POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering and
Master's Degree in Computer Engineering

Politecnico
di Torino

Master's Degree Thesis

ROS-based autonomous navigation and object recognition for a mobile manipulator operating in a warehouse environment

Supervisor	Candidates
Prof. Marina INDRI	Antonio RAGAZZO
	Federico MARESCA

December 2022

Summary

One of the most challenging problems nowadays is the development of robotic systems capable of carrying out increasingly problematic tasks with a high level of autonomy. In this sense, this thesis work aims to develop a software architecture for a mobile manipulator (Locobot WX250) that can: (i) recognize an obstacle along its path, and (ii) perform pick and place tasks without being aware of how the map is constructed. The mobile manipulator software architecture is implemented using ROS (Robot Operating System) to communicate within its modules. ROS represents the state of the art in middleware software and has a modular structure that can be updated, enriched, or simplified at any moment without corrupting the whole system. Furthermore, along with ROS, the manipulator's tasks are developed using the MoveIt framework. This choice allows the use of different possible motion planning algorithms, leaving the decision on which is the most suitable depending on the end user's necessities. The obstacle awareness for the manipulator is brought through Octomap, while the algorithms picked for solving the motion planning problem are: Stochastic Trajectory Optimization for Motion Planning (STOMP), Covariant Hamiltonian Optimization for Motion Planning (CHOMP), and Open Motion Planning Library (OMPL). Thanks to the rich framework of ROS and widely available, open-source packages we were able to test various techniques for localization and mapping, path planning and obstacle avoidance. The navigation stack that is at the core of the mobile base software uses odometry and sensor data (both Lidar and RGB-D), is highly modular and can be customized for each use case. For mapping and localization we adopted RTAB-Map, a graph based SLAM algorithm, that uses real time loop-closure on previously seen locations and that makes use of all sensors available on the Locobot. Obstacle detection is achieved through Lidar and RGB-D sensors, both of which are used to construct occupancy grid costmaps that are then used by the global and local path planner to instruct the mobile base's movements. For global planning we explore both A* and Dijkstra's algorithms and for local planning Trajectory Rollout, Dynamic Window Approach(DWA) and Timed Elastic Band (TEB) algorithms. After a thorough comparison we selected A* for its computational speed and Timed Elastic Band for its flexibility and stability. We also explored and added a social-navigation layer for

human avoidance during path planning with a Gaussian based costmap approach. Our architecture makes use of these packages to implement a search of known and unknown spaces following an ARTag request. The Locobot will reach the ARTag and perform a pick task with the manipulator. After a successful pick operation it will place the requested object in a previously chosen location. To validate our algorithm, we tested the robot both in a simulated environment, thanks to Gazebo, and in a real application using the laboratory as an adapted warehouse.

Table of Contents

List of Tables

List of Figures

Acronyms

POE
Product of exponentials

ROS
Robot Operating System

OMPL
Open Motion Planning Library

CHOMP
Covariant Hamiltonian Optimization for Motion Planning

STOMP
Stochastic Trajectory Optimization for Motion Planning

STOMP-M
Memorey-based Stochastic Trajectory Optimization for Motion Planning

CNN
Convolutional Neural Network

APF
Artificial Potential Field

RRT
Rapidly-exploring Random Trees

SLAM
Simultaneous Localization and Mapping

RFID

Radio Frequency Identification

DOF

Degree of Freedom

EKF

Extended Kalman Filter

LIDAR

Light detection and ranging

RGB

Red-Green-Blue

RGB-D

Red-Green-Blue-Depth

GA

Genetic Algorithms

TEB

Timed Elastic Band

HOG

Histogram of Oriented Gradients

EB

Elastic Band

DWA

Dynamic Window Approach

Chapter 1

Introduction and problem statement

1.1 Robotics nowadays

Robotics nowadays is a science that is deeply involved in technological progress in many aspects. More and more robots are employed to carry out tasks that require high reliability and precision. For example, in this case, we can cite an example of a robotic manipulator, shown in Figure 1.1, constructed to teleoperate in scenarios where human intervention could be dangerous or its presence is not strictly necessary.
The main advantage behind employing robots is that they can operate in dangerous environments but can also be deployed in industries where it is required to perform repetitive tasks faster. For this reason, they operate in many big companies' production chains. In general, we can say that every robotic system is composed of a certain number of subsystems:

- **power source**, for powering the other subsystems composing the structure.
- **actuation** for moving the robot parts.
- **sensing** for perceiving the surrounding environment.
- **manipulation** for manipulating items or, more in general, performing tasks that require interaction with other objects.
- **locomotion** for changing the robot's position in the environment.

Each of these subsystems contributes to differentiating the kind of robotic system we are dealing with.

Figure 1.1: Example of manipulator working in a dangerous environment (foundry) for die casting in metal manufacturing [1].

The increasing interest in developing more specific solutions to work with robots to accomplish tasks resulted in the birth of numerous solutions regarding the software structure. In this thesis work, we will analyze some of them and give particular attention to the state of the art while looking at the available commercial solutions. We intend to develop an application that could be furtherly developed in the future; for this reason, we will preferably look at open-source solutions.
In our particular thesis work, we have to develop the software architecture for a mobile robot that is commercially available and comes with a certain number of software solutions already built-in.

1.2 Classification of robotic systems

When talking about robots, it is crucial to understand the type of robotic system we are dealing with. In particular, a robotic system can be classified according to different criteria. We can differentiate based on the environment where they will work if it is constructed previously or the system has to explore, according to their position on the map if they are fixed in a precise location or they can move in the surroundings, and, indeed, if they have to approach or operate along with humans or in a map where only the robots are allowed to work. Moreover, we can have teleoperated robots or autonomous systems that only need human supervision, not intervention.
Specifically talking about the robot employed in this thesis work, we are dealing with a mobile manipulator capable of moving on the map while having a manipulator for grasping the objects. The robot will operate in an environment where we can have human interaction and will not require, in our intention, to be teleoperated.

1.2.1 Description of a mobile manipulator

When talking about a mobile manipulator, we highlight that it is a system capable of moving in a fixed frame, like a map, changing its position. For example, considering the mobile manipulator in Figure 1.2, it has a manipulator on top and then a mobile base so that it can move to reach a suitable position before manipulating the desired object. Thanks to this, we can consider the mobile base as an additional degree of freedom of the manipulator so that each position in the map can be potentially reachable in a favourable pose; this principle will be examined in depth in the following chapters and deployed for the application presented in this thesis work.
In Figure 1.2, we can observe an example of a mobile manipulator developed for a research paper:

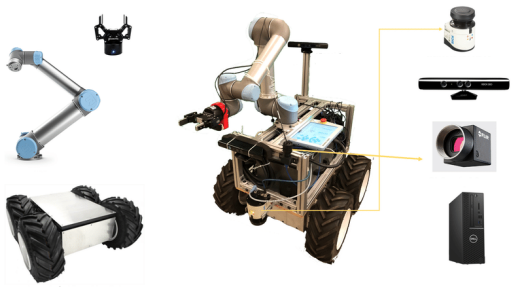


Figure 1.2: Prototype of a mobile robot [2], the mobile base is constituted by differential wheels while on top we have an anthropomorphic manipulator.

In general, we can observe that a mobile manipulator is composed of three main subsystems: the sensors implemented for describing the surrounding environment, the manipulator responsible for grasping the objects and the mobile base accountable for navigating the robot.

1.2.2 Example of mobile manipulators employments

Nowadays, multiple solutions are implemented in real-life applications that use mobile manipulators to carry out tasks regarding picking and placing objects. In this paper [3], for example, they constructed a robot that interfaces with the customers in a supermarket environment. Its objective is to pick the objects chosen by the user which can be identified using RFID (Radio Frequency Identification). The robot comprises a mobile base and a 6 DOF arm for manipulating objects. The main drawback in this implementation is that the items must carry a system for RFID and, as we can see from the user interface, the subset of the products to be fetched has to be constructed before.

Another example of a mobile manipulator used for assistance is presented in [4]. This paper discusses the usage of a teleoperated mobile manipulator, presented in Figure 1.3, for performing kitchen tasks, such as preparing food. The robot is intended to be used by users with disabilities to perform simple tasks in a house environment. The GUI implemented, in this case, allows, through the camera, to control the robot in its operations.
From these examples, we can understand that the main objective of these solutions is to automate repetitive operations avoiding involving human operators.

Figure 1.3: Robot prototype developed for [4] composed by a mobile base and a manipulator.

1.3 Problem statement

As anticipated by the examples presented, the problem that we tried to address regards the construction of the software for a mobile manipulator that can work almost as a plug-and-play solution. Many of the applications presented or cited before work in a specific environment with the supervision of a human operator; our goal is to create a solution that can work stand-alone following only a request for a product from the user. The solutions presented in the previous section are much more specific for the implementations decided by the authors; in this sense, we prefer to construct a solution that can be easily modified and replicated for any robot that has a similar hardware structure. In particular, the robot has first to map the ambient, then, according to an identification system, locate the object of interest and grasp it without, in this phase, being aware of the object or how the shelves where the item is located are made. We think that to make the robot as efficient and flexible as possible, these tasks must be performed online time by time

when necessary; moreover, they must be independent of the map exploited for the navigation tasks.

1.4 Goal of the thesis

Now we can formally present the objective of our thesis work, focusing on the problem to be addressed.

1.4.1 Problem description

The goal of this thesis is to program an already-built mobile robot so that it can accomplish tasks of picking and placing in a dynamic environment. We explored different possibilities for implementing the state of the art in terms of obstacle avoidance, human recognition, object recognition, and motion planning for both the manipulator and the mobile base. Each of these capabilities will be implemented via ROS packages, while the communication with the algorithm will happen through topics and services. The application designed to accomplish our goal will be implemented as a ROS node, independent of the robot itself, and possibly work with all the robots with a similar software architecture.

1.4.2 Environment description

The target environment for the application is a warehouse that has the necessity of picking stored items and placing them in a depot. This environment will have static obstacles, such as shelves, walls, and dynamic obstacles with varying degrees of predictability, varying from other mobile manipulators to humans. Since the arm payload and the maximum height reachable by the arm are not suitable for a real warehouse, we decided to scale the problem by recreating a room with some obstacles, humans, and some shelves where the items will be stocked. The robot will construct the map before being able to satisfy the requests from the user.
Moreover, as items to be manipulated, we consider some medicines that have a suitable dimension for the gripper constructed and are compliant for the maximum payload.

1.4.3 Functionalities required

It is required, by this application, that the robot is able to accept a request for an item from the user, identify the requested item between all those in the scene and then manipulate it safely without damaging the object and harming the operators. The algorithm is also responsible for detecting dynamic obstacles like humans and modifying the computed mobile base trajectory online to avoid collisions.

1.5 Thesis structure

This thesis structure is composed so that we will analyze the current state of the art in Chapter 2. Going into Chapter 3, we will describe, particularly, the solutions present on the market regarding ROS. The hardware selected is described in Chapter 4, while the software structure developed will be presented in Chapters 5, 6, and 7. Finally, we will present the results and tests in Chapter 8. The conclusions and some considerations will be presented in Chapter 9.

Chapter 2

State of the art

2.1 Available Mobile Manipulator Solutions

The solutions developed in the research field are currently focusing on designing service robots, like the ones cited in Chapter 1, that are increasingly autonomous. As an example of an application, we suggest the following paper [5] where a mobile base has been developed to perform shelf analysis. This solution presents some stimulating features such as the implementation of OpenCV to deal with object recognition. Moreover, as middleware, it implements ROS so that the software is composed of open-source packages. Simultaneous Localization and Mapping (SLAM) is the solution selected in this work to map the environment where the robot will operate. The main deficiency behind this solution is that it still needs human intervention to pick an object from the shelf. The OpenCV software, which performs object recognition, is undoubtedly less precise in pose estimation than the ARTag markers package provided by ROS because of the uncertainty derived by the image classification operation.
Another application similar to this thesis work's objective that expresses the efforts made to automate repetitive works is the one presented in [6], where the researchers present a solution for goods picking in a supermarket environment. The paper proposes different solutions such as an innovative trajectory planning strategy in joint space for grasping different types of items present in an emulated standard grocery store. Grasping is done by implementing a suction cup different from the gripper mounted on the Locobot WX250; its working principle is limited to the presence or absence of objects with a plain surface where the grasper can adequately stick. Moreover, the software architecture is not ROS-based, so changing the hardware will force rewriting the entire software losing the possibility to reuse the algorithm for different manipulators. We are looking for a solution that can be adapted to different hardware types without losing its efficiency and reliability.

Moreover, according to our analysis, another weakness is that the motion planning algorithm does not provide any online obstacle avoidance algorithm, so this solution may work under the assumption that the items can be freely picked without any obstacle.

2.2 Middleware software for communication

Currently, on the market, there are several possible choices of middleware structure to communicate within robots' submodules. In particular, we can mention the following ones:

- ROS Robot Operating System
- ZeroMQ library
- RT-Middleware

Between those cited we will analyze ROS in the next chapter because the Locobot already comes with the packages intended for use with this communication middleware.
The idea behind implementing a middleware such as the ones mentioned above relies on the fact that, in this way, there is a faster and easier way to communicate with all the software packages composing the architecture of a robot. Although in the research field many systems implement these structures in real applications they are not so popular; it is preferred, instead, to develop systems intended for the specific application. We can differentiate them based on the level of abstraction of the hardware that the code can provide; some of them are at High-Level and then easier to adapt to different hardware solutions, while others are at a Low-Level stage, and hence they are more specific for the target hardware.

2.2.1 RT-Middleware

RT-Middleware is a software platform used for constructing robotic systems by the composition of multiple software modules that characterize the robot's functional elements. The open-source project correlated to this framework is the OpenRTM-aist which stands for Open-source and open architecture Robot Technology Middleware implemented by National Institute of Advanced Industrial Science and Technology (AIST). The objective of this structure is to implement the functionalities hierarchically provided by the modules to construct a robotic system. The communication between each component is made through ports to exchange data and information, as shown in Figure 2.1.

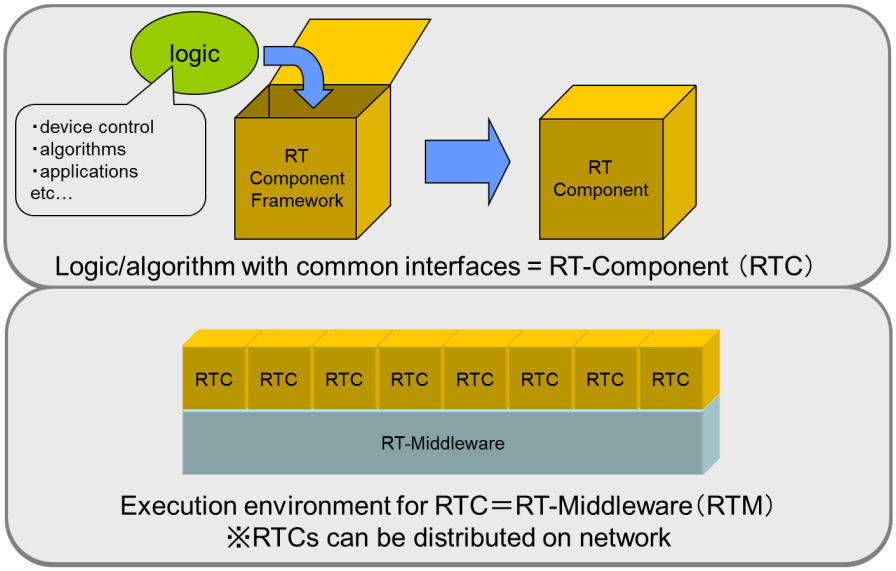


Figure 2.1: An example of RT functionalities are wrapped into an RT component that communicates through ports in the RT-Middleware [7]

2.2.2 ZeroMQ

ZeroMQ is a Low-Level middleware networking library. As stated in [8], it can run without a broker and works through the use of patterns in messaging such as pub/sub, request/reply, client/server, etc.
The main drawback behind this implementation is that it needs to be implemented at a low level inside the software that our robot runs so the code becomes application-oriented and loses its generality. Because of its complex implementation to make the robot communicate within its modules, it is less attractive than the other two solutions proposed.

2.2.3 ROS Robot Operating System

ROS is an open-source software meta-operating system for developing robotic applications. ROS can provide many functionalities such as hardware abstraction, process management, multiprocess communication, and package management. Its structure is made of three crucial components that are: nodes, services, and topics. The whole working of the system is based on the communication between nodes which is possible thanks to the existence of topics. As we can see from Figure 2.2, a node could be a publisher or/and a subscriber of a certain topic. Using this approach, we can draw schemes where we can clearly understand the communication that each node maintains with the others. For debugging purposes this is clearly

helpful, mainly when we deal with a lot of nodes and topics; an example of a network established when starting a robot is reported in Figure 2.3.

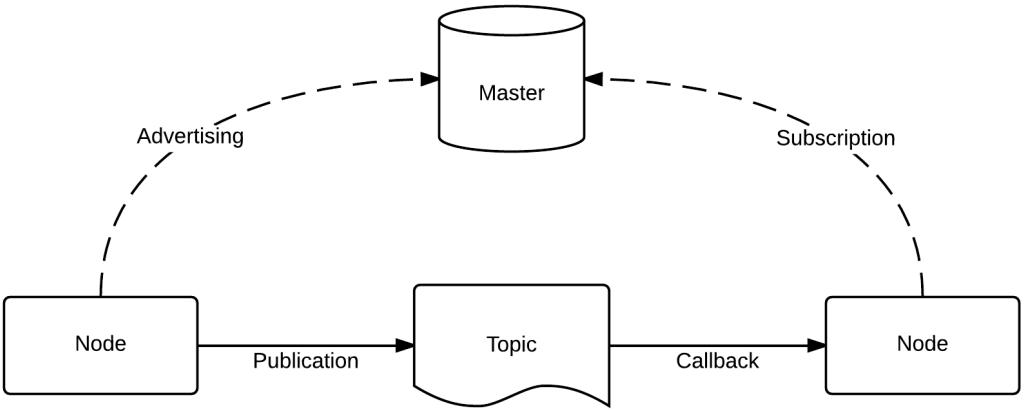


Figure 2.2: An example of ROS communication nodes while publishing and subscribing to a topic.[9]

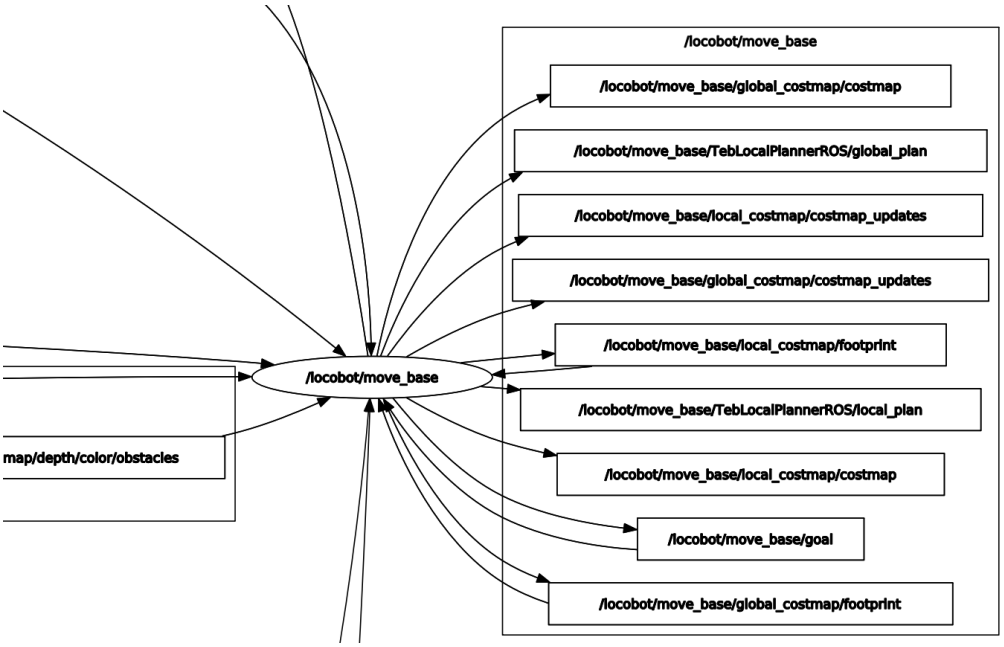


Figure 2.3: An example of a ROS communication structure where is involved a simulation in Gazebo and the visualization in RViz

The main advantage of implementing a system like ROS for managing the robots is that it does not need particular hardware to run; instead, it aims to deal, possibly, with different computers in the same fashion without losing the specificities of the application. We must remember that ROS was born from the need to avoid re-implementing the basic features of each software architecture from the beginning for each particular robot or computer while guaranteeing fast solutions to the most common problems.

2.3 Autonomous Navigation

Autonomous navigation is the capability of a mobile base to reach a goal position and plan its path within an environment without external commands or human intervention. Generally, sensors are placed on the mobile base to allow it to sense obstacles in its path. Also, a map has to be constructed to localize the base's position and to calculate the path from the base's current position and the goal. To tackle this subject, we will first discuss SLAM and its applications, then path planning and finally object and human detection.

2.4 Simultaneous Localization and Mapping

The goal of SLAM is to obtain an accurate estimate of the robot's location in space while traversing and constructing a map of such space. It is one of the most important problems in autonomous navigation since its solution would solve both the need for accurate and continuous localization of a robot and for precise maps of the environment. The SLAM problem is tightly tied to the noisy nature of the available sensors. So the problem is described in terms of a probability computation. In an unknown environment, the movement of the robot along its path from time 0 to time T can be described by a sequence of variables $x1 : T = x1, \ldots, xT$. The sensors take a sequence of odometry measurements $u1 : Tu1, \ldots, uT$ and external measurements of the environment $z1 : T = z1, \ldots, zT$. Solving the full SLAM problem consists of estimating the posterior probability of the robot's trajectory $x1 : T$ and the map $\mathbf{m}$ of the environment given all the measurements plus an initial position x0

$$p(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{z}_{1:T}, \mathbf{u}1 : T, \mathbf{x}0) \tag{2.1}$$

2.4.1 Overview of SLAM algorithms

Of course, the formulation of the general SLAM problem does not lead to one single solution so, over the years, there have been many approaches to solving it. The

following SLAM algorithms have been proposed over the years and are some of the most used and studied in the field.

Extended Kalman Filter SLAM

Extended Kalman Filter (EKF) SLAM was one of the earliest algorithms proposed to solve the SLAM problem [10] . The locations of the robot and features of the environment are described in a single state vector and a covariance matrix represents the relationships between the robot and environment state estimates. It uses two successive phases to accurately predict the robot's poses :

- prediction step
- update step

At first, odometry measurements are used to calculate an a-priori estimate of the robot's pose, then, during the update phase, exteroceptive measurements are used to update the estimate through the use of features in the environment to calculate an a-posteriori estimate. An example of EKF SLAM is Hector SLAM [11], it achieves fast results with low resource consumption together with very low-drift odometry. However, it does not solve the full SLAM problem, since it does not perform loop closures to reduce errors in the state estimate.

Particle Filter SLAM

Particle filters can be visualized as a set of guesses in the state space. At each step, a set of particles are stochastically generated from the previous sets of particles. Then after a sensor message, the update step occurs. In this step, the set of particle filters is assigned a certain probability or weight given this new information. After the update, a resampling step is used to prune the set of particles so as to keep only the most probable ones. The main issue with particle filters is that in the context of SLAM, the dimensionality of the state space is such that generating a large enough set of particles for each feature is costly and so scales exponentially with the dimension of the state space. To counter this issue Particle Filter SLAMs have used optimizations such as the conditional independence hypothesis and Rao-Blackwellization, where the particles are drawn from the robot's path posterior and the associated map's probability distribution is represented as a Gaussian distribution given the selected particles. A well-known and well-researched Particle Filter SLAM is FastSLAM [12].

Graph-Based SLAM

The most intuitive way of approach to SLAM is by envisioning the solution as a graph where the robots poses and landmarks are the nodes connected with

edges that contain the sensor measurements [13]. Due to the noisy nature of the measurements, there may be contradictory perceptions and fuzzy landmark distance so not all edges will be consistent with prior knowledge . This problem can then be seen as a graph optimization problem, where the least contradictory graph is extracted from the originally constructed one. Each successive location node x_{i-1}, x_i is connected by an edge that contains odometry information u_i and other edges to map nodes m_i that contain landmark information sensed by the sensors at time step i. Some SLAM algorithms have used this intuition to build solutions that have more recently seen a resurgence. The SLAM method chosen for this thesis work belongs to this family of algorithms and uses this same intuition.

Sensors For SLAM

SLAM nowadays is tackled using a variety of sensors regardless of the paradigm used (EKF, Particle Filter, Graph, etc). The most common sensors are RGB(Red Green Blue), RGB-D(Red Green Blue Depth), 2D LIDAR (Light Detection and Ranging), 3D LIDAR, IMU(Inertial Measurement Unit) and Wheel Encoders. RGB and RGB-D belong to the family of camera sensors, RGB sensors are widely available and low cost, so they are used by many algorithms, especially since the surge in popularity of Convolutional Neural Networks (CNNs) and Visual machine learning classifiers. RGB-D are cameras with stereo or time of flight capabilities that enhance the frames they capture with depth information. Usually, they are less costly than LIDAR sensors but are affected by lighting conditions and have a reduced horizontal Field of View (FoV). 2D and 3D LIDAR (Light detection and ranging) uses the time of flight to gather obstacle distance information by measuring the time it takes for a beam of light to travel to and reflect from an obstacle. The main advantage of 2D LIDARs over normal cameras is their wide horizontal FoV and longer depth measurement range. Their main drawback is the lack of vertical resolution, meaning that sensing only occurs on the plane orthogonal to the laser direction. While more expensive, LIDAR is more accurate and does not suffer from lighting conditions as much as the RGB sensors, however, it loses accuracy when measuring around reflective and transparent surfaces. In [14], the analysis deals with RGB-D sensor-based vSLAM algorithms (with examples such as SLAM++, ElasticFusion, and Dense Visual SLAM). Our experiments take place in an office so typical lighting issues that arise when doing SLAM outdoors are not an issue, however, there are many reflective and glass walls that can cause problems in both kinds of sensors.

A research work presented in [15], investigates various ROS-based visual SLAM methods and studies their feasibility and applicability in a homogeneous indoor environment (defined as a office-style environment with homogeneously painted walls and containing reflective surfaces such as windows and mirrors). A stark

comparison between 2D LIDAR based Hector Slam and RGB-D based RTAB-Map. Both performed well but the former was much more accurate with a maximal trajectory deviation of 0.18m against the latter's 0.67m. This environment closely mirrors ours, so following their conclusion, RGB-D is a good choice together with LIDAR when there are glass or large monochrome walls.

2.5 Mobile Base Path Planning

SLAM builds up a global map that will be used for localization during the robot's movement. When a new target is received, odometry data are used to calculate the current pose and the global planner will calculate a possible path to reach the target path. During navigation, the local planner will then take into account the local costmap information to try to follow as closely as possible the plan laid out by the global planner, while avoiding obstacles that are not present on the global costmap, such as moving objects or objects that have been moved after the construction of the map. The main differences in the global and local planners are summarized in Table 2.1.

Global Path Planning	Local Path Planning
Map based	Sensor based
Apriori knowledge of obstacles	Only local and current obstacles
Calculate a path from start to goal pose	Calculate a path to follow the global path
Slower	Faster

Table 2.1: Comparison of global and local path planners

2.5.1 Global Path Planning Algorithms

Artificial Potential Fields

First introduced in [16] this path-planning algorithm uses the concept of repulsive and attractive forces to move the robot in the surrounding space. Obstacles generate repulsive forces that push away the mobile base and the goal attracts it. At any given point the attractive and repulsive potentials can be used to calculate the total potential U and the next movement towards the goal by identifying the most promising negative gradient.

Rapidly exploring Random Trees

A rapidly exploring random tree (RRT) [17] is an algorithm that explores a given space by building a random tree that fills the space. The tree expands incrementally at each interaction by sampling the available free space and adding these samples

to the tree. This kind of technique though simple is efficient at exploring the free configuration space. The procedure is, in fact, biased towards the space that has yet to be visited by the tree.

Genetic Algorithm

Genetic Algorithms (GA) were first presented in [18]. They take inspiration from Darwinian evolution. First, a generation of solutions is calculated and their quality is evaluated, then this generation is used to breed the next generation. Each generation is also mutated to ensure diversity in the population pool.

Dijkstra Algorithm

Dijkstra's algorithm is a highly influential shortest path finding algorithm first proposed by Edsger W. Dijkstra in 1959 [19]. It visits unvisited graph edges from lowest to largest cost in a BFS (Breadth First Search) fashion. The solution to a path-finding query using Dijkstra will always result in the shortest possible path since it visits all possible edges in every direction. By translating a 2D occupancy grid to a graph using the cells of the grid as nodes and the cost of those cells in the edges connecting them we can use Dijkstra to compute the shortest path between cells.

A*

A* was first proposed in 1968 [20] by Stanford researchers. While Dijkstra finds the shortest path from a starting location to all possible locations, A* only takes into account the shortest paths from a starting location to a target location. This change allows the algorithm to use heuristics to improve its speed. A simple heuristic, in this case, could be: given the a-priori knowledge of the goal location, if two cells have the same cost then choose the one that is in a straight line the closest to the goal. This can result, if the heuristic is admissible (it cannot overestimate the cost of reaching the goal), in the optimal solution. The example of the straight line heuristic is not admissible since it does not take into consideration the fact that even if a cell is closer in a straight line there could be an obstacle in that cell's direction, greatly increasing the cost down the line.

D*

D* is a heuristic algorithm that was originally presented in [21] and has been further modified and developed since then. The main difference between D* and A* is that D* searches for the path starting from the goal and backtracking to the start.

2.5.2 Local Path Planning Algorithms

Dynamic Window Approach (DWA)

The Dynamic Window Approach [22] is a real-time planner that is used in local path planners to calculate the best admissible velocity for a mobile base to reach a goal. First, it generated a set of valid velocities in the state space and then selects the optimal velocity from the set. The trajectory is chosen by using a look-ahead function of the next iteration or dynamic window. This makes sure to allow for safe movement taking into account the dynamics of the robot such as stopping in case of obstacles.

Trajectory Rollout

Trajectory Rollout [23] is a predecessor of DWA. It does not take into account the dynamic constraints of the robot such as maximum acceleration or velocities.

Elastic Band (EB) and Timed Elastic Band (TEB)

The online path planning method that was proposed in [24] works by generating a complete path, called elastic band, at the start of the planning sequence. During the movement, the encountered obstacles that had not initially been taken into account during path planning generate artificial forces that deform the elastic band in real-time. While deforming, the elastic band still attempts to follow the global path. The forces used are repulsive external ones generated from the obstacles and an internal contracting force that is a property of the elastic band. The external forces are generated from the obstacles as stated above. The internal one keeps the path tight and as short as possible while attempting to come back to the original state, the global path. This approach has been subsequently been expanded into the timed elastic band technique [25]. It modifies the previous method's approach by adding constraints related to the dynamic nature of the robot and obstacles to the real-time calculation of the path. The dynamic constraints inform the external and internal forces of the elastic band by taking into account the limited acceleration and speed of the mobile base and the movement of obstacles. While the previous method could only optimize for the shortest possible path, the objective function of this new approach can be changed to find the fastest path or a mix of both by weighting their contributions in the final path.

2.5.3 Obstacle Recognition, Tracking and Path Prediction

This step is an important part of the thesis project since the robot must be able to work in an environment that can contain humans and other moving parts. Obstacles can vary from simple items such as a moving cart to complicated ones like humans.

Recognition is the simplest step as real-time and effective visual methods are already available (for both LIDAR and RGB-D data).
Obstacle tracking and path prediction are relative to the robot's ability to analyze obstacle information and predict the obstacle movement direction or goal; this is much more complicated than mere recognition since the path prediction may change together with the type of object. Just taking into consideration humans as obstacles, the velocity direction may change drastically from one moment to another with no connection to current velocity and pose. The robot must be able to react, dynamically changing its path plan and adapting it to the new conditions. Obstacle path avoidance is the last step where the previously calculated path is changed to reflect the current and future (predicted) real-world situations. Various techniques exist to tackle this problem. In a highly controlled and predictable setting, like a warehouse, the use of QR codes or other easy-to-read markers on both human and non-human obstacles would be a safe and optimal solution since it removes the guesswork and CPU-Load of Machine Learning and other probability-based techniques. This reduces the plug-and-play capabilities of the software by requiring prior environment preparation by placing markers on objects and obstacles, this also may make some obstacles invisible to the robot in case the code is misplaced or not placed at all. Objects should be both recognized and, if they are moving, tracked to update the movement prediction that is being made online.

Methods for Recognition

Object recognition has been a rich field of study ever since the birth of Viola-Jones Detectors [26] that use haar wavelets to look for facial features and Histogram of oriented gradients (HOG) [27] that identify objects by looking at the distribution of edge directions. Neural Networks (NN) have emerged as a natural fit for this task and are widely used in industrial applications [28] together with traditional image processing [29]. NNs have had a recent renaissance with the speed up of computing and the use of GPU cores for image processing, they usually require a GPU for real-time detection but are stable and can identify humans where other more traditional methods (such as HOG) fail. In particular, CNNs are at a mature stage of study and many models are being used for human recognition. Recently for real-time human detection, speed up in algorithms together with hardware improvements have made it possible to use these algorithms in an online fashion[30]. Other least computationally intensive classifiers can be used:

- **Random Forest** (RF)
- **Histogram of oriented gradients** (HOG)
- **Haar Cascade**

The most popular is the Kalman Filter, as shown in [31] where a 2D LIDAR and Kalman Filter is used to track legs and predict the position of the leg clusters. HOG detector

2.5.4 Social Navigation

Social navigation is the study of the path planning and behaviour of a robot in a social setting; it can include human behaviour prediction, such as reading social cues. This aspect of navigation also includes the study of proxemics, a term coined by anthropologist Edward T. Hall in 1963 [32]. It is the study of human behaviour in space, social cues and interactions. Regarding this thesis, the most relevant aspect of proxemics is the robot's ability to respect a human's personal space and infer their movement by through detection and tracking. Hall divided human interpersonal distances from closest to farthest :

- Intimate space
- Personal space
- Social space
- Public space

To respect these spaces path planning must take into account the difference between a box and a human, one may be passed more closely while the other must be kept farther away. A work presented [33] divides path planning for social navigation into reactive and predictive planning.

- **Reactive Planning** Reactive planning takes into account, at all time steps, every possible movement of the robot and removes any movement that may cause a collision. This planning approach is the most conservative and safe but it cannot take into account human behaviour or read social cues. It can also incur into the Freezing Robot Problem [34], where the robot stops moving since any course of action could lead to a collision.

- **Predictive Planning** Predictive planning models in some way human behaviour. By being able to understand human movement it can plan a path optimally when near people and move in a "human-friendly" way. This of course is a probabilistic approach and so can lead to a collision in some cases when the movement prediction is wrong and the robot can't stop in time to avoid it.

Obstacle Path Prediction

Path prediction can be subdivided into 3 major methods: Physics-Based, Pattern-Based and Planning-Based [35].

- **Physics-based** Physics-based prediction methods use some dynamic equation to predict human motion without taking into account human decision-making and by simply modelling people as static linear velocity objects.
- **Pattern-based**

  Using machine learning one can use data sets of human movement to infer patterns of behaviour and, given a certain position, velocity and environment predict the most probable path for a tracked human.
- **Planning-based**

  Planning-based methods output hypothetical movement plans for tracked humans and other moving objects using cost functions and other heuristics either reward functions or statistical learning.

A linearization of previous tracking information to infer future paths is simple but rudimentary and can lead to errors, especially if tracking humans; learning human movement patterns with neural networks instead is effective but complex and hardware intensive. Also datasets and annotation can be difficult for this kind of pattern learning, since the amount of data to be learnt is much larger than, for example, a simple color image for a classifier.

2.6 Manipulator motion planning algorithms

Motion planning for manipulators is a problem that nowadays seems challenging to be solved without giving a remarkable computational effort. For this thesis work, we analyzed different approaches, two of them are based on the state of the art in terms of improving existing algorithms while the other is a framework that implements many algorithms that are currently valid alternatives in terms of reliability.

2.6.1 Memory-based Stochastic Trajectory Optimization STOMP-M

The following algorithm is presented in [36] and is an improvement of an existing algorithm name STOMP [37]. The main idea behind this modification is that we can use **precedent-computed trajectories** to speed up the computation of the new one. In particular, as explained previously, it stores information about the

previously computed trajectory along with data about the presence of obstacles. When a new goal position is provided, it matches the request with the previous data to find if previously a similar path has been computed. Although we do this, it is not guaranteed that this trajectory is better than the one obtained with interpolation so, to ensure this, the result is then compared with the linear interpolation, and the best between them is then selected.
However, the algorithm wholly avoids the matching phase if the current goal is far from the ones present or the obstacles in the planning scene change. The algorithm is the same as for E2STOMP, which is an improved version of STOMP. However, it also includes a search in memory for previously computed trajectories at the beginning and a comparison with the linear interpolation at the end.

2.6.2 Bidirectional Potential Guided RRT*

The approach presented in [38] is an upgrade of an existing algorithm already used in many applications where the manipulator has to perform pick and place tasks. The approach attempts to improve the performance of the standard **RRT*** algorithm by supporting the computation with another algorithm named **Artificial Potential Field (APF)**. This algorithm is particularly useful when working in a scene with many obstacles; it is based on describing the target pose as an attractive force while the obstacles are represented as repulsive forces; in this way, an artificial potential field is described. The target point is considered reached when the repulsive forces are balanced with the attractive ones.
The APF algorithm, as cited in [38], presents two possible problems that can cause falling in a local minimum where the algorithm is stuck or unable to reach the destination:

- The target point is in a zone with many obstacles and the forces are balanced before approaching the target point
- Obstacles are positioned so that there is a local minimum zone where the algorithm is trapped and can not continue path planning

The trajectory computation is still carried out almost entirely by APF, while RRT* is invoked only when there is the risk of falling in one of the two cases above. In this manner, searching for a possible trajectory is faster and more robust guaranteeing a faster convergence with respect to standard RRT

2.7 Item detection

Object detection is one of the hardest tasks to accomplish with a low level of uncertainty and high reliability. During this thesis work, we analyze the possibility of implementing an "**online**" recognition based on deep-learning techniques and an "**offline**" one which relies on the presence of markers attached to the item of interest.

2.7.1 Item recognition via Convolutional Neural Network

Going in deep into the analysis of the possible deep-learning methods, an interesting approach is presented in [39] where the objective is to classify a series of retail items that, as in our case, are stored on shelves.
It explains that the most common problem for identifying an object is that the images are taken each time with different ambient conditions so that is difficult to forecast the actual result of the identification. To overcome this problem the images are classified according to their blurriness level. Then, the author, explains that a **VGG16** neural network is implemented, which is a CNN with 16 different layers. Each time a layer analyzes a smaller subset of the total image until it reaches the identification of the object. This gave fundamental results in terms of the reliability of this system. In our specific case, the main disadvantage of implementing this solution is that the robot cannot perform such a high computational effort and the recognition must be done faster. For this reason, we present the alternative method using markers to recognize the object.

2.7.2 Item recognition via a set of markers

In this case, image recognition is performed by characterizing each object that has to be grasped with markers. Actually, on the market, there are different types of markers such as **AprilTag**, **ARTag**, and **QR code**. In our analysis, we decided to consider the introduction of **ARTags**. Examples of markers are shown in Figure 2.4. An essential aspect of our thesis work's objective is identifying an object that has to be grasped. For what concerns this aspect we decided to analyze the role that the ARTags [40] have in this field since they are widely employed to accomplish this task and they are not so heavy to employ from the computational point of view. An ARTag is a fiducial marker system to support augmented reality and it works through a series of "images" like the one presented in Figure 2.5.

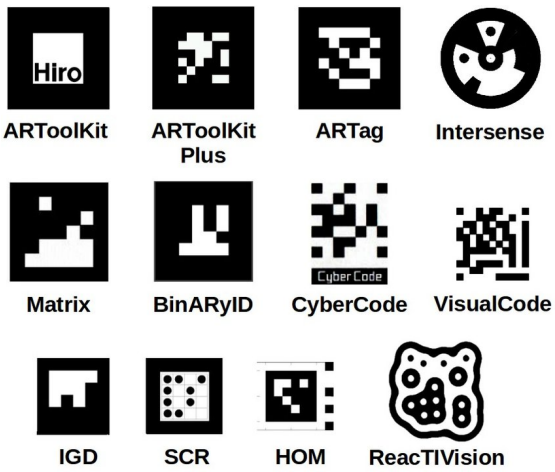


Figure 2.4: Example of the current available fiducial marker systems [41]

Figure 2.5: An example of an ARTag implemented in our system

It has the advantage of providing some information such as an id. In particular, they can be generated so that each is associated with a unique id that can be useful to identify an item in our specific case. Moreover, they work with the depth camera, so their position and orientation can be easily tracked over time. As well presented in [40] they are a bi-tonal system comprising a total of 2002 possible unique planar markers. The first 1001 consists of markers with a black square border and a grid of 6x6 white cells; the other 1001 are the opposite in terms of colours. The algorithm works in such a way that it locates the entire ARTag in the scene then analyzes the interior and samples the black and white square with a series of '1' and '0'. The risk of a false positive in identifying a marker or confusing the markers between each other is quantified to be approximately $< 0.0039\%$ as stated in [40].
The main drawback behind this approach is that it relies on the fact that each item is associated uniquely with an item. Since the subset of possible markers is finite, the objects can not be more than a certain number so the types of items to

be stored must be known a-priori. The main advantage is that, given the structure of the robot, it is capable of being adapted to different scenarios without being aware of the object to be grasped but recognizing it by its marker.

2.8 Grasping techniques

The challenge related to the grasping problem is identifying the optimal grasping position in an object with possibly irregular shape and/or density. For this reason, many papers suggest using a convolutional neural network to identify the object and then select the grasping target. An example of this is presented in [42] where the authors suggest employing a CNN (convolution neural network) to detect the item (as presented in the "object detection" section) and then estimate its pose in the space. This is made to determine the best point to approach the object to be grasped.
Considering our specific application, the hardware composition of the manipulator forced us not to consider this as a problem that can be handled since the gripper lacks any kind of force feedback sensor that would have helped in developing some grasping algorithm; for this reason, this problem, cannot be addressed in this thesis work.

Chapter 3

ROS state of the art

Because of the advantages previously presented, Trossen Robotics, the producer of our robotic platform, decided to choose ROS (Robot Operating System) as middleware software. ROS is the state of the art regarding flexibility, reliability, and adjustability for the open-source solutions available on the market. Considering specifically our tasks, as a working environment for grasping and manipulating, we choose MoveIt, which is widely supported and has a lot of potential for development. In particular, ROS is intended to have a modular structure where we can add or remove functionalities without corrupting the whole system. Each function, such as obstacle avoidance, could be included by compiling its ROS package and tuning the specific parameters in a configuration file. The mobile base instead uses as the core of its navigation stack the one reported in [43], the standard package for ROS navigation. Other ROS packages will be leveraged to achieve the correct obstacle avoidance and manipulation goal.

3.1 MoveIt planning framework

Along with ROS, we implemented a planning framework with the objective of taking care of controlling the manipulation part of the tasks. In particular, MoveIt [44] provides two interfaces to deal with the two main aspects of manipulation: the **move_group interface** and the **planning_scene interface**. The first one, as shown in Figure 3.1, performs direct and inverse kinematic, collision checking between the obstacles and the manipulator, planning and executing the trajectories while reading and interpreting the measurements of the robot's sensors.

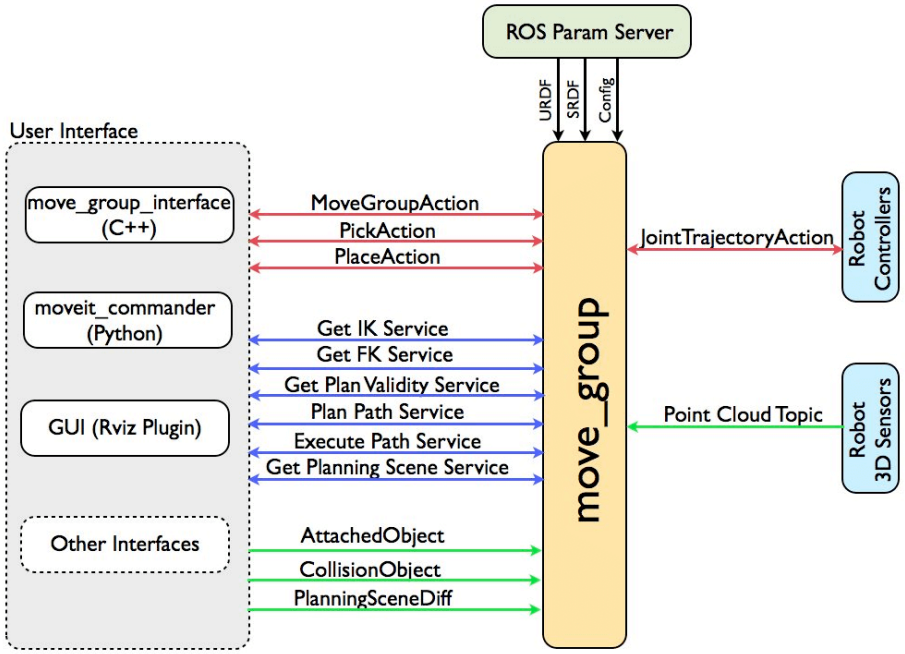


Figure 3.1: Structure of the move_group interface which governs the planning request for the manipulator as described by MoveIt website [45]

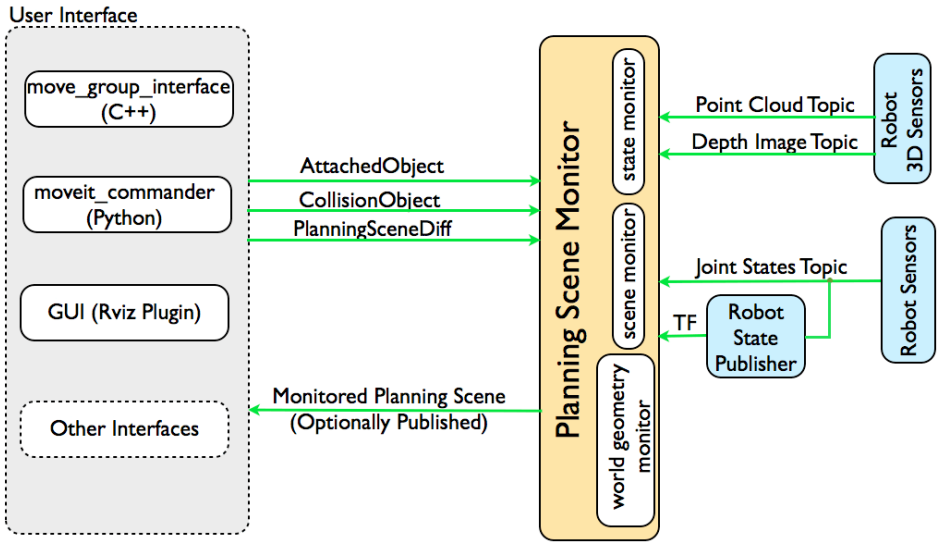


Figure 3.2: Structure of the planning scene monitor, which controls the update of the planning scene and the obstacles present in the environment as described by MoveIt website [45]

The main point of using move_group to manage the manipulation is that it can invoke the **motion planning algorithm** and, based on the planning scene and on the instructions provided by the user, it can compute, plan and execute a path following the limitations given by the constraints. These provisions could regard the following aspects of the manipulator's motion:

- Obstacles along the path
- Waypoints to go through when moving the arm
- Tolerance constraints for both the path and the goal position
- Limitations about joint angles, velocities, accelerations, and effort

The planning scene, instead, is supervised by the planning scene monitor, as we can see in Figure 3.2. It has the task of managing the objects in the scene and the robot itself. The purpose of using such a tool is to allow to add the objects to be grasped in the planning space so that the robot is aware of their presence and not they are not considered obstacles. The objects can be added to the scene by coding or reading sensor measurements, depending on how the software is programmed. Considering the last case, an example of a tool that takes a sensor reading and provides an occupancy grid in the planning scene is Octomap. Once the whole system is implemented, the goal is to update the scene via the planning scene monitor with the new information provided by the sensors for reaching correctly the goal position that the user gave via the move_group interface. Along with MoveIt, it is possible to invoke RViz so that we can plan arbitrary poses for the manipulator or for the mobile base and have a look at what the sensors are measuring.

3.2 What is a motion planning algorithm

A motion planning algorithm is, in general, a computational solution to a problem that consists in finding a way to connect two points through a series of valid configurations that the robot can assume. In this sense, the algorithm must consider different aspects and constraints so that the input request, an ideal goal pose, is transformed, starting from a consistent start pose, in a series of valid configurations assumed by the manipulator. In Figure 3.3, we report an example of what a motion planner aims to do to find a feasible trajectory.

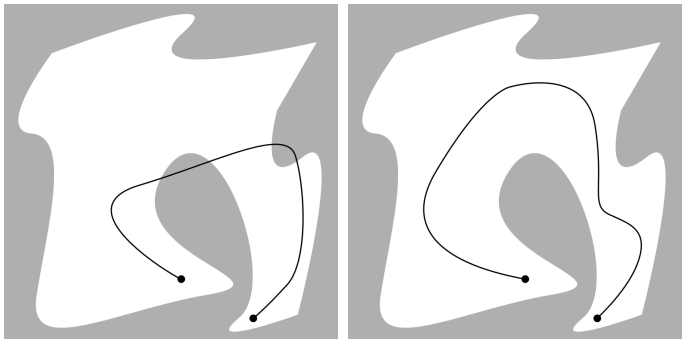


Figure 3.3: In this figure, we can represent an invalid path (left) and a valid path (right). The grey space represents the obstacles on the map, while the white area is the free space. [46]

It is demonstrated that the problem of finding a suitable path is of type *PSPACE-complete*, so it is difficult to find a mathematical formulation that can be considered valid for every situation. In this sense, most algorithms can find a solution, if it exists but cannot prove the non-existence of a valid path for accomplishing the task. Previously, we analyzed two innovative motion planning algorithms that are currently the state of the art. Since we presented ROS and MoveIt as possible solutions, we will discuss the motion planning algorithms that MoveIt exploits to solve the planning problem.

3.3 Motion planning algorithms for mobile manipulators

Motion planning algorithms are fundamental when considering a robot that works in a dynamic environment where avoiding obstacles is essential to prevent harming humans and damaging objects. In this sense, MoveIt provides a series of manipulation algorithms. According to the MoveIt website [44],[45] the motion planners available are the following:

- **Covariant Hamiltonian Optimization for Motion Planning** (CHOMP)
- **Stochastic Trajectory Optimization for Motion Planning** (STOMP)
- **Open Motion Planning Library** (OMPL)
- **Pilz Industrial Motion Planner**
- **Search-Based Planning Library** (SBPL)

In our work thesis, we decide to analyze the first three solutions because two are already tested and fully operative in MoveIt (OMPL and CHOMP) while the last one, STOMP, presents an approach that can guarantee surprisingly good performances.

3.3.1 CHOMP description

CHOMP planner [47] implements gradient-based trajectory optimization procedures. This algorithm is structured such that it can generate and optimize the trajectory reacting to the surrounding environment by quickly changing its path while maintaining control of the joint velocities and accelerations. In particular, the objective of the algorithm is to compute, as said previously, a trajectory between two predefined points that are q_{init} and q_{goal}; this computation is made by discretizing the trajectory in a set of n waypoints $q_1, ..., q_n$ and evaluating, at each waypoint, the dynamical quantities such as velocity and acceleration. Furthermore, the optimization is made by evaluating a cost function, as said in [48], that is formed in the following way:

$$U(\xi) = f_{prior}(\xi) + f_{obs}(\xi) \tag{3.1}$$

where f_{prior} measures the cost of dynamical quantities, and it is assumed to be independent of the environment; f_{obs} instead quantifies the cost of being near to an obstacle with the trajectory. The technique proposed in [48] aims to improve the trajectory at each iteration of the algorithm by minimizing a local approximation of (3.1) given in the following form:

$$U(\xi) \approx U(\xi_k) + g_k^T(\xi - \xi_k) \tag{3.2}$$

where $g_k^T = \nabla(\xi_k)$. By exploiting this approximation, the update can be written in the following formal form:

$$\xi_{k+1} = arg\,min\{U(\xi_k) + g_k^T(\xi - \xi_k) + \frac{\lambda}{2}||\xi - \xi_k||_M^2\} \tag{3.3}$$

CHOMP is said to be a method based on covariant because, as we can see from (3.3), the update depends only on the trajectory itself and not on the representation used (waypoint based in our case). Thanks to this formulation of the problem as a covariant optimization, we can optimize directly in the space of trajectories.

3.3.2 STOMP description

STOMP planner [49] is an optimization-based motion planner based on the PI^2 (Policy Improvement with Path Integrals) algorithm. Its characteristic is that it can produce smooth paths for the manipulator to avoid colliding with obstacles. Moreover, it can optimize arbitrary cost functions that take into account, for example, the motor efforts. The approach relies on the fact that, at each iteration, noisy trajectories are generated; after that, the cost of each trajectory is evaluated through a suitable cost function and then the candidate path is updated according to the results obtained.
In the case of STOMP, the goal of the algorithm is, as for the previously cited procedures, to find a smooth trajectory that minimizes the cost function associated with collisions and other constraints. As for CHOMP, also, in this case, we define an initial point, a goal point, and a series of waypoints equally distanced in time T. For the solution to the problem, we have to assume that the initial point and the goal point are fixed during optimization. The objective is to solve the following optimization problem [37]:

$$\min_{\tilde{\theta}} \mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\theta}_i) + \frac{1}{2}\tilde{\theta^T}\mathrm{R}\tilde{\theta}\right] \tag{3.4}$$

where:

$$\tilde{\theta} = \mathcal{N}(\theta, \Sigma) \tag{3.5}$$

$\tilde{\theta}$ expresses a noisy parameter vector with mean value θ and variance Σ, R expresses the control cost and $q(\tilde{\theta}_i)$ is an arbitrary state-dependent cost function. The main objective behind this approach is that the algorithm allows us to use an arbitrary cost function $q(\tilde{\theta}_i)$ that could be not derivable, non-differentiable, or non-smooth. The algorithm can be iteratively solved, and it needs some precomputed values such as the matrix **A**, the matrix **R**, and the matrix **M**. First, we can define A such that when multiplied by the position vector produces accelerations, as can be seen in Figure 3.4. If we want to have a glance at how the algorithm works and how the other two matrices are related to **A**, we can have a look at Figure 3.5:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix}$$

$$\ddot{\boldsymbol{\theta}} = \mathbf{A}\boldsymbol{\theta}$$

$$\ddot{\boldsymbol{\theta}}^{\mathrm{T}}\ddot{\boldsymbol{\theta}} = \boldsymbol{\theta}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\mathbf{A})\boldsymbol{\theta}$$

Figure 3.4: How matrix **A** is defined

THE STOMP ALGORITHM

- **Given:**
  - Start and goal positions x_0 and x_N
  - An initial 1-D discretized trajectory vector $\boldsymbol{\theta}$
  - An state-dependent cost function $q(\boldsymbol{\theta}_i)$
- **Precompute:**
  - $\mathbf{A}$ = finite difference matrix
  - $\mathbf{R}^{-1} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}$
  - $\mathbf{M} = \mathbf{R}^{-1}$, with each column scaled such that the maximum element is $1/N$
- **Repeat until convergence of trajectory cost Q($\boldsymbol{\theta}$):**
  1) Create K noisy trajectories, $\tilde{\boldsymbol{\theta}}_1 \ldots \tilde{\boldsymbol{\theta}}_K$ with parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k = \mathcal{N}(0, \mathbf{R}^{-1})$
  2) For $k = 1 \ldots K$, compute:
     a) $S(\tilde{\boldsymbol{\theta}}_{k,i}) = q(\tilde{\boldsymbol{\theta}}_{k,i})$
     b) $P\left(\tilde{\boldsymbol{\theta}}_{k,i}\right) = \frac{e^{-\frac{1}{\lambda}S(\tilde{\boldsymbol{\theta}}_{k,i})}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\tilde{\boldsymbol{\theta}}_{l,i})}]}$
  3) For $i = 1 \ldots (N-1)$, compute: $[\delta\tilde{\boldsymbol{\theta}}]_i = \sum_{k=1}^{K} P(\tilde{\boldsymbol{\theta}}_{k,i})[\boldsymbol{\epsilon}_k]_i$
  4) Compute $\delta\boldsymbol{\theta} = \mathbf{M}\delta\tilde{\boldsymbol{\theta}}$
  5) Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$
  6) Compute trajectory cost $Q(\boldsymbol{\theta}) = \sum_{i=1}^{N} q(\boldsymbol{\theta}_i) + \frac{1}{2}\boldsymbol{\theta}^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}$

Figure 3.5: The STOMP pseudo-code [37] solved iteratively

3.3.3 OMPL description

OMPL library [50] is a collection of sample-based motion planning algorithms. It is based on the assumption that to find a feasible solution, we do not have to look for all the possible configurations but we can define a set of uniform random samples that can be linked until we find a collision-free path that can connect our start position effectively with our goal position. A more accurate and convenient solution can be found if we enlarge the set of random configurations analyzed. The necessity to adopt this approach is to give a solution in a reasonable amount of time. In the library, we can choose between many algorithms such as PRM (Probabilistic Roadmap Method), RRT (Rapidly-exploring Random Trees), SPARS (SPArse Roadmap Spanner algorithm), EST (Expansive Space Trees), KPIECE (Kinematic Planning by Interior-Exterior Cell Exploration), STRIDE (Search Tree with Resolution Independent Density Estimation), PDST (Path-Directed Subdivision Trees), FMT (Fast Marching Tree algorithm), BFMT (Bidirectional Fast Marching Tree algorithm) and QRRT (Quotient-Space RRT), and others. An important distinction can be made to distinguish the most suitable for our application:

- **Geometric planner**: this class of planners only contemplates geometric or kinematic constraints. It is based on the assumption that if a trajectory is feasible then it can be transformed into a dynamically feasible path.
- **Control-based planners**: this category, instead, also takes care of differential constraints. They implement state propagation instead of interpolation to generate a feasible path.

: Here, in Figure 3.6, we can have a look at the structure and the calls that can be made to the algorithm:

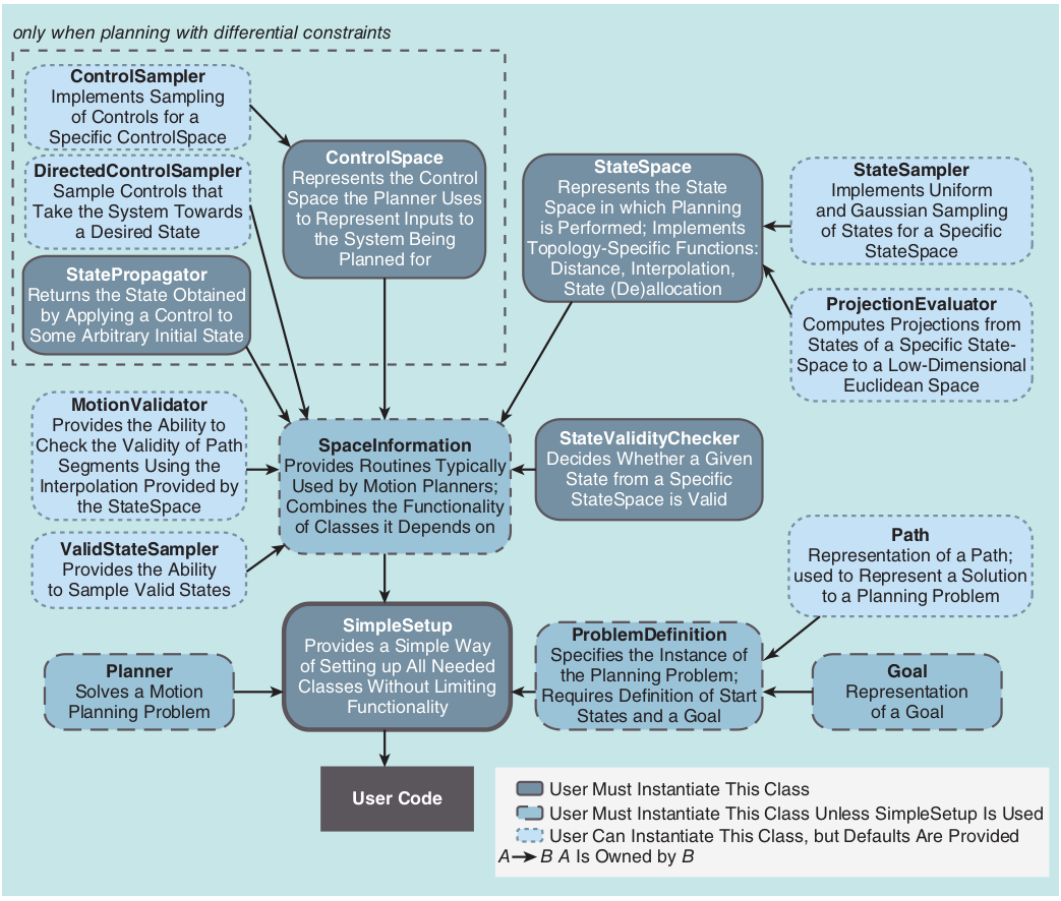


Figure 3.6: An overview of the OMPL structure [51] with the calls and the functions characterizing the algorithms.

Regardless of the algorithm chosen for motion planning, OMPL provides a common structure that interfaces with the motion planning algorithm. The core of the library is composed of the following structures:

- Definition of the **state space**: it is the representation of the search spaces, in the more generic possible way. This approach is intended to be applied to many different applications so that the motion planning algorithms can be applied at each state space defined.
- **State validation and integration**: their duty is to check if a state is valid or not. In particular, it checks if a path does not collide with any obstacles or, in general, respects the predefined constraints. After that, if two states are considered valid, it analyzes if their interpolation is still valid.
- **Samplers**: they perform the sampling functionality. OMPL includes four

types of samplers: state space samplers, valid state samplers, control samplers, and direct control samplers. As stated in [51], selecting a suitable sampler can significantly influence the computational time required by the planning algorithm.

- **Goal representation**: this analyzes the request for a given state and decides if it is a valid goal state or not. It offers an indication of how to reach the goal region. This is important because it addresses the search for the goal.
- **Planning algorithms**: those are the algorithms cited before that can be purely geometric or control-aware.

3.4 Obstacle detection and avoidance

3.4.1 Octomap as obstacle detection

The obstacle detection task is trivial to be solved since we need a not-so-heavy solution, from the computational point of view, that can be easily included in MoveIt. Moreover, the solution selected must accurately describe the obstacles present in the scene using the depth camera and the lidar sensor. In our case, we decide to implement Octomap [52]. It is fully compatible with MoveIt and has some important characteristics for our application: it is compact and updatable. The entire system relies on constructing a 3D occupancy grid map that has the following characteristics:

- The map is capable of modelling the environment without prior information. In case we are dealing with a robot with a static camera frame, as in our case, the occupancy grid will be constructed according to the field of view of the camera. Each time the framing changes the map will be reconstructed.
- It is possible to update the map every time is needed and, eventually, to delete it. This is particularly important if we think this tool can be implemented when considering multiple robots moving in the same environment.
- As said previously, when considering this application, the size of the map is not important to be known in advance because in case it is needed, it can be dynamically expanded.
- When storing the information on the map, it is not required to have much space in the disk because it will be stored efficiently.

An example of how Octomap can efficiently model the obstacles is presented in Figure 3.7. We can emphasize that the smaller the cube for describing the

object more precise the representation will be. As described in the tool's official documentation, the cube's size, such as other parameters, can be tuned to satisfy our needs.

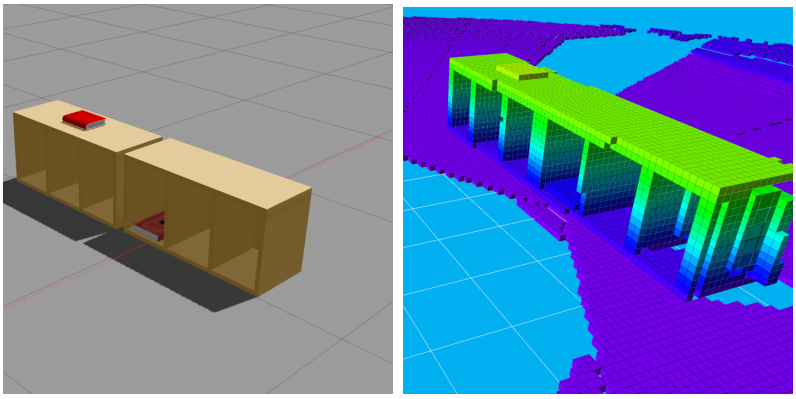


Figure 3.7: An example of the obstacles sampling made by Octomap (right) of the environment (left)

3.5 Mobile Base Navigation

3.5.1 Navigation Stack

The ROS architecture uses the **Navigation** [43] package for the robot movement. The inputs are external sensors and odometry from the robot and the outputs are goal poses translated to velocity commands to move the robot. Odometry is information that comes from sensors that can calculate a change in position relative to a known position. Other typical sensors are LIDARs and RGB-D sensors. To use the navigation stack the robot must, of course, have ROS installed and have nodes that publish sensor data in the correct message format for ROS to read. There must also be a description of the robot available to be able to calculate the transforms for the joints of the robot.

3.5.2 Transform tree

The transform tree is a description of the robot at a geometric level. The rotation and translations between every frame that makes up the robot. Every frame has a transform relationship with at least one other frame, this way a tree is built, from which the entire robot can be described. For example, starting from the laser frame one can find the pose of the wheel frame. The package in charge of handling the transform tree is called **tf** [53], it maintains the correspondence between the coordinate frames contained in the transform tree.

3.5.3 Costmap 2D Package

The Costmap 2D package [54] uses sensor data and the **tf** package to build a map of the world surrounding the robot. The map is based on a 2D or 3D occupancy grid: a grid of cells that contain occupancy information in the form of integers. A value less or equal to one represents free space while a value of 254 is an obstacle that represents a certain collision. As the title of the package shows, occupancy is a cost that needs to be taken into account while planning the movement of the robot around the space.

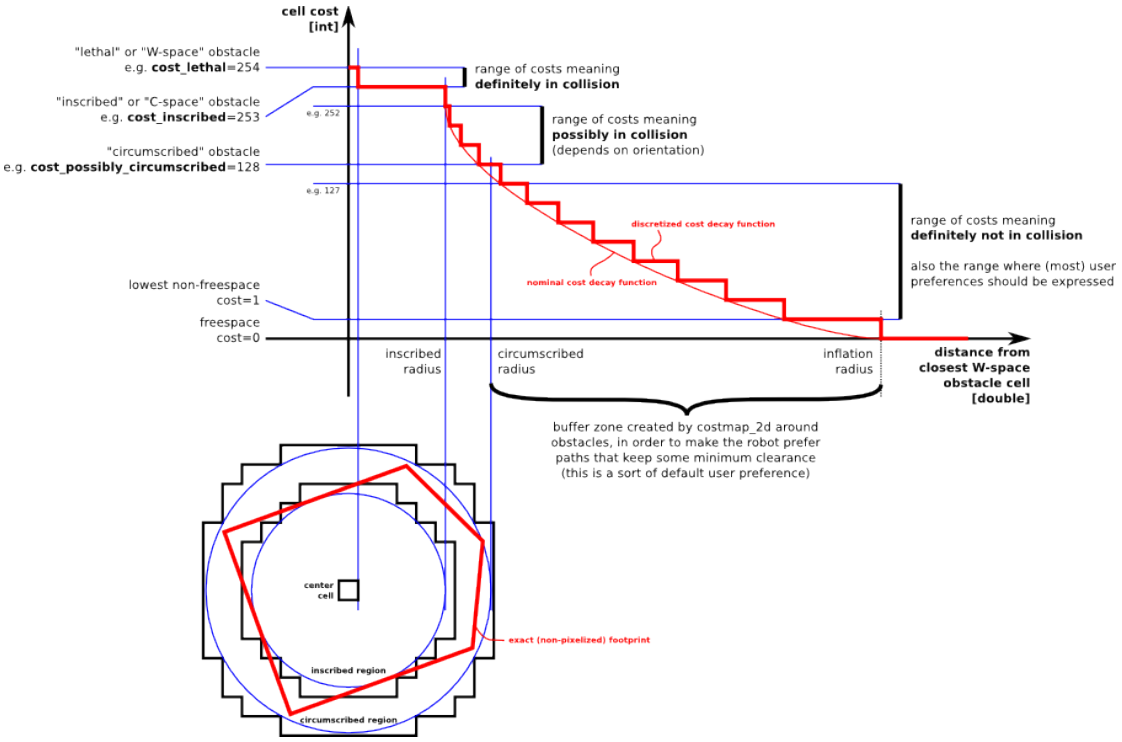


Figure 3.8: Overview of the cell value related to occupancy cost [54]

When an obstacle is found by reading sensor information it inflates the cost around it by a certain **inflation radius**, this variable is usually user defined.

This package makes use of **layers**. These are separate occupancy grid maps, divided by functionality, that work in an additive manner. While layer types can be further customized by writing custom plugins the predefined layer types are:

- **Static Map Layer** : The static map layer contains information on static obstacles that are, usually, permanent. Obstacles that can belong to this category are, for example, walls, shelves and other heavy furnishings. This is usually information that is already known before even moving the robot.

Maps built with SLAM are usually loaded into this layer.

- **Obstacle Map Layer** : The obstacle map layer carries sensor information about the current obstacles in the environment. Any obstacle that is not included in the static map layer is shown here.
- **Inflation Layer** : The inflation layer does not contain real-world information but is used as an optimization layer. It inflates the cost around lethal obstacles. This radius is usually the same or similar to the radius of the mobile base. This way the resulting costmap can more accurately represent the free configuration space.

3.5.4 Move Base Package

The **move_base** package [55] as part of the navigation package[56] . The navigation package is a 2D navigation stack that receives sensor and odometry data and sensor transforms to handle the navigation packages such as the planner packages, both global and local, build the global and local costmaps and send Twist messages to the mobile base for movement. The **move_base** package is tasked with handling the goal-reaching action of the navigation stack. Given a goal and all sensor information and obstacle information it will try to reach the goal location.

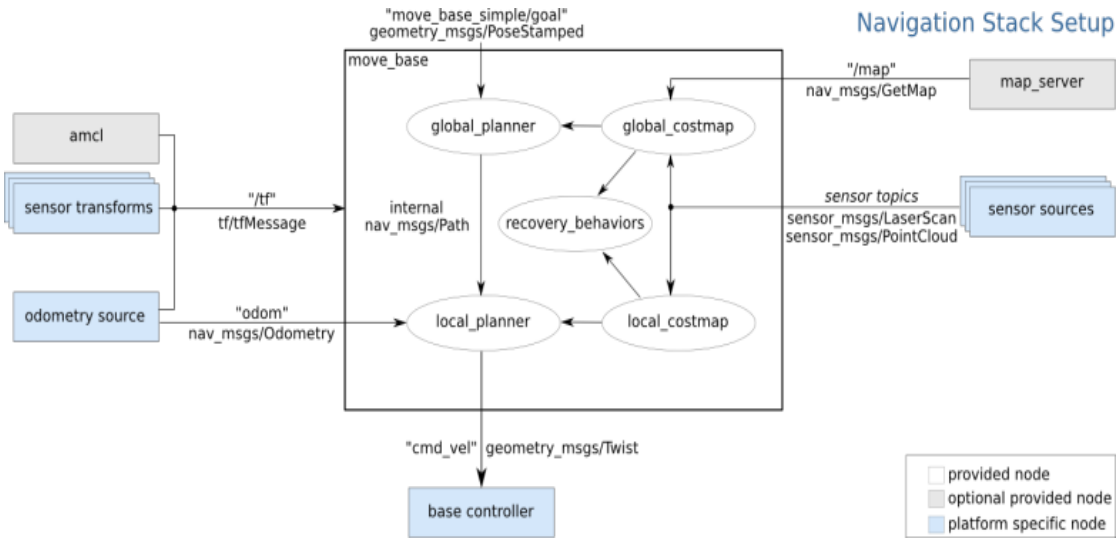


Figure 3.9: High level move_base package overview from [55]

As shown in Figure 3.9 the package takes information from the map publisher, the sensor streams and the transform tree to build a local and global costmap. The global map contains a-priori information about obstacles inside all of the mapped

space. The local costmap usually has a much lower size than the global map, limited by the range of the sensors on the robot. The global and local planner use their respective maps to plan the robot's trajectory. While the former uses only information from the global costmap and the received goal, the latter uses both the plan generated by the global planner and the obstacle information from the local costmap. The resulting trajectory is then translated to velocity commands for the robot base.

The base package comes with a global and local path planner preinstalled, but these can be changed at any time with custom plugins and modified easily through the use of configuration files.

Global and Local Planner Plugins

Path planners are implemented as plugins for the **move_base** package [55], by modifying its configuration files. Both the global and local planners are further modified according to the plugins used. We leveraged this flexibility in our human path avoidance method, by adding costmap layers to these planners.

Chapter 4

Hardware description of the Locobot WX250

The robot selected for our thesis work is the Locobot WX250 produced by Trossen Robotics [57]; here in Figure 4.1, we can see an image of the mobile platform taken from their website:

Figure 4.1: Mobile manipulator Locobot WX250-6DOF. Front view (left), lateral view (right)

We can identify three subsystems composing the robot itself that we will analyze in deep in the following sections: the **sensors**, the **manipulator**, and the **mobile base**. For evaluating the sensors measurements, the robot comes with a **NUC computer**, shown in Figure 4.2, with the following characteristics: 8th Gen Intel Dual-Core i3, 8GB DDR4 Ram, 240GB Solid State Drive (SSD), Intel Iris Plus Graphics 655, Wifi, Bluetooth 5.0, Gigabit Ethernet, USB, Thunderbolt 3, Ubuntu 20.04. The entire system has two batteries: one for the Kobuki base and another of 50000 mAh responsible for powering up the computer, the sensors, and the manipulator's motors.

Figure 4.2: Intel NUC NUC8i3BEH Mini PC

4.1 Robot hardware description

4.1.1 Mobile base

The mobile base is a **Kobuki** base that has **two driving wheels** so that the rover can steer thanks to the fact that each wheel is independent. In front, instead, we have a freewheel. The base is capable of sensing if we are near a cliff or, thanks to the **active bumper**, to sense if we are slamming into an obstacle. In Figure 4.3, we can see an image of the Kobuki base [57]:

Figure 4.3: Kobuki YMR-K01-W1, mobile platform used for navigation

4.1.2 6 DOF Manipulator

The manipulator is the second subsystem composing the entire robot. It is controlled thanks to the **DYNAMIXEL X Series** servos, and it is an **anthropomorphic arm** with six degrees of freedom. It has a spherical wrist, so it is capable of reaching numerous poses in space. In Figure 4.4, we report an image of the arm and the main technical characteristics in Table 4.1.

Figure 4.4: WidowX 250 Robot Arm used for manipulation

Degrees of freedom	6
Reach	680 mm
Span	1360 mm
Working payload	250 g
Repeatability	1 mm

Table 4.1: Arm main characteristics

Product of exponentials method

If we want to have a look at the direct and the inverse kinematics of the manipulator, we have first to introduce the **product of exponential** method (POE). As can be seen from the official vendor website [57] of the manipulator, the transformation between the base frame and the end-effector frame can be described by two matrices and the vector imposing the joint angles. The method is well presented in [58] with a particular emphasis on the fact that it could be an alternative to the Denavit-Hartenberg convention. This technique utilizes a first matrix $\mathbf{M}$, which describes the initial position of the manipulator defined as "zero configuration":

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0.452575 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.36025 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then, we can define a second matrix $\mathbf{S}$, which gives us the contribution of the i^{th} joint angle to the linear and angular motion of the end-effector frame. For each joint of the kinematic chain, we can define $\mathbf{q}$ as the vector of the position of the

joint and the vector ω as the axis of rotation of the joint with respect to the base frame. Analyzing the theory we can write that:

$$\mathbf{S_i} = \begin{pmatrix} -\omega_\mathbf{i} \times \mathbf{q_i} \\ \omega_\mathbf{i} \end{pmatrix}$$

where:

- $\mathbf{v_i} = -\omega_\mathbf{i} \times \mathbf{q_i}$ expresses the linear motion
- $\omega_\mathbf{i}$ expresses the angular motion

So, for each joint i^{th}, we can derive its contribution to the end-effector in terms of linear and angular motion. Below we report the computations in order to obtain the **S** matrix which expresses the contribution of each joint to the final pose of the gripper.

$$\mathbf{S_i} = \begin{bmatrix} 0 & -0.11025 & -0.36025 & 0 & -0.36025 & 0 \\ 0 & 0 & 0 & 0.36025 & 0 & 0.36023 \\ 0 & 0 & 0.05 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the following Table 4.2, we can resume the computation made in order to obtain the proposed S matrix.

$Joint_n$	q_i	ω_i	$-\omega_i \times q_i$
$Joint_1$	$(0,0,0)^T$	$(0,0,1)^T$	$(0,0,0)^T$
$Joint_2$	$(0,0,0.11025)^T$	$(0,1,0)^T$	$(-0.11025,0,0)^T$
$Joint_3$	$(0.05,0,0.36025)^T$	$(0,1,0)^T$	$(-0.36025,0,0.05)^T$
$Joint_4$	$(0.22155,0,0.36025)^T$	$(1,0,0)^T$	$(0,0.36025,0)^T$
$Joint_5$	$(0.3,0,0.36025)^T$	$(0,1,0)^T$	$(-0.36025,0,0.3)^T$
$Joint_6$	$(0.365,0,0.36025)^T$	$(1,0,0)^T$	$(0,0.36025,0)^T$

Table 4.2: Table of exponential parameters

Once having computed that, for a certain configuration of the joints, we can compute the matrix from the base frame to the end-effector frame as:

$$\boldsymbol{T}(\theta) = e^{[\boldsymbol{S_1}]\theta_1} \cdots e^{[\boldsymbol{S_6}]\theta_6} \boldsymbol{M} \tag{4.1}$$

$e^{[S_i]\theta_i}$ will be composed of a rotational and a translational part. The rotational part will be derived as follows:

$$\boldsymbol{R} = \boldsymbol{I} + \boldsymbol{\omega_m}\sin(\theta) + \boldsymbol{\omega_m^2}(1 - \cos(\theta)) \tag{4.2}$$

where:

$$\boldsymbol{\omega_m} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

Instead, the translational part will be derived as follows:

$$\boldsymbol{t} = (\boldsymbol{I} - \boldsymbol{R})(\boldsymbol{\omega} \times \boldsymbol{v}) + \boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{v}\boldsymbol{\theta} \tag{4.3}$$

Finally, we obtain that:

$$e^{[\boldsymbol{S_i}]\theta_i} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

What is the purpose and the advantage of this representation with respect to the classical Denavit-Hartenberg convention? As stated in [59], generally, for a calculator, is simpler from the computational point of view to use this method because the inverse kinematic, as proved in the cited paper, will not involve the computation of the Jacobian but will exploit an iterative algorithm.

4.1.3 Gripper description

The mobile manipulator comes with a gripper composed of two parts that can span from an open configuration with 4 cm of space between the fingers to a configuration where they are closed. From experience matured analyzing multiple medicines boxes, we comprehend that the average width of a package is about 6.5 cm. For this reason, and thanks to the fact that the end-effector can be easily substituted, we produced a model of the gripper that we will implement. It is about 4 cm larger when open so that the gripper can grasp a greater number of types of items without being too precise in positioning. In Figure 4.5 and Figure 4.6, we can have a look at the new end-effector and the measures used for construction, respectively.

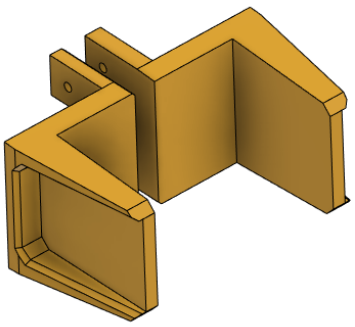


Figure 4.5: New gripper designed to substitute the original one

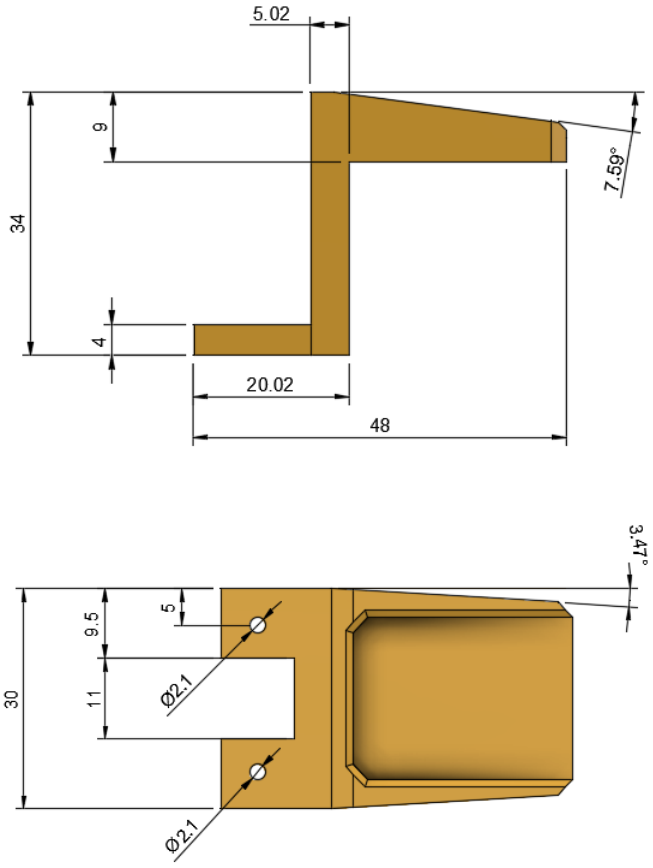


Figure 4.6: Lateral view (top) and top view (bottom) of the finger with some relevant measurements

The gripper once drawn is then printed using a 3D printer. The fingers produced

are made in PLA plastic which has good properties in durability and flexibility. The measures of the new gripper in **millimetres** are reported in Figure 4.6.

4.2 Robot sensors

The sensors available on the Locobot are :

- **Wheel Encoder**
- **RPLidar A2M8** (360° 2D Lidar)
- **Intel RealSense D435** (Stereo RGB-D)

The **RPLIDAR A2M8** is a 360° 2D Lidar and stands on top of the LocoBot at 62.28 cm. The main advantage of a 2D Lidar over normal cameras is their wide horizontal FoV, longer depth measurement range and measurement consistency. Their main drawback is the lack of vertical resolution, meaning that sensing only happens on the plane orthogonal to the laser direction. This means that obstacles that are lower or higher than the 2D Lidar height cannot be detected, in an office-like environment this can be anything from chairs to boxes to sensitive equipment such as computer towers. While effective in gathering direct, high-resolution range information, the Lidar sensor available is not sufficient to ensure a correct and safe exploration of the target space. The **Intel RealSense D435** is an RGB-D sensor mounted with tilt and pan capabilities. It consists of an RGB module for image capture and a stereoscopic system (Intel RealSense Module D430) for depth perception. RGB sensors are widely used in robotics and the added Depth information only increases their usefulness. The colour information taken from the camera is one of the most important differences from LiDAR and is exploited in many tasks such as segmentation or object detection. They also tend to be cheaper than LiDAR sensors; however, the FoV of cameras is typically limited on the horizontal axis as is the depth detection range. The vertical axis instead is larger than the A2M8 and gives full 3D depth perception up to 6m, but with an error rate larger than 2% after the 2m range.

Figure 4.7: Intel® RealSense™ Depth Camera D435

Figure 4.8: RPLIDAR A2M8 360° Laser Range Scanner

Chapter 5

Description of the robot's communication structure

5.1 Communication structure

In our intentions, the robot's software architecture has to maintain a modular structure so that the debugging operations will be simpler for whoever wants to update or enrich the application's functionalities. Moreover, we thought that each subsystem should have its own controller so that the robot would work even with a subsystem failure.
The structure is composed of three nodes, as can be seen in Figure 5.1:

- **Base controller**: it is responsible for exchanging information with the communication manager about the working status of the hardware or the goal position's planning success or failure.

- **Arm controller**: it is accountable for planning the motion of the servos to reach the desired pose. It also controls the opening or closure of the gripper by publishing to a specific topic. As for the base controller, the arm controller will also communicate the hardware working status to the communication manager.

- **Communication node**: its task is to manage the ARTag present in the scene and the communication with base and arm. In particular, the node reads the IDs of the ARTags present in the scene and compares them with the ID request. When there is a match between a requested id and a marker present in the environment it will send the commands to start the pick/place routine. Along with these operations, it has a failure-handling mechanism.

In Figure 5.1, we present the core structure governing the mobile manipulator's behaviour.

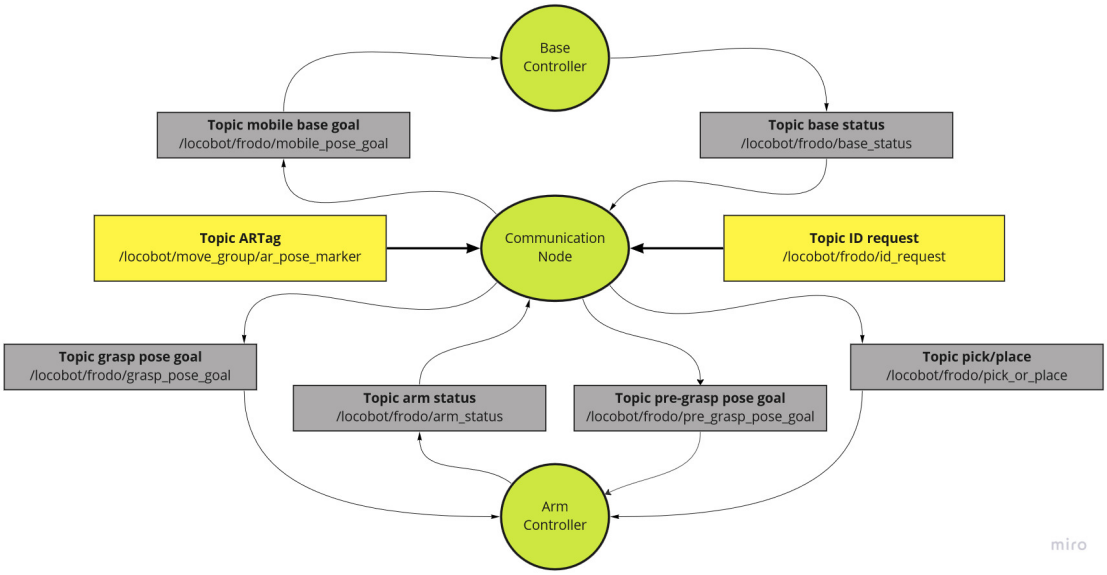


Figure 5.1: Structure of the software implemented for performing the task

Now we will deeply analyze how the **communication node** deals with an id request, how it **searches** for an id not present in the scene, and how it manages the **pick** and **place** phases. Essentially the goal of the communication node is to deal with a request for an id and manage the operations of the mobile base and the arm. This is achieved through the status topics and sending a pose goal to each component. For safety reasons, we decided to deal with each component individually, so the arm can run only if the base is idle and vice-versa.

5.2 Item request handling

The first thing that has to be done when receiving a request for an id is to search if it is present in the robot's field of view. The software will listen to the **ar_pose_marker** topics which contain details such as pose and orientation characterizing each marker. To ensure that the arm and the mobile base have a correct reading at each time instant without having to perform any transformation online we set up two different **ar_track_alvar** nodes so that ones computes the positions for the mobile base (so with respect to "map" frame) and another one that estimates the position with respect to "locobot/base_footprint" which is the **planning frame** of MoveIt.
The id request is handled following the pseudocode presented in Algorithm 1:

Algorithm 1 Algorithm for handling id requests through a callback

```
Require: BASE_IDLE
Require: ARM_IDLE
  if ARTag FOUND then                          ▷ Request correct, start picking
    Send goal to the mobile base
  else if ARTag NOT_FOUND then
    Enter in search mode      ▷ Search phase explained in the following section
  else
    return                                     ▷ Request malformed, return error
  end if
```

5.3 Search phase

If an item is **not found** among all the available in the scene the robot enters a function that performs a series of predefined actions presented in the following Algorithm 2. The function has some parameters that can be tuned according to the application's necessities: we can tune the spots to be investigated and the degrees of rotation that the mobile base performs at each iteration.

Algorithm 2 Algorithm implemented for the search function of the robot

```
Require: BASE_IDLE
Require: ARM_IDLE
  while ARTag NOT_FOUND do
    if retry ≤ 2 then          ▷ If reached maximum retry exit with error state
      while Rotation ≤ 360 degrees do
        Rotate base about 30 degrees     ▷ Rotate about 30 degrees on itself
        if ARTag FOUND then
          return FOUND
        end if
        Move base to another spot          ▷ Decided previously by the user
      end while
    else
      return NOT_FOUND
    end if
  end while
```

The algorithm presented above tries to explore the surrounding environment moving the base first with a rotation and then searching in another spot. For each spot, a complete rotation of 360 degrees is made to ensure that the environment is being explored. In Figure 5.2 we can have a look at how the search phase is performed:

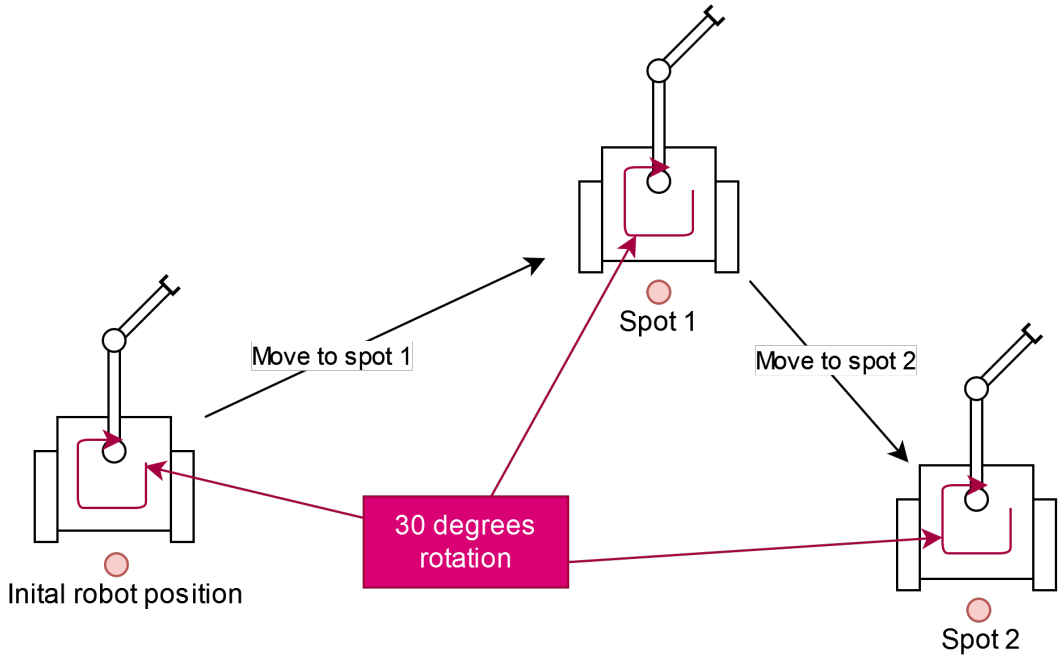


Figure 5.2: How the algorithm of search works

If, for example, a request marker is found during any moving action, the software will break the search function to enter the picking phase.

5.4 Fail handling

The communication manager is also accountable for managing situations in which the arm or the mobile base fails. In particular, each component is expected to enter a repositioning phase that changes depending on who failed. This failure handling is based on assumptions: the robot base or arm hardware can not fail, and the robot does not get stuck with obstacles. Having said that, we managed only the fails related to the software part or, more in general, the ones related to failed planning for both the base and the manipulator. For this reason, the mobile base and the manipulator will check, before executing any movement, the feasibility of the planned path.

5.4.1 Arm fail

To deal with an arm failure that can be caused, for example, by incorrect positioning we exploit the mobile base as another degree of freedom so that the arm can be

repositioned. If the arm fails, we decide to reposition the base so that the behaviour is similar to the one described in the search phase. The following pseudocode (Algorithm 3) describes the working principle implemented in the code:

Algorithm 3 Arm fail handling in the communication manager

```
Require: BASE_IDLE
Require: ARM_FAIL
  if PICK then
      if retry ≤ 2 then                    ▷ If reached maximum retry exit
          Reposition base
          Compute then new ARTag pose
          Send goal to arm
      else
          Repositioning failed
      end if
  else if PLACE then
      if retry ≤ 2 then                    ▷ If reached maximum retry exit
          Reposition base
          Send goal to arm                 ▷ Do not need ARTag for place
      else
          Repositioning failed
      end if
  end if
```

The arm will enter **fail mode** so the communication manager can handle the repositioning request. The request is composed of different phases. In the first phase, the arm controller node will send to the communication manager the status **FAIL** to communicate that probably the position assumed by the base is wrong so it requires a repositioning. This phase consists in:

- Return to the **HOME** position
- Restart a search phase to re-estimate the correct marker position

This procedure can be done a maximum of two times before considering the item unreachable and returning home ready to take another command. An example of an arm fail-handling procedure is presented in Figure 5.3.

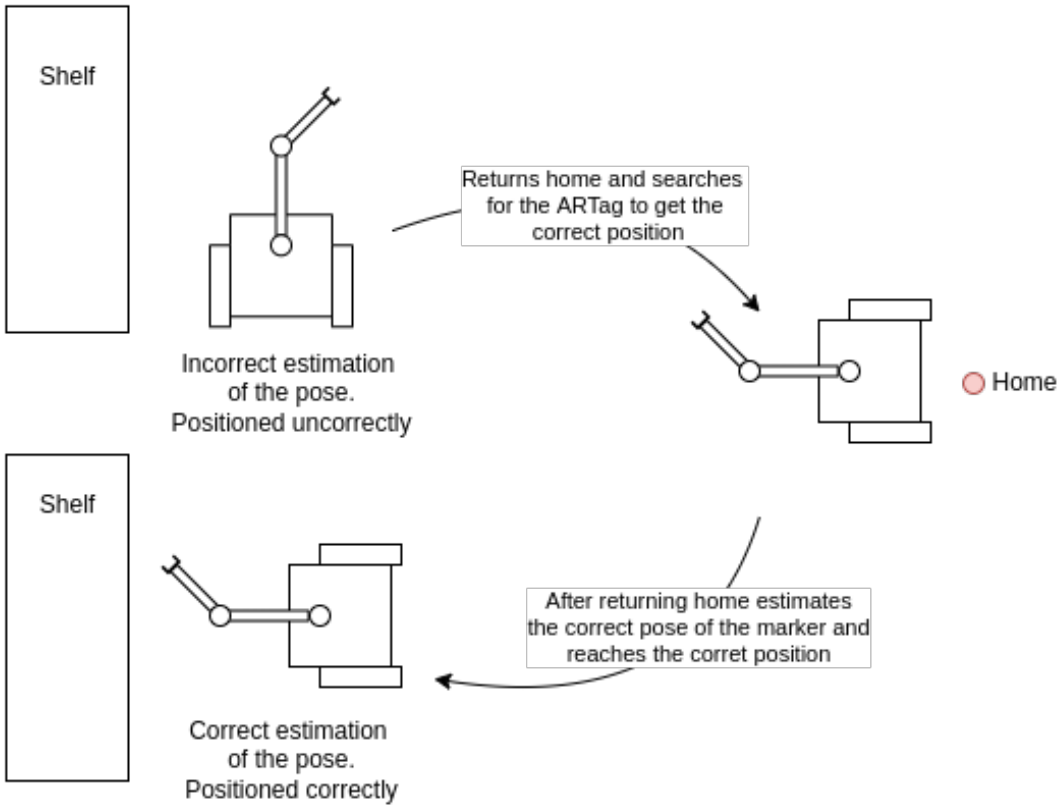


Figure 5.3: How the mobile base is intended to reposition returning from the incorrect pose to home and then repositioning in front of the correct shelf.

5.4.2 Base fail

Goals sent to the mobile base are already confirmed as valid using the exposed API on the navigation stack of the global path planner. These functions let us check the validity of a goal by generating a plan without executing it. When an id request is matched to a visible ARTag in the scene the communication node calculates a goal pose in front of it. It then checks with the aforementioned API if the given goal is feasible or not. If not, then the communication manager node tries to increase the distance of the goal pose from the ARTag until a correct plan is generated, it attempts this step no more than twice before giving up on the given goal. This is done following the intuition that, due to sensor noise and map uncertainty, a found ARTag pose might not match the real position or it might be slightly inside the shelf.

5.5 Pick and place routines

The pick-and-place routine is constructed considering the interaction between the mobile base and the arm that can interface through the communication node. Figure 5.4 shows the main steps performed by the software to pick and place an object.

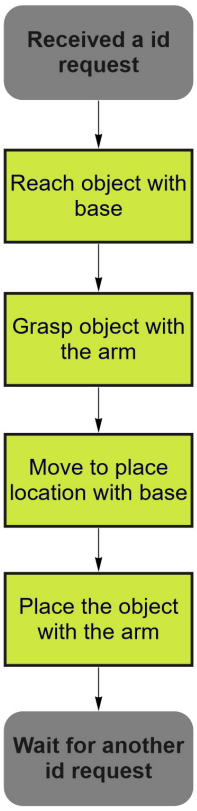


Figure 5.4: Actions performed for pick and place of an object

In all the intermediate steps, the system that has to run next waits until the previous one has finished; this is achieved by implementing the status topic where the subsystem can communicate whether they are running, failing, or succeeding. The movement of each subsystem could be divided into two sub-movements, one needed to move up to a certain distance from the target and one in which the component effectively reaches the goal pose. The reasoning behind doing this is to correct the pose estimation of the marker as we move toward it. The different phases will be further explained in the following chapters regarding the software architecture of the manipulator and the base.

Chapter 6

Software architecture of the manipulator

6.1 Communication structure

Now we can go in deep to analyze the role of the arm controller. It is constructed such that the node exchanges some information with the communication node. The topics used are the following: **arm_status**, **pick_or_place**, **grasp_pose_goal**, and **pre_grasp_pose_goal**. A general view of the communication structure is sketched in Figure 6.1.

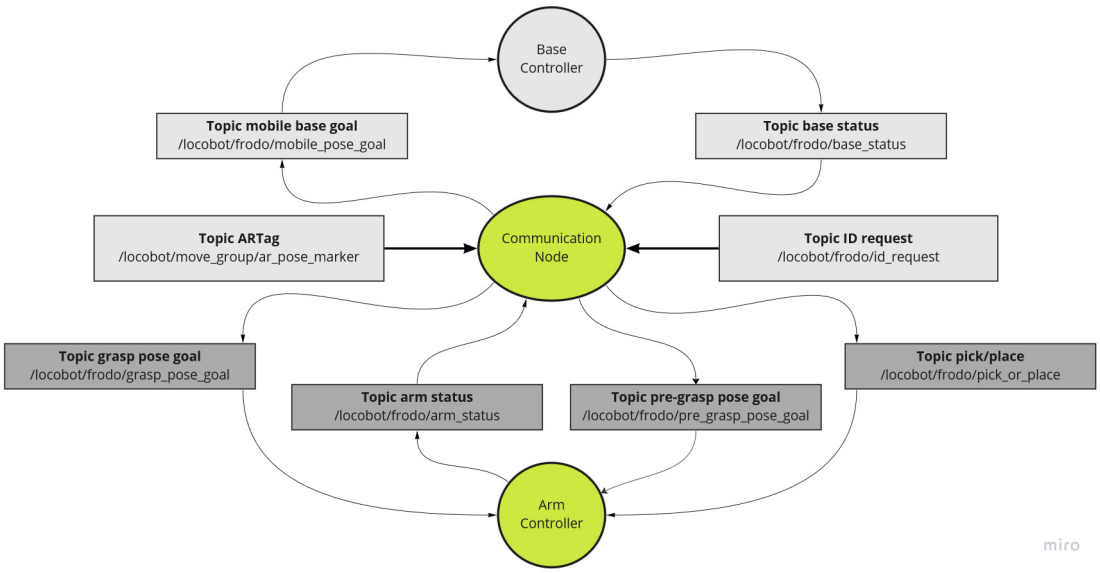


Figure 6.1: Manipulator node communication structure

6.1.1 Description of the arm_status topic

The first topic that we will analyze is the **arm_status** topic. It is capable of communicating four status that characterizes the working flow of the arm:

- **ARM FAIL**: it is implemented to allow the replanning of the mobile base if the pick or place actions fail.
- **ARM SUCCESS**: this status is the one sent at the end of a pick/place action that is performed successfully. After 5 seconds the status is changed back to **ARM IDLE**.
- **ARM IDLE**: this is a necessary step to make the communication node aware that the arm completed its motion. Without this status published the base cannot move.
- **ARM RUNNING**: this message is exchanged only to make the user aware that the **arm_controller** is running.

6.1.2 Description of the pose goal topics

These topics, **grasp_pose_goal** and **pre_grasp_pose_goal**, are constructed only to exchange the two pose goals, that the arm will exploit to perform its motion. The messages sent and received are of type **geometry_msgs/PoseStamped**, so they store information such as position, orientation, and frame of reference for which those values are computed.

6.1.3 Description of the pick_or_place topic

The routine of movements to be performed changes depending if we are dealing with picking or placing. We construct the following topic to make the arm aware of the sequence of action to be executed. As we know, the options are **PICK** or **PLACE**.
According to this subdivision, we split the routine into two parts depending on what has to be performed, as can be seen in Figure 6.2. The main difference between these two phases regards the update of the planning scene and the presence or not of some intermediate poses to approach the target position.
Once the **arm controller** has received the goal pose and the action to be achieved, the routine starts and performs the desired moves taking into account the planning scene. Here we report the main steps that the manipulator performs in each situation:

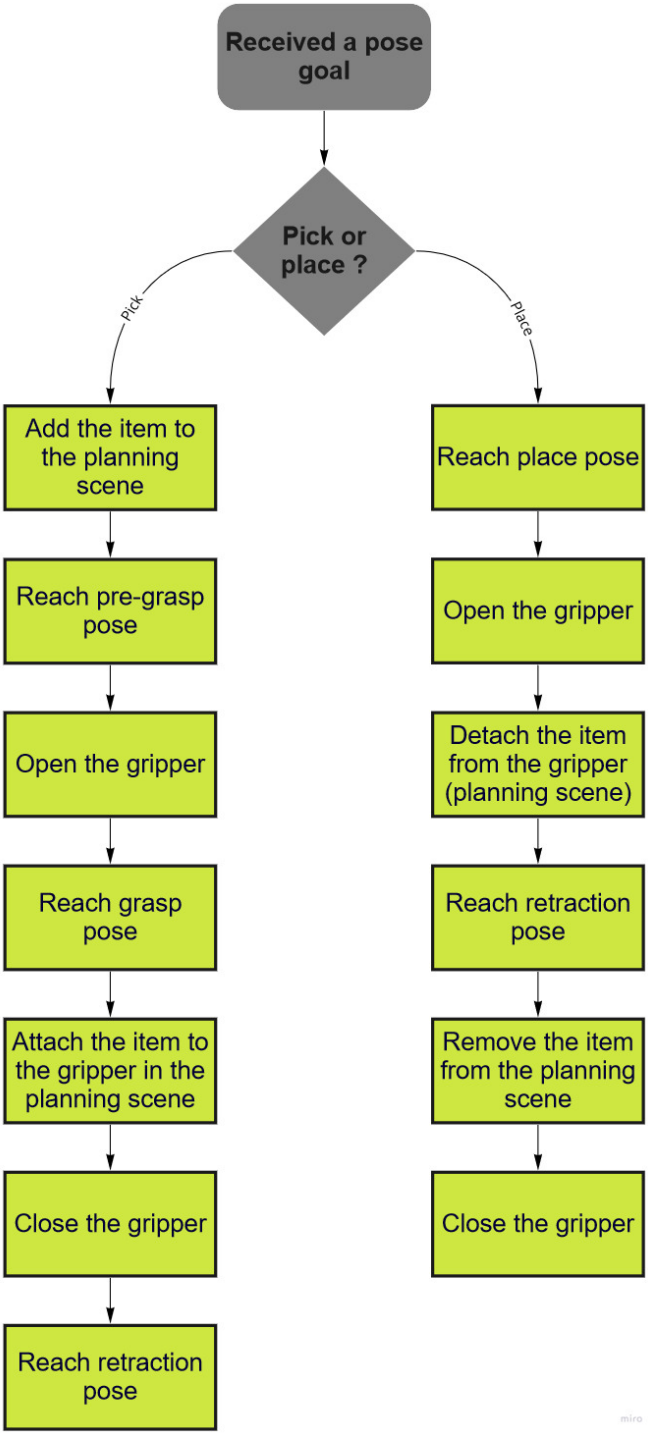


Figure 6.2: Flowchart for understanding in which order the actions are performed.

Pick routine

As reported in Figure 6.2, the picking routine is the one that has the highest number of actions to be performed. The steps that take place, from the motion planning point of view, are:

1. Perform the motion to go in the **pre-grasp pose** as can be seen in Figure 6.3 and Figure 6.4.
2. Give the command to **Open** the gripper.
3. Move in the **grasp pose** as depicted in Figure 6.5 and Figure 6.6.
4. Give the command to **Close** the gripper.
5. Perform the motion to go in the **retraction pose** as described in Figure 6.7 and Figure 6.8.

At the same time, we have to update the planning scene so that the motion planner will take into account the presence of the item attached to the end-effector. To do so we have to perform the following actions in sequence:

1. **Add** the medicine in the planning scene.
2. **Attach** the item to the gripper so that the motion planner can consider the item's presence.

In the following figures, we can describe how the grasp, pre-grasp, and retraction poses are computed. First, we must remember that we have the position and the orientation of the ARTag and the end-effector in the space with respect to the map frame. Knowing this, we can obtain the roto-translational transformation from the ARTag frame to the map frame. To obtain the poses, we define a point with the orientation of the end-effector 10 centimetres along the z-axis of the ARTag then we apply the following transformation.
Given:

$$\mathbf{T}^{map}_{ARTag\,frame} \in \mathbb{R}^{4x4}$$

which is the homogenous roto-translational matrix that expresses the rotation and translation from the map frame to the ARTag frame and it is composed of:

$$\mathbf{T}^{map}_{ARTag\,frame} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\mathbf{R} \in \mathbb{R}^{3x3} \text{ rotational square matrix}$$

$$\mathbf{t} \in \mathbb{R}^{3x1} \text{ translation vector}$$

we can define the vector of the new position in the ARTag frame as:

$$\mathbf{pre\_grasp}_{ARTag\,frame} \in \mathbb{R}^{3x1}$$

$$\mathbf{pre\_grasp}_{ARTag\,frame} = \begin{bmatrix} 0 \\ 0 \\ 0.1 \end{bmatrix}$$

and then apply the transformation to calculate its components in the map frame by multiplying the $\mathbf{pre\_grasp}_{ARTag\,frame}$ vector for the $\mathbf{T}^{map}_{ARTag\,frame}$ matrix:

$$\mathbf{pre\_grasp}_{map} \in \mathbb{R}^{3x1}$$

$$\begin{bmatrix} \mathbf{pre\_grasp}_{map} \\ 1 \end{bmatrix} = \mathbf{T}^{map}_{ARTag\,frame} \begin{bmatrix} \mathbf{pre\_grasp}_{ARTag\,frame} \\ 1 \end{bmatrix} \tag{6.1}$$

so that we obtain the $\mathbf{pre\_grasp}_{map}$ position vector, which is the coordinates that the end-effector has to reach before performing the grasp of the object.
The first phase, as said previously, consists of the approach to the object to be grasped. The end-effector assumes the **pre-grasp pose**. To know how the gripper is positioned, we can look at Figures 6.3 and 6.4.

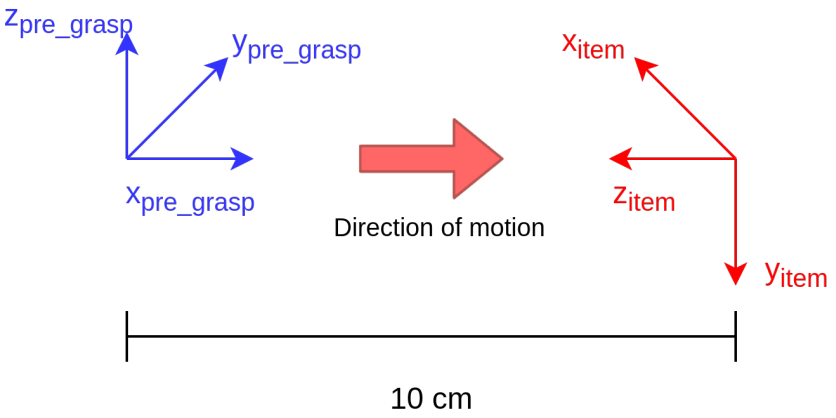


Figure 6.3: Definition of the pre-grasp pose with respect to the item, 10 cm away along the z-axis of the item

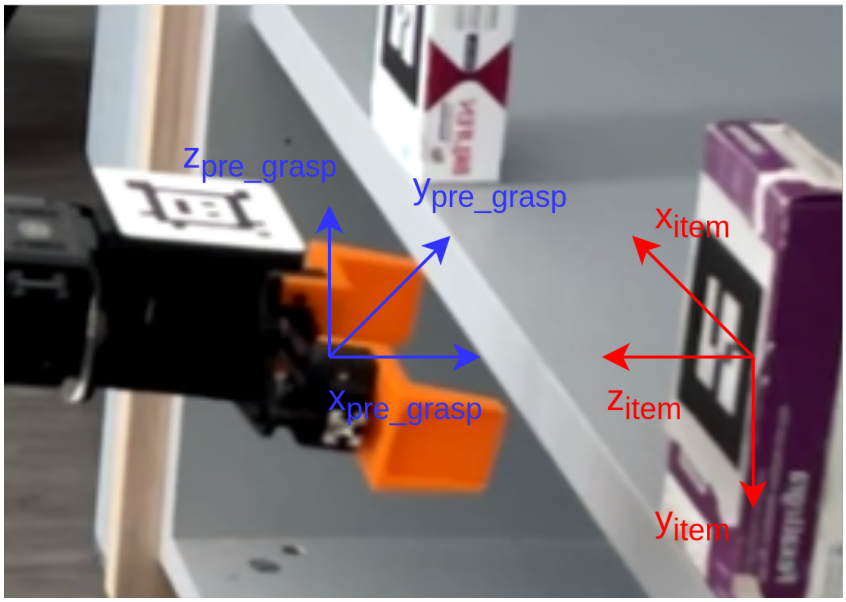


Figure 6.4: Pre-grasp pose example taken from the experimental phase where we can see the end effector and the item to be fetched

Then, we can define the grasp pose as the same coordinate of the marker with a different orientation. Figures 6.5 and 6.6 depict the situation when reaching this pose.

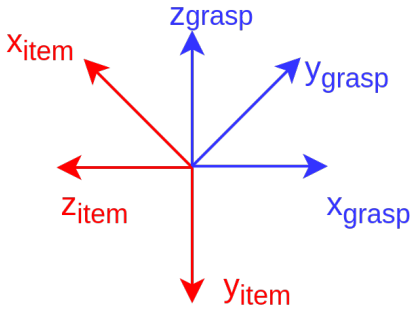


Figure 6.5: Pose of the end-effector after reaching the grasp pose for fetching the item

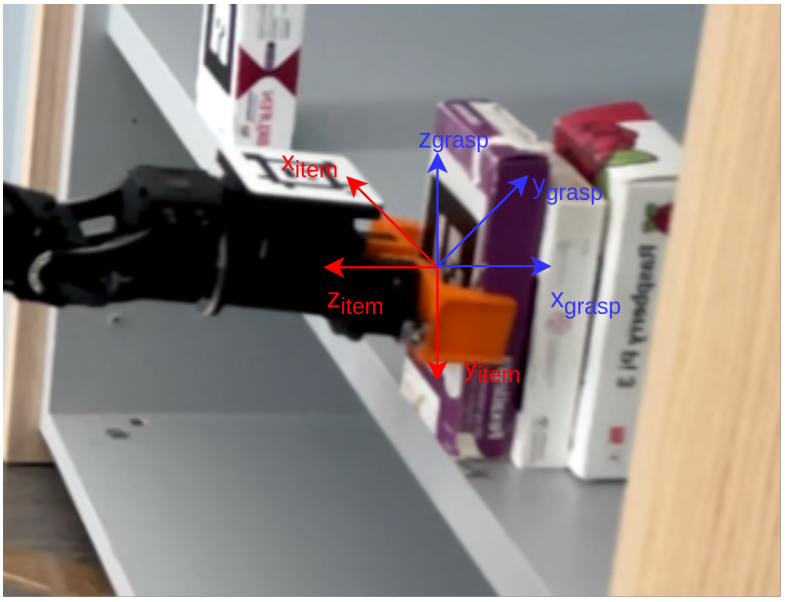


Figure 6.6: The end-effector here grasped correctly the item reaching its position with a different orientation

Similarly to what we say for the pre-grasp pose, we can define the retraction pose as 10 centimetres away along the x-axis and 10 centimetres higher to simulate the action normally a human would perform. The resultant final position after retraction can be seen in Figure 6.7 and Figure 6.8.
Given:

$$\mathbf{T}^{map}_{ARTag\,frame} \in \mathbb{R}^{4x4}$$

which is the homogenous roto-translational matrix that expresses the rotation and translation from the map frame to the ARTag frame and it is composed of:

$$\mathbf{T}^{map}_{ARTag\,frame} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

$\mathbf{R} \in \mathbb{R}^{3x3}$ rotational square matrix

$\mathbf{t} \in \mathbb{R}^{3x1}$ translation vector

we can define the vector of the new position in the ARTag frame as:

$$\mathbf{retraction}_{ARTag\,frame} \in \mathbb{R}^{3x1}$$

$$\mathbf{retraction}_{ARTag\,frame} = \begin{bmatrix} 0 \\ -0.1 \\ 0.1 \end{bmatrix}$$

and then apply the transformation to calculate its components in the map frame by multiplying the $\mathbf{retraction}_{ARTag\,frame}$ vector for the $\mathbf{T}^{map}_{ARTag\,frame}$ matrix:

$$\mathbf{retraction}_{map} \in \mathbb{R}^{3x1}$$

$$\begin{bmatrix} \mathbf{retraction}_{map} \\ 1 \end{bmatrix} = \mathbf{T}^{map}_{ARTag\,frame} \begin{bmatrix} \mathbf{retraction}_{ARTag\,frame} \\ 1 \end{bmatrix} \tag{6.2}$$

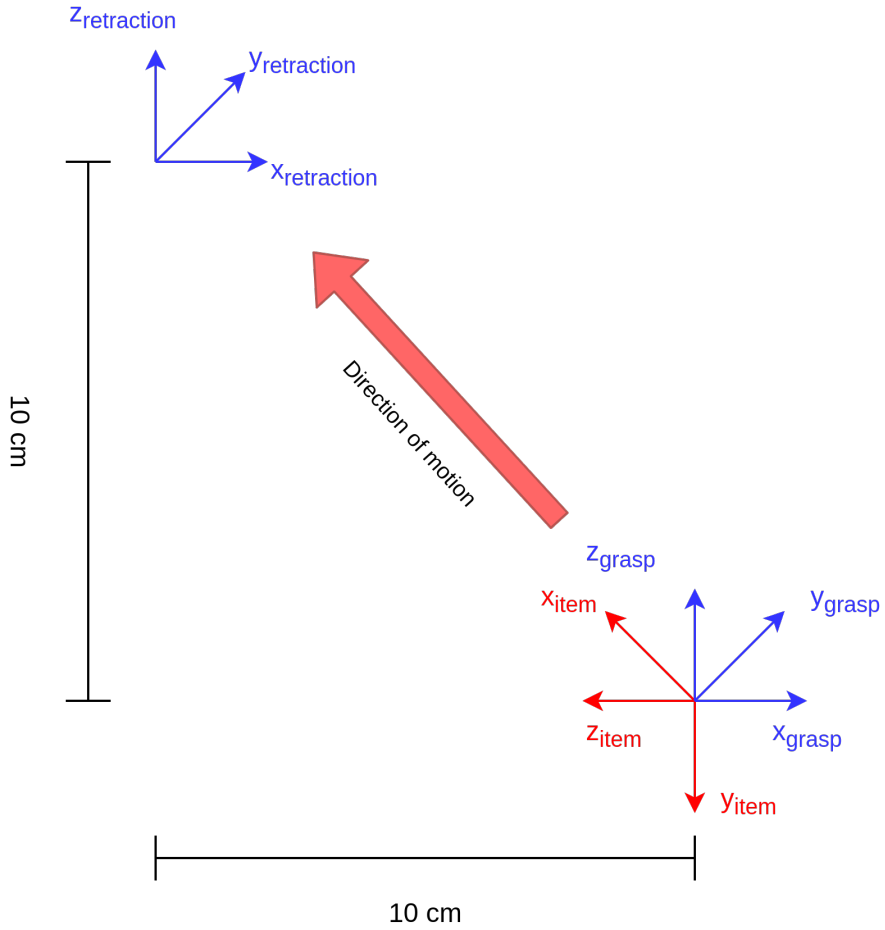


Figure 6.7: Definition of the retraction pose starting from the grasp pose, 10 cm away along the z-axis and the y-axis of the ARTag frame

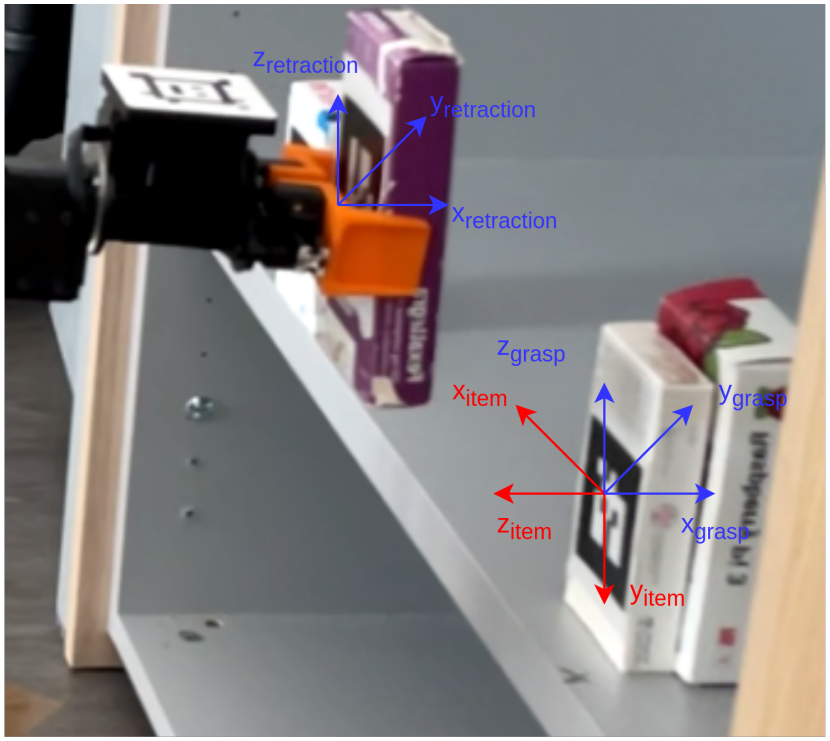


Figure 6.8: Here the picking is finished so the manipulator reaches the retraction pose with the item in its hand

Place routine

After having performed the picking routine we have to place the object so that, after the base moved correctly, the arm has to do the following actions that constitute the **place routine**:

1. Perform the motion to go in the **target place pose** as shown in Figure 6.9.
2. Give the command to **Open** the gripper.
3. Perform the motion to go in the **retraction pose** as depicted in Figure 6.10.
4. Give the command to **Close** the gripper.

Also, in this case, the planning scene has to be updated to consider the object's presence in the grasping hand. Since we added and attached the item in the previous routine, we have only to perform two actions:

1. **Detach** the medicine from the end-effector.
2. **Remove** the object from the planning scene.

The position reached to place the item is shown in Figure 6.9 where we can see the item's reference frame and the end-effector's reference frame.

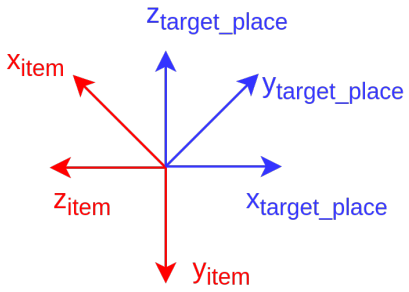


Figure 6.9: The end-effector with the object attached reaches the placing pose that the user previously decided through the code.

The reasoning for defining the post-place pose is the same exploited for the pre-grasp position. As previously, we define the position as the one 10 centimeters positive along the z-axis of the ARTag frame depicted in Figure 6.10.
Given:

$$\mathbf{T}^{map}_{ARTag\,frame} \in \mathbb{R}^{4x4}$$

which is the homogenous roto-translational matrix that expresses the rotation and translation from the map frame to the ARTag frame and it is composed of:

$$\mathbf{T}^{map}_{ARTag\,frame} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

$\mathbf{R} \in \mathbb{R}^{3x3}$ rotational square matrix

$\mathbf{t} \in \mathbb{R}^{3x1}$ translation vector

we can define the vector of the new position in the ARTag frame as:

$$\mathbf{retraction}_{ARTag\,frame} \in \mathbb{R}^{3x1}$$

$$\mathbf{retraction}_{ARTag\,frame} = \begin{bmatrix} 0 \\ 0 \\ 0.1 \end{bmatrix}$$

and then apply the transformation to calculate its components in the map frame by multiplying the $\mathbf{retraction}_{ARTag\,frame}$ vector for the $\mathbf{T}^{map}_{ARTag\,frame}$ matrix:

$$\mathbf{retraction}_{map} \in \mathbb{R}^{3x1}$$

$$\begin{bmatrix} \mathbf{retraction}_{map} \\ 1 \end{bmatrix} = \mathbf{T}^{map}_{ARTag\,frame} \begin{bmatrix} \mathbf{retraction}_{ARTag\,frame} \\ 1 \end{bmatrix} \tag{6.3}$$

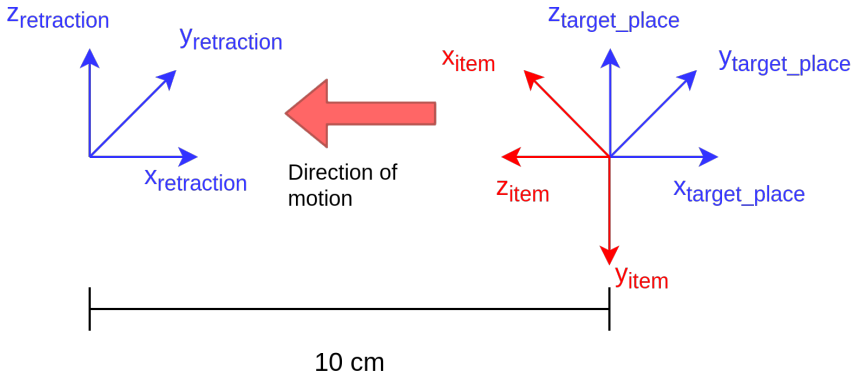


Figure 6.10: Position of the end-effector frame during a place action

6.2 Obstacle avoidance

Octomap [52] has been implemented in the robot to accomplish the obstacle avoidance task. The planning framework is always MoveIt which provides a plugin that can interact with the motion planning algorithm avoiding colliding with the objects in the scene. The scene model obtained by implementing Octomap is updated according to a specific rate established in a configuration file. The map can be canceled each time we want by calling a rosservice.

6.2.1 Octomap

The implementation of Octomap in the robot is achieved by adding a new launch file to the ones present in the original packages coming with the robot to maintain a modular structure as the original of Locobot, where we can define some parameters depending on the characteristics of our application. In particular, in our case, we set the following parameters reported in Table 6.1:

Parameter name	Value	Description
octomap_frame	"locobot/base_footprint"	Frame to which we refer Octomap
octomap_resolution	0.05 meters	Maximum accuracy in mapping
maximum_range	4 meters	Measurement range of the camera

Table 6.1: Octomap parameters used by the plugin in case we use the real robot

Here is the file implemented where, as can be seen, we can specify the fixed reference frame of the robot and tune some parameters based on the camera implemented. In our case, looking at the datasheet of the Intel Realsense camera, we imposed a maximum range of 4 meters. Since this is the file implemented in the simulation, a rough resolution is selected so that the computational effort is not so high. In the simulated robot, the **octomap_resolution**, where we can simulate an infinite dense pointcloud instead, the parameter is substituted with the one reported in Table 6.2:

Parameter name	Value	Description
octomap_resolution	0.02 meters	Maximum accuracy in mapping

Table 6.2: Octomap parameters implemented by the plugin in case we use the simulation of the robot

Once we declare the launch file, we can have a look at the specific configuration of the plugin [60] invoked to collaborate with MoveIt. It is composed of the following parameters:

- **sensor_plugin**: which specifies the plugin name that has to be invoked
- **point_cloud_topic**: which is the topic where we read the PointCloud messages
- **max_update_rate**: this sets the rate at which Octomap should update the map of the scene
- **filtered_cloud_topic**: that is the topic where we publish the filtered cloud of points for debugging reasons

In our particular case, the parameters used for the simulation of the robot are reported in Table 6.3:

Parameter	Value
sensor_plugin	"occupancy_map_monitor/PointCloudOctomapUpdater"
point_cloud_topic	"/locobot/camera/depth_registered/points"
point_subsample	1.0
padding_offset	0.1
padding_scale	1.0
max_update_rate	10.0

Table 6.3: Parameters for running Octomap in a simulated environment

For the real robot instead, as previously, in order to reduce the computational effort of the CPU of our NUC we decided to impose some delays in the code and to have an update rate 10 times slower as can be seen in Table 6.4:

Parameter	Value
sensor_plugin	"occupancy_map_monitor/PointCloudOctomapUpdater"
point_cloud_topic	"/locobot/rtabmap/depth/color/voxels"
point_subsample	1.0
padding_offset	0.1
padding_scale	1.0
max_update_rate	1.0

Table 6.4: Parameters for running Octomap in a real environment

6.3 Object recognition

As mentioned before, we decided to implement ARTags to recognize the medicines on the shelves. In particular, they will be used to retrieve the position and the orientation of those markers with respect to the camera frame. In our application, we could associate a marker with each object that must be grasped.This way, the object's pose can be retrieved and translated into a pose goal for the end-effector.

6.3.1 ARTag markers

The implementation in the robot, both in the real case and in the simulation, is obtained by exploiting a package of ROS called "**ar_track_alvar**" [61] that will generate a node subscribed to the topics of the camera and will compute the pose of the marker relative to the camera frame. To launch the node, we generate a launch file that constraints the parameters necessary to the package; their descriptions are reported in Table 6.5:

Name	Type	Description
marker_size	double	width in cm of the black square marker
max_new_marker_error	double	threshold to detect new marker
max_track_error	double	maximum tracking error of ARTag
camera_image	string	topic that provides camera frames for detecting ARTags
camera_info	string	topic of the calibration parameters
output_frame	string	frame to which we attach the frame of the marker

Table 6.5: Table of the parameters ar_track_alvar

For both the simulated and the real robot, we chose to use the following values reported in Table 6.6:

Name	Description
marker_size	5.65 cm
max_new_marker_error	0.08 cm
max_track_error	0.02 cm
camera_image	"/locobot/camera/color/image_raw"
camera_info	"/locobot/camera/color/camera_info"
output_frame	"locobot/base_footprint"

Table 6.6: Selected parameter for ar_track_alvar node

In Figure 6.11 we can see an ARTag recognized by the package where it placed a frame directly connected to the Locobot camera frame; this relation is clearer when analyzing the transformation tree characterizing the robot as in Figure 6.12.

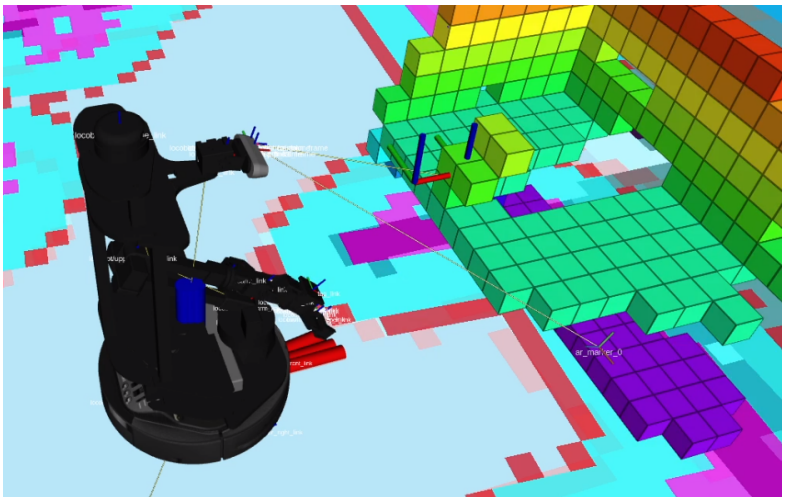


Figure 6.11: Visualization of the detected ARTag in Rviz

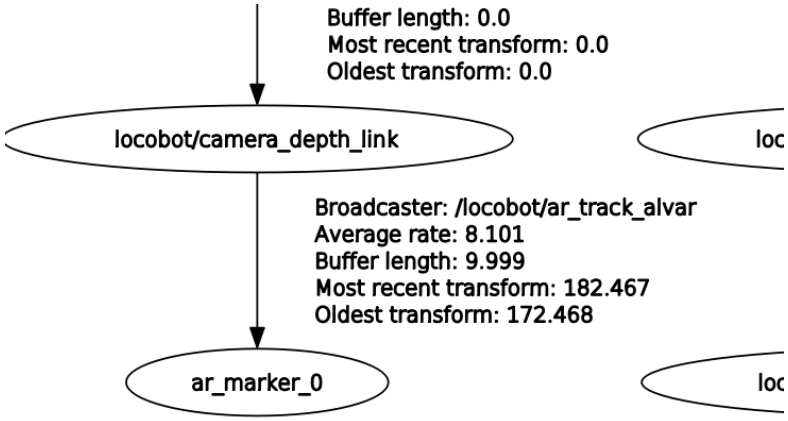


Figure 6.12: Representation of the transformation tree where we can appreciate the presence of the ARTag as connected to the camera link

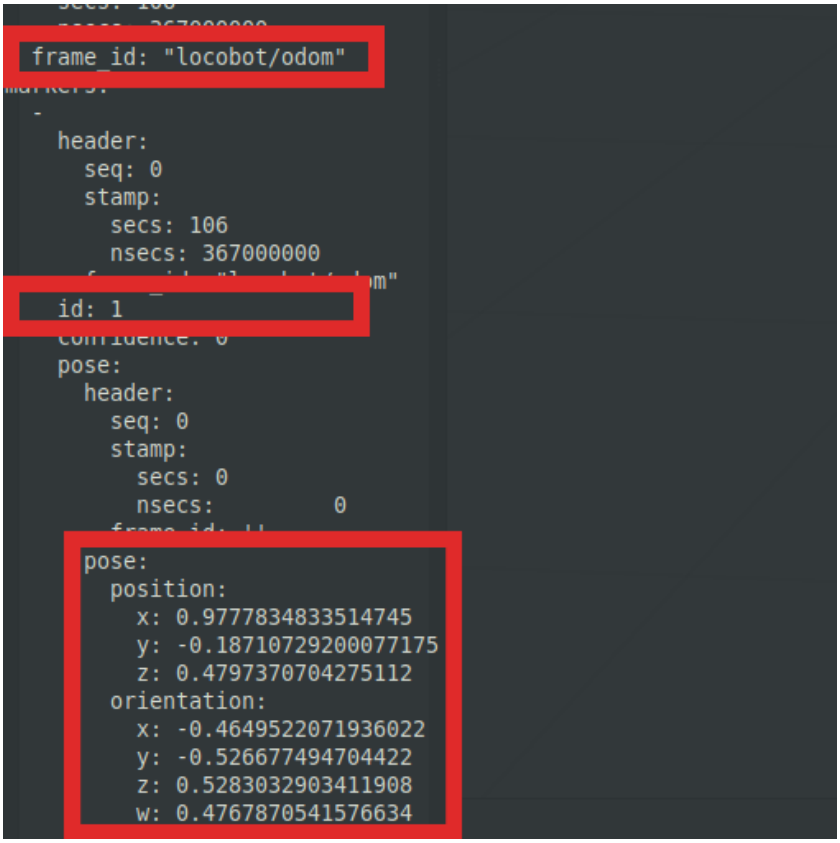


Figure 6.13: Message published on the dedicated topic where we can find the necessary information such as pose and ARTag id [61]

The node implemented, when detecting an ARTag, publishes a message of type "ar_track_alvar_msgs/AlvarMarkers", which gives us different information such as frame of reference, ID of the marker, position, and orientation, in the topic "/locobot/move_group/ar_pose_marker" as highlighted in Figure 6.13.

6.4 MoveIt framework

Before giving the details of each motion planner parameter we can have a look at the possible meaning of each descriptor, highlighting the most significant ones. Then, each planner is analyzed, explaining the possible tuning that can be made.

6.5 ROS mobile manipulator's motion planning algorithms

In this section, we will introduce the parameters that can be tuned per each motion planning algorithm, together with a brief description of their role.

6.5.1 OMPL planning library

Our particular application's main objective is to avoid collision with the shelves when picking a medicine. Since the main point is not optimizing quantities such as acceleration, velocities, and jerks, we only need to consider the geometric planner. Specifically, we take into consideration three algorithms that are widely used in MoveIt and come as default planners: **RRTConnect**,**RRT** and **RRT*** [50].
We analyze the role and the parameters of **RRT***, which, differently from the others, allows us to select the cost function to be optimized during the motion planning. In Table 6.7 we report the description of the parameters.

Parameter	Description
optimization_objective	It allows us to select the cost function to be minimized
range	it represents the maximum motion that can be added to the tree of motion
goal_bias	it represents the probability to choose, in the joint space, the actual goal state in attempt to go towards in exploring the space
delay_collision_checking	when set to 1 (true) delays the collision checking so that the planners sorts first the paths with the lowest cost function then it analyzes if they are coherent with the collision check.

Table 6.7: Parameters present in RRT* that we can select to tune the behavior of the motion planner

The cost function that can be selected in the **optimization_objective** parameter are the following ones:

- PathLengthOptimizationObjective
- MechanicalWorkOptimizationObjective
- MaximizeMinClearanceObjective
- StateCostIntegralObjective
- MinimaxObjective

Planner settings

We test the functionalities of **RRT*** as a motion planning algorithm. After evaluating the behavior of the motion planner with different parameters, we selected the following values reported in Table 6.8: We can see that the selected

Parameter	Values
optimization_objective	PathLengthOptimizationObjective
range	0.0
goal_bias	0.05
delay_collision_checking	1

Table 6.8: Parameters of RRT* selected for our implementation

cost function optimizes the path length above all the other quantities. This is essential since we must optimize the duration of the battery, and longer paths

would have meant more energy consumption. The other cost function considered was MechanicalWorkOptimizationObjective, but it produced longer trajectories with respect to the one selected.

6.5.2 STOMP planner

STOMP is the most promising motion planner among the available ones. It has good tunability and stable behavior in scenes dense with obstacles. Furthermore, it has the following parameters that can be tweaked for each planning group; for simplicity, we will divide them into classes depending on what they can influence. The classes are the following: **Optimization parameters**, **Noise Generator parameters**, **Cost Function parameters**, and **Update Filter parameters**.

Optimization parameters

The optimization parameters are the constants responsible for effectively changing the optimizer's behavior. They regard, principally, how the optimizer tries to deal with the cost function. The configuration file allows to tune the descriptors explained in Table 6.9. Before we describe which values we selected for this class

Parameter	Description
num_timesteps	The number of timesteps the optimizer can take to find a solution before terminating.
num_iterations	Number of iterations that the planner can take to find a good solution while optimization.
num_iterations_after_valid	Maximum iterations to be performed after a valid path has been found.
num_rollout	The number of noisy trajectories.
max_rollouts	The combined number of new and old rollouts during each iteration should not exceed this value.
initialization_method	This is the initialization method chosen to select how to initialize the trajectory
control_cost_weight	This is the percentage of the trajectory accelerations cost to be applied in the total cost calculation.

Table 6.9: Optimization parameters characterizing the STOMP algorithm

of parameters, we have to specify how many initialization methods there are. There are four methods: **Linear interpolation**, **Cubic Polynomial**, **Minimum control cost**, and **Fill trajectory**. The last method is of particular interest for our application because it is the one involved when using a pre-processor for STOMP, while the others, in our application, showed similar behavior and performances, so we decided to use **Linear interpolation**.

In our specific case, considering STOMP as a single planner and not as a post-processor, we can resume the values assigned for each parameter in Table 6.10:

Parameter	Value
num_timesteps	60
num_iterations	100
num_iterations_after_valid	0
num_rollout	30
max_rollouts	30
initialization_method	1 (Linear Interpolation)
control_cost_weight	0

Table 6.10: Optimization parameter values chosen for characterizing the STOMP algorithm

Noise Generator parameters

This subset of parameters helps to define the noisy trajectory the algorithm can explore to find a collision-free path. In particular, it is composed of:

- **class**: by this parameter, we can set different classes of noise generation:
  - NormalDistributionSampling
  - GoalGuidedMultivariateGaussian
- **stddev**: this is an array containing the amplitude of noise that can be applied at each joint. In the case of the Locobot, the arm is of dimension 1x6. Larger values in this class correspond to larger motion of the joints

In Table 6.11, we report the values assigned in our configuration file corresponding to these two constants:

Parameter	Value
class	stomp_moveit/NormalDistributionSampling
stddev	[0.05, 0.8, 1, 0.8, 0.4, 0.4]

Table 6.11: Noise generator parameters values chosen for characterizing the STOMP algorithm

Cost Function parameters

Depending on the cost function selected, we can have different parameters that can be tweaked. The cost functions available are the following: **CollisionCheck**, **ObstacleDistanceGradient**, and **ToolGoalPose**. The first one is the cost function that in this paragraph will be explored since the objective of our motion planning algorithm is to prevent colliding with obstacles. In Table 6.12, we report the description of the parameters to be tuned, while Table 6.13 shows the values assigned for this plugin.

Parameter	Description
collision_penalty	This is the value assigned to a collision state
cost_weight	This is the weight of the cost function
kernel_window_percentage	The multiplicative factor used to compute the **window_size** for doing kernel smoothing
longest_valid_joint_move	This parameter indicates how far can a joint move in between consecutive trajectory points

Table 6.12: Cost Function parameters for characterizing STOMP algorithm

Parameter	Value
class	stomp_moveit/CollisionCheck
collision_penalty	1
cost_weight	1
kernel_window_percentage	0.2
longest_valid_joint_move	0.05

Table 6.13: Cost Function parameters values selected for characterizing STOMP algorithm

Update Filter parameters

The algorithm, as explored in the state of the art, is based on updating the trajectory computed to find a better one in terms of smoothness and path length; for this reason, we can also set some parameters defining our update filter such as the ones reported in Table 6.14.

The possible classes available are: **PolynomialSmoother** and **Constrained-CartesianGoal**. In Table 6.15, we resume our configuration for what concerns the update filter:

Parameter	Description
class	Update filter function used for improving the trajectory
poly_order	This is the order of the polynomial function used for smoothing trajectories

Table 6.14: Update Filter parameters selected for characterizing STOMP algorithm

Parameter	Value
class	stomp_moveit/PolynomialSmoother
poly_order	6

Table 6.15: Update Filter parameters values selected for characterizing STOMP algorithm

6.5.3 CHOMP planner

CHOMP is the planning algorithm that performs the worst among the three analyzed. It is highly tunable in its parameters but is also subject to problems related to local minima and has many difficulties in finding a feasible path in narrow ambient with many obstacles. The parameters characterizing the algorithm are described in Table 6.16.
The last parameter, **trajectory_initialization_method**, is important when using a pre-processor for the CHOMP algorithm. It is possible to select between the following methods; the last one is used when implementing two algorithms simultaneously.

- Quintic-spline
- Linear
- Cubic
- FillTrajectory

Parameter	Description
planning_time_limit	This value represents the maximum time, in seconds, that the planner can take to find a solution
max_iterations	This represents the maximum number of executions that the planner can take to find a good solution
max_iterations_after_collision_free	Maximum number of iterations to be performed after having found a collision-free path to improve the path itself
smoothness_cost_weight	This parameters sets the weight of the cost function that CHOMP is actually optimizing
obstacle_cost_weight	This variable sets the weight of avoiding the obstacles in the final cost function (e.g. 0 means obstacles ignored)
learning_rate	Learning rate to find the global/local minima
smoothness_cost_jerk	Weight of jerk in being minimized in the cost function
smoothness_cost_acceleration	Weight of acceleration in being minimized in the cost function
smoothness_cost_velocity	Weight of velocity in being minimized in the cost function
ridge_factor	This represents the noise added to the total cost function matrix in order to avoid obstacles, doing this could worse the smoothness
use_pseudo_inverse	Enable pseudo inverse calculations or not
pseudo_inverse_ridge_factor	If pseudo inverse is enabled, set its ridge factor
joint_update_limit	Set the update limit for robot joints
collision_clearance	Minimum distance between the arm and the obstacles to be maintained
collision_threshold	This parameter represents the collision threshold cost that needs to be maintained to avoid collisions
use_stochastic_descent	If set to true (1) we do not use all the points of the trajectory to find a solution but only one. With this parameter on the convergency is guaranteed but it may take more time to find a solution
enable_failure_recovery	If set to true CHOMP, when can not find a solution, tries, by tweaking some parameters, to find a feasible path
max_recovery_attempts	Number of maximum attempts that CHOMP can do to find a good path
trajectory_initializaiton_method	This parameter sets the interpolation method that CHOMP will use when trying to reach the goal state.

Table 6.16: CHOMP parameters used for tuning the behavior of the motion planning algorithm

Planner settings

For our particular application, we selected the values reported in Table 6.17 for the parameters previously described. According to our experience derived by performing some simple tests in actuating poses in an environment where we have to pick from shelves, we will leave CHOMP as a possibility for performing the motion. However, using it along with RRT* as a pre-processor is preferable.

Parameter	Value
planning_time_limit	10 sec
max_iterations	200
max_iterations_after_collision_free	5
smoothness_cost_weight	0.1
obstacle_cost_weight	1
learning_rate	0.01
smoothness_cost_jerk	0
smoothness_cost_acceleration	1
smoothness_cost_velocity	0
ridge_factor	0.01
use_pseudo_inverse	false
pseudo_inverse_ridge_factor	0.0001
joint_update_limit	0.1
collision_clearance	0.2
collision_threshold	0.07
use_stochastic_descent	true
enable_failure_recovery	true
max_recovery_attempts	5
trajectory_initializaiton_method	Linear

Table 6.17: CHOMP parameter values for our application

6.5.4 Planning adapters

Using multiple planning algorithms is a possibility given by MoveIt [45] to enforce the computation of a trajectory. In particular, we can use, for example, RRT* as a pre-processor for STOMP/CHOMP. The result of this approach is to produce robust paths. In our robot, we provided different possibilities:

- Use RRT* as pre-processor for STOMP
- Use RRT* as pre-processor for CHOMP

This results in longer computational time required but better accuracy and smoother paths.

Chapter 7

Software architecture of the mobile base

7.0.1 Our SLAM Algorithm: RTAB-Map

We tested two graph-based SLAM algorithms on the Locobot platform: Slam-Toolbox [62] and RTAB-Map (Real Time Appearance Based Mapping) [63]. Slam Toolbox's contribution to the SLAM research space is the ability to effectively map large spaces as large as 9000 m^2 in real time while keeping an accurate estimate of the robot's pose by using only Lidar to map a 2D environment of the surrounding space. RTAB-Map uses a variety of sensors from RGB-D cameras to 2D and 3D Lidar both separately and combined by synchronizing their information using time stamp information. It uses a bag of words approach for loop closure detection and memory management for long term SLAM to prune the graph size so as to be able to keep Graph Optimization and Loop Closure detection within real-time speeds. While SlamToolbox was easy to use and integrate in the Locobot navigation stack the constrain on sensor usage (only 2D Lidar) quickly became an issue since the office space contains many reflective and glass walls. This meant that during SLAM there were missed measurements and issues with localization. So RTAB-Map seemed a natural choice due to its availability and high flexibility. RTAB-Map has several outputs, most important for the navigation process is the 2D Occupancy Grid [64]. It also gives the option of generating an OctoMap when using 3D Lidar or RGB-D (3D occupancy grid) and an optional dense Point Cloud.

7.1 Path Planners Plugin Choice

In the following section, we explain the choice made for path planners and the ROS plugins that we will use to implement them on the mobile base.

7.1.1 Global Planning Plugin Choice

The Locobot has the standard navFn package[65] already available as a global planner package, however, it only allows for Djikstra's algorithm for path calculation. We chose the **Global Planner** package[66], another option from the ROS Noetic navigation stack. This package gives the option of using A* and has much more flexible settings. There is a tradeoff between the two path planners: Djikstra is better for smoother and shorter paths while A* leads to faster computing time. A* would be far better for a dynamic environment, allowing the Locobot to recompute a global path when it encounters a moving obstacle large enough that does not permit the local path planner to keep to the global plan within the set constraints. However, A* paths tend to be angular and non-natural, taking the shape of stair-like patterns (this is where most of the distance losses with respect to Djikstra are). A good alternative, proposed in [67], is the use of A* global path planner with a local planner that allows for path smoothing which shortens the resulting path (TEB or EBAND).

7.1.2 Local Planning Plugin Choice

The paper [68] presents a comparative study of navigation local planners, in particular:

- Dynamic Window Approach (DWA)
- Elastic Band
- Timed Elastic Band (TEB)

It points out previous studies on these algorithms such as [67] where DWA was shown to have issues with dynamic obstacle avoidance. Of course, there are computing power requirements to be mindful of, and DWA is the least computationally expensive while TEB is heavier on resources. TEB also was found to be faster and produced smoother paths when going around obstacles For our use case the most important aspect is accuracy since the starting pose of the mobile base is of utmost importance for the arm to successfully pick up the required object. We chose TEB [69] since it compliments the global path planner A* well, by smoothing the global plans stair-like pattern.

7.2 Path Planning Setup

The move base package comes preinstalled on the Locobot and by modifying its parameters in the **move_base_params.yaml** configuration file we are able to choose the local and global planner plugins. Most importantly the *base_local_planner* and *base_global_planner* are set from the default values to the TEB local planner plugin 7.2.2 and the Global Planner plugin 7.2.1. The other settings handle the frequency at which the **move_base** package sends commands, recalculates the global path and oscillates in recovery mode.

Move Base settings	Value
base_global_planner	global_planner/GlobalPlanner
controller_frequency	10
controller_patience	20
planner_frequency	1
planner_patience	15
max_planning_retries	2
oscillation_timeout	10
oscillation_distance	0.2
base_local_planner	"teb_local_planner/TebLocalPlannerROS"
base_global_planner	"global_planner/GlobalPlanner"

Table 7.1: Global planner plugin parameter settings

7.2.1 Global Planner Settings

The global planner [66] plugin is used by modifying the **global_planner_params.yaml**.

Global Path Planner Parameter	Value
use_dijkstra	false
allow_unknown	false
use_grid_path	true
orientation_mode	2

Table 7.2: Global planner plugin parameter settings

We set **use_dijkstra** to false to use A* as a global planning algorithm. The other settings are default except for the orientation mode. This setting sets the intermediate poses for the global plan to face a forward direction. This way we can help the local planner avoid using backwards velocities.

7.2.2 Local Planner Settings

The TEB local planner package [69] was built for differential drive robots that could move in reverse. While the Locobot has this capability we wanted to discourage it since we only have an RGB-D detector in front and could only detect obstacles behind it with the laser. Since it is only 2D LIDAR it might hit obstacles that are lower than its plane of detection. The settings for the local planner are shown in Table 7.3.

Local Path Planner Parameter	Value
acc_lim_x	0.3
acc_lim_y	0
acc_lim_theta	3.2
max_vel_x	0.3
min_vel_x	0.1
max_vel_theta	1.0
min_in_place_vel_theta	0.4
escape_vel	-0.1
escape_reset_dist	0.05
include_dynamic_obstacles	true
holonomic_robot	false
vx_samples	3
meter_scoring	true
path_distance_bias	0.75
goal_distance_bias	1.0
occdist_scale	0.01
dwa	true
min_obstacle_dist	0.3
inflation_dist	0.47
dynamic_obstacle_inflation_dist	0.47
weight_kinematics_forward_drive	1000
allow_init_with_backwards_motion	false
max_vel_x_backwards	0.01

Table 7.3: Local planner plugin parameter settings

Acceleration and velocity were kept unchanged. Prohibiting backward motion is impossible since this planner only takes in soft constraints and calculates the optimal path for every weighted objective and penalty. This means that even if the penalty for backwards motion is set to the maximum possible value there are situations where the robot might move in reverse. Setting the weights to such high values can also negatively impact the optimization function slowing down the convergence speed to the optimal solution.

So to discourage backwards motion we set:

- **weight_kinematics_forward_drive** :1000

  This weight pushes the choice towards forward directions (positive x velocities). A small value will allow backward driving and setting it to 1000 almost forces it to always choose forward drive.

- **allow_init_with_backwards_motion** :false

  Setting this to false means that the robot cannot start moving from a static pose with a backwards velocity, this discourages car-like behaviours for repositioning.

- **max_vel_x_backwards** :0.01

  This setting limits the speed of the robot to a slow pace. This further discourages backwards motion.

7.3 Human detection and path avoidance

As stated previously in section 1.4.3 human detection is an important part of this thesis work, both for safety requirements and path planning efficacy. The Locobot has an Intel NUC mini desktop with no dedicated GPU. This lack of computing power meant that we had to choose algorithms that could be reliable while needing as few resources as possible. For human detection, though highly effective, machine learning algorithms such as neural networks and image detection CNNs were not possible due to their resource usage. For human detection, we used a random forest classifier for the LIDAR sensor coupled with a HOG-based point cloud library detector [70] for the RGB-D sensor. We decided to use a reactive approach to path avoidance together with a physics-based method for path prediction.

The chosen packages are

- **People package**[71]
- **Social Navigation Layer** [72]
- **Spencer people tracking**[73]

7.3.1 People package

The people package is a software stack that contains various algorithms used for people tracking and detection. At first, we tested the leg detector package [74] that is part of this stack but it resulted in a high amount of false positives. Also, its inefficient implementation based on python meant that running the leg detector negatively affected the performance of other ROS nodes.

This package was useful since it defines a **people_msg** ROS message. This is used by the **social layer** package [72] to read the detected people published by the detector nodes. The **people_msg** defines 5 types of messages :

- **people_msgs/PositionMeasurement**: This is the standard message used by the package. It contains a **geometry_msgs/Point** message giving the position of the person and a reliability value for the detection.
- **people_msgs/PositionMeasurementArray** : This is an array of **people_msgs/PositionMeasurement**.
- **people_msgs/Person**: This message contains more detailed information about the detected persons. It has a position **geometry_msgs/Point** message, a velocity **geometry_msgs/Point** message and a reliability value for the detection.
- **people_msgs/PersonStamped**: This message contains a header together with a **people_msgs/Person** message.
- **people_msgs/People**: This is a message containing an array of **people_msgs/Person** messages.

These messages are published on the **People** topic. No settings were changed for this package since we used it only for its message functionalities.

7.3.2 Social Navigation Layer

This package adds two custom layers to the [54] package. These layers use information from the **People** topic to add an inflated cost to the obstacles recognized as humans. The two layers are:

- **social_navigation_layers::ProxemicLayer**: This layer uses the theory of proxemics [32] to add a gaussian cost to the detected humans, this cost forces the path planner that reads it to take this cost into account during the path calculation.
- **social_navigation_layers::PassingLayer**: This layer adds a similar cost to that of the proxemic layer, but adds a gaussian cost area to a desired side of the detected human. This way if the robot has to pass by the human it is forced to plan a path on the other side.

The **ProxemicLayer** was added to the global and local path planner. This is done by adding it to the plugin list in the global and local path planner configuration files. The gaussian cost that is added is deformed by the velocity of the person.

When stationary the gaussian is a circle shape with a cost that decreases along the radius. When a person is moving then the velocity deforms the gaussian function towards the direction of movement, resulting in an oval shape that envelops the space in front of the person. This is done to take into account the temporal aspects of a moving obstacle, so as to discourage path plans that use the cells in front of it. The faster the speed the more deformed the gaussian. This is shown in Figure 8.26.

The settings used for the plugin are reported in Table 7.4.

Social Layer Parameters	Default Value
enabled	True
cutoff	10.0
amplitude	77.0
covariance	0.25
factor	5.0
keep_time	0.75

Table 7.4: Social Layer plugin parameter settings

The most important settings are **amplitude** and **factor**.

- **amplitude** scales the size of the gaussian up or down. If this value is too small then the inflation radius will overtake the proxemic inflation and this plugin will have no effect, if it is too large then the path planners will have trouble finding an optimal path.

- **factor** is a multiplicative factor that is used to weigh by how much to deform the gaussian function. The larger the factor then the larger the deformation. A value that is too small will result in a circular Gaussian even when a human is moving at fast speeds, while a value that is too large will result in overcorrection. The robot will correct its path a far distance ahead of a slow-moving person as if it was running.

Social Navigation Layer settings

The default values were too high for our environment. An **amplitude** value of 77 meant that the gaussian generated occupied most of the corridor available for testing, making the robot give more than 2.5m of space to the human. This caused the robot to always stop when encountering a person since it could not path plan around the cost. We found that a value of 40 was more reasonable and to increase the effect of speed on the shape of the gaussian we doubled the value of **factor** to 10.

Spencer People detection Parameters	Value
amplitude	40
factor	10

Table 7.5: Spencer Launch file settings

7.3.3 Spencer People Tracking Package

The package we used for human detection and tracking comes from the work presented in [75] and [76] by Linder, T et al. It reviews recent work in human tracking and explores the use of a multi-modal online LIDAR/RGB-D people tracking framework. Some of the detectors available in this framework can be leveraged by the Locobot since they make use of low computational effort algorithms like Nearest Neighbours and Random Forest classifiers. It uses also a groundHOG classifier that, however, we could not use since it requires an Nvidia GPU with Cuda library capabilities. The detection and tracking models are compatible with our sensors but were both trained and tested on different sensors with different configurations. For example, the random forest model for the laser leg detector was trained with front and rear 2D LIDAR SICK LMS500 at 70 and 75 cm height with a more accurate angle resolution. The RGB-D data was taken from an Asus Xtion Pro Live at a height of 1.6m whereas the Locobot has a Realsense stereo camera at around 60 cm height. By tweaking the parameters of the following files we were able to achieve human detection and tracking.

Tracking_on_robot.launch file settings

The whole package is launched using the **tracking_on_robot.launch** file. The following settings 7.6 were modified in order to adapt the package to the LocoBot.

Spencer People detection Parameters	Value
height_above_ground	0.54
use_upper_body_detector	False
use_pcl_detector	False
use_hog_detector	True
base_footprint_frame_id	locobot/base_footprint

Table 7.6: Spencer Launch file settings

Laser detector settings

The laser detector uses a random tree classifier model trained on a 2D LIDAR SICK LMS500 at a different height from our LIDAR. Nevertheless, by changing the settings as shown in 7.7 we were able to achieve stable human detection.

Laser detector parameters	Value
model_prefix	lms500_0.25deg_height70cm_fr_rathausgasse"
decision_threshold	0.25
min_avg_distance_from_sensor	0
laser_max_distance	4
laser	locobot/scan
detector_type	random_forest

Table 7.7: Laser detector launch file settings

The model is easily modifiable by changing the **model_prefix** parameter. The **decision_threshold** and **laser_max_distance** parameters were the most important settings to change in order to achieve detection.

Point Cloud Library detector settings

The Point Cloud Library detector is modified from an available classifier from the Point Cloud Library [70]. It uses a HOG support vector machine to identify full-body images of humans. We chose this detector since the other detector in the framework, an upper-body classifier, needs a dedicated GPU to work.

Point Cloud Library detector parameters	Value
input_topic	/locobot/rtabmap/depth/color/obstacles
camera_info_topic	/locobot/camera/color/camera_info
base_link_frame	locobot/base_footprint
detection_frame	pcl_people_detector_front_link
ground_coeffs	0 0 -1 0

Table 7.8: Point Cloud Library detector launch file settings

We modified the input topics for this package as shown in 7.8 to allow it to read the depth information from the sensors.

People Tracker settings

The people tracker settings in **freiburg_people_tracking.launch** are kept as is.

7.3.4 Tracked People Translator Node

Having set up the previously described packages we needed a way for the detections to be seen by the social navigation layers. The Spencer People Tracking package uses its own ROS messages to publish detections and tracking information and publishes them to the **/spencer/perception/tracked_persons** topic. The Social Navigation Layers package expects detections to be published to the **/People**

topic. To merge these two packages we implemented a C++ ROS translator note named **tracked_people_translator**. This node is launched together with the detectors and subscribes to **/spencer/perception/tracked_persons**, reads the messages and translates them to **people_msgs/People**.

7.4 Software Architecture for the Base

7.4.1 Base Control node structure

As stated in the overview of our software architecture we use a decoupled algorithm. The entire task, from the object pickup request to the placement operation, is planned by the communication manager that commands the manipulator and the base individually through dedicated channels.

As we can see from Figure 7.1 the node structure is composed by:

- The **base controller** node is responsible for performing the navigation task. It receives the goal pose for the mobile base and calculates and executes a navigation plan.
- The **communication** node controls the entire execution logic. It sends the goal pose and receives information on the base status to send commands to the manipulator when the base successfully reaches the goal location.

7.4.2 Description of the mobile base goal topic

The mobile base goal topic is used by the communication manager to send goals to the **base_controller**. The messages are **geometry_msgs::PoseStamped** and contain the desired position and orientation with respect to the frame of reference that is defined in their header. The**base_controller** node uses this information as is without any modifications.

7.4.3 Description of the base status topic

The **base_controller** node publishes its status to the **base_status** topic. The **communication_manager** node subscribes to this topic. The messages are **std_msgs::Int64** and are simple macros defined in a header file.

- **BASE TO GOAL**: this status lets the communication manager know that the base is moving to the goal and has yet to conclude its movement.
- **BASE GOAL OK**: this status is temporarily sent to the **base_status** to signal the successful completion of the movement.

- **BASE GOAL FAIL**: this status is necessary for the re-planning steps described in section 5.4.2.
- **BASE IDLE**: while sending a goal to the base, the communication manager checks for this flag. If the base is idle then it can receive new goals if not the communication manager waits for it to become available.

7.4.4 Description of the no marker

This topic contains a simple boolean message, it is a redundancy in view of future software expansions. When sending a mobile goal message the **base_controller** can read this topic to know whether the movement is to reach an ARTag or towards a goal that is unrelated to the markers. The home and deposit goals are such examples.

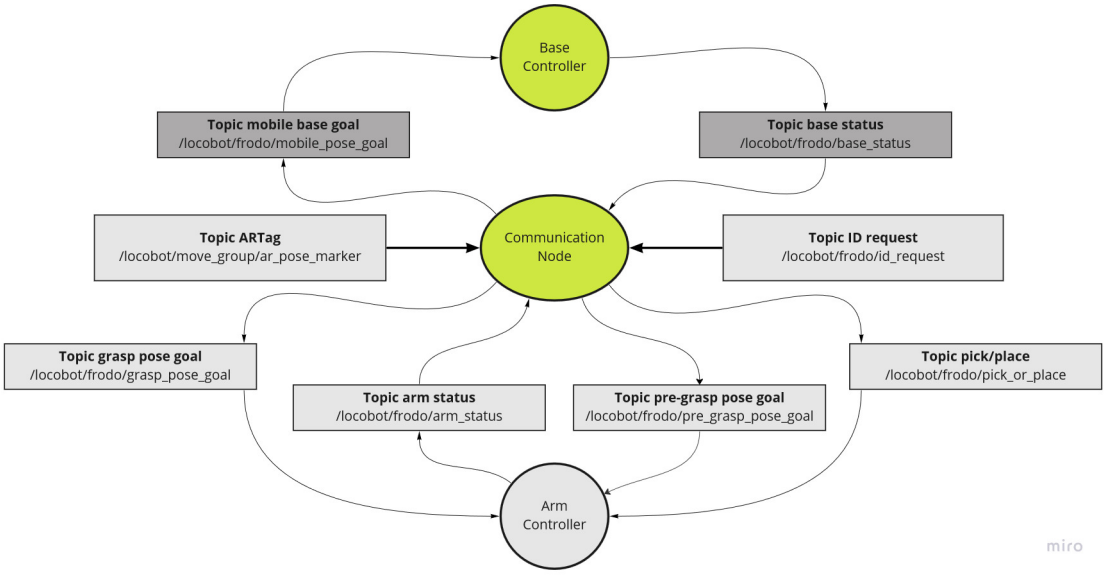


Figure 7.1: Structure of the software implemented for controlling the base

Chapter 8

Simulation and experimental results

After explaining our application's software architecture, we performed different tests to determine this structure's limits and advantages. The tests we performed can be split into two types depending on the environment where the robot is executed: the ones performed in simulation through Gazebo and the ones where the deployment is in our laboratory where we positioned some fake medicines with ARTags to be picked up and placed.

8.1 Simulation setup in Gazebo

The map implemented in Gazebo, which describes a possible warehouse environment, has some obstacles that the mobile base can avoid during path planning. The items to be manipulated are positioned on shelves, representing a possible situation we can face in reality. We can look at the world built in Gazebo in Figure 8.1.

8.2 Working of the robot in simulation

Figures 8.2 - 8.7 show the different working phases of our algorithm previously described performed to pick an object in a simulated environment. They appear in order of execution.

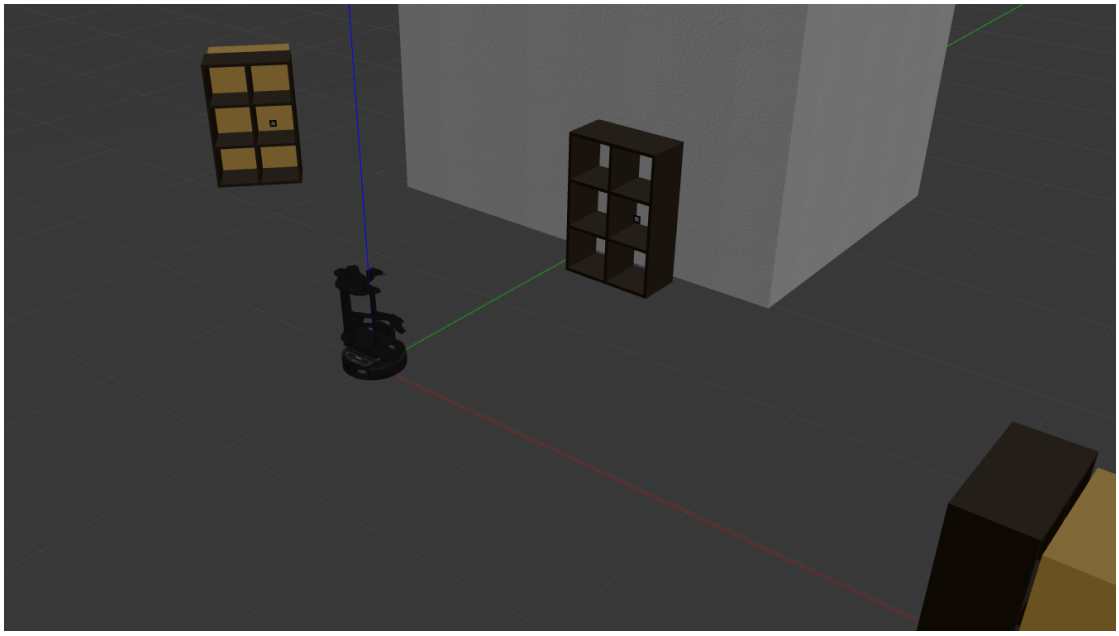


Figure 8.1: Map constructed in Gazebo to test the working of the robot in simulation.

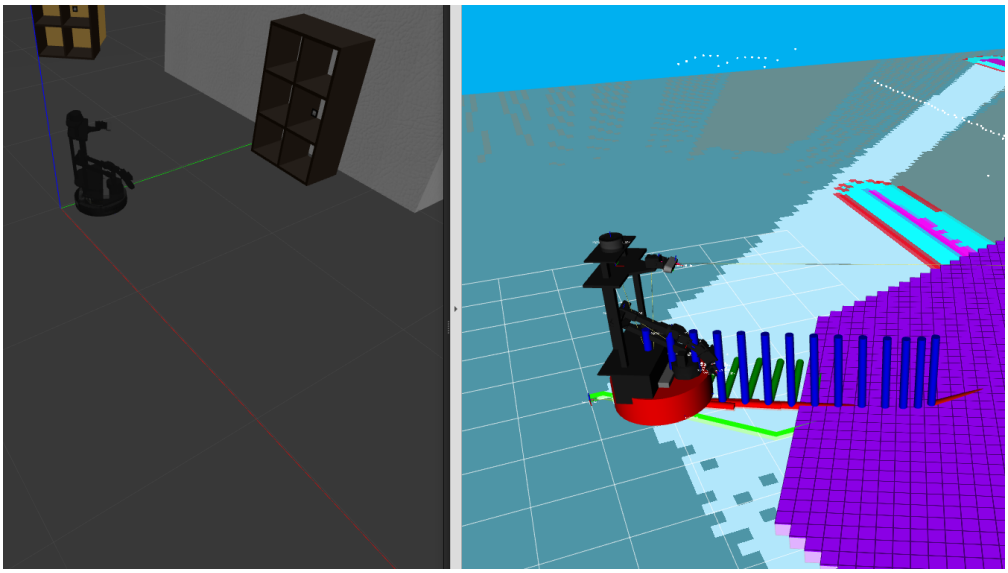


Figure 8.2: The robot received the command, so it is going to approach the medicine stored on the shelf, here only the mobile base is working.

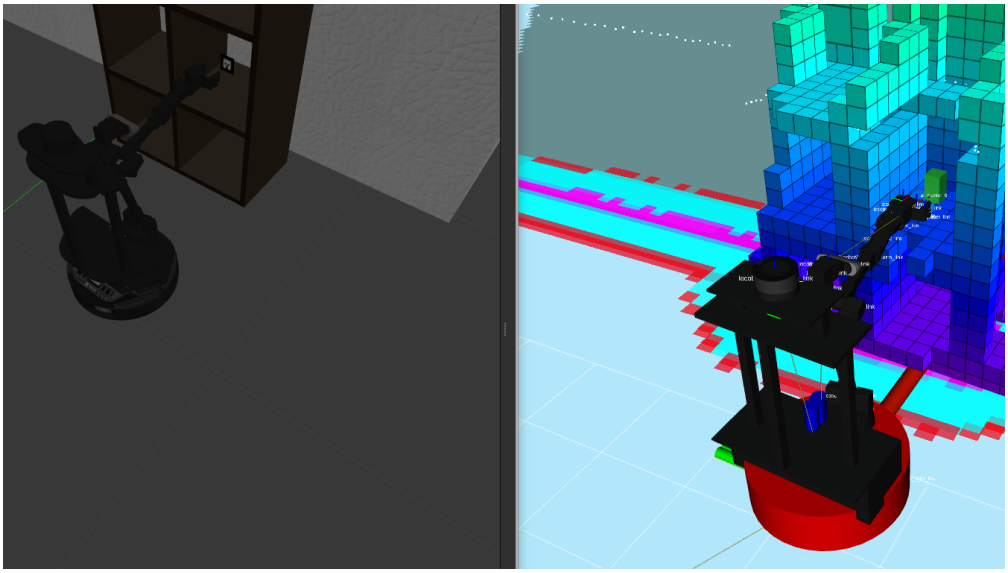


Figure 8.3: The base reached the desired pose so now the manipulator arrives at its **pre-grasp pose**. The item to be grasped is added to the planning scene as an object (**green box**).

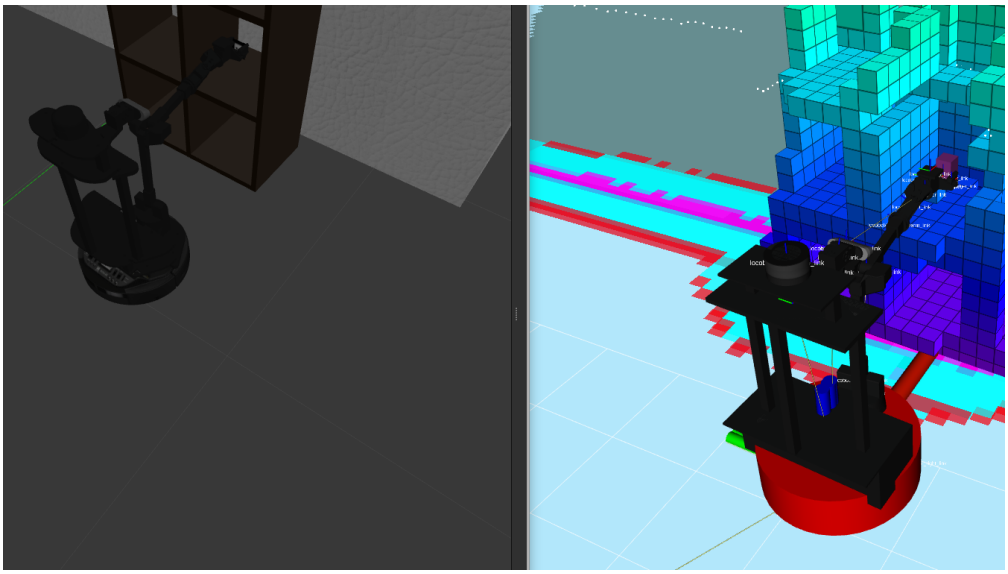


Figure 8.4: Now the manipulator is ready to pick the object, the fingers of the end effector are opened, and the box added previously in the planning scene is attached to the end effector link so that the robot is aware of its presence.

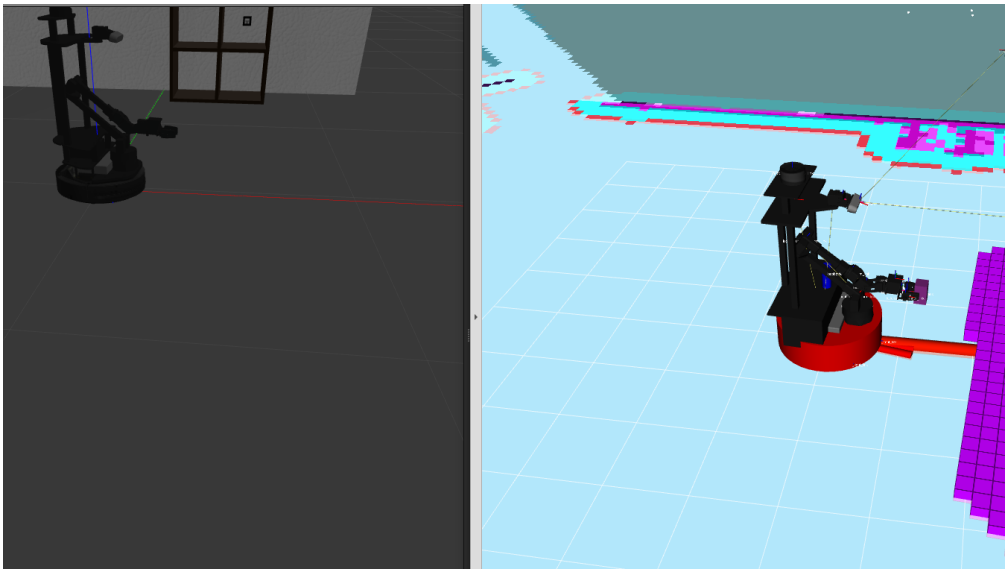


Figure 8.5: The object has been picked so now it is running the place pipeline. The base returned home.

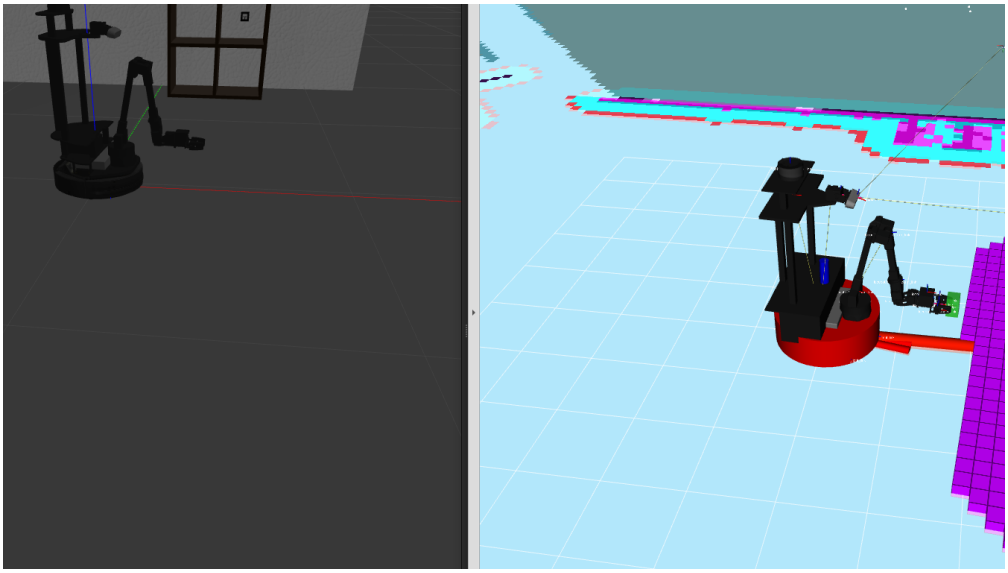


Figure 8.6: The manipulator reaches the **place pose**, then it opens the gripper and finally detaches the item (box returns green instead of purple)

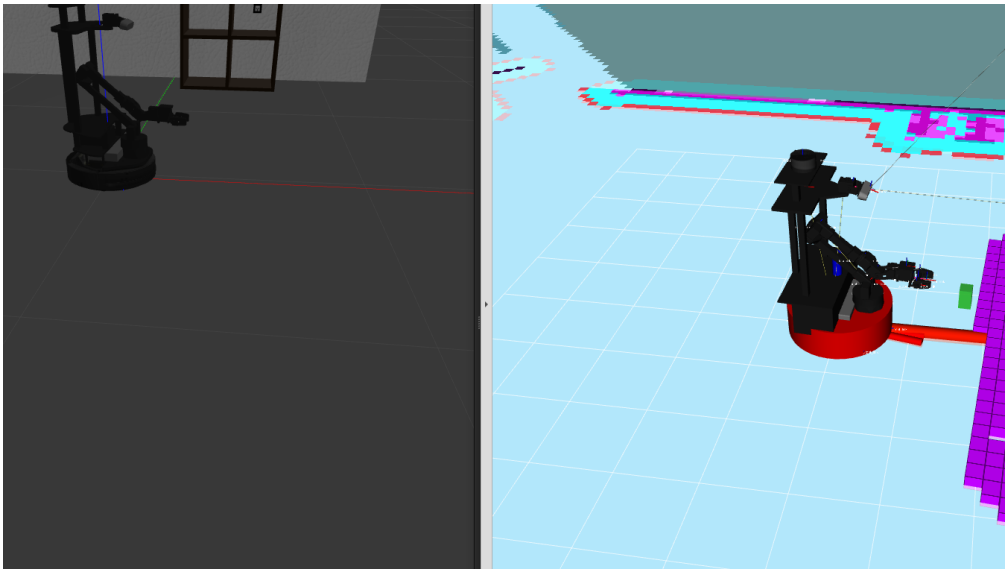


Figure 8.7: The manipulator finishes its operations, so it returns in]**home pose**. The robot is ready to take another command

8.3 Laboratory setup

The laboratory where we studied and tested our robot is smaller than a real warehouse. However, it could accurately represent the obstacles that a mobile robot can face, such as humans and other furniture. This has also been done because the mobile manipulator could reach a maximum of approximately one meter, so real shelves are unsuitable for our tests.
The tests are conducted in an environment that is composed of three items, two of them are depicted in Figure 8.8 while an overall view of how the three elements are positioned in the scene is visible in Figure 8.10:

- **Depot**, where the grasped medicines will be placed after completing the picking phase. This position is known as in a real warehouse, and it is defined with respect to the map constructed starting from the home position.

- **Home**, where the robot starts its research at each iteration and where it returns as soon as a place operation is finished.

- **Shelf**, where the items to be grasped are stored. To validate our algorithm we positioned the shelf in three different positions completely different from each other.

Figure 8.8: In the first figure (left), we can have a look at the robot in its **home** position, which is also the starting point of our research phase; instead, in the second figure (right), we can see the **depot** zone where the robot will deposit the grasped item.

Figure 8.9: In these figures, we can have a look at possible positions assumed by the shelf for validating the algorithm. These positions are marked with duct tape on the floor to assure the repeatability of the experiments.

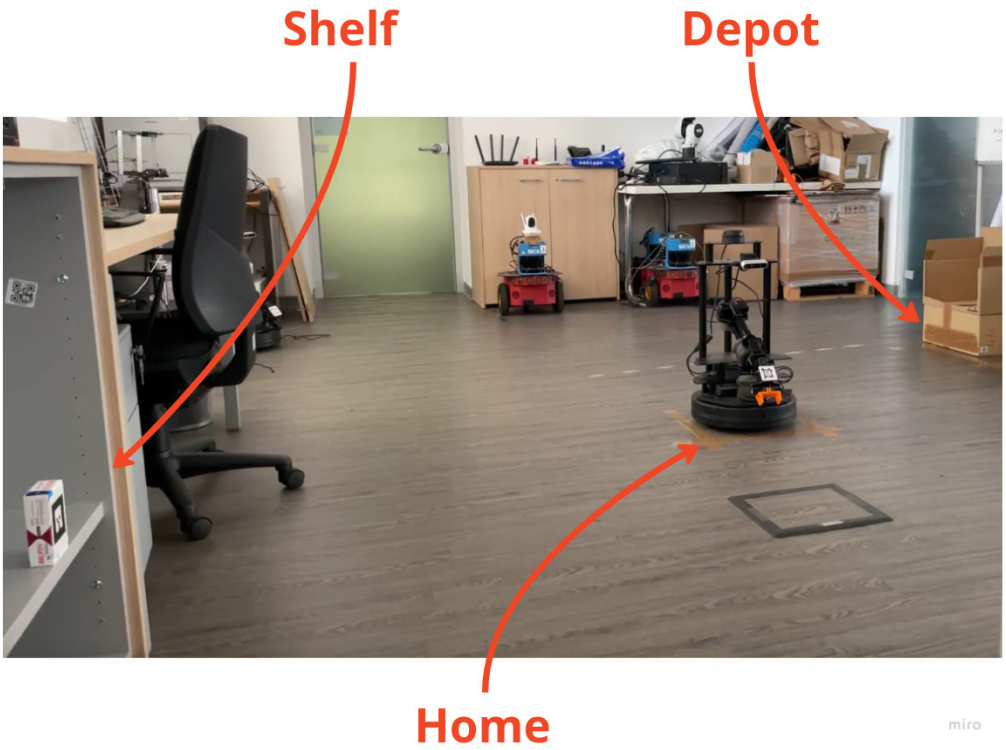


Figure 8.10: Here we show an overall view of the environment where we conducted our experiments. It comprises three objects: the home, the depot, and the shelf where the items are stored.

As we can see from Figure 8.9 we positioned on a shelf two boxes, each with a different ARTag id, so that we could test if the robot effectively picks the right one. Moreover, to test the algorithm more deeply, we also selected three positions and orientations that the shelf can assume. In particular, this test let us understand that the algorithm works finely and the pose estimation is done correctly and independently from the position of the markers in the space. The possible configurations are shown in Figure 8.9.

8.4 Working of the robot in a real environment

The simulated and the real robot perform almost the same actions to successfully carry a request from the user. Here we report the main steps for completing a pick and place action in order of execution.
The first step starts when the robot unit receives a request for an ID. In this case, we enter the search phase that ends when the requested marker is found. When the ARTag is found, the mobile base receives the command to move and reach the desired pose, as shown in Figure 8.11.

Figure 8.11: The robot received the command, so it is going to approach the medicine stored on the shelf, here, only the mobile base is working.

Once the base reaches the position in front of the shelf, the manipulator starts running while the base is idle. The arm controller receives the pre-grasp and the grasp pose goal, and the pick command. The routine starts with the update of the Octomap, and then, as we can see in Figure 8.12, the manipulator reaches the pre-grasp pose.

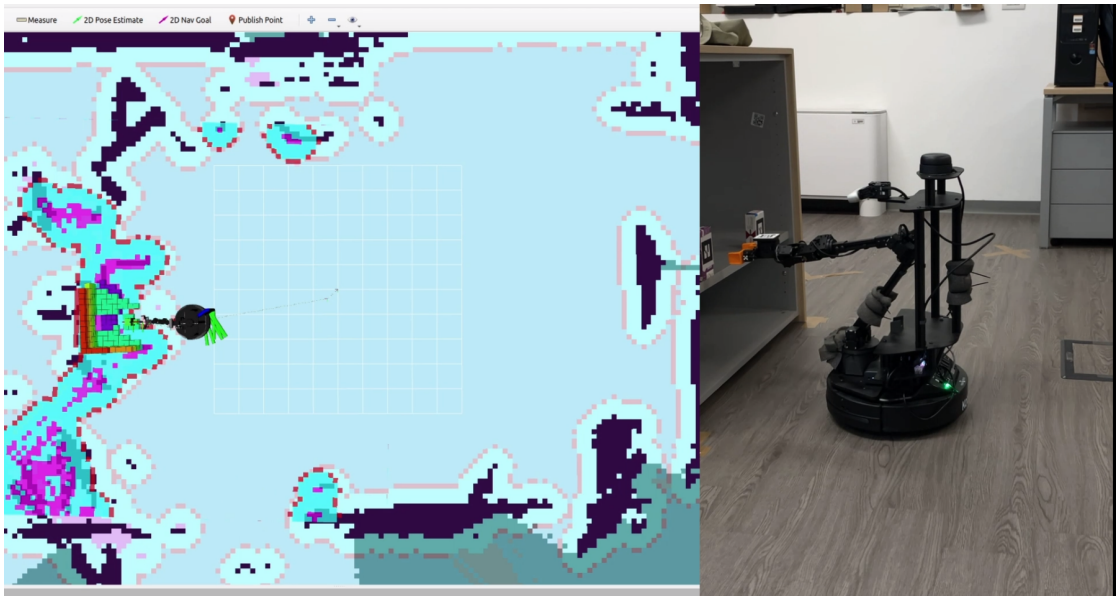


Figure 8.12: Manipulator reaches the pre-grasp pose after positioning in front of the shelf.

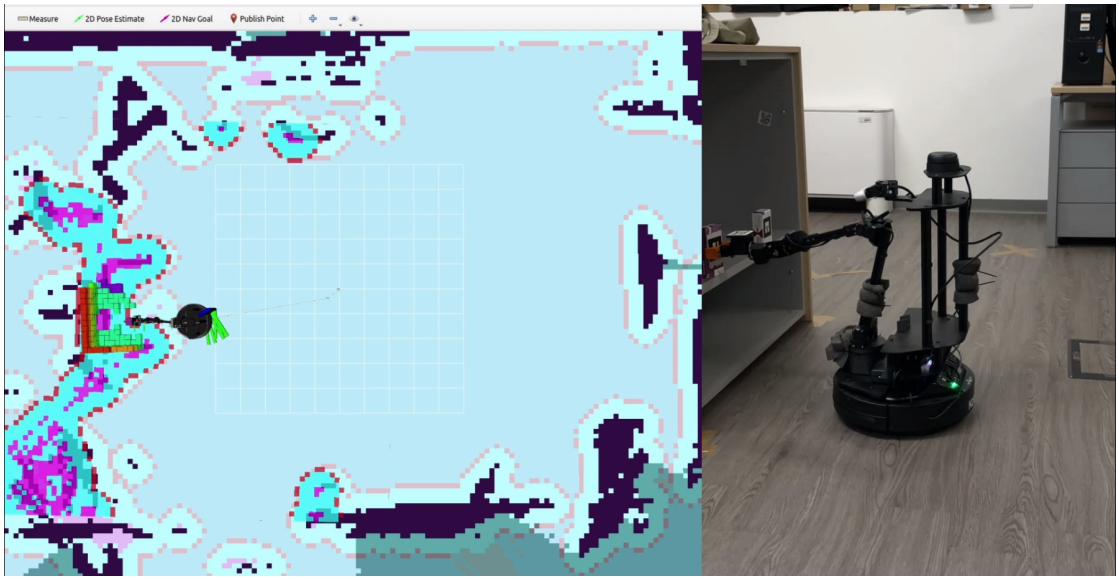


Figure 8.13: Now the manipulator is ready to pick the object, and the fingers of the end-effector are opened.

Having completed the previous step, the arm will run to reach the grasp pose. In this phase, the gripper has been opened so that we can approach the item positioned

on the shelf. Before being able to grasp, as described in the chapter about the software architecture of the manipulator, the planning scene is updated with the item to be picked, and then, finally, the arm can move toward the marker. Figure 8.13 shows the situation when these operations are completed. In this part of the manipulation, different tools are involved, such as the **planning_scene_monitor** and the **move_group** interface. Each of them plays an important role: the first one governs the obstacle present in the scene to be taken into account by the anthropomorphic arm, while, the second one, invokes the motion planning algorithm, RRT^*, that solves the motion planning problem.

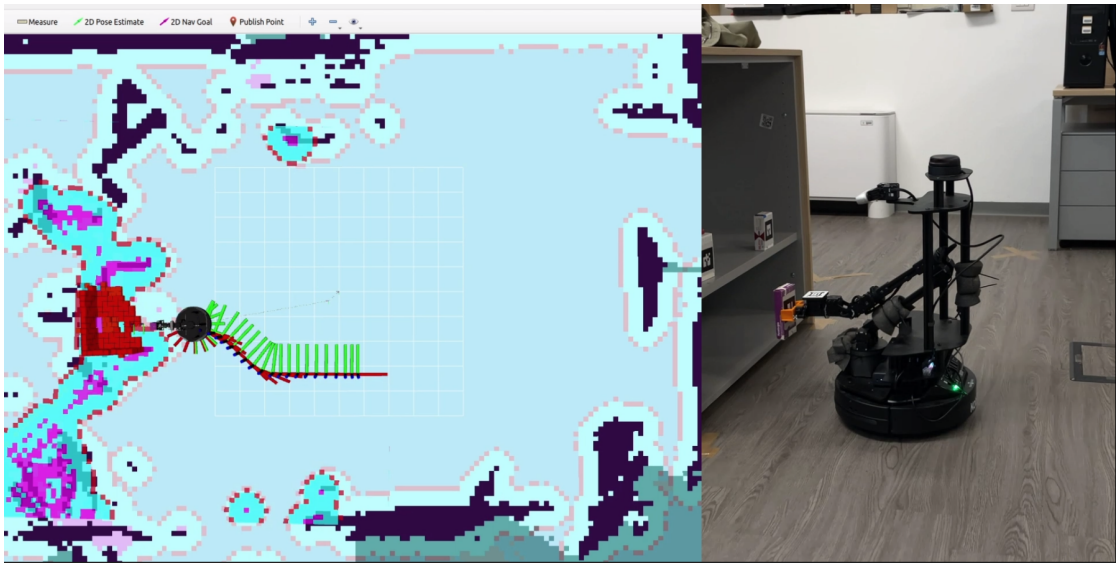


Figure 8.14: The object has been picked, so now it is running the place pipeline.

Once the grasping part is finished, the arm will return to idle status, as shown in Figure 8.14, and the mobile base will receive the pose of the **depot** as the goal pose.
The mobile base reached its final position to complete the placing routine. Now, as for the picking action, the manipulator will start its operations while the mobile base will remain in idle status. The actions to be performed in this stage will involve the arm for reaching the place position and then the gripper for releasing the item in the depot. The first phase is shown in Figure 8.15 while the second one is depicted in Figure 8.16.

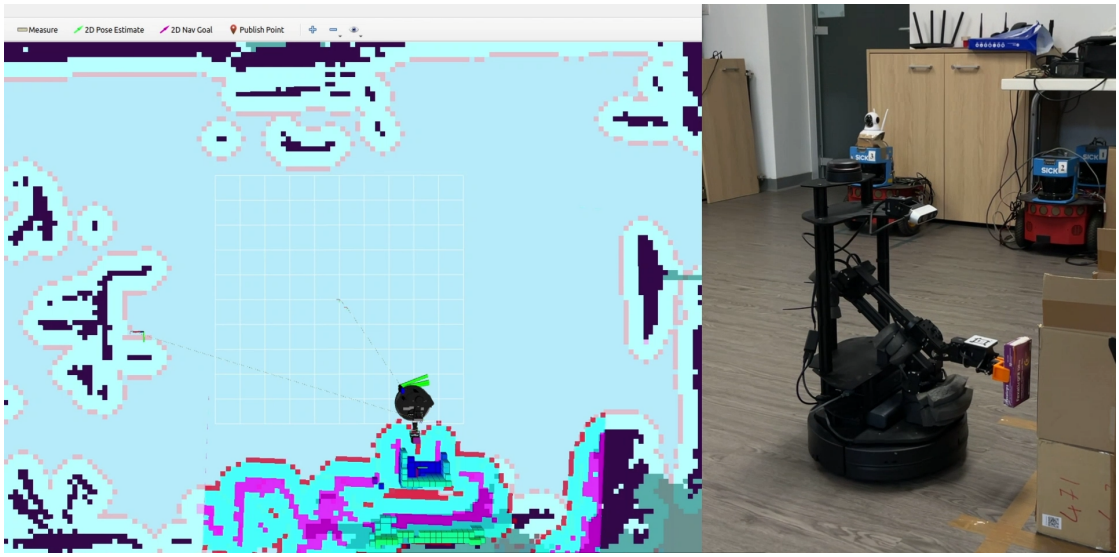


Figure 8.15: The mobile base reaches the **place pose**, now the arm should start running

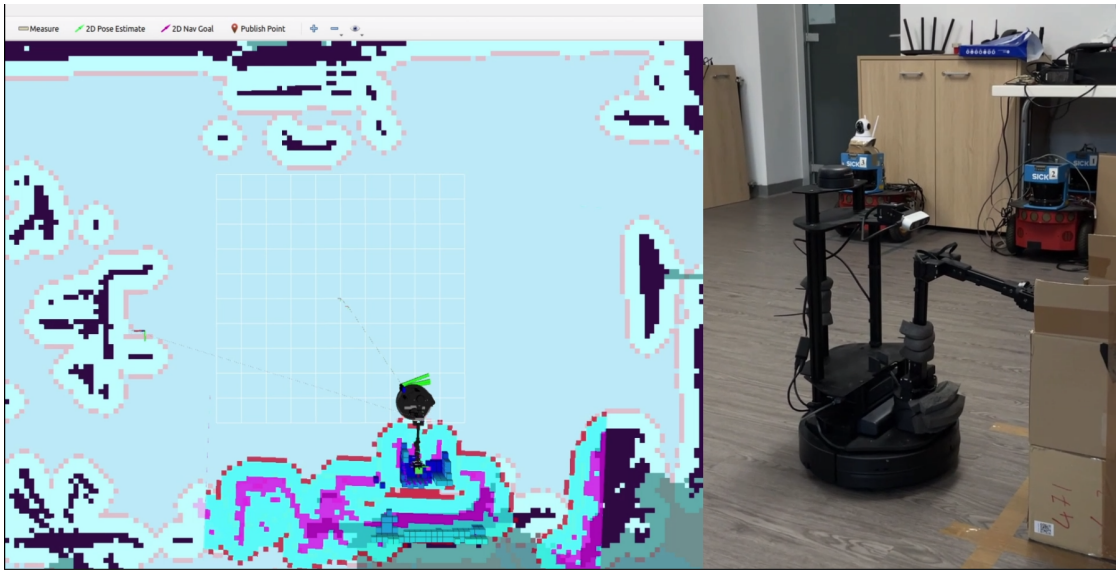


Figure 8.16: The manipulator reaches the **place pose**, then it opens the gripper and finally detaches the item (box returns green instead of purple)

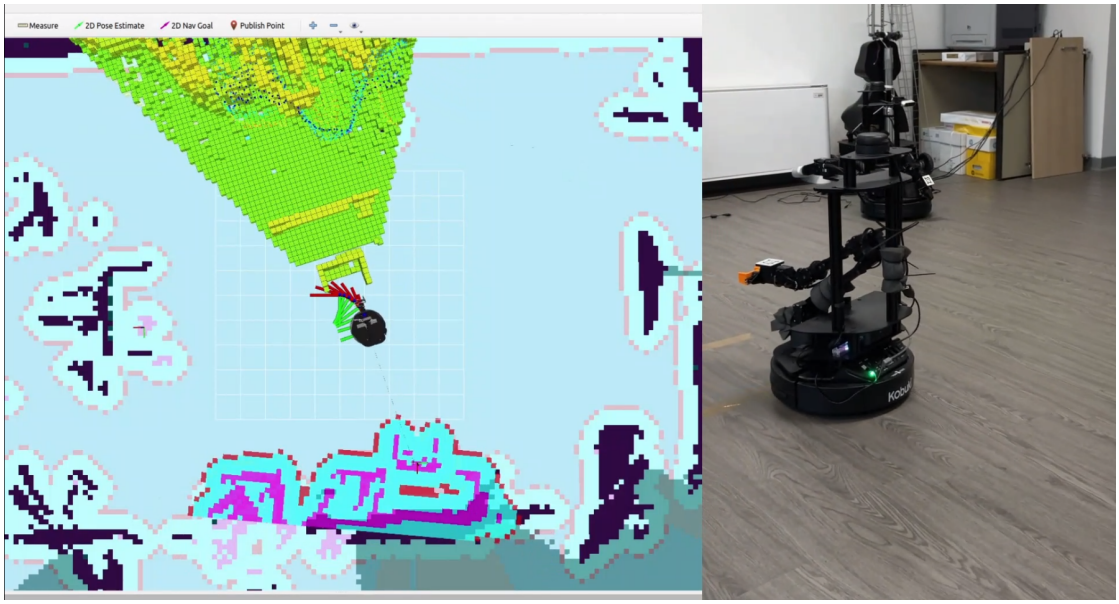


Figure 8.17: The manipulator finishes its operations, so it returns in **home pose**. The robot is ready to take another command

Finally, the mobile robot completed all the operations to be performed in order to pick and place the item requested by the user at the beginning of the test. Now the mobile robot will return to its **home** position to be ready for the subsequent request. The time necessary to carry on the whole process is about six minutes; it could be drastically reduced by improving some components and the timing of certain operations, as discussed in the final chapter. In Figure 8.17 the mobile manipulator is returning home after having completed its task.

8.5 Motion planner tests

The motion planners implemented in MoveIt currently working finely are STOMP, CHOMP, and the algorithms of OMPL. The algorithm can be selected before the robot starts giving a command as an argument of the launch file. Figure 8.20 shows what happens in the planning scene when the arm approaches a shelf to pick an item. The obstacles in the scene are represented as boxes in RViz.

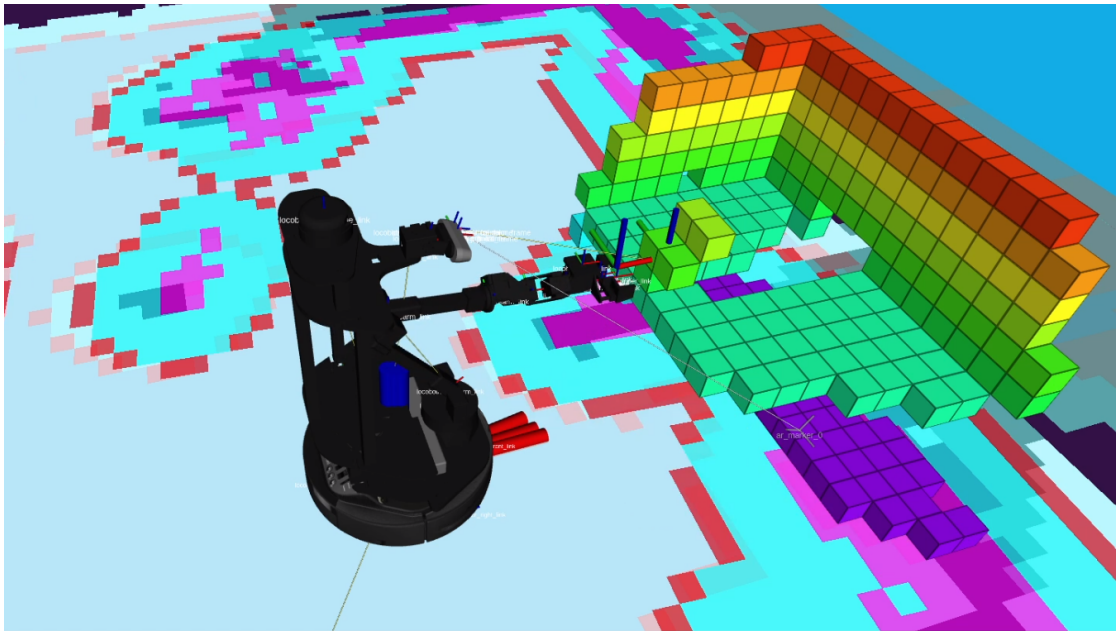


Figure 8.18: Here is an image of the robot performing a pick routine. The software is Gazebo for simulation and RViz for visualization.

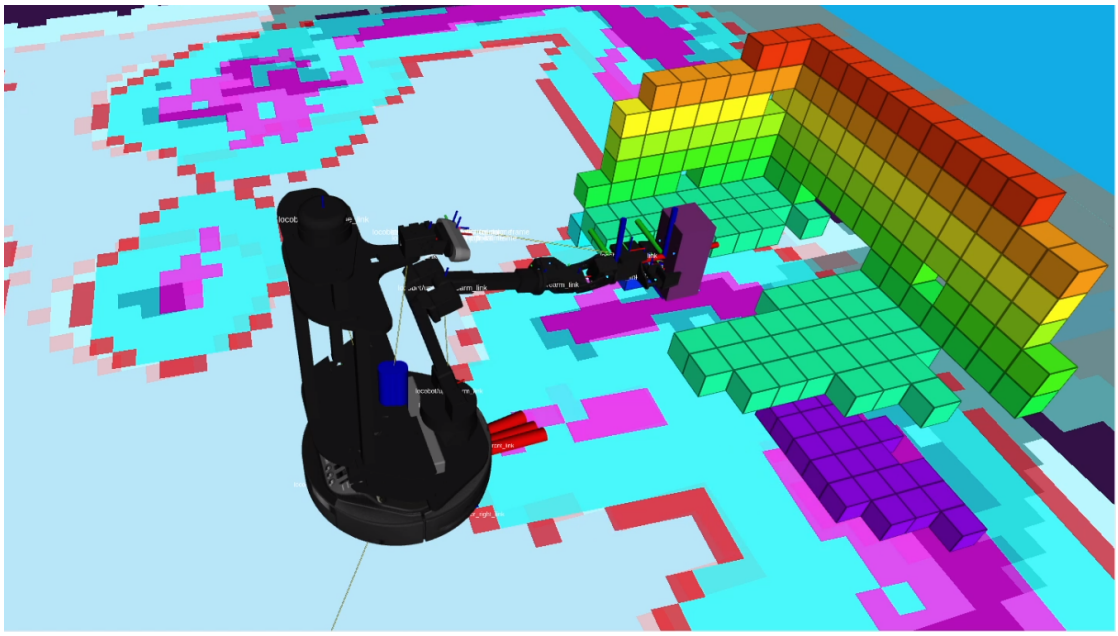


Figure 8.19: The planning scene after adding the box to be picked, the voxels are cleaned around the zone where the pick happens

Then, if we add an object to be picked in the planning scene, as shown in Figure 8.19, the **planning_scene_monitor** automatically cleans the planning scene in that part of the scene so that the arm can reach the object. In conclusion, the main drawback behind this solution is represented by the limitations given by the hardware so that the obstacle map takes some seconds before being constructed. This means that the manipulator now takes about two minutes to perform a pick or place action. For what we tested, the best working algorithm is **RRT*** so we selected this solution as default for all our tests.

8.6 Object detection testing

8.6.1 ARTag test

We performed a simple experiment to analyze which size of ARTag to experiment with. We test, in particular, the maximum error in retrieving the pose of the ARTag when the robot is positioned at a certain distance. In this way, we obtain the maximum recognition distance and the uncertainty in computing the position at a certain distance. Furthermore, the sizes under test are compatible with the medicines box, so they must be a manageable size.

ARTag dimension [cm]	Maximum distance recognition [m]	Uncertainty in position computation [cm]
5x5	2.1	$0.6 \div 1$
4x4	2	$1 \div 3$
3x3	1.5	> 5
2x2	1.2	> 5

Table 8.1: Results from the experiment to determine the best choice for the size of the markers

As a result of this experiment, we selected the ARTag of size 5.5x5.5 cm maximizing the size because we noticed that by enlarging the dimension the estimation of the pose gets better and more precise.
In conclusion, we must remember that the marker is positioned on a well-lit shelf for optimal recognition of the item.

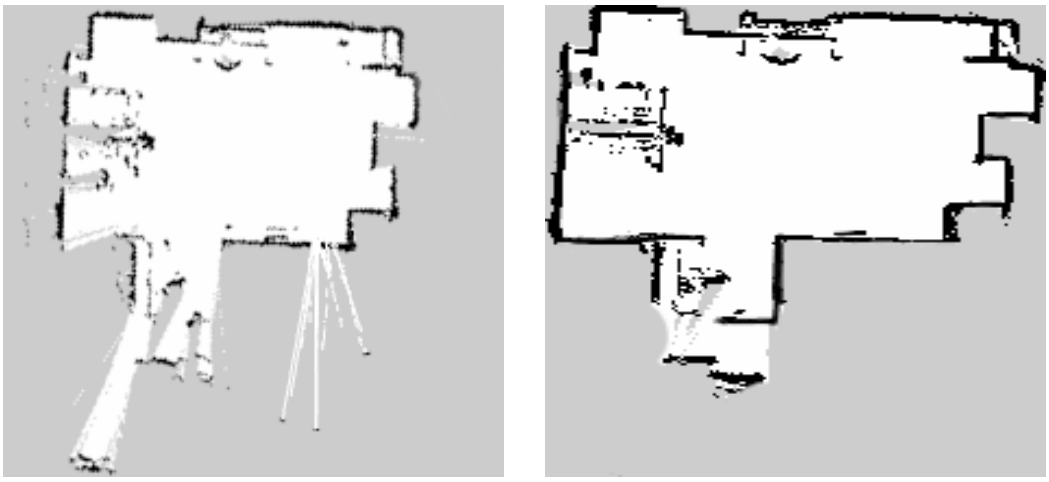
Figure 8.20: Here is a picture of the medicine with the marker positioned on the front.

8.7 Mobile Base Testing

8.7.1 SLAM Testing

The results for Slam are shown in Figures 8.21 and 8.22. We tested SlamToolbox [62] and RTABMap [63]. The results of the first package, being based only on LIDAR sensor results are highly susceptible to the reflective and transparent surfaces of the office. Especially in 8.21 we can see the effects of the noise coming from the glass surfaces. The second image is the result of SLAM from the RTABMap package.

The complete office area was mapped as shown in Figure 8.22. The left image shows the raw results of SlamToolbox, the noise comes from the glass surfaces where the LIDAR depth measurements reach the opposite side of the room or are noisy from the reflections. Especially there is an issue with angular drift when mapping in the office corridor (bottom of image), which appears to be at an angle with respect to the office room (top right of image).



(a) Office SlamToolbox Results **(b)** Office RTABMap Results

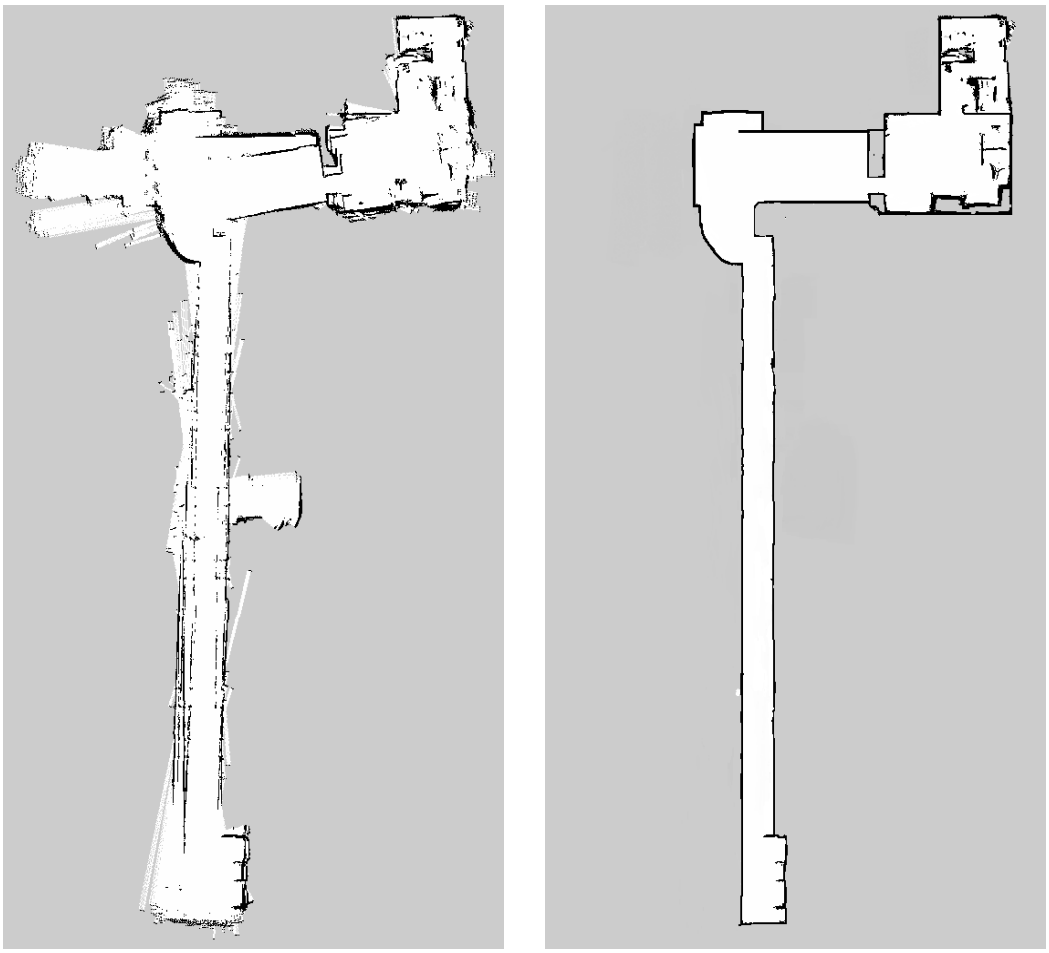
Figure 8.21: Office grid map

8.7.2 Path Planning testing

Path planning was tested by setting up different scenarios and environments for the robot to traverse as listed below. The Locobot behaves correctly in all of the tests. Even without human detection, A* global planning together with TEB local planning reacts quickly to a sudden obstacle in front of the mobile base, stopping without collision. Adding human detection further improves this behaviour. The images show Rviz [77], the visualization software we used when testing on the real robot. This allows us to visualize topic outputs, the map and how the robot sees the space around it. We chose a fixed starting and goal position so as to make the tests repeatable. The starting and goal positions can be seen in image (a) of 8.24.

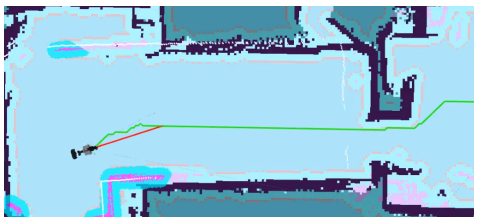
No Obstacles

We first performed a simple test with no obstacle to record a baseline shown in Figure 8.24. As expected from Section 7.1.2, the global planner finds a stair-like path (green) and the local planner (red) follows it using curving patterns that achieve a shorter route overall.



(a) Raw Map Result from SlamToolbox **(b)** Clean Map from RTABmap Person

Figure 8.22: Complete office occupancy grid map

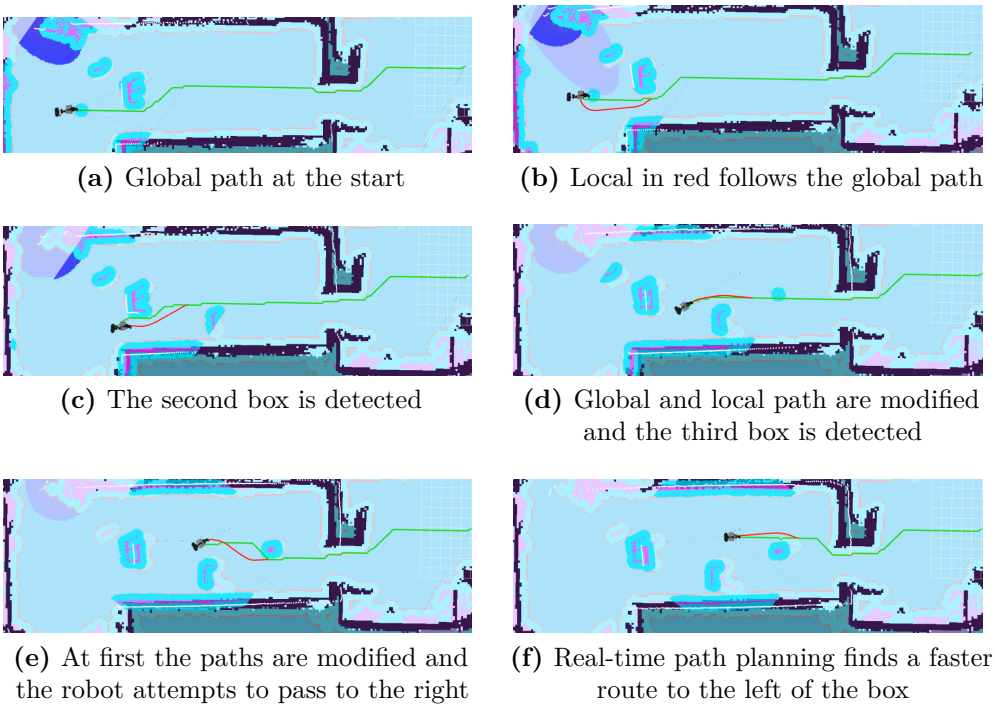


(a) Global path in green and Local path in red

Figure 8.23: Path planning with no obstacles.

Obstacles present

To test the local planner at first we placed static obstacles that did not appear on the map. The boxes were of different heights to test both the LIDAR and RGB-D sensors.



(a) Global path at the start

(b) Local in red follows the global path

(c) The second box is detected

(d) Global and local path are modified and the third box is detected

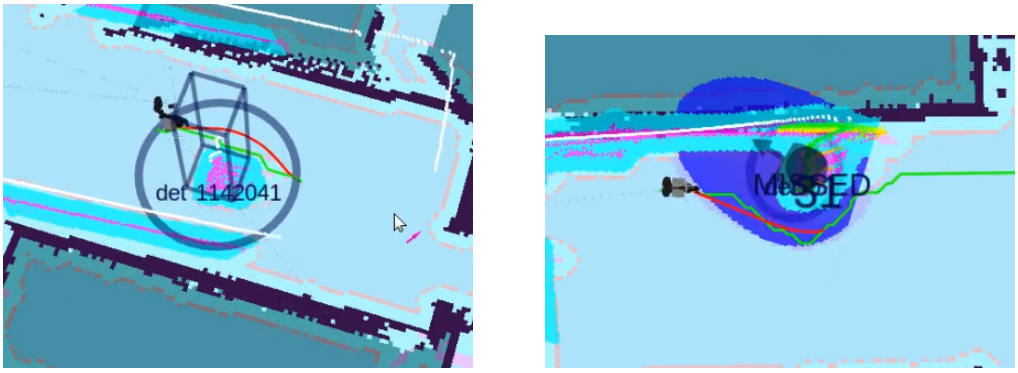
(e) At first the paths are modified and the robot attempts to pass to the right

(f) Real-time path planning finds a faster route to the left of the box

Figure 8.24: Obstacles path planning.

Static Person

The images in Figure 8.25 show how the gaussian cost around a standing person changes the path planning. On the left (a) we detect a person and do not build the gaussian around them, the global path planner sticks to the obstacle and the local planner respects the inflation radius. On the right (b) the detected person has a gaussian. This way we see both the path planners increase their distance to respect the added cost.



(a) Person not recognized leads to the robot passing closely

(b) Detected human generates a gaussian that forces a larger pass-by distance

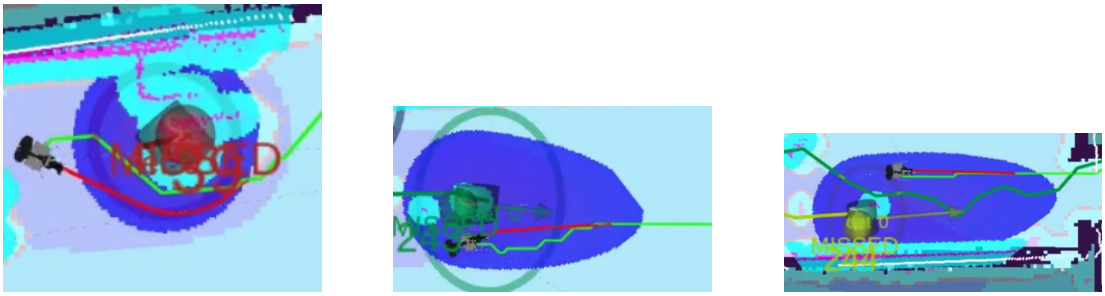
Figure 8.25: Standing person path planning.

Moving Person with Obstacles

We tested the gaussian function's ability to change shape according to different speeds as stated in Section 7.3.2. By passing the robot in front and from behind we tested the ability of the LIDAR to detect humans without confirmation from the RGB-D based detector. Different modalities tested where :

- Person stopping in front
- Person passing from behind
- Person passing in front
- Two people passing from behind

The robot reacts as expected in all of these scenarios. The gaussian is deformed as shown in Figure8.26. The human costmap shape forces the robot to move out of the way of the person passing from behind or in front. If the human steps in front of the robot it stops as expected and waits for the costmap to return to free space.



(a) Static Person

(b) Slow Person

(c) Fast Person

Figure 8.26: Images showing how speed affects the gaussian function shape (colour purple)

Chapter 9

Conclusions and future works

Working on this project has been stimulating and has undoubtedly helped us deepen our knowledge of the mechanism governing ROS as middleware operating systems. The platform where we developed our application, the Locobot WX250, is a good research platform that, in most cases, guarantees good performance, especially from the computational point of view. Furthermore, the application we developed is available on GitHub to improve and eventually test other systems since our primary goal was to develop the most general and flexible software architecture possible.
The main disadvantage of this application relies on the hardware that composes the mobile manipulator. The motors are servos that can guarantee a certain level of precision which is not suitable for applications of this kind; moreover, the torque that can be provided could carry only payloads up to 250 grams. Another drawback is that ROS, as we know, still needs to be implemented in the industrial system as it is because it is subject to message losses and latency in refreshing topics. Industrial applications need to be real-time and guarantee a certain level of security that ROS could not provide. Therefore, future work is likely attractive to implement ROS2, which is more industrial-oriented. For what concerns the camera, the Intel D435 provides, as we understood from our experiments, limitations for what concerns the point cloud. To guarantee a better working of Octomap, we would prefer a camera with a dense point cloud so that the obstacle grid constructed time by time could have a higher resolution. Furthermore, we noticed that the NUC provided with the robot is limited in computational power, so some operations require some delays that, with more powerful hardware, such as an Nvidia Jetson, could be avoided.
We propose improvements from the hardware point of view regarding, for example, installing a camera on the arm's end-effector and implementing suction cups instead

of fingers. The suction cup is certainly a more complex system but can guarantee the grasping of numerous types of items independently of their shape or width.
Though human detection was surprisingly effective, it suffered from false positives and issues with tracking. A time-consuming but necessary improvement would be the recording and annotating of new datasets and training of new models from the Locobot sensors.
Further improvements can also be made from the software point of view. For example, the architecture governing the robot could be made to provide better handling of errors in case of either software or hardware failure. In addition, with a more powerful NUC and a camera mounted on the manipulator, we could substitute the marker recognition with proper recognition of the objects with a neural network. Employing a neural network for object recognition will lead to a higher effort from the computational point of view but will result in a more flexible system.

Bibliography

[1] KUKA Roboter GmbH.
https://commons.wikimedia.org/wiki/File:Automation_of_foundry_with_robot.jpg (cit. on p. 2).

[2] Vivek Annem, Pradeep Rajendran, Shantanu Thakar, and Satyandra Gupta. «Towards Remote Teleoperation of a Semi-Autonomous Mobile Manipulator System in Machine Tending Tasks». In: June 2019. DOI: 10.1115/MSEC2019-3027 (cit. on p. 3).

[3] Leo Pauly, M. V. Baiju, P. Viswanathan, Praveen Jose, Divya Paul, and Deepa Sankar. «CAMbot: Customer assistance mobile manipulator robot». In: *2015 IEEE Bombay Section Symposium (IBSS)*. 2015, pp. 1–4. DOI: 10.1109/IBSS.2015.7456644 (cit. on p. 3).

[4] Maria E. Cabrera, Tapomayukh Bhattacharjee, Kavi Dey, and Maya Cakmak. «An Exploration of Accessible Remote Tele-operation for Assistive Mobile Manipulators in the Home». In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. 2021, pp. 1202–1209. DOI: 10.1109/RO-MAN50785.2021.9515511 (cit. on p. 4).

[5] Rohit Nilesh Mehta, Hitaishi Vijay Joshi, Inziya Dossa, Rahul Gyanch Yadav, Sarika Mane, and Mansing Rathod. «Supermarket Shelf Monitoring Using ROS based Robot». In: *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*. 2021, pp. 58–65. DOI: 10.1109/ICOEI51242.2021.9452895 (cit. on p. 7).

[6] Wenqian Hu, Jie Qiu, Fan Zhang, Qian Yang, Pengbo Wang, and Changing Geng. «Control and Fetching Strategy of Goods-Picking Robot in the Self-service Supermarket». In: *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. 2019, pp. 1285–1288. DOI: 10.1109/CYBER46603.2019.9066628 (cit. on p. 7).

[7] documentation RT-Middleware website.
https://openrtm.org/openrtm/en/doc/aboutopenrtm/rtmiddleware (cit. on p. 9).

[8] ZeroMQ library.
`https://zeromq.org/get-started/` (cit. on p. 9).

[9] *ROS master-node-topic communication*. `https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png` (cit. on p. 10).

[10] Randall Smith, Matthew Self, and Peter Cheeseman. «A stochastic map for uncertain spatial relationships». In: *Proceedings of the 4th international symposium on Robotics Research*. 1988, pp. 467–474 (cit. on p. 12).

[11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. «A Flexible and Scalable SLAM System with Full 3D Motion Estimation». In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011 (cit. on p. 12).

[12] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. «FastSLAM: A factored solution to the simultaneous localization and mapping problem». In: *Aaai/iaai* 593598 (2002) (cit. on p. 12).

[13] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. «A tutorial on graph-based SLAM». In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (cit. on p. 13).

[14] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. «Visual SLAM algorithms: A survey from 2010 to 2016». In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), pp. 1–11 (cit. on p. 13).

[15] Ilmir Z Ibragimov and Ilya M Afanasyev. «Comparison of ROS-based visual SLAM methods in homogeneous indoor environment». In: *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*. IEEE. 2017, pp. 1–6 (cit. on p. 13).

[16] Oussama Khatib. «Real-time obstacle avoidance for manipulators and mobile robots». In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404 (cit. on p. 14).

[17] Steven M LaValle et al. «Rapidly-exploring random trees: A new tool for path planning». In: (1998) (cit. on p. 14).

[18] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992 (cit. on p. 15).

[19] Edsger Wybe Dijkstra. «A note on two problems in connexion with graphs:(Numerische Mathematik, 1 (1959), p 269-271)». In: (1959) (cit. on p. 15).

[20] Peter E Hart, Nils J Nilsson, and Bertram Raphael. «A formal basis for the heuristic determination of minimum cost paths». In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107 (cit. on p. 15).

[21] A. Stentz. «Optimal and efficient path planning for partially-known environments». In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 3310–3317 vol.4. DOI: 10.1109/ROBOT.1994.351061 (cit. on p. 15).

[22] D. Fox, W. Burgard, and S. Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977 (cit. on p. 16).

[23] Brian Gerkey and Kurt Konolige. «Planning and control in unstructured terrain». In: (Jan. 2008) (cit. on p. 16).

[24] Sean Quinlan and Oussama Khatib. «Elastic bands: Connecting path planning and control». In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 802–807 (cit. on p. 16).

[25] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. «Trajectory modification considering dynamic constraints of autonomous robots». In: *ROBOTIK 2012; 7th German Conference on Robotics*. VDE. 2012, pp. 1–6 (cit. on p. 16).

[26] P. Viola and M. Jones. «Rapid object detection using a boosted cascade of simple features». In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517 (cit. on p. 17).

[27] Navneet Dalal and Bill Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893 (cit. on p. 17).

[28] Chuyi Li et al. «YOLOv6: a single-stage object detection framework for industrial applications». In: *arXiv preprint arXiv:2209.02976* (2022) (cit. on p. 17).

[29] Radu Adrian Ciora and Carmen Mihaela Simion. «Industrial applications of image processing». In: *Acta Universitatis Cibiniensis. Technical Series* 64.1 (2014), pp. 17–21 (cit. on p. 17).

[30] Yandong Li, Zongbo Hao, and Hang Lei. «Survey of convolutional neural network». In: *Journal of Computer Applications* 36.9 (2016), p. 2508 (cit. on p. 17).

[31] Angus Leigh, Joelle Pineau, Nicolas Olmedo, and Hong Zhang. «Person tracking and following with 2d laser scanners». In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 726–733 (cit. on p. 18).

[32] Edward T Hall. «A system for the notation of proxemic behavior». In: *American anthropologist* 65.5 (1963), pp. 1003–1026 (cit. on pp. 18, 81).

[33] Jiyu Cheng, Hu Cheng, Max Q-H Meng, and Hong Zhang. «Autonomous navigation by mobile robots in human environments: A survey». In: *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2018, pp. 1981–1986 (cit. on p. 18).

[34] Peter Trautman and Andreas Krause. «Unfreezing the robot: Navigation in dense, interacting crowds». In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 797–803 (cit. on p. 18).

[35] Carlos Medina Sánchez, Matteo Zella, Jesús Capitán, and Pedro J Marrón. «From Perception to Navigation in Environments with Persons: An Indoor Evaluation of the State of the Art». In: *Sensors* 22.3 (2022), p. 1191 (cit. on p. 19).

[36] Yunfan Wang and Xian Guo. «Memory-based Stochastic Trajectory Optimization for Manipulator Obstacle Avoiding Motion Planning». In: *2022 7th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. 2022, pp. 188–194. DOI: 10.1109/ACIRS55390.2022.9845580 (cit. on p. 19).

[37] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. «STOMP: Stochastic trajectory optimization for motion planning». In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 4569–4574. DOI: 10.1109/ICRA.2011.5980280 (cit. on pp. 19, 29, 30).

[38] Wang Xinyu, Li Xiaojuan, Guan Yong, Song Jiadong, and Wang Rui. «Bidirectional Potential Guided RRT* for Motion Planning». In: *IEEE Access* 7 (2019), pp. 95046–95057. DOI: 10.1109/ACCESS.2019.2928846 (cit. on p. 20).

[39] Pankaj Wajire, Savita Angadi, and Lokesh Nagar. «Image Classification for Retail». In: *2020 International Conference on Industry 4.0 Technology (I4Tech)*. 2020, pp. 23–28. DOI: 10.1109/I4Tech48345.2020.9102699 (cit. on p. 21).

[40] M. Fiala. «ARTag, a fiducial marker system using digital techniques». In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. 2005, 590–596 vol. 2. DOI: 10.1109/CVPR.2005.74 (cit. on pp. 21, 22).

[41] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Manuel Marín-Jiménez. «Automatic generation and detection of highly reliable fiducial markers under occlusion». In: *Pattern Recognition* 47 (June 2014), pp. 2280–2292. DOI: 10.1016/j.patcog.2014.01.005 (cit. on p. 22).

[42] Reynaldo Cruz Villagomez and Jhon Ordoñez. «Robot Grasping Based on RGB Object and Grasp Detection Using Deep Learning». In: *2022 8th International Conference on Mechatronics and Robotics Engineering (ICMRE)*. 2022, pp. 84–90. DOI: 10.1109/ICMRE54455.2022.9734075 (cit. on p. 23).

[43] Eitan Marder-Eppstein. *Navigation Package*. https://wiki.ros.org/navigation?distro=noetic, Last accessed on 2022-10-30 (cit. on pp. 24, 34).

[44] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. «Reducing the barrier to entry of complex robotic software: a MoveIt! case study». In: *arXiv preprint arXiv:1404.3785* (2014) (cit. on pp. 24, 27).

[45] Ioan A. Sucan, Sachin Chitta. *MoveIt website*. https://moveit.ros.org/, Last accessed on 2022-10-30 (cit. on pp. 25, 27, 75).

[46] Wikipedia contributors. *Motion planning — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Motion_planning&oldid=1109255335. [Online; accessed 26-September-2022]. 2022 (cit. on p. 27).

[47] *Covariant Hamiltonian Optimization for Motion Planning*. https://www.nathanratliff.com/thesis-research/chomp (cit. on p. 28).

[48] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. «CHOMP: Gradient optimization techniques for efficient motion planning». In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817 (cit. on p. 28).

[49] *Stochastic Trajectory Optimization for Motion Planning*. http://wiki.ros.org/stomp_motion_planner (cit. on p. 29).

[50] *Open Motion Planning Library*. https://ompl.kavrakilab.org/index.html (cit. on pp. 31, 68).

[51] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. «The Open Motion Planning Library». In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651 (cit. on pp. 32, 33).

[52] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. «OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees». In: *Autonomous Robots* (2013). Software available at https://octomap.github.io. DOI: 10.1007/s10514-012-9321-0. URL: https://octomap.github.io (cit. on pp. 33, 63).

[53] Eitan Marder-Eppstein. *Transform tree Package*. http://wiki.ros.org/tf, (cit. on p. 34).

[54] Eitan Marder-Eppstein. *Costmap 2d Package*. http://wiki.ros.org/costmap_2d?distro=noetic, (cit. on pp. 35, 81).

[55] *Move Base Package*. http://wiki.ros.org/move_base (cit. on pp. 36, 37).

[56] *Navigation Package*. http://wiki.ros.org/navigation?distro=noetic (cit. on p. 36).

[57] Trossen Robotics. *Trossen robotics webpage*. https://www.trossenrobotics.com/. [Online; accessed 20-July-2022] (cit. on pp. 38–40).

[58] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*. Vol. 200. Springer, 2016, pp. 19–21 (cit. on p. 40).

[59] Yunhao Pan, Jingdi Tang, Qingyuan Zhao, and Shengyi Zhu. «Forward and Inverse Kinematics Modeling and Simulation of Six-axis Joint Robot Arm Based on Exponential Product Method». In: *2020 IEEE 3rd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*. 2020, pp. 372–375. DOI: 10.1109/AUTEEE50969.2020.9315719 (cit. on p. 42).

[60] MoveIt documentation for Octomap. https://ros-planning.github.io/moveit_tutorials/doc/perception_pipeline/perception_pipeline_tutorial.html (cit. on p. 64).

[61] ar_track_alvar documentation. http://wiki.ros.org/ar_track_alvar (cit. on pp. 65, 67).

[62] 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 (cit. on pp. 76, 101).

[63] Mathieu Labbé and François Michaud. «RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation». In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446. DOI: https://doi.org/10.1002/rob.21831. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21831. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831 (cit. on pp. 76, 101).

[64] Alberto Elfes. «Using occupancy grids for mobile robot perception and navigation». In: *Computer* 22.6 (1989), pp. 46–57 (cit. on p. 76).

[65] *NavFN Ros*. http://wiki.ros.org/navfn?distro=noetic (cit. on p. 77).

[66] *Global Planner*. http://wiki.ros.org/global_planner (cit. on pp. 77, 78).

[67] Maximilian Pittner, Markus Hiller, Florian Particke, Lucila Patino-Studencki, and Joern Thielecke. «Systematic analysis of global and local planners for optimal trajectory planning». In: *ISR 2018; 50th International Symposium on Robotics*. VDE. 2018, pp. 1–4 (cit. on p. 77).

[68] B Cybulski, Agnieszka Wegierska, and Grzegorz Granosik. «Accuracy comparison of navigation local planners on ROS-based mobile robot». In: *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*. IEEE. 2019, pp. 104–111 (cit. on p. 77).

[69] *Timed ELastic Band Ros*. `http://wiki.ros.org/teb_local_planner` (cit. on pp. 77, 79).

[70] Radu Bogdan Rusu and Steve Cousins. «3D is here: Point Cloud Library (PCL)». In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 2011 (cit. on pp. 80, 84).

[71] *People Package*. `http://wiki.ros.org/people` (cit. on p. 80).

[72] *Social Navigation Layers Package Planner*. `http://wiki.ros.org/navigation_layers` (cit. on pp. 80, 81).

[73] *Spencer People Tracking Package*. `https://github.com/spencer-project/spencer_people_tracking` (cit. on p. 80).

[74] *Leg Detector Package*. `http://wiki.ros.org/leg_detector` (cit. on p. 80).

[75] Timm Linder, Stefan Breuers, Bastian Leibe, and Kai O Arras. «On multi-modal people tracking from mobile platforms in very crowded and dynamic environments». In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 5512–5519 (cit. on p. 83).

[76] Timm Linder and Kai O Arras. «People detection, tracking and visualization using ros on a mobile service robot». In: *Robot Operating System (ROS)*. Springer, 2016, pp. 187–213 (cit. on p. 83).

[77] *Rviz Software Package*. `http://wiki.ros.org/rviz` (cit. on p. 102).