


Figure 1.2: The described system loading and placing building material during the MBZIRC 2020 contest. [27]

However, their approach is based on many simplifying assumptions which may be suitable as a first one-of-a-kind robotic pick-and-place system in a real-world scenario and

application but is not robust enough for more complex tasks. For example, the grasping pipeline is trained to handle only bricks, i.e. solid parallelepiped objects, for which the grasping pose is straightforward to compute. Furthermore, the robot is not able to autonomously decide where to place the brick, but it is only able to place it in a predefined position. Also, the arm controller is quite primitive, since it doesn't handle collisions with the mobile base appropriately, and the arm is not able to avoid obstacles in its workspace.

Go Fetch: Mobile Manipulation in Unstructured Environments This work [3] presents a mobile manipulation system that combines perception, localization, navigation, motion planning and grasping skills into one common workflow for "fetch-and-carry" applications in unstructured indoor environments. The integration across the various modules is experimentally demonstrated in the video [2], showing the task of finding a commonly available object in an office environment, grasping it, and delivering it to a desired drop-off location.

Figure 1.3: A picture of RoyalYumi in action. It features a two-arm ABB Yumi, a Clearpath Ridgeback mobile base, two Hokuyo 2D LiDARs, an Intel RealSense D435 and a Skybotix VI-Sensor. [3]

The research [3] is an example of a mobile manipulation system for pick-and-place tasks in an indoor environment that adopted many simplifications in order to achieve the desired results. The system uses a series of heuristics to approach the object and the grasping phase, as can be seen from the demonstration video [2]. The navigation uses a feature-based map and localization modules, while the arm controller is handled by MoveIt! [22]

solver coupled with the ROS framework [25]. This system is not very well integrated as each phase of the task is carried out by a different module, and shows how slow and inefficient the robot is in picking the banana. The grasping phase uses multiple views of the object in order to compute a better grasp pose, which works well but seems to be overly complex given the predefined task. Overall, the system can be regarded as a starting point for mobile manipulation and one of the first works in dual-arm manipulability.

Although traditional methods have led to promising mobile manipulation skills in some specific tasks, mobile manipulation tasks require the explicit programming of **hard-to-engineer behaviors** and often fail in more complex tasks where the decision-making process is hard. In addition, such solutions are generally very inflexible and error-prone due to the impossibility of modeling all the uncertainty of dynamic industrial environments when those are programmed.

1.3. Deep Reinforcement Learning: a Data-Driven Approach

Explicit programming is often needed in practice to account for uncertainties in the environment and sensors used, as well as to solve highly variable problems efficiently. Explicit behavior programming is therefore often tedious and impractical, and more flexible solutions are needed in environments where the robot must be adaptable. Alternatively, data-driven approaches address the main limitations of traditional methods and propose to learn robotic behaviors from real experience, thus alleviating the cost of modeling complex behaviors. This approach allows them to use deep neural networks to model the uncertainties of the environment, which leads to a more robust controller compared to traditional ones. Unlike deep learning (DL), the reinforcement learning (RL) paradigm allows to automatically obtain the experience needed to learn robotic skills through trial-and-error and allows to **learn complex decision-making policies**.

With RL, the explicit modeling of the problem is no longer required since the learned models are grounded in real experience. Recently, the combination of DL and RL, also known as Deep Reinforcement Learning (DRL), has made it possible to tackle complex decision-making problems that were previously unfeasible. It combines the ability of DL to model very high dimensional data with the ability of RL to model decision-making agents through trial and error. DRL has proven to be the state-of-the-art technology for learning complex robotic behaviors through the interaction with the environment and the training solely guided by a reward signal [12].

While ML-based methods are generally used for offline forecasting, DRL is generally used online in sequential decision-making problems. In fact, DRL allows one to autonomously learn complex control policies through trial and error and only guided by a reward signal. In the case of robotics, the most common use case is to use such algorithms to model agents capable of performing continuous control of robots.

DRL has been successfully applied in a wide variety of areas such as **robotics, computer vision and video games**. Taking into account the difficulty of modeling complex decision-making robotic skills, DRL offers a promising way to take advantage of the experience gathered by interacting with the environment to autonomously learn complex robotic behaviors. In particular, the field of DRL applied to robotics has recently gained popularity due to the remarkable performance obtained in applications with high decision-making and control complexity. Applications range from manipulation to autonomous navigation and locomotion. [9]

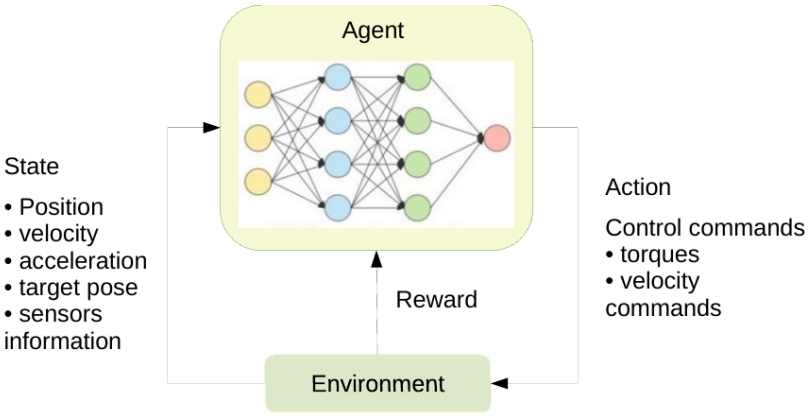


Figure 1.4: A schematic diagram example for robotic manipulation control using a data-driven approach such as DRL [12]

Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled by Deep Reinforcement Learning The work [8] presents one of the first and pioneering approaches to DRL-based methods for pick and place tasks. Their work focused on the positioning problem, consisting of a local navigation problem where the robot must move to a desired position moving by small distances in a confined environment to reach the target object. They relied on DRL for controlling the mobile wheeled base robot, while the arm controller was handled by MoveIt! framework [22]. This can be regarded as a first step towards more complex tasks, as it shows the foundations of DRL-based methods for mobile manipulation. The method had some flaws, like the imprecise navigation due to errors in localization and odometry, which the network was not able to overcome since it was not fed video data stream. However, it paved the way for a lot of other works,

many of which are mentioned in this chapter.

Fully Autonomous Real-World Reinforcement Learning with Applications to Mobile Manipulation A work from Berkeley AI research [28] show *ReLMM*, a model that can learn continuously on a real-world platform without any environmental instrumentation, without human intervention, and access to privileged information, such as maps, objects positions, or a global view of the environment. Their method employs a modularized policy with components for manipulation and navigation, where manipulation policy uncertainty drives exploration for the navigation controller and the manipulation the module provides rewards for navigation. They trained the policy on a room cleanup task, where the robot must navigate to and pick up items scattered on the floor. The robot **learns entirely from its sensors** in a real-world environment, without any simulation and minimal human intervention. Furthermore, the entire learning process is efficient enough for real-world training. On top of this, the robot can continually gather data at scale and improve its performance over time, with the auto-reset functionality.

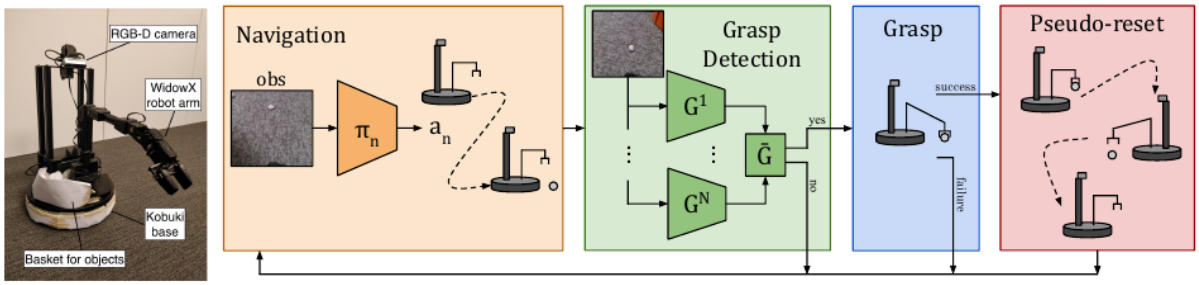


Figure 1.5: ReLMM partitions the mobile manipulator into a navigation policy and grasping policy. Both policies are rewarded when an object is grasped [28]

Although the results shown in [28] demonstrate efficient performance in the pick and place task, there are many issues circumvented by simplifying the problem. For example, the robot is very small, a modified version of Turtlebot, running in a small and contained environment. They didn't address any safety and collision issues since the platform mounted bumping sensors, and any collision would not harm the robot at all. Doing so enabled them to train the policy online without any prior simulation. Furthermore, the kinematics of the mobile base and the robotic arm are very simple, and having the stereo camera mounted on top of the robotic platform allowed them to easily detect the objects on the floor. This work paves the way for more complex systems and tasks, but it is still far from being a general solution for mobile manipulation.

1.3.1. Challenges in Data-Driven approaches

Two of the most important challenges here concern **sample efficiency and generalization**. The goal of DRL in the context of robotic manipulation control is to train a deep policy neural network, to detect the optimal sequence of commands for accomplishing the task. The current state of the algorithm can include the angles of joints of the manipulator, the position of the end effector, and their derivative information, like velocity and acceleration. The output of this policy network is an action indicating control commands to be implemented to each actuator, such as torques or velocity commands. When the robotic manipulator accomplishes a task, a positive reward will be generated. With these delayed and weak signals, the algorithm is expected to find out the most successful control strategy for the robotic manipulation [12].

The challenges of learning robust and versatile manipulation skills for robots with DRL are still far from being resolved in real-world applications. Currently, robotic manipulation control with DRL may be suited to fault-tolerant tasks, like picking up and placing objects, where failure will not cause significant damage. It is quite attractive in situations, where there are too many variables that make explicit modeling algorithms not work effectively [12].

However, even in this kind of application, DRL-based methods are not widely used in real-world robotic manipulation. The reasons are multiple, including sample efficiency and generation, where more progress is still required, as both the gathering experiences by interacting with the environment and the **collection of expert demonstrations** for imitation learning are expensive procedures, especially in situations where robots are heavy, rigid and brittle, causing high costs in case of failures.

Another very important issue is the **safety guarantee**. Unlike in simulation tasks, we need to be very careful that learning algorithms are safe, reliable and predictable in real scenarios. This is especially true when moving to applications that require safe and correct behaviors with high confidence, such as surgery or household robots taking care of the elderly or the disabled. There are also other challenges including but not limited to the algorithm explainability, the learning speed, and high-performance computational equipment requirements. [12]

Learning positioning policies for mobile manipulation operations with deep reinforcement learning The work proposed in [9] is a practical example of a DRL-based approach facing these challenges and the limitations to overcome (as well as the potentialities). The mobile platform in figure 1.6 is used in an industrial environment for an approaching task. The robot learns to navigate to the desk where the target object

is located, and then uses the MoveIt! planner to check whether the trajectory to pick the object is feasible. The robot's localization is based on AMCL and the learned policy serves as a controller for local navigation tasks. Their work shows many shortcomings of this approach, such as the integration between the DRL policy and localization package, with noise in the real environment affecting negatively the performance of the robot. As a result, the video presented shows an inefficient and jiggly movement because of the non-smooth control policy. They mention the necessity of mounting a stereo camera for navigation, to reduce the errors in the localization and improve the navigation of the robot.

Figure 1.6: KUKA robot mounted on a mobile platform for pick and place tasks in industrial environments [9]

Deep Reinforcement Learning Based Mobile Robot Navigation Using Sensor Fusion The problem of unstable and imprecise navigation with learned policies is overcome in the paper [34], which proposes a DRL-based approach for navigation in dynamic environments. The proposed method is based on the Deep Deterministic Policy Gradient (DDPG) algorithm, which is a model-free, off-policy actor-critic algorithm that uses deep neural networks to represent the policy and the critic functions. The proposed method is

evaluated in a simulated environment where the robot learns to navigate effectively and smoothly while avoiding unknown dynamic obstacles. This work shows the right direction towards more robust navigation and obstacle avoidance systems.

1.4. Whole-body mobile manipulator control

In most control approaches to mobile manipulation, base and manipulator operations are separated, since at any given time only one primary control objective is active. This separation principle is then augmented by a switching layer that determines the currently pertinent control objectives. The advantage of such a control formulation lies in its simplicity, i.e., priorities can be separated among the arm and the base with individually designed different control algorithms employed for each subsystem [29].

Instead, we refer to **whole-body control** as a unified control framework that considers the mobile manipulator as a single system (arm manipulator mounted on a mobile wheeled/legged robot). Despite the disadvantage that unified control needs to adhere to a single control framework, it allows for the exploitation of mobile manipulation in the true sense of the term, wherein the manipulator and mobile base can be controlled at the same time. This can lead to several advantages during task achievement and makes the robot more dynamic in terms of its capabilities. The formulation for this type of control involves considering the onboard manipulator as an extended joint space of the mobile base, where the motion controller considers both the base and manipulator state. As a result, base control can be completed simultaneously without affecting much the performance of the end-effector manipulability [29].

1.4.1. MPC+IK for articulated object manipulation

Articulated Object Interaction in Unknown Scenes with Whole-Body Mobile Manipulation In a work conducted by the universities of Zurich and Toronto [18], the researchers demonstrated a successful application of whole-body control for a mobile manipulator. The proposed system introduces a two-stage architecture for autonomous interaction with large articulated objects in unknown environments.

In the first stage, an object-centric planner focuses solely on the object, providing the action-conditional sequence of states for manipulation using RGB-D data. The second stage involves an agent-centric planner that formulates whole-body motion control as an optimal control problem, ensuring safe tracking of the generated plan, even in scenes with moving obstacles. The system proposed in [18] demonstrates effectiveness in handling

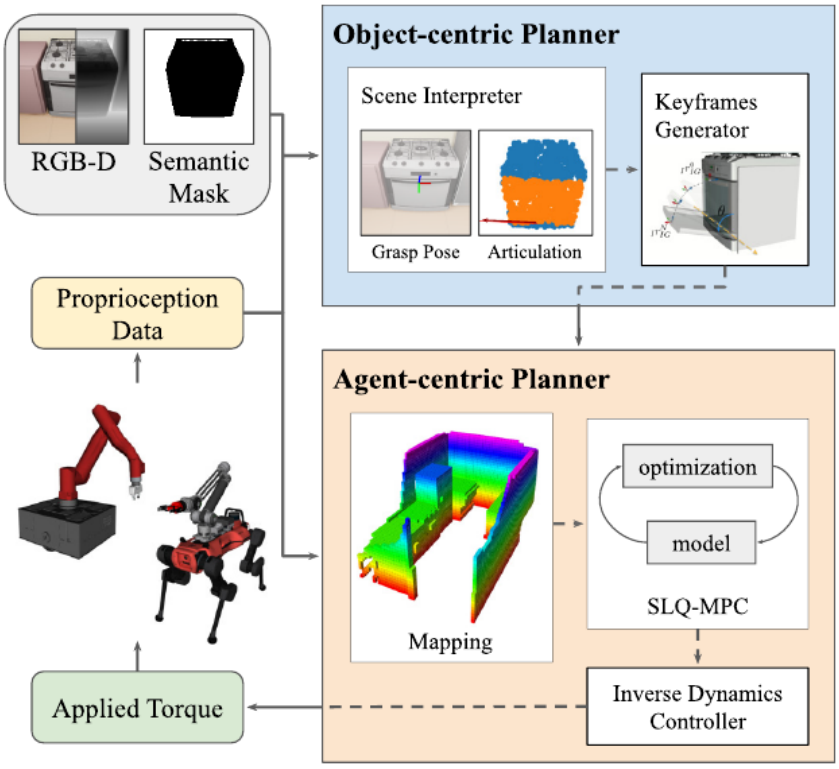


Figure 1.7: The two-level hierarchy in the proposed framework. The object-centric planner comprises a scene interpreter and keyframe generator. It uses perceptual information to generate task space plans. The agent-centric planner follows the computed plan while satisfying constraints and performing online collision avoidance. [18]

complex static and dynamic kitchen settings for both wheel-based and legged mobile manipulators. A comparison with other agent-centric planners reveals a higher success rate and lower execution time. Hardware tests on a legged mobile manipulator further confirm the system's capability to interact with various articulated objects in a real kitchen. The approach combines **object-centric** and **agent-centric** planning, leveraging MPC-based solutions for improved success rates and reduced execution times, particularly in articulated object manipulation scenarios. The contributions include the extension of collision-free whole-body MPC for mobile manipulation, benchmarking in hyper-realistic simulation, and successful hardware experiments showcasing the system's real-world applicability.

The work [18], published in 2022, is a successful real-world application of whole-body control using an MPC-based approach combined with IK solvers for articulated object manipulation. Furthermore, they achieved good performance in both dynamic and static real-world environments, which is a very challenging task for most of the existing methods.

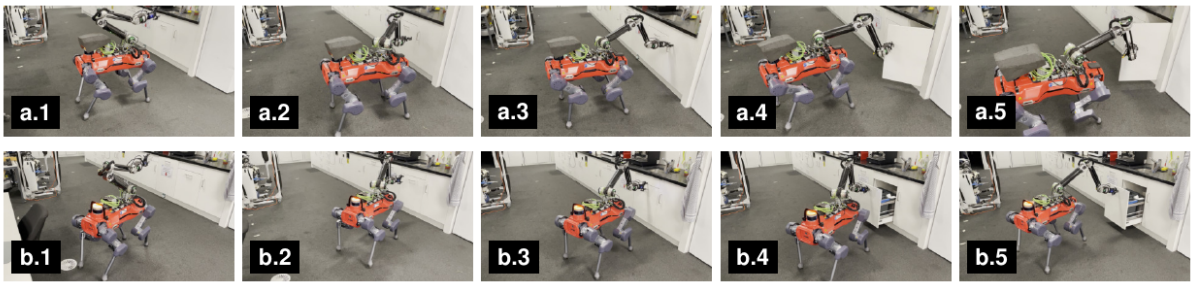


Figure 1.8: Legged mobile manipulation of articulated objects in the kitchen test scenario: (a) Drawer, (b) Cabinet. Throughout the interaction, we set the robot's gait schedule to trot. Only while grasping the handle, the robot enters stance mode. [18]

However, the proposed method is limited mostly concerning the grasping capabilities, which were hard-coded into the known interactive objects (in their case, the kitchen appliances handles), meaning that explicit behavior programming and tuning were needed for the grasping task. They propose data-centric methods to overcome these limitations.

This research proved the feasibility of this approach, which many other researchers claimed to be unfeasible and way too complex to be implemented in real-world applications. However, it is worth noting that the proposed method is not a general solution for all robot hardware configurations, and extending it to other robots would require a lot of effort and time since it boils down to an optimal control problem.

1.4.2. Deep Reinforcement Learning for high DoF control

Deep Whole-Body Control: Learning a Unified Policy for Manipulation and Locomotion The research paper [4] addresses the challenges in controlling legged manipulators with attached arms, proposing a novel approach to learn a unified policy for whole-body control using deep reinforcement learning. The standard hierarchical control pipeline is critiqued for its inefficiency, requiring significant engineering to coordinate arm and leg movements. The proposed method, Regularized Online Adaptation, aims to bridge the Sim2Real gap, and Advantage Mixing is introduced to overcome local minima during training. The authors present a low-cost legged manipulator design and demonstrate that their unified policy enables dynamic and agile behaviors across various tasks.

The paper emphasizes the limitations of current hierarchical models, advocating for learning-based methods like reinforcement learning to reduce engineering efforts and improve generalization. However, it critiques existing learning-based approaches for semi-coupling legs and arms, highlighting issues of coordination, error propagation, and non-

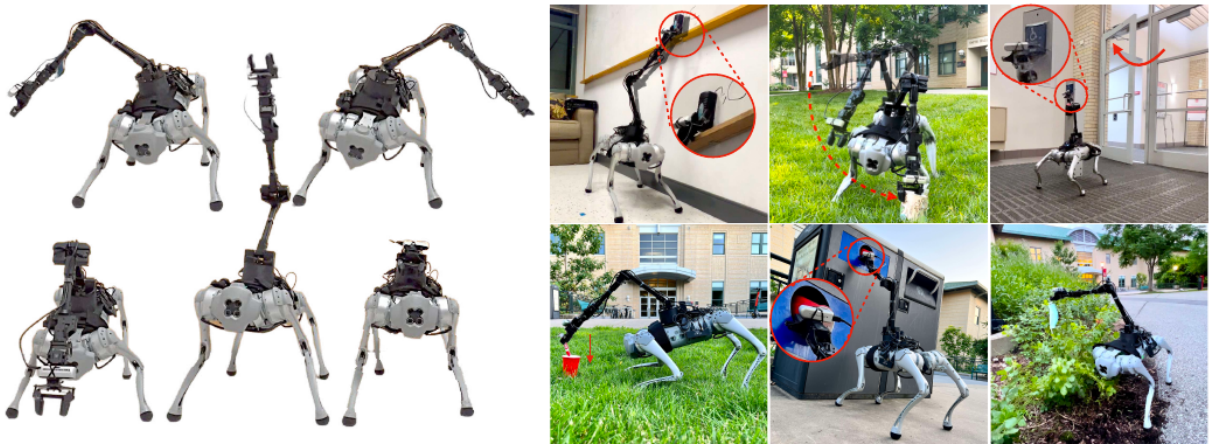


Figure 1.9: Framework for whole-body control of a legged robot with a robot arm attached. The left half shows how whole-body control achieves a larger workspace by leg bending and stretching. The right half shows different real-world tasks, including wiping the whiteboard, picking up a cup, pressing door-open buttons, placing, throwing a cup into a garbage bin and picking in clustered environments. [4]

smooth motions. The proposed unified policy not only allows coordination but also enhances the capabilities of individual components, such as the robot dynamically adjusting leg movements to extend the arm's reach [4].

The challenges in scaling standard *sim2real* reinforcement learning to whole-body control are discussed, including the high degree of freedom, conflicting objectives, and dependencies between manipulation and locomotion. The paper introduces a hardware setup for a low-cost, fully untethered-legged manipulator and outlines a method for learning a unified policy to control both legs and the arm. The authors leverage causal structure in action space and regularization for domain adaptation to enhance stability and speed up learning.

The proposed method is evaluated through tasks like teleoperation and vision-guided tracking, demonstrating successful picking tasks using visual feedback from an RGB camera. Comparative analysis with a baseline method (MPC+IK) across various pick-up tasks measures success rate, average time to completion, IK failure rate, and self-collision rate. The authors conclude by acknowledging the preliminary nature of their results and highlight potential extensions, such as incorporating vision-based policies and addressing challenges in general-purpose object interaction [4].

As the authors of the paper [4] state, the main limitation (but also the core idea behind the control input) is the fact that the robot requires a human operator to provide the

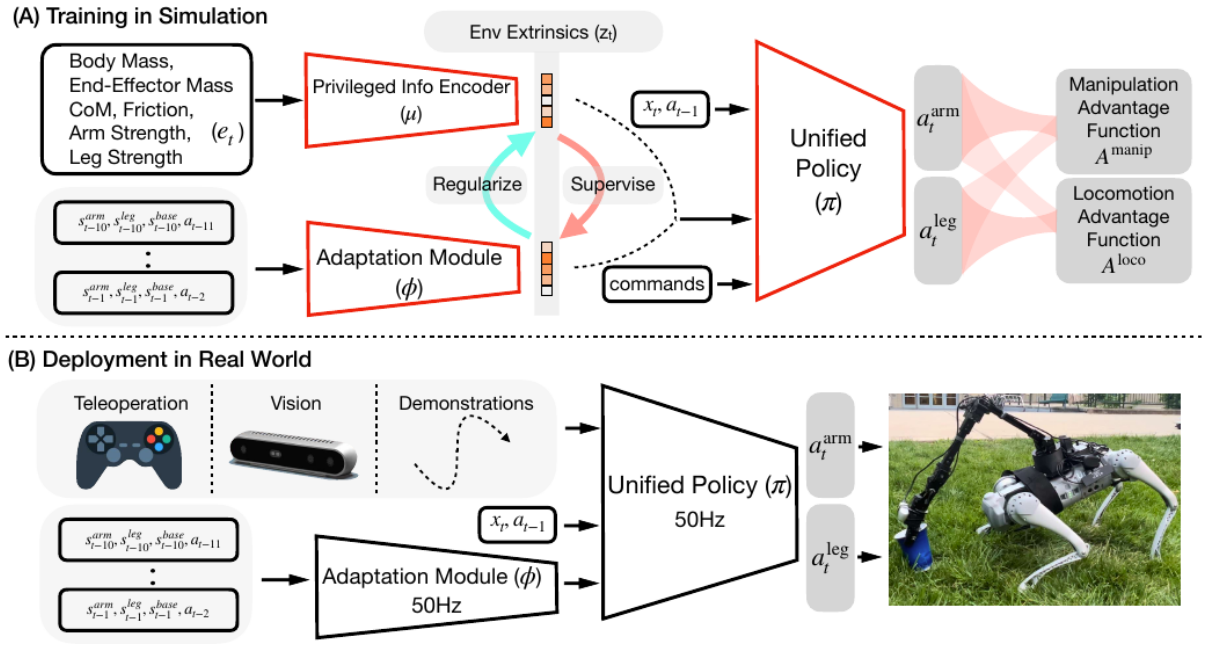


Figure 1.10: Whole-body control framework. During training, a unified policy is learned by conditioning on the environment extrinsic. During deployment, the adaptation module is reused without any real-world fine-tuning. The robot can be commanded in various modes including teleoperation, vision and demonstration replay. [4]

robot objectives, which are then translated into the control inputs. In fact, the robot is not able to autonomously plan its high-DoF motion trajectory, but instead, it is only able to either **track the end effector pose** given by the human operator or to track the April marker in the human's hand. This limitation can be overcome with appropriate task training, but it is not a general solution. However, the proposed method is very promising, since it can achieve very good results in the robot body-hand coordination, which is a very challenging task for most of the existing methods. The movements look very smooth and natural, and the robot can perform simple tasks in a dynamic environment. The April marker tracking mode shows also the effectiveness of the use of a stereo camera for tracking the objective, meaning that promising results can be achieved in other tasks. This research used Nvidia Isaac Gym [19] as a simulation environment [21], which proved to be very powerful for training and overcoming the simulation-to-reality gap.

Learning Mobile Manipulation through Deep Reinforcement Learning [30] This paper presents a mobile manipulation system that leverages deep reinforcement learning for unstructured environments. It integrates state-of-the-art algorithms with visual perception, adopting an efficient framework that separates visual processing from control. This design enables seamless generalization from simulation training to real-

world scenarios, utilizing only on-board sensors. Notably, the **transferability of policies** from simulation to real robots is a key strength, demonstrating the system's autonomy in grasping diverse objects across varied scenarios. The evaluation centers around a challenging mobile picking task, encompassing object recognition, collision-free robot-arm control, and object picking based on the learned policy.

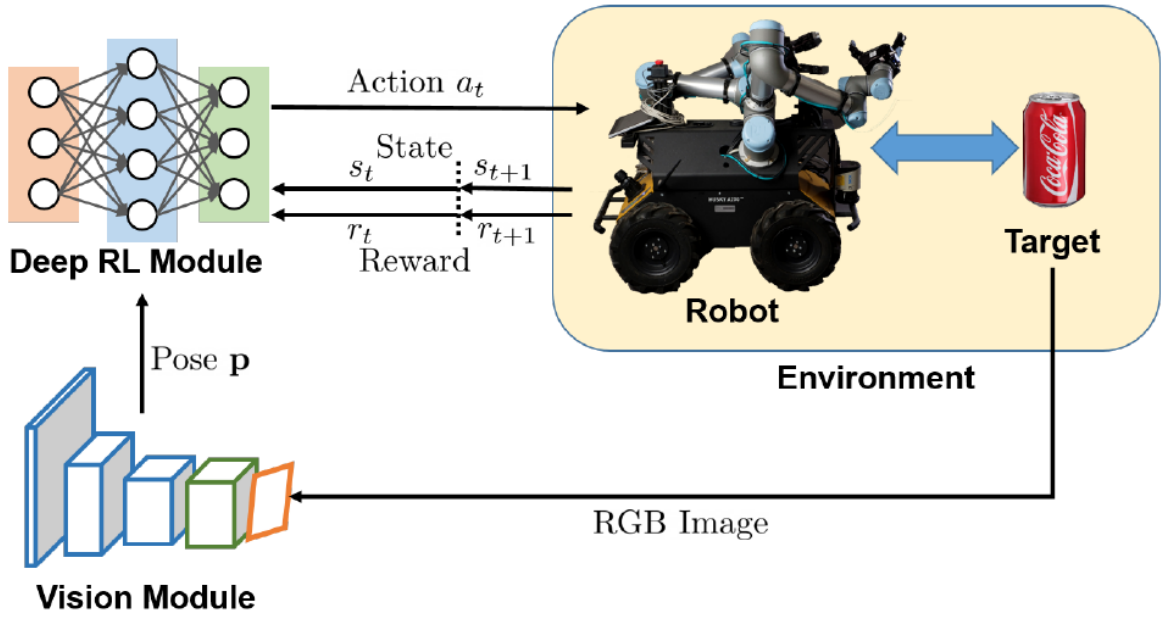


Figure 1.11: Learning-based mobile manipulation control framework. There are mainly two parts, deep reinforcement learning module and vision module. First, the vision module estimates the object 6 degrees of freedom pose p from images captured by an onboard RGB stereo camera. Then, based on the object pose p and current robot state st, deep reinforcement learning module predicts an action for the robot to act. A new state s_{t+1} and a reward r_{t+1} are received after action.[30]

Comparative assessments with state-of-the-art reinforcement learning algorithms highlight the stability and efficacy of the Proximal Policy Optimization (PPO) based system. Real-world experiments further confirm the system's ability to autonomously execute mobile grasping, overcoming challenges posed by the intricate nature of the mobile base, arm, gripper, and vision subsystems. Acknowledging differences between simulation and real-world dynamics, the paper addresses the need for closer coupling between mobile base and arm motions in future work. Overall, this work represents a significant contribution to the field, showcasing the potential of deep reinforcement learning for autonomous mobile manipulation in complex, unstructured environments.

The method proposed in [30] is very promising since it can achieve very good results in robot body-hand coordination, which is a very challenging task, especially in dynamic

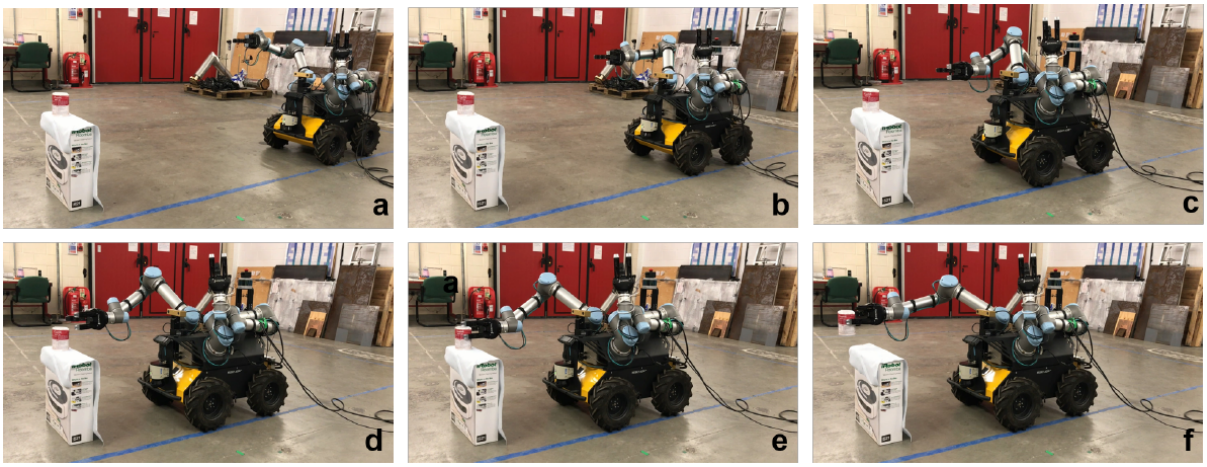


Figure 1.12: Real mobile grasping process for a soup can. (a) is starting, (b,c,d) is approaching, (e) is grasping, and (f) is picking up. [30]

environments. The movements look fluid and without any jitter, and the robot can perform simple tasks in a dynamic environment. This approach is one of the first to achieve complex control using vision-based perception together with deep reinforcement learning. However, it is far from being an optimal solution, since it achieves simple tasks in a very controlled environment.

Multi-Task Reinforcement Learning based Mobile Manipulation Control for Dynamic Object Tracking and Grasping This research paper [31] is a continuation of work demonstrated in their previous research paper [30] mentioned above. [31] addresses the challenges associated with agile control of mobile manipulators in unstructured environments, particularly focusing on dynamic object tracking and grasping. The authors propose a multi-task reinforcement learning-based control framework that aims to achieve general dynamic object tracking and grasping capabilities. The framework utilizes various dynamic trajectories as a training set, incorporating random noise and dynamics randomization to enhance policy generalization.

Experimental results demonstrate the trained policy's ability to adapt to unseen dynamic trajectories, achieving a $0.1m$ tracking error and a 75% grasping success rate for dynamic objects. The proposed method is successfully deployed on a real mobile manipulator, showcasing its potential for real-world applications. The contributions of this work include the development of a versatile control framework and its successful deployment in unstructured environments, addressing the challenges of mobile manipulation with dynamic objects.

In [31] they use the proximal policy optimization (PPO) algorithm to train and learn a

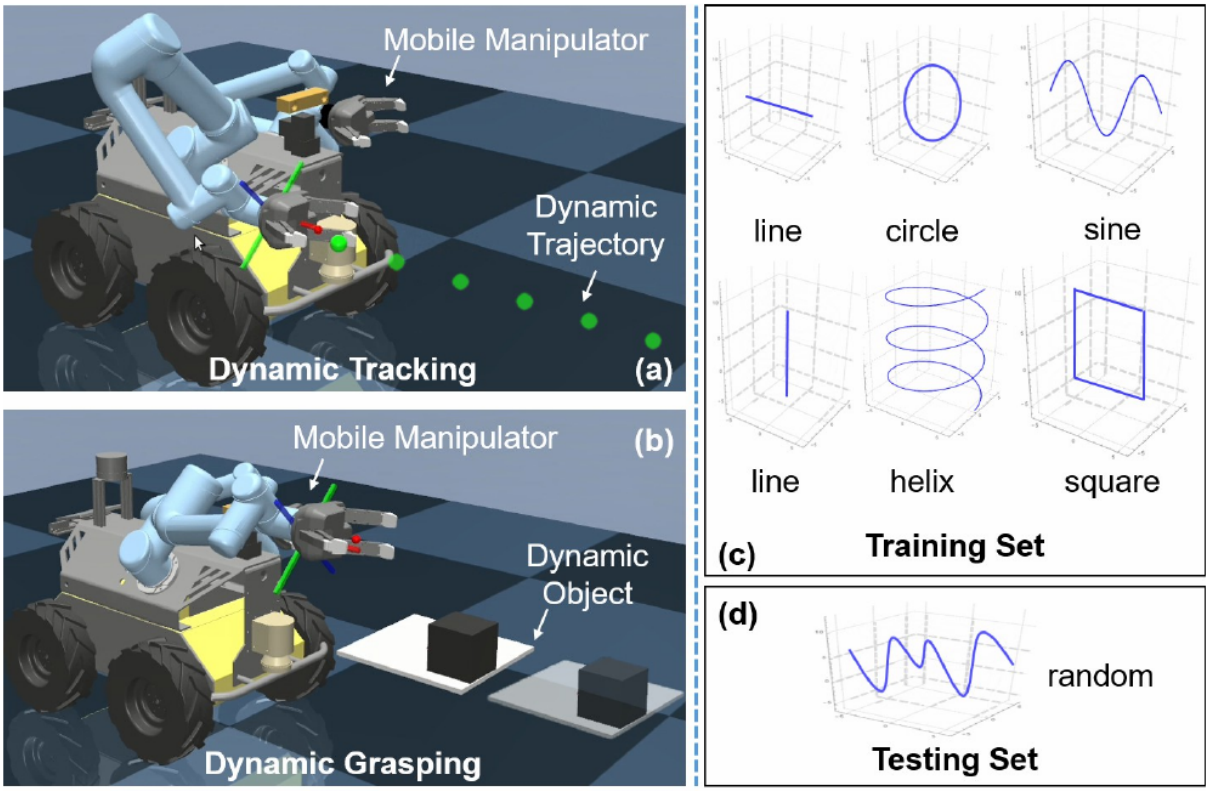


Figure 1.13: (a) Dynamic trajectory tracking task with a mobile manipulator. (b) Dynamic object grasping task with a mobile manipulator. (c) Several basic trajectories as multi-task RL training sets. (d) Random trajectories as multi-task RL testing set.[31]

policy, but the method is general and can be applied to most on/off-policy RL algorithms. PPO is one of the state-of-the-art RL algorithms that is easy to implement and tune, and performs relatively well. The policy is learned through a deep neural network.

The images 1.14 above show the training and testing process of the proposed method. They created a realistic simulation environment that enabled efficient parallel training of the policy, and simulation of the dynamic objects and tracking. They also managed to correctly transfer the learned policy to the real robot, which is a very challenging task for most of the existing methods. Artificial noise addition was essential for the training process since it allowed the policy to generalize better to unseen trajectories and environments.

Overall, this research provides valuable insights and a practical solution for advancing the field of agile mobile manipulation. This is one of the very few works that addresses the problem of dynamic object tracking and grasping, which is a very challenging task for most of the existing models, due to the difficulty in modeling and the extensive training required. Although the model doesn't show very high success rate in the tasks shown, it

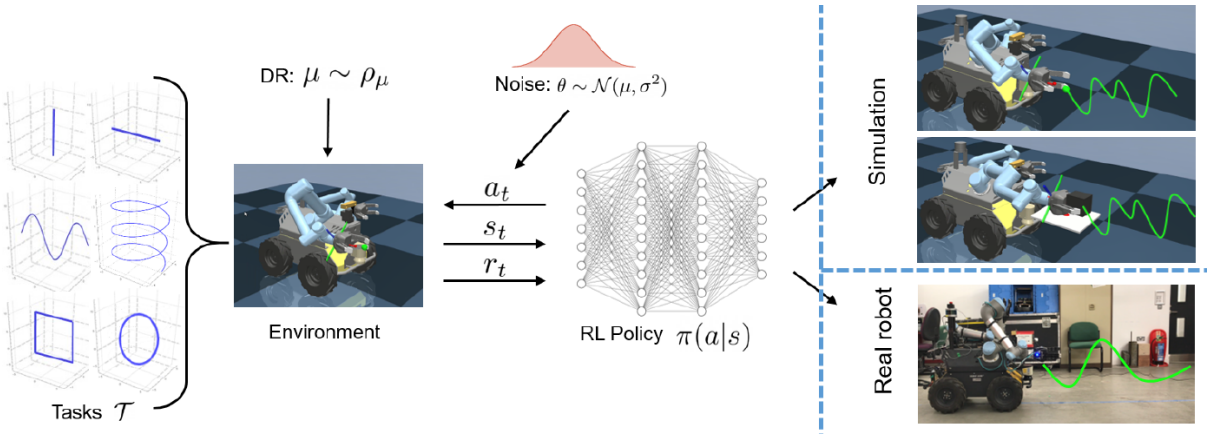


Figure 1.14: (a) In the multi-task RL training, six basic trajectories are used as the task training set to train a general policy. To improve the robustness, gaussian noise is added to the action and observation space in each training episode. (b) The RL testing includes simulation and real-world for policy evaluation.[31]

shows promising results and a direction of research that can be further explored.

1.4.3. Mobile manipulation with Imitation Learning

Imitation learning from human-provided demonstrations is a promising tool for developing generalist robots [5], as it allows people to teach arbitrary skills to robots. Indeed, direct behavior cloning can enable robots to learn a variety of primitive robot skills ranging from lane-following in mobile robots to simple pick-and-place manipulation skills to more delicate manipulation skills like spreading pizza sauce or slotting in a battery. However, many tasks in realistic, everyday environments require whole-body coordination of both mobility and dexterous manipulation, rather than just individual mobility or manipulation behaviors. Two main factors hinder the wide adoption of imitation learning for mobile manipulation.

- We lack accessible, plug-and-play hardware for whole-body teleoperation. Teleoperation for whole-body control requires a human to provide demonstrations of the task to the robot, using a specific and often expensive hardware setup. Furthermore, mobile manipulators, especially bimanual mobile manipulators, can be costly if purchased off the shelf.
- Prior robot learning works have not demonstrated high-performance bimanual mobile manipulation for complex tasks. The same goes for single manipulators in complex tasks in dynamic, cluttered or unknown environments.

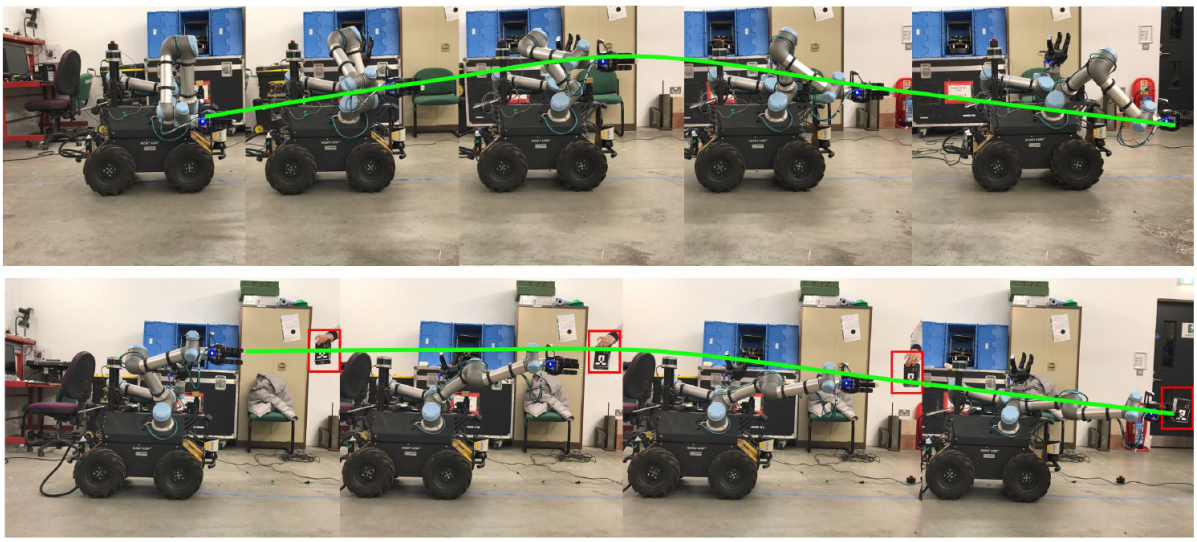


Figure 1.15: Snapshots of the real robot experiments. The upper row shows a mobile tracking process in which the end-effector tries to track the target trajectory. The lower row shows a mobile grasping process in which the object moves randomly. [31]

Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation On the hardware front, the researchers from Stanford University in their paper [5] present a low-cost and whole-body teleoperation system for collecting bimanual mobile manipulation data. Mobile ALOHA extends the capabilities of the original ALOHA [35], the low-cost and dexterous bimanual puppeteering setup [35], by mounting it on a wheeled base.

Figure 1.16: Low-cost mobile manipulation system that is bimanual and supports whole-body teleoperation. The system costs 32k USD including onboard power and compute. Left: A user teleoperates to obtain food from the fridge. Right: Mobile ALOHA can perform complex long-horizon tasks with imitation learning. [5]

The main contribution of the research paper [5] is the hardware setup of a low-cost

bimanual mobile manipulation system supporting whole-body teleoperation. Using data collected with Mobile ALOHA, performing supervised behavior cloning results in more accurate behavior mimicking. Also co-training with existing static ALOHA datasets [35] boosts performance on mobile manipulation tasks. The robot can not yet improve itself autonomously or explore to acquire new knowledge. In addition, the Mobile ALOHA demonstrations are collected by two expert operators.

The software architecture upon which the robot system in figure 1.16 relies is the one presented in [35], which is a modular architecture that allows for easy integration of functionalities. The architecture is described below.

Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware Fine manipulation tasks, such as threading cable ties or slotting a battery, are notoriously difficult for robots because they require precision, careful coordination of contact forces, and closed-loop visual feedback. Performing these tasks typically requires high-end robots, accurate sensors, or careful calibration, which can be expensive and difficult to set up. The researchers in [35] present a low-cost system that performs end-to-end imitation learning directly from real demonstrations, collected with a custom teleoperation interface. Imitation learning, however, presents its own challenges, particularly in high-precision domains: errors in the policy can compound over time, and human demonstrations can be non-stationary. To address these challenges, they developed a simple yet novel algorithm, Action Chunking with Transformers (ACT), which learns a generative model over action sequences. ACT allows the robot to learn multiple difficult tasks in the real world.

The proposed method is evaluated on a wide variety of complicated tasks, and the robot can perform well in most of them. However, their method has some limitations, mostly due to the difficulty in perception of the environment and the lack of training data. However, the architecture proposed shows very promising results from a few training examples.

In some extremely complex tasks (the ones in which even humans find some difficulty), the robots fail consistently, as reported in [35]. The failures can then be attributed to the difficulty in perception and lack of data, since the input video sequences were created from standard RGB cameras, therefore no depth information was available. Furthermore, object semantic understanding would have been very useful for the robot to understand the task deeply and perform better. The lack of data is also a big issue since the robot was trained by teleoperation. They believe that pretraining, more data and better perception are promising directions to tackle these extremely difficult tasks.

Shortcomings of imitation learning approaches The main shortcomings of imitation learning approaches are the lack of generalization and the difficulty in perception

of the environment. Imitation learning is feasible in a general context and environment proven that it is given enough data and the right perception tools:

- A large dataset of demonstrations collected with human teleoperation
- Human-made demonstrations available offline for training and evaluation
- A hardware component with a similar kinematic structure to the human demonstrator. The hardware-mimicking structure must also not occupy too much space, since it would affect the real robot's physical constraints.
- Mobile arms are controlled by each degree of freedom since it would not be possible to give a proper demonstration by inputting only the end-effector pose and getting the joints' positions by inverse kinematics.
- Appropriate depth perception and semantic object understanding would be ideal for the robot to understand the task deeply and perform better.

1.4.4. Comparison of Model-Based and Data-Driven approaches

Table 1.1: Summary of the main differences between model-based and data-driven approaches for robotic manipulator controls

Approach	Model-Based	Data-Driven
Control Strategies	• Model Predictive Controllers (MPC) • Whole-Body Inverse Kinematics (IK) Solver	• Deep Reinforcement Learning (DRL) • Imitation Learning
Features	• Requires explicit modeling of system dynamics and kinematics • Suitable for simple tasks, unsuitable for complex tasks • Planning over end-effector pose or grasp in the workspace	• No explicit modeling of system dynamics • Learning from experience in simulation environments • High-level planning over tasks, object detection, manipulation or other objectives
Interpretability and Adaptability	• Explitic modeling implies high system interpretability • Adaptable to many tasks but requires behaviors re-programming	• Learned policies have very limited interpretability • Learning from experience allows high adaptability, given proper GPU-parallelized training
Advantages	• Small simulation-to-reality gap • Adaptable to many tasks but with explicit programming • No training required • Safer operation due to explicit physical limitations modeling	• No explicit modeling of system dynamics required • Learning from experience allows high generalization • Can perform well in unknown or dynamic environments • Can provide high body-hand movement coordination
Disadvantages	• Requires very accurate physical models for seamless integration • Doesn't perform well in complex tasks or dynamic environments • Difficult to adapt to complex tasks (low generalization) • High computational cost for the solver in high DoF systems	• Requires large amounts of training data and extensive training • Very long time needed to fine-tune the hyperparameters • May result in unstable and jiggly movements • May result in unsafe behaviors in real-world applications if not properly trained • Suffers a lot from the simulation-to-reality gap

1.5. Addressing the Simulation-to-Reality Gap

The simulation-to-reality gap is a well-known problem in robotics, which is the difference between the performance of a robot in simulation and the real world. The simulation-to-reality gap is a major challenge in robotics, as it is difficult to accurately model the real world in simulation. This is especially true for mobile manipulation, where the robot must interact with the environment to perform its task.

Many works addressed this problem explicitly in the past, such as [12] and [7], proposing methods to make simulations more realistic and to improve the generalization of the learned policies.

Nvidia has developed some tools, such as Nvidia Isaac Gym [19] and Nvidia Isaac Sim [21], which are very powerful tools for simulating robots and environments. Isaac Sim is the simulation engine for Isaac Gym, which is a framework for training and testing robots in simulated environments with DRL and other tensor-based ML techniques. Isaac Gym is built on top of Nvidia Omniverse, which is a platform for real-time simulation and collaboration. Nvidia Omniverse allows also bridging the simulation with ROS thanks to the Isaac ROS bridge [20]. Overall, these tools enable reducing the simulation-to-reality gap, since they support many different robots, environments and perception sensors, so they are very powerful for training and testing DRL policies if properly set up.

A Sim-to-Real Pipeline for Deep Reinforcement Learning for Autonomous Robot Navigation in Cluttered Rough Terrain A work from Toronto [7] proposes a sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. Sim-to-real strategies have been developed for robot navigation tasks. For example, domain randomization can be applied to visual parameters such as texture, lighting, and object placement in synthetic environments to improve generalizability. The paper [7] addresses the **sim-to-real gap** in training robots for 3D terrain navigation by incorporating three sim-to-real strategies. Firstly, to account for depth camera measurement errors in 3D mapping that affect terrain steepness accuracy, the authors vary terrain steepness during training using a uniform distribution.

Secondly, the paper tackles disturbances in robot motion during interactions with rough terrain, such as slippage and insufficient traction. Both 3D terrain interactions and latency from visual odometry measurements contribute to disturbances in robot travel distance and yaw rotation angle. The third strategy focuses on addressing robot pose estimation errors arising from image and feature association errors. These errors, caused by lens distortion and ambiguous features, impact the accuracy of the robot's estimated 6 DOF

pose. The paper integrates these errors into the inputs of the DRL network to improve the model's performance in the face of localization inaccuracies.

1.6. Object Detection and Grasping

Grasp planning for mobile manipulators is a challenging problem tackled in several ways in the literature. On the one hand, grasping requires coordination within a very challenging high-dimensional constrained configuration space (mobile base, manipulator, gripper). On the other hand, grasping requires detecting objects and constructing data-driven geometrical representations, to produce effective grasping plans in the presence of statistical data uncertainties. Many of the traditional grasp planners (designed for stationary manipulators) can be used for mobile manipulators once the mobile base has been fixed.

However, a **generic grasping pipeline** is desirable, which achieves arm-base-gripper coordinated grasping given the information about object pose and the operating environment. Such coordinated manipulator and mobile base motion approaches are explored to find a feasible grasp or to identify an approach position that can lead to a successful grasp. (and only the manipulator moves for grasping). This may not be optimal as grasping can happen with the mobile base and the manipulator moving when the gripper is closing [29].

Broadly, automated grasping can be categorized into the following approaches [1]:

1. grasp using prior information from scene/objects
2. grasp using hand-eye coordination through learning directly from raw sensor data
3. grasp using template matching
4. grasp by detecting proper grasping pose using deep learning-based approaches
5. other field-specific approaches

Systems in each category have one or more limitations that are detailed in the following subsections. The majority of the existing systems are static, where a robotic system is fixed in an environment surrounded by the objects in its workspace.

Automated Object Manipulation Using Vision-Based Mobile Robotic System for Construction Applications The system designed and deployed for pick-and-place in a structured construction environment [1] integrates scene understanding and autonomous navigation with object grasping. To achieve this, two stereo cameras and a robotic arm are mounted on a mobile platform. This integrated system uses a global-to-

local control planning strategy to reach the objects of interest (i.e., bricks, wood sticks, and pipes). Then, the scene perception, together with grasp and control planning, enables the system to detect the objects of interest, pick them, and place them in a predetermined location depending on the application. The system is implemented and validated in a **construction-like environment** for pick-and-place activities. The results demonstrate the effectiveness of this fully autonomous system using solely onboard sensing for real-time applications with end-effector positioning accuracy of less than a centimeter.

However, the researchers mention also the shortcomings of the system, since the robot was developed for a field-specific application and, therefore non-adaptable to more generic use case scenarios. The system uses a heuristic-based approach to detect the bricks to grasp and pick up. Furthermore, the navigation pipeline relies on a static environment with no dynamic obstacles, since the arm manipulator does not employ any collision avoidance in the trajectory planning [1].

Autonomous Robotic Manipulation: Real-Time, Deep-Learning Approach for Grasping of Unknown Objects The work [26] proposes a novel approach for grasping unknown objects. The researchers present a full grasping pipeline proposing a real-time data-driven deep-learning approach for robotic grasping of unknown objects using MATLAB and deep convolutional neural networks. The proposed approach employs RGB-D image data acquired from an eye-in-hand camera centering the object of interest in the field of view. The arm control is based on **visual servo-ing techniques**, i.e. the robot arm is controlled based on the visual feedback from the camera. Their approach aims at reducing propagation errors and eliminating the need for complex hand-tracking algorithms, image segmentation, or 3D reconstruction, which are often either infeasible or too prone to errors. The proposed approach can efficiently generate reliable multi-view object grasps regardless of the geometric complexity and physical properties of the object in question. The system employed is a 7-DoF robotic manipulator controlled with an IK solver and a parallel gripper with overactuated fingers.

One of the main limitations of the approach in [26] is the fact that the grasping pipeline is implemented in MATLAB, therefore hardly portable and hardly replicable on other robotic hardware. Furthermore grasping with a parallel gripper is very limited, and the approach can work well with a limited set of objects. The authors demonstrated a good grasping capability with many different and unknown objects, but the approach cannot be generalized well without creating a multi-view perspective of the target object, to gain more understanding of the object's shape and geometry. Although the approach is very promising, there is room for improvement, especially in the CNN for grasping pose detection, which is the core of the grasping pipeline.

1.6.1. Grasping Soft Objects

Grasping soft objects is a challenging task for robotic manipulators, as it requires precise and effective control of the robot's end-effector to avoid damaging the object. Soft objects can deform under the pressure of the robot's gripper, making it difficult to grasp them effectively. Many existing methods for grasping soft objects rely on force control to regulate the pressure applied by the robot's gripper, but these methods can be difficult to implement and may not be effective for all types of soft objects.

Instead, some other solutions rely on open-loop control strategies to grasp soft objects, which do not require force feedback but may not be as effective as force control methods. These strategies employ **soft fingers or force-compliant grippers** that can conform to the shape of the object, reducing the risk of damage during grasping. Soft fingers can be made from materials like silicone or rubber that can deform to fit the shape of the object, providing a more secure grip without applying excessive pressure.

The inherent difficulty in grasping soft objects is the lack of a well-defined grasp configuration, as the object can deform and change shape during the grasping process, as well as the inevitable deformation of the soft gripper itself. This makes it difficult to predict the behavior of the object and the gripper during the grasping process. As a result, creating simulated environments for training and testing grasping policies for soft objects is challenging, as it requires accurate modeling of the object's deformation properties and the gripper's compliance to ensure realistic results.

Many existing methods for grasping soft objects that rely on force or torque control can be difficult to adapt to different types and shapes of soft objects. This often makes the only viable solution to create control strategies pre-tuned for specific soft objects, which limits the applicability of the method to only known objects of a specific shape but can be effective for many applications. The other difficulty in implementing adaptable control strategies for grasping soft objects is the need to create ad-hoc solutions engineered for specific soft gripper configurations, which are hard to generalize to other robots' end effectors or grippers.

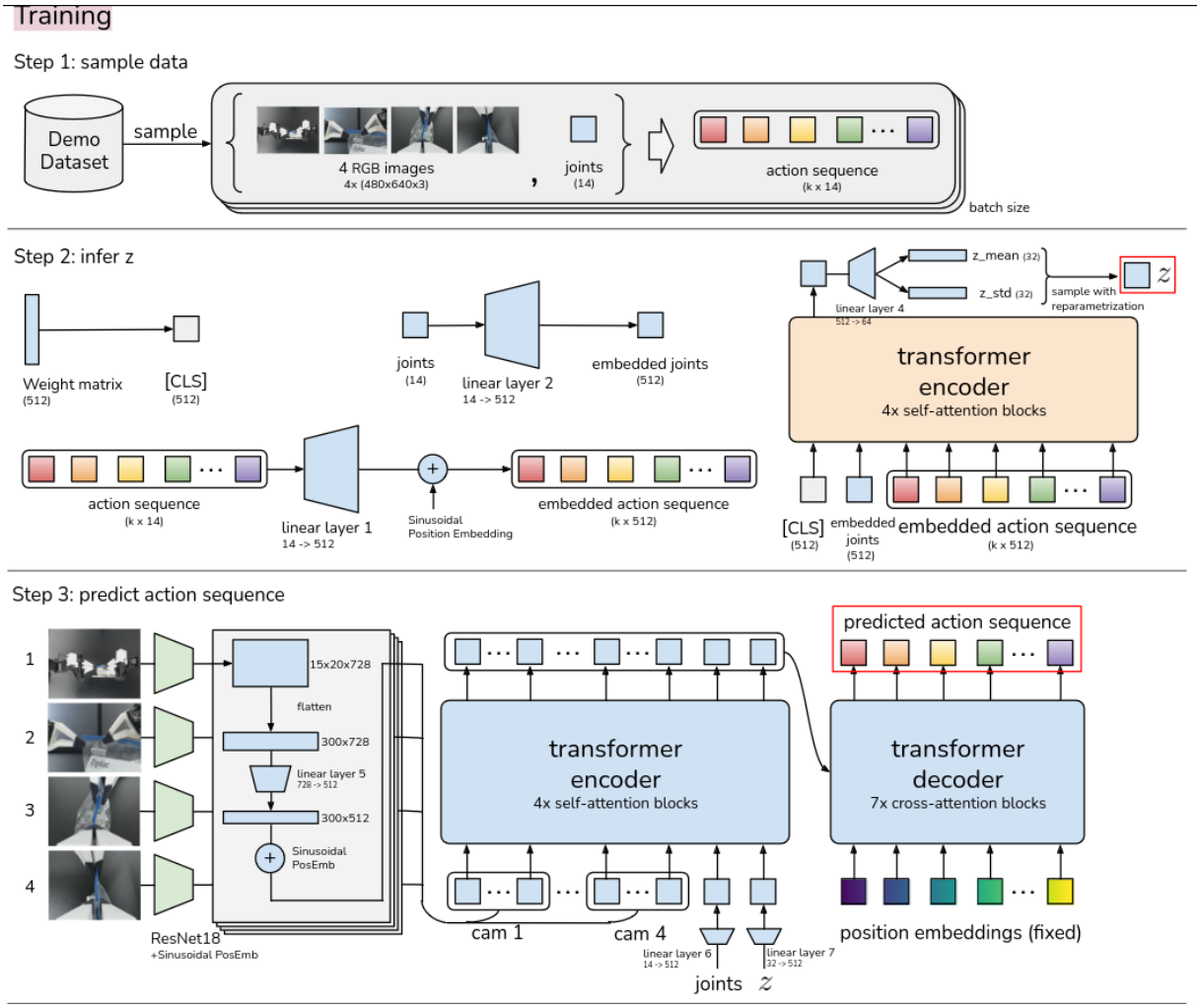


Figure 1.17: Detail architecture of Action Chunking with Transformers (ACT). First, they train ACT as a Conditional VAE (CVAE), which has an encoder and a decoder. The encoder of the CVAE compresses action sequence and joint observation into z, the style variable. The encoder is discarded at test time. The decoder or policy of ACT synthesizes images from multiple viewpoints, joint positions, and z with a transformer encoder, and predicts a sequence of actions with a transformer decoder. z is simply set to the mean of the prior (i.e. zero) at test time [35].

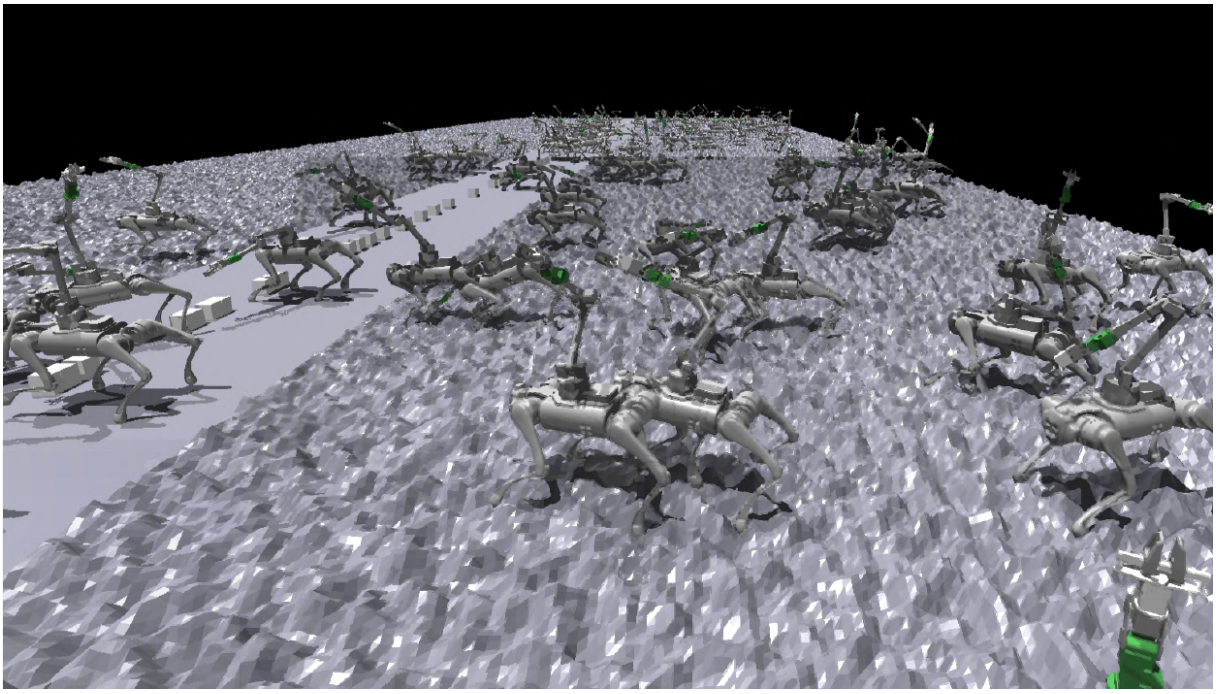


Figure 1.18: Simulated environment with Nvidia Isaac Gym [20]. The screenshot depicts a simulated environment with many legged mobile manipulators trained in parallel using DRL, as in [18]. The environment is simulated using Nvidia Isaac Sim [21]

.

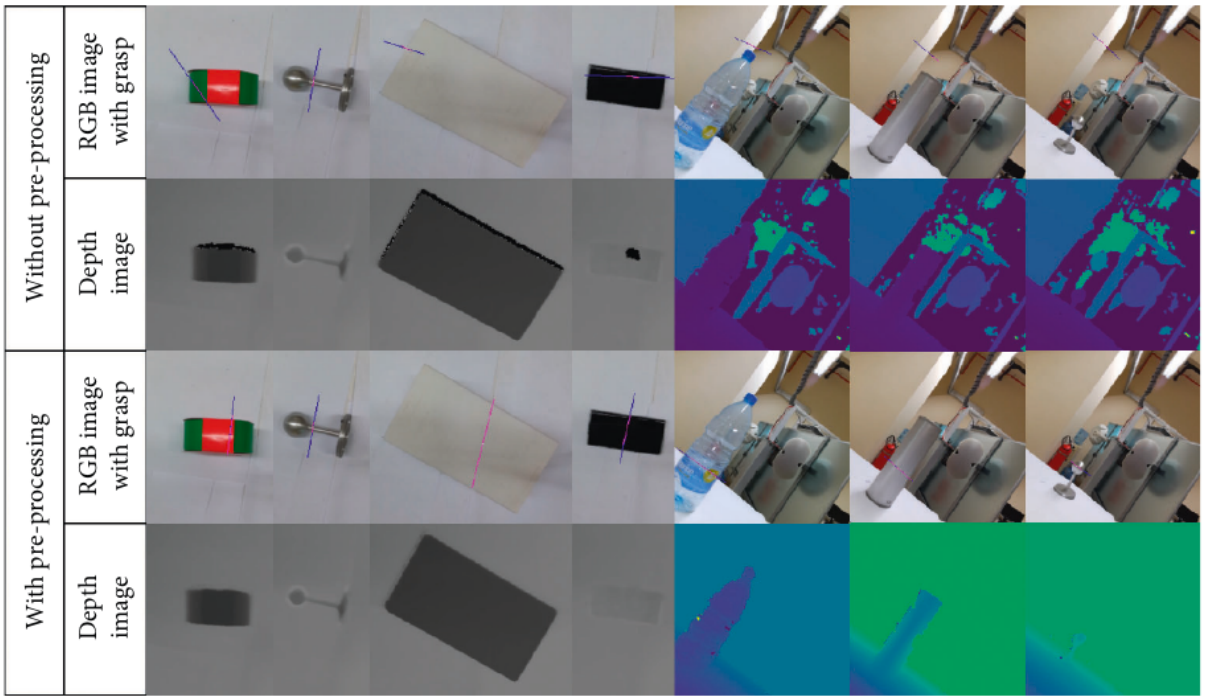


Figure 1.19: Grasp generation results: comparison between grasp generated by the GG-CNN with and without RGB-D image preprocessing for shiny and black objects [26]

2 | Robotic Platform for Mobile Manipulation

This chapter describes the hardware configuration for the robotic platform used for the project. The platform is composed of a mobile robot base, a robotic arm manipulator, sensors, a soft gripper actuator, 3D-printed mounts, batteries, power management systems, and other electronic devices. The chapter will also discuss the issues faced during the development of the mobile manipulation platform.

2.1. Mobile Robot Platform

The mobile robot is an *AgileX Scout 2.0* robot, depicted in Figure 2.1. This robot is a skid-steering robot, suitable for outdoor and indoor environments. Designed for robotics research and development, the Scout 2.0 is an unmanned ground vehicle (UGV). This autonomous mobile robot offers a robust mechanical design along with capable mobility performance. Built to endure diverse conditions, Scout 2.0 features rugged materials and protective casings, ensuring longevity and reliability during missions. Scout 2.0 offers aluminum T-slot rails for secure mounting of external sensors or kits. On these rails, a variety of sensors, computers, or other devices are mounted, allowing the creation of a mobile robotic platform for a wide range of applications, all powered by the **onboard battery system**.

It supports CAN bus protocol for connections and provides open-source SDK and software resources for expanded capabilities. Its maximum speed is $1.5m/s$, and it can carry a payload of 50 kg. The robot is powered by a $24V$ battery, which provides a range of 15 km maximum. The robot is controlled by a ROS-based software system, which allows the control of the robot's speed using a ROS topic and receiving odometry data from the robot's encoders.

On the mobile robot, an *Intel NUC 12* computer 2.2 is mounted. This **computer** is used to run all the control software and the perception algorithms. The computer is connected

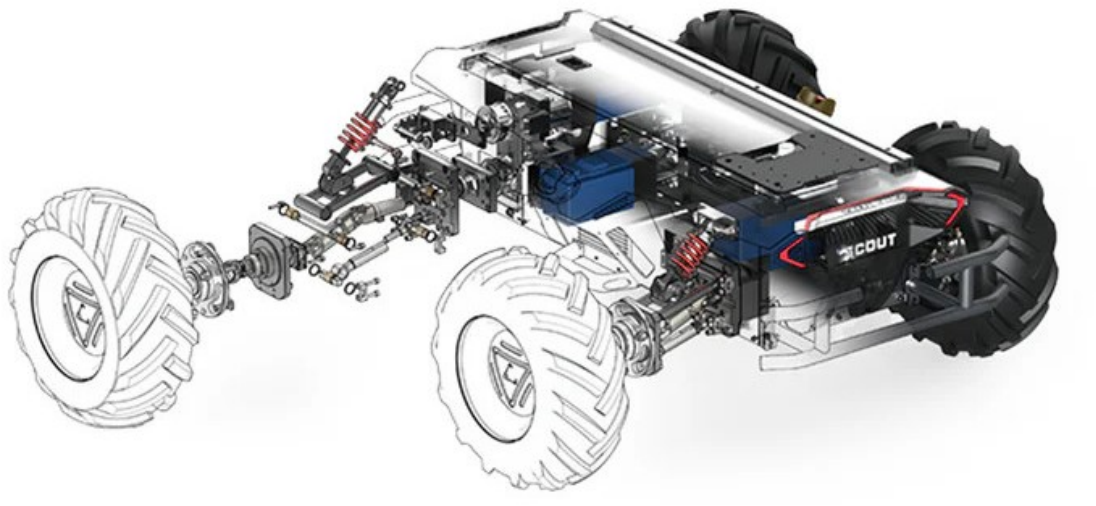


Figure 2.1: Scout 2.0 employs 200W brushless servo motors to drive each wheel independently. Its double-wishbone suspension with shock absorbers ensures stability on rough terrain, enabling it to tackle obstacles up to 10cm tall effortlessly.

to a switch, which is used to connect the computer to the robotic arm's embedded control system, to the LiDAR sensor, and Wi-Fi hotspot router. The mobile robot base is connected to the computer via the CAN bus, which is used to receive data from the robot's encoders and send data to control the robot's speed and direction. The on-board router is used to provide a Wi-Fi hotspot for remote access. The computer's Wi-Fi is used to connect to the internet, allowing the robot to be controlled and monitored remotely via a remote desktop connection. This computer has the following technical specifications:

- Intel Core i7-12700H CPU
- 32GB DDR4 RAM
- 1TB NVMe SSD
- Intel Iris Xe Graphics
- Kubuntu 22.04 operating system

The robot is equipped with an *TP-Link Archer MR200* router and a *Netgear GS108* switch. The router was necessary to establish a **remote connection** from a personal laptop to the robot's computer, allowing the control and monitoring of the robot from a remote location. This is essential for the development of the project, as it allows to work safely with the robot, ensuring that everything works smoothly and stopping the system in case of any unexpected behavior or software crashes and malfunctions.

Figure 2.2: Intel NUC 12 computer mounted on the robot

The robotic arm manipulator is also connected to the switch, allowing the robot's computer to control the manipulator and receive data from its motors' encoders. The LiDAR is connected to the switch, allowing the robot's computer to receive pointcloud data from the LiDAR sensor, at a fast transmission rate.

2.2. Robotic Arm Manipulator

The robotic arm manipulator used for the project is a *Igus ReBeL 6-DoF* **cobot**, depicted in Figure 2.3. Cobot is a term used to describe a collaborative robot, which is a robot designed to work alongside humans in a shared workspace. This cobot is a lightweight, compact, and affordable robotic arm, suitable for research and development in robotics. It is produced by the German company Igus, which specializes in the production of robotic components for low-cost automation. The robotic arm is composed of six joints, each driven by a DC motor with an integrated encoder. The outer contour and mechanical components of the ReBeL utilize Igus plastic polymers, making it particularly inexpensive and the lightest cobot on the market. Its lightweight (8.2 kg) and compact design make it suitable for mounting on top of mobile robot platforms, such as the Scout robot, without affecting the robot's mobility and stability. The maximum payload of the arm is 2 kg, compatible with the project's requirements.

The **advantages** of the ReBeL cobot are:

- Lightweight and compact design
- Plastic arm, inexpensive and cost-effective
- Easy to install and operate with the provided software, whose interface is shown in 2.4

Figure 2.3: Igus ReBeL 6-DoF robotic arm

- Plug and play proprietary control system, or open-source control option
- Can be powered with compact batteries other than the power supply

The **disadvantages** of the ReBeL cobot are:

- Limited reach and workspace due to the joints' design and physical constraints
- Limited precision and repeatability
- The plastic gear components are not as durable and reliable as metal components

This robotic arm was the ideal choice for the project since the open-source control option allows the development of the control software for the arm, based on ROS2 software packages. The lightweight and compact design made it suitable for mounting on top of the SCOUT 2.0 robot, without affecting the robot's mobility and performance. The arm's easy installation and operation allows one to quickly set up the arm and start developing the control software and perception algorithms for the project. The robotic arm is provided with its external power supply, but it can also be powered using batteries since it does not require a high current to operate. This feature is essential for the project, as the robotic arm can be mounted on a mobile robot platform without relying on a power supply connected to the wall outlet. The batteries used to power the robotic arm are placed on top of the mobile robot, as shown in 2.17.

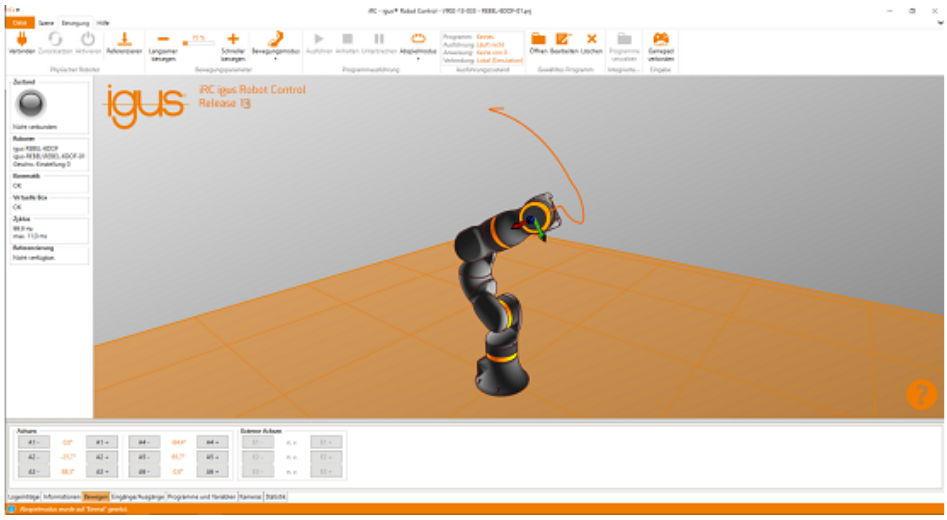


Figure 2.4: Igus ReBeL proprietary control software interface

Issues with Igus Rebel motors' encoders and calibration

Due to the low-cost design of the ReBeL cobot, the arm's motors are equipped with encoders that are not as accurate as the encoders used in more expensive metal-only robotic arms. This caused the arm's positioning accuracy to be lower than the project's requirements, and the repeatability to be not good enough for the project's objectives. The accuracy specified in the technical documentation of the arm is $\pm 0.1mm$, which is acceptable for the project's needs, but the arm's actual accuracy was not as good as specified. Many tests were conducted to evaluate the arm's end effector positioning accuracy, and the results showed that the error changes depending on the arm's configuration and the height from the base of the arm's flange. The arm's accuracy is not consistent, and the repeatability suffers from it. The tests showed also that the arm's accuracy is not dependent on the calibration of the arm's motors but on the backlash and clearance of the gear components, which are not adjustable and cannot be calibrated.

A characteristic of the robotic arm's motors is that they have a certain amount of backlash and clearance, which affects the amount of free movement of the arm's joints before the motor starts to move the joint. This backlash and clearance in the gear components results also in the arm jiggling and vibrating when moving to a different joint position, and not reaching the desired position with a smooth velocity profile. Furthermore, the arm's internal gears are made of plastic, which is not as resilient and strong as metal gears. This results in the arm suffering from its own weight, and the end effector moving a few centimeters down vertically from the given pose when the end effector moves far away from the center of gravity of the robot's base. This is a critical issue, as the position

error does not depend linearly on the arm's configuration, and the arm's accuracy is not consistent across the workspace. This characteristic of the robotic arm implies that an artificial compensation of the arm's positioning error is necessary, to ensure that the arm's end effector reaches the desired position with higher precision, even though the error cannot be completely eliminated.

2.3. Sensors and Perception

The mobile robot platform is equipped with several sensors, which are essential for the project's objectives. The sensors used are perception systems that are used for mapping and localization, obstacle avoidance, object detection, and recognition. Two main sensors are installed on the mobile manipulation robot: a 3D LiDAR sensor and an RGB-D stereo camera sensor. The LiDAR sensor is in a fixed position, on top of the mobile robot base to have a 360° field of view, while the camera sensor is mounted on the robotic arm's flange, allowing the camera to move with the arm's end effector.

2.3.1. 3D LiDAR

The main sensor for environment perception for localization and navigation is mounted on top of the sensors framework, a structure placed on top of T-slot rails that are bolted to the mobile robot's chassis. The sensor is a *Ouster OS1-64* LiDAR sensor, as shown in Figure 2.5. The OS1 offers clean, dense data across its entire field of view for accurate perception and crisp detail in industrial, automotive, robotics, and mapping applications. This sensor is a **64-plane LiDAR sensor**, capable of providing a 360-degree field of view with a range of 120 meters. The sensor has a resolution of $\pm 0.1cm$ and a stable scan rate of $10Hz$ at 1024 points resolution. The vertical and horizontal scan resolution are $\pm 0.01°$. Its minimum range of 0.5 meters makes it suitable for indoor environments, while its maximum range of 120 meters makes it suitable also for outdoor environments. This sensor was employed to create maps of the environments and also to localize the robot within the environment. It proved also useful for dynamic obstacle avoidance, thanks to its high resolution and scan rate.

2.3.2. RGB-D Stereo Camera Sensor

The robotic arm is equipped with a *Intel Realsense D435* RGB-D stereo camera sensor, shown in Figure 2.6. The camera is mounted on the *wrist* (the last joint link before the arm's flange) of the robotic arm, allowing the camera to move with the arm's end effector.

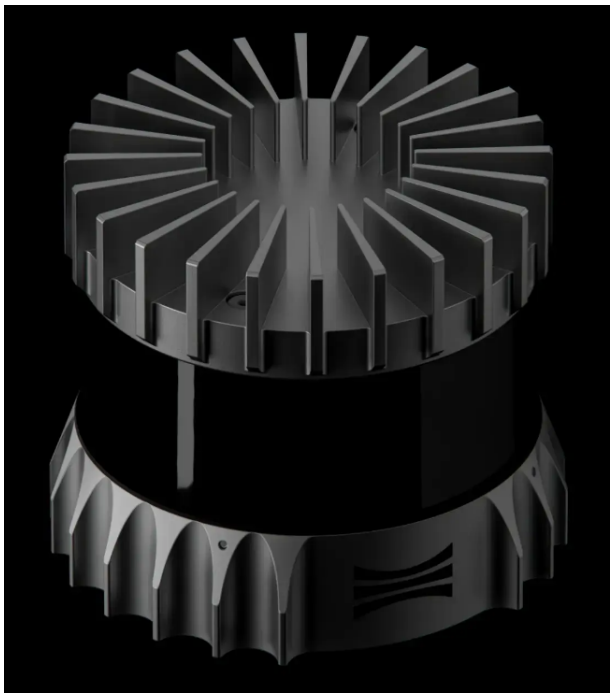


Figure 2.5: Ouster OS1-64 LiDAR sensor

The camera is used for object detection and recognition, and for ArUco markers detection and pose estimation. This camera is chosen for this project because it is relatively cheap and provides both RGB and depth images, which are essential for perception tasks in this robotic application. The camera is also lightweight and compact, making it suitable for mounting on the cobot.

The Intel RealSense D435 is a stereo depth camera that is designed for capturing RGB and depth images. The camera is equipped with a global shutter and a rolling shutter, which allows it to capture images with a resolution of 1920×1080 pixels at 30 frames per second. Throughout the development of the project, the resolution of the Realsense camera was reduced to 640×480 pixels at 30 frames per second. This allows for faster processing of the images since having a higher resolution would not benefit the perception algorithms used in the project. The camera has a field of view of 85.2° horizontal, 58° vertical, and 94° diagonal. The depth camera works within a range of 0.3 meters to 3 meters while maintaining high accuracy and precision.

2.3.3. Intel Realsense Calibration

For robotic applications employing perception sensors, it is essential to calibrate the sensors to provide accurate and consistent data to the software. The calibration of the camera is necessary to correct the distortion of the images and to provide accurate depth

Figure 2.6: Intel RealSense D435 RGB-D stereo camera

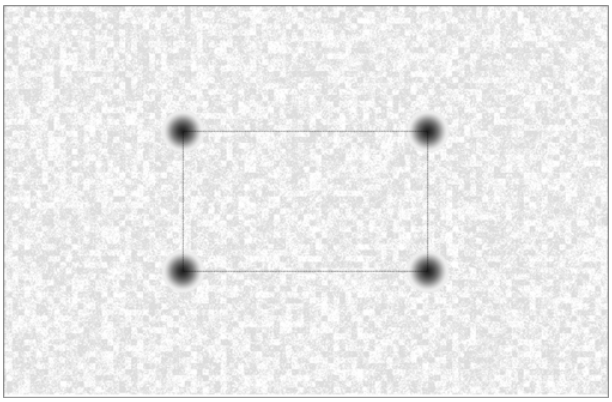


Figure 2.7: Calibration pattern used for the camera's depth image calibration

estimations from the depth sensor. The calibration was fundamental for the correct development of the project, as the depth sensor employed was not already calibrated by the manufacturer. This was a critical issue, as the depth sensor was used for many perception tasks. Performing the calibration process enabled the camera to provide depth estimations consistent with the real-world distances of the objects in the environment. It also enabled the estimation of the distance to the ArUco markers with high precision, by using the correctly calibrated camera's intrinsic parameters. A test was conducted to measure the error of the camera's depth sensor before and after the calibration process, and the results showed a significant improvement in the depth estimations after the calibration process.

The calibration processes for the depth and image sensors was performed using the proprietary software provided by Intel: *Interl Realsense Self-Calibration Tool*, an application for **automatic on-chip calibration**. This software allows to write the parameters for the sensors directly on the camera's EEPROM, ensuring that the calibration parameters are stored on the camera and are not lost when the camera is disconnected from the computer. The calibration process required many trials to find the best possible calibration parameters, but the results were satisfactory. The process is quite straightforward as the software is completely automatic. Two calibration procedures were carried out:

- **Intrinsic parameters calibration**: this calibration process was used to calibrate the camera's intrinsic parameters, such as the focal length, principal point, and distortion coefficients. This calibration was necessary to correct the distortion of the images and to provide accurate depth estimations from the RGB sensor. The calibration process required the camera to capture a series of images of a calibration pattern (checkerboard pattern) from different angles and distances. The calibration software then used these images to estimate the intrinsic parameters of the camera.
- **Depth sensor calibration**: this calibration process required the camera to capture a series of depth images of a calibration sheet, shown in Figure 2.7, which was a flat surface with known distances between the points. The calibration software then used these depth images to estimate the depth sensor's parameters, such as the depth scale factor and the depth offset. These parameters were necessary to correct the depth estimations of the camera's depth sensor.

After the calibration process, the camera's depth sensor was providing accurate and consistent depth estimations, which were consistent with the real-world distances of the objects in the environment.

2.4. Soft Gripper Actuator

The robotic arm is equipped with a Soft Gripper Pneumatic Actuator from *Soft Gripping*, depicted in Figure 2.8 while gripping an apple. This gripper currently used for the project, is composed of three soft fingers, which are actuated by a pneumatic pump. Other versions of the gripper are available and provide two or more fingers. The fingers are made of **silicone rubber**, which is soft and flexible, allowing the gripper to grasp and manipulate objects of different shapes and sizes. The silicone material of the fingers is also non-slip, which ensures a secure grip on the objects. The softness and flexibility of the silicone make it ideal for handling delicate and fragile objects. The other advantage is that silicone is non-toxic and food-safe, making it suitable for handling products in food industries.

The soft gripper is controlled by a pneumatic pump, shown in Figure 2.9, which provides compressed air to the fingers, allowing them to open and close. The pneumatic fingers can exist in three different states: they can be relaxed (i.e. not connected to the pneumatic pump), opened when negative pressure is applied, and closed when positive pressure (1 bar) is applied. The pneumatic pump that controls the soft gripper allows the user to set the pressure value used when closing the fingers, using a valve. The pressure value cannot be changed dynamically via electronic control but is instead fixed and set manually. Figure 2.10 shows the soft gripper mounted on the end effector with the fingers opened

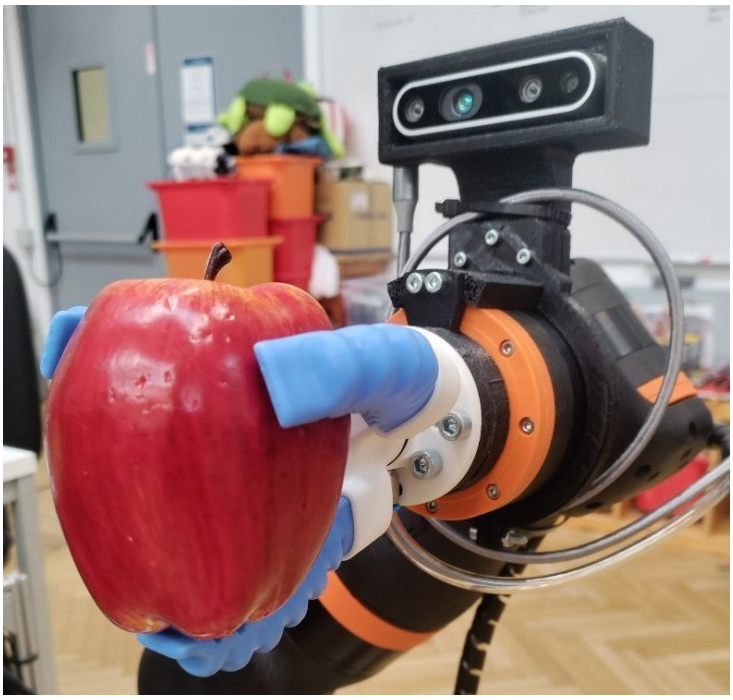


Figure 2.8: Soft Gripper Pneumatic Actuator handling an apple

and closed.

The pneumatic pump provided with the soft gripper has only one output tube used for providing positive pressure to the fingers, which closes them. The negative pressure for opening the fingers is instead provided by an internal valve, that lowers the pressure compared to the atmospheric pressure. Other versions of the pneumatic pump provide also another output tube, which allows the user to control the negative pressure value used when opening the fingers, but this feature requires an external vacuum compressor to work.

The gripper is mounted on the robotic arm's end effector, allowing the arm to grasp and manipulate objects in the environment. It is placed close to the robotic arm's flange to ensure that the mobility and reach of the end effector are not affected by the gripper's size. Furthermore, the soft gripper is very lightweight and compact, making it suitable for mounting on the cobot.

The pump provides four digital pins for controlling its operation, which are used to open and close the fingers. A simple system capable of controlling the pump's operation was created using an **Arduino UNO microcontroller**. The Arduino UNO is connected to the robot's computer via USB, allowing the computer to send commands to the Arduino via the serial port. The Arduino UNO is then connected to the pneumatic pump via a relay module, composed of four different relays, each controlling a different digital pin of the

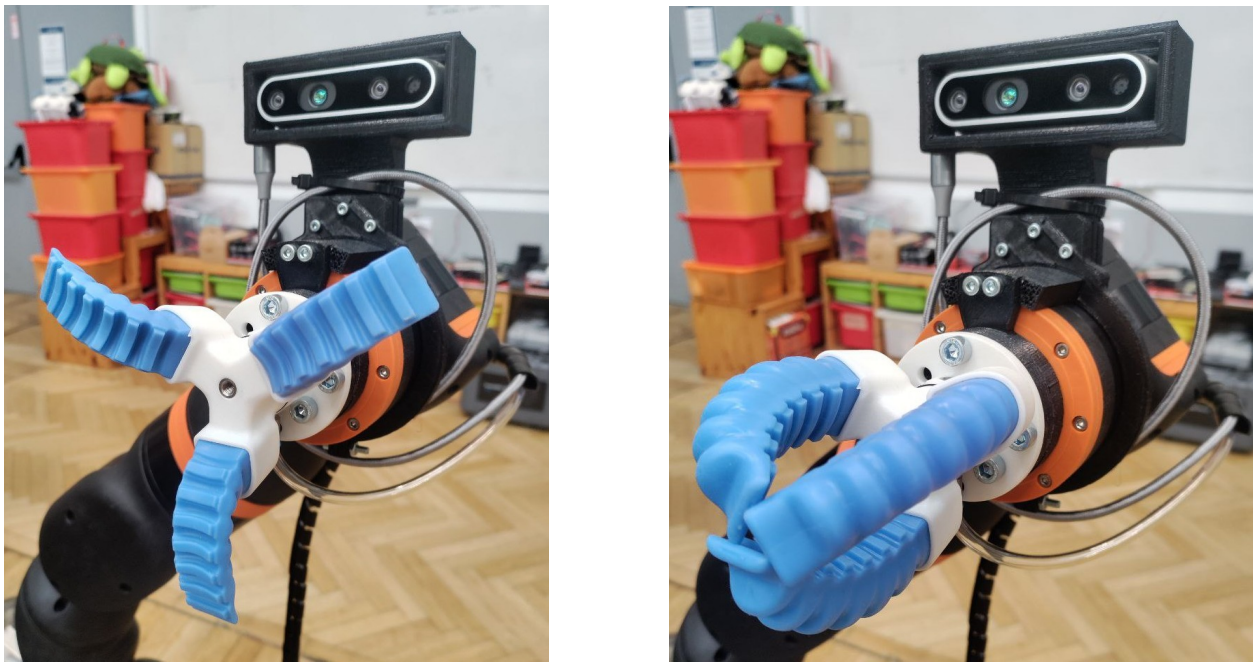
Figure 2.9: Pneumatic Pump control box secured on top of the mobile robot

pump. The relays were necessary to provide an output voltage of 24V, while the Arduino UNO provides only 5V via its digital pinout. The relays are very inexpensive and fast in switching, allowing efficient control of the pump's operation with the microcontroller's digital pin outputs.

The installation of the relays requires simple circuitry for the control system, so it can be easily positioned on the mobile robot platform, without occupying much volume. Figure 2.11 shows the Arduino UNO microcontroller and the relay module used to control the pneumatic pump, installed on the mobile robot platform. Figure 2.12 shows the circuit diagram of the connections between the microcontroller and the pneumatic pump digital pinout. In the circuit diagram, the Arduino UNO digital pins are connected to the relay module and provide the input switch for the relays. The relays are then connected to the pneumatic pump's digital pins, which control the pump's operation. All relays are connected to the same ground and power supply of the Arduino UNO.

2.5. 3D Printed Mounts Design

Mounting the sensors and electronic devices on the mobile robot platform required the design and 3D printing of custom mounts. The mounts were designed using the *Fusion 360* CAD software, which allowed me to create precise and accurate models of the mounts.



(a) Soft Gripper opened

(b) Soft Gripper closed

Figure 2.10: Soft Gripper mounted on the end effector

The mounts were then 3D printed using the laboratory's 3D printer, which is a *Creality CR-10S* 3D printer. The 3D printer employs the Fused Deposition Modeling (FDM) technique, which uses a thermoplastic filament to create the 3D models layer by layer. The material of choice for the 3D prints was *PETG* (Polyethylene Terephthalate Glycol), which is a strong and durable material, suitable for mechanical parts and mounts. The PETG material is also resistant to mechanical stress and heat, making it the ideal material for the mounts for these kinds of applications.

Two versions of the mount were designed and printed for the cobot's flange:

- *Mount V1* 2.13: a mount for the Realsense camera and a digital button on top of a cylinder used for pressing buttons on a control panel. The mount was designed to be robust and easy to switch with other mount extensions. This mount was used for the first demos and tests of the project.
- *Mount V2* 2.14: a mount for the Realsense camera and the soft gripper as an end-effector. This mount was designed to be compact, robust, and more effective for the final versions of the project.

The 3D-printed mounts were designed to be lightweight and compact, to ensure that the robot's mobility and stability were not affected. The mounts were also designed to be robust and durable, to withstand the vibrations and shocks of the mobile robot platform.

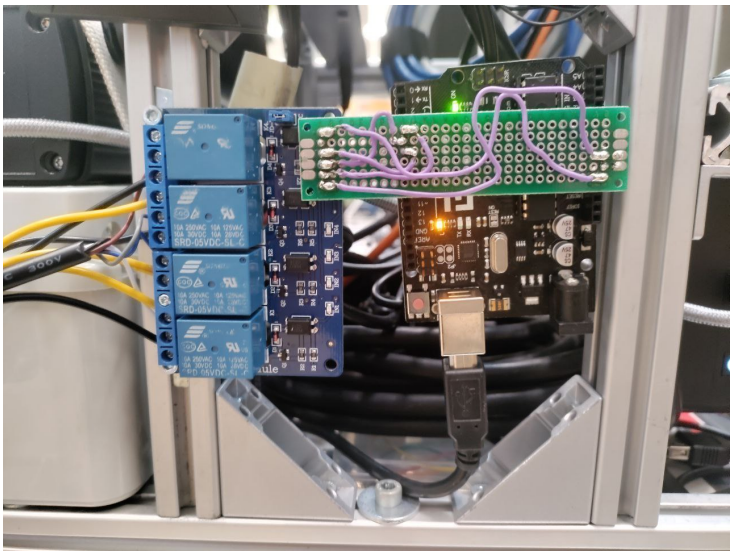


Figure 2.11: Arduino UNO microcontroller and relay module used to control the pneumatic pump.

The mounts are designed to be mounted with several screws and bolts, ensuring that the sensors and electronic devices are securely attached to the robot. These mounts demonstrated to be very effective and reliable, as they withstood the vibrations while maintaining the sensors in their fixed position.

2.5.1. MountV1

The first version of the mount, alias *Mount V1*, shown in Figure 2.13, was designed to be robust and allow to switch easily the mount extensions. MountV1 is lightweight and the cylinder placed on it is long enough to ensure that the cobot's end effector would be able to reach objects not in the immediate vicinity of the robot. The mount is also compact, with the stereo camera mounted in front of the cobot's flange, and the digital button mounted on top of a cylinder used for pressing buttons on a control panel. The 3D print resulted in a very robust and durable structure.

After many tests, the cylinder was then shrunk to a smaller length, to ensure that the cobot's end effector mobility was not affected negatively. This allowed the cobot's end effector to reach objects in the vicinity with fewer limitations and constraints on its orientation, at the expense of a slightly reduced reach. The digital button mounted on the extremity of the end effector was initially used as a digital feedback mechanism for the pressure of the buttons on the control panel. This button was later removed, as it was not necessary for the project's objectives. The mount was then used for the first demos and tests of the project, and it proved to be very effective.

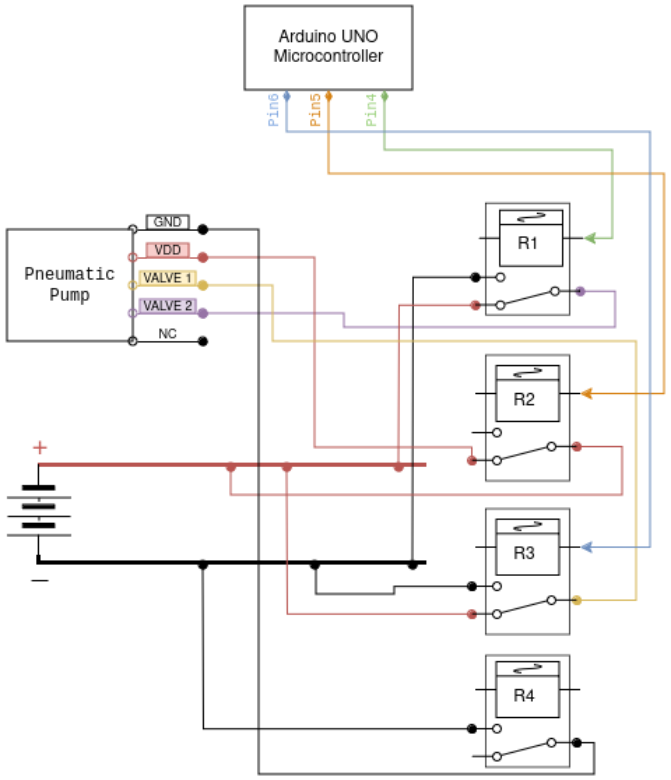
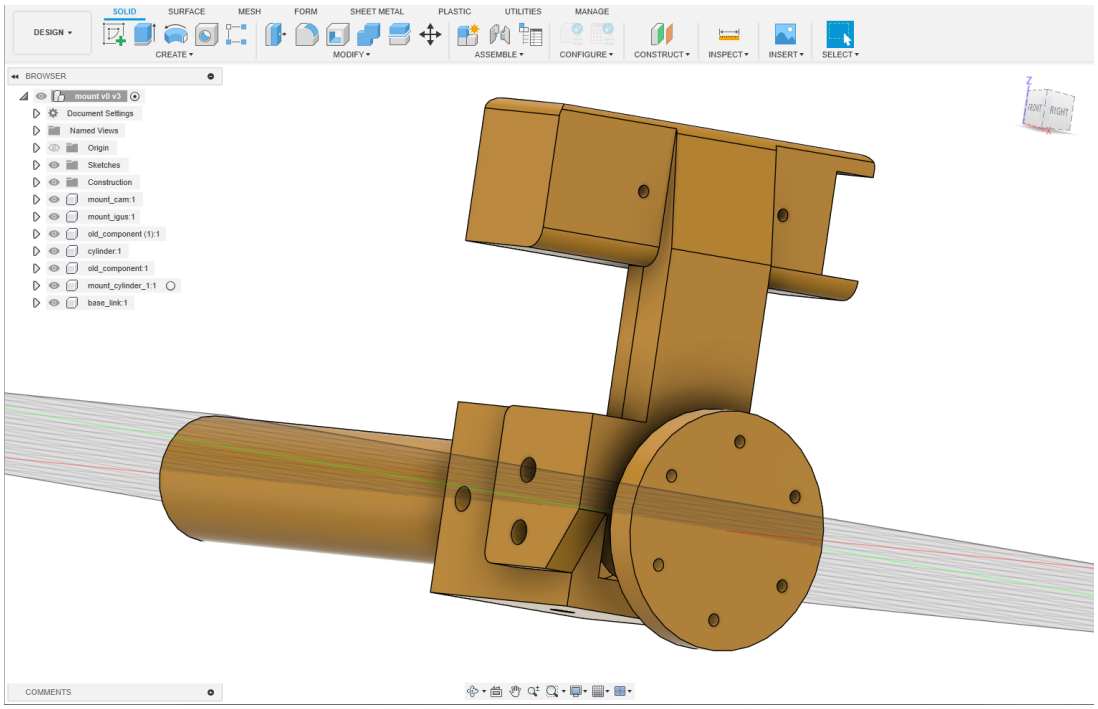


Figure 2.12: Circtuit schematic for the pneumatic pump control system using an Arduino UNO and relay modules.

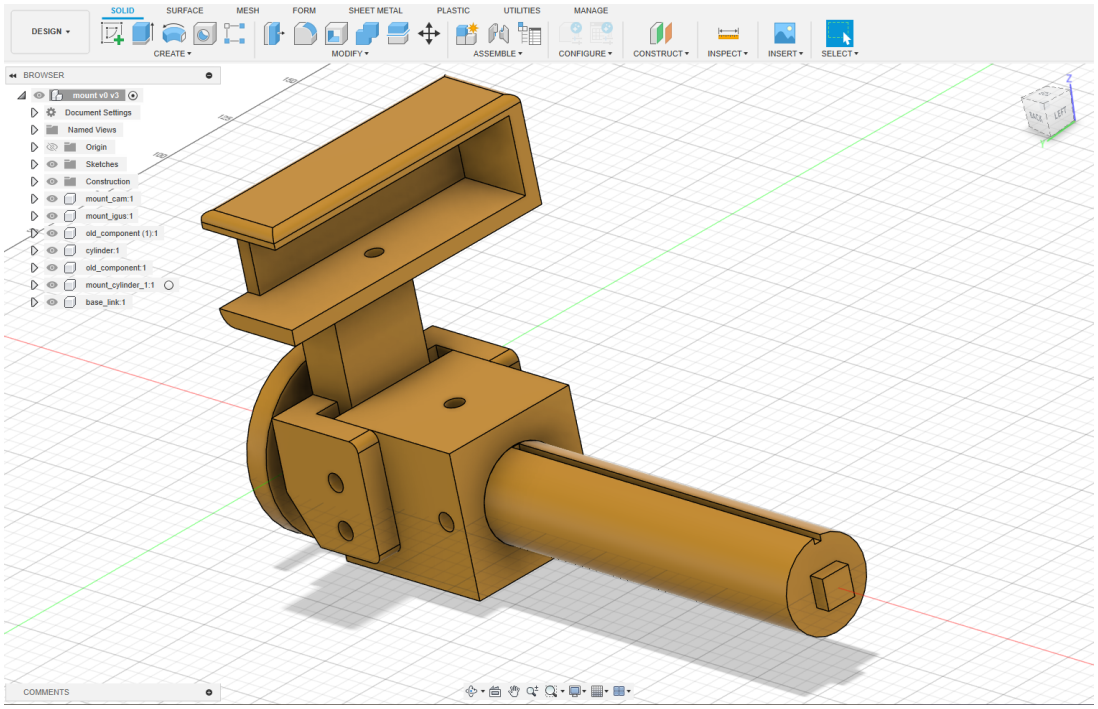
One of the main issues encountered with the mount was the **reduced field of view of the stereo camera**, due to its installation on the cobot's flange. The camera was placed at about 7 cm from the center of the cobot's flange, making it possible for the camera's field of view not to be obstructed by the cylinder, while maintaining the mount compact. Many tests and applications showed that the best configuration would be to mount the camera on top of the wrist, to enlarge the field of view. MountV1 was used for the first applications but then replaced with an improved version for the following activities.

2.5.2. MountV2

The second version of the mount, alias *Mount V2*, was designed to be compact, robust, and more effective for the agricultural applications of the project. The mount, shown in Figure 2.14, comprises three parts: the flange connector, the camera mount, and the soft gripper mount. The system is composed of multiple components, each individually 3d printed, to ensure an efficient printing process that allows printing the components separately if they break or get damaged. This was an important feature, considering that

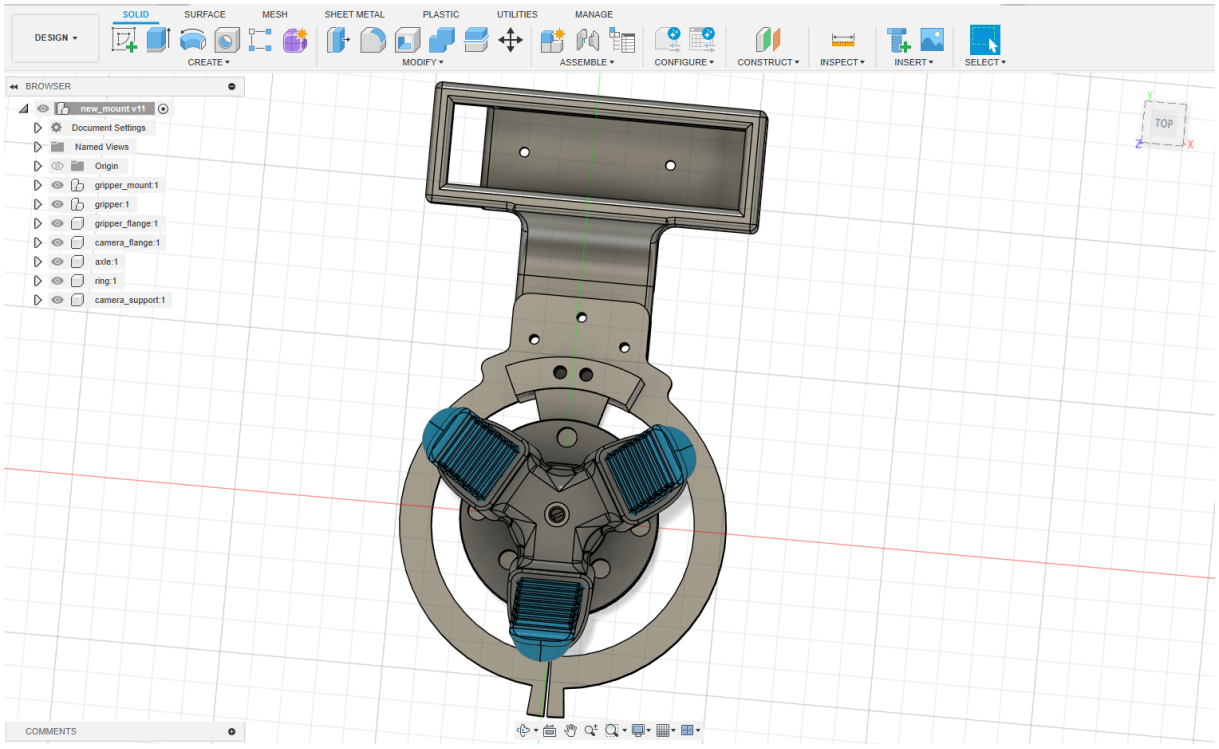


(a) Back side view of the 3D design

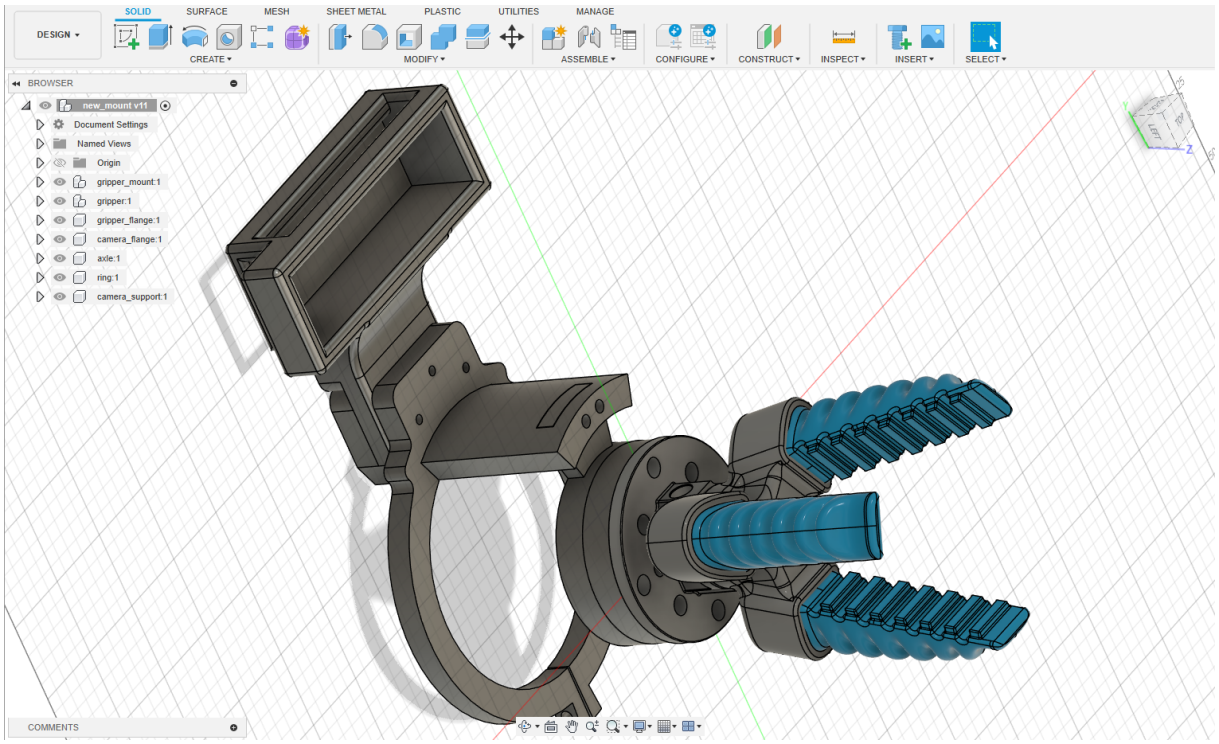


(b) Frontal side view of the 3D design

Figure 2.13: MountV1 Design screenshots from *Autodesk Fusion 360*



(a) Front view of the 3D design



(b) Side view of the 3D design

Figure 2.14: MountV2 Design screenshots from *Autodesk Fusion 360*

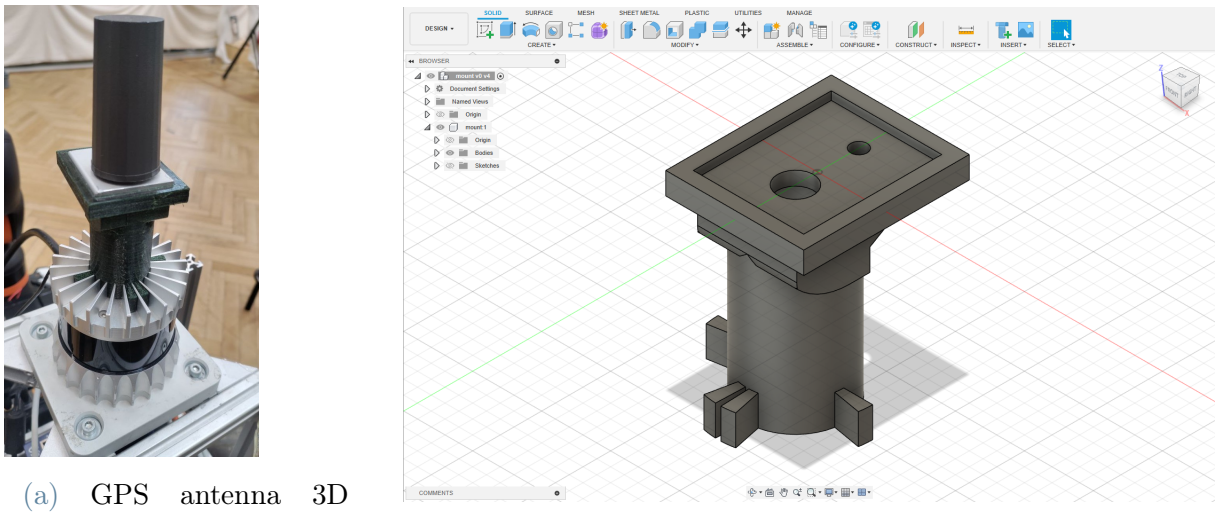
Figure 2.15: 3d-printed mountV2 on the arm's wrist

each component was redesigned multiple times after failed prints or design inefficiencies.

The **soft gripper mount** is used to mount the soft gripper on the cobot's flange connector, allowing the cobot's end effector to grasp and manipulate objects in the environment. The **flange connector** is used to connect the mount to the cobot's flange, ensuring that the mount is securely attached using screws. The flange connector was designed to be lightweight and compact, to ensure that the 3d printing process would be fast and structurally sound. This piece is also robust and durable, to withstand the vibrations and shocks of the cobot's movements. The **camera mount** is used to mount the stereo camera on top of the cobot's wrist, ensuring that the camera has a wider field of view compared to its previous version. The camera mount is designed to be attached to the flange connector on the cobot's wrist with screws and bolts, ensuring that the camera is securely attached to the cobot while preventing vibrations that could offset the sensor position and orientation. The camera mount hosts the camera cable angled connector. This angled connector is magnetic, and it prevents the cable from being pulled out of the camera when the cobot's end effector moves around. This was a critical feature, set to avoid the camera's cable being broken or damaging the internal USB-C port of the stereo camera. The MountV2 installed on the cobot is shown in 2.15.

2.5.3. GPS Antenna Support

Another mount was designed and created: the **mount for the GPS antenna**, which was used for outdoor localization and navigation tasks. The GPS antenna mount was designed to be placed on top of the LiDAR sensor, ensuring that the antenna had a clear view of



(a) GPS antenna 3D printed mount

(b) 3D design

Figure 2.16: GPS antenna mount design and 3D-print on top of the LiDAR

the sky and the satellites. The mount was also designed to be lightweight and quick to install, without affecting the LiDAR sensor's field of view, and without using any screws or bolts. Figure 2.16b shows the mount installation on top of the LiDAR sensor, while Figure 2.16a shows the 3D design of the mount. While the GPS was not used directly in this project, the mount was part of a general redesign of the mobile robot platform, specifically the sensor framework support structure. The redesign was necessary to install the robotic arm and achieve a mobile manipulator configuration while moving the GPS to a location that does not impede the movement of the arm.

2.5.4. 3D printer maintenance and configuration

The AIRLab's 3D printer is a valuable resource for researchers and students, and achieving high-quality prints requires specific conditions. In order to optimize the 3d printing results and meet these quality standards, a series of maintenance and calibration procedures were undertaken to address issues such as **stringing and under-extrusion**, which significantly impacted the quality of 3D printed objects. The printer's bed was calibrated, to ensure that the prints were of high quality and accurate. The nozzle was substituted with a new one, and the hot end was unclogged, to ensure that the filament could extrude correctly and with the best flow. Cleaning the printer's bed and the extruder proved to be useful too, and ensured that the prints were sticking correctly to the bed while avoiding the warping of the prints. After these maintenance operations, the printer was working correctly, and the prints were of higher quality.

Figure 2.17: Lead batteries mounted onboard for the cobot and pneumatic pump

2.6. Batteries and Power Management

The mobile robot's internal battery is capable of powering all the onboard sensors and computational units. To meet the specific voltage and current requirements of these devices, two DC/DC converters are employed, one of which is shown in Figure 2.18:

- one DC/DC converter with an output voltage of 12V at a maximum of 15A for the on-board computer, router, and switch
- one DC/DC converter with an output voltage of 24V at a maximum of 5A for the LiDAR.

The mobile robot platform's internal batteries are not sufficient to power the robotic arm and the pneumatic pump mounted on board. To power these devices, an external power supply is used, which provides 24V at a maximum of 10A. The cobot is powered by two 12V **lead batteries** 2.17 with 9Ah of power capacity, which provides the necessary power to the robotic arm motors and the pneumatic pump. The batteries are mounted and secured on the robot's base, ensuring that the robot is powered and operational during its missions. There are two battery packs available, which can be switched easily when one of them is discharged.

The pneumatic pump requires an external power supply, hence it was necessary to provide power via the **onboard batteries**. The pneumatic pump must be controlled at 24V, as

Figure 2.18: DC/DC converter used to power the robot's sensors and computer

the pump's solenoid valve requires this voltage to operate. The pump is powered by a 24V lead battery, which is the same battery powering the robotic arm. It was necessary to create a system for powering both the robotic arm and the pneumatic pump with the same batteries, due to the limited space available on top of the mobile robot platform. A convenient and cheap solution found is to use Molex cables and connectors that connect both the cobot and the pump to the same battery pack. These Molex cables proved to be a reliable and optimal solution, as they allow to switch easily and quickly between the onboard batteries and the external cobot power supply. The Molex cable management is shown in 2.19.

The cobot's proprietary power supply provides 24V at a maximum of 10A, which is enough to power the robotic arm and the pneumatic pump simultaneously, since the pneumatic pump doesn't require a high current to operate. The output connector of the external power supply is also a Molex cable, so that's why Molex connectors and cables were used to connect the robotic arm and the pneumatic pump to the power systems. Both the batteries and the external power supply make use of the same connectors to power the cobot and the pneumatic pump.

2.7. Mobile Manipulation Setup

Figures 2.21 and 2.20 show the complete mobile manipulation setup, with the robotic arm mounted on top of the Scout robot, and the soft gripper mounted on the robotic arm's end effector. The setup shown in the figures is ready for the "soft grasping" demos, where the robot is tasked with grasping and manipulating objects in simulated agricultural environments.

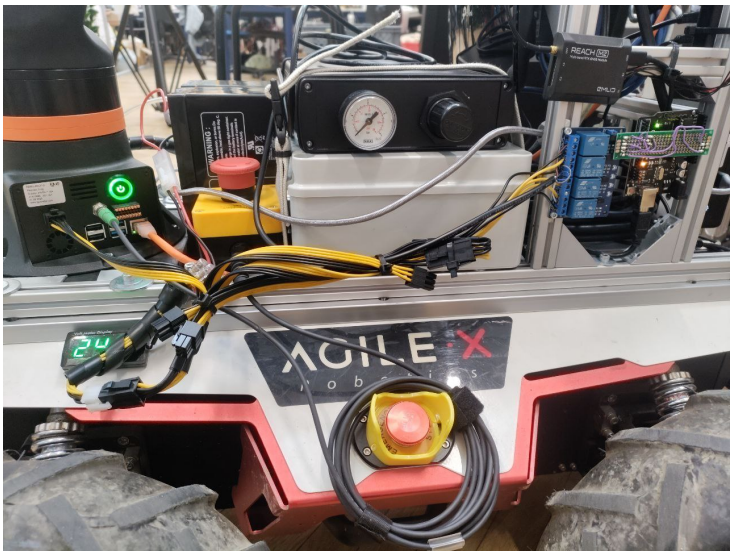


Figure 2.19: Molex connectors and power management for the cobot, pump, and relays

Figure 2.20: Lateral view of the robots

Figure 2.21: Lateral view of the robots

3 | Software Architecture and Simulation Environments

This chapter discusses the software architecture of the mobile manipulation system and the simulation environments used for testing and development. The software architecture is based on the Robot Operating System 2 (ROS2) middleware, which provides a flexible and modular framework for developing robotic applications. The simulation environments are based on Rviz2 and Ignition Gazebo, which provide realistic simulation environments for testing algorithms before deployment.

3.1. ROS2 Control Interface for Igus Rebel Arm

The first step in developing the software architecture for the mobile manipulator is to interface the Igus Rebel Arm with ROS2. The Igus Rebel Arm is a 6-DOF robotic arm that can be controlled by either the CAN binary bus or the Ethernet interface (proprietary CRI protocol). The CAN bus interface is used for low-level access to the arm's joints, while the Ethernet interface is used for high-level control and monitoring of the arm's state. The ROS2 control interface for the CAN bus was already implemented by the control software provider *Commonplace Robotics*. However, this type of connection requires a proprietary cable. Moreover, an Ethernet connection is more flexible, since the manipulator can be connected to the switch and controlled by any computer on the local network.

Developing the **ROS2 control interface** for the Ethernet interface required understanding the **CRI protocol**, which is a protocol based on plain text messages. The CRI protocol is used to send commands to the arm in the form of joint positions (i.e. rotation of the motors in radians) or velocities (i.e. jogs) and to receive feedback from the arm in the form of joint positions (values provided by the motors' encoders). The Igus Rebel cobot provides a *Raspberry Pi* embedded computer that runs the control software for the arm's motors and acts as the CRI server, which listens for commands on the Ethernet interface and sends feedback to the client. The CRI client is the ROS2 control interface, while the CRI server acts as a bridge between the arm's motors embedded closed loop

motor driver controllers and the control software (either the software provided by the manufacturer or the ROS2 control interface).

Controlling the cobot using ROS2 requires a hardware interface, used to command and control the robot by interfacing with the CRI communication protocol. ROS2-Control is the framework provided with ROS2 that makes it possible to develop such hardware and control interfaces. The hardware interface is implemented as a ROS2 lifecycle node that is interfaced directly with the Joint Trajectory Controller, which is a ROS2 controller that can be used to control the arm using joint trajectory messages, containing the desired joint positions or velocities. While the Joint Trajectory Controller is a standard robot controller provided within ROS2-Control that can work with any robot arm and configuration, the hardware interface was developed and implemented specifically for the Igus Rebel Arm and its CRI communication protocol. The implementation of the hardware interface is based on the *System Interface* provided by ROS2-Control, which is a type of interface that supports joints and actuators. The System Interface accesses the hardware via the CRI protocol and is managed by the Controller Manager and the Resource Manager.

The Joint Trajectory Controller is then handled by MoveIt2, which is a ROS2 motion planning framework that can be used to plan and execute trajectories for the arm [22]. **MoveIt2** generates motion plans for the robot using the robot's kinematic model and the obstacles in the environment. The Joint Trajectory Controller receives the computed trajectories from MoveIt2 and sends them to the arm using the hardware interface.

The diagram in Figure 3.1 shows an overview of how the implemented ROS2 hardware interface works, and how it is connected to the ROS2 Controller and MoveIt2 motion planners. The hardware interface relies on the hardware components abstraction layer, a set of classes that provide an interface to the hardware components of the robot, such as the arm's motors and encoders. The state interface is used to read the state of the robot's joints (reads the position values from the motors' encoders), while the command interface is used to send commands to the robot's joints (sends the desired joint positions or velocities to the motors). The diagram also shows the connection between the ROS2 Controller Manager and the MoveIt2 controller managers, which are responsible for managing the execution of the trajectories generated by MoveIt2's planners and sending commands to the appropriate ROS2 controllers based on the desired robot motion. The *FollowKJointTrajectoryController* is the interface for the underlying action server that sends the joint trajectories to the Joint Trajectory Controller.

Integrating the MoveIt2 libraries with the ROS2 framework requires MoveIt2 controller managers, a set of controllers that can be used to control the robot's motion. MoveIt2

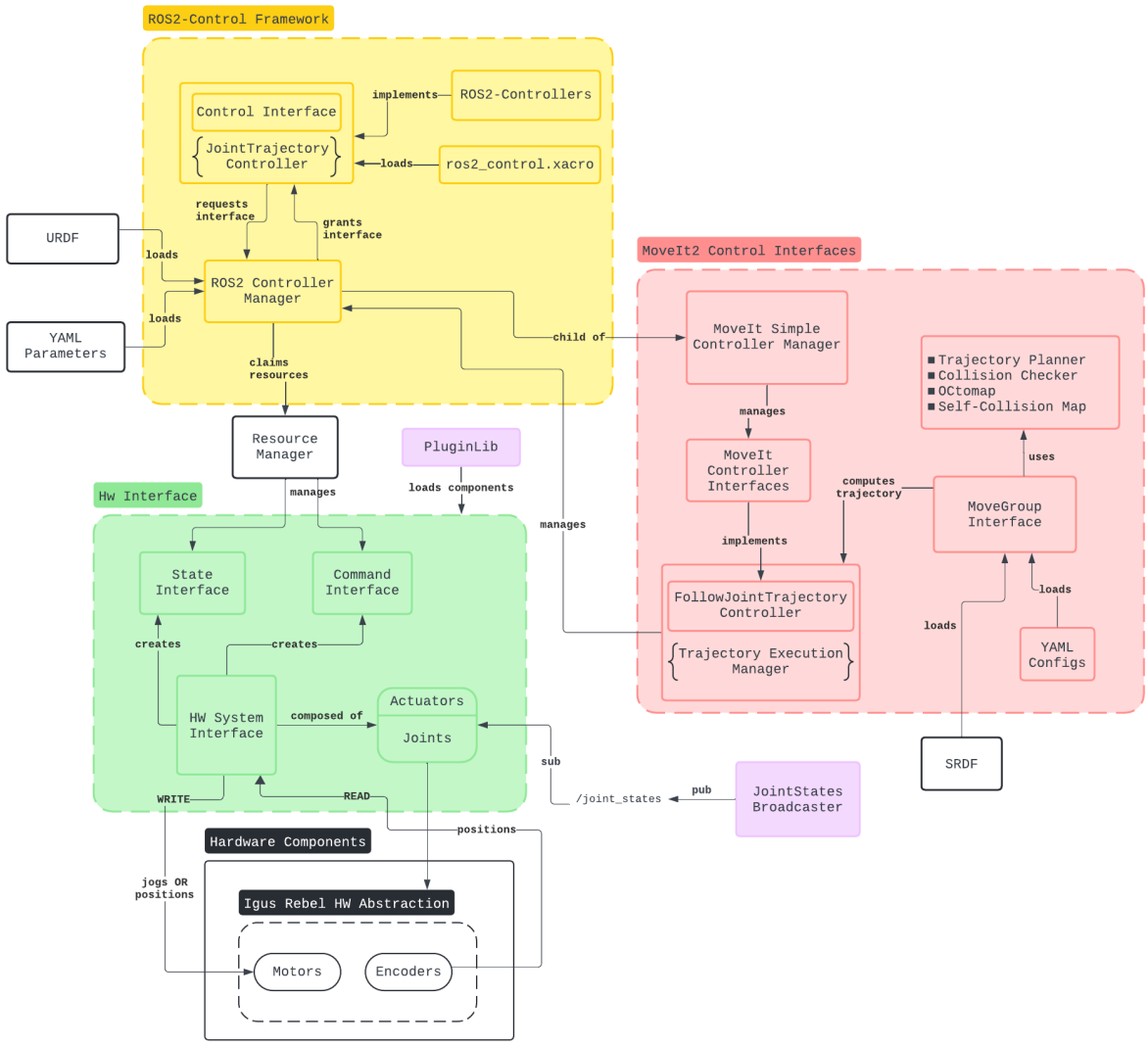


Figure 3.1: ROS2-Control and MoveIt2 Interfaces Architecture

controller managers act as the bridge between high-level motion planning in MoveIt2 and the low-level control of robot hardware. They are responsible for managing the execution of trajectories generated by MoveIt2's planners, sending commands to the appropriate ROS2 controllers based on the desired robot motion. These ROS2 controllers such as the Joint Trajectory Controller, in turn, interact with the hardware interfaces to translate the control commands into actions that the robot can execute. This **layered architecture** allows for modularity and flexibility in the robot control system, enabling the integration of various controllers and hardware interfaces without affecting the high-level motion planning capabilities of MoveIt2. *MoveItSimpleControllerManager* is the controller manager used with MoveIt2 to control the robotic arm using the Joint Trajectory Controller and the *FollowJointTrajectory* action for sending motion execution goals.

3.1.1. Joint Trajectory Controller

The Joint Trajectory Controller is a ROS2 controller that can be used to control the arm using joint trajectory messages. These messages are joint-space trajectories on a group of joints. The controller interpolates in time between the points so that their distance can be arbitrary. Trajectories are specified as a set of waypoints to be reached at specific time instants, which the controller attempts to execute as well as the underlying hardware allows. Waypoints consist of either positions or velocities, accelerations are optional.

The Joint Trajectory Controller can be operated either in position, velocity, or acceleration mode, depending on the type of trajectory message that the hardware interface can handle. The position mode is used to send joint positions to the arm, while the velocity mode is used to send joint velocities in the form of jog values (i.e. velocities in percentage of the maximum velocity). The Joint Trajectory Controller can be operated either in closed-loop or open-loop mode, depending on the presence of the feedback that is used to control the arm. In closed-loop mode, the controller uses the joint positions received from the arm as feedback to adjust the trajectory to the desired position, by controlling the arm via velocity commands. In open-loop mode, the controller sends the trajectory to the arm without any feedback, which can be useful for testing the arm's performance without any feedback control.

The closed-loop velocity control requires a **PID controller** to adjust the velocity commands based on the difference between the desired and actual joint positions. The PID controller requires tuning the gains to achieve a stable and smooth trajectory execution. Tuning the gains is a challenging task that requires either the system's model or a realistic simulated robot model that mimics the real robot's behavior and dynamics. Moreover, this is a time-consuming task that requires extensive testing with many trials and errors to find the best gains for the PID controller. Insufficiently tuned gains can lead to strong oscillations in the joint positions and the arm not reaching the desired pose stably. In the worst case, the arm can collide with obstacles in the environment or even damage itself. In the case of the Igus Rebel Arm, it was not possible to carry out the PID gains tuning due to the lack of a realistic simulated robot model and the impossibility of testing the dynamics and impulse responses of the robotic arm in a safe environment. Therefore, the open loop controller was used throughout the project.

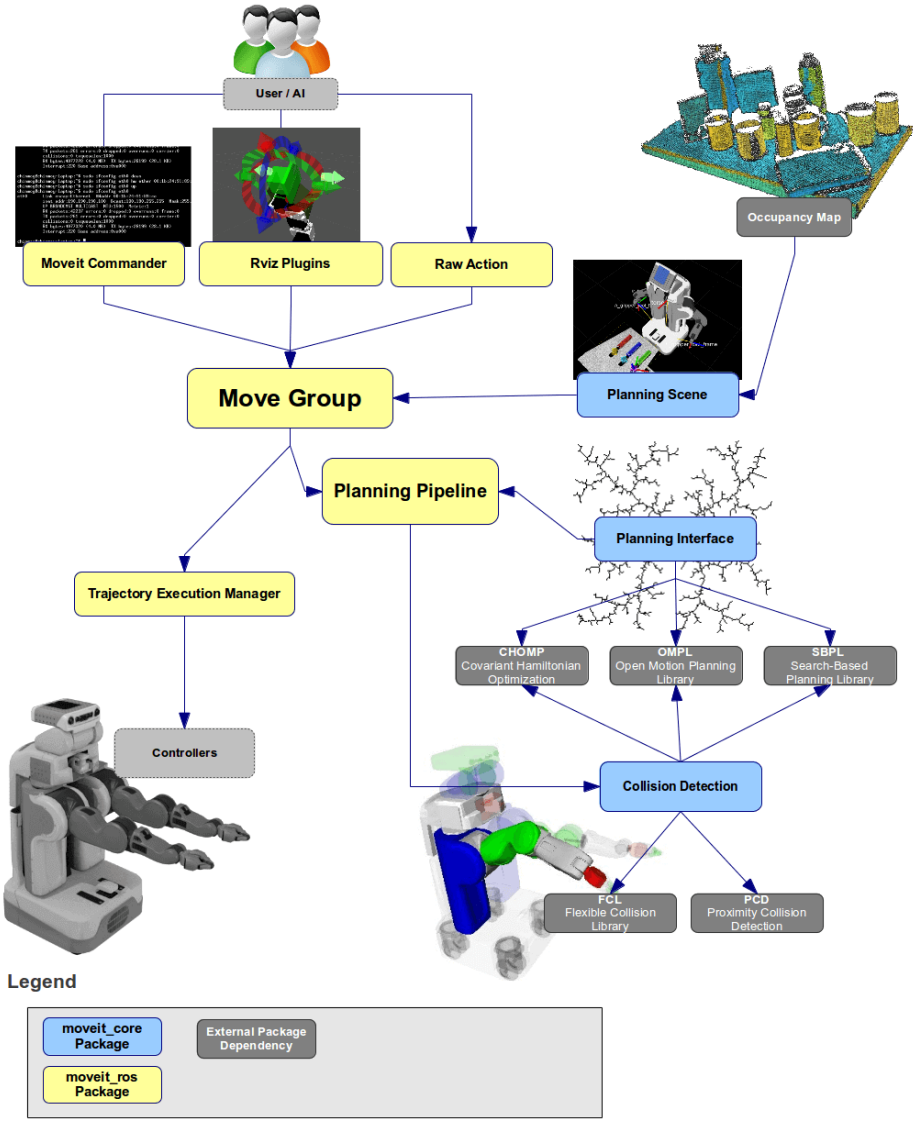


Figure 3.2: MoveIt2 General Architecture

3.2. MoveIt2 and RViz2 Simulation Environment

MoveIt2 is a framework based on ROS2 that provides a set of tools for motion planning, kinematics, control, perception, and manipulation. It is a powerful tool for controlling robotic arms and mobile bases, and it can be used to plan and execute arm trajectories. MoveIt2 is based on the ROS2 middleware and provides a flexible and modular framework for developing robotic applications.

Figure 3.2 shows the general **architecture of MoveIt2**. The MoveIt2 framework consists of several components, including the Planning Scene, the Planning Pipeline libraries, the Kinematics Solver, the Collision Checker, the Trajectory Execution Manager, and the

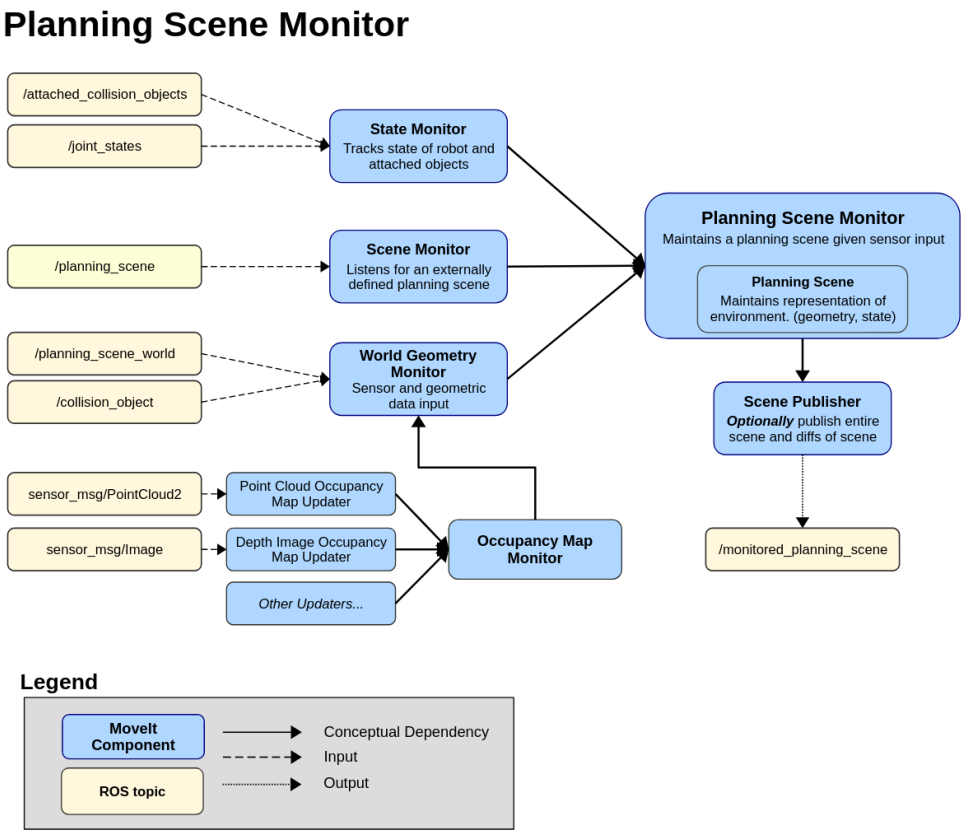


Figure 3.3: Planning Scene and Occupancy Mapping Architecture

Occupancy Mapping tools. The Planning Scene is a representation of the robot's environment, including the robot's state, the obstacles in the environment, and the robot's kinematic model. The diagram in Figure 3.3 shows how the Planning Scene interacts with the MoveGroup interface, providing the robot's state and kinematic model, and the Occupancy Map, providing the obstacles in the environment. The Planning Pipeline libraries are used to generate motion plans for the robot, using the robot's kinematic model and the obstacles in the environment. The Kinematics Solver is used to compute the robot's joint positions for a given end-effector pose, using the inverse kinematics computations based on the robot's kinematic model. The Collision Checker is used to check for collisions between the robot and the obstacles in the environment, using the robot's kinematic model and the obstacles' geometries. The Trajectory Execution Manager is used to execute the motion plans generated by the Planning Pipeline libraries, using the robot's joint positions and velocities. The Occupancy Mapping tools are used to generate volumetric 3D occupancy maps of the surrounding environment, using depth perception sensors to construct a map of the obstacles in the environment.

MoveIt2 provides the *MoveGroupInterface* class, a high-level interface to the MoveGroup

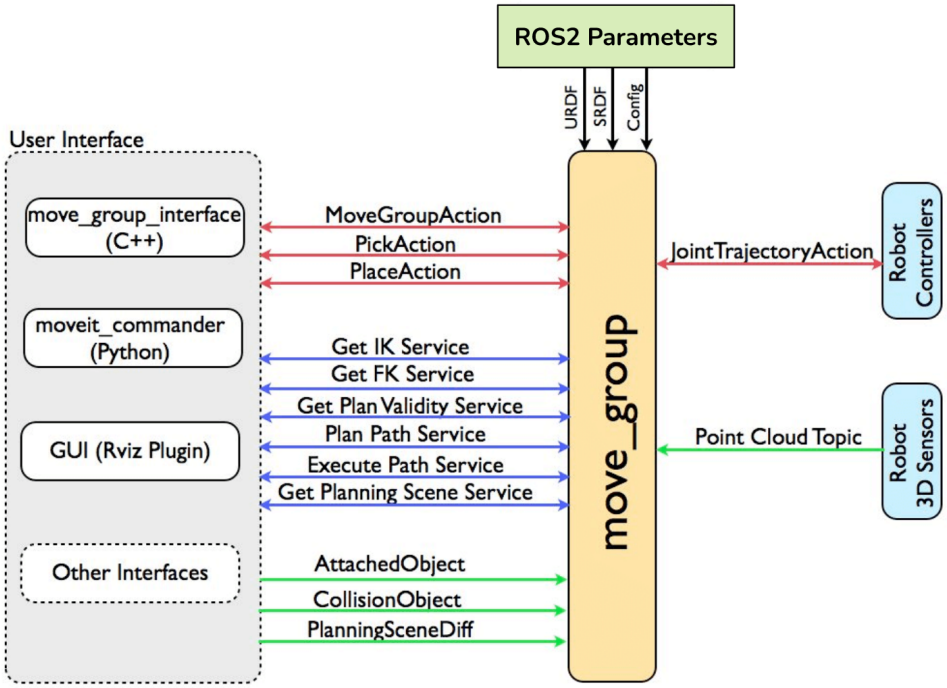


Figure 3.4: MoveGroup Interface

node. The node provides a set of methods for controlling the robot's motion planning and execution. Figure 3.4 shows the architecture and the modules with which it interacts. The *MoveGroupInterface* provides also methods for interacting with the robot's planning scene, including adding and removing obstacles, setting the robot's state, and setting the robot's end-effector pose. The *MoveGroupInterface* can be used to plan and execute trajectories for the robot. It allows to define and compute joint-space goals, Cartesian-space goals, and pose goals for the robot's end-effector, which is used to compute the joint space trajectories.

MoveIt2 can be used along with RViz2, a 3D visualization tool for ROS2 to visualize the robot's motion in both simulated and real environments. RViz2 provides a set of tools for visualizing the robot's kinematic model, the robot's state, the obstacles in the environment, and the robot's motion plans. Figure 3.5 shows a screenshot of the RViz2 interface with the control panels dedicated to the MoveIt2 planning and execution tools. The Figure shows the Igus Rebel cobot mounted on the Scout mobile robot, with blue collision boxes to prevent the arm from colliding with the sensors mounted on top. With this interface, it is possible to send joint space goals to the robot and control it via the underlying hardware interface. There is also an interactive marker that allows the user to set the robot's end-effector pose by dragging it around in the RViz2 interface.

To ease the development of motion planning and execution for the Igus Rebel arm, a

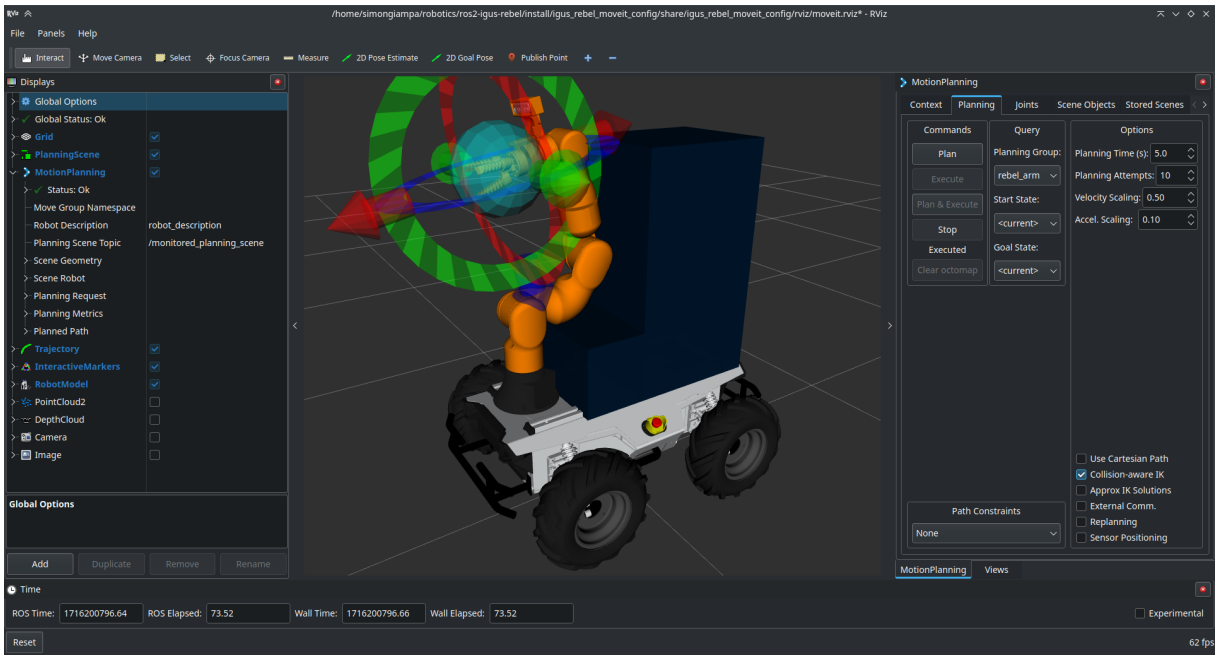


Figure 3.5: RViz2 Interface with MoveIt2 Control Panels and the mobile manipulation robot.

custom ROS2 library was developed. The library incorporates the *MoveGroupInterface*, providing high-level functions in C++ that allow to plan and execute different types of motion, as well as computing the end-effector position and orientation quaternion from sensor input data. This library is based on the MoveGroup lower-level functionalities and makes use of different planners to generate a valid motion trajectory given the planning scene, and current and target joint positions. The planning and execution functions choose the most suitable planner based on the type of motion to be executed. They also switch planners in the cases of failed motion planning generation.

The planner *Pilz Industrial Motion Planner* is used for linear cartesian trajectories since it is more likely to generate feasible plans for this task, compared to the other planners. The *OMPL* (Open Motion Planning Library) and *STOMP* (Stochastic Trajectory Optimization) motion planners are instead used for generating trajectories with joint-space or cartesian-space targets. The kinematic solver used throughout the project is *KDL* (Kinematics and Dynamics Library), which is a C++ library that provides a set of tools for computing the forward and inverse kinematics of the robot's kinematic model. For collision detection, the *FCL* (Flexible Collision Library) is used, which is a C++ library that provides a set of tools for detecting collisions between the robot and the obstacles in the environment. Its advantage is the integration with Octomap for 3D collision checking, which is used in the MoveIt2 framework.

One important issue encountered when executing a planned trajectory was the **imprecision of the robotic arm's motors' encoders**, which caused the end effector to not reach the target pose with the desired precision. This problem was partially overcome by adding a function that artificially compensates for the error in the end effector's position, by adding a small offset to the target pose. The formula used for the compensation is based on empirical measurements of the error. The measurements were taken by commanding the end effector to reach a target pose and then measuring the distance on the z-axis (vertical axis) between the end effector's final position and the target pose. The measurements were taken for poses at different heights and distances from the robot's base. The function is based on the equation $z' = z + \frac{1-z}{25}$, where z is the target position on the z-axis (height) and z' is the corrected target height. The values used in the formula are expressed in meters. This function does not provide an accurate compensation for the error, but it is effective in increasing the precision of the end effector's position.

3.3. Robotic Arm Visual Servoing

The MoveIt2 library provides also a framework for visual servo-ing, a technique used to control the robot's end-effector pose using visual feedback from a camera. The visual servo-ing framework in MoveIt2 is based on the *MoveItServo* package, which provides a set of tools for controlling the robot's end-effector pose using servo-ing algorithms. The servo-ing algorithms are used to compute the robot's joint positions for a given end-effector pose. A test was set up for the integration of the servo-ing techniques with visual input from the camera. The test was developed to perform visual servo-ing with the Igus Rebel arm, where the visual input is provided by the estimation of an ArUco marker's pose in the camera's field of view, and the servo-ing algorithm is used to track the marker's pose and move the arm's end-effector as close as possible orthogonally to the marker's pose.

The servo-ing algorithms provided by MoveIt2 enable **real-time control and teleoperation** of the robot arm's end-effector pose. The teleoperation allows the user to control the robot's end-effector pose using a joystick controller. The real-time control consists of controlling the robot's end-effector pose using camera visual feedback in real-time without pre-computing the joint trajectories. Realtime control is useful for controlling the robot's end-effector pose in dynamic environments, where the obstacles are moving and the robot needs to adapt to the changes in the environment.

Visual servo-ing functionality was implemented and achieved partial success, operating reliably with input Cartesian poses near the arm's initial configuration. The servo-ing algorithm's reliance on accurate initial end-effector pose estimates presented challenges in

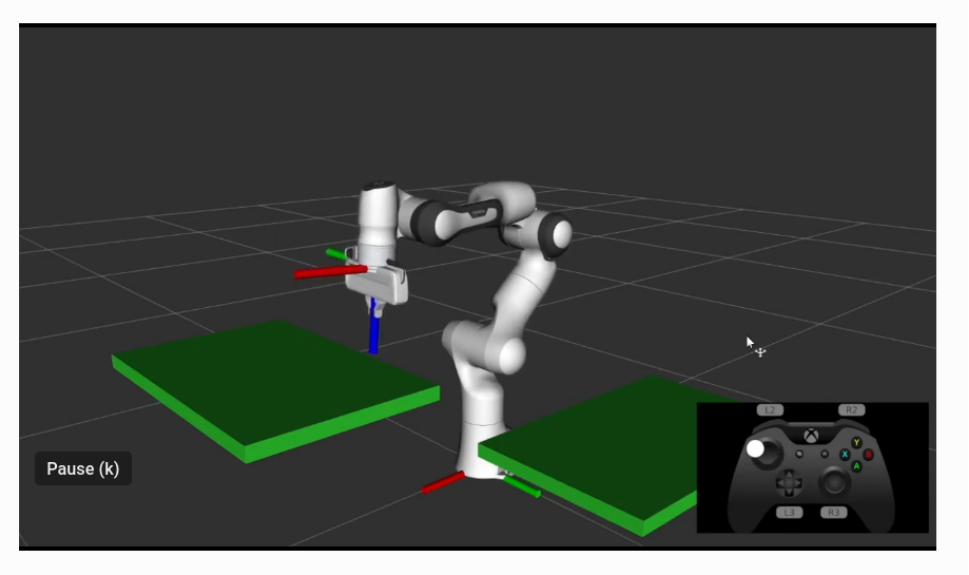


Figure 3.6: Teleoperation with a joystick controller in RViz2 with the Franka Emika Panda

achieving convergence to desired poses under varying conditions. The servo-ing algorithm was tested with dynamically generated input cartesian poses, and the algorithm performance evaluation is based on the arm's ability to reach the desired poses consistently. *PlotJuggler* is the software used to plot the trajectory of each joint in time, which is necessary for monitoring and measuring the degree of the oscillations and overshooting of the arm's joints during the servo-ing process until convergence to the input pose is achieved.

Open-loop control tests highlighted the algorithm's sensitivity to joint position errors, hindering its ability to reach desired poses consistently. While switching to closed-loop mode offered the potential for error compensation, it requires a PID controller to operate. Since the PID controller's performance was hampered by the need for further gain tuning to mitigate strong oscillations, the tests were conducted only in the simulated environment, where a PID controller was easier to tune. The effort required to identify the optimal gains for the PID controller was beyond the gain obtained using this approach. Despite these challenges, the implemented visual servo-ing package remains available within the repository as a foundation for future development. However, due to the limitations encountered in achieving stable and reliable performance, it was not incorporated into the final mobile manipulation system implementation.

3.4. Collision Avoidance with Octomap

The MoveIt2 library provides a framework for collision avoidance using **Octomap**, a library for generating volumetric 3D occupancy maps of the surrounding environment [6].

Octomap is a 3D probabilistic occupancy grid representation of the robot's environment. It divides the space into voxels (3D cubes) and assigns probabilities to each voxel, indicating whether it's occupied or free. MoveIt2 integrates data from depth sensors, such as a depth camera or a LiDAR, to update the Octomap in real time. Figure 3.7 shows the integration of Octomap inside MoveIt2, where the voxels defining the occupancy mapping are inside the Planning Scene. The motion planning algorithms can use the voxels to plan a trajectory that avoids them.

MoveIt2's collision checker uses Octomap to efficiently check for collisions between the robot's planned trajectory and the environment. By checking if the voxels along the path are occupied, it can determine if the robot will collide with any obstacles. The probabilistic nature of Octomap helps deal with sensor noise and uncertainty. If a voxel has a high probability of being occupied, it's treated as an obstacle.

MoveIt2's motion planners, such as *OMPL* (Open Motion Planning Library), use Octomap to generate collision-free paths. The planners avoid occupied voxels to ensure the robot's motion doesn't lead to collisions. If the environment changes during execution (e.g., an obstacle moves), MoveIt2 can use the updated Octomap to quickly replan a new collision-free path, allowing for dynamic obstacle avoidance. This allows for a dynamic representation of the environment, even if obstacles move.

However, the reality is far from the ideal scenario. The Octomap library is not yet fully integrated with MoveIt2, resulting in a **poorly optimized** Octomap probabilistic representation of the environment. In fact, the library tracks the free space as well as the occupied space, meaning that the entire workspace volume of Octomap is indexed within the data structure for volumetric representation, resulting in a memory and computationally intensive update process. This results in low-frequency updates of the voxels. Octomap library for ROS2 is a yet incomplete porting of its ROS1 counterpart.

The ability to update the Octomap in realtime when part of the environment changes is still a missing feature. In the current version of the software, Octomap will only update a few voxels around the part of the environment that changed, without fully removing the voxels corresponding to the obstacle that moved and that is not present in the scene anymore. This leads to **false positives in the collision checking**, which prevents the robot from executing the planned trajectory. This feature is critical for dynamic obstacle avoidance because the robot needs to avoid the voxels corresponding to the obstacles and touch the voxels corresponding to the object that the end effector needs to grasp. This means that not only does the robot need to avoid the obstacles but also understand which voxels the robot arm can collide with. The Octomap Library is still under development,

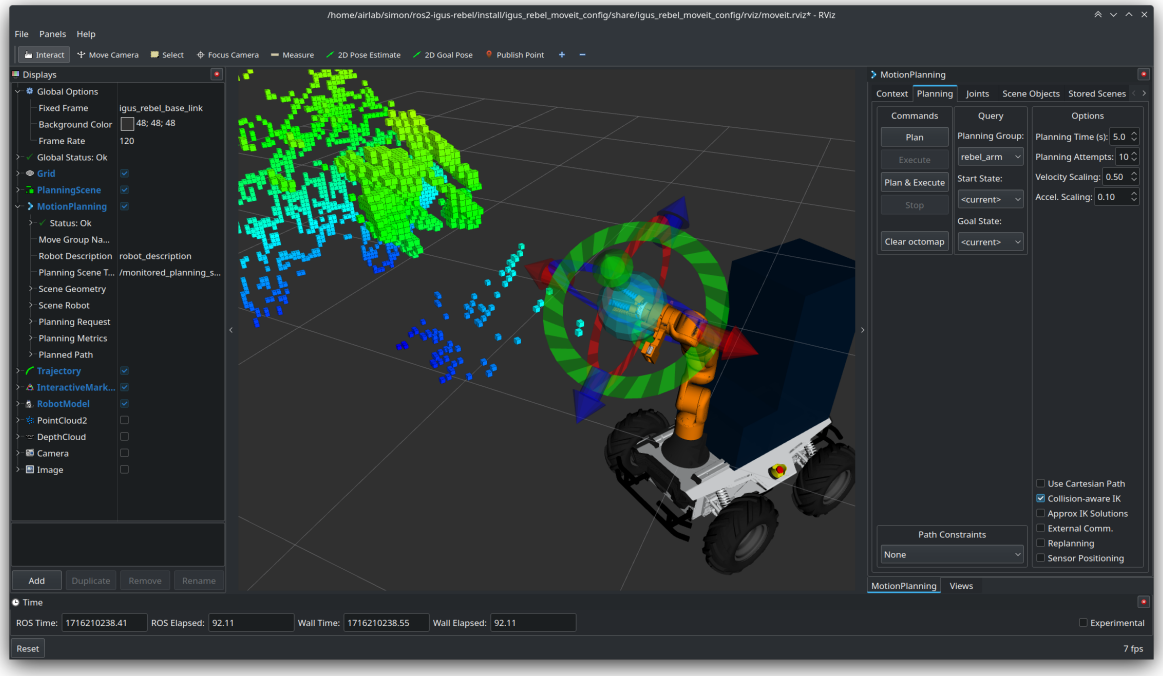


Figure 3.7: Mobile robot using the depth camera to construct the Octomap with the obstacles nearby

and it is expected to be correctly integrated with MoveIt2 in the future.

3.5. Soft Gripper Pneumatic Pump Actuation

The soft gripper is actuated using a ROS2-control interface that acts as a hardware interface to the Arduino UNO microcontroller that controls the pneumatic pump. The hardware interface works by providing a ROS2 service server that listens for commands to open or close the gripper and sends the corresponding commands to the Arduino UNO microcontroller via serial communication. Serial communication uses the UART protocol to send and receive plain text messages. The serial data transfer is done using the *thermios* library, which is a POSIX-compliant library for serial communication in Linux, supporting the C language.

The **Arduino UNO microcontroller** is programmed to control the pneumatic pump by changing the state of the relays connected to its digital pins. The Arduino UNO listens in the serial port for string commands that it interprets as the pins to be set high or low, to open or close the gripper. The pneumatic pump is connected to the Arduino UNO via a relay module with 4 relays, one for each digital pin of the pump. Two of which are the VCC and GND pins, and the other two are the GRIP and RELEASE pins. The GRIP

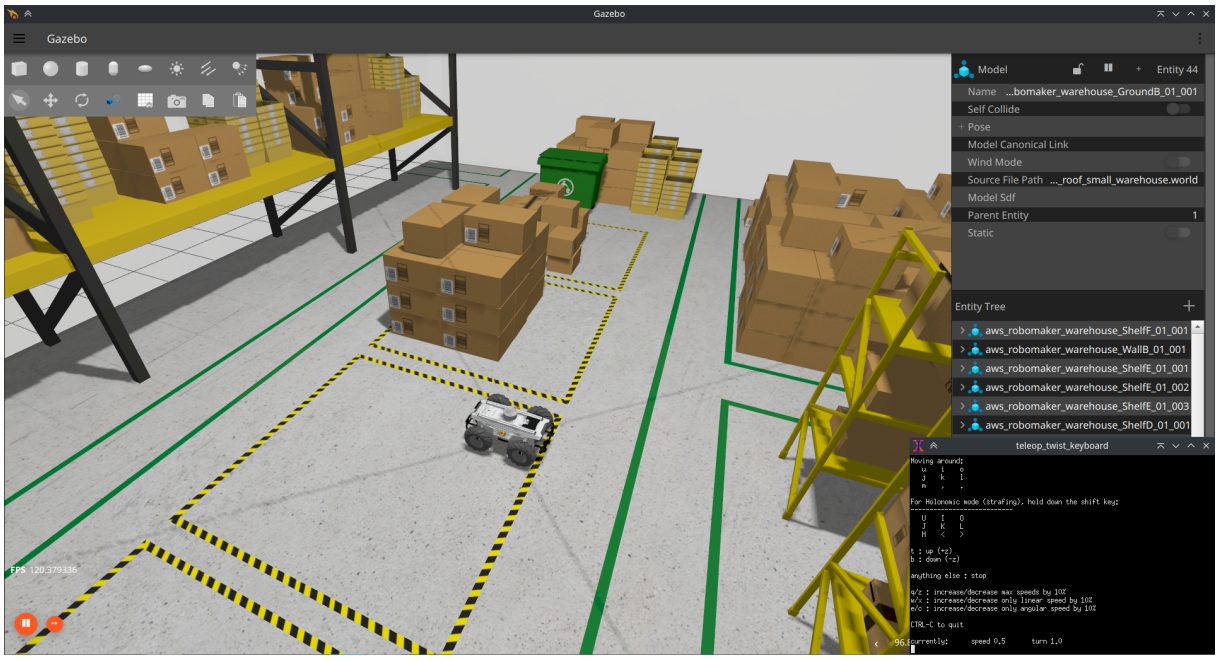


Figure 3.8: Ignition Gazebo Simulation Environment

pin is used to close the gripper, while the RELEASE pin is used to open the gripper.

3.6. Ignition Gazebo Simulation Environment

Ignition Gazebo is a useful open-source simulation environment for robotics and autonomous systems. It provides a realistic and customizable 3D environment for testing and developing robotic algorithms and applications. **Ignition Gazebo** is the simulation environment of choice for the project, as it proved to be very useful in testing the mobile robot base in simulated environments before deploying the algorithms on the real robot. Simulating the robot in a virtual environment allows for testing the algorithms in a controlled and repeatable environment, without the risk of damaging the real robot or the laboratory environment. The simulations were essential for the development of the software and the navigation algorithms, as they allowed for testing the robot's behavior in different scenarios and environments. It allowed me to tune Nav2's parameters thoroughly and ensure the robot would avoid obstacles and navigate safely.

The simulated environment in Ignition Gazebo used for the project is a **warehouse** with various obstacles such as walls, shelves, and boxes, which the robot had to navigate around to reach its goal. Figure 3.8 shows a screenshot of the simulated warehouse environment in Ignition Gazebo. The warehouse environment was designed to be challenging for the robot, with narrow passages and tight spaces, to test the robot's ability to navigate in complex

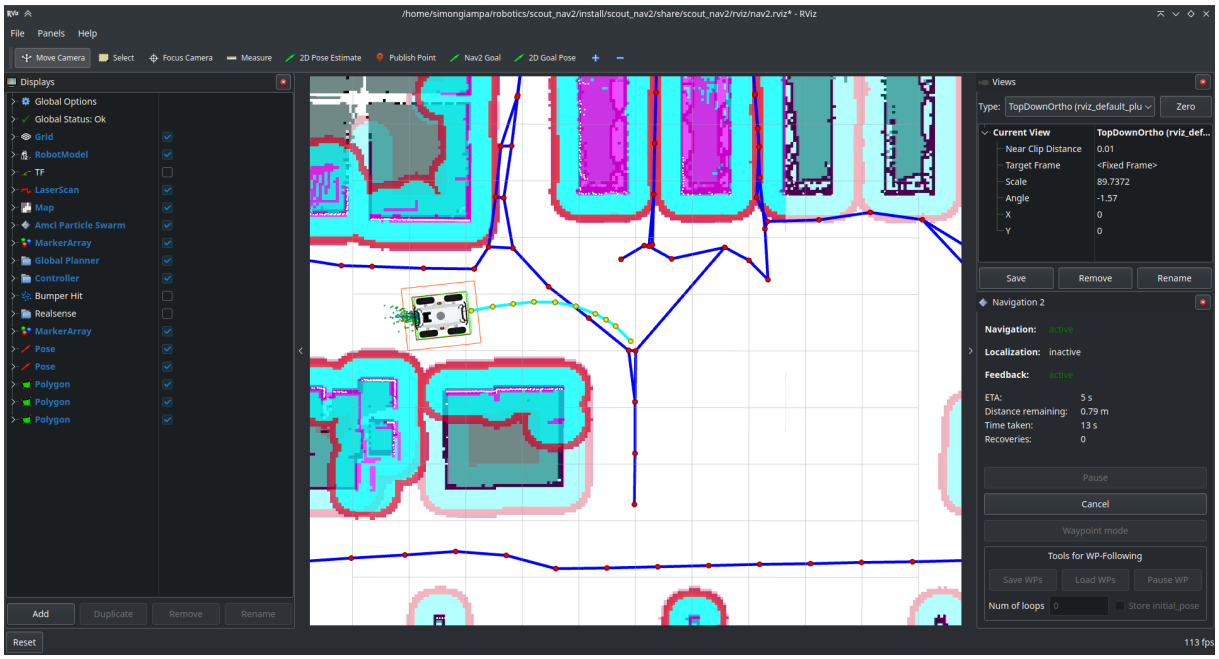


Figure 3.9: Mobile robot navigating in the simulated warehouse environment

environments. Ignition Gazebo also provides the possibility to add dynamic obstacles, i.e. objects not present in the map already loaded in the simulation environment. This feature was useful for testing the robot's dynamic obstacle avoidance algorithm, which allows the robot to navigate safely in environments with unknown obstacles.

The simulation also includes the sensors mounted on the robot, such as a 2D LiDAR and a 3D LiDAR sensor, which were used for perception and obstacle avoidance. The localization algorithms, such as AMCL and SLAM Toolbox, were tested using the simulated odometry data created by the robot's wheels' motion in the simulated environment. Figure 3.9 shows the simulated mobile robot navigating in the warehouse environment while avoiding the boxes and shelves in its path.

3.7. Autonomous Navigation with NAV2

Nav2 is a powerful open-source software framework used for autonomous navigation within the ROS2 ecosystem [17]. It provides a comprehensive set of tools and algorithms for enabling robots to navigate complex environments intelligently. With Nav2, robots can perceive their surroundings, localize themselves, plan optimal paths, and execute those paths while avoiding obstacles. Its modular architecture allows for customization and integration with various sensors, such as LiDAR and cameras, making it adaptable to different robot platforms and use cases. Nav2's flexibility and robust features make it

a popular choice for both research and industrial applications in fields like robotics and autonomous vehicles [16].

Nav2's **modular architecture** enables the seamless integration of various plugins that contribute to its robust autonomous navigation capabilities. Costmap plugins, such as static and obstacle layers, create a real-time representation of the robot's environment, highlighting obstacles and free space. Collision monitors continuously assess the robot's planned path against this costmap ensuring safe navigation. Localizers, like AMCL, estimate the robot's position within the environment, while mappers like SLAM create and update maps of the surroundings. Planners, such as global planners (e.g., Hybrid A*) and local planners (e.g., DWB), work in tandem to generate collision-free paths for the robot to follow, enabling efficient and fast navigation. Additionally, plugins like controllers and recovery behaviors further enhance Nav2's ability to handle unexpected scenarios and ensure the robot reaches its goal.

Given these premises and features, Nav2 was selected as the primary navigation system for the mobile manipulator robot. The Nav2 framework is used to plan and execute trajectories for the mobile base, using the robot's LiDAR sensor to perceive the environment and avoid obstacles. The framework operates within a ROS2 composable node architecture, which allows for the integration of various plugins and algorithms to achieve efficient intra-process communication and data sharing between the various components and threads of the multiple plugins. The parameters for Nav2 were configured specifically to work with the AgileX Scout robot, using the SLAM Toolbox algorithm for mapping and localization. Figure 3.10 and Figure 3.11 show the autonomous navigation of the robot in the hallways and AIRlab in building 7 of Politecnico di Milano, respectively.

3.7.1. Nav2 Parameters Tuning

Finding the right parameters for the Nav2 framework required tuning hundreds of parameters to achieve optimal performance. The parameters to be set are related to all the plugins that are included within Nav2. Many tests in different and complex environments were performed to find the best parameters suitable for the AgileX Scout robot. The parameters were tuned in both simulated and real environments, to ensure that the robot can navigate safely and efficiently in different scenarios.

Localization Algorithms: The algorithms under test were Adaptive Monte Carlo Localization (AMCL) and SLAM Toolbox. The results of the tests showed that AMCL has a worse performance for in-place rotations. SLAM Toolbox algorithm has better performance and improved reliability in terms of localization accuracy, even in changing

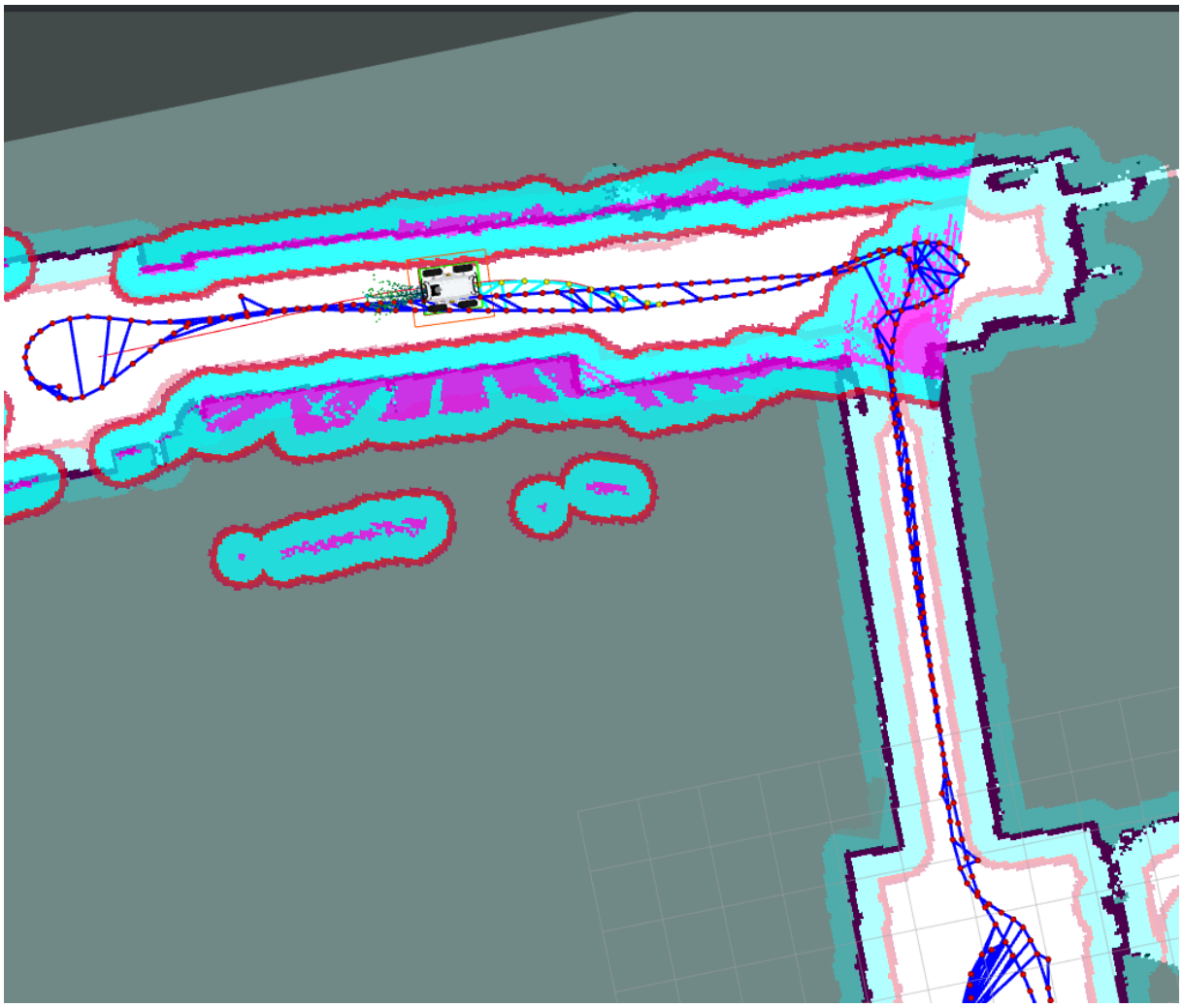


Figure 3.10: Autonomous Navigation with Nav2 in the hallways of building 7 of Politecnico di Milano

environments with dynamic obstacles. Therefore it was selected as the primary localization algorithm. The pose graph generated by the SLAM Toolbox algorithm is displayed in the maps, as shown in Figure 3.10 and Figure 3.11, where the blue lines represent the pose graph created during map creation, and the cyan lines represent the robot's the path traversed during navigation, estimated by the algorithm and connected to one of the nodes in the pose graph.

Global Planner: SMAC planners [13] are used as global path planners, in particular the Hybrid A* global planner. Hybrid A* performs well in generating optimal paths for the robot to follow, even in cases where an unknown obstacle appears in the environment. The Hybrid A* global planner uses an A* search algorithm to generate optimal paths for the robot to follow, which allows the robot to compute efficient paths and adjust

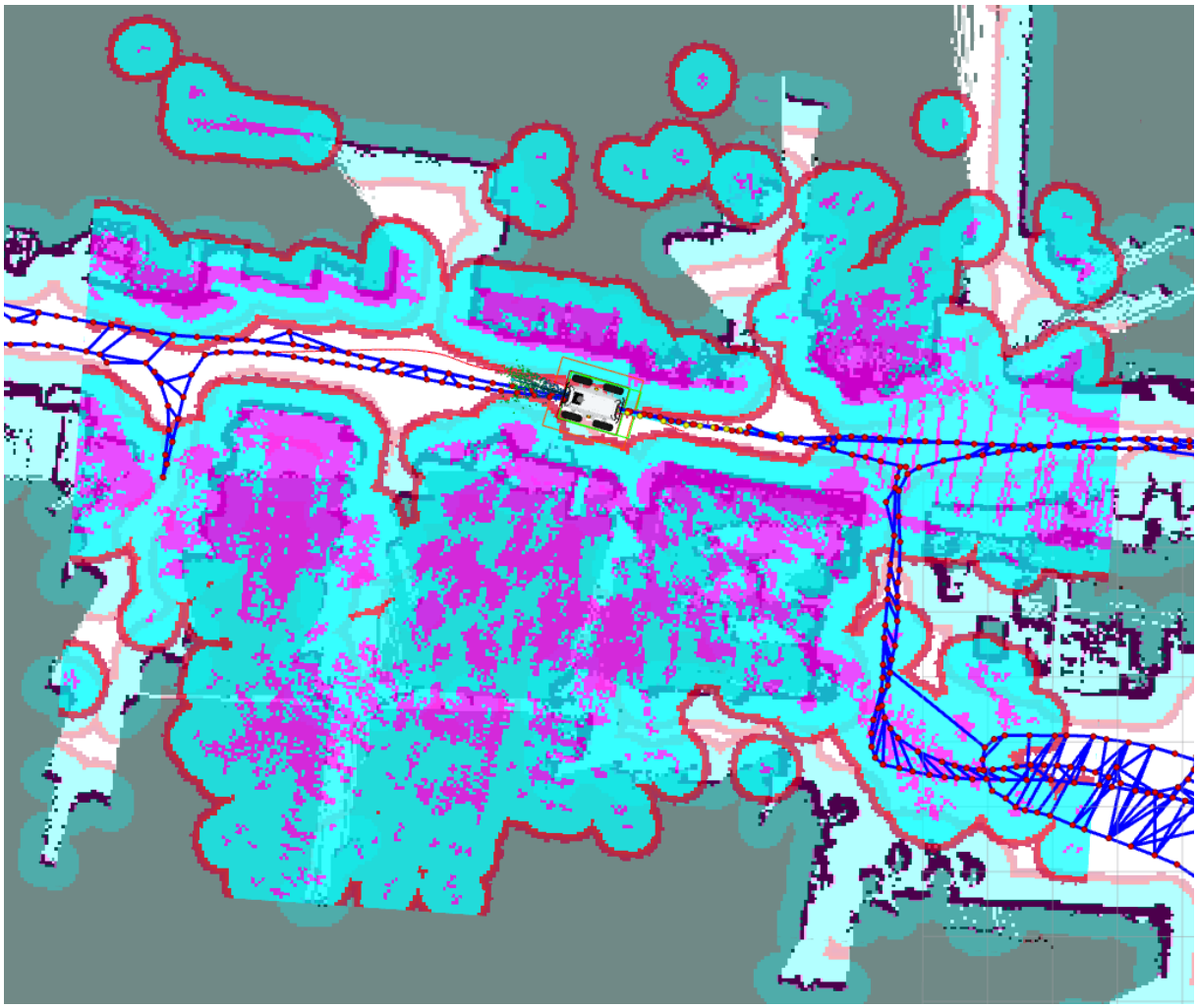


Figure 3.11: Autonomous Navigation in cluttered and dynamic environment with moving obstacles (AIRlab)

quickly to changes in the environment. The Hybrid A* global planner is based on the motion generation algorithm, which generates kinematically feasible paths for the robot to follow, and the path selection algorithm, which selects the best path from the sample paths based on the cost function. The motion generation algorithms available are the *Dubin* and *Reeds-Shepp* algorithms, which both generate kinematically feasible motion trajectories. The Reeds-Shepp algorithm enables the generation of paths also in the reverse direction, which is useful for navigating in reverse, especially for a skid-steering drive robot such as the Scout. However, probably due to some bugs within Ignition Gazebo, the Reeds-Shepp algorithm works only with the real Scout robot, and not with the simulated one in the Gazebo environment.

Local Planner (Controller): Two local planners were tested: DWB and MPPI. The MPPI (Model Predictive Path Integral Controller) [32] local planner uses a **model pre-**

dictive control approach to generate collision-free paths for the robot to follow, which allows the robot to navigate efficiently in complex environments with tight spaces and obstacles. The MPPI local planner also provides better performance in terms of trajectory tracking and collision avoidance, compared to the DWB local planner. The DWB local planner uses a dynamic window approach to generate collision-free paths for the robot to follow, which can be less effective in tight spaces and complex environments. The sampled trajectories of the local planner are visualized in Figure 3.10 and Figure 3.11, in front of the robot's footprint, showing the robot's sampled trajectories to follow the global path.

Recovery Behaviors: The default recovery behaviors provided by the recovery behavior tree plugin of Nav2 perform well in recovering the robot from unexpected scenarios, such as getting stuck or colliding with obstacles. However, in the most difficult scenarios, using the default recovery behaviors, the robot was not able to recover quickly from the situations where it was stuck near an obstacle, and it was unable to move for long periods. Therefore some slight modifications to the default recovery behaviors were applied to improve the robot's response in recovering from the scenarios where the robot would be unable to move, making it faster to recover and continue navigating towards the goal. The modified recovery behaviors include a combination of back-up and rotate behaviors, which allow the robot to back up and rotate in place to position itself in a different configuration, enabling it to generate a new path and continue navigating towards the goal.

Local Costmap: The local costmap plugin is used to generate a local costmap of the robot's surroundings, highlighting obstacles and free space. The local costmap plugin uses the robot's LiDAR sensor to perceive the environment and update the costmap in realtime. The local costmap plugin is used by the local planner to generate collision-free paths for the robot to follow. The local costmap includes the unknown obstacles in the environment, which are represented as pink areas in the costmap, as shown in Figure 3.11. The unknown obstacles are detected by the LiDAR sensor and updated in the costmap, allowing the robot to avoid collisions with objects not present on the map. The local costmap can be configured with these plugins:

- **Inflation Layer**: The inflation layer is used to inflate the obstacles in the costmap, creating a buffer around the obstacles to ensure that the robot avoids collisions.
- **Obstacle Layer**: The obstacle layer is used to update the costmap with the obstacles detected by the 2D LiDAR sensor, highlighting the obstacles in the costmap. This layer was eventually substituted with the voxel layer, thanks to the more effective 3D LiDAR sensor perception.
- **Static Layer**: The static layer is used to update the costmap with the static

obstacles in the map

- **Voxel Layer**: The voxel layer is used to update the costmap with the voxelized obstacles in the map, using the 3D LiDAR sensor to perceive the environment and update the costmap in realtime.

Global Costmap: The global costmap plugin is used to generate a costmap of the entire map that the robot is navigating in, highlighting the static obstacles and free space. The global costmap plugin is used by the global planner to generate optimal paths for the robot to follow. The global costmap is configured to use only the static layer inflated by the inflation layer, to ensure that the robot avoids collisions with the obstacles already present in the map. The global costmap is represented in Figure 3.10 as the blue and red areas, where the blue areas are the *lethal* space (i.e. must be avoided) and the red areas are the inflated obstacle areas with high cost.

Collision Monitor: The collision monitor plugin is used to continuously assess the robot's planned path against the costmap, ensuring that the robot navigates safely and avoids collisions with obstacles. The collision monitor plugin is used by Nav2 to ensure that the robot avoids the obstacles in the costmap, such that they do not overlap with the robot's footprint and locally planned path.

Mapping Algorithm: The algorithm of choice for mapping is *SLAM Toolbox*, a 2D SLAM algorithm that uses the robot's LiDAR sensor to create a 2D map of the environment [14]. SLAM Toolbox creates a 2D map of the environment used by the global planner to generate optimal paths for the robot to follow. The SLAM Toolbox proved to be effective in creating accurate 2D maps of the environment, even in dynamic environments with moving obstacles. This algorithm demonstrated good performance in terms of mapping accuracy and reliability, both in the simulated and real environments.

Velocity Smoother: The velocity smoother plugin is used to smooth the robot's velocity commands, ensuring that the robot moves smoothly and efficiently in the environment. The velocity smoother plugin is used by the local planner to obtain trajectories with a smooth velocity profile, computed from the initial optimal paths, which allows the robot to navigate efficiently and avoid jerky movements. The velocity smoother is configured also to avoid unnecessary stops and starts, which can wear out the robot's transmission gears and reduce the robot's battery life.

Spatio-Temporal Voxel Costmap Layer [15]: The dynamic obstacle avoidance algorithm is a critical component of Nav2, as it allows the robot to navigate safely in dynamic environments with moving obstacles. The dynamic obstacle avoidance algorithm uses the

robot's LiDAR sensor to perceive the environment and detect the moving obstacles in realtime. Dynamic obstacles update the local costmap following different plugins, such as the voxel layer, which is used to update the costmap with the voxelized obstacles in the environment. The voxel layer uses the 3D LiDAR sensor to perceive the environment and update the costmap. However, one of the shortcomings of the voxel layer is that it doesn't remove old voxels that correspond to obstacles that are no longer present in the environment. This leads to false positives in the collision checking, which prevents the robot from executing the planned trajectory, especially in highly dynamic environments with fast-moving obstacles. The spatio-temporal voxel costmap layer (STVL) is an open-source plugin that addresses this issue by updating the costmap efficiently and accurately. It replaces the traditional voxel grid approach with a **sparse voxel grid** implemented using OpenVDB, a **high-performance C++ library**. This allows STVL to handle large, dynamic environments with ease, such as the one shown in Figure 3.12, reducing computational overhead. By leveraging temporal information and applying a voxel grid filter to sensor data, STVL can better handle noisy and dense sensor readings, especially in proximity to objects. This results in smoother, more reliable navigation and improved robot performance in continuously evolving scenarios. STVL's adaptability, efficiency, and ability to integrate with various sensor types made it the ideal substitute for the voxel layer in Nav2. Figure 3.11 shows the effectiveness of the STVL in a dynamic and cluttered environment, such as the AIRLab, where the robot is navigating despite many unknown obstacles and moving objects in a tight space.

3.7.2. Optimal DDS configuration

In the initial stages of the experimental activities, we encountered various issues related to the LiDAR. Such as low message frequency, significant gaps in measurement production, and crashes in the applications. This would lead the robot not to navigate correctly, and often not perceiving the obstacles in the immediate vicinity. Overall Nav2 was not reliable and robust, especially in dynamic and cluttered environments. Figure 3.13 shows the pointcloud generated from the 3D LiDAR sensor in a cluttered environment. When the sensor is working correctly, it can provide a detailed and accurate representation of the environment, at a constant and stable frequency. However, the LiDAR sensor was not working correctly, so the pointcloud data was received with gaps and delays.

The LiDAR sensor, integral to localization and obstacle avoidance algorithms, exhibited low and unreliable frequency, ranging from 3Hz to 8Hz, instead of its nominal 10Hz. This variability, primarily attributed to the combined effects of the router connecting the robot to the network and the default Data Distribution Service (DDS) middleware configuration

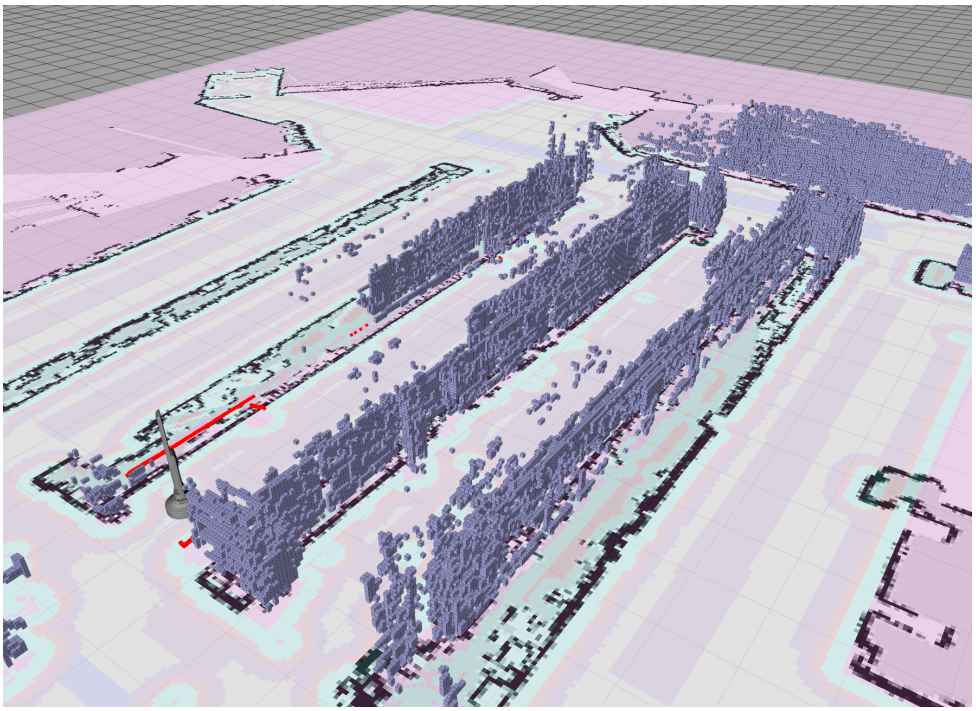


Figure 3.12: STVL in action in a dynamic and cluttered warehouse environment

within ROS2, caused significant disruptions to the software's operation.

The DDS middleware, configured to broadcast all ROS2 packets to the laboratory network, introduced delays and packet loss due to the network's inability to handle the high-frequency, heavy-weight LiDAR data streams. This unreliability in data transmission not only affected the LiDAR sensor but also tampered with the proper initialization and function of the IMU sensor used in SLAM algorithms.

To resolve these issues, the DDS middleware was meticulously reconfigured to ensure that all ROS-related nodes and data streams remained local to the robot's computer, preventing them from traversing the router and laboratory network. This change effectively eliminated the broadcast-induced delays and packet loss. Additionally, ROS2 was configured to utilize the Cyclone DDS middleware, a lightweight and fast implementation particularly suited for real-time, high-frequency heavy-data transmission. Cyclone DDS proved capable of reliably handling the heavy LiDAR data packets at high frequencies, ensuring their complete and timely transmission.

Following these configuration changes, the LiDAR sensor consistently delivered a stable scan rate of 10Hz, resolving the software crashes and malfunctions previously observed. The system was further tested with the LiDAR operating at a higher frequency of 20Hz with lower resolution, and stability was maintained.

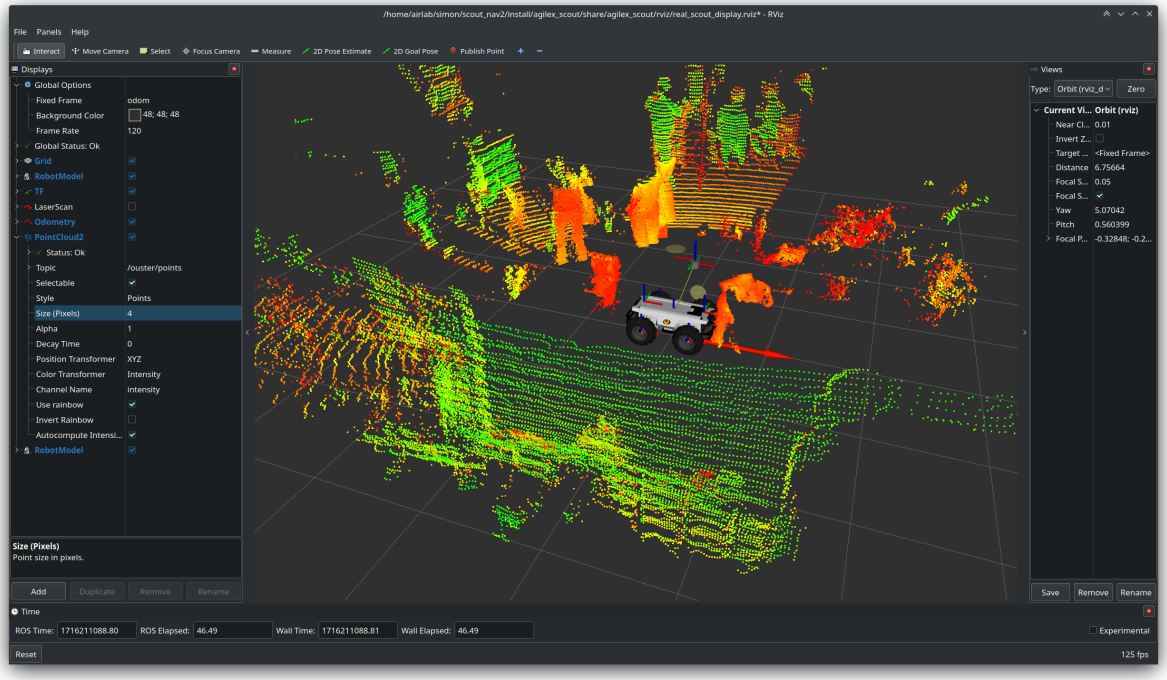


Figure 3.13: 3D LiDAR sensor in a cluttered environment

This solution not only fixed the immediate LiDAR frequency issue but also had cascading positive effects on other system components. The IMU sensor, crucial for the SLAM algorithm, was now able to initialize correctly with a stable LiDAR data stream, eliminating odometry drift. Numerous other algorithms and software components benefited from the improved data reliability, resulting in robust and reliable navigation performance for autonomous tasks.

The Cyclone DDS configuration file, shown in 3.1, establishes settings for a domain with ID "any," indicating it can participate in any DDS domain. It restricts communication to the local loopback interface ("lo") and explicitly disables multicast for both discovery and data transmission. This configuration also allows Cyclone to automatically assign unique identifiers to participants within the domain. For peer discovery, it targets only the local machine ("localhost"). This configuration prioritizes local, unicast communication and controlled participant identification.

```
<CycloneDDS xmlns="https://cdds.io/config"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Domain id="any">
        <General>
            <Interfaces>
                <NetworkInterface name="lo"/>
            </Interfaces>
```

```
8               <AllowMulticast>false</AllowMulticast>
9           </General>
10          <Discovery>
11              <ParticipantIndex>auto</ParticipantIndex>
12              <Peers>
13                  <Peer Address="localhost"/>
14              </Peers>
15              <MaxAutoParticipantIndex>120</MaxAutoParticipantIndex>
16          </Discovery>
17      </Domain>
18  </CycloneDDS>
```

Listing 3.1: Cyclone DDS configuration file

3.8. Parking Algorithm for Mobile Robot

A **parking algorithm** was designed for the mobile robot to autonomously approach the target location to better perform manipulation activities. The parking algorithm takes as input the location in the map of the target spot that the robotic arm must reach, and it outputs the optimal parking pose to give the robotic arm enough space to reach the target. The algorithm is designed and structured to ensure that the mobile robot can park in a position where the robotic arm can interact with the target object, without colliding with the obstacles in the environment nor with the target object itself. The mobile robot must park so that it faces the opposite direction of the target object because the robotic arm is mounted on the back of the robot.

The parking algorithm considers both the robot's footprint and uses a heuristic to approximate the robotic arm's reachability constraints to ensure that the robot can park in a position where the robotic arm should be able to reach the target object. This heuristic does not guarantee that the robotic arm can always reach the target object, but it works as a good approximation of the robotic arm's workspace. The algorithm also considers the costmap generated by Nav2 to compute the optimal parking pose that minimizes the cost of the footprint, meaning that the robot will be the furthest possible from the obstacles in its vicinity.

The parking algorithm is then used by the Nav2 commander APIs to send the parking pose as a ROS2 action goal, which generates a collision-free path for the robot to follow to reach the parking pose. The robot then executes the planned trajectory and parks near the computed parking pose. While the mobile robot navigates autonomously to the parking pose, it publishes feedback data containing its current position and its distance

from the parking pose.

The parking algorithm is non-deterministic, meaning that it can compute different parking poses for the same target pose, depending on the obstacles in the environment. The algorithm's pseudocode is displayed in 3.1.

Algorithm 3.1 Parking Pose Computation Algorithm

Require: Target $t = (x_t, y_t, \theta_t)$
Require: Costmap C, Robot's Footprint F
$n \leftarrow 50$ ▷ number n of parking pose candidates p_c
$r \leftarrow 0.4$ ▷ parking radius in meters from the target position
$P_{current} \leftarrow$ current robot's position in the map
$V_c \leftarrow \emptyset$ ▷ list of valid candidate poses
function RANK(p)
    $w_{cost} = 0.5, w_{dist} = 0.2, w_{orientation} = 0.3$
    $cost = cost(F(p), C)$ ▷ cost in the costmap for given pose and footprint
    $dist = ||P_{current} - p||^2$ ▷ Euclidean distance between poses
    $orient = |\theta_t - \theta_p|$ ▷ difference between target and candidate orientation
    return $w_{cost} \cdot cost + w_{dist} \cdot dist + w_{orientation} \cdot orient$
end function
for $i = 1$ to n **do**
    $\phi \sim Uniform(-\phi_{max}, \phi_{max})$ ▷ random angle
    $x_c \leftarrow x_t + r \cdot \cos(\theta_t + \phi)$ ▷ candidate x coordinate
    $y_c \leftarrow y_t + r \cdot \sin(\theta_t + \phi)$ ▷ candidate y coordinate
    $\psi \sim Uniform(-\psi_{max}, \psi_{max})$ ▷ random orientation
    $\theta_c \leftarrow \theta_t + \psi$ ▷ candidate orientation
    $p_c \leftarrow (x_c, y_c, \theta_c)$ ▷ candidate parking pose
        ▷ discard the poses having their footprint F with lethal cost in costmap C
    if $cost(C, F(p_c)) \neq LETHAL$ **then**
        add p_c to V_c
    end if
end for
$V_{ranked} \leftarrow [\mathbf{rank}(p_{c,1}), \ldots, \mathbf{rank}(p_{c,n_c})] \quad \forall p_c \in V_c$
$V_c \leftarrow sorted(V_{ranked}, V_c)$ ▷ sort list of candidates by their rank
repeat
    pick $P_{candidate} \in V_c$ ▷ pick parking pose from the highest ranked candidates
    if $\exists$ traversable path from $P_{current}$ to $P_{candidate}$ **then**
        save parking pose $P_{parking} \leftarrow P_{candidate}$
    end if
until a traversable path is found
if $\exists P_{parking}$ **then**
    navigate towards $P_{parking}$ with computed traversable path
end if

The algorithm 3.1 prioritizes poses that are closer to the target, have lower costs in the costmap, and are aligned with the target orientation. The use of random sampling allows for exploring a variety of potential parking spots. It also ensures that the chosen parking pose is reachable by checking for a traversable path. The algorithm uses the following inputs and parameters:

- t: the target pose (final position and orientation): (x_t, y_t, θ_t).
- C: the costmap: grid-based representation of the environment, where each cell has a cost associated with it.
- F: the robot's footprint: shape and size of the robot, which is used to check for collisions with obstacles.
- n: the number of parking pose candidates to generate (default: 50)
- r: the parking radius around the target position (default: 0.4 meters)

The algorithm 3.1 consists of the following steps:

1. **Rank Function** definition: A function that evaluates the quality of a candidate pose based on cost, distance, and orientation. It calculates a weighted sum of these factors: cost (based on the costmap), distance (from the current position), and orientation (difference from the target pose). The weights determine the relative importance of each factor.
2. **Candidate Generation and Filtering**: The algorithm iterates n times to generate candidate poses:
   (a) A random angle (ϕ) is chosen within specified limits.
   (b) The candidate pose's x and y coordinates (x_c, y_c) are calculated based on the target position, parking radius, and random angle.
   (c) A random orientation (ψ) is chosen within limits.
   (d) The candidate orientation (θ_c) is calculated.
   (e) The candidate pose (p_c) is formed using (x_c, y_c, θ_c).
   (f) The cost of the robot's footprint at p_c is checked. If it's not lethal (i.e., not colliding with an obstacle), p_c is added to the list of valid candidates (V_c).
3. **Ranking and Sorting**: The rank function is applied to each valid candidate pose in V_c. The list of candidates (V_c) is sorted in ascending order based on their ranks (lower rank is better).

4. **Path Planning and Navigation**: The algorithm repeatedly picks the highest-ranked candidate pose from V_c. It checks if a traversable path exists from the current position ($P_{current}$) to the candidate pose. If a path is found, the candidate pose is saved as the parking pose ($P_{parking}$). This process continues until a traversable path is found or all candidates are exhausted.

5. **Navigation (if applicable)**: If a suitable parking pose ($P_{parking}$) is found, the robot navigates towards it using the computed traversable path.

Experimental tests were performed to evaluate the accuracy and reliability of the parking algorithm. The tests were conducted in both simulated and real environments, with different configurations of obstacles and target poses. The parking algorithm was able to compute effective parking poses in most cases, but it struggled with tight spaces and narrow passages, where the robot had limited space to park. The biggest limitation of this algorithm is that it **does not consider the exact feasible workspace** of the robotic arm when computing the parking pose, which can lead to the robot parking in a position where the robotic arm cannot reach the target object. It instead uses a heuristic to approximate the robotic arm's workspace, which is not always accurate, leading to suboptimal parking poses in some cases. This limitation is due to the impossibility of predicting the cartesian target pose that the cobot will have to reach, as it depends on the object's position and orientation in the environment. The parking algorithm is a good starting point for further development and improvement to address these limitations and ensure that the robot can park in a useful position also for the robotic arm.

Despite the parking algorithm being effective in most cases, there were some cases where the robot parked too close or too far from the object where the robotic arm would interact. This was mainly due to the **imprecision of the localization algorithm** and the local planner's inability to reach the exact parking pose, especially in tight spaces. This problem resulted in a precision error in the final position of the robot, in the range of $\pm$15cm from the desired parking pose. This problem was critical because an error of just a few centimeters could result in the robot being too close to the object to find it or interact with it, or too far. Since no ideal solution exists that can compensate for the localization and navigation errors, a workaround was implemented. When the robot cannot interact with the object because of the distance to it, the robot chooses and navigates to a different parking pose, and tries to interact with the object from the new position.

3.9. ArUco Marker Detection and Pose Estimation

ArUco markers are square markers with unique black-and-white patterns that can be easily detected and identified by computer vision algorithms. Detecting and estimating the pose of an ArUco marker from an image involves a two-step process. First, the ArUco marker is detected in the image using the ArUco library available in OpenCV. This library provides functions to detect various ArUco dictionaries and families. Once detected, the marker's corners are extracted, and its unique ID is identified. The second step involves estimating the pose of the ArUco marker, which refers to its position (translation vector) and orientation (rotation vector) with respect to the camera. The function for pose estimation takes the detected marker corners, the marker size, and the camera's intrinsic parameters (i.e. focal length, principal point, and distortion coefficients) as input and returns the pose of the marker in the form of a rotation vector and a translation vector.

The intrinsic parameters of the camera are crucial for accurate pose estimation. These parameters describe the camera's internal characteristics, such as the focal length, which determines the field of view, and the principal point, which is the center of the image. The distortion coefficients model the lens distortion, which can cause straight lines to appear curved in the image. By incorporating these parameters into the pose estimation process, we can compensate for the camera's inherent distortions and obtain a more accurate estimate of the ArUco marker's pose in the real world. It was also necessary to calibrate the camera's intrinsic parameters using a checkerboard calibration pattern and the automatic calibration software tool provided by the RealSense SDK.

The dictionary for the ArUcos used throughout the project is the 4x4. It is useful for locating the marker signaling the control panel from a distance, instead of the 7x7 dictionary markers initially used. The 4x4 markers require fewer pixels to be represented, therefore the minimum area of pixels required for detection is smaller. This implies that the 4x4 markers can be detected from a greater distance than the 7x7 markers.

3.9.1. Multi-ArUco Plane Estimation Algorithm

One of the challenges faced was estimating with precision the orientation of small ArUco markers from the camera feed. The pose estimation algorithm works well for markers that appear large in the image, but it struggles with the ones that appear smaller due to the size and distance from the camera. The estimation of the orientation was the most challenging part, as the pose estimation algorithm often returns orientation values that oscillate between different values, making it difficult to determine the correct orientation

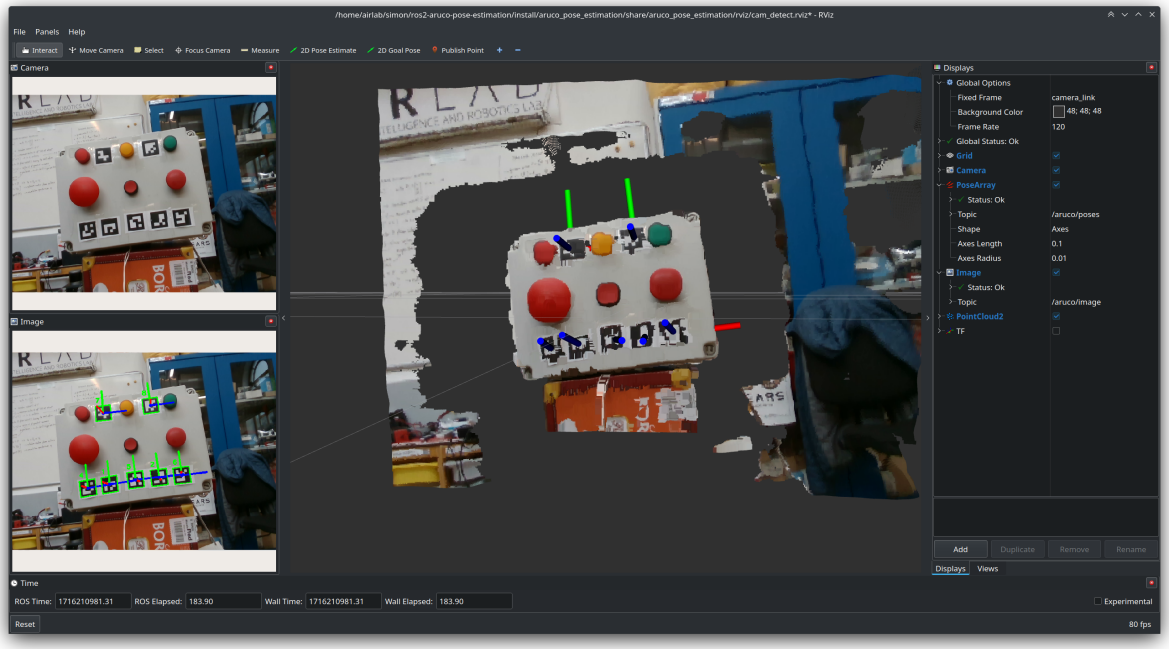


Figure 3.14: Multi-ArUco Plane Estimation algorithm in action. The screenshot shows RViz2 displaying on the top left corner the input image, the bottom left the image with the detected markers drawn on top of it, and in the center the colored pointcloud captured by the depth sensor.

of the marker. To address this issue, a multi-ArUco plane estimation algorithm was designed for estimating the orientation of a plane over which multiple ArUco markers are placed. Figure 3.14 shows the algorithm in action.

The algorithm works by detecting multiple ArUco markers in the image and estimating their poses using the pose estimation algorithm. The poses of the ArUco markers are then used to estimate the orientation of the plane on which the markers are placed. The algorithm assumes that the ArUco markers are **coplanar**, meaning that they have different positions but the same orientation. By estimating the orientation of the plane that passes through the markers, we can determine the correct orientation of the markers from the vector normal to the plane. The algorithm processes a list of ArUco markers poses and returns the poses with their estimated orientation, meaning that all processed poses share the same orientation value.

The algorithm makes use of the following statistical data analysis techniques:

- **RANSAC (Random Sample Consensus)**: RANSAC is an iterative algorithm used to estimate the parameters of a mathematical model from a set of observed data points that contain outliers. RANSAC works by randomly selecting a subset

of data points and fitting a model to them. The model is then evaluated against the remaining data points, and the points that are consistent with the model are considered inliers. The process is repeated multiple times to find the best-fitting model with the maximum number of inliers. RANSAC is used to discard the outlier data points observed in the ArUco markers' poses.

- **Least Squares Estimation (LSE)**: LSE is a mathematical method used to find the best-fitting curve that minimizes the sum of the squared differences between the observed data points and the model's predictions. LSE is used by the RANSAC model to fit the plane that passes through the ArUco markers' poses.
- **Principal Component Analysis (PCA)**: PCA is a statistical method that identifies patterns in data by transforming it into a new coordinate system, where the data points are uncorrelated. PCA is used to find the principal components of the data, which are the directions of maximum variance. In the context of this algorithm, PCA is used to find the vector passing through points lying on a line. It is used as a technique for outlier removal and noise reduction in the data.
- **Singular Value Decomposition (SVD)**: SVD is a matrix factorization technique that decomposes a matrix into three matrices, which represent the singular vectors and singular values of the original matrix. SVD is used as an optimization technique to find the best-fitting plane that passes through the given points. It works as a technique for reducing the dimensionality of the data, by reducing the impact of noisy and irrelevant data points.

The perception algorithm works as illustrated in 3.2. The points used as input for the algorithm are the 3D coordinates of the ArUco markers detected in the image.

Algorithm 3.2 Multi-ArUco Plane Estimation Algorithm

Require: points $P = [p_1, \ldots, p_n]$ with $p_i = (x_i, y_i, z_i)$ coordinates of all points
Require: collinear points $T = [t_1, \ldots, t_n]$ with $t_i = (x_i, y_i, z_i)$ assumed to be collinear

function RANSAC(P)
    $\tau \leftarrow 0.01$ ▷ distance threshold in meters
    repeat
        $S =$ random subset of points from P
        $c = \text{centroid}(S)$
        $n = SVD(S - c)$ ▷ Compute SVD from subset of points centered in 0
        inliers $= 0$ ▷ number of inliers for S
        for all s_i in S **do**
            $d = n \cdot s_i$ ▷ distance between normal n and point s_i
            if $||d||^2 < \tau$ **then** ▷ if point s_i is close enough to the plane defined by n
                inliers ++
            end if
        end for
    until inliers number is maximized
    return n ▷ Returns normal vector n to the best fitting plane to S
end function

$n =$ **RANSAC**(P)
$c = centroid(T)$
$B =$ **PCA**(T) ▷ compute **PCA** of collinear points without centroid
$d = B[2]$ ▷ direction of highest variance is the last eigenvector in B
$V_x \leftarrow d, \quad V_z \leftarrow n$
$V_y \leftarrow V_z \times V_x$
$Rot = [V_x, V_y, V_z]$ ▷ compose 3d rotation matrix from vectors
$q = rot2quat(Rot)$ ▷ convert rotation matrix to quaternion q
update points in P with the orientation quaternion q

The algorithm 3.2 requires the following inputs:

- P: A set of 3D points (x, y, z coordinates) from the list of detected ArUco markers estimated poses, assumed to be all on the same plane (coplanar).
- T: A subset of points from P that are known to be collinear.

The algorithm consists of the following steps:

1. **RANSAC Plane Fitting**: The RANSAC function is applied to the full set of points (P) to obtain the normal vector (n) of the plane.
   (a) Initialization: A distance threshold (τ) is set to determine how close points need to be to a plane to be considered inliers.
   (b) Iteration: The algorithm iteratively performs the following:
      i. A random subset of points (S) is selected from P.
      ii. The centroid (c) of S is calculated.
      iii. Singular Value Decomposition (**SVD**) is applied to S after centering it around the origin. This yields a normal vector (n) representing the best-fitting plane to S.
      iv. The number of inliers is counted. Inliers are points from S whose distance to the plane defined by n is less than τ.
   (c) Termination: The loop continues until the maximum number of inliers is found.
   (d) Output: The normal vector (n) of the best-fitting plane is returned.
2. **Plane Orientation Refinement**:
   (a) Normal Calculation: The RANSAC function is applied to the full set of points (P) to obtain the normal vector (n) of the plane.
   (b) Centroid Calculation: The centroid (c) of the collinear points (T) is calculated. The centroid is removed from the points to ensure that the plane passes through the origin.
   (c) Direction Vector: Principal Component Analysis (PCA) is performed on T, and the eigenvector corresponding to the highest eigenvalue is selected as the direction vector (d). This vector lies within the plane and represents the direction of greatest variance among the collinear points.
   (d) Coordinate Frame Construction: An orthonormal basis is constructed:
      - V_x: The direction vector (d).
      - V_z: The plane's normal vector (n).
      - V_y: The cross product of V_z and V_x.
   (e) Rotation Matrix and Quaternion: The basis vectors are used to create a rotation matrix (Rot), which is then converted to a quaternion (q). This quaternion

represents the orientation of the plane.

3. **ArUco Markers Poses Update**: The points in P are updated by applying the rotation quaternion (q) to align them with the estimated plane orientation.

4. **Output**: The algorithm produces a quaternion (q) that describes the orientation of the plane in 3D space. All ArUco markers' poses are updated with this orientation, ensuring that they share the same orientation.

3.10. Object Detection with YOLOv8

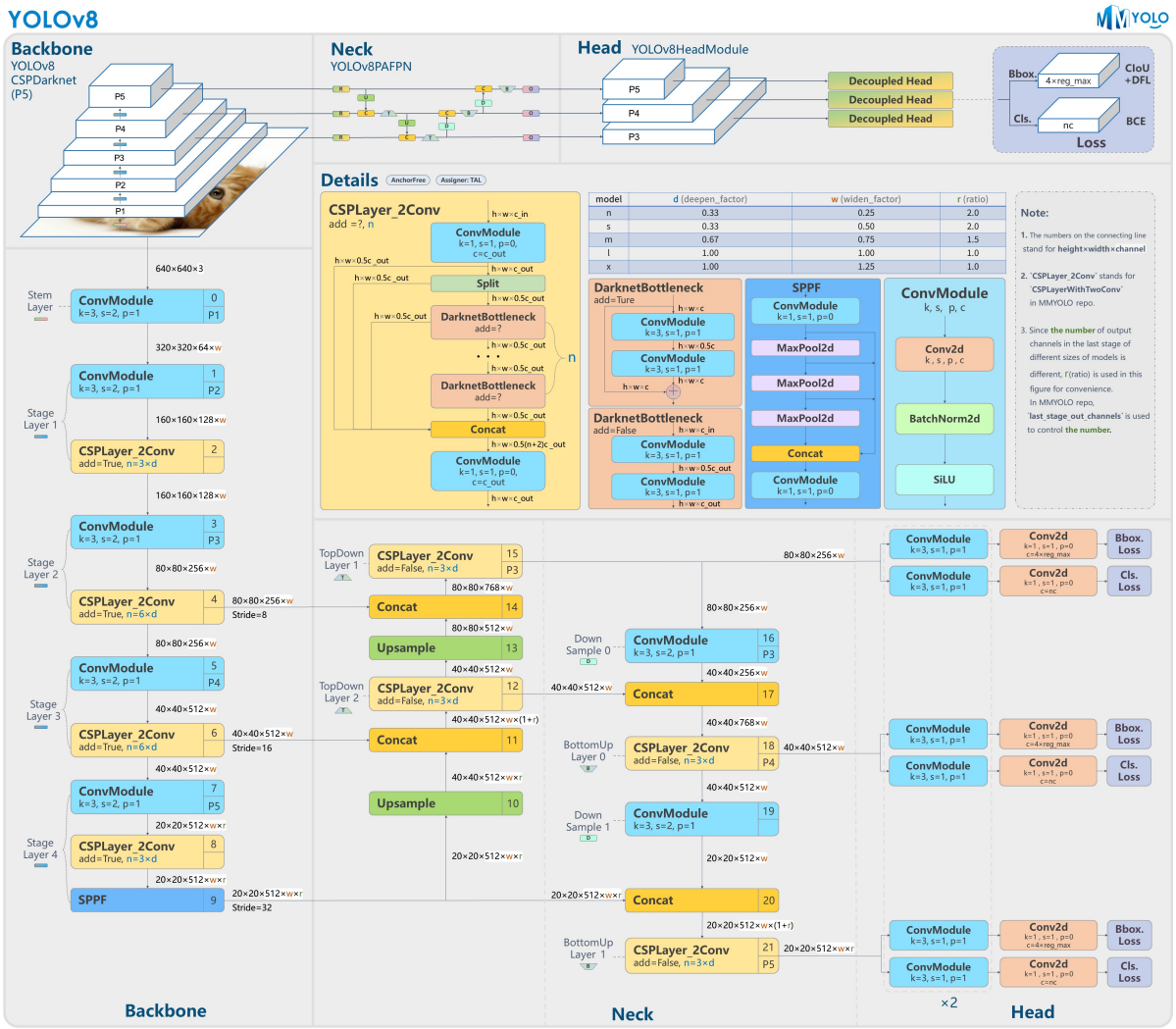


Figure 3.15: Architecture overview in detail of the YOLOv8 architecture

For the project, it was also necessary to detect objects in the environment, such as colored balls and apples, which the robot had to grasp and move to a different location. For this

task, YOLOv8 object detection is employed, trained on a custom dataset. Figure 3.16 shows the YOLOv8 model in inference in realtime, predicting the colors of the balls in the field of view of the camera.

YOLO (You Only Look Once) is a cutting-edge, real-time object detection algorithm that has revolutionized computer vision [23]. Unlike traditional methods that require multiple passes over an image, YOLO analyzes the entire image in a single pass, making it fast and efficient. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously, achieving high accuracy while maintaining speed. YOLO has evolved through several versions (YOLOv2, YOLOv3, etc.), each improving upon the previous iteration in terms of speed, accuracy, and ability to detect small objects. Its versatility and effectiveness have led to widespread adoption in various applications, including autonomous driving, robotics, and image analysis tools. This was the object detection neural network of choice for the project, as it provided the speed and accuracy needed for detecting objects in real-time from the camera feed. It is also lightweight, making it ideal to run on CPUs without the need for a GPU. The inference time for a single image on an Intel i7 12th gen CPU was around 0.15 seconds, which is fast enough for the time requirements of the application. However, when running along other ROS2 nodes using other CPU-intensive algorithms, the inference time increased to 0.4 seconds, which was still acceptable for the demos and tests.

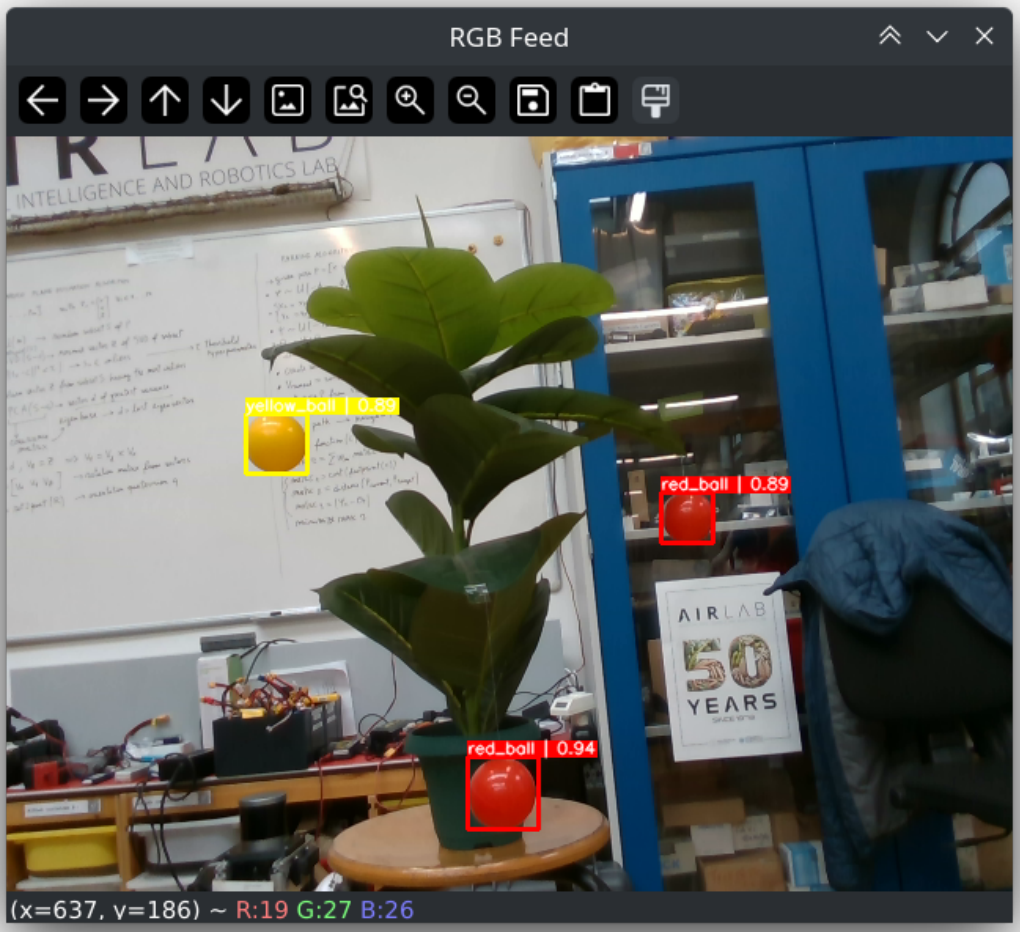


Figure 3.16: YOLOv8 detecting colored balls in realtime from the RGB camera feed. The screenshot displays the image with the bounding boxes colored with the same color as the predicted class. For each prediction, the class probability is associated.

The **YOLOv8** neural network model is a state-of-the-art object detection model [24] that combines the best features of previous YOLO versions to achieve superior performance. This model is used for the object detection task, such as detecting colored balls or apples from the camera feed. The YOLOV8 architecture is shown in Figure 3.15. It shows the backbone architecture of the YOLOv8 model, which consists of a series of convolutional layers that extract features from the input image and pass them to the detection head, which predicts the bounding boxes and class probabilities for the objects in the image. The backbone architecture used in the training is CSPDarknet, which is a state-of-the-art backbone. The neck is a feature pyramid network (FPN) that combines features from

different scales to improve the model's performance in detecting objects of different sizes.

The YOLOv8 model is trained on a **custom dataset of colored balls and apples**, which includes images of balls and apples from different angles, distances, and lighting conditions. The dataset was collected using the RealSense camera mounted on a tripod, which allowed me to capture images of the objects from different perspectives and distances. The dataset was annotated manually with the bounding boxes and class labels, which were used to train the YOLOv8 model. The YOLOv8 model is provided by **Keras**, a high-level deep-learning library that provides user-friendly APIs for building and training neural networks. Since the custom training dataset collected is of small size, data augmentation techniques were employed to increase the dataset's size and diversity, which helped improve the model's generalization and robustness. The **data augmentation techniques** included random rotations, translations, and flips of the images, which created variations of the original images and helped the model learn to detect the balls from different perspectives and orientations. Many types of image augmentation techniques were included, such as brightness and contrast adjustments, and Gaussian noise addition, which further increased the dataset's size and variability. The image augmentation algorithms were implemented using the *Albumentations* library, a powerful image augmentation library for computer vision tasks, designed for object detection tasks. It is essential to properly adjust the bounding boxes after each image is augmented, to ensure that the bounding boxes are still correctly positioned around the objects of interest.

As training hyperparameters, the batch size is 32, which is enough for a small dataset, a learning rate scheduler with exponential decay, and the early stopping callback to stop the training when the validation loss stops decreasing. The model is compiled with the *Adam* optimizer, which is a popular optimizer for training deep neural networks, and with the YOLOv8 Backbone, which is a state-of-the-art backbone architecture that provides the best candidate features for object detection tasks. The metrics used to evaluate the model's performance are the **mean Average Precision (mAP)** and the **Intersection over Union (IoU)**, which are standard metrics for object detection tasks. These metrics are incorporated within the *COCO* evaluation metrics, which are the standard evaluation metrics provided by the **KerasCV** library [33].

The model is trained with the combination of two loss functions:

- **Box Loss**: The box loss function is used to penalize the model for incorrect predictions of the bounding boxes of the objects. The box loss function computes the difference between the predicted bounding boxes and the ground-truth bounding boxes, using the $CIoU$ (Complete Intersection over Union) loss function, which is a

Metric	IoU	Area	Value
AP	0.50:0.95	all	0.452
AP	0.50	all	0.616
AP	0.75	all	0.552
AP	0.50:0.95	small	0.232
AP	0.50:0.95	medium	0.512
AP	0.50:0.95	large	0.673
AR	0.50:0.95	all	0.590
AR	0.50:0.95	small	0.284
AR	0.50:0.95	medium	0.659
AR	0.50:0.95	large	0.780

Table 3.1: Evaluation metrics for the trained YOLOv8 model

state-of-the-art loss function that accounts for the object's aspect ratio and orientation [36].

- **Class Loss**: The class loss function is used to penalize the model for incorrect predictions of the object classes. The class loss function computes the difference between the predicted class probabilities and the ground-truth class labels, using the *Binary Cross-Entropy* loss function, which is the loss function of choice for multi-class classification tasks that use multi-hot encoded labels instead of one-hot encoded labels. After some tests, the class loss was switched to the *Binary Penalty Reduced Focal CrossEntropy* loss function, a variant of the focal loss function that reduces the penalty for misclassifications, and focuses more on the correct classification of the objects [11]. This loss function helped improve the model's accuracy.

The table below 3.1 shows the **evaluation metrics** for the object detection model, which includes the Average Precision (AP) at different Intersection over Union (IoU) thresholds, the Average Recall (AR), and the AP for different object sizes (small, medium, large). The Intersection over Union thresholds define the level of overlap required between the predicted bounding box and the ground-truth bounding box for the prediction to be considered correct. The AP is a measure of how well the model detects objects across different levels of overlap, while the AR is a measure of how well the model finds all ground truth objects. The AP and AR are computed for different object sizes to evaluate the model's performance on objects of different scales. The evaluation metrics provide insight into the model's accuracy, recall, and generalization capabilities, which are essential for assessing the model's performance on unseen data.

The table below 3.2 shows the hyperparameters and model parameters used in the training and compiling of the YOLOv8 model. The hyperparameters include the number of epochs,

Hyperparameters	**Value**
Epochs	42 (Early Stopping)
Batch Size	32
Image Size	640×480
Initial Learning Rate	0.001
Learning Rate Scheduler	Staircase Exponential Decay
Optimizer	Adam
Momentum	0.9
Weight Decay	0.0005
Global Clip Norm	10.0
Classification Loss	Binary Penalty Reduced Focal CrossEntropy
Box Loss	Complete Intersection over Union
Model Parameters	**Value**
Model Size	Small
Trainable Parameters	14M
Trainable Layers	4
Number of Classes	5 (4 balls, 1 apples)
Prediction Decoder	Multi-Class Non-Maximum Suppression
Confidence Threshold	0.7
IoU Threshold	0.4

Table 3.2: Hyperparameters and model parameters used in YOLOv8 training

batch size, learning rate, optimizer, and loss functions used in the training process. The model parameters include the classification loss, box loss, model size, number of trainable parameters, and the number of classes detected by the model.

One problem faced with this network was the **high false positive rate** of the trained YOLOv8. The neural network was trained on a relatively small dataset of images, and it was not able to generalize well enough to new images. The neural network was detecting objects that were not present in the image, and this resulted in the algorithm computing grasping poses for objects that were not balls or apples. This problem was addressed by increasing the threshold for the class confidence score, which is the minimum confidence score required for the object to be considered a valid detection. By increasing the threshold, the number of false positives was reduced, and the algorithm became more accurate in detecting the objects of interest. The threshold is set to 0.7, which is a good trade-off between accuracy and false positives.

3.11. Pose Estimation with Object Detection and Depth Perception

The object's pose estimation task involves estimating the position of an object in 3D space using the depth-sensing camera and the object detection neural network. The object detection algorithm detects the object in the RGB image and provides the bounding box coordinates, class label, and confidence score. The depth-sensing camera provides the depth map of the scene, which contains the distance of each pixel from the camera. By combining the object detection results with the depth map, we can estimate the object's position in 3D space relative to the camera, which is essential for the robot to interact with the object effectively. The perception algorithm is described in detail in the following section.

This perception algorithm works by creating a **segmented pointcloud** of the object from the depth map, using the bounding box coordinates provided by the object detection algorithm, and the projected depth values from the depth image. This perception algorithm is implemented to estimate the center position of detected colored balls or apples. The segmented pointcloud contains only the points that belong to the object, which are used to estimate the object's position in 3D space. The segmentation process involves extracting the points within the bounding box and filtering out the points that are not part of the object using a color mask corresponding to the predicted class label. The segmented pointcloud is then used to estimate the object's center position by fitting an ideal sphere of known radius to the segmented points, as described in algorithm 3.3. This center position is then used to compute the optimal grasping pose for the robot to interact with the object, as described in algorithm 3.4.

This perception algorithm is used to estimate the optimal grasping pose for the robot's end-effector to grasp the object. The algorithm 3.4 works with both colored apples, that have a spherical shape with measurable radius, and with apples, that have a more irregular shape. The apples are approximated as spheres for the grasping pose estimation, and their radius is approximately 5 cm. This value corresponds to half of the width of the upper part of the apple, which is the part that the robot's end-effector will grasp.

3.11.1. Algorithm for Object's Center Estimation

The algorithm for the object's center estimation from the surface pointcloud is the one described in 3.3. This algorithm uses a random sample consensus ($RANSAC$) algorithm to estimate the object's center, assuming that the object can be approximated as a sphere of

Algorithm 3.3 Sphere Barycenter Estimation from Object Detection

Require: RGB image I, Depth image D
Require: predicted bounding box $B = (x, y, w, h)$, predicted class label $\hat{y}$
sphere radius range r_{min}, r_{max}, tolerance ϵ
$d_{max} \leftarrow 1.5$ ▷ maximum depth of useful points in meters
$I_{crop} \leftarrow I[y : y + h, x : x + w]$ ▷ Crop the image to the bounding box
$D_{crop} \leftarrow D[y : y + h, x : x + w]$ ▷ Crop the depth image to the bounding box
$P \leftarrow get\_pointcloud(D_{crop})$ ▷ Get the pointcloud from the depth image
filter pointcloud P by removing points with $z \geq d_{max}$
$colormask \leftarrow get\_colormask(\hat{y})$ ▷ Get the color mask based on predicted class label
$P_s \leftarrow \emptyset$ ▷ Object's surface segmented pointcloud
for each pixel $p \in I$ **do**
    if color of p is within $colormask$ **then**
        $P_s \leftarrow P_s \cup P(p)$ ▷ Apply color mask filter and add point to surface pointcloud
    end if
end for
$n_{min} \leftarrow 4$ ▷ minimum number of points passing through a unique sphere
for a fixed number of iterations **do** ▷ Random sample consensus algorithm
    $S \leftarrow$ random subset of n_{min} points from P_s
    $c, r \leftarrow$ center and radius of sphere fit to subset S
    if $r_{min} \leq r \leq r_{max}$ **then**
        Calculate *inliers*: points within a distance threshold ϵ of the sphere's surface
        Calculate *MLESAC* score based on the number of inliers and the residual error
        if $score > best\_score$ **then**
            Update $best\_model \leftarrow (c, r)$, $best\_score \leftarrow score$
        end if
    end if
end for
Refine $best\_model$ by fitting the *inliers* using least squares method
return $best\_model$

known radius. The algorithm uses *MLESAC* (Maximum Likelihood Estimation Sample Consensus) to estimate the sphere's center from the segmented pointcloud. The function used for sphere fitting to a pointcloud is provided by *SACSegmentation* package from *Pointcloud Library*. This package provides a robust and fast implementation of the RANSAC algorithm for fitting geometric shapes to pointcloud data.

The algorithm 3.3 works as follows:

1. **Input**: RGB image I, Depth image D, predicted bounding box $B = (x, y, w, h)$, predicted class label $\hat{y}$
2. **Parameters**: Sphere radius range (r_{min}, r_{max}), tolerance threshold ϵ, maximum depth threshold d_{max}.

3. **Preprocessing**: Crop RGB and depth images based on object detection bounding box.
4. **Pointcloud Generation**: Create a 3D point cloud from the cropped depth image, filtering out points beyond a maximum depth threshold.
5. **Pointcloud Object Segmentation**: Apply a color mask based on the predicted class label to isolate points belonging to the object's surface. The color mask is expressed as a range of HSV values (hue, saturation, value). The color used in the mask is equal to the color of the colored ball, or a mix of red and yellow for the apples.
6. **RANSAC Sphere Fitting**: Iteratively fit spheres to random subsets of points, evaluating their fit based on inlier count and residual error using an *MLESAC* scoring metric. The best-fitting sphere model is updated based on the highest score.
7. **Model Refinement**: Refine the best-fitting sphere model using least squares optimization on the inliers. It will provide a more accurate estimate of the sphere's center and radius.
8. **Output**: Return the refined sphere model, representing the estimated center coordinates and estimated radius of the object, within the specified range.

3.11.2. Algorithm for Grasp Pose Estimation

Picking objects requires the software to know where the object is located in the environment and how to grasp it. Since explicit programming of how to grasp objects is quite difficult and not extensible to different objects, the algorithm neglects the grasping strategy (meaning the positioning of the fingers on the object) and focuses on the object's position estimation. The algorithm for computing the optimal grasping pose focuses solely on the object's center, which is computed from the object's perceived pointcloud data. This simplification allows the software to be fast in computing the grasping pose, even though the generated grasping poses are not optimal for all objects. The algorithm is based on the assumption that the object can be approximated as a sphere and that getting the end effector sufficiently close to the object's surface is good enough to grasp it. This simplified approach is acceptable because of the mechanical characteristics of the gripper. The silicone fingers are compliant with the shape of the object and account for small variations in shape.

The algorithm used for computing the grasping pose from the object's center is described in algorithm 3.4. The object's center is estimated using algorithm 3.3, using the depth and

RGB images. The algorithm generates a list of possible candidate grasping poses based on the object's center and the object's known radius. For each candidate, it checks whether an inverse kinematic solution exists for the end effector at the candidate grasping pose. If a solution exists, the candidate grasping pose is added to the list of feasible grasping poses. The algorithm then uses a heuristic to choose which candidate grasping pose to use, based on the list of feasible grasping poses. It selects the candidate at one fourth of the list's size. The first candidate in the list will correspond to grasping poses of the object from the top, while the last candidate will correspond to grasping poses from the bottom. This heuristic is used to choose a grasping pose from a position closer to the top because it is usually easier for the robotic arm to reach it, compared to the ones closer to the bottom, due to the kinematic constraints of the robotic arm placed on the mobile robot base.

If no feasible grasping poses are computed, the algorithm returns an error message, and the robot does not attempt to grasp the object. Otherwise, the robot moves to the selected grasping pose and attempts to grasp the object.

Compared to traditional rigid mechanical grippers, pneumatic grippers offer a unique advantage due to their flexible and deformable fingers. While the complexity of simulating the grasping process accurately increases, the necessity for such simulation is diminished. The inherent adaptability of soft gripper fingers allows them to conform to the object's shape and size without the need for precise control over finger deformation. This inherent compliance eliminates the risk of damaging objects during grasping, even without detailed simulation. The pneumatic gripper's ability to apply pressure and let the fingers adapt to the object simplifies the control process and enhances the gripper's versatility.

An important note is that the algorithm 3.4 does not guarantee finding a feasible grasping pose for a certain object, even if it exists. The algorithm is based on a simplified model of the object and of the end effector and it restricts the range of search for a grasping pose to the vertical plane passing through the object's center and the robotic arm's base. This simplification is necessary to make the algorithm fast and robust, but it also results in the impossibility of guaranteeing the successful computation of a feasible grasping pose for any object, so it comes at the expense of less reliability.

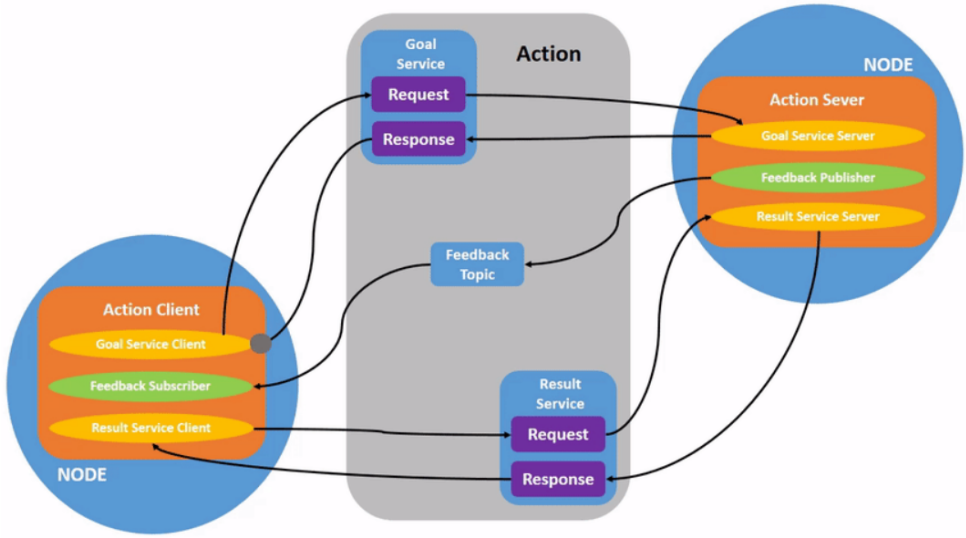


Figure 3.17: ROS2 Actions Client-Server Architecture

3.12. ROS2 Actions Client-Server Architecture for High-Level Tasks

Leveraging the ROS2 Actions client-server architecture made it possible to implement high-level tasks for the mobile manipulator robot. The **Actions architecture** is a powerful and flexible way to define and execute complex tasks in a distributed system, such as a robotic system composed of many nodes that need to coordinate and communicate to perform high-level behaviors. The Actions architecture is built on top of the ROS2 middleware, which provides a robust and reliable communication system. It allows for asynchronous communication between nodes, enabling the robot to perform multiple tasks concurrently and efficiently. The Actions architecture is based on the concept of goals, which represent the desired outcome of a task, and results, which represent the task's outcome.

The goal request is a non-blocking call sent by the client to the server, which executes the task asynchronously, generating feedback messages to inform the client about the task's progress. The server sends a result message to the client once the task is completed or aborted, indicating the outcome of the task. The client can monitor the task's progress with the feedback messages and cancel it if necessary, providing a robust and reliable way to manage high-level behaviors. The advantage of using the Actions architecture is the on-demand asynchronous execution of tasks, allowing for multiple tasks to be executed concurrently without blocking the system. This architecture decouples the task's definition from its execution, allowing for easy code reusability and extensibility across different

applications.

The architecture diagram of the ROS2 Actions client-server is shown in Figure 3.17. The client sends a goal message to the server, which processes the goal and generates feedback messages to inform the client about the task's progress. The server then sends a result message to the client once the task is completed. The client can also send a cancel message to the server to stop the task prematurely.

- **ROS2 Action Server**: the servers are nodes that handle the task execution, and they are responsible for processing the goals, generating feedback messages, and sending the result messages to the clients. The servers are implemented on top of the underlying algorithms and functionalities that perform the tasks, provided by a separate node inside the same package.
- **ROS2 Action Client**: the clients are nodes that send the goals to the servers, monitor the task's progress with the feedback messages, and receive the result messages once the task is completed. The clients are implemented as unique nodes that handle the sequence of actions to be executed, decoupled from the underlying algorithms and instructions that perform the tasks.

The ROS2 Actions architecture is employed to implement the demonstrations explained in the sections 4.2.4 and 4.3.5. The high-level tasks, such as moving the robot to a specific location, and manipulating objects or interacting with them, are implemented as Actions servers, which are called by the Actions clients to execute the tasks. The diagrams in Figures 4.6 and 4.17 show the architecture of the implementations of the demonstrations and how the clients and servers interact across multiple components, using the ROS2 middleware for communication.

Algorithm 3.4 Grasp Pose Estimation from Object's Barycenter

Require: $p = (x, y, z)$ estimated object's center in the camera frame

$C \leftarrow \emptyset$ ▷ Set of candidate grasping poses
$n_{candidates} \leftarrow 50$ ▷ Number of candidate grasping poses
$\theta_{min}, \theta_{max} \leftarrow -\pi, \pi/3$ ▷ Range of angles for the orientation of the end effector
$g \leftarrow 0.05$ ▷ Grasping distance from the object's center in meters
transform p into the robot's base frame
for θ in linspace$(\theta_{min}, \theta_{max}, n_{candidates})$ **do**
    $v_c = \frac{(x,y,z)}{||(x,y,z)||}$ ▷ Vector from the base to the object's center
    $v_l = -v_c \cdot g \cdot \cos(\theta)$ ▷ Longitudinal component v_l
    $p_v = \frac{(x,y,0)}{||(x,y)||}$
    $v_v = (v_c \times p_v) \times v_c$ ▷ Vertical component v_v
    $v_v = v_v \cdot g \cdot \sin(\theta)$
    $v_{grasp} = v_v + v_l + v_c$ ▷ Grasping vector v_{grasp}
    $v_{grasp,x} = \frac{-v_{grasp}}{||v_{grasp}||}$
    $plane_{xy} = (0, 0, 1)$
    $v_{grasp,y} = plane_{xy} \times v_{grasp,x}$
    $v_{grasp,z} = v_{grasp,x} \times v_{grasp,y}$
    $rot_{grasp} \leftarrow$ Rotation matrix $[v_{grasp,x}, v_{grasp,y}, v_{grasp,z}]$
    $q_{grasp} \leftarrow$ Quaternion representation of rot_{grasp}
    $q_{grasp} = \frac{q_{grasp}}{||q_{grasp}||}$ ▷ normalize quaternion
    grasping candidate position $\leftarrow v_{grasp}$
    grasping candidate orientation $\leftarrow q_{grasp}$
    if $\exists$ I-K solution for (v_{grasp}, q_{grasp}) **then**
        $C \leftarrow C \cup (v_{grasp}, q_{grasp})$ ▷ add the candidate grasping pose to the list
    end if
end for
$size \leftarrow |C|$ ▷ get the size of the list of candidate grasping poses
if $size \geq 1$ **then**
    $i = size \cdot 1/4$ ▷ get the candidate in position 1/4 of the list
else
    return *No feasible grasping poses found*
end if
return $C[i]$ ▷ return the selected candidate grasping pose

4 | Experimental Setups and Demonstrations

This chapter will describe the experiments conducted to demonstrate the capabilities of the mobile manipulation robotic system. The experiments consist of three demonstrations named "ArUco Follower", "Button Presser", and "Object Picking". These demos are meant to showcase the capabilities of the robot in various tasks such as following a marker, pressing buttons, and picking objects. The first demo is a preview of the robotic arm's autonomous control software. The second demo showcases the capabilities of the entire system in performing high-level tasks in industrial environments, such as pressing buttons on an industrial control panel. The third demo is meant to showcase the mobile manipulation capabilities in an agricultural environment, such as picking fruits.

The experiments are conducted in a controlled realistic environment to test the robustness and reliability of the system. The demos have been tested inside the Artificial Intelligence and Robotics Laboratory at Politecnico di Milano. The laboratory has enough space to allow testing the efficiency of the autonomous navigation software while testing also the obstacle avoidance algorithm in a cluttered and changing environment.

The objectives of these demonstrations are to evaluate the robot's performance within the various software components and how well they integrate with the hardware components. The experiments will also highlight the challenges faced during the development and testing of the robotic system and how they were overcome. The demos are meant to be a proof of concept of the robotic system's capabilities in simulated scenarios that are as close as possible to more realistic environments. They are also meant to show the potential of mobile manipulation robots in performing various complex tasks that are currently performed by humans. The objective is not to replace humans but to assist them in performing tasks that are dangerous, repetitive, or require high precision.

4.1. ArUco Follower Demo

The ArUco Follower demo is a demonstration of the robotic arm's autonomous control software. The demo consists of the robotic arm following an ArUco marker with the end effector. The demo showcases the motion planning and execution libraries with MoveIt2 and the integration of the ArUco marker detection and pose estimation algorithms with the robotic arm's control software. The demo tests the autonomous control software of the robotic arm in tracking and following a moving target. Figure 4.1 shows the demo in execution, with *MountV1* as end effector (but without the cylinder presser). The cardboard shown to the camera displays an ArUco marker, tracked by the robotic arm.

In this demo, the end effector, equipped with the camera, will move in a position and orientation that points toward the center of the ArUco marker. The algorithm for **tracking the marker** computes the end effector's target position such that the arm can always reach it, and the orientation of the end effector is always pointing towards the marker. The position of the end effector will be exactly the marker's position if the marker is sufficiently close to the arm, otherwise, the end effector will be positioned to point the camera toward the marker. If the marker is not visible, the end effector will remain in the last known position until the marker is visible again. If the marker moves to a different position in the camera frame and stays visible, the algorithm will compute the new target pose and the end effector will move to it. The computation of the end effector's target pose is based on geometrical calculations from the marker's position in the camera frame. The marker's pose is transformed in the robot arm's base frame, and the target pose is computed based on the robotic arm's workspace. The target pose is then sent to the motion planning and execution libraries to compute and execute the trajectory to reach the target pose.

A demonstration incorporating the MoveIt2 Servo node for real-time end-effector control via ArUco marker pose targeting was implemented and tested. However, due to the ongoing development of MoveIt2 Servo at the time of writing and its associated stability limitations, the resulting trajectories exhibited undesirable jerkiness and did not consistently achieve the target pose within a short timeframe. Additionally, the requirement for precise PID tuning of the closed-loop joint trajectory controller presented further challenges in achieving optimal real-time performance. Given these constraints, the decision was made to utilize the standard MoveIt2 planning and execution libraries for the final demonstration, ensuring greater reliability and smoother operation.

Figure 4.1: ArUco Follower demo during execution. The cardboard shows an ArUco marker.

4.2. Button Presser Demo

The "Button Presser" demo is a complex demonstration that showcases the potential of mobile manipulation robots in performing high-level tasks in industrial environments. The demo consists of the robotic arm pressing buttons on an industrial control panel, in an autonomous way. The objective is to show the capabilities of the robot in performing tasks that are currently performed by humans. The demo is meant to be a proof of concept of the robotic system's integration of multiple software components and the orchestration of the robotic arm and mobile base to interact with the environment without human intervention. The demo was developed in the context of a larger project in collaboration with a company. The aim of the project is to use mobile manipulation robots in industrial plants, for monitoring sensors and various equipment, and intervening in case of emergency, while avoiding human intervention in dangerous environments.

The purpose is to demonstrate the feasibility of the mobile manipulation robot in interacting with the control panel autonomously and effectively. It was also important to demonstrate the system's reliability and robustness in performing such a complex task, in terms of accuracy in pressing buttons and the time taken to press all the buttons.

4.2.1. Buttons Setup Box and End Effector Setups

For this demo, the *MountV1* was employed, mounted on the robotic arm's end effector. The *MountV1* is a custom-designed 3D-printed mount attached to the cobot's flange, which allows the installation of the stereo camera and the cylinder presser. The cylinder presser is a cylinder-shaped tool used for pressing buttons. In *MountV1* the camera is placed in front of the flange, and this resulted in a reduced field of view of the camera. Despite this camera position, the camera was able to detect the ArUco markers on the control panel and the buttons to be pressed. Since the *MountV1* proved to be quite effective in the demo, the only change made to this version was to shorten the cylinder presser to prevent the mobility of the end effector from being reduced due to the length of the tool.

Further in the development of the demo, the end effector configuration was switched to *MountV2*, which is the next version of *MountV1*. The biggest improvement of *MountV2* is the camera's position, which is placed on the wrist of the cobot, allowing a greater field of view of the camera. This proved more effective in finding the ArUco markers on the control panel from a closer distance.

Using *MountV1*, the end effector was slipping during the execution of the linear trajectories, due to the roundness of the button caps. This is due to the low friction between the digital button and the round button cap, having both surfaces made of plastic and a small contact area. The slipping of the end effector caused the robotic arm to fail to press the buttons effectively. To overcome this problem, the end effector was replaced with *MountV2*, which has a vacuum suction cap on its tip, instead of having a cylinder presser. The end effector presents a vacuum suction capo used to press the buttons effectively. This solution is more effective in pressing the buttons, thanks to the greater friction between the plastic button cap and the silicon suction cup. The vacuum suction cap presses the buttons reliably, with less slipping, thanks also to the greater contact area between the cap and the button.

The Realsense camera was used only for ArUco markers detection and pose estimation. So the software component related to the perception of the control panel didn't make use of the infrared cameras for depth estimation. The position of the markers is in fact computed using the RGB image as input and the equations for 3D camera projection on a 2D matrix for the pose estimation.

The control panel is a custom-designed box with three buttons of different sizes and seven ArUco markers (having dictionary 4x4) placed around the buttons. Figure 4.2 shows the control panel. The control panel is small and portable and can be placed in any position

Figure 4.2: Control panel with 3 buttons and ArUco markers.

and orientation to test the robustness of the system. The control panel has also three different lights, one for each button, that indicate whenever a button is pressed.

The reason for having multiple ArUco markers instead of just one or three for the three buttons, is to have a more robust and reliable detection of the buttons' positions. Having multiple markers allows the system to detect the buttons' orientation in a more precise way that is also robust to noise. The markers are placed around the buttons in a way that the plane estimation algorithm can compute the plane of the markers effectively, relying on a greater number of points to estimate the plane.

4.2.2. End Effector Positioning and Linear Trajectories

The implementation of the algorithms for pressing buttons and planning the trajectories is handled inside ROS2 nodes. One ROS2 node subscribes to the ArUco markers detection topic containing the estimated markers' poses and their IDs. Given the markers' poses, and the relative positioning of the markers with respect to the buttons, the node computes the position of the buttons in the robot's base frame. The node also computes the orientation of the end effector, such that the vector from the arm's flange and the end effector tip faces the plane of the markers orthogonally, and this is needed to compute the orientation required to press the buttons.

The **algorithm** for computing the sequence of target poses that the end effector must

reach uses the **estimated position and orientation of the buttons**. For each button, the algorithm computes the pose immediately above the button, meaning that the end effector faces the button orthogonally from a fixed distance. Then the algorithm generates the linear path that the end effector must follow to reach the pose where the button is pressed. The linear path is then reversed to get the path required to lift the end effector from the button in order to release it and return to the starting point above the button. This sequence of target poses computation is repeated for each button on the control panel. The ROS2 node uses these target poses and linear paths to plan and generate the trajectories for the arm to reach the target poses and follow the linear paths. Once the trajectory plans are generated, they are executed.

The linear motion trajectory generation is the most challenging part of the trajectory planner, because the algorithm that computes the trajectories must take into account several constraints, such as the static collisions with external bodies and the robotic arm's self-collision mapping. There are two main methods to generate a linear trajectory for the end effector in cartesian space:

- **Cartesian Linear Motion Planning generation via a sequence of waypoints**: this method generates a sequence of pose waypoints (sequence of positions and orientation) that the end effector must follow to reach the target pose. The trajectory is generated by interpolating the sequence of waypoints in space and time. The trajectory planner generates a trajectory that follows the waypoints with maximum deviation defined as the *end effector jump*, which is the maximum total deviation in joint space that each joint can have from the average joints' positions. This parameter controls how much the joints can move from their average configuration during the execution of the linear trajectory. Setting the end effector jump to zero means that the joints will not move from their average configuration, and the end effector will follow the waypoints exactly.
- **Constrained Cartesian Motion Planning**: this method creates a motion plan request with the addition of position and orientation constraints on the end effector that must be respected while moving toward the final pose. The constraints are defined as a position box constraint, meaning that the end effector must stay within a box (rectangular cuboid) in the cartesian space, and an orientation constraint, defined as a quaternion representing the orientation and the maximum angle of deviation from the orientation axis defined by the quaternion.

Both methods were tested and implemented successfully to be used with the Igus Rebel cobot. The constrained cartesian motion planning method proved to be unreliable due

to the high probability of not being able to generate a valid trajectory. Adding only one constraint to the motion plan made the trajectory planner able to find a solution most of the time. But adding two constraints, one for the position and one for the orientation, resulted in failed trajectory generation almost every time. This limitation is caused by the physical characteristics of the robotic arm. To successfully generate a constrained cartesian motion plan, the robotic arm must have 7 DoF, while the Igus Rebel cobot has only 6 DoF. This makes the constrained method not suitable for this robotic arm. The reason behind this is that constraining a motion plan with two constraints for a 6-DoF robotic arm results in an overdetermined system of equations, and the trajectory planner is not able to find a feasible solution, because the constraints add more equations than the number of DoF of the robotic arm, resulting in a system of equations with no feasible solution.

Due to this limitation, only the cartesian linear motion planning method was used to generate linear trajectories following a sequence of waypoints. The end effector's jump was set to zero to make the end effector follow the waypoints exactly, to avoid random jumps in the cobot's configuration. The linear trajectories were generated successfully and executed correctly most of the time. Sometimes the trajectory planner failed to find a solution, even when attempting to create a trajectory plan multiple times. This was due to the complexity of the environment that the planner must take into account. After many attempts with different motion planners, such as OMPL, STOMP, and the Pilz industrial motion planner, the one that proved to be the most reliable and robust for generating linear trajectories was the Pilz industrial motion planner.

The **reliability of the trajectory planner** in generating linear trajectories for the end effector was a crucial aspect of the demo and posed a challenge during the development and testing of the demo. The planner failed to find a feasible solution frequently. In order to increase the probability of finding a feasible solution, an increase in the tolerances of the planner, such as the maximum deviation of the end effector from the waypoints, and an increase in the number of attempts to generate a trajectory plan made the planner more reliable.

4.2.3. Mobile Button Presser Demo

The "mobile button presser" demo is a complex demonstration to showcase the mobile manipulation robot's capabilities in pressing buttons on an industrial control panel autonomously. The demo consists of the mobile manipulation robot navigating to the control panel's location, detecting the control panel, and pressing the buttons in a predefined se-

quence.

The **complete demo**, illustrated in the flowchart in Figure 4.3, consists of the following steps:

1. The mobile manipulation robot starts from a random location and does not know where the control panel is located.
2. The robotic arm performs an inspection routine using the camera mounted on the end effector to search for a specific ArUco marker, which signals where the control panel is located.
3. Once the specified marker is detected, and its distance is computed, the algorithm computes the position of the control panel in the map frame of reference.
4. The robotic arm parks itself to not obscure the field of view of the LiDAR, which is essential for localization and autonomous navigation.
5. The mobile base navigates to the control panel's location autonomously, while avoiding obstacles in the environment.
6. The mobile robot parks in front of the control panel at a fixed distance, facing the opposite side of the control panel, so that the robotic arm can move freely without colliding with the mobile base and reach the buttons. Figure 4.4 shows this stage of execution of the demo.
7. The robotic arm uses the onboard camera mounted on the end effector to inspect the close environment to search for the ArUco markers on the control panel, indicating the locations of the buttons to be pressed.
8. Once the camera detects the ArUco markers, the algorithm computes the position and orientation of the buttons placed on the panel in the robot's base frame.
9. For each button, it finds the end effector target poses in the cartesian space required to press the buttons. It also creates the linear paths for the end effector necessary to press and release the buttons.
10. The robotic arm executes the computed trajectories, and the end effector presses the buttons in the predefined sequence. Figure 4.5 shows the end effector pressing one of the buttons, at this stage of execution of the demo.
11. The control panel lights up the corresponding light for each pressed button.

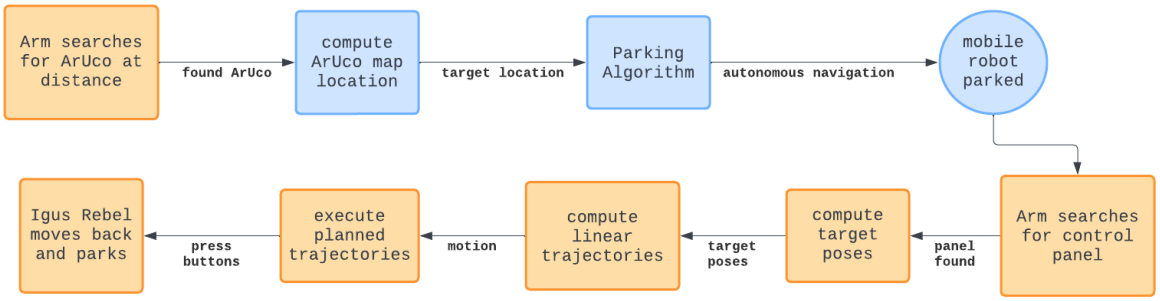


Figure 4.3: Flowchart of the Mobile Button Presser demo execution

4.2.4. Mobile Button Presser Demo Implementation and Architecture

This demonstration is handled by one ROS2 node (action client) that orchestrates the actions of the mobile base and the robotic arm. The client node sends the goal to three different action servers:

- **Parking Action Server**: this server is responsible for computing the parking algorithm, given the position of the control panel in the map frame. After the parking pose is computed, the server sends a navigation goal to the nav2 stack to navigate autonomously to the parking pose. The action returns the result containing information about how close the robot is to the parking pose when the autonomous navigation is completed.
- **Button Finder Action Server**: this server is responsible for searching for a specific ArUco marker in the room. The server executes a "searching movement", which is a sequence of predefined poses that the robotic arm must follow to search for the marker. The server also executes all trajectories until the marker is detected. The server returns once the marker is detected, and the distance from the cobot to the marker is computed.
- **Button Presser Action Server**: this server is responsible for pressing the buttons on the control panel. It first computes the "searching movement" to search for the ArUco markers on the control panel. Once the markers are detected and the control panel is located, the server computes the target poses and linear paths for pressing the buttons. The server then executes the trajectories to press the buttons in the predefined sequence. The server keeps track of how many trajectories have been planned and successfully executed, with respect to the total number of trajectories to be executed. The server returns the success rate of the executed trajectories.

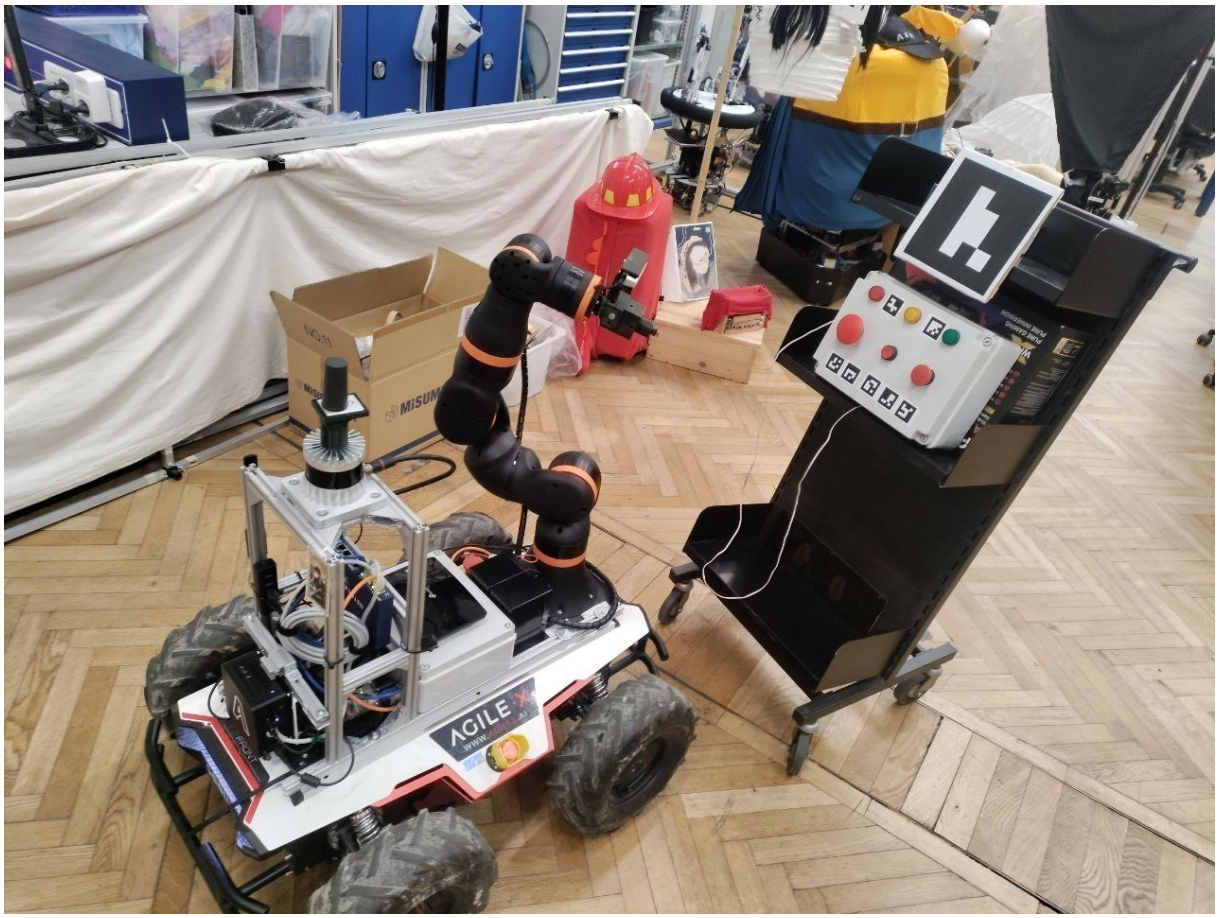


Figure 4.4: Mobile Button Presser Demo in execution

The server terminates its execution once all the buttons have been pressed.

The architecture of the whole system is composed of multiple software components that interact with each other using ROS2 Actions and Topics. The architecture, shown in Figure 4.6, is designed to be modular and scalable, to allow the integration of new components and functionalities in the future. The architecture is composed of the following macro components:

- **Multi-ArUco Plane Detection and Marker Pose Estimation**: this component is responsible for detecting the ArUco markers in the camera's field of view and estimating the markers' poses. The component relies on the Multi-ArUco Plane Detection algorithm to detect the markers and correct their estimated poses using the plane estimation algorithm. This algorithm is explained in section 3.9. The component publishes the markers' corrected poses on a ROS2 topic.
- **MoveIt2 Servers**: this component includes the MoveIt2 Planning and Execution functionalities and the two action servers for finding the parking pose and pressing

Figure 4.5: Cobot pressing buttons on the control panel using the *MountV1* end effector

the buttons. The MoveIt2 servers are responsible also for computing and executing the linear trajectories generated by the trajectory planner, required to press the buttons. The demo execution pipeline is handled within threads executed on-demand by the MoveIt2 servers, given the goal sent by the client node.

- **Nav2 Servers**: this component is responsible for the autonomous navigation of the mobile base using Nav2. It includes also the action server for computing the parking algorithm and the action client exploiting Nav2 Commander APIs to send navigation goals to the mobile base. The structure of the parking algorithm is explained in section 3.8, while the autonomous navigation integration with ROS2 is explained in section 3.7.

- **Client Node**: this component is responsible for orchestrating the actions of the mobile base and the robotic arm. The client node is a ROS2 node that integrates the action clients for the parking, button finder, and button presser action servers. The orchestration of the actions is handled by different threads that are executed

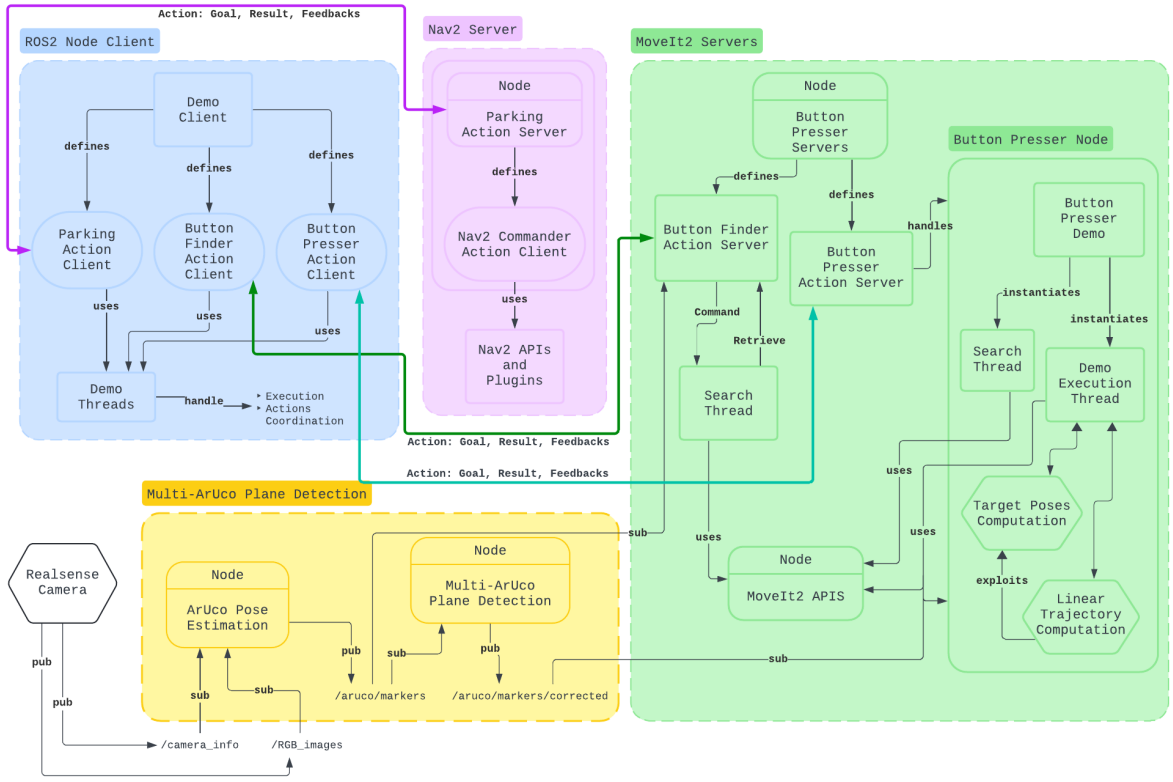


Figure 4.6: Architecture diagram of the Mobile Button Presser demo

on-demand by the client node, given the goal sent by the user. Multiple threads are present since each one is responsible for handling different tests and demo versions.

4.3. Object Picking Demo

The "Object Picking" demo is a more complex demonstration that showcases the mobile manipulation robot's capabilities in picking objects from the environment autonomously. The demo consists of the robotic arm picking colored balls or apples from an artificial plant, autonomously. The objective is to demonstrate the robotic system's integration of multiple software components and the orchestration of the robotic arm and mobile base to interact and grasp objects of different shapes and sizes, using a soft robotic hand gripper. The soft gripper enables the robotic arm to grasp fragile objects without damaging them, and it is also able to grasp objects of different shapes and sizes, thanks to the flexibility and adaptability of the silicone fingers. For this demo, the *MountV2* is used, since it is the only one that supports the soft gripper.

The colored balls used in the demo are small plastic balls of different colors. The balls are used as a test ground for the grasping capabilities of the soft gripper since they are small

enough to be easy to grasp and the plastic material enables high friction between the fingers and the ball. Instead, the artificial apples are a little more challenging to grasp, because of their irregular shape. The artificial apples are used to simulate a realistic and more complex scenario, where the objects to be picked are closer to objects appearing in real-world environments, such as fruits on trees.

The **objective of the demo** is to apply the mobile manipulation robot in an agricultural environment, which is often more challenging and irregular than in industrial environments. The demo is meant to be a proof of concept of the autonomous control of a robot to pick and place objects in a realistic environment, such as a plantation.

For this demo, three different versions were developed, each with different levels of complexity and challenges. The first version is used as a test for the algorithm for object pose estimation and the grasping capabilities of the soft gripper. The second version is a picking task where the manipulator picks up objects and drops them into a basket positioned on the mobile base, using the neural network for object detection and perception algorithms. The third version is used to test the robot's navigation and obstacle avoidance capabilities in conjunction with the object detection neural network and the perception algorithms. In this version, the task is pick and place, where the placement of the objects is in a predefined location, physically separated from the picking location.

4.3.1. Plants, Colored Balls, and Apples Setup

The first setup for testing and demonstrating the demo uses a small artificial plant with colored balls attached to it, as shown in Figure 4.9. The second more realistic setup uses a large flat surface with artificial apples placed on it, simulating a more realistic espalier apple tree, as shown in Figure 4.7a. This sort of tree is used in agriculture to grow apples flatly and vertically, to save space, and to make the apples more accessible for picking. Figure 4.7b shows an espalier apple tree, grown in a yard. The apples are placed on the tree at different heights and distances from each other, to simulate a more realistic scenario where the apples are not all at the same height and distance from the robot. The apples are also placed in a way that the robot must move around the tree to reach all the apples, and this is meant to test the robot's navigation and obstacle avoidance capabilities.

The colored balls and apples are attached to the plant with a nylon string. At the extremity of the string, a small magnet is secured using two different methods:

- **Version 1 (Hot Glue)**: the magnet is attached to the string with hot glue. For the apples, the magnet is attracted to another magnet placed on a screw that is screwed

(a) Simulated espalier apple tree (b) Real espalier apple tree

Figure 4.7: Espalier apple trees

Figure 4.8: Magnetic attachment of an apple to the nylon string

into the plastic apple. In the case of the colored balls, the magnet is attracted to another magnet glued onto the ball. To enhance robustness, the balls have two magnets: one inside and one glued onto the external surface. Figure 4.8 shows the magnetic attachment of an apple to the nylon string.

- **Version 2 (3D Printed Magnetic Hook)**: the second iteration replaces the hot glue with a 3D printed hook. This version is designed exclusively for the apples since they can accommodate the screw where the magnet is attached. The magnet is securely placed within the hook, without the need for hot glue, and the nylon string is wrapped around it, as shown in Figure 4.10a. This eliminates the need for hot glue, which can be unreliable when bonding to magnets, resulting in a more

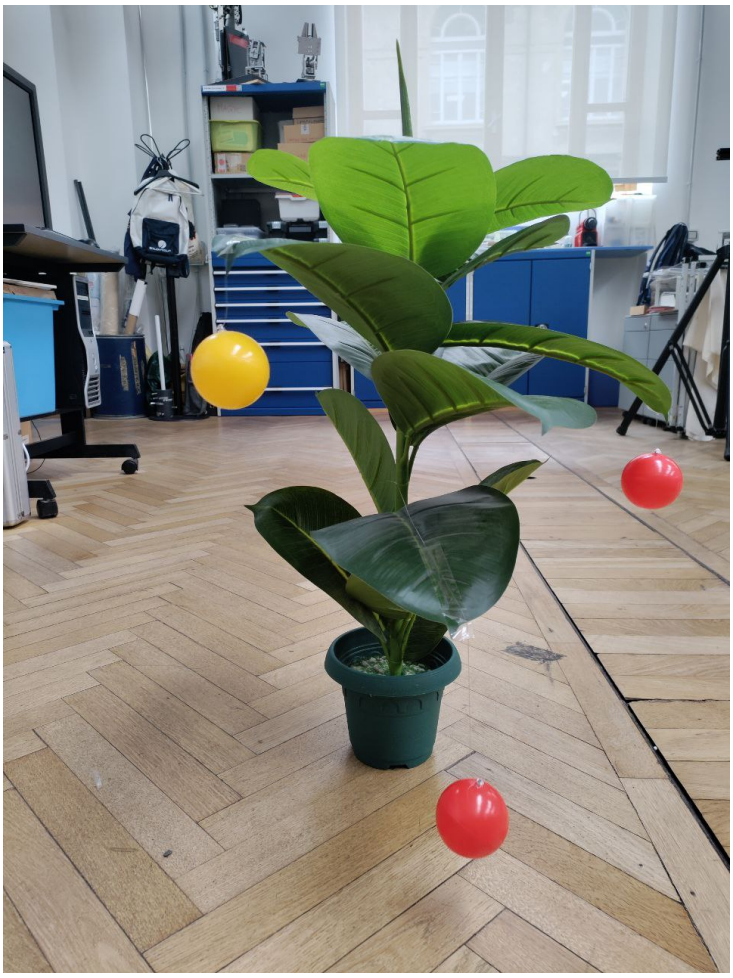
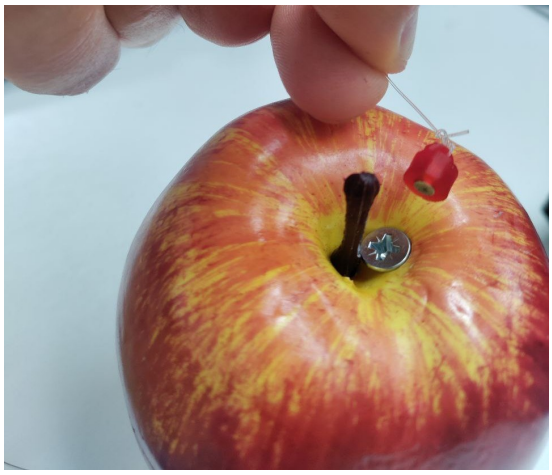


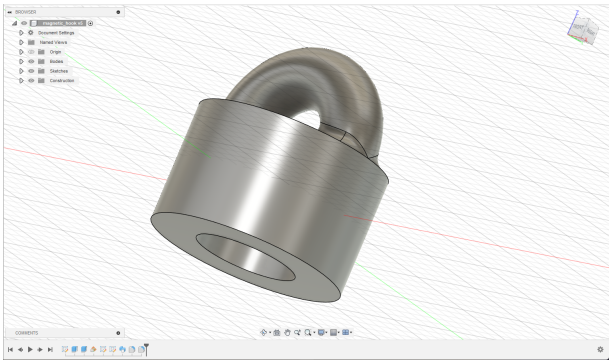
Figure 4.9: Simulated plant with colored balls

robust attachment. The 3D-printed hook is small and lightweight, maintaining the overall design's unobtrusive nature, and ensuring very fast printing times and quick installation. The 3D design is shown in Figure 4.10b. It is printed in white PLA (Polylactic Acid), making it strong and durable, with a smooth surface finish.

Both versions are effective in attaching the objects to the plant and making them detachable without excessive force, allowing the robotic arm to easily detach them without straining the motors. The magnets are small and lightweight, minimally impacting the object's weight and shape. The nylon string is thin and transparent, remaining invisible to the camera and preserving the object's appearance. The key advantage of the 3D printed hook (Version 2) is its enhanced reliability and ease of assembly compared to the hot glue method (Version 1).



(a) Magnetic 3D-printed hook attached to an apple



(b) Magnetic hook 3D design

Figure 4.10: Magnetic hook for attaching apples

4.3.2. DemoV1 with Manual User Input

This first version of the demo (*DemoV1*) is a test for the grasping capabilities of the soft gripper and the autonomous control of the robotic arm, without any advanced perception algorithm implemented. This demo is also meant to be used when the robotic arm is fixed in a specific location. The demo consists of the robotic arm picking colored balls from the artificial plant, with manual user input for selecting the target object to be picked. The user selects the target object by clicking on the object in the camera's field of view, and the robotic arm moves to the target object's position and grasps it. The user can select only one object at a time.

When starting the demo, an image window appears on the screen, showing the camera's RGB image feed. The user can **click on the image** and the coordinates of the mouse click are recorded, and broadcast on a ROS2 topic. The object that the user clicks on is assumed that it can be approximated as a sphere. The algorithm for object position estimation computes the object's position in the camera frame using the pixel coordinates, the camera's intrinsic parameters, and the depth image from the infrared camera. The result is a (x, y, z) position vector of the visible point clicked by the user in the camera frame. This is the point of the pointcloud on the surface of the object, which roughly corresponds to the object's visible surface center. By projecting a ray from the camera's optical sensor to the computed point, the algorithm computes the object's approximate center in the camera frame. The center is then transformed into the robot's base frame, and used as input to the grasping pose estimation algorithm to compute the target pose

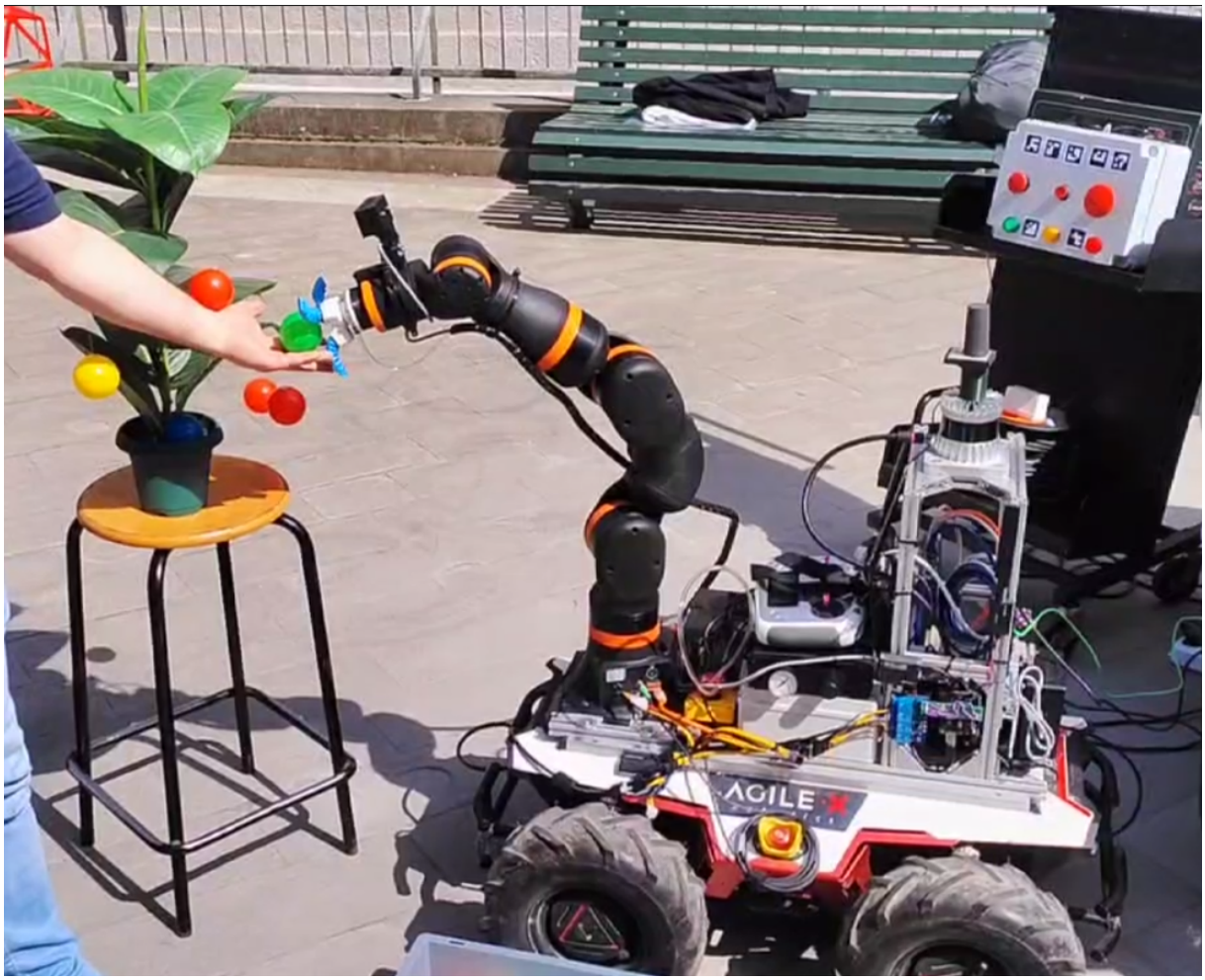


Figure 4.11: Cobot grasping a colored ball from the hand of a volunteer

for the end effector required to position the robot where the object can be grasped.

Once the grasping pose is computed, the robot moves to the target pose and the end effector grasps the object. The robot then moves to a standard pre-defined position and drops the object, assuming that a basket is placed exactly underneath the robot's end effector in the dropping position. The robot then returns to the starting position and waits for the user to select the next object to be picked. Figure 4.11 shows this demo during its execution, using colored balls as objects to be grasped. The demo worked well both with the balls attached to the simulated tree and with the ball in the hands of human volunteers, as in Figure 4.11.

One of the issues faced during the development of the *DemoV1* for ball picking was the **reflectivity** of the balls in the **infrared spectrum** in the presence of strong sunlight. The balls were reflecting the sunlight and the infrared camera was not able to compute the depth in the spots of the balls' surface with the highest reflectivity. This resulted in

the balls' surface appearing as holes in the depth image. So if the user clicks exactly in the spot where the ball is very reflective, the depth value computed would be either wrong or infinite and would result in the algorithm crashing or computing an infeasible grasping pose. This effect was more pronounced with yellow balls. The immediate solution is to click in the least reflective spots, but this is a temporary workaround and not a definitive solution to the problem. *DemoV2* offers a more robust solution to this problem.

4.3.3. DemoV2 with Object Detection Neural Network

In demo version 2 (*DemoV2*), the demo does not require any user input for selecting the object to be picked. Instead, it relies on a neural network for detecting the objects in the camera's field of view. The neural network used for the object detection task is described in section 3.10. The YOLOv8 neural network is trained to detect the colored balls and the apples, and it outputs the object's bounding box and class label with the confidence score. The predicted bounding box is used as the starting point to obtain a pointcloud containing the points on the entire object's surface. The pointcloud is then used to compute the object's center in the camera frame, and the grasping pose is computed as in the previous version of the demo. The algorithm 3.3 assumes that the object can be approximated as an ideal sphere of known radius. This algorithm is very robust for the colored balls, since they are spheres, and less robust and reliable for the apples, because of their irregular shape. The algorithm is able to compute the object's center even from a small portion of the object's surface, and this is a key feature of the algorithm because it does not require the entire object to be visible in the camera's field of view.

DemoV2 solved the infrared reflectivity problem because it relies on the entire pointcloud of the object's surface, and not just on a single point. This means that even if there is a hole in the pointcloud, the algorithm 3.3 is still able to compute the object's center correctly (even though likely less precisely). In the cases where the pointcloud presents points with wrong depth values, the algorithm is still able to find the correct solution, as RANSAC is robust to outliers and noise in the data.

DemoV2 is programmed to take into account the **slow performance of the neural network** during inference on the CPU. The neural network runs on the CPU and it is not able to process the images fast enough to provide real-time object detection. The neural network takes around 0.5 seconds to process a single image on the Intel NUC onboard computer, when other algorithms and ROS2 nodes are running in parallel, especially during the execution of the complete demos. Since the task is not time-critical, the neural network can run in the background and provide the predictions as soon as they are

available. The code is implemented with multiple parallel threads that can handle and synchronize the data flow between the neural network and the other algorithms. The code uses the predictions from the neural network coupled with the depth and RGB images taken at the moment of inference.

4.3.4. Mobile Fruit Picking Demos

The **complete demo** is based on DemoV2 and extended to include the mobile base's navigation and obstacle avoidance capabilities. The demo consists of the mobile manipulation robot navigating to the apple tree's location, detecting the apples using the neural network, and picking the apples in a predefined sequence. Figure 4.12 shows different stages of the execution of this demo, where the robot is picking an apple from the tree. The robot then drops the apples into a basket. There are two versions of the complete demo:

1. **DemoV2 with pick and place on robot**: the robot picks the apples from the tree and drops them on a basket that is positioned on the mobile robot base. Figure 4.13 shows the basket positioned on the mobile robot base, with the apples collected from the tree using the described sequence of actions. The robot does not need to navigate to a separate location to drop the apples, but it is sufficient for the robotic arm to move to a predefined position so that the end effector is immediately above the basket. This version is simpler and less challenging than the first one, but it was implemented to have a demo that is faster to execute and more realistic of a real-world scenario where the robot would operate in an agricultural field.

2. **DemoV2 with pick and place**: the robot picks the apples from the tree and navigates to a predefined location to drop the apples in a basket. The basket is located in a position separate from the tree, and the robot must navigate to the basket's location to drop the apples. The robot must avoid obstacles in the environment while navigating to the basket's location. This version is more complex and challenging because the robot must navigate to two different locations. Once the mobile robot parks next to the basket's location, the robotic arm searches with the camera for an ArUco marker, which signals the position of the basket. The basket with the ArUco signal marker is shown in Figure 4.14. Then an algorithm finds a feasible pose for the end effector such that it gets placed immediately above the basket, where it can release the grip on the grasped object and drop it, as shown in Figure 4.15. The sequence of actions executed for this demo is shown in the flowchart in Figure 4.16.

The **object picking routine**, explained in algorithm 4.1 is the routine that the robot follows to pick up objects from a plant. This algorithm is suitable for both apples and colored balls since they are both approximated as spheres. This routine defines the sequence of actions to be executed by the robot to search for an object to pick, grasp, and drop in a basket, which can be located either on the robot itself or in a separate location far from the tree. The picking routine differs based on the version of the demo. When the robot must place the object in the basket positioned on the mobile robot itself, or just next to the robot itself, the robotic arm moves to the dropping pose and drops the object, as explained in the algorithm 4.1. Instead, when the robot must pick and place in separate locations, the robotic arm moves to its parking position and does not release the object until the mobile base has reached the dropping location and the camera recognizes the basket using the ArUco marker.

Algorithm 4.1 **Object Picking Routine**

Require: Neural Network for object detection NN
Require: Depth Image feed D
max object distance $z_{max} = 1.5$ in meters
Generate a list of pose waypoints W
for each waypoint w in W **do**
    plan and execute trajectory from current pose to w
    check if there are objects detected by NN
    for each object o detected by NN **do**
        compute **segmented pointcloud** using algorithm 3.3
        compute centroid point P from segmented pointcloud
        compute distance z from centroid P
        get confidence score c of object o from vector C
        compute **priority score** $s = c \cdot 1/4 + (z_{max} - z)/z_{max} \cdot 3/4$
    end for
    Choose object o having highest score s
end for
Estimate **object's center** using algorithm 3.3
Estimate **grasping pose** G using algorithm 3.4
if $\exists$ feasible pose G **then**
    compute pre-grasping pose G'
    plan and execute trajectory to G'
    open the gripper
    plan and execute linear trajectory from G' to G
    close the gripper to grasp the object
    plan and execute linear trajectory from G to G'
    move back to waypoint w
    move to dropping pose d
    open the gripper to drop the object
    turn off the gripper
else
    move to next waypoint
end if

The algorithm 4.1 is the main algorithm that the robot follows to pick objects from the tree. The algorithm is executed for each waypoint in the list of poses that the robot must follow to search for objects to pick. If there are multiple objects detected by the neural

network in a unique searching pose, the algorithm chooses the object with the highest priority score, which is computed based on the object's distance from the robot and the confidence score of the object's detection. The priority score is used to choose the object that is closest to the robot and has the highest confidence score. The algorithm then computes the object's center and the grasping pose, and it executes the trajectory to grasp the object. The robot then moves to the dropping pose and drops the object into the basket.

4.3.5. Mobile Fruit Picking Demo Implementation and Architecture

This complete demo requires the **integration of multiple software components**, such as the neural network for object detection, the perception algorithms for computing the object's center, the grasping pose estimation algorithm, the trajectory planning algorithms, and the navigation and obstacle avoidance algorithms. The demo is a complex orchestration of multiple algorithms wrapped in ROS2 action servers. The action servers are responsible for executing the actions and the algorithms required to pick the objects. The action client node is responsible for orchestrating the actions of the mobile base and the robotic arm, and for sending the goals to the action servers. The action client node also handles the sequence of actions in case of any failure of any software component and logs the feedback data received from the action servers. The ROS2 action servers used in the demo are:

- **Parking Action Server**: this server is the same as the one used for the other complete demo. The only difference with the other demo is that the location from which the robot must compute the parking position is not given by the result of another action, but is read from a configuration file. Both poses where to pick objects and the dropping location are hardcoded in a configuration file for simplicity and ease of use.
- **Picking Action Server**: this server is responsible for executing the first part of the object-picking routine. It is responsible for searching for objects to pick and checking whether there exists a feasible grasping pose for the objects in the camera frame. If it finds a feasible grasping pose for an object, it plans and executes the trajectories to grasp the object. The server returns the result of the planning and execution of the trajectories. It also provides logging feedback about the objects detected and the grasping poses computed.
- **Dropping Action Server**: this server is responsible for executing the second part

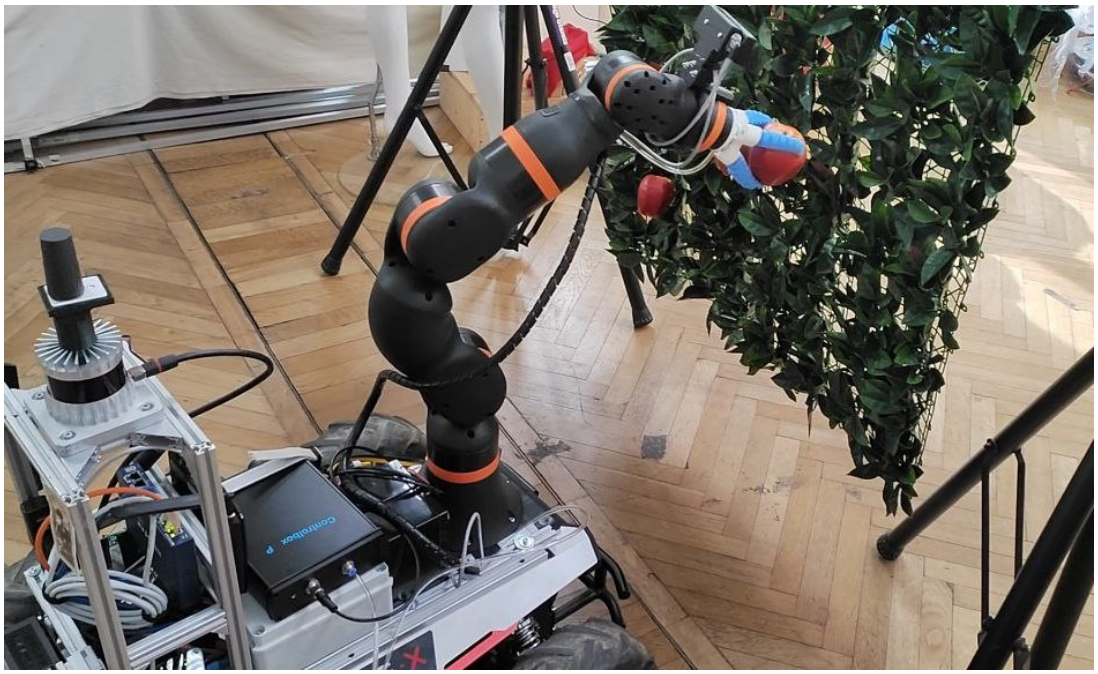
of the object-picking routine. It is responsible for moving the robot to the dropping location and dropping the object into the basket. This server is called only if the dropping location is separate from the picking location. It searches for an ArUco marker signaling the dropping location and computes a feasible pose for the end effector to drop the object. Then it moves to that pose and drops the object. The server returns the result of the execution of the trajectories and the dropping action. It also provides logging feedback about finding the ArUco signaling the dropping location.

The architecture of the whole system is composed of multiple software components that interact with each other using ROS2 Actions, Services, and Topics. The architecture, shown in Figure 4.17, is designed to be modular and scalable, to allow the integration of new components and functionalities in the future. The architecture is composed of the following macro components:

- **ROS2-Control Gripper Interface**: this component is responsible for actuating the soft gripper, using a ROS2 service as an interface to the gripper's ROS2 hardware interface. The hardware interface used has a structure analogous to the one used for the control of the robotic arm, as described in section 3.1. The ROS2 service is handled by the MoveIt2 APIs node, which provides a convenient interface to couple the trajectory planning and execution with the gripper's control, necessary for the manipulation tasks.
- **MoveIt2 Servers**: this component includes the MoveIt2 Planning and Execution functionalities and the two action servers for finding the parking pose and pressing the buttons. The MoveIt2 servers are responsible also for computing and executing the linear trajectories generated by the trajectory planner, required for grasping the detected objects. These servers handle also the perception algorithms for computing the object's center and the grasping pose, as described in the section 3.11. They use the YOLOv8 neural network predictions to execute the pipeline of actions required to pick the objects. The demo execution pipeline is handled within threads executed on-demand by the MoveIt2 servers, given the goal sent by the client node.
- **Nav2 Servers**: this component is responsible for the autonomous navigation of the mobile base using Nav2. It includes also the action server for computing the parking algorithm and the action client exploiting Nav2 Commander APIs to send navigation goals to the mobile base. The structure of the parking algorithm is explained in section 3.8, while the autonomous navigation integration with ROS2 is explained in section 3.7.

- **Client Node**: this component is responsible for orchestrating the actions of the mobile base and the robotic arm. The client node is a ROS2 node that integrates the action clients for the parking, button finder, and button presser action servers. The orchestration of the actions is handled by different threads that are executed on-demand by the client node, given the goal sent by the user. Multiple threads are present since each one is responsible for handling different tests and demo versions.
- **Neural Network Node**: this component is responsible for running the YOLOv8 neural network inference on the RGB images taken by the camera. The neural network node is a ROS2 node that subscribes to the RGB image topic and publishes the predictions on a ROS2 topic, as described in section 3.10. The neural network node is implemented as a wrapper around the Tensorflow model and uses the Tensorflow Python API to load the model and run the inference.
- **ArUco Pose Estimation node**: this component is responsible for computing the ArUco markers' poses in the camera frame.

(a) The cobot opens the gripper in the proximity of the apple (at the computed grasping pose) to grasp it



(b) The cobot grasps the apple from the plant

Figure 4.12: Mobile Fruit Picking demo during execution

Figure 4.13: Basket positioned on the mobile robot base

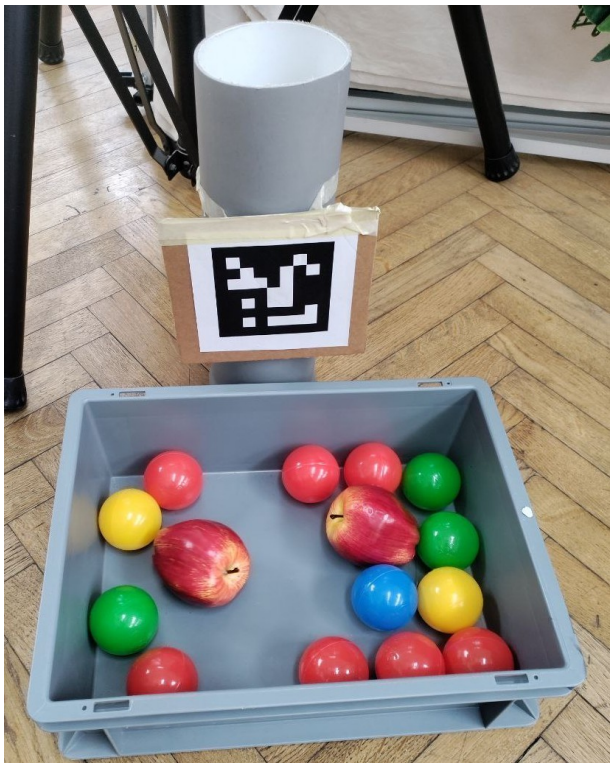


Figure 4.14: Basket at the dropping location

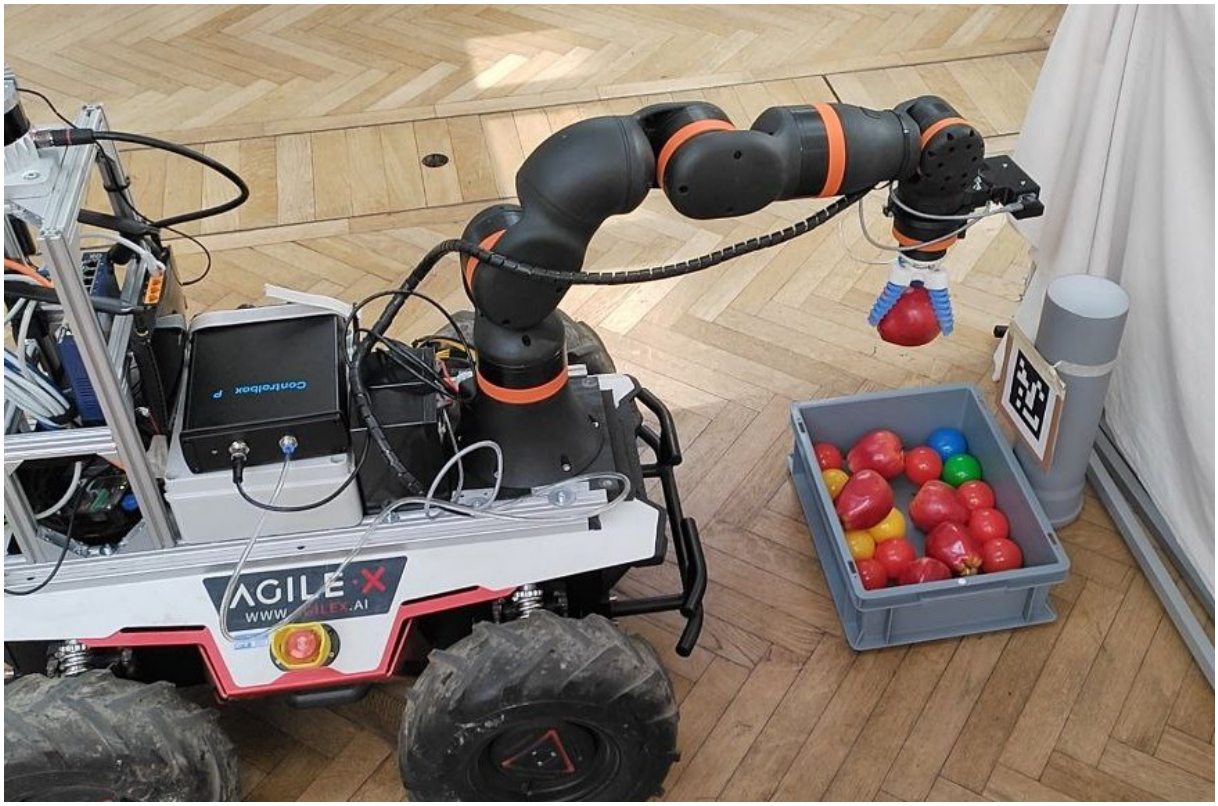


Figure 4.15: Cobot dropping the apple in the basket, at the dropping location, during the mobile fruit picking demo

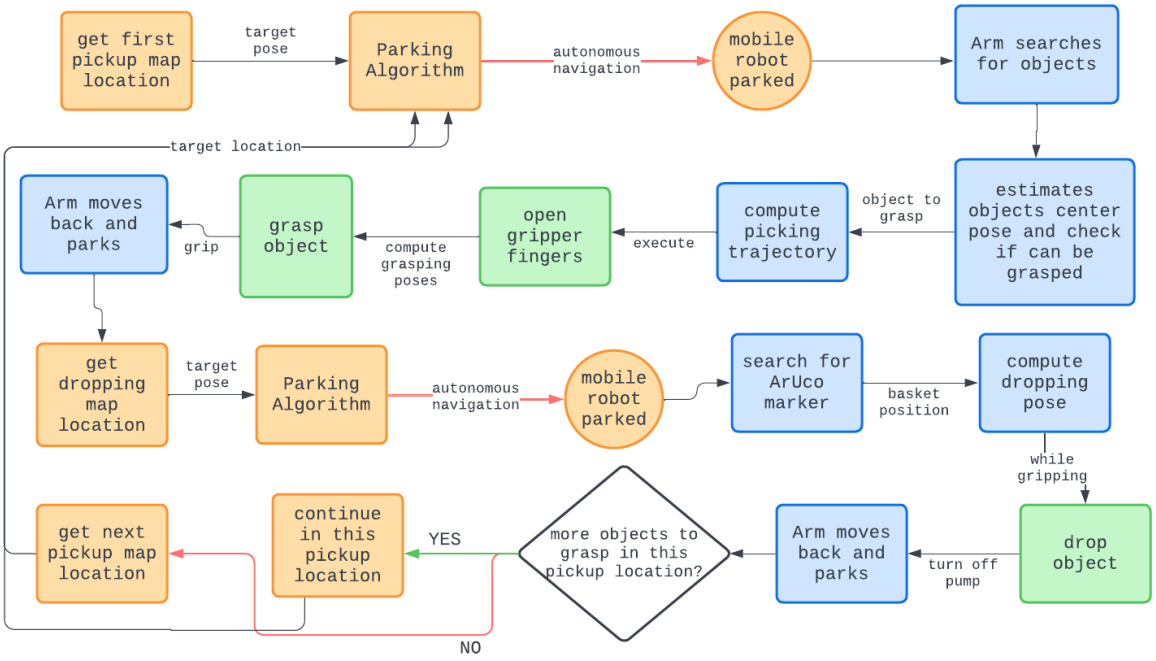


Figure 4.16: Flowchart for the Mobile Fruit Picking Demov2 with pick and place

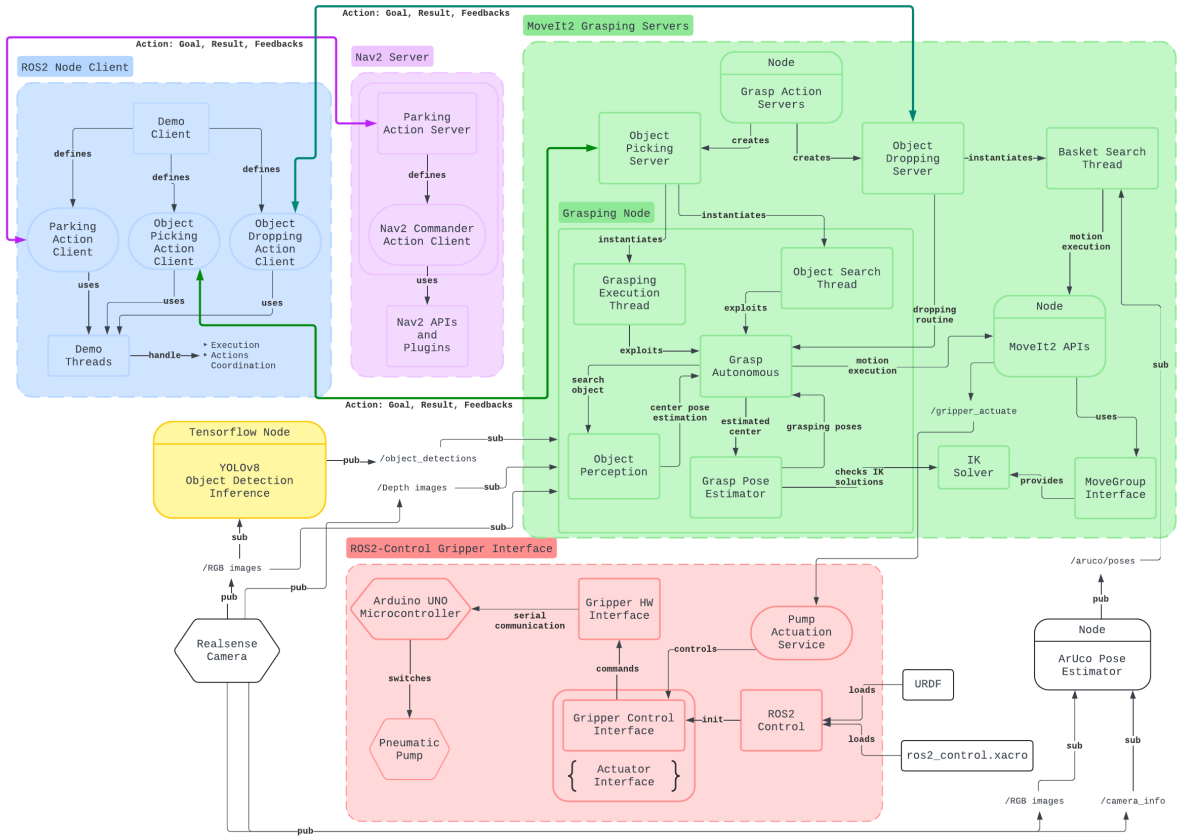


Figure 4.17: Architecture diagram of the Mobile Fruit Picking demo

5 | Conclusions and Future Work

This thesis has explored the challenges and potential solutions for developing a mobile manipulator robot capable of operating in dynamic, unstructured environments. The primary goal is to demonstrate the feasibility of autonomous navigation and object manipulation using various end-effector configurations, not to surpass human capabilities, but to establish a foundation for future research and more complex robotic systems. This work has addressed the need for adaptable and intelligent robots that can perform a wide range of tasks in real-world scenarios, such as in agriculture, and industries, using affordable and accessible hardware and software tools.

Key contributions of this work include the implementation of a modular software framework using ROS2, enabling seamless integration of diverse functionalities such as perception, planning, and control. A novel approach to visual end effector servo-ing, though not complete, showcases the potential for enhancing object manipulation capabilities. The successful integration of a pneumatic soft gripper, while highlighting the complexities of modeling its deformable nature, demonstrated the value of such grippers in handling delicate objects. This feature is particularly relevant for applications in agriculture, where the ability to grasp and manipulate fragile produce is essential.

The experience gained in developing this system has emphasized the critical importance of robust perception and localization in dynamic environments, particularly concerning sensor data fusion and data processing. The project also underscores the challenges of integrating different hardware components and the need for meticulous calibration and configuration. It also highlights the importance of developing a comprehensive simulation environment to test and validate the system before deployment in the real world, which can help to identify potential issues in a safer environment and improve the system's performance and reliability. The most valuable lesson learned from this project is the importance of a systematic and iterative approach to robot development, starting from simple tasks in a simulated scenario and gradually increasing complexity to ensure the system's adaptability and scalability to real-world applications.

The mobile manipulator system presented in this thesis serves as a proof of concept for

automating tasks commonly found in agricultural and industrial settings. The modular software design and comprehensive documentation provide a valuable starting point for future researchers to extend and refine the system's capabilities. The current implementation shows the limitations of the system in terms of reliability, particularly in the grasping and manipulation of objects, which can be improved through more sophisticated grasping strategies and control algorithms.

The legacy of this project lies in its potential to inspire further research and development in the field of mobile manipulation. Future work could focus on enhancing more advanced perception algorithms and exploring deep learning techniques to improve object recognition and grasping strategies. A key area for improvement is the integration of ROS2 behavior trees for more complex task planning and execution, enabling more articulated and adaptive behaviors in the robots.

Ultimately, this research aims to contribute to the broader goal of creating adaptable and intelligent robotic systems capable of performing complex tasks in real-world environments, using low-cost robotic platforms and open-source software. The project showcases the potential of mobile manipulators to revolutionize industries such as agriculture, industrial automation, and logistics, by providing flexible and cost-effective solutions to a wide range of applications. By addressing the challenges and limitations of the current system, future research can build upon this work to develop more sophisticated and capable robotic systems with the use of imitation learning and deep reinforcement learning techniques to learn complex manipulation tasks. These advancements will pave the way for a new generation of robots that can operate autonomously in dynamic and unstructured environments, transforming the way we work and live in the future.

Bibliography

[1] Khashayar Asadi et al. "Automated Object Manipulation Using Vision-Based Mobile Robotic System for Construction Applications". In: *Journal of Computing in Civil Engineering* 35.1 (2021), p. 04020058. DOI: 10.1061/(ASCE)CP.1943-5487.0000946. eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29CP.1943-5487.0000946.

[2] ETH Zurich Autononous Systems Lab. *Go Fetch: Mobile Manipulation in Unstructured Environments*. Published on YouTube by the Autonomous Systems Lab, ETH Zurich. 2020. URL: https://www.youtube.com/watch?v=videoID.

[3] Kenneth Blomqvist et al. "Go Fetch: Mobile Manipulation in Unstructured Environments". In: (2020). DOI: 10.48550/arXiv.2004.00899. arXiv: 2004.00899 [cs.RO]. URL: https://doi.org/10.48550/arXiv.2004.00899.

[4] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. "Deep Whole-Body Control: Learning a Unified Policy for Manipulation and Locomotion". In: *Conference on Robot Learning (CoRL)*. 2022. DOI: https://doi.org/10.48550/arXiv.2210.10044. arXiv: 2210.10044. URL: https://manipulation-locomotion.github.io/.

[5] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". In: (2024). DOI: 10.48550/arXiv.2401.02117. arXiv: 2401.02117 [cs.RO]. URL: https://mobile-aloha.github.io/.

[6] Armin Hornung et al. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013). DOI: 10.1007/s10514-012-9321-0. URL: https://octomap.github.io.

[7] Han Hu et al. "A Sim-to-Real Pipeline for Deep Reinforcement Learning for Autonomous Robot Navigation in Cluttered Rough Terrain". In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6569–6576. DOI: 10.1109/LRA.2021.3093551.

[8] Ander Iriondo et al. “Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning”. In: *Applied Sciences* 9.2 (2019). ISSN: 2076-3417. DOI: 10.3390/app9020348. URL: https://www.mdpi.com/2076-3417/9/2/348.

[9] Ander Iriondo et al. “Learning positioning policies for mobile manipulation operations with deep reinforcement learning”. In: *International Journal of Machine Learning and Cybernetics* 14.9 (Sept. 2023), pp. 3003–3023. DOI: 10.1007/s13042-023-01815-8. URL: https://doi.org/10.1007/s13042-023-01815-8.

[10] Sanghoon Ji et al. “Learning-Based Automation of Robotic Assembly for Smart Manufacturing”. In: *Proceedings of the IEEE* 109.4 (2021), pp. 423–440. DOI: 10.1109/JPROC.2021.3063154.

[11] Hei Law and Jia Deng. “CornerNet: Detecting Objects as Paired Keypoints”. In: (2019). arXiv: 1808.01244 [cs.CV].

[12] Rongrong Liu et al. “Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review”. In: *Robotics* 10 (2021), p. 22. DOI: 10.3390/robotics10010022. URL: https://doi.org/10.3390/robotics10010022.

[13] Steve Macenski, Matthew Booker, and Joshua Wallace. “Open-Source, Cost-Aware Kinematically Feasible Planning for Mobile and Surface Robotics”. In: (2024). arXiv: 2401.13078 [cs.RO].

[14] Steve Macenski and Ivona Jambrecic. “SLAM Toolbox: SLAM for the dynamic world”. In: *Journal of Open Source Software* 6.61 (2021), p. 2783. DOI: 10.21105/joss.02783. URL: https://doi.org/10.21105/joss.02783.

[15] Steve Macenski, David Tsai, and Max Feinberg. “Spatio-temporal voxel layer: A view on robot perception for the dynamic world”. In: *International Journal of Advanced Robotic Systems* 17.2 (2020). DOI: 10.1177/1729881420910530. URL: https://doi.org/10.1177/1729881420910530.

[16] Steve Macenski et al. “From the desks of ROS maintainers: A survey of modern and capable mobile robotics algorithms in the robot operating system 2”. In: *Robotics and Autonomous Systems* 168 (Oct. 2023), p. 104493. ISSN: 0921-8890. DOI: 10.1016/j.robot.2023.104493. URL: http://dx.doi.org/10.1016/j.robot.2023.104493.

[17] Steven Macenski et al. “The Marathon 2: A Navigation System”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[18] Mayank Mittal et al. "Articulated Object Interaction in Unknown Scenes with Whole-Body Mobile Manipulation". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 1647–1654. DOI: 10.1109/IROS47612.2022.9981779. URL: https://www.pair.toronto.edu/articulated-mm/.

[19] NVIDIA. *Isaac Gym*. 2024. URL: https://developer.nvidia.com/isaac-gym.

[20] NVIDIA. *Isaac ROS bridge*. 2024. URL: https://developer.nvidia.com/isaac-ros.

[21] NVIDIA. *Isaac Sim*. 2024. URL: https://developer.nvidia.com/isaac-sim.

[22] David Pick and MoveIt! Developers. *MoveIt! Documentation*. 2023. URL: https://moveit.picknik.ai/main/index.html.

[23] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (2015). arXiv: 1506.02640 [cs.CV].

[24] Dillon Reis et al. "Real-Time Flying Object Detection with YOLOv8". In: (2023). arXiv: 2305.09972 [cs.CV].

[25] Open Robotics. *ROS 2 Documentation*. 2023. URL: https://docs.ros.org/en/rolling/index.html.

[26] Malak H. Sayour, Sharbel E. Kozhaya, and Samer S. Saab. "Autonomous Robotic Manipulation: Real-Time, Deep-Learning Approach for Grasping of Unknown Objects". In: *Journal of Robotics* 2022 (June 2022). ISSN: 1687-9600. DOI: 10.1155/2022/2585656. URL: https://doi.org/10.1155/2022/2585656.

[27] Petr Štibinger et al. "Mobile Manipulator for Autonomous Localization, Grasping and Precise Placement of Construction Material in a Semi-Structured Environment". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2595–2602. DOI: 10.1109/LRA.2021.3061377.

[28] Charles Sun et al. "Fully Autonomous Real-World Reinforcement Learning with Applications to Mobile Manipulation". In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, Nov. 2022, pp. 308–319. URL: https://proceedings.mlr.press/v164/sun22a.html.

[29] Shantanu Thakar et al. "A Survey of Wheeled Mobile Manipulation: A Decision-Making Perspective". In: *Journal of Mechanisms and Robotics* 15.2 (July 2022), p. 020801. ISSN: 1942-4302. DOI: 10.1115/1.4054611. URL: https://doi.org/10.1115/1.4054611.

[30] Cong Wang et al. "Learning Mobile Manipulation through Deep Reinforcement Learning". In: *Sensors* 20.3 (2020). ISSN: 1424-8220. DOI: 10.3390/s20030939. URL: https://www.mdpi.com/1424-8220/20/3/939.

[31] Cong Wang et al. "Multi-Task Reinforcement Learning based Mobile Manipulation Control for Dynamic Object Tracking and Grasping". In: *2022 7th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. July 2022, pp. 34–40. DOI: 10.1109/ACIRS55390.2022.9845515.

[32] Grady Williams et al. "Aggressive driving with model predictive path integral control". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[33] Luke Wood et al. "KerasCV". In: (2022).

[34] Kejian Yan, Jianqi Gao, and Yanjie Li. "Deep Reinforcement Learning Based Mobile Robot Navigation Using Sensor Fusion". In: *2023 42nd Chinese Control Conference (CCC)*. 2023, pp. 4125–4130. DOI: 10.23919/CCC58697.2023.10240555.

[35] Tony Z. Zhao et al. "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware". In: (2023). DOI: 10.48550/arXiv.2304.13705. arXiv: 2304.13705 [cs.RO]. URL: https://tonyzhaozh.github.io/aloha/.

[36] Zhaohui Zheng et al. "Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation". In: (2021). arXiv: 2005.03572 [cs.CV].

List of Figures

1.1 The continuum in the literature in regards to control methodology ranging from model-based to end-end data-driven control [29] 9
1.2 The described system loading and placing building material during the MBZIRC 2020 contest. [27] 9
1.3 A picture of RoyalYumi in action. It features a two-arm ABB Yumi, a Clearpath Ridgeback mobile base, two Hokuyo 2D LiDARs, an Intel RealSense D435 and a Skybotix VI-Sensor. [3] 10
1.4 A schematic diagram example for robotic manipulation control using a data-driven approach such as DRL [12] 12
1.5 ReLMM partitions the mobile manipulator into a navigation policy and grasping policy. Both policies are rewarded when an object is grasped [28] 13
1.6 KUKA robot mounted on a mobile platform for pick and place tasks in industrial environments [9] 15
1.7 The two-level hierarchy in the proposed framework. The object-centric planner comprises a scene interpreter and keyframe generator. It uses perceptual information to generate task space plans. The agent-centric planner follows the computed plan while satisfying constraints and performing online collision avoidance. [18] 17
1.8 Legged mobile manipulation of articulated objects in the kitchen test scenario: (a) Drawer, (b) Cabinet. Throughout the interaction, we set the robot's gait schedule to trot. Only while grasping the handle, the robot enters stance mode. [18] 18
1.9 Framework for whole-body control of a legged robot with a robot arm attached. The left half shows how whole-body control achieves a larger workspace by leg bending and stretching. The right half shows different real-world tasks, including wiping the whiteboard, picking up a cup, pressing door-open buttons, placing, throwing a cup into a garbage bin and picking in clustered environments. [4] 19

1.10 Whole-body control framework. During training, a unified policy is learned by conditioning on the environment extrinsic. During deployment, the adaptation module is reused without any real-world fine-tuning. The robot can be commanded in various modes including teleoperation, vision and demonstration replay. [4] 20
1.11 Learning-based mobile manipulation control framework. There are mainly two parts, deep reinforcement learning module and vision module. First, the vision module estimates the object 6 degrees of freedom pose p from images captured by an onboard RGB stereo camera. Then, based on the object pose p and current robot state st, deep reinforcement learning module predicts an action for the robot to act. A new state s_{t+1} and a reward r_{t+1} are received after action.[30] 21
1.12 Real mobile grasping process for a soup can. (a) is starting, (b,c,d) is approaching, (e) is grasping, and (f) is picking up. [30] 22
1.13 (a) Dynamic trajectory tracking task with a mobile manipulator. (b) Dynamic object grasping task with a mobile manipulator. (c) Several basic trajectories as multi-task RL training sets. (d) Random trajectories as multi-task RL testing set.[31] 23
1.14 (a) In the multi-task RL training, six basic trajectories are used as the task training set to train a general policy. To improve the robustness, gaussian noise is added to the action and observation space in each training episode. (b) The RL testing includes simulation and real-world for policy evaluation.[31] 24
1.15 Snapshots of the real robot experiments. The upper row shows a mobile tracking process in which the end-effector tries to track the target trajectory. The lower row shows a mobile grasping process in which the object moves randomly. [31] 25
1.16 Low-cost mobile manipulation system that is bimanual and supports whole-body teleoperation. The system costs 32k USD including onboard power and compute. Left: A user teleoperates to obtain food from the fridge. Right: Mobile ALOHA can perform complex long-horizon tasks with imitation learning. [5] 25

1.17 Detail architecture of Action Chunking with Transformers (ACT). First, they train ACT as a Conditional VAE (CVAE), which has an encoder and a decoder. The encoder of the CVAE compresses action sequence and joint observation into z, the style variable. The encoder is discarded at test time. The decoder or policy of ACT synthesizes images from multiple viewpoints, joint positions, and z with a transformer encoder, and predicts a sequence of actions with a transformer decoder. z is simply set to the mean of the prior (i.e. zero) at test time [35]. 33
1.18 Simulated environment with Nvidia Isaac Gym [20]. The screenshot depicts a simulated environment with many legged mobile manipulators trained in parallel using DRL, as in [18]. The environment is simulated using Nvidia Isaac Sim [21] 34
1.19 Grasp generation results: comparison between grasp generated by the GG-CNN with and without RGB-D image preprocessing for shiny and black objects [26] 34

2.1 Scout 2.0 employs 200W brushless servo motors to drive each wheel independently. Its double-wishbone suspension with shock absorbers ensures stability on rough terrain, enabling it to tackle obstacles up to 10cm tall effortlessly. 36
2.2 Intel NUC 12 computer mounted on the robot 37
2.3 Igus ReBeL 6-DoF robotic arm 38
2.4 Igus ReBeL proprietary control software interface 39
2.5 Ouster OS1-64 LiDAR sensor 41
2.6 Intel RealSense D435 RGB-D stereo camera 42
2.7 Calibration pattern used for the camera's depth image calibration 42
2.8 Soft Gripper Pneumatic Actuator handling an apple 44
2.9 Pneumatic Pump control box secured on top of the mobile robot 45
2.10 Soft Gripper mounted on the end effector 46
2.11 Arduino UNO microcontroller and relay module used to control the pneumatic pump. 47
2.12 Circtuit schematic for the pneumatic pump control system using an Arduino UNO and relay modules. 48
2.13 MountV1 Design screenshots from *Autodesk Fusion 360* 49
2.14 MountV2 Design screenshots from *Autodesk Fusion 360* 50
2.15 3d-printed mountV2 on the arm's wrist 51
2.16 GPS antenna mount design and 3D-print on top of the LiDAR 52

2.17 Lead batteries mounted onboard for the cobot and pneumatic pump 53
2.18 DC/DC converter used to power the robot's sensors and computer 54
2.19 Molex connectors and power management for the cobot, pump, and relays 55
2.20 Lateral view of the robots 56
2.21 Lateral view of the robots 57

3.1 ROS2-Control and MoveIt2 Interfaces Architecture 61
3.2 MoveIt2 General Architecture 63
3.3 Planning Scene and Occupancy Mapping Architecture 64
3.4 MoveGroup Interface 65
3.5 RViz2 Interface with MoveIt2 Control Panels and the mobile manipulation robot. 66
3.6 Teleoperation with a joystick controller in RViz2 with the Franka Emika Panda 68
3.7 Mobile robot using the depth camera to construct the Octomap with the obstacles nearby 70
3.8 Ignition Gazebo Simulation Environment 71
3.9 Mobile robot navigating in the simulated warehouse environment 72
3.10 Autonomous Navigation with Nav2 in the hallways of building 7 of Politecnico di Milano 74
3.11 Autonomous Navigation in cluttered and dynamic environment with moving obstacles (AIRlab) 75
3.12 STVL in action in a dynamic and cluttered warehouse environment 79
3.13 3D LiDAR sensor in a cluttered environment 80
3.14 Multi-ArUco Plane Estimation algorithm in action. The screenshot shows RViz2 displaying on the top left corner the input image, the bottom left the image with the detected markers drawn on top of it, and in the center the colored pointcloud captured by the depth sensor. 87
3.15 Architecture overview in detail of the YOLOv8 architecture 91
3.16 YOLOv8 detecting colored balls in realtime from the RGB camera feed. The screenshot displays the image with the bounding boxes colored with the same color as the predicted class. For each prediction, the class probability is associated. 93
3.17 ROS2 Actions Client-Server Architecture 101

4.1 ArUco Follower demo during execution. The cardboard shows an ArUco marker. 107
4.2 Control panel with 3 buttons and ArUco markers. 109

4.3 Flowchart of the Mobile Button Presser demo execution 113
4.4 Mobile Button Presser Demo in execution 114
4.5 Cobot pressing buttons on the control panel using the *MountV1* end effector 115
4.6 Architecture diagram of the Mobile Button Presser demo 116
4.7 Espalier apple trees 118
4.8 Magnetic attachment of an apple to the nylon string 118
4.9 Simulated plant with colored balls 119
4.10 Magnetic hook for attaching apples 120
4.11 Cobot grasping a colored ball from the hand of a volunteer 121
4.12 Mobile Fruit Picking demo during execution 129
4.13 Basket positioned on the mobile robot base 130
4.14 Basket at the dropping location 130
4.15 Cobot dropping the apple in the basket, at the dropping location, during the mobile fruit picking demo 131
4.16 Flowchart for the Mobile Fruit Picking Demov2 with pick and place 131
4.17 Architecture diagram of the Mobile Fruit Picking demo 132

List of Tables

1.1 Summary of the main differences between model-based and data-driven approaches for robotic manipulator controls 28

3.1 Evaluation metrics for the trained YOLOv8 model 95
3.2 Hyperparameters and model parameters used in YOLOv8 training 96

List of Algorithms

3.1 **Parking Pose Computation Algorithm** 83
3.2 **Multi-ArUco Plane Estimation Algorithm** 89
3.3 **Sphere Barycenter Estimation from Object Detection** 98
3.4 **Grasp Pose Estimation from Object's Barycenter** 103
4.1 **Object Picking Routine** 125

Acknowledgements

Here you might want to acknowledge someone.