Introduction

Background

Robots were originally designed to assist or replace humans by performing repetitive and/or dangerous tasks that humans usually prefer not to do, or are unable to do because of physical limitations imposed by extreme environments. With the continuous developments in mechanics, sensing technology, intelligent control and other modern technologies, robots have improved autonomy capabilities and are more dexterous.

Articulated robots are among the most common robots used today. They look like human arms and that is why they are also called robotic arms or manipulator arms. 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 [12].

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 itself 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 [12].

Challenges and Research

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 reality fits the assumptions made by the model. Well-known planning and control methods have been widely used for scheduling mobile manipulation behaviors, for example using the NAV2 navigation stack and 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.

Objectives of the Project

This Thesis project aims to develop a mobile manipulation system that can perform simple manipulation tasks in a dynamic environment. The system is based on 2 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 simple 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 in a completely autonomous fashion, without human intervention, and minimal human supervision. The core focus of the project is the complete autonomy of the system, including navigation, object detection, manipulation, and task planning.

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

The final objective of the project is the development and realisation 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 an industrial environmentm, 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 fake apples (or colored balls) from a plastic tree.

Structure of the Thesis

This Thesis is structured as follows:

- Chapter 1: **Introduction**. This chapter provides an overview of the thesis project, the challenges and the objectives.
- Chapter 2: **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 3: **Robotic Platform for Mobile Manipulation**. This chapter describes the robotic platform used in the project, including all the hardware components and sensors.
- Chapter 4: **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 5: **Experimental Setup and Demonstrations**. This chapter describes the experimental setups used to test the system and demonstrate the capabilities of the entire system.
- Chapter 6: **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

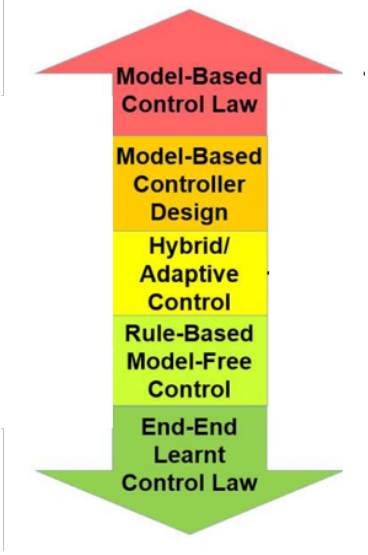


Figure 1.1: The continuum in the literature in regards to control methodology ranging from model-based to end-end data-driven control [29]

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.

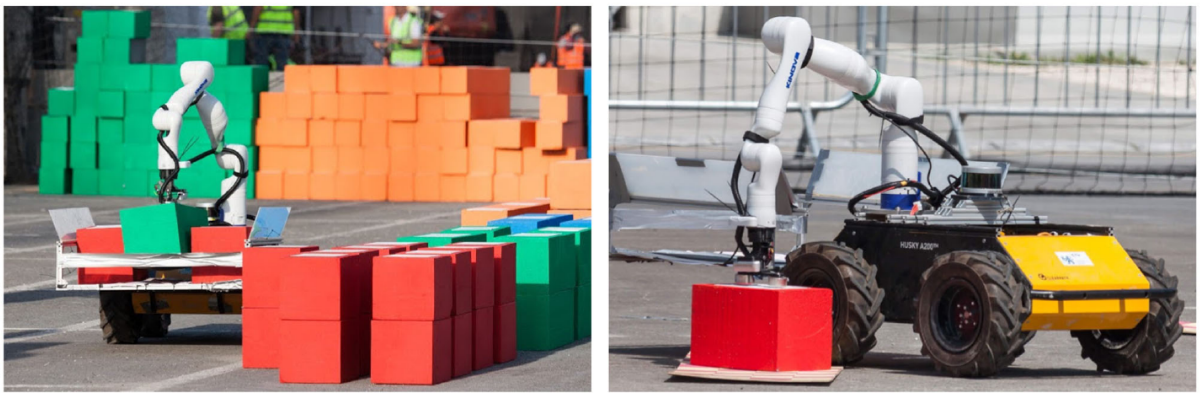


Figure 1.2: The described system loading and placing building material during the MBZIRC 2020 contest. [27]

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]

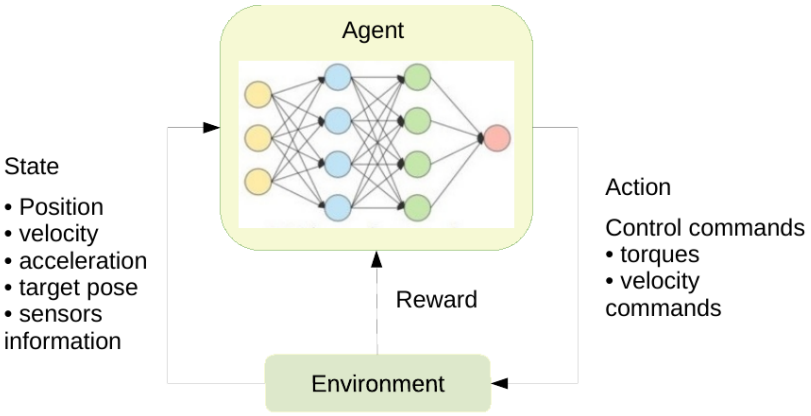


Figure 1.4: A schematic diagram example for robotic manipulation control using a data-driven approach such as DRL [12]

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.

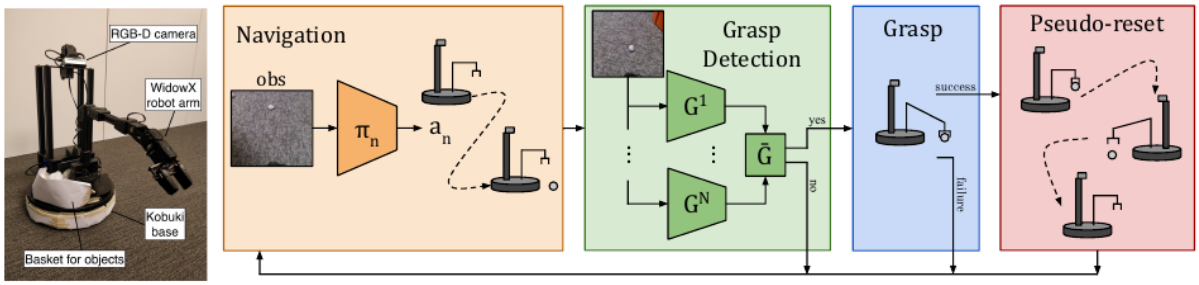


Figure 1.5: ReLMM partitions the mobile manipulator into a navigation policy and grasping policy. Both policies are rewarded when an object is grasped [28]

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

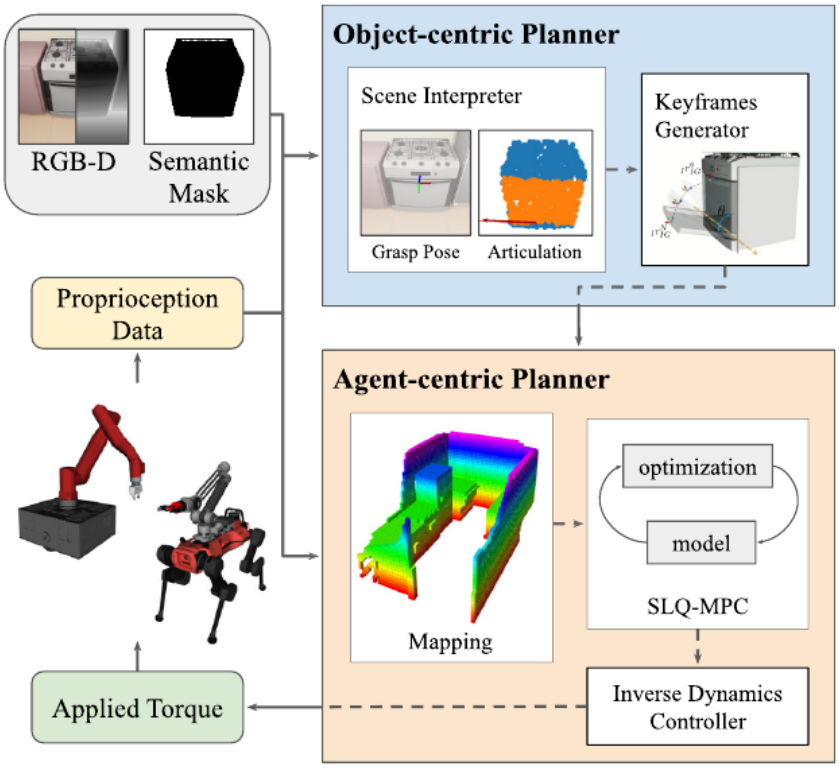


Figure 1.7: The two-level hierarchy in the proposed framework. The object-centric planner comprises a scene interpreter and keyframe generator. It uses perceptual information to generate task space plans. The agent-centric planner follows the computed plan while satisfying constraints and performing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

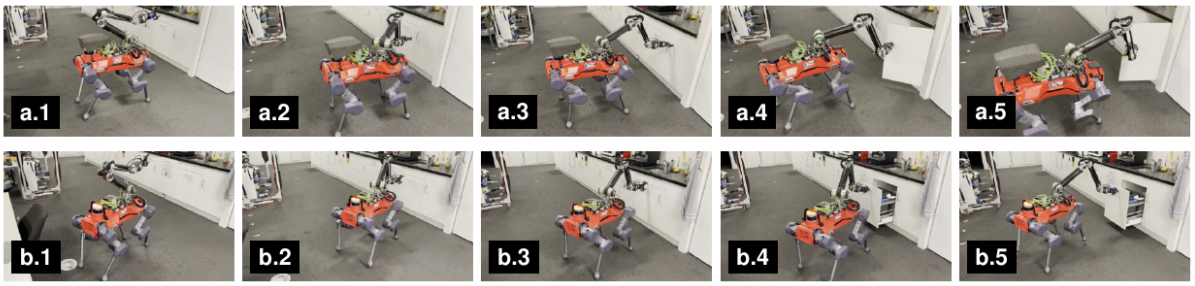


Figure 1.8: Legged mobile manipulation of articulated objects in the kitchen test scenario: (a) Drawer, (b) Cabinet. Throughout the interaction, we set the robot's gait schedule to trot. Only while grasping the handle, the robot enters stance mode. [18]

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-

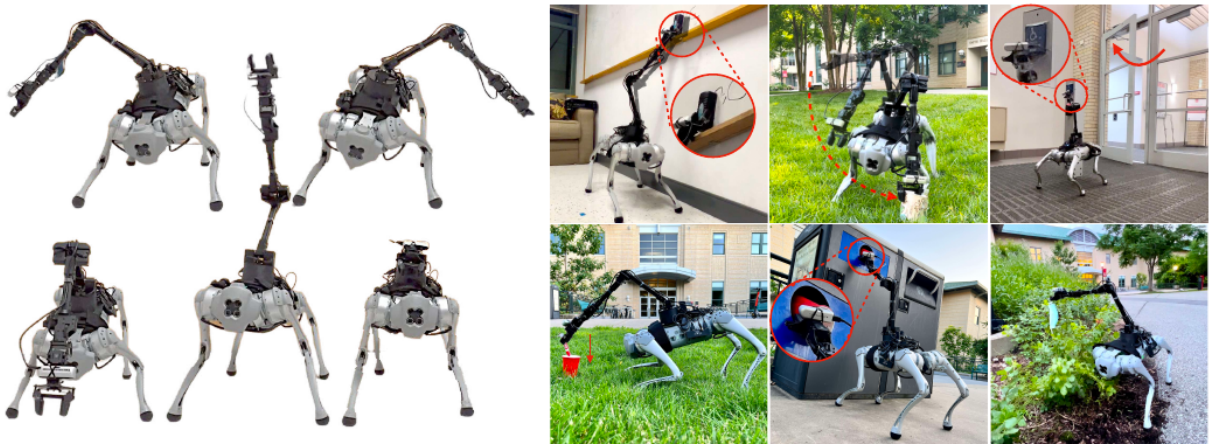


Figure 1.9: Framework for whole-body control of a legged robot with a robot arm attached. The left half shows how whole-body control achieves a larger workspace by leg bending and stretching. The right half shows different real-world tasks, including wiping the whiteboard, picking up a cup, pressing 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

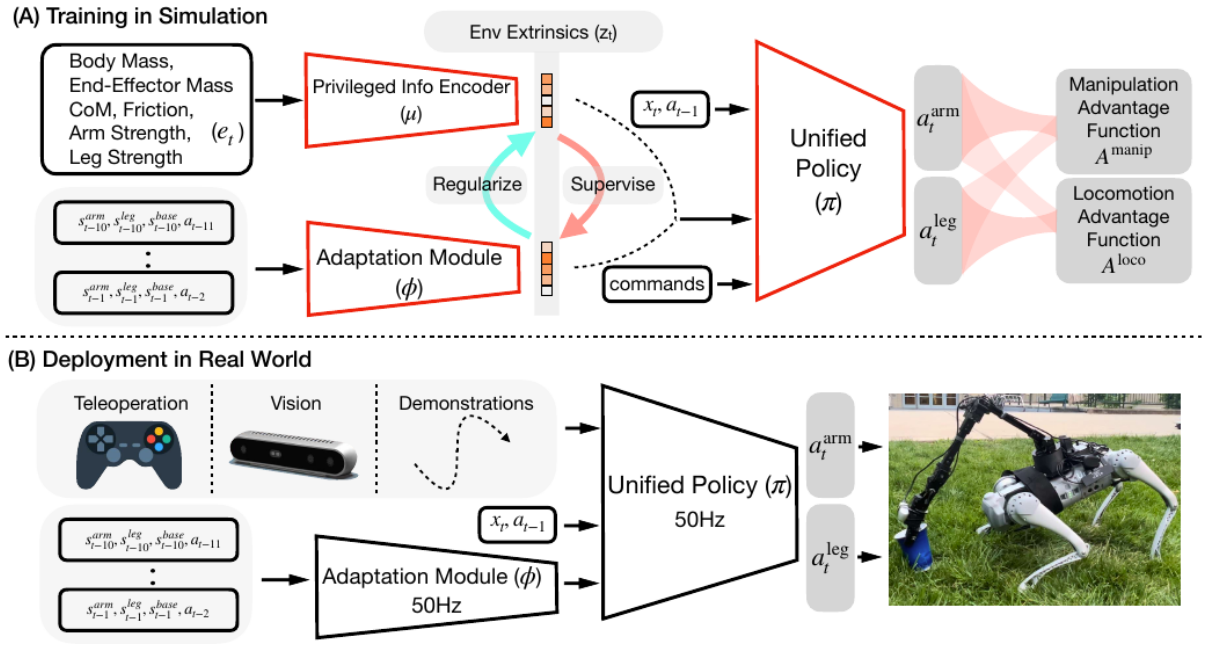


Figure 1.10: Whole-body control framework. During training, a unified policy is learned by conditioning on the environment extrinsic. During deployment, the adaptation module is reused without any real-world fine-tuning. The robot can be commanded in various modes including teleoperation, vision and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

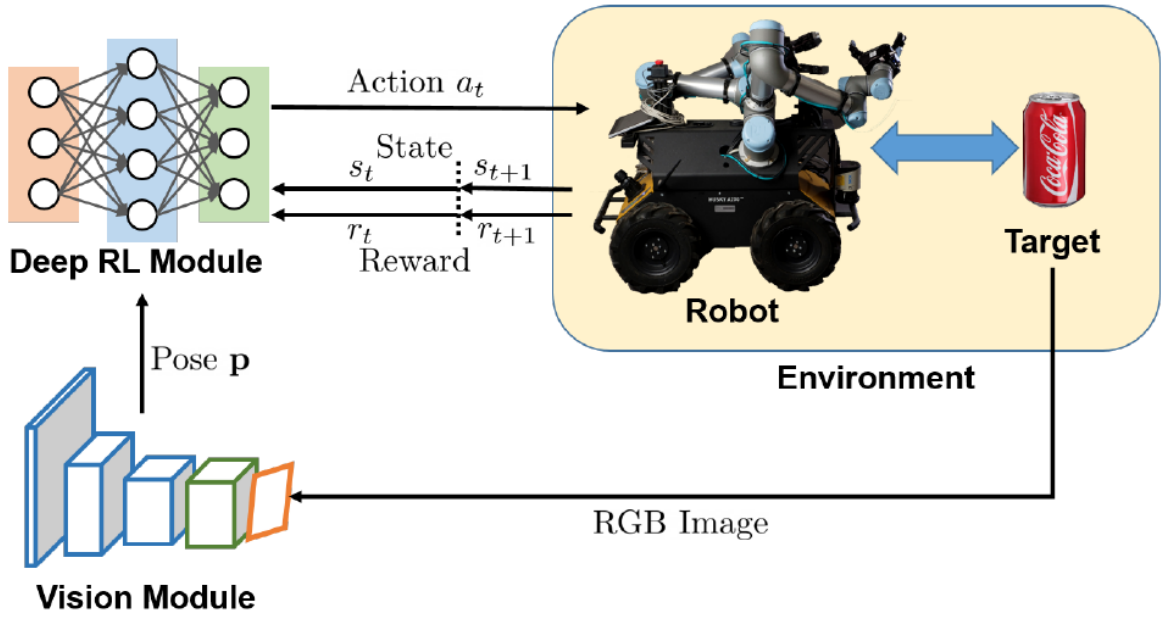


Figure 1.11: Learning-based mobile manipulation control framework. There are mainly two parts, deep reinforcement learning module and vision module. First, the vision module estimates the object 6 degrees of freedom pose p from images captured by an onboard RGB stereo camera. Then, based on the object 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

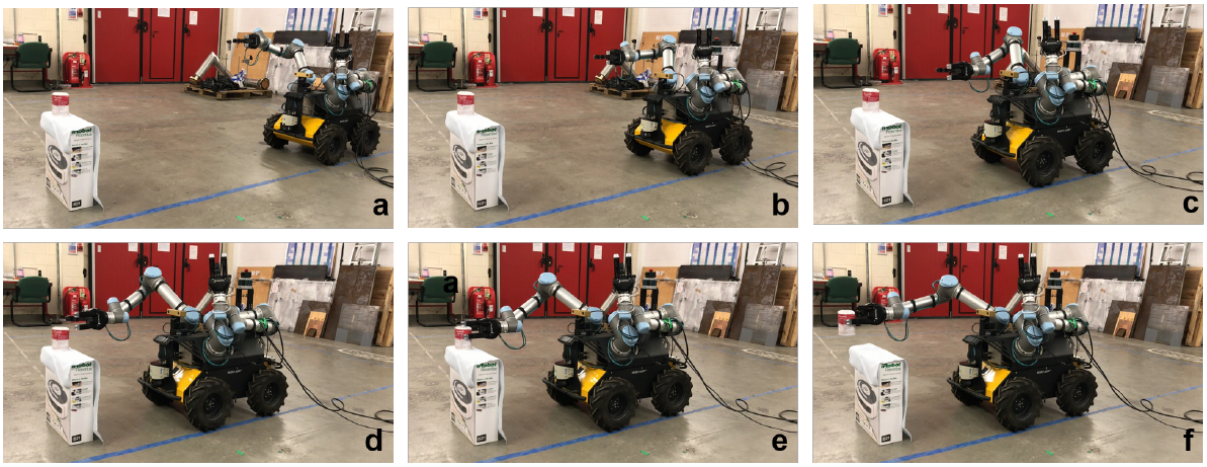


Figure 1.12: Real mobile grasping process for a soup can. (a) is starting, (b,c,d) is approaching, (e) is grasping, and (f) is picking up. [30]

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

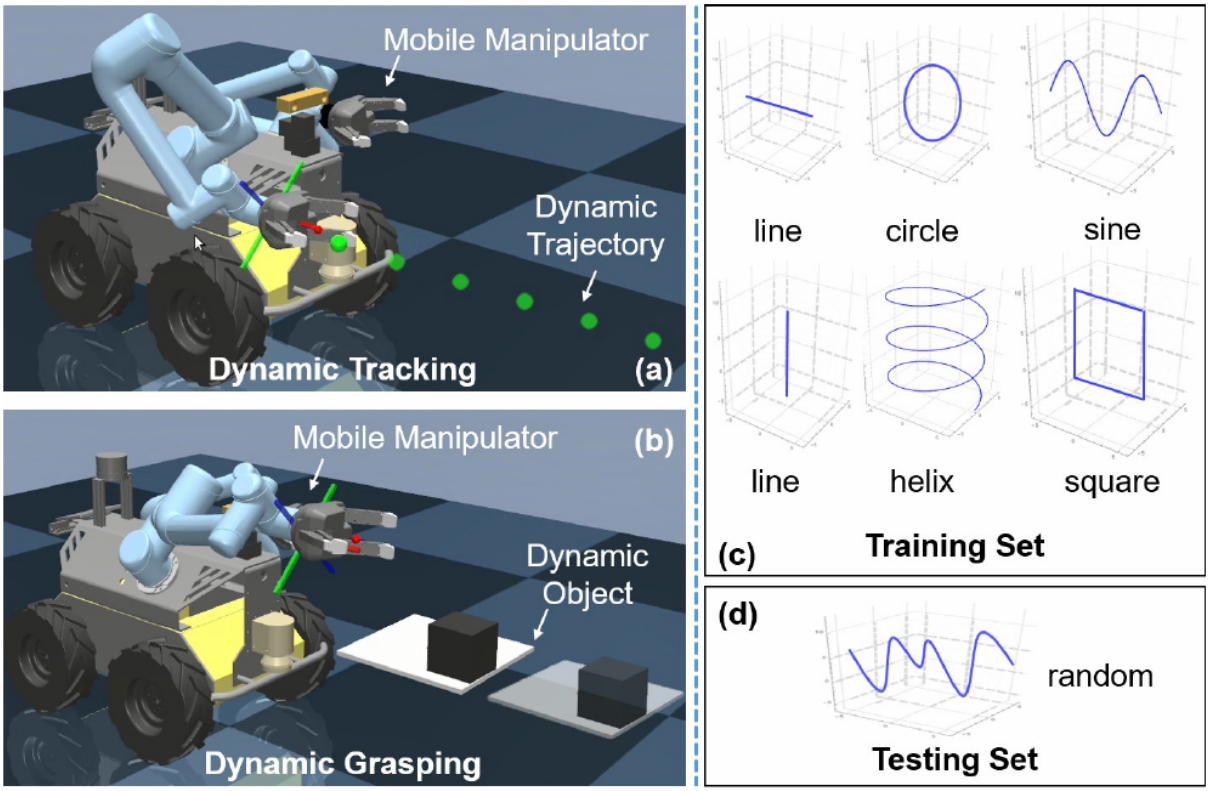


Figure 1.13: (a) Dynamic trajectory tracking task with a mobile manipulator. (b) Dynamic object grasping task with a mobile manipulator. (c) Several basic trajectories as multi-task RL training sets. (d) Random trajectories as multi-task RL testing set.[31]

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

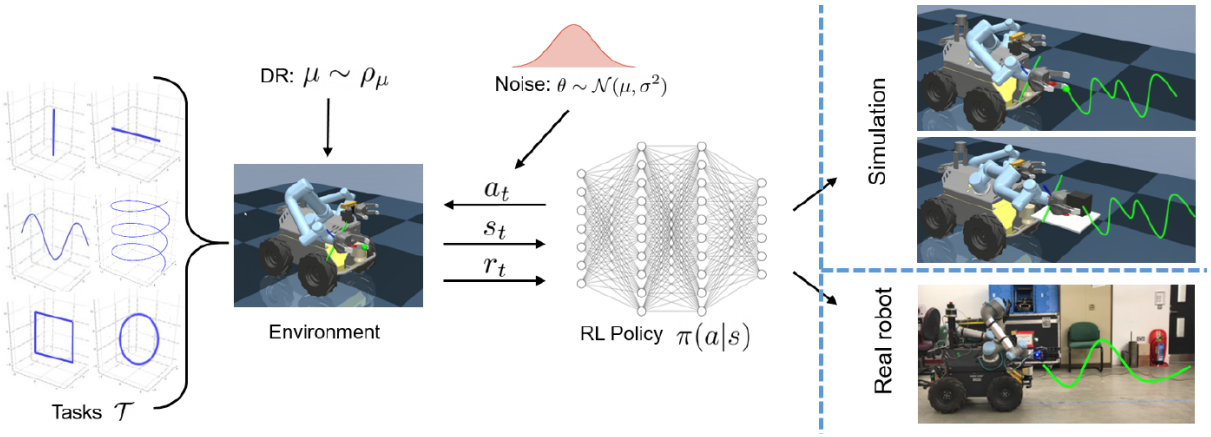


Figure 1.14: (a) In the multi-task RL training, six basic trajectories are used as the task training set to train a general policy. To improve the robustness, gaussian noise is added to the action and observation space in each training episode. (b) The RL testing includes simulation and real-world for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

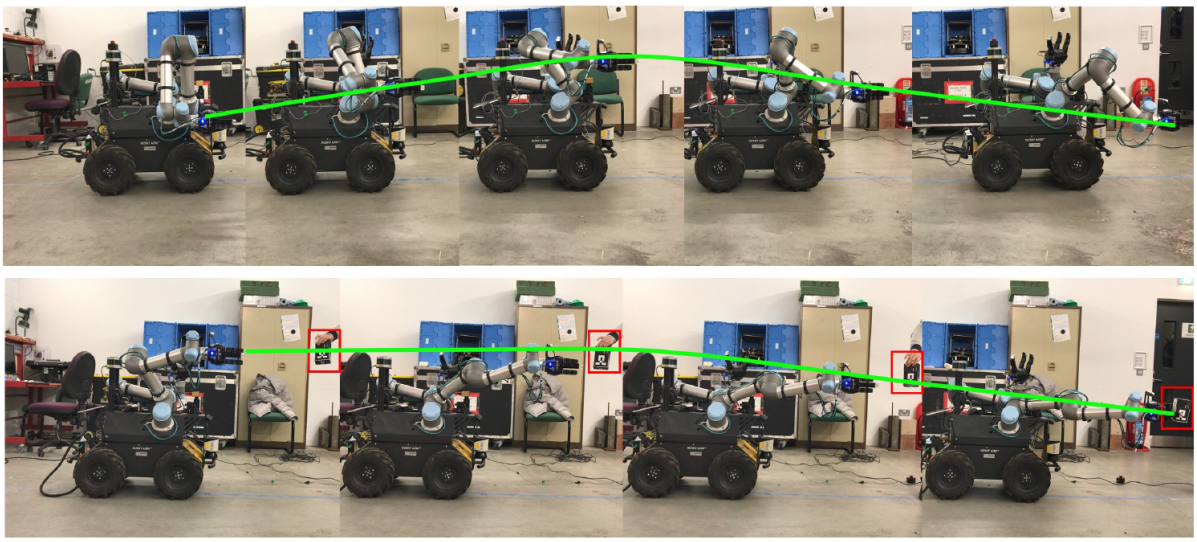


Figure 1.15: Snapshots of the real robot experiments. The upper row shows a mobile tracking process in which the end-effector tries to track the target trajectory. The lower row shows a mobile grasping process in which the object moves randomly. [31]

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.

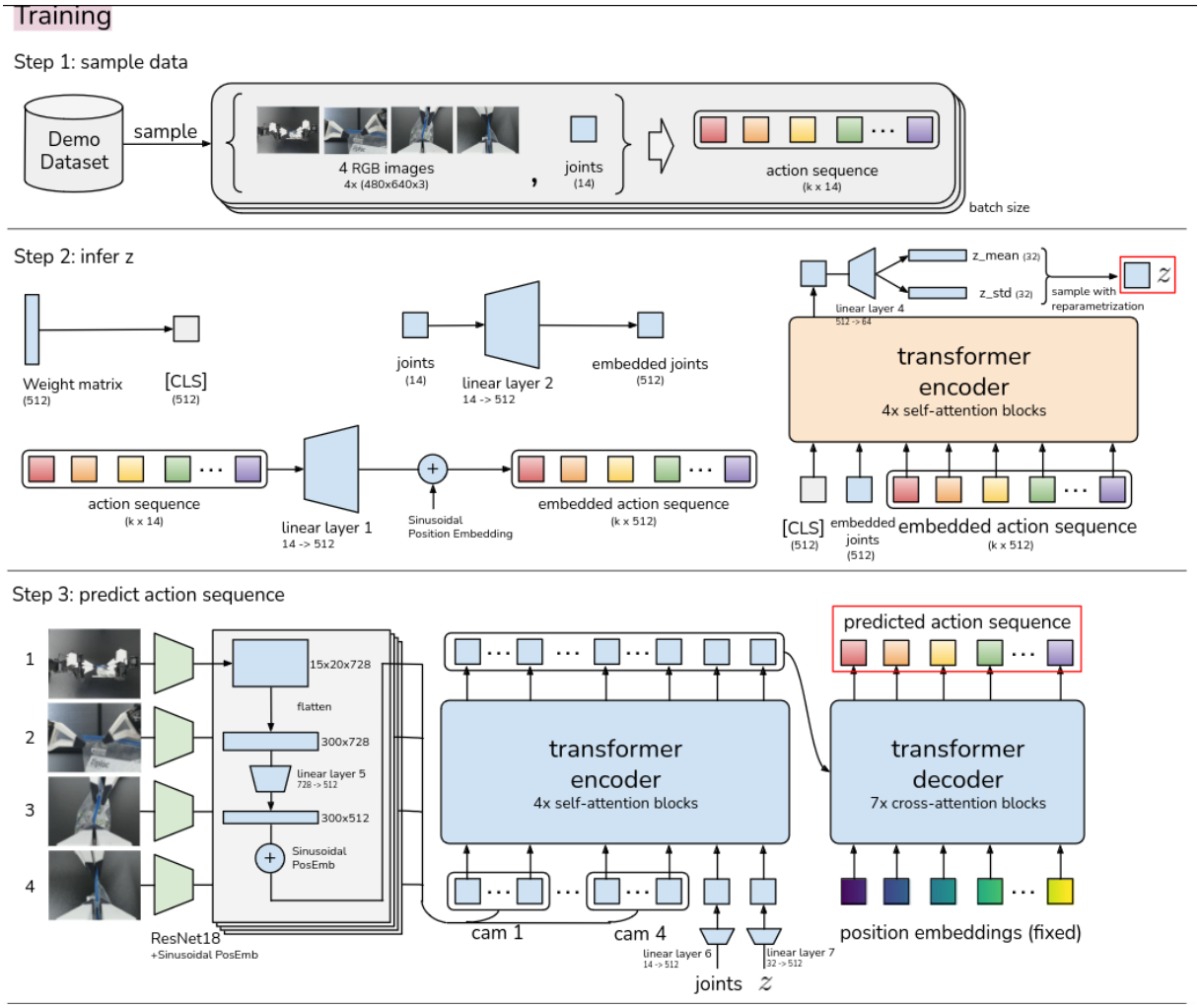


Figure 1.17: Detail architecture of Action Chunking with Transformers (ACT). First, they train ACT as a Conditional VAE (CVAE), which has an encoder and a decoder. The encoder of the CVAE compresses action sequence and joint observation into z, the style variable. The encoder is discarded at test time. 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].

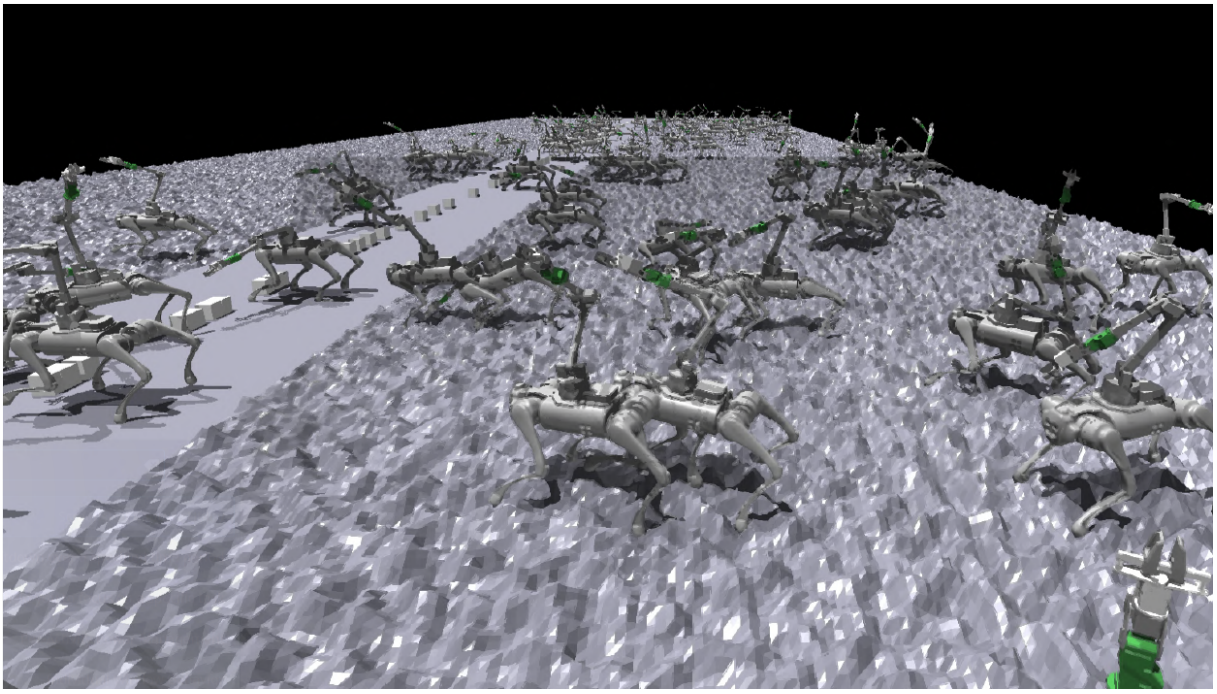


Figure 1.18: Simulated environment with Nvidia Isaac Gym [20]. The screenshot depicts a simulated environment with many legged mobile manipulators trained in parallel using DRL, as in [18]. The environment is simulated using Nvidia Isaac Sim [21]

.

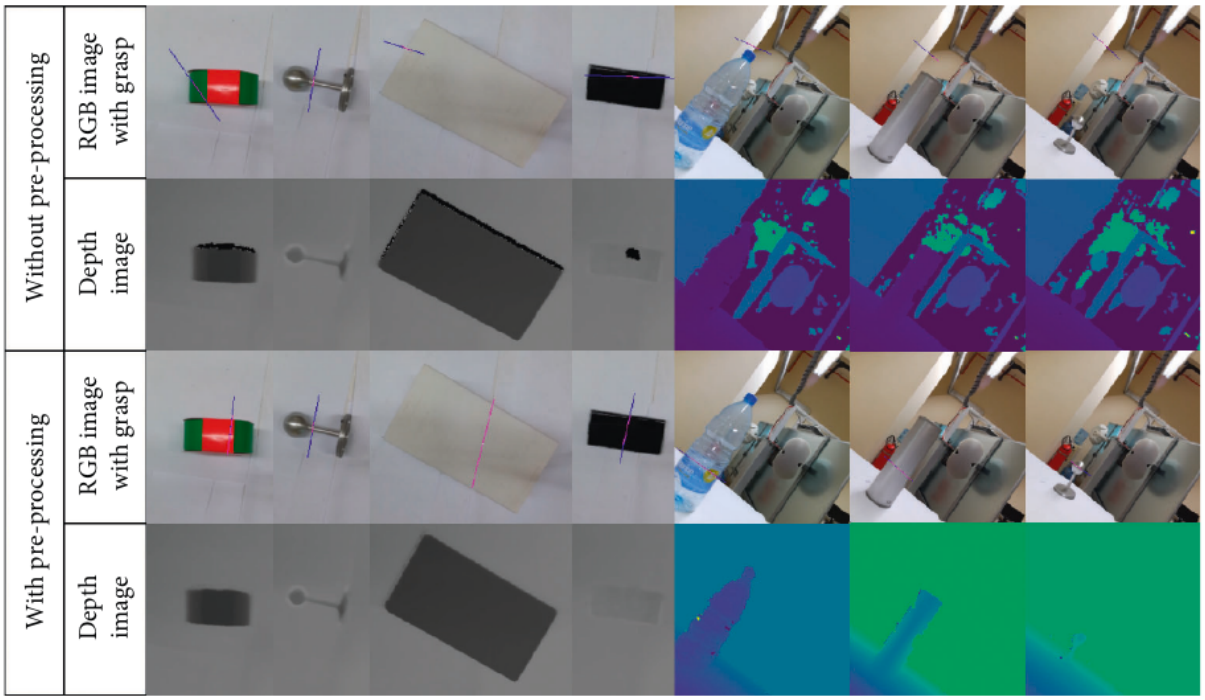


Figure 1.19: Grasp generation results: comparison between grasp generated by the GG-CNN with and without RGB-D image preprocessing for shiny and black objects [26]

2 | Robotic Platform for Mobile Manipulation

This chapter will describe the robotic platform used for the project, specifically the hardware components. The platform is composed of a mobile robot base, a robotic arm manipulator, sensors and perception systems, 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 used for the Thesis project is an *AgileX Scout 2.0* robot. 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 is CE-certified and 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, and without relying on power cables to the wall, thanks to its **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 is mounted. This **computer** is used for running all the control software and the perception algorithms. The computer is connected

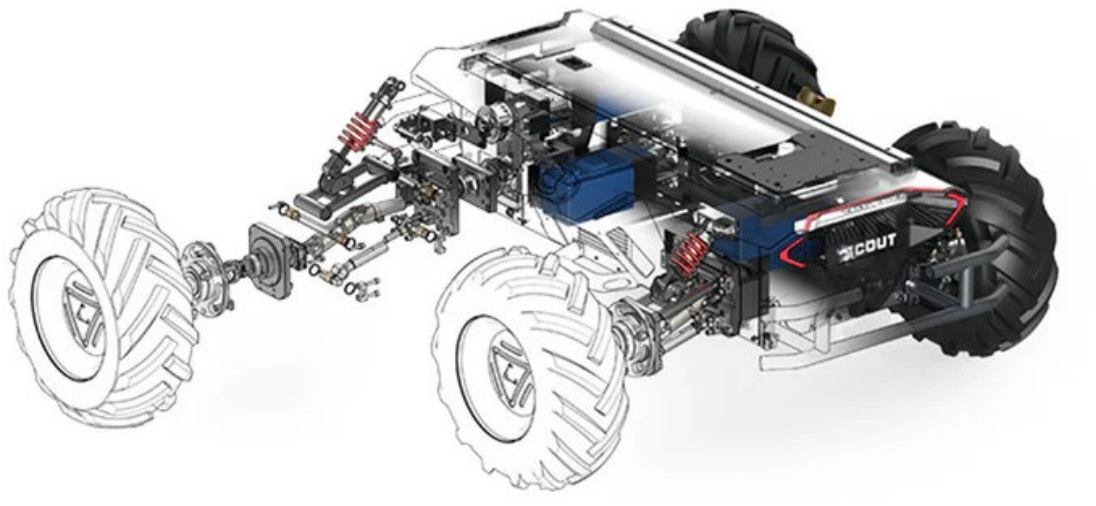


Figure 2.1: SCOUT employs 200W brushless servo motors to drive each wheel independently. Its double-wishbone suspension with shock absorbers ensures stability on rough terrain, enabling it to tackle obstacles up to 10cm tall effortlessly.

.

to a switch, which is used to connect the computer to the robot's control system and all the sensors and electronic devices mounted on the robot. The computer is also connected to a router, which is used to connect the computer to the laboratory's network and the internet. This allows the robot to be controlled 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 is used to connect the robot to the laboratory's network and the internet. 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 was essential for the development of the project, as it allowed me to work safely with the robot, ensuring that everything was working 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 switch is used to connect the robot's computer to the robot's control system and all the sensors and electronic devices mounted on the robot. The switch is connected to the router, allowing the robot's computer to communicate with the laboratory's network and the internet. 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 Thesis project is a *Igus ReBeL 6-DoF* **cobot**. 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 and compact design makes it suitable for mounting on top of mobile robot platforms, such as the Scout robot. The maximum payload of the arm is 2 kg, which is more than enough for the project's requirements. The weight is 8.2 kg, which is light enough to be carried around by the mobile robot base, without affecting the robot's mobility and stability.

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

- Lightweight, sleek and compact design

Figure 2.3: Igus ReBeL 6-DoF robotic arm (cobot)

.

- Plastic arm, inexpensive and cost-effective
- Easy to install and operate
- Plug and play proprietary control system, or open-source control option

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 allowed me to develop 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 allowed me to quickly set up the arm and start developing the control software and perception algorithms for the project.

Issues with Igus Rebel motors' encoders and calibration

Throughout the development of the project, I encountered several issues with the Igus ReBeL cobot. The issues arose when the arm's motors' encoders were not providing accurate position data to the control software. This caused the arm not to move to

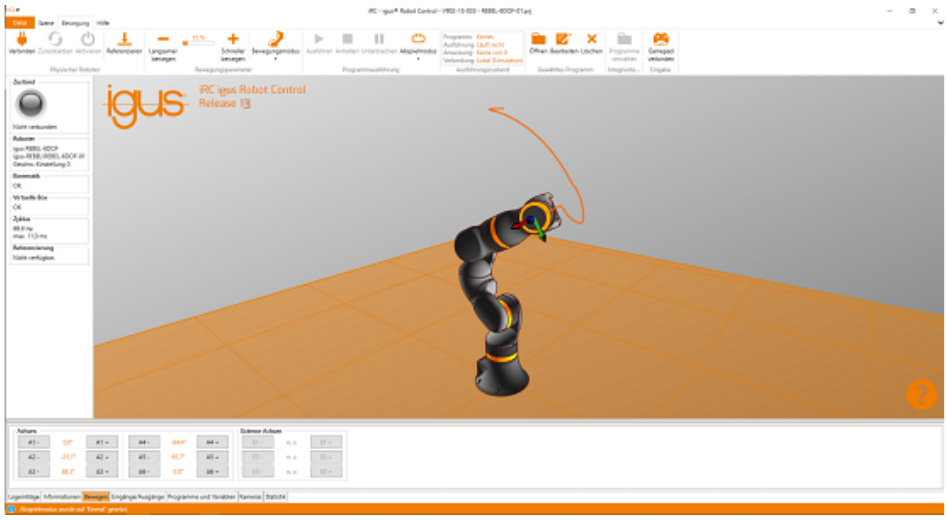


Figure 2.4: Igus ReBeL proprietary control software interface

.

the defined joint positions with an accuracy of the end effector of $\pm 0.1mm$, as specified by the manufacturer in the technical documentation. This was mainly due to both the calibration of the motors' encoders and the plastic gear components' backlash and play. The backlash and play in the gear components caused the arm's joints to have a certain amount of free movement before the motor started to move the joint. This free movement caused the arm's joints to be in an incorrect position, which was not detected by the motor's encoders. This resulted in the arm jiggling and vibrating when moving to a new joint position, and not reaching the desired position with a smooth velocity profile.

Furthermore, the main issue encountered was the **calibration of the motors' encoders**. This problem slowed down the development of the project since the software solutions for the calibration of the encoders and the digital zeroing of the joints were not successful in providing a working solution for the arm's accurate positioning. The calibration of the encoders was necessary to provide accurate position data to the control software, allowing the arm to move to the desired joint positions with high precision. The problem encountered was that the software for the automatic calibration of the motor controllers did not work as expected, resulting in errors and faulty calibration of the encoders. The only solution was to replace the entire robot arm with a new one, correctly calibrated by the manufacturer.

2.3. Sensors and Perception

The mobile robot platform is equipped with several sensors and perception systems, which are essential for the project's objectives. The sensors and perception systems are used

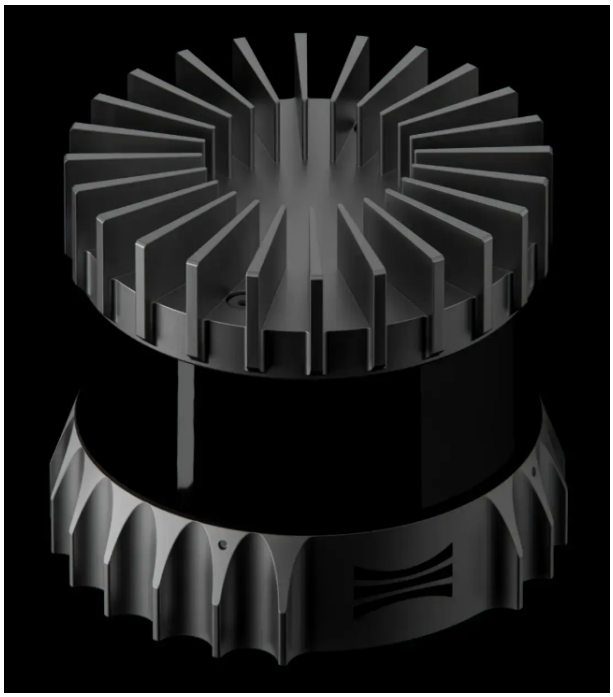


Figure 2.5: Ouster OS1-64 LiDAR sensor

.

for mapping and localization, obstacle avoidance, object detection and recognition, and manipulation tasks. Two main sensors are installed on the robot: a 3D LiDAR sensor and an RGB-D stereo camera sensor. The LiDAR sensor is in a fixed position, on top of the robot to have a 360-degree field of view, while the camera sensor is mounted on the robotic arm's wrist, allowing the camera to move with the arm's end effector.

2.3.1. 3D LiDAR Sensor

The main sensor for environment perception for localization and navigation is mounted on the robot's T-slot rails. The sensor is a *Ouster OS1-64* LiDAR sensor. 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.

Figure 2.6: Intel RealSense D435 RGB-D stereo camera

.

2.3.2. RGB-D Stereo Camera Sensor

The robotic arm is equipped with a *Intel Realsense D435* RGB-D stereo camera sensor. This camera is mounted on the *wrist* of the robotic arm, allowing the camera to move with the arm's end effector. The camera is used for object detection and recognition, and also for Aruco markers detection and pose estimation. This is the camera of choice for robotic applications, as it provides both RGB and depth images, which are essential for perception tasks in robotics. 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, even though the ROS2 driver provided by Intel reduces the resolution to 640×480 pixels at 30 frames per second. 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. Issues with Intel Realsense calibration

The camera provided by the laboratory was not calibrated correctly, especially the depth estimations were not accurate. In fact, the depth images provided depth estimations that were not consistent with the real-world distances of the objects in the environment. I was able to discover such issues when I started using the camera's depth sensor for the perception tasks of the project. I realized that the estimated distance from the camera and the Aruco markers (the distance estimated using geometrical calculations and the camera's intrinsic parameters) was **not consistent** with the estimated distance to the marker provided by the camera's depth sensor. This inconsistency was due to the camera's depth sensor not being calibrated correctly.

This issue was critical for the project, as the depth sensor was used for many perception

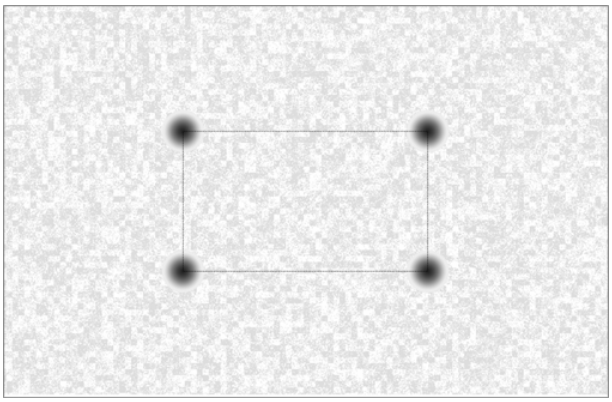


Figure 2.7: Calibration pattern used for the camera's depth image calibration

.

algorithms. I managed to solve this issue by calibrating the camera's depth sensor using the proprietary software provided by Intel: *Interl Realsense Self-Calibration Tool*, an application for **automatic on-chip calibration**. The calibration process was straightforward and required a few minutes to complete, even though many trials were needed to find the best possible calibration parameters. 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 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.

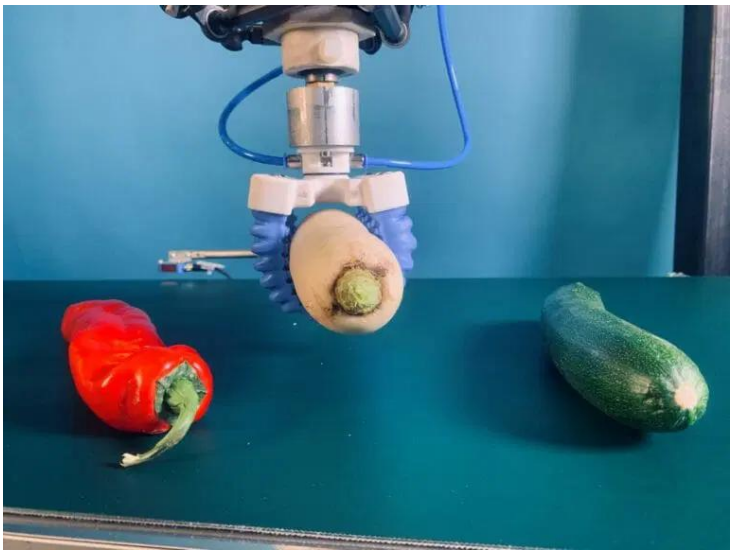


Figure 2.8: Soft Gripper Pneumatic Actuator handling vegetables

.

2.4. Soft Gripper Actuator

The robotic arm is equipped with a Soft Gripper Pneumatic Actuator from *Soft Gripping*. This gripper is composed of 3 soft fingers, which are actuated by a pneumatic pump. 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. It is also essential in food industries, where the gripper is used to handle delicate and fragile objects, such as fruits. The other advantage is that silicone is non-toxic and food-safe, making it suitable for handling food products.

The soft gripper is controlled by a pneumatic pump, which provides compressed air to the fingers, allowing them to open and close. The pneumatic system works at a pressure of 1 bar when the fingers are closed, and at a negative pressure of 0 bar when the fingers are opened. The pneumatic pump that controls the soft gripper allows the user to set the pressure value used when closing the fingers, but the pressure value cannot be changed dynamically via electronic control.

The pneumatic pump provided with the soft gripper has only one output tube used for providing positive pressure to the fingers. The negative pressure is provided by the environment, as the fingers are opened by the air pressure inside the fingers, which is lower than the air pressure outside the fingers. Other versions of the pneumatic pump provide also a negative pressure output tube, which allows the user to control the pressure value used when opening the fingers, but this feature requires an external vacuum compressor

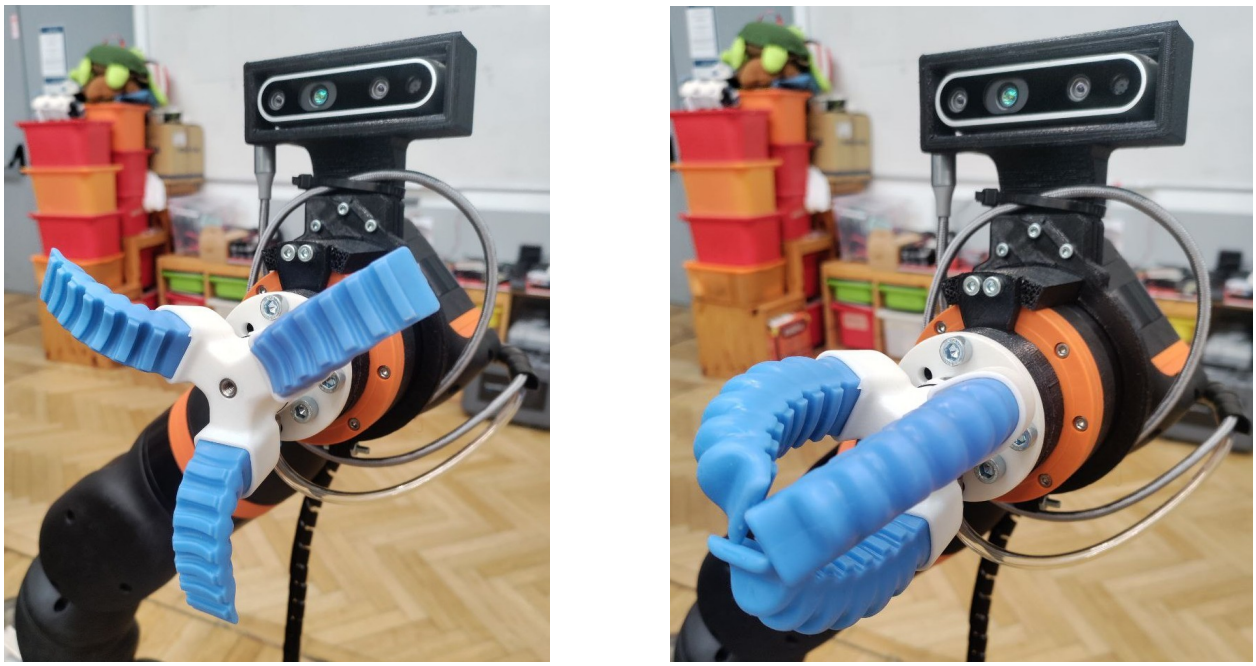
Figure 2.9: Pneumatic Pump control box secured on top of the mobile robot

.

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 strategically 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 pneumatic pump is provided without any power supply, so it was necessary to create a system to power the pump using the **onboard batteries**. The pump is powered by a 24V lead battery, which is the same battery powering the robotic arm. I created a system for powering both the robotic arm and the pneumatic pump with the same battery, using Molex cables and connectors. These Molex cables proved to be a reliable and optimal solution, as they allowed me to switch easily between the onboard batteries and the external cobot power supply. In fact, the cobot's 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 cable of the external power supply is also a Molex cable, so that's why I used Molex connectors and cables to connect the robotic arm and the pneumatic pump to the cobot's power supply.



(a) Soft Gripper opened

(b) Soft Gripper closed

Figure 2.10: Soft Gripper mounted on the end effector

The pneumatic pump must be controlled at 24V, as the pump's solenoid valve requires this voltage to operate. The pump provides 4 digital pins for controlling its operation, which are used to open and close the fingers. I created a simple system capable of controlling the pump's operation 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 4 different relays, each controlling a different digital pin of the 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 cheap and quick enough to switch between the digital pins of the pump, allowing an efficient control of the pump's operation, and the installation of a simple circuitry for the control system directly on the mobile robot platform.

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. The mounts were then 3D printed using the laboratory's 3D printer, which is a *Creality*

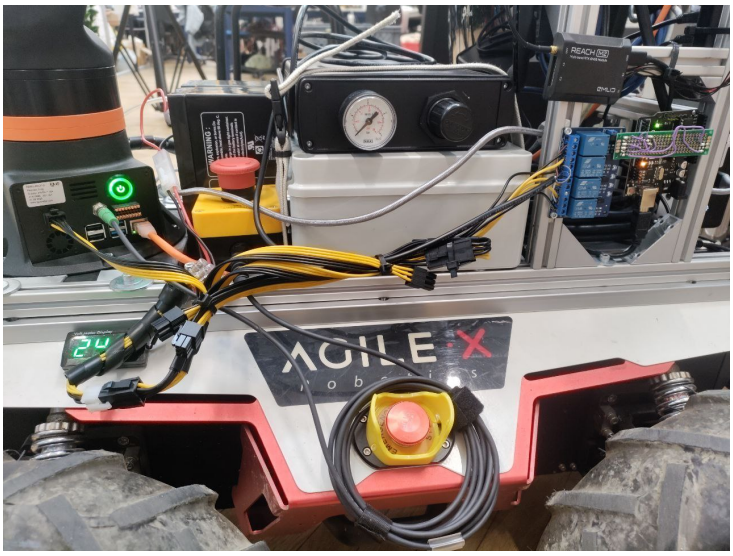


Figure 2.11: Molex connectors and power management for the cobot, pump, and relays

.

CR-10S 3D printer. I designed and printed two versions of the mount for the cobot's flange:

- *Mount V1*: 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*: 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.

I also designed and created another **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 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. I never used the GPS in my project, but the mount that I created was used by other researchers in the laboratory for their projects.

The 3D printer that I used in the AIRLab is a Fused Deposition Modeling (FDM) printer, 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

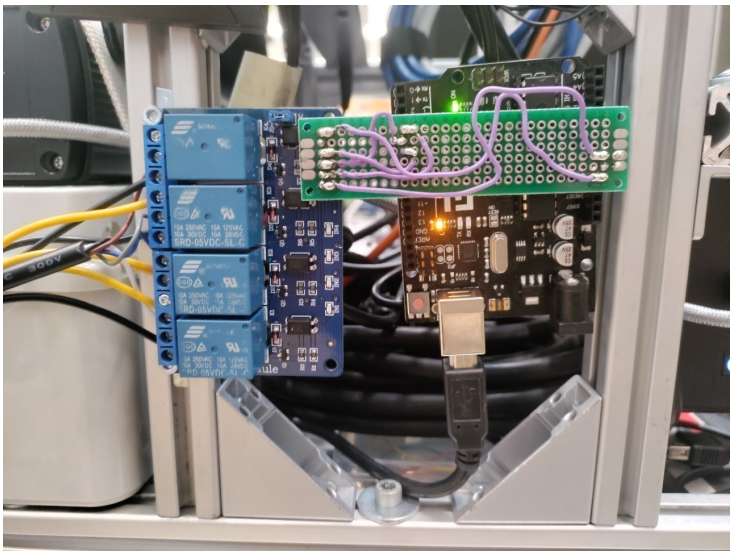


Figure 2.12: Arduino UNO microcontroller and relay module used to control the pneumatic pump

.

the mounts for these kinds of applications.

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 very robust and durable, to withstand the vibrations and shocks of the mobile robot platform. The mounts are not very easy to install and remove, since they 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. Mount V1

The first version of the mount, alias *Mount V1*, was designed to be robust and easy to switch with other mount extensions. The mount was designed to be lightweight and long, to ensure that the cobot's end effector would be able to reach objects not in the immediate vicinity of the robot. The mount was also designed to be 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 mount was also designed to be very robust and durable.

After many tests, the cylinder was then shrunk to a smaller length, to ensure that the cobot's end effector mobility and reach were not affected negatively. This allowed the cobot's end effector to reach objects in the vicinity with fewer limitations and constraints

Figure 2.13: GPS antenna 3d-printed mount on top of the LiDAR

.

on its orientation. The digital button placed on the tip 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.

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 the right distance 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. After many tests, I realized that the best configuration would be to mount the camera on top of the wrist, to enlarge the field of view. The Mount V1 was used for the "button presser demo", and replaced with its second improved version for the next demos.

2.5.2. Mount V2

The second version of the mount, alias *Mount V2*, was designed to be compact, robust, and more effective for the final versions of the project and the "soft grasping" demos.

Figure 2.14: 3d-printed mountV2 on the arm's wrist

.

The mount comprises 3 parts: the flange connector, the camera mount, and the soft gripper mount. The flange connector is used to connect the mount to the cobot's flange, ensuring that the mount is securely attached to the cobot. 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. 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** was designed from scratch because the flange provided by the cobot's manufacturer was not compatible with the soft gripper mount. The flange connector was designed to be lightweight and compact, to ensure that the 3d printing process would be fast and structurally sound. The flange connector was also designed to be robust and durable, to withstand the vibrations and shocks of the cobot's movements. 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 is designed to host 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.

2.5.3. Issues with the laboratory's 3D printer

The 3D printer in the AIRLab is a printer that is used by many researchers and students for their projects. In the past, very little maintenance was done on the printer, and the printer was not calibrated correctly. This caused many issues with the 3D prints, such as **stringing, and under-extrusion**, which affected negatively the quality of my prints. Therefore I had to carry out maintenance on the printer to fix these issues. I calibrated the printer's bed and the extruder, to ensure that the prints were of high quality and accurate. I substituted the nozzle with a new one, and unclogged the hotend, to ensure that the filament was extruded correctly and in the right amount. I also cleaned the printer's bed and the extruder, to ensure that the prints were sticking correctly to the bed. After these maintenance operations, the printer was working correctly, and the prints were of higher quality, even though they were still far from being perfect. I was not able to clean the printer's extruder gears and the filament feeder, as they were not accessible to me due to some screws being stripped.

2.6. Batteries and Power Management

The mobile robot's internal battery is capable of powering all the sensors and computational units that are mounted. To meet the specific voltage and current requirements of these devices, two DC/DC converters are employed: - 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 sensor.

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** with 9Ah of power capacity, which provide 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 2 batteries available in the laboratory, which can be switched easily when one of them is discharged.

Another tool used to power the robots is a *Santino*, a holy figure card presenting *Saint Staianet*, the patron saint and protector of the mobile manipulation robot. This figure, placed in the front, helped and protected the robot during the development and testing of the project, ensuring that everything was working fine and smoothly. The Santino was

Figure 2.15: Lead batteries mounted onboard for the cobot and pneumatic pump
.

also used to provide a spiritual and religious touch to the project, ensuring that the robot was blessed and protected during its missions.

2.7. Complete Mobile Manipulation Setup

Figures 2.19 and 2.18 show the complete mobile manipulation setup, with the robotic arm mounted on top of the SCOUT 2.0 robot, and the soft gripper mounted on the robotic arm's end effector. The setup is ready for the "soft grasping" demos, where the robot is tasked with grasping and manipulating objects in simulated agricultural environments.

Figure 2.16: DC/DC converter used to power the robot's sensors and computer

.

Figure 2.17: Santino

.

Figure 2.18: Front view of the robots

.

Figure 2.19: Lateral view of the robots

.

3 | Software Architecture and Simulation Environments

This chapter discusses the software architecture of the mobile manipulation robotic 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 the algorithms before deployment on the real robots.

3.1. ROS2 Control Interface for Igus Rebel Arm

The first step in developing the software architecture for the mobile manipulation robotic system 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 manufacturer *Commonplace Robotics*, but the requirement was the development of a ROS2 control interface for the Ethernet interface since the CAN bus interface requires a proprietary cable, which is not available to me. Using the Ethernet interface makes it easier to plug it into the switch and control the arm from 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 string messages. The CRI protocol is used to send commands to the arm in the form of joint positions or velocities (jogs) and to receive feedback from the arm in the form of joint positions. Controlling the cobot using ROS2 requires a hardware interface, used to command and control the robot by interfacing with the hardware via the **CRI protocol**. The hardware interface is implemented as a ROS2 lifecycle node that is interfaced directly with the Joint Trajectory Controller,

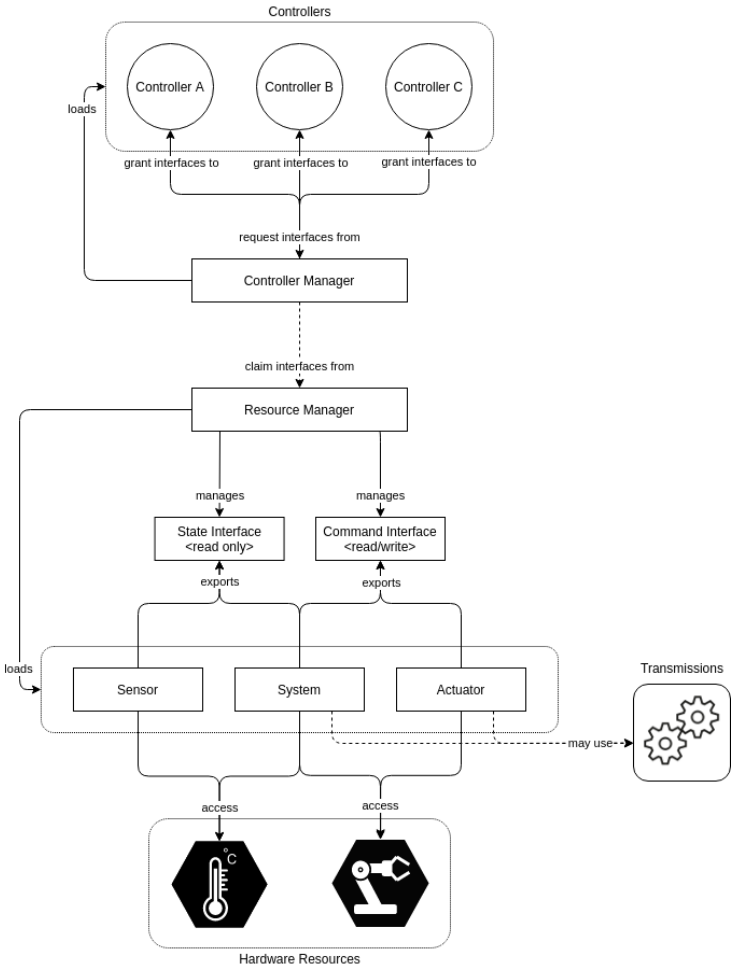


Figure 3.1: ROS2-Control Architecture example

which is a ROS2 controller that can be used to control the arm using joint trajectory messages. This 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].

The diagram in figure 3.1 shows an overview of how *ROS2-Control* works. The hardware interface implemented for controlling the Igus ReBel arm is a ROS2 lifecycle *System Interface*, 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 controller manager employed is provided by MoveIt2, in order to handle the cooperation between the hardware interface and the joint trajectory controller, which receives the computed trajectories from MoveIt2.

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 mechanism allows. Waypoints consist of positions, and optionally velocities and accelerations.

It can be operated either in position, velocity, or acceleration mode, depending on the type of trajectory message that it handles. 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 (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 type of 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 the desired performance of the arm. I did not manage to find the best gains for the PID controller, for a stable and smooth trajectory execution, so I used the open-loop mode for controlling the arm.

MoveIt controller managers, somewhat a misnomer, are the interfaces to your custom low-level controllers. A better way to think of them is controller interfaces. The included *MoveItSimpleControllerManager* is sufficient for the robot controllers that provide ROS2 actions for *FollowJointTrajectory*.

3.2. MoveIt2 and RViz2 Simulation Environment

MoveIt2 is a ROS2 framework 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 trajectories for the arm. MoveIt2 is based on the ROS2 middleware and provides a flexible and modular framework for developing robotic applications.

The figure 3.2 shows the general **architecture of MoveIt2**. The MoveIt2 framework

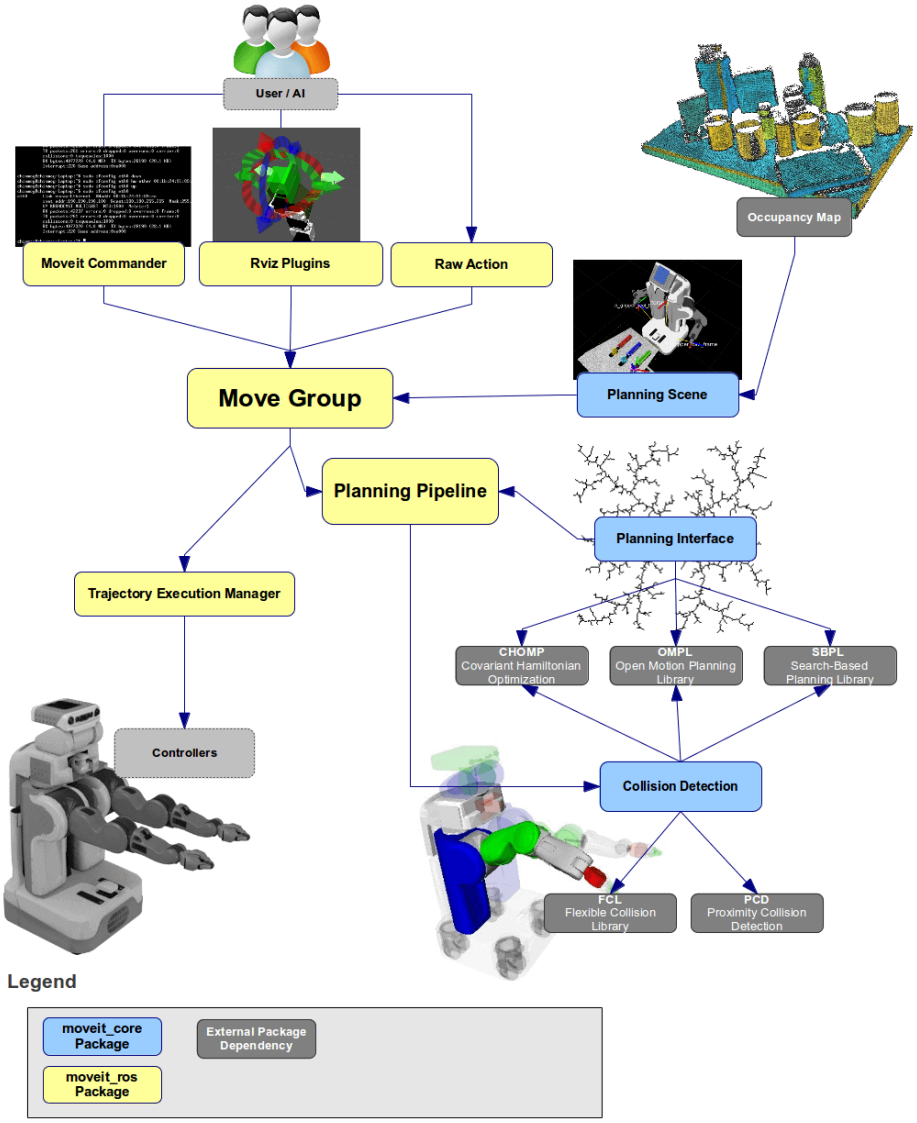


Figure 3.2: MoveIt2 General Architecture

consists of several components, including the Planning Scene, the Planning Pipeline libraries, the Kinematics Solver, the Collision Checker, the Trajectory Execution Manager, and the 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 the architecture of the Planning Scene and the modules with which it interacts. 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

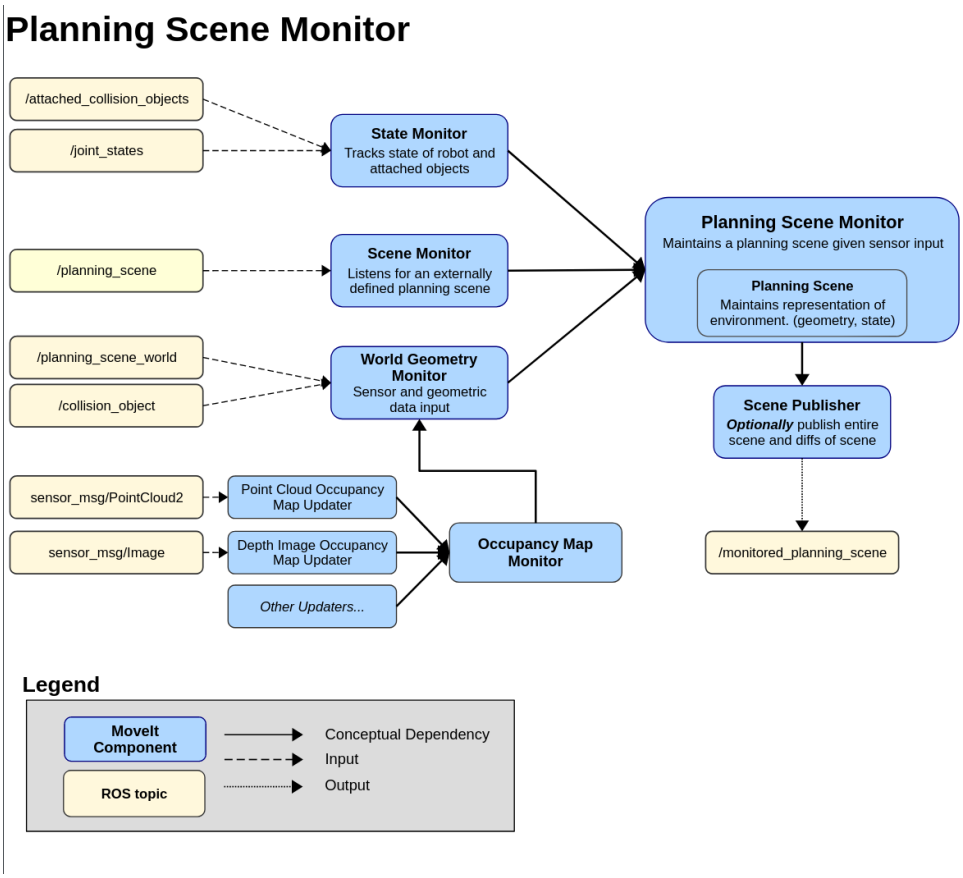


Figure 3.3: Planning Scene and Occupancy Mapping Architecture

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, which is a high-level interface to the MoveGroup node, which is the class that 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, which is a 3D visualization tool for ROS2 that

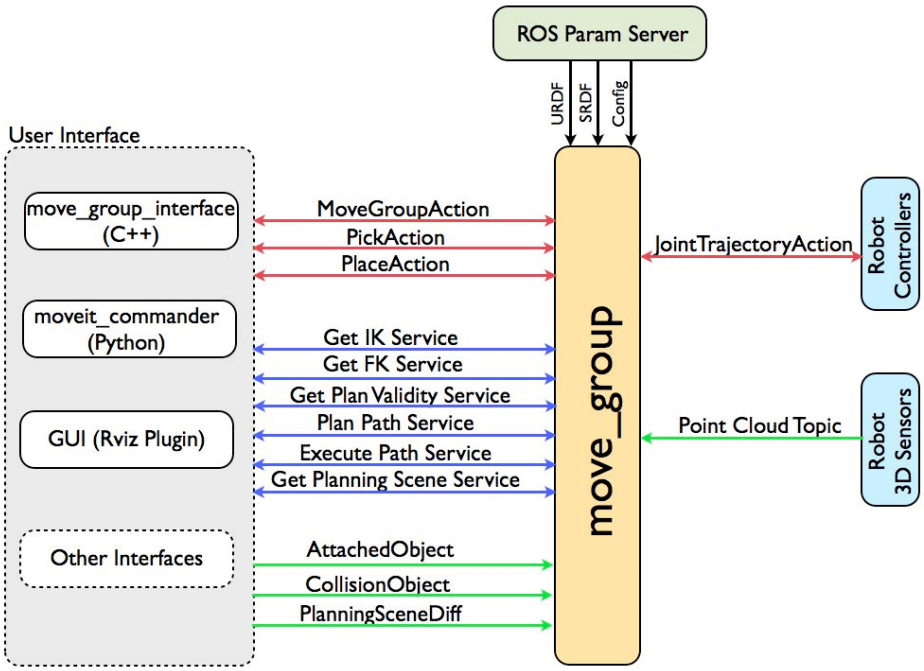


Figure 3.4: MoveGroup Interface

can be used 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.

I developed a custom library that 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 makes use of different planners to obtain the highest probability of generating a valid motion trajectory given the planning scene, and current and target joint positions. The planner *Pilz Industrial Motion Planner* is used for linear cartesian trajectories since it outperforms the other planners for this task. 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++

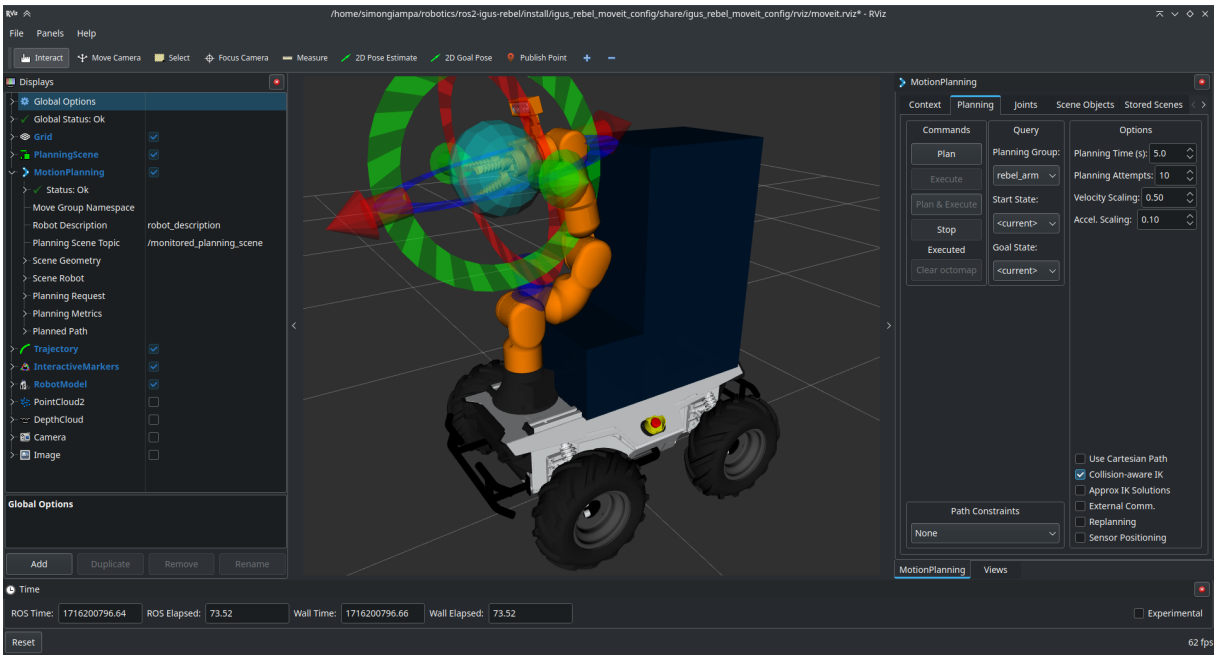


Figure 3.5: RViz2 Interface with MoveIt2 Control Panels and the mobile manipulation robot.

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.

3.3. Robotic Arm Visual Servoing

The MoveIt2 library provides also a framework for visual servo-ing, which is 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. I developed the code to integrate these tools with the camera visual feedback, in order to perform visual servo-ing experiments with the Igus Rebel Arm.

The servo-ing algorithms provided by MoveIt2 allow also for **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 allows the user to control the robot's end-effector pose using camera visual feedback in

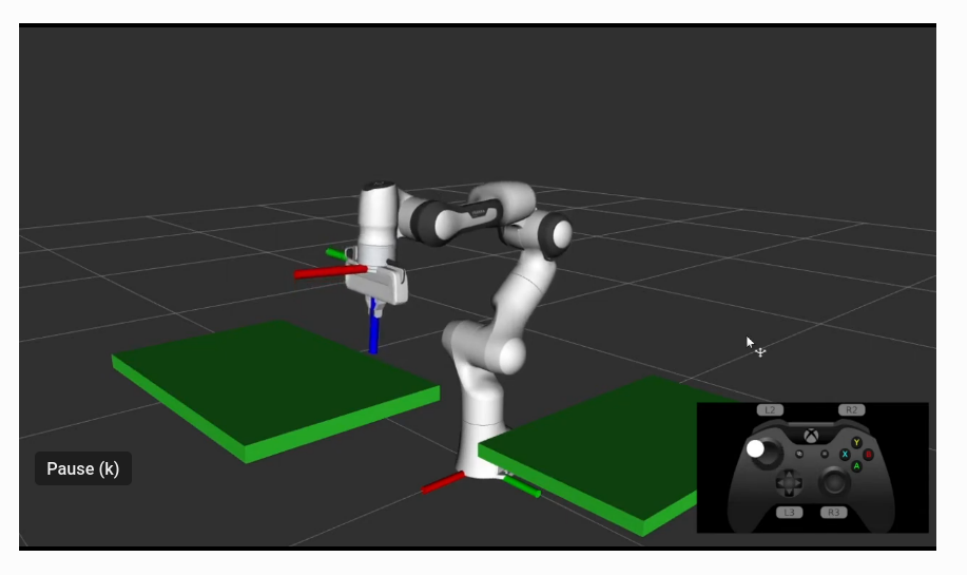


Figure 3.6: Teleoperation with a joystick controller in RViz2 with the Franka Emika Panda

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.

After many trials and errors, I managed to get the **visual servo-ing** working only with input cartesian poses close to the initial pose of the arm. The servoing algorithm requires a good initial guess of the end-effector pose, in order to converge to the desired pose. Since I tested the visual servo-ing with the arm in the open-loop mode, the servo-ing algorithm was not able to compensate for the errors in the joint positions, which led to the arm not reaching the desired pose. Then I switched to the closed-loop mode, but the PID controller gains were not tuned properly, which led to strong oscillations in the joint positions and the arm not reaching the desired pose stably. I did not manage to find the best gains for the PID controller, so I had to abandon the visual servo-ing experiments with the arm. The package is still available in the repository, but it is not used in the final implementation of the mobile manipulation robotic system. Furthermore, the servo-ing algorithms using cartesian input poses rely exclusively on ROS2-Iron distribution.

3.4. Collision Avoidance with Octomap

The MoveIt2 library provides a framework for collision avoidance using the **Octomap** library, which is 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

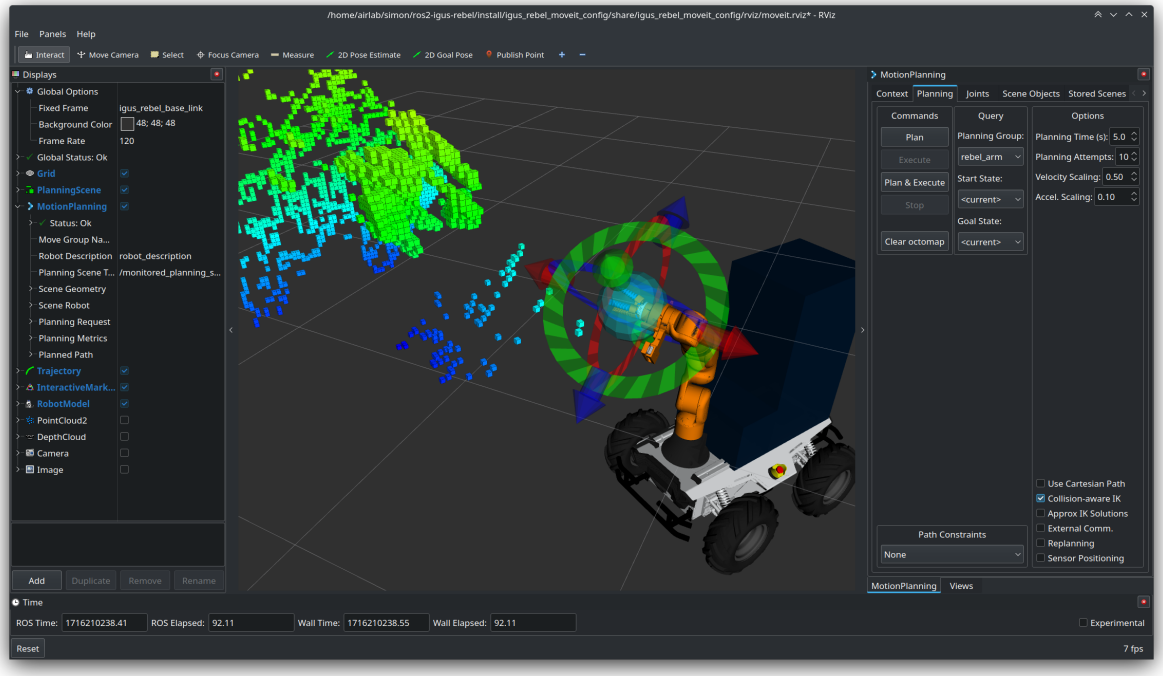


Figure 3.7: Mobile robot using the depth camera to construct the Octomap with the obstacles nearby

to each voxel, indicating whether it's occupied (by an obstacle) or free. MoveIt2 integrates data from various sensors, such as the depth camera 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, so the motion planning algorithms can use the voxels to plan a trajectory that avoids them.

MoveIt2's collision checker uses the 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 the 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 the MoveIt2 interfaces, resulting in a **poorly optimized** Octomap probabilistic representation of the environment. In fact, the library takes track of the free space as well as the occupied space, resulting in a memory and computationally intensive update process, meaning that the time taken for each Octomap update is very high. The Octomap library for ROS2 is a porting of its ROS1 counterpart, which is fully functional but still lacks some features. One of the features that are missing is the ability to update the Octomap in realtime when part of the environment changes. 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, 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. The serial communication handles with the UART protocol, which is used to send and receive string 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 pin is used to close the gripper, while the RELEASE pin is used to open the gripper.

3.6. 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, I decided to use Nav2 as the primary navigation stack for the mobile manipulation robotic system. The Nav2 stack 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 stack 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 stack. I created and configured the Nav2 stack to work with the AgileX Scout robot, using the SLAM Toolbox algorithm for creating maps of the laboratory. Figure 3.8 and figure 3.9 show the autonomous navigation of the robot in the hallways and AIRlab in building 7 of Politecnico di Milano, respectively.

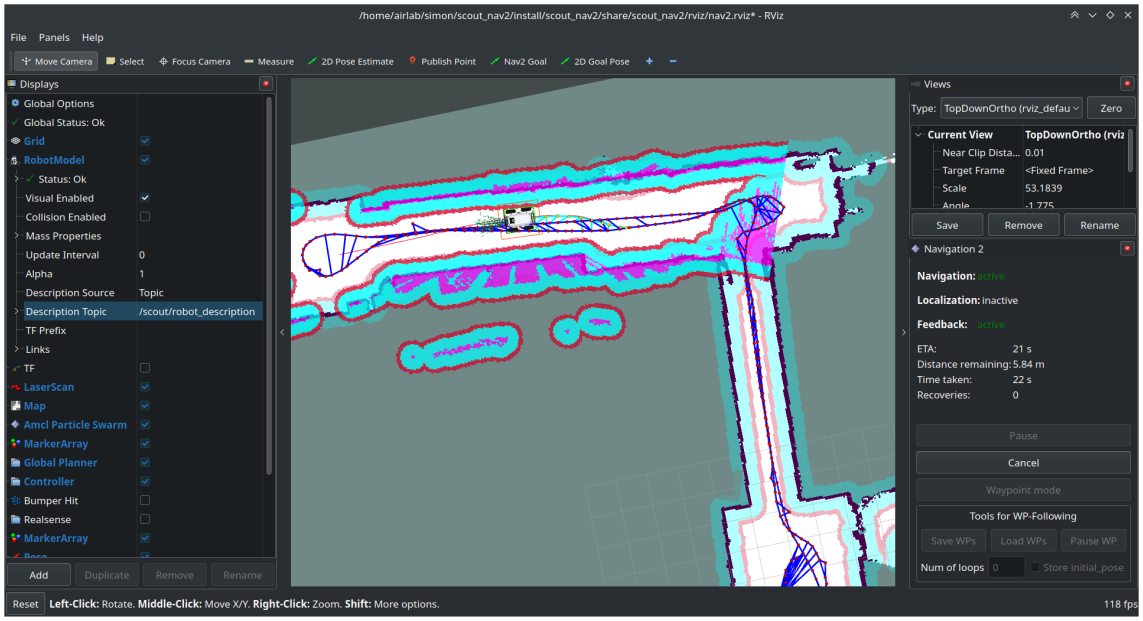


Figure 3.8: Autonomous Navigation with Nav2 in the hallways of building 7 of Politecnico di Milano

3.6.1. Nav2 Parameters Tuning

Finding the right parameters for the Nav2 stack was a challenging task, as the stack has many parameters that need to be tuned to achieve optimal performance. The parameters to be set are related to all the plugins that are part of the Nav2 stack. Many tests in different and complex environments were performed to find the best parameters suitable for the AgileX Scout robot. The parameters were set based on many days of intense testing in both simulated and real environments, to ensure that the robot can navigate safely and efficiently in different scenarios.

Localization Algorithms: I tested and applied both AMCL and SLAM Toolbox algorithms for localization, and I found that the AMCL algorithm performed worse in terms of scan matching when the robot rotates in place. Therefore, I decided to use the SLAM Toolbox algorithm for localization, which provided better performance and improved reliability in terms of localization accuracy, even in maps with dynamic obstacles and changing environments. The pose graph generated by the SLAM Toolbox algorithm is displayed in the maps, as shown in figure 3.8 and figure 3.9, where the blue lines represent the pose graph created during map creation, and the cyan lines represent the robot's path traversed during navigation, estimated by the algorithm, and connected to one of the nodes in the pose graph.

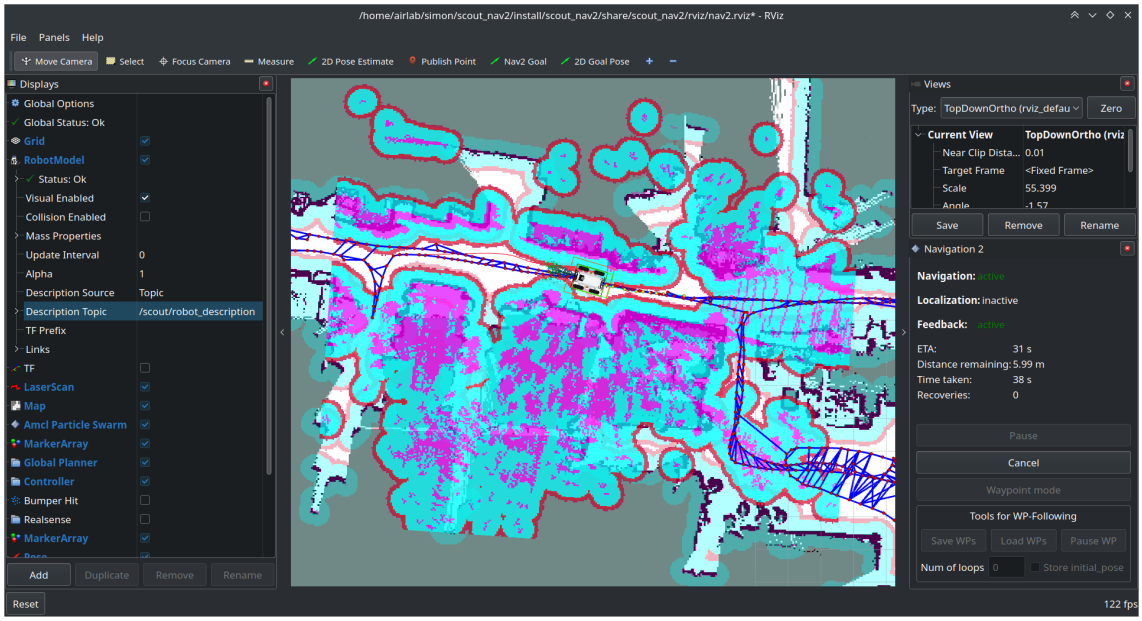


Figure 3.9: Autonomous Navigation in cluttered and dynamic environment with moving obstacles (AIRlab)

Global Planner: I tested and used the SMAC planners [13], in particular the Hybrid A* global planner. I found that it performed 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 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. After many tests, I found that the Reeds-Shepp algorithm enabled to generate 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, for some reason still unknown to mankind and Steve Macenski, the Reeds-Shepp algorithm works only in the real environment, and not in the simulated environment in Ignition Gazebo.

Local Planner (Controller): I tested and applied the DWB and MPPI local planners, and I found that the MPPI local planner performed better in tracking the global plan and avoiding tight obstacles. The MPPI (Model Predictive Path Integral Controller) [32] local

planner uses a **model predictive 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, 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 3.8 and 3.9, in front of the robot's footprint, showing the robot's sampled trajectories to follow the global path.

Recovery Behaviors: I tested the default recovery behaviors provided by the Nav2 stack, and I found that the default recovery behaviors performed well in recovering the robot from unexpected scenarios, such as getting stuck or colliding with obstacles. So I decided to modify slightly the default recovery behaviors to improve the robot's performance in recovering from the scenarios where the robot would be stuck near an obstacle and unable to move. 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 obstacle in the environment, which is represented as pink areas in the costmap, as shown in figure 3.9. The unknown obstacle is 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. I later substituted this layer 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 3.8 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 the nav2 stack 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 the *SLAM Toolbox* algorithm, which is a 2D SLAM algorithm that uses the robot's LiDAR sensor to create a 2D map of the environment [14]. The SLAM Toolbox algorithm is used to create a 2D map of the environment, which is 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. I managed to use it correctly in both simulated and real environments, and it provided good performance in terms of mapping accuracy and reliability.

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 smooth trajectories from the computed 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 the Nav2 stack, 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 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 3.10, 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 the Nav2 stack. Figure 3.9 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.

3.6.2. Issues with 3D LiDAR for autonomous navigation

Throughout the development of the project, I encountered several issues with the NAV2 stack. Sometimes the software crashed at random times, unpredictably, and without any apparent reason. Some other times the dynamic obstacle avoidance algorithm did not work as expected, causing the robot to collide with obstacles that were not perceived by the LiDAR sensor in the surrounding environment. Overall, the robot's navigation stack was not reliable and robust, which was a significant issue for the project's development, especially for the autonomous navigation tasks in the laboratory's environment (cluttered and dynamic). Figure 3.11 shows the pointcloud generated from a cluttered environment and the expected view from a correctly functioning sensors.

After many trials and tests, my supervisor and I discovered that the LiDAR pointcloud data's low and unreliable frequency was the main cause of all these problems. The LiDAR sensor works at a nominal frequency of 10Hz. The LiDAR sensor was not providing a stable scan rate of 10Hz, but it was fluctuating between 3Hz and 8Hz, and only in a few cases, it was able to provide the nominal 10Hz scan rate. Once we discovered the source

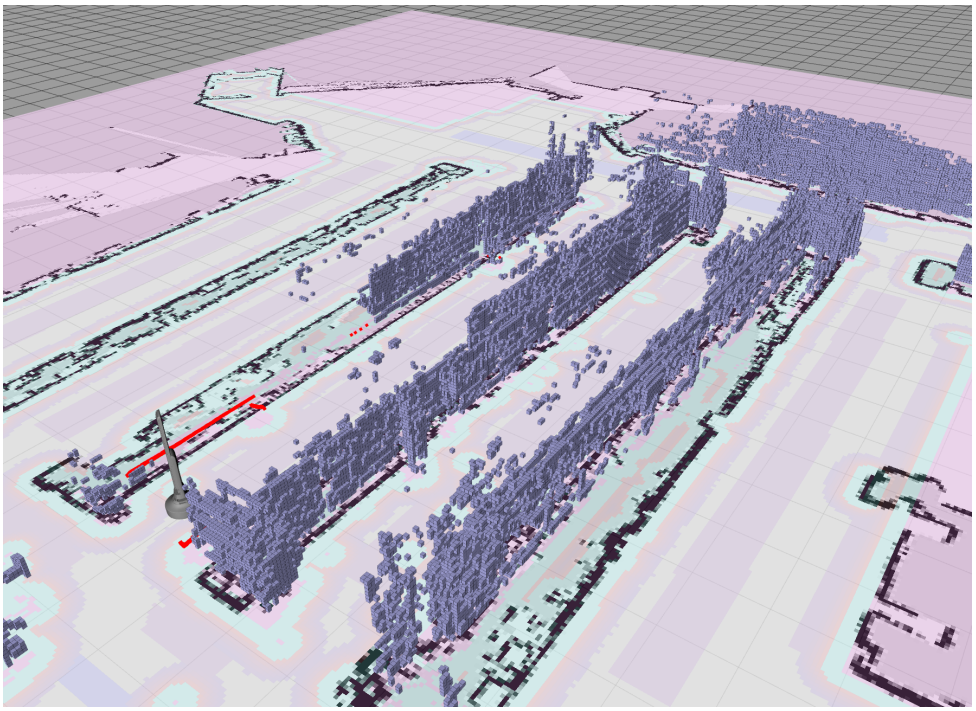


Figure 3.10: STVL in action in a dynamic and cluttered warehouse environment

of the problem and ensured that the LiDAR sensor was working correctly (meaning that the sensor was not malfunctioning or working at a lower but valid data rate), we tried to find the culprit of the low and unreliable frequency of the LiDAR sensor.

After many attempts, I found out the cause of the problem: the culprit was the **router** used to connect the robot's computer to the laboratory's network and the internet and the DDS (*Data Distribution Service*) middleware used internally by ROS2 for the communication between the nodes in a local network. Basically, the DDS middleware was configured to broadcast all ROS2 packets and data streams to the laboratory's network, causing delays in the transmission and packet loss, as the network was not able to handle the high-frequency and heavy-weight data streams of the LiDAR sensor. In fact, by just unplugging the router from the robot's computer, the LiDAR sensor was able to provide a stable scan rate of 10Hz, without any fluctuations or drops in the frequency, since no broadcast packets were sent to the network. Since the router was necessary for the remote control and monitoring, it was simply just not an option to unplug it from the robot's computer.

For the correct operation of the localization and obstacle avoidance algorithms, it is fundamental that the LiDAR sensor works at a **stable frequency**, without any fluctuations or drops in the scan rate. This requirement is essential because these algorithms rely on the coherent and continuous data stream for time interpolation and filtering of the point-

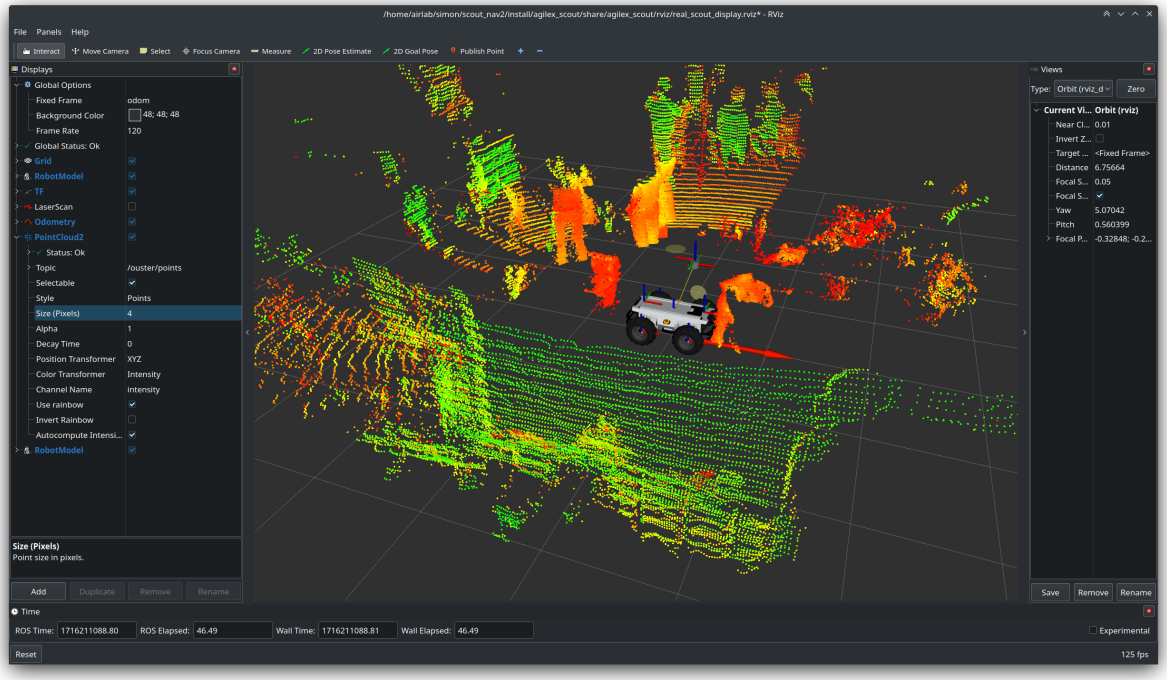


Figure 3.11: 3D LiDAR sensor in a cluttered environment

cloud data. This is why the LiDAR sensor's low and unreliable frequency was causing the software to crash and malfunction.

The solution was to configure properly the **DDS middleware** to ensure that all ROS-related nodes and data streams were handled inside the robot's computer only, and not going through the router and the laboratory's network. Having the data streams in the local machine only, ensured that the LiDAR sensor's data was not distributed among the network, causing delays in the transmission and packet loss, which were the main causes of the low and unreliable frequency of the LiDAR sensor. Furthermore, we configured ROS2 to use the *Cyclone DDS* middleware, which is a lightweight and fast DDS implementation, suitable for real-time and high-frequency data streams. This DDS implementation was able to handle heavy-weight data packets at high frequencies, ensuring that the LiDAR sensor's data was transmitted and received correctly in its entirety, without any losses or delays in the transmission.

After this configuration, the LiDAR sensor was able to provide a stable scan rate of 10Hz, and the software stack was working correctly, without any crashes or malfunctions. Furthermore, I was also able to test the LiDAR sensor at a higher frequency of 20Hz, with lower resolution, and it was working fine and stable. This solution was essential also for the correct operation of the IMU sensor for the SLAM algorithm, which required a stable

and continuous data stream of the LiDAR at the moment of startup of the sensor in the software stack. Before the fix of the LiDAR sensor's frequency, the IMU sensor was not able to start correctly, due to the lack of data from the LiDAR sensor, and the computed odometry data was drifting too much to be useful. Many other algorithms and software benefitted from this fix. The robot's navigation stack was now reliable and robust, and the autonomous navigation tasks were working correctly and smoothly.

3.7. 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 new version of the Gazebo simulator, which is widely used in the robotics community for simulating robots and sensors for perception systems. **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 stack and the navigation algorithms, as they allowed for testing the robot's behavior in different scenarios and environments. It allowed me to tune the navigation stack's parameters thoroughly and ensure the robot would avoid obstacles and navigate safely.

The simulated environment in Ignition Gazebo was a **warehouse** 3.12 with various obstacles such as walls, shelves, and boxes, which the robot had to navigate around to reach its goal. 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 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.

I was able to simulate also the sensors mounted on the robot, such as a 2D LiDAR and a 3D LiDAR sensor, which were used for perception and obstacle avoidance. I also tested the robot's localization algorithms, such as AMCL and SLAM Toolbox, using the simulated odometry data created by the robot's wheels' motion in the simulated environment. Figure 3.13 shows the simulated mobile robot navigating in the warehouse environment while avoiding the boxes and shelves in its path.

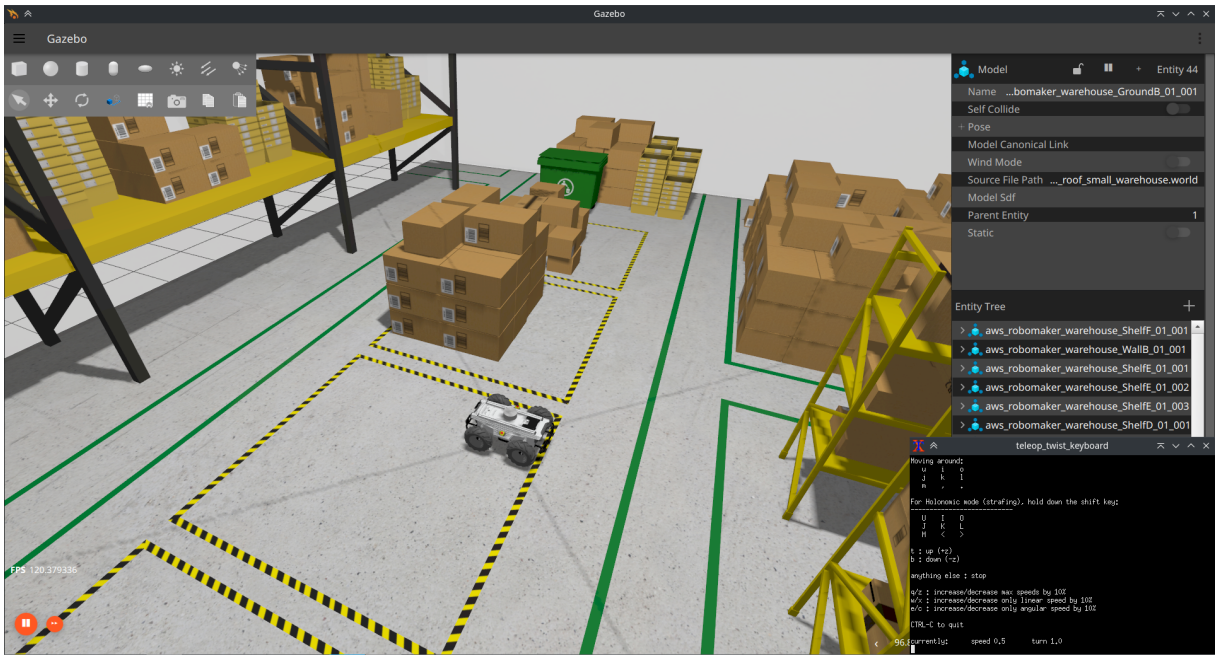


Figure 3.12: Ignition Gazebo Simulation Environment

3.8. Parking Algorithm for Mobile Robot

I designed a **parking algorithm** for the mobile robot to autonomously park near a target spot in the laboratory. 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 parking pose next to the target pose where the mobile robot must park to give the robotic arm enough space to reach the target. I had to structure and design the algorithm to ensure that the mobile robot could park in a position where the robotic arm could 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 the robotic arm's workspace to ensure that the robot can park in a position where the robotic arm can reach the target object. The algorithm also considers the costmap generated by the navigation stack 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 to the robot's navigation stack, 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

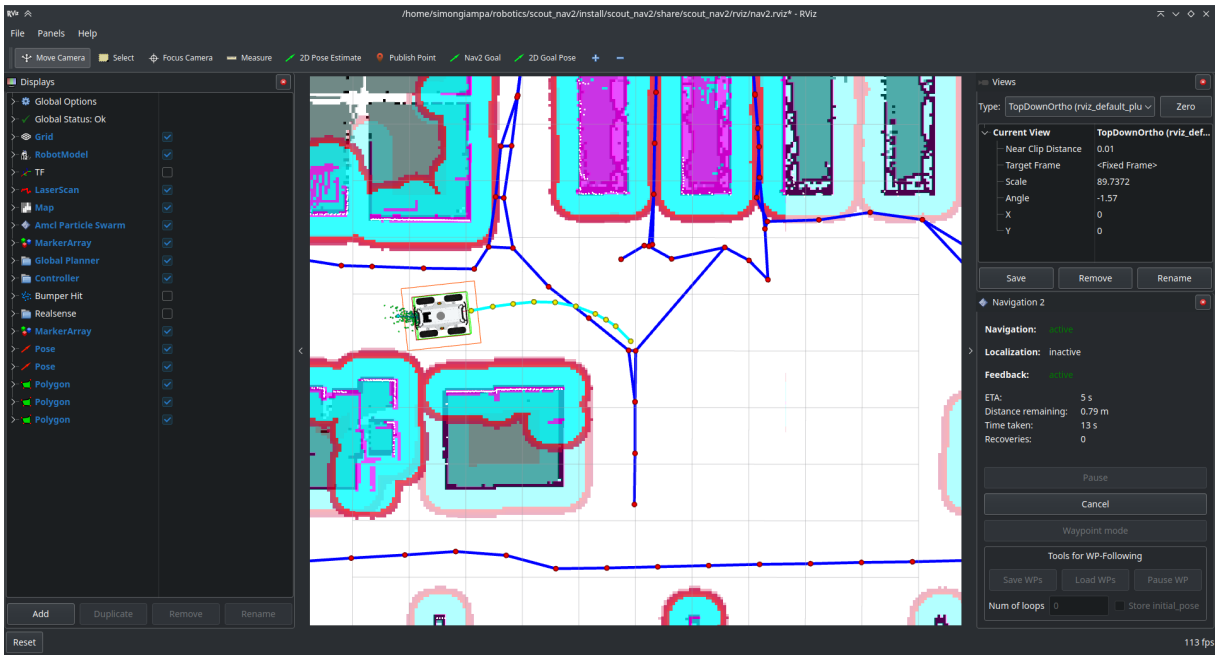


Figure 3.13: Mobile robot navigating in the simulated warehouse environment

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 is explained 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

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 cannot consider the 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. 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.

3.9. Aruco Marker Detection and Pose Estimation

Aruco markers are synthetic 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) and orientation (rotation) with respect to the camera. The function for pose estimation takes the detected marker corners, the marker size, and the camera's intrinsic parameters (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. I had also to calibrate the camera's intrinsic parameters using a calibration pattern and the automatic calibration software tool provided by the RealSense SDK.

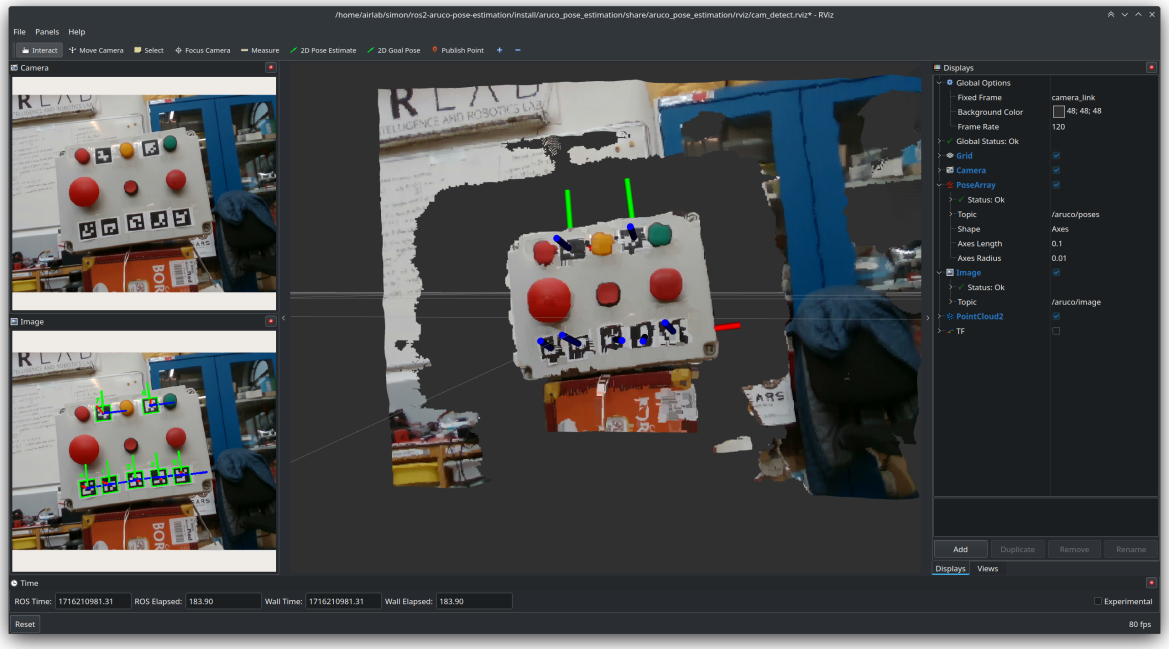


Figure 3.14: Multi-Aruco Plane Estimation algorithm in action. The screenshot shows RViz2 displaying on the top left corner the input image, bottom left the image with the detected markers drawn on top of it, and in the center the colored pointcloud captured by the depth sensor.

3.9.1. Multi-Aruco Plane Estimation Algorithm

One of the challenges I faced was estimating 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 of the marker. This is especially true for small markers that appear far away from the camera. To address this issue, I developed a multi-Aruco plane estimation algorithm that estimates 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 that I implemented 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:

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
        $c =$ 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

3.10. Object Detection with YOLOv8

For the project, it was also necessary to detect objects in the environment, such as colored balls, which the robot had to grasp and move to a different location. For this task, I used the YOLOv8 object detection algorithm, trained on a custom dataset that I collected and annotated. Figure 3.15 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 incredibly fast and efficient. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously, achieving impressive 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 my applications. 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.

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. I used this model for my object detection task, such as detecting colored balls from the camera feed. The YOLOV8 architecture is shown in figure 3.16. 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 the default one (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.

I trained the YOLOv8 model on a custom dataset of colored balls, which included images of the balls from different angles, distances, and lighting conditions. I collected the dataset using the RealSense camera mounted on a tripod, which allowed me to capture images of the balls from different perspectives and distances. The dataset was annotated manually with the bounding boxes and class labels of the balls, which were used to train the YOLOv8 model. I used the YOLOv8 model provided by **Keras**, which is a high-level deep-learning library that provides a user-friendly API for building and training neural networks. Since the training dataset I collected is of small size, I used data augmentation techniques 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. I included also other types of image augmentation techniques, 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, which is a powerful image augmentation library for computer vision tasks, specially designed for object detection tasks. It is essential to adjust properly 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, I used a batch size of 16, which is enough for a small dataset such as the one that I created, 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 2 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 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, I switched to the *Binary Penalty Reduced Focal CrossEntropy* loss function, which is 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.

3.11. Pose Estimation with Object Detection and Depth Sensing

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.

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. 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. This center position is then used to compute the optimal grasping pose for the robot to interact with the object.

This perception algorithm is used to estimate the optimal grasping pose for the robot's end-effector to grasp the object. The algorithm was first designed to work only with colored balls (of spherical shape), but I extended it to work with apples too, which are more complex objects with a more irregular shape.

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

I leveraged the ROS2 Actions client-server architecture to implement high-level tasks for the mobile manipulation robotic system. The Action architecture is a powerful and flexible way to define and execute complex tasks in a distributed system, such as a robotic

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 client sends a goal to the server, which executes the task and returns the result to the client. 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 that it decouples the task's definition from its execution, allowing for easy reusability and extensibility of the tasks 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. I used this architecture to implement high-level tasks:

- **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.

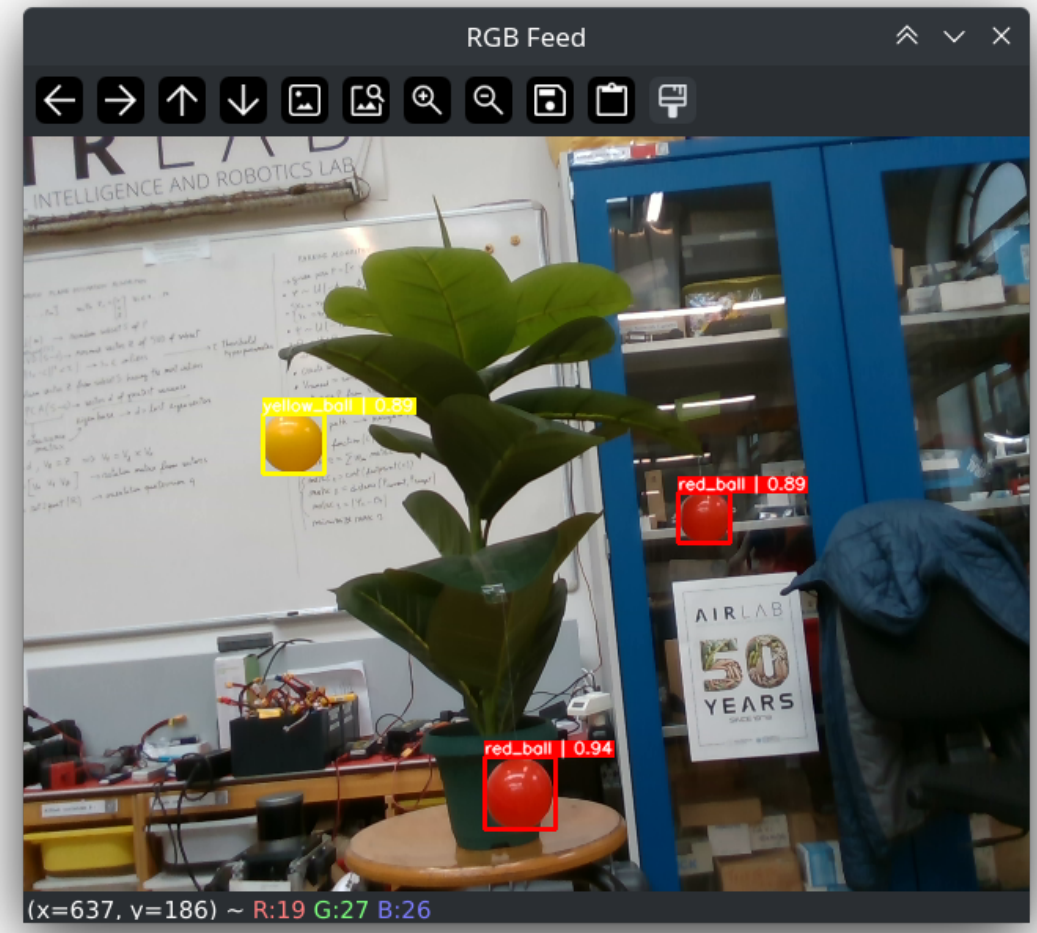


Figure 3.15: YOLOv8 detecting colored balls in realtime from the RGB camera feed. The screenshot displays the image with the bounding boxes colored with the same color as the predicted class. For each prediction, the class probability is associated.

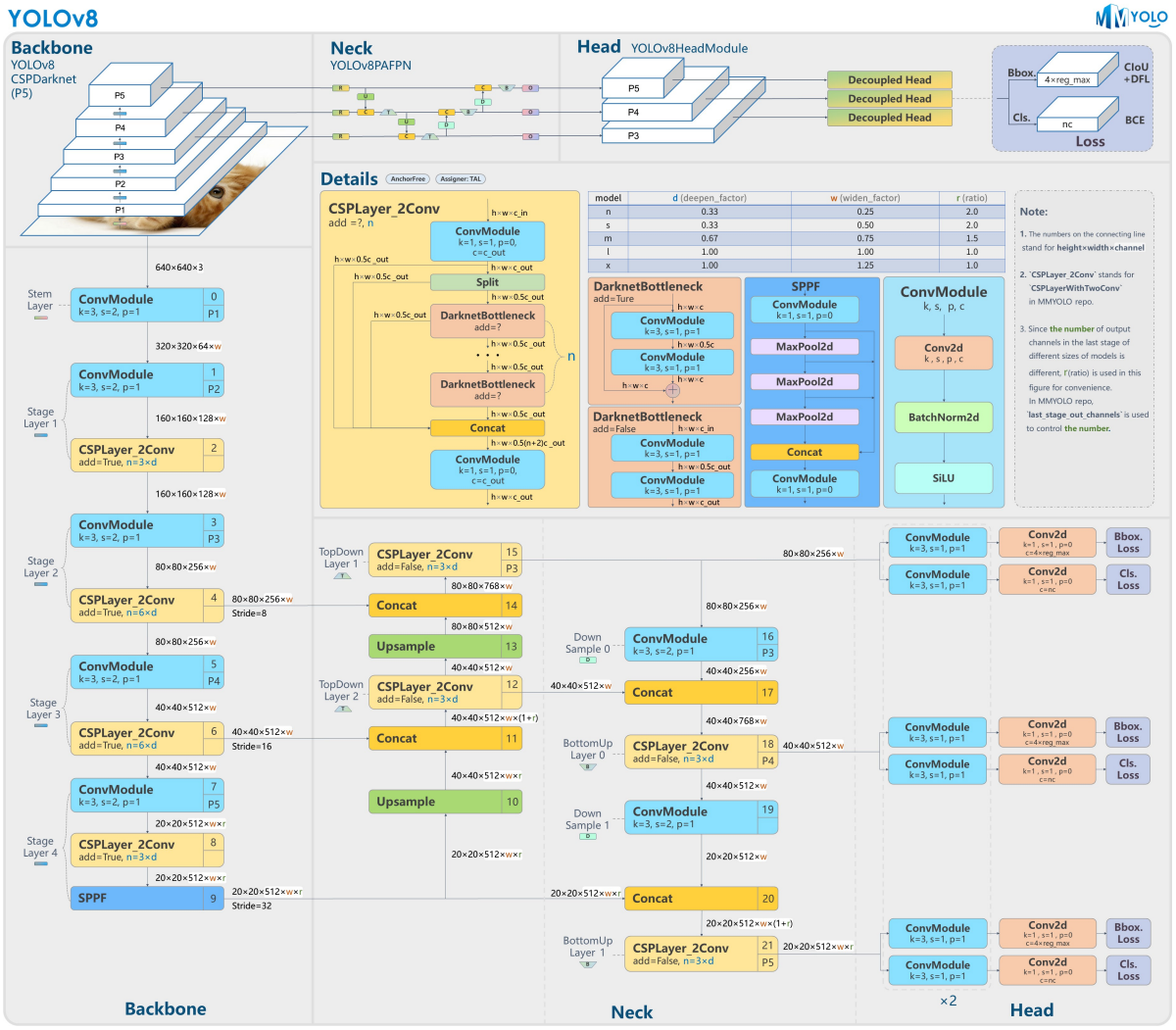


Figure 3.16: Architecture overview in detail of the YOLOv8 architecture

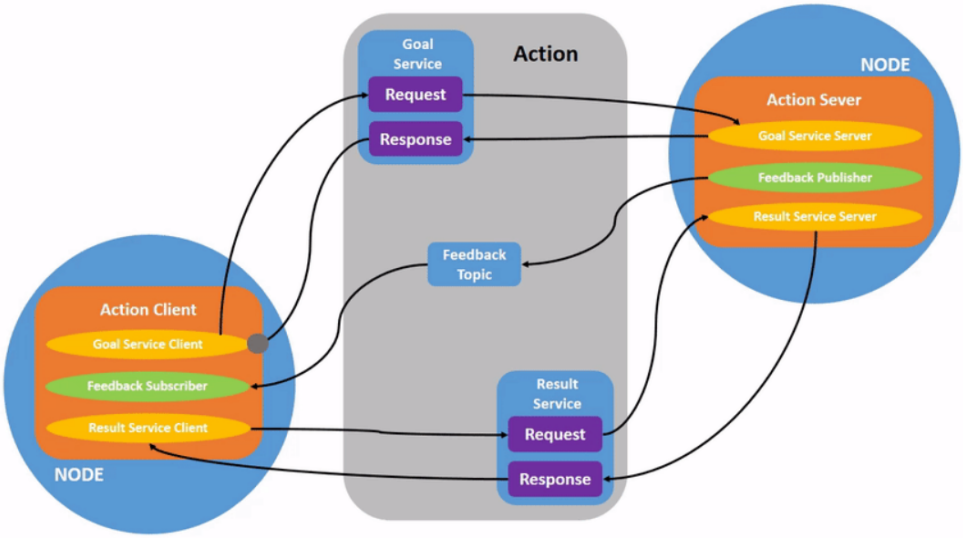


Figure 3.17: ROS2 Actions Client-Server Architecture

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].