POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

ROS2-Based AMR System for Mapping and Navigation in Unknown Indoor Environments

Supervisors
Prof. Alessandro Rizzo
Ing. Orlando Tovar Ordonez
Edoardo Todde

Candidate
Miriana Casuccio

Academic Year 2023-2024

Acknowledgements

I would like to express my sincere gratitude to Professor Alessandro Rizzo for trusting my work and giving me the opportunity to do this thesis. A special thanks goes to the entire CIM 4.0 team, especially Orlando Tovar, for providing me with the opportunity to conduct my thesis in such an inspiring environment. I am particularly grateful to my tutor, Edoardo Todde, for his guidance during my work, and not only.

I would also like to acknowledge the contribution of my fellow thesis for making the whole process enjoyable and productive.

Thanks should be given to all my family who, starting from the beginning of my academic journey, have always supported me. In difficult times they stood by me and never doubted me. You made me the person I am today.

Last but not least, a heartfelt thank you to all my dearest friends who have patiently listened to my endless monologues and complaints. It's been a privilege to share both the highs and lows with you.

Abstract

The rapid advancement of autonomous systems and automation technologies continues to revolutionize industrial processes, aligning with the goals of Industry 4.0. This thesis presents an enhanced Autonomous Mobile Robot (AMR) system intended for advanced indoor navigation and exploration, building on the groundwork established by the FIXIT project at CIM4.0. The primary objective of this research is to develop and implement a robust SLAM (Simultaneous Localization and Mapping) algorithm utilizing the latest capabilities of ROS2 (Robot Operating System 2).

A key focus of this study is a comprehensive comparison of different SLAM approaches using the Nav2 library within the ROS2 framework. This analysis covers various algorithms available in Nav2, including grid-based and topological mapping methods, as well as different localization techniques such as AMCL (Adaptive Monte Carlo Localization) and EKF (Extended Kalman Filter). The comparison evaluates these approaches based on mapping accuracy, computational efficiency, and adaptability to dynamic environments.

Based on this analysis, an advanced SLAM methodology is developed, integrating the most effective elements from the compared approaches. This custom solution leverages Nav2's modular architecture and ROS2's improved distributed computing capabilities, allowing for efficient path planning and map optimization. The entire system is implemented using ROS2, taking advantage of its enhanced tools for simulation, visualization, and real-world deployment. Rigorous testing is conducted in various simulated environments using updated versions of RViz and Gazebo, which are now more tightly integrated with the middleware. These simulations demonstrates the robot's improved capabilities in active exploration, obstacle avoidance, and efficient mapping, showcasing the benefits of this approach.

Finally, in the carefully controlled laboratory environment at CIM 4.0, real-world experiments were carried out to assess the robustness and performance of the created AMR system. The outcomes show how accurately and dependably the AMR can navigate through a variety of situations on its own, including unknowns areas and dynamic barriers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As one of the cornerstones of the third industrial revolution, industrial automation is playing an increasingly central role in industrial production. Defined as the set of technologies and solutions designed to manage a machine or process in an automatic way, its use not only improves people's work by helping them with specific tasks but even taking their place, thus preventing dangerous situations. There are many types of autonomous systems, the main are grouped into Automated Judged Vehicle (AGV) and Automated Mobile Robot (AMR). The first ones follow predefined paths, usually by means of ground guides (magnetic tapes, induction wires) or visual markers; programming is relatively simple and requires careful configuration of the working environment. Whereas, AMRs use a combination of sensors (lasers, cameras, sonar) to autonomously navigate an unknown and dynamic environment, creating maps and locating themselves within them. Programming is more complex and requires the use of artificial intelligence algorithms for perception, route planning, and decision making.

In an Industry 4.0 environment, where production lines may change or be updated frequently, the flexibility of AMRs is a basic necessity. Industry 4.0 is a combination of interconnected cutting-edge technologies that have the potential to transform manufacturing in all its forms, reflecting the automated and connected industrial systems that constitute industrial evolution. This is where Competence Industry Manufacturing 4.0 comes in.

The objective of this thesis is to develop and implement a simultaneous localization and mapping (SLAM) solution for autonomous exploration of a dynamic unknown environment. Autonomous exploration of an unknown environment is a very challenging task of great interest in mobile robotics research. To perform exploration, an extension of the SLAM algorithm, called Active SLAM, is required, which autonomously plans paths while mapping and localizing within an environment. First we focused in a thorough research on all the methods studied so far for autonomous driving, starting from global path planning to more specific local path planning including obstacle avoidance; then moving on to implement the main packages, such as 'slam toolbox' and 'nav2', of ROS2 (Robot Operating System), a meta operating system widely used in robotics for these purposes. The implementation was first done in a simulation environment, taking advantage of various software such as Gazebo and RViz, and then finally moved on to testing

them experimentally on the AMR.

1.1 Structure of thesis

The thesis is structured in the following order:

- The second chapter introduces the research and subsequent state of the art study
- The third chapter describes the ROS platform, its components, the difference between ROS and ROS2 and explains why this change.
- The SLAM algorithm is the main topic of the fourth chapter, which also describes its different varieties, including passive or active SLAM and full or online SLAM.
- The results of the algorithm in a simulation environment, using tools like RViz and Gazebo, are displayed in the fifth chapter. With a deeper look at the packages used.
- The rover's hardware design is covered in detail in the sixth chapter, beginning with the sensors that are employed and ending with the boards that are chosen.
- Lastly, the results are examined and recommendations for further work are provided in the eighth chapter.

Chapter 2

State of the art

2.1 Path Planning

Autonomous exploration presents several challenges, the main ones of which include path planning, obstacle detection, and localization and mapping (i.e. slam). Determining the optimal path to explore the unknown environment requires efficient algorithms that balance exploration and efficiency; in addition, identifying and avoiding obstacles such as walls, people, and/or objects is crucial for the robot's safety; and last, creating accurate maps of the environment and estimating the robot's position are complex. Sensors must be robust to operate under variable and uncertain conditions. Motion planning has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game, architectural design, robotic surgery, and the study of biological molecules. [28]

For path planning there are several methods in the literature, a primary distinction is made between global and local planning. The first one refers to navigation in a known environment by calculating the shortest path from an initial point to the target point. While as for local navigation, it refers to the ability to move in unknown environments through the information acquired by the sensors it is equipped with which it is able to avoid obstacles and thus collision with them. The first method used for global routing, and thus navigation in an a priori known environment, is the Dijkstra algorithm. And the subsequent developments are A*, D*, D-light and lastly RRT/RRT*. Instead, the methods used that enable autonomous robot navigation in an environment with the static or dynamic obstacles, having the advantage of constantly reprogramming the path whenever the robot encounters obstacles, are mainly: potential field, field histogram, DWA.

2.2 Global Path Planning

Global path planning algorithms are used in mobile robotics to find a coherent path from the starting point to the goal point, taking into account the eventual occurrence of obstacles in the navigation environment. There are several global path planning algorithms that can be exploited for navigation in indoor and outdoor environments. the most widely used are undoubtedly the A*, D* and RRT algorithms.

2.2.1 A* Algorithm

A* is an extension of Dijkstra's algorithm introduced by Peter Hart, Nils Nilsson and Bertram Raphael in 1968. [12] It is widely used to find the shortest path using a best-first search; its efficiency comes from its ability to use heuristic information to guide the search to the desired destination, combining the cost of the path from the initial node to the current node and an estimate of the cost of the current node to the target node.

$$f(x) = h(x) + g(x) \tag{2.1}$$

Where $h(x)$ is a heuristic estimate of the distance to the target location and $g(x)$ denotes the cost from the starting node to the current node.

The heuristic component $h(x)$ of the function must be an admissible heuristic, that is, it cannot overestimate the distance to the target. The heuristic is called monotone or consistent if it meets the supplementary requirement

$$h(x) \leq d(x, y) + h(y) \tag{2.2}$$

for each edge x, y of the graph, and d specifies the edge length.

The heuristic determines the time complexity of A*. In the worst-case scenario, the number of nodes visited is exponential; however, if the area of search is a tree, it is polynomial.

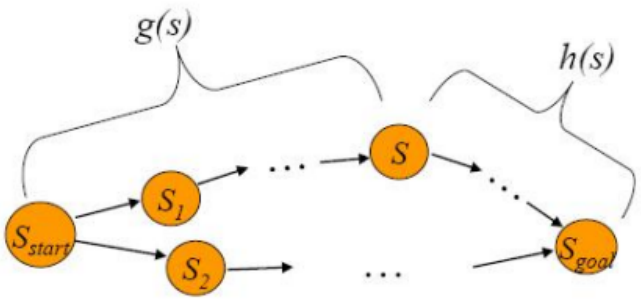


Figure 2.1. A* Algotirhm [4]

2.2.2 D* and D-Light Algorithms

The D* algorithm is a path-finding algorithm designed for iterative computation of shortest paths in dynamic environments. It is particularly suitable for mobile robotics applications, where the environment may change over time and the agent must be able to adapt to these changes.

The first to introduce this method as D* was Anthony Stentz in 1994. The name is derived from the term 'Dynamic A*', in that it behaves like A* but with the difference that the costs of the arcs can change during the running of the algorithm. Thus, if the A* algorithm traverses the graph from beginning to end, D* traverses it in reverse starting from the target node. The exact cost of the target point is known to each expanded node, which then calculates the cost with the next node. The process ends when the expanded node then becomes the starting node.

Assumptions are made by the robot to identify the shortest path, for example, the unknown part of the terrain to check whether it is obstacle-free or not. It updates its map when it notices new changes, and replans the route if necessary. The procedure is repeated until the target coordinates are obtained or it is established that they cannot be achieved.

While, D-Lite* algorithm is a variant of the D* algorithm that seeks to optimize performance and computational complexity without losing the ability to handle dynamic environments. It, therefore, uses a number of optimizations and strategies to reduce computational complexity such as, for example, updating the path only as necessary, i.e., not periodically but if any changes occur.

2.2.3 Rapidly-exploring Random Tree Algorithm

In complex and unknown configuration spaces, this algorithm is commonly used to quickly find the robot's path.

The algorithm starts from the single node corresponding to the starting position, and randomly generates a point in the configuration field thus creating a tree extending each time towards the nearest node. The new node is checked if it is connected according to guidelines, such as if the path taken is unobstructed. The whole thing is repeated until a valid path is found. The algorithm begins with the single node that corresponds to the starting position and randomly generates a point in the configuration field, forming a tree that reaches the nearest node. The new node is checked to ensure that it is connected according to guidelines, such as ensuring that the path taken is unobstructed. The whole thing is repeated until a valid path is found. After arriving at destination,it's possible to recover the path by climbing the tree from the destination back to the starting position.

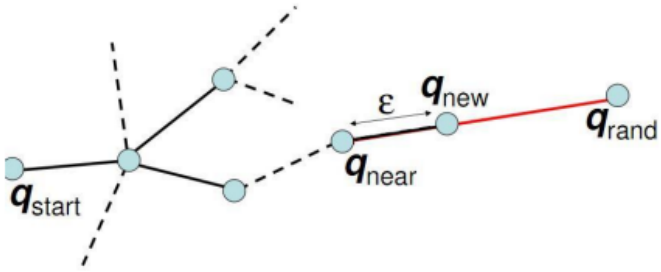


Figure 2.2. Development of RRT* Algotirhm [4]

The goal of RRT*, instead, is to locate routes that are more optimal and efficient than the basic RRT algorithm, as it updates and optimizes them during the search process. In situations where cost minimization is crucial, it is particularly useful, such as when planning the movement of autonomous robots in environments with cost constraints.

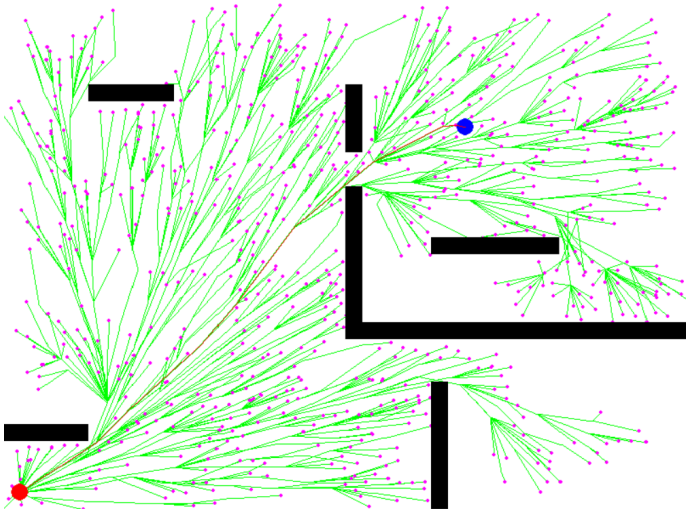


Figure 2.3. RRT* Algotirhm [9]

2.3 Local Path Planning

The real-time perception system's data provides local route planning, which is crucial for responding to constraints and changes in the environment. Robots in a local route planner are frequently guided by a global route that was created with the help of a global planning strategy. The robot follows the easiest path until it runs into obstacles. To avoid obstacles, the robot deviates from the path and simultaneously updates important information, such as the distance between its current position and the target point. Continuous awareness of the distance between the target point and its current position is necessary for this type of route planning to achieve the target precisely.

2.3.1 Artificial Potential Field method

The Artificial Potential Field (APF) method was introduced by Khabit and Krogh in 1995 [8]. The gradient descent search strategy, which seeks to minimize the potential function, is the basis of the APF methodology.

APF method is based on the assumption of modelling the surrounding environment as a force field, where a positive attraction value is assigned to the target or target point position, creating an attractive force field that pushes the agent towards the target; while a positive repulsion value is assigned to obstacles in the environment, thus creating a repulsive force field that distances the agent from obstacles. When the agent approaches obstacles, the repulsive force is stronger, but it diminishes as he moves away.

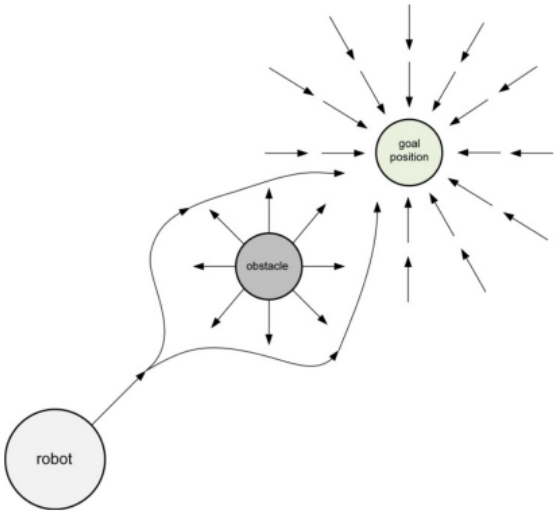


Figure 2.4. Artificial Potential Field [17]

The gravitational potential function is:

$$U_{att}(X) = \frac{1}{2} k \rho^2(X_R, X_G) \tag{2.3}$$

where, k is the attractive gain factor, $\rho(X_R, X_G) = ||X_G - X_R||$ is the distance between robot and target. Thus, the attractive field's negative gradient represents the attractive function:

$$F_{att}(X) = -\nabla U_{att}(X) = k(X_R - X_G) \tag{2.4}$$

Instead, the repulsive field is:

$$U_{rep}(X_i) = \begin{cases} \frac{1}{2}\eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})^2 & \rho(X_R, X_i) \leq \rho_0 \\ 0 & \rho(X_R, X_i) > \rho_0 \end{cases} \tag{2.5}$$

where, η_x is the repulsive gain factor, $\rho(X_R, X_i)$ is the distance between robot and obstacle, ρ_0 is the obstacle influence distance, which is connected with the dimension of the robot; the repulsive force is:

$$F_{rep}(X_i) == -\nabla U_{rep}(X_i) \begin{cases} \eta_x(\frac{1}{\rho(X_R,X_i)} - \frac{1}{\rho_0})\frac{e_i R}{\rho^2(X_R,X_i)} & \rho(X_R, X_i) \leq \rho_0 \\ 0 & \rho(X_R, X_i) > \rho_0 \end{cases} \tag{2.6}$$

The force that is ultimately applied to the robot is:

$$F = F_{att}(X) + \sum_{i=1}^{i=n} F_{rep}(X_i) \tag{2.7}$$

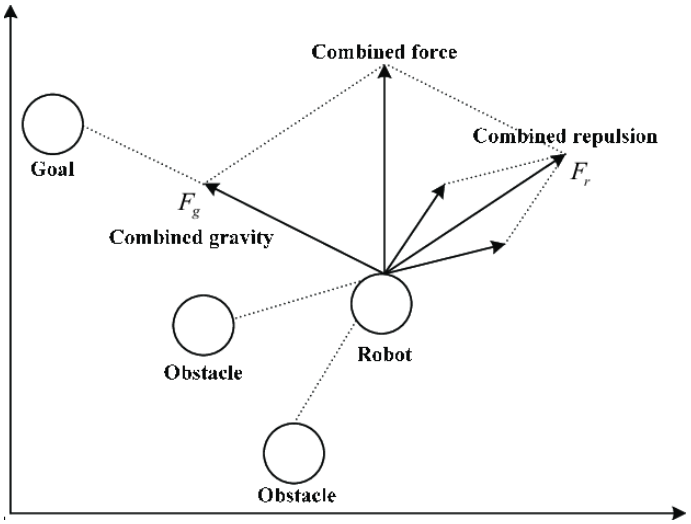


Figure 2.5. Diagram of Artificial Potential Field [30]

By using the potential field as a guide, the agent can plan their own movement. It is possible to use path feedback control to trace the gradient of the potential field and move towards regions with the lowest potential, which is the destination, while avoiding regions with the highest potential and obstacles.

2.3.2 Field Histogram

In this method, the space surrounding the robot is divided into a regular grid of cells and sensory data is represented in each cell using a histogram.

Sensors like cameras, lidar, and/or ultrasonic sensors are used by the robot to capture data about its environment. This data includes information about obstacles, edges of objects, walls, and other objects. For each grid cell, a histogram is updated based on the sensory data acquired by the robot. To make navigation decisions, the robot can use the field histogram to interpret the surrounding environment.

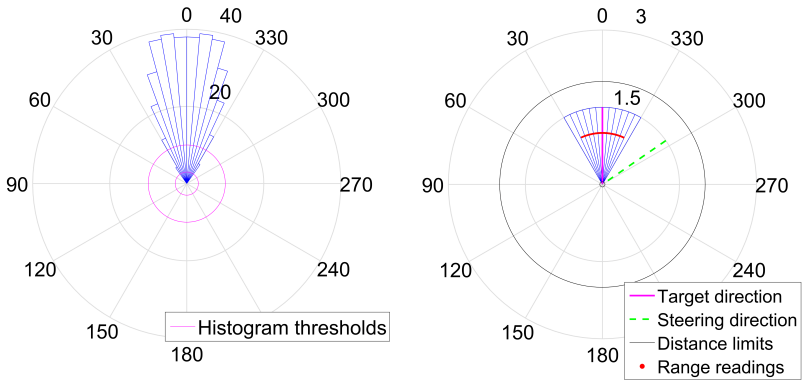


Figure 2.6. Examples of Field Histogram: on the left Polar Obstacles Density, on right Masked Polar Histogram [21]

2.3.3 Dynamic Window Approach

The Dynamic Window Approach (DWA), a velocity-based local planner, determines the ideal collision-free robot velocity needed to complete a task [7]. A Dynamic Window Approach (DWA) for a mobile robot transforms a desired Cartesian target into appropriate linear (v) and angular (w) velocities. The DWA's primary goals are to define a feasible velocity search space and determine the optimal velocity within that space. The search space encompasses velocities that, considering the robot's dynamic constraints and potential future velocity trajectories, enable safe navigation and collision avoidance.

The dynamic window is a region in the control space of vehicle that represents all possible speeds and steering positions that the vehicle can assume. This window is calculated by considering the dynamic capabilities of the vehicle, such as maximum speed, acceleration and maximum rotational speed. For each possible combination of linear speed and steering within the dynamic window, candidate trajectories are generated. These trajectories represent potential routes that the vehicle could follow, each candidate trajectory is evaluated using cost criteria which take into account objectives such as: minimize distance from destination, avoid obstacles, Reduce the deviation from the desired trajectory, maintain a comfortable speed. After evaluating all the trajectories, the one that minimizes the overall cost according to the defined criteria is selected and becomes the movement plan for the vehicle at the next instant.

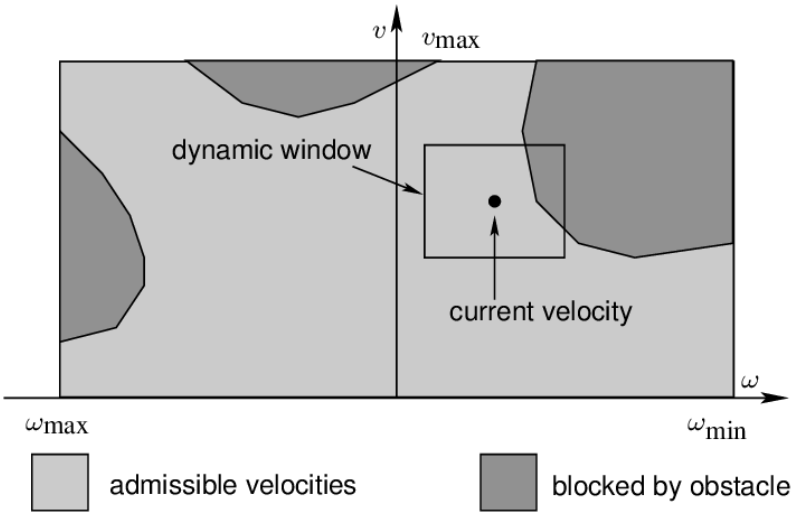


Figure 2.7. Dynamic Window Approach [26]

Chapter 3

ROS2 architecture with comparison with ROS

3.1 Introduction to ROS2

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. Since ROS was started in 2007, by a personal project of Keenan Wyrobek and Eric Berger (students at Stanford University and creators of the Stanford Personal Robot Program) with the aim of creating unique software for robotics [22], a lot has changed in the robotics and ROS community. The goal of the ROS2 project is to adapt to these changes, leveraging what is great about ROS and improving what isn't. [19] Such us, making it more robust, scalable and suitable for a wide range of robotic applications including autonomous robots, collaborative robots and drones.

ROS2 is one of the leading open-source platforms for developing robotic projects. Offers a rich set of tools for robot creation, control and monitoring, including advanced features such as graphical representation of parameters and 3D simulation. Its modular and intuitive design makes it suitable for a wide range of applications, from small home automation to complex industrial systems.

ROS2 also supports the analysis and visualization of robotic data through different modes: text-based tools (CLI) for examining connections between components, graphical interfaces (rqt) for visual representation of system architecture and 3D visualizers (RViz2) for real-time simulation of the robot in the environment.

3.1.1 Key features of ROS

To fully understand the functionalities of ROS, it is good to fix some basic concepts before we go further:

- Nodes: In an ROS system, a node is an autonomous computational process, conceived as a software module dedicated to a specific task. This granular modularity enables a flexible system organization, allowing the complex behaviour of a robot to be subdivided into well-defined sub components. Each node, identified by a unique name, manages a specific aspect of robotic behaviour such as sensory perception, motion planning, actuator control or decision making.

- Message: It acts as a communication unit between the nodes of a computational graph. Their user-defined structure allows for precise synchronization between the data exchanged, ensuring consistency of information within the system. The possibility of creating personalized messages lets to adapt the communication to the specific needs of each application, from simple sensory measurements to complex representations of the state of a system.

- Package: ROS packages are the fundamental mechanism for managing dependencies and breaking down code into coherent units. Each package clearly defines its dependencies on other packages, allowing easy system installation and configuration. This modular structure that groups related resources such as nodes, configuration files, datasets and libraries facilitates collaboration between developers and the creation of open-source projects.

The main mechanisms for communication between nodes within a distributed robot system are:

- Topic: The topics in ROS are the asynchronous and unidirectional communication channel through which nodes exchange messages. They are strongly typed communication channels, that is each topic is associated with a specific type of message; the typing of messages ensures compatibility between the data exchanged and prevents runtime errors. By adopting a publishing/subscription model, topics allow multiple nodes to publish messages on the same topic and other nodes to subscribe to receive them. This flexibility allows for the creation of distributed and scalable systems, in which nodes can communicate without a priori knowledge of their target audience. They are typically used to transmit real-time data, such as information on sensors or control commands and can be implemented using various types of messages, such as strings, numbers or custom data structures.

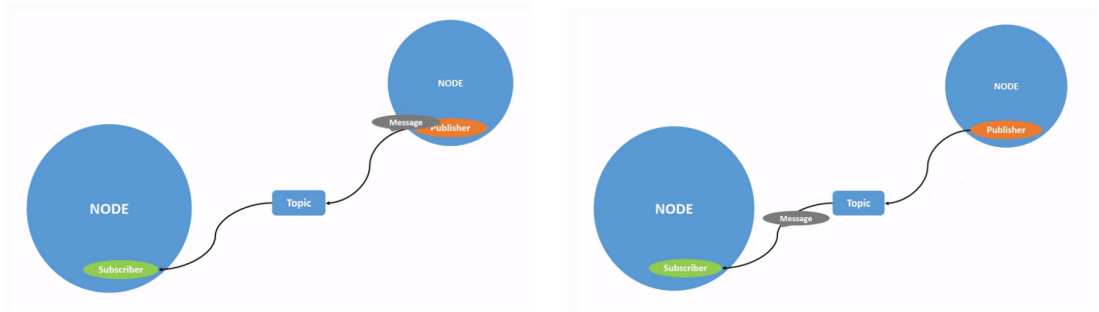


Figure 3.1. Example of Publisher-Subscriber [2]

- Service: The services provide a synchronous and bi-directional communication mechanism, implementing a form of Remote Procedure Call (RPC). This mode of interaction allows a client node to request a specific operation from a server node. The server node, after processing the request, returns a response to the client. This mechanism is particularly useful for operations that require closer interaction between nodes, such as controlling devices or accessing shared data.

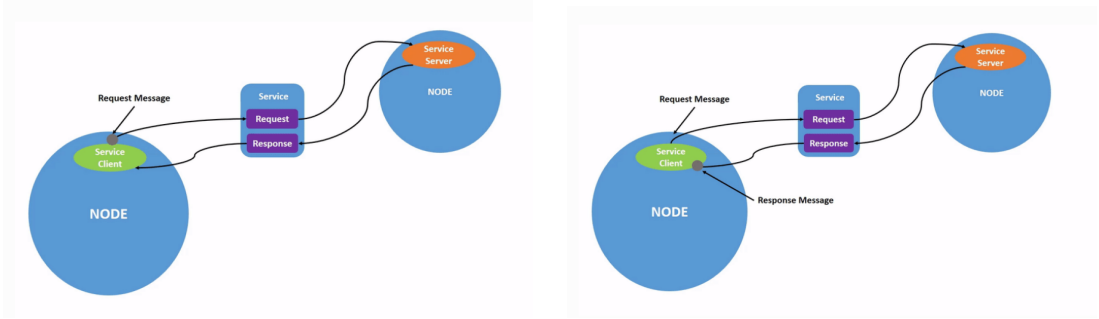


Figure 3.2. Example of Service-Client [2]

- Action: Actions are a powerful asynchronous communication mechanism for performing complex operations and monitoring their progress in real time. An action is defined by a goal, which specifies the end goal, and feedback, which provides information on the current status of execution. A client node can send a request to initiate an action and receive periodic feedback on the status of execution, allowing informed decisions to be made based on progress. Actions also support deletion and preemption mechanisms, making them particularly suitable for handling complex sequences of actions and for responding to unforeseen events.

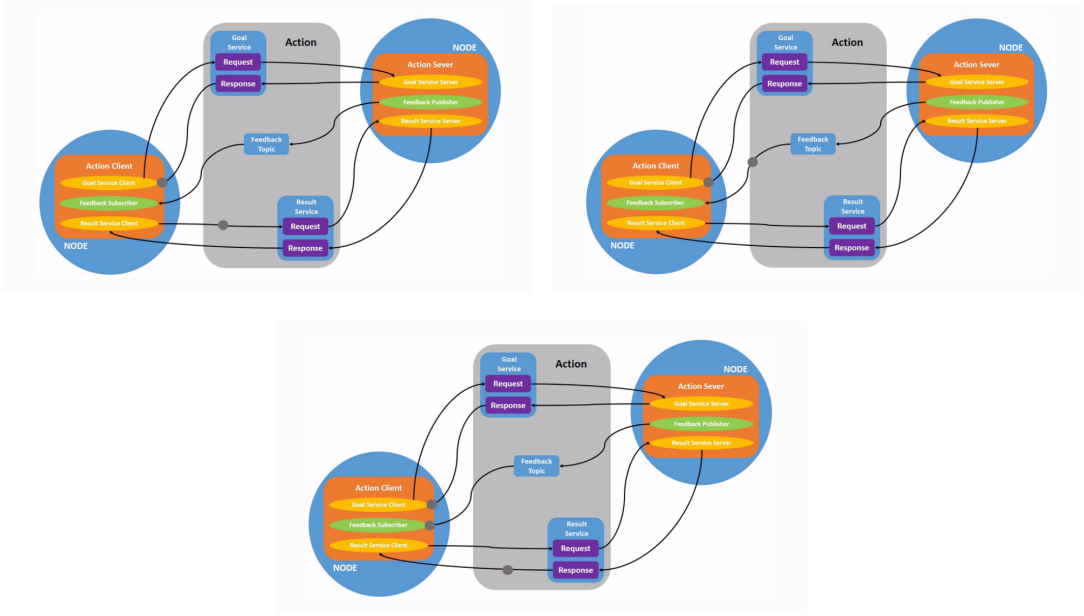


Figure 3.3. Example of Action with feedback [2]

3.2 ROS vs ROS2

ROS, while revolutionising the landscape of robotics research, had inherent limitations that inhibited its large-scale adoption in industry. The lack of determinism, limited scalability, security and real-time deficiencies, as well as hardware support that was not always optimal made ROS unsuitable to meet the stringent requirements required by industrial applications. ROS2, developed as an evolution of ROS, is designed to overcome these limitations by introducing significant improvements in service quality, error handling, security and real-time support.

- Security: It should be safe, with suitable encryption where necessary.
- Embedded Systems: ROS2 should be able to run on embedded platforms.
- Diverse networks: Robots must be able to run and communicate across enormous networks, ranging from LAN to multi-satellite hops, to accommodate the wide range of situations in which they may operate and communicate.
- Real-time computing: Need to be able to execute calculation in realtime reliably since runtime efficiency is critical in robotics.
- Product readiness: Need must conform to applicable safety/industrial standards so that it is ready for market. [5]

These innovations make ROS2 a more robust and flexible platform, capable of dealing with the complexities of industrial environments and accelerating the adoption of robotics in ever wider sectors.

The decision to use ROS2 over its predecessor was driven by the dynamic evolution of the robotics community. A gradual transition to ROS2 over the past few years has resulted in a notable rise in the user base as well as the creation of new tools and packages. Because of its increasing popularity, ROS2 has developed into a very vibrant ecosystem with lots of resources, guides, and community support. Furthermore, by staying up to date with emerging technologies and market trends, ROS2 maintains its position as a cutting-edge platform for the creation of robotic applications and gives programmers the chance to collaborate with other specialists to solve challenges.

3.2.1 Security in the network protocol

The choice of Data Distribution Service (DDS) as the underlying communication protocol in ROS2 represents a significant shift in the robotics landscape. Indeed, DDS, thanks to its flexibility in the configuration of Quality of Service (QoS), its inherent scalability and robustness, offers a solid foundation for the realization of distributed and reliable robotic systems, overcoming the limitations of the TCP protocol used in ROS, which was less suitable for managing the complexities typical of modern robotic environments.

3.2.2 Communication between nodes

In ROS, the ROS Master played a central role as a naming and registration service, acting as an intermediary for communication between nodes. Its disruption caused the isolation of new nodes, compromising scalability and resilience of the system. ROS2, instead, has eliminated the ROS Master, adopting DDS technology, thus bringing a decentralization of communication. This choice has allowed to achieve a peer-to-peer communication between the nodes, significantly improving fault tolerance and system scalability.

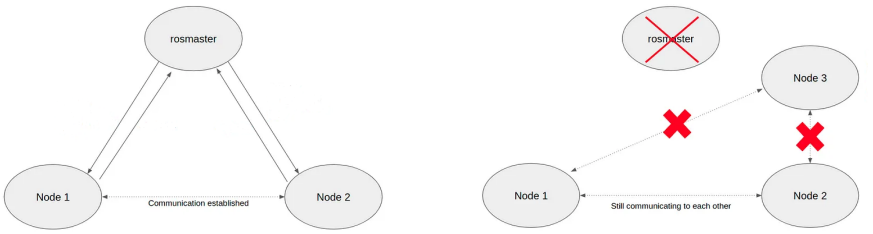


Figure 3.4. Communication between nodes in ROS (left) and ROS2 (right) [5]

3.2.3 Navigation

move base

ROS's navigation stack is a collection of software packages that allow mobile robots to travel autonomously in their environment. ROS relies on the *move base* package as its central control unit, which coordinates duties such as path planning, obstacle avoidance, and velocity control.

ROS2 introduces Nav2, which provides greater flexibility through the use of behavior trees (BT) and a modular architecture. Nav2 enables more flexibility of robot behavior by enabling a diverse set of planning and control algorithms.

Both types require sensor data, such as LiDAR or cameras, to create maps of the surroundings and determine safe courses.

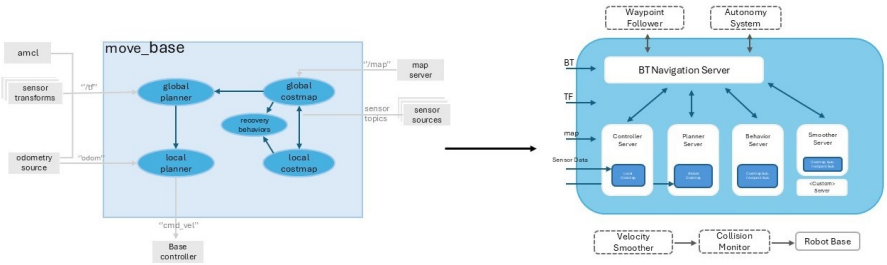


Figure 3.5. Navigation in ROS vs ROS2

Component	Description	ROS	ROS2
Planning	Path plan and obstacle avoidance	Global/local plans	Task-specific servers
Control	Robot motion control	Base controller	Task-specific servers
Lifecycle	Management of component states	Implicit	Lifecycle manager
Modularity	System structure	Less modular	Highly modular
Real-Time	Performance characteristics	Focus on real-time	Enhanced real-time

Table 3.1. Comparison of ROS and ROS2 Navigation Stacks

gmapping

For many years, the gmapping package has been a cornerstone of SLAM in ROS. However, it has several inherent limitations, particularly in terms of accuracy in dynamic environments and computational complexity. Furthermore, with the introduction of ROS2 and more advanced SLAM algorithms, it has become increasingly common to seek alternatives to gmapping. There are numerous SLAM packages available for ROS2, including:

- The *slam toolbox* package, more detailed in the next paragraph, has become a standard in ROS2. It provides a strong and flexible SLAM implementation, with support for various sensor types and the ability to create 2D and 3D maps.

- Google-developed *Cartographer* is known for its exceptional performance in complex and dynamic environments. It is especially suitable for mobile robots that travel at high speeds.

- *Hector* SLAM, designed specifically for flying robots, is an excellent choice for drones and other aircraft.

3.2.4 Slam toolbox

ROS and ROS2 are two generations of frameworks for developing robotic software. Despite sharing the goal of providing a flexible and modular environment for the development of robotic applications, there are significant differences, particularly in the context of SLAM.

ROS2, as previously described, employs a peer-to-peer approach, eliminating the need for a master node and achieving generally lower latency compared to ROS, improving the reactivity of the SLAM system. Furthermore, the reliability of SLAM is also improved with the introduction of QoS support ensuring message delivery. ROS2 can manage more data than ROS, allowing for more efficient processing of sensor data and producing more detailed maps. The tools used in both frameworks, such as RViz for visualization, rqt for debugging, and rosbag for data recording and replication, are similar, although ROS2 has a more modern and intuitive user interface. In addition, ROS2 introduces new features, such as support for system-level debugging, which helps solve problems.

3.2.5 Controller Manager

The Controller Manager is a key component of both ROS and ROS2, organizing controller lifecycles, interacting with hardware components, and delivering critical services to the larger ROS ecosystem. While the core role is identical between the two frameworks, major architectural and functional variations emerge. ROS2 features a more modular and adaptable design, particularly through connection with the *ros*2 *control* framework. This versatility improves customization and adaptability, allowing for smooth integration with a broader range of robotics systems. Furthermore, ROS2 emphasizes real-time performance and determinism. ROS2 is ideal for applications that require precise and timely control because it supports real-time scheduling and reduces jitter in the control loop. ROS2 significantly improves hardware component lifecycle management, giving users finer-grained control over controller states and allowing for seamless connection with other framework components. These changes, together with the Controller Manager's robust integration with the *ros*2 *control* framework, make it easier to create and deploy robotic applications.

3.3 ROS Workspace

In ROS, workspaces serve as the fundamental architecture for organizing and managing the development of software packages. Imagine them as modular containers that house all the components necessary to implement a specific robotic functionality: source code, configuration files, and various resources. This block-based structure offers several advantages, including

- Modularity and reusability: each package within a workspace can be developed and tested independently, facilitating the management of complex projects and promoting the reuse of code in different contexts.
- Isolation: workspaces allow the creation of isolated development environments, preventing conflicts between different versions of packages or incompatible dependencies.
- Efficiency: thanks to tools like CMake and catkin, compilation and dependency management are automated, significantly accelerating development times.
- Collaboration: workspaces facilitate collaboration among development teams, allowing each member to work on specific packages without interfering with the work of others.

3.4 RViz

RViz2 is a powerful open source 3D visualization platform, designed to give robotics developers an intuitive way to view and interact with their projects. From 3D representations of robots and their sensors to map and route visualizations, RViz2 provides a wide range of tools for creating custom user interfaces. Thanks to its flexibility, developers can adapt RViz2 to the specific needs of each application, from simulation to real robot teleoperation.

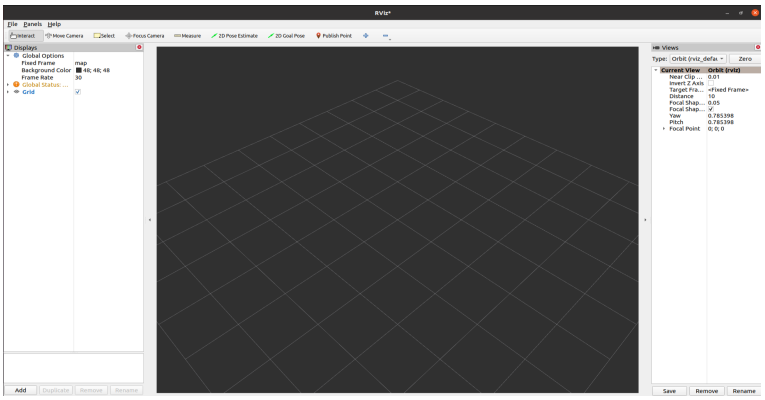


Figure 3.6. RViz2 basic interface

RViz2 marks a substantial architectural shift from its predecessor. The modular architecture, which includes smaller, more independent components, improves scalability and maintainability. This granular framework allows for the production of bespoke visualizations and plugins, which tailor the tool to specific study or industrial purposes. Furthermore, RViz2's user interface has been simplified to give a more natural experience, which speeds up the process of creating and managing complicated visualizations. The inclusion of tf2, ROS2's transformation library, ensures a robust and flexible handling of spatial interactions across different coordinate frames. This is especially useful for multirobot systems and complex situations. Finally, RViz2's flexible plugin architecture allows developers to construct highly customized visualizations and tools, expanding the platform's capabilities beyond conventional offerings.

3.4.1 TF2

Transformation library (tf2) is a key component of ROS2 which manages the transformations between the different coordinate systems used by a robot. This framework allows for precise and efficient conversion of measurements from sensors and robot components, greatly simplifying the programming of complex robotic applications. By organizing the relationships between the different reference systems of a robot in a tree structure, it allows to efficiently calculate the transformations between any frame pair and thanks to its buffering system and the library of transformations, tf2 ensures robust and efficient management of spatial information. The accuracy of transformations is crucial for the correct execution of many robotic operations, such as object tracking, navigation and manipulation.

Actually, tf2 acts as a link between the different components of a robot system, unifying information from different sensors and allowing the robot to build a coherent representation of the surrounding world. In addition, tf2 supports a wide range of transformations including rotations, translations and scaling, and can be used to manage both static and dynamic coordinate systems.

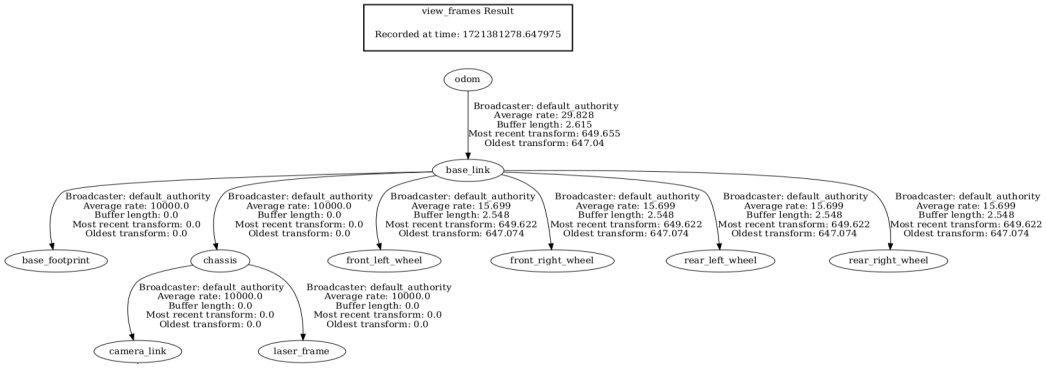


Figure 3.7.

3.4.2 Robot model

Using the URDF (Unified Robot Description Format) library, this component provides a complete and flexible three-dimensional visual representation of the robot. The URDF describes the geometric structure, movements and physical characteristics of the robot, and also supports the visualization of sensors and other components, providing a complete picture of the robotic system. You can view the model from different angles, explore its movements and interact with it intuitively. It is possible to customize the display and interaction with the robot model:

- Visual enabled: allows to activate or deactivate the graphical display of the 3D model of the robot. This feature is useful for analyzing the geometry of the robot, checking the correctness of the assembly and identifying any errors in the modeling.

- Collision enabled: enables collision control between the robot and obstacles in the simulated environment. This feature is essential for movement planning and to prevent damage to the robot or its surroundings.

- Links: provides a visual representation of the robot's structure, showing the relationship between the different components. This representation is useful for understanding the kinematics of the robot and for identifying the points of articulation.

Figure 3.8. Collision on the left, visual enabled on the right

3.4.3 Grid map

The 'grid map' component provides a discrete representation of the surrounding environment, divided into a grid of cells. Each cell indicates the state of occupation of the corresponding space, allowing the robot to build an internal map of the environment and plan its movements safely and efficiently.

The map can be viewed and edited interactively, allowing the user to annotate obstacles, define regions of interest and customize the view. The 'grid map' supports different file formats and integrates seamlessly with other navigation system components, such as the locator and path planner.

3.5 Gazebo

Gazebo is a versatile simulation platform capable of modeling robotic and sensor applications in both indoor and outdoor 3D environments. Its architecture is based on a topic-based Publish/Subscribe communication mechanism, allowing processes to exchange data efficiently.

Each simulated object in Gazebo can have multiple controllers attached, responsible for managing the object's behavior and generating its state. These controllers publish their data to shared memory using Gazebo's interfaces (Ifaces). This allows for interprocess communication between the robot control software and Gazebo, regardless of the programming language or hardware platform.

Gazebo leverages high-performance rigid body physics engines such as Open Dynamics Engine (ODE), Bullet, Simbody, and DART to simulate the dynamic behavior of objects in the environment. The 3D graphics rendering is handled by the Object-Oriented Graphics Rendering Engine (OGRE), providing a visually appealing and realistic simulation experience. [25]

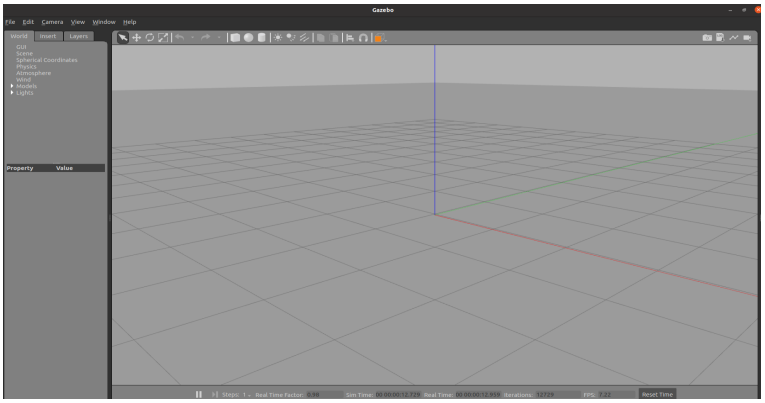


Figure 3.9. Gazebo interface

Chapter 4

Active SLAM algorithm

4.1 SLAM Algoritm

Simultaneous localization and mapping (SLAM) algorithm, first proposed by Smith in 1986 [18], is used in an extensive range of applications, especially in the domain of robotics and augmented reality.

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in robotics and computer vision of constructing a detailed map of an unknown environment while simultaneously determining the location of a mobile agent within that environment. Despite the seemingly circular nature of the problem, a number of algorithms have been developed that, by exploiting filtering and optimization techniques, allow the problem to be solved efficiently. SLAM algorithms are based on sensors such as lidar, radar, cameras and IMUs, which provide information about the surrounding environment. Information from these sensors is fused to create an accurate representation of the map and location of the robot. Popular SLAM algorithms include the Particle Filter, Kalman Filter and GraphSLAM, each with its own strengths and weaknesses. The applications of SLAM are vast and range from mobile robotics to augmented reality.

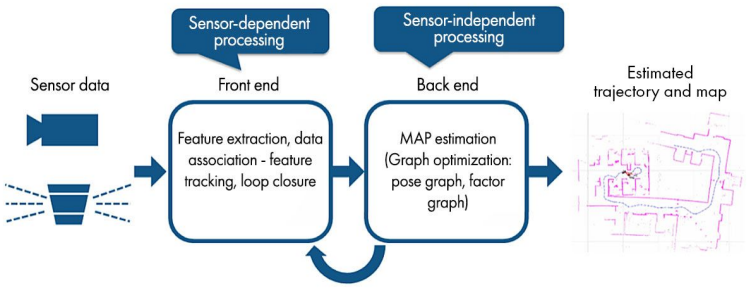


Figure 4.1. Processing flow SLAM [20]

4.1.1 Slam paradigms

There are three different SLAM paradigms, from which most others are derived.

- Kalman Filter-based approaches: This family of SLAM algorithms estimates the robot's location and the locations of a set of environmental features using a single state vector. The uncertainty in these estimates, which includes correlations between the vehicle and feature state estimates, is represented by an associated error covariance matrix. [13].
- Particle Filters: The techniques based on particle filters differ from Kalman filters in that they can effectively handle the localization problem without addressing non-Gaussian models or system non-linearity. [11].
- Graph-Based Optimization Techniques: The Full SLAM problem is resolved by the Graph-Based optimization techniques (of which GraphSLAM is the most widely used). [10].

Murphy [3] was the first to use the Rao-Blackwellized particle filter (RBPF) to tackle the SLAM problem in dynamic interior situations, in 2000. While RBPF provides a powerful framework for dealing with nonlinear and non-Gaussian state spaces, its computational demands can be significant due to the huge number of particles necessary for accurate mapping. This has resulted in substantial study into decreasing the computing complexity of particle filters. A common approach is the Sequential Importance Resampling (SIR) filter, which involves four primary steps:

1. Prediction: Particles are generated based on the predicted state transition model.
2. Correction: Importance weights are assigned to each particle based on the likelihood of the observation given the particle's predicted state.
3. Resampling: Particles are resampled with probabilities proportional to their weights to avoid degeneracy.
4. Map Estimation: For each resampled particle, a corresponding map estimate is computed.

However, as the trajectory length increases, the SIR algorithm may become computationally inefficient due to the necessity to re-evaluate particle weights from the start at each time step. To address this, researchers investigated strategies for generating recursive formulas for importance weights, thereby lowering the computing overhead.

In literature [29],Graph-based optimization algorithms, such as the widely-used Karto SLAM and Google's Cartographer, have become a cornerstone of Simultaneous Localization and Mapping (SLAM) problems. These algorithms represent the SLAM problem as a graph, where each node corresponds to a robot pose and each edge represents a constraint or relationship between two poses.

In the context of graph-based SLAM, the construction of this graph typically involves two primary steps.

Front-end: This stage focuses on data acquisition and preprocessing. Sensor data, such as odometry measurements or visual information, is collected and integrated to establish initial estimates of robot poses and their interrelationships.

Back-end: The back-end optimization process refines the initial pose estimates by minimizing the discrepancies between the estimated poses and the constraints represented by the graph edges. This is achieved through iterative optimization techniques that adjust the node positions to satisfy the edge constraints as closely as possible. By formulating the SLAM problem as a graph, these algorithms can effectively handle complex environments, incorporate various sensor modalities, and provide robust solutions for localization and mapping tasks.

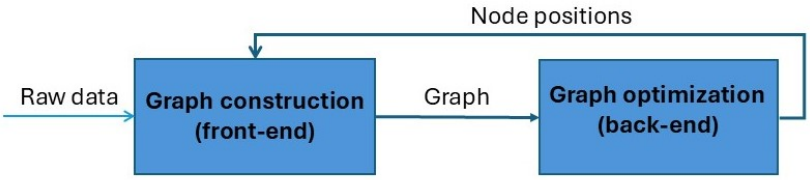


Figure 4.2. The process of graph optimization

4.1.2 Slam methods

For the front-end component, there are several SLAM methods including visual SLAM, lidar and multi-sensor.

- Visual SLAM is a fundamentally important technique in robotics and computer vision, which aims to simultaneously determine the location of a mobile agent within an unknown environment and construct a visual map of that environment. By exploiting information that can be derived from sequences of images acquired by different types of cameras, such as stereocameras, RGB-D cameras, and monocular cameras, the vSLAM enables detailed three-dimensional representations of the surrounding environments.

  A crucial aspect of vSLAM is depth estimation. While stereo and RGB-D cameras provide depth information directly, monocular cameras require the use of more sophisticated techniques, such as feature detection and the use of a priori geometric models, or fusion with data from other sensors, such as inertial measurement units (IMUs).

- SLAM lidar uses laser sensors to construct detailed maps of an environment. Laser sensors emit pulses of light and measure the time it takes for light to reflect off objects and return to the sensor, thus providing precise measurements of distances. Point clouds generated by LIDARs are ideal for constructing 3-D maps because of their high density and accuracy. There are several point cloud registration algorithms, such as ICP, NDT and LOAM, which allow estimating the relative transformation between successive scans and constructing coherent maps of the environment.

- Multi-sensor SLAM is a natural evolution of SLAM algorithms, exploiting the complementarity of different sensors to achieve a more robust and accurate estimation of the location and map of the environment. By combining sensors such as cameras, LIDAR, IMU and GPS, it is possible to overcome the inherent limitations of each individual sensor and obtain a more complete and detailed representation of the surrounding environment.The fusion of data from different sensors requires accurate sensor calibration and precise time synchronization. In addition, it is necessary to address the uncertainty associated with sensor measurements, which can be handled using Bayesian filtering and estimation techniques.

4.2 Online and Full SLAM

The SLAM problem can be formulated in two main ways: online and full slam.

In the first approach, the robot's current state and the map are estimated incrementally, based on the most recent sensor measurements and control inputs. Online SLAM algorithms, such as the Kalman Filter and Particle Filter, are computationally efficient but may suffer from drift over time. The Online SLAM problem can be formulated as follows:

$$bel(x_t, m) = p(x_t, m|z_{1:t}, u_{1:t}) \propto p(z_t|x_t, m) \cdot p(x_t|x_{t-1}, u_t) \cdot bel(x_{t-1}, m) \tag{4.1}$$

Where x_t is the current robot pose, m the map, $z_{1:t}$ the sequence of sensor observations and $u_{1:t}$ the sequence of control inputs.

While in the second approach, the entire trajectory of the robot and the complete map are estimated jointly. Full SLAM algorithms, such as GraphSLAM, can provide more accurate results but are computationally more demanding. The Full SLAM problem can be formulated as follows:

$$bel(x_{0:T}, m) = p(x_{0:T}, m|z_{0:T}, u_{0:T}) = p(m|x_{0:T}, z_{0:T}) \cdot p(x_{0:T}|z_{0:T}, u_{0:T}) \tag{4.2}$$

Where $x_{0:T}$ is the robot's trajectory, m the map, $z_{0:T}$ the sequence of sensor observations and $u_{0:T}$ the sequence of control inputs.

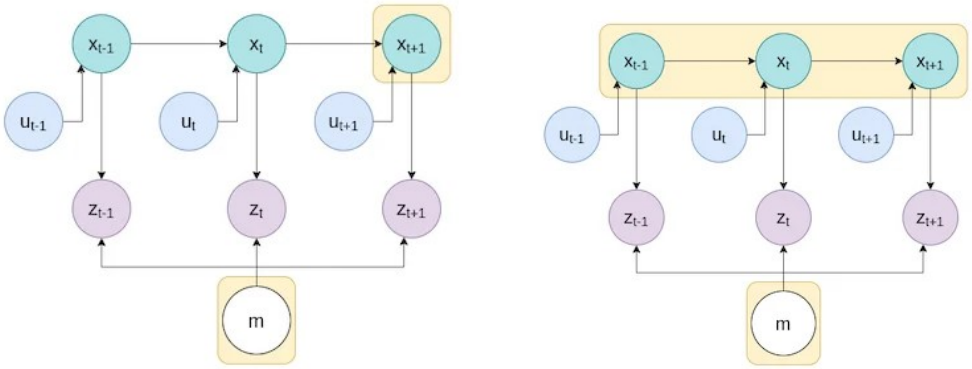


Figure 4.3. Online SLAM on the left, full SLAM on the right [16]

4.3 Passive and Active SLAM

Within the field of SLAM, a fundamental distinction lies between passive and active approaches to data acquisition. In passive SLAM, the robot merely perceives its surroundings through available sensors without actively influencing the information acquisition process. In contrast, active SLAM involves the robot interacting with its environment, such as through planned movements or emitting signals, to improve the quality of acquired information and reduce uncertainties in position and map estimation. Active SLAM offers numerous advantages, including more efficient exploration of unknown environments, resolution of ambiguities in observations, and increased robustness of the localization system. However, it requires careful planning of the robot's actions and can lead to increased computational complexity. The choice between passive and active SLAM depends on the specific requirements of the application and the available computational resources.

A critical aspect of active SLAM lies in robot motion planning. Through motion planning algorithms, the robot determines the optimal sequence of actions to acquire the most relevant information about the environment. These actions may include rotational or translational movements, emitting acoustic or light signals, or utilizing active sensors such as touch. The choice of planning strategy depends on various factors, including the environment's complexity, sensor characteristics, and mission objectives. For instance, in dynamic environments, planning must adapt in real-time to accommodate unexpected changes in environmental conditions. Moreover, active SLAM can be combined with machine learning techniques to further enhance robot performance, allowing it to learn environmental models and adapt its exploration strategies based on experience.

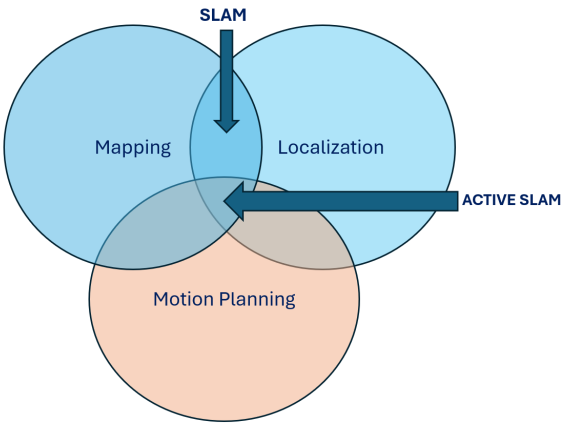


Figure 4.4. SLAM

4.3.1 Active SLAM

As illustrated in Figure 4.5, a typical SLAM system comprises two primary components: the front-end and the back-end. The front-end module uses sensor data from Light Detection and Ranging (LiDAR) sensors, cameras, and Inertial Measurement Units (IMUs) to extract features, create data associations, and estimate the robot's relative posture in relation to its surroundings. Techniques such as Iterative Closest Point (ICP) and loop closure are used to align sensor data and recognize when the robot returns to previously visited locations. The backend module is in charge of optimizing the global map and robot poses. It combines the front-end's relative pose estimates into a pose-graph representation, with nodes representing robot poses and edges representing their relative restrictions. The back-end module refines the estimated state vector, comprising robot postures and landmark positions, by solving a nonlinear optimization problem using iterative methods such as Gauss-Newton or Levenberg-Marquardt, resulting in a globally consistent and accurate map.

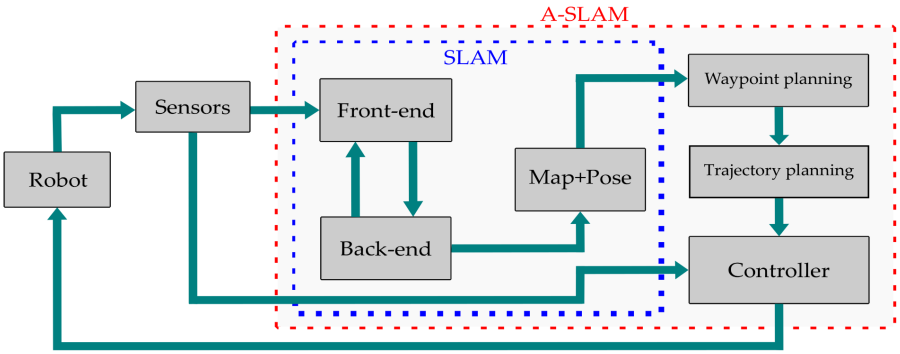


Figure 4.5. SLAM and A-SLAM architecture [1]

Active SLAM (A-SLAM) expands upon classic SLAM by combining trajectory planning and control modules to enable autonomous robot navigation. The purpose of A-SLAM is to optimize the robot's trajectory to reduce uncertainty in map representation and localization. To maximize map quality, it is necessary to balance exploration (covering new areas) and exploitation (revisiting locations to close loops). A-SLAM can autonomously lead the robot towards its goal by combining approaches from information theory, control theory, and reinforcement learning, planning waypoints and producing appropriate trajectories while taking into account environmental restrictions and impediments.

To handle the computational issues associated with Active SLAM (A-SLAM), it is frequently divided into three basic submodules, as shown in Figure 4.6. Initially, the robot detects prospective goal places within its environment, taking into account both exploration and exploitation objectives. There are numerous sorts of maps that the robot can employ to represent its seen environment:

- Topological Maps: These maps provide a simplified, graph-based representation of the environment, emphasizing connectivity and topological interactions across regions.
- Metric maps provide a more detailed representation of the environment by employing information points (landmarks) or full 3D point clouds to collect geometric and spatial information.
- Semantic Maps: Unlike topological and metric maps, semantic maps name and categorize items in the environment, providing information about the nature and meaning of certain aspects (for example, impediments, free space, specific objects).

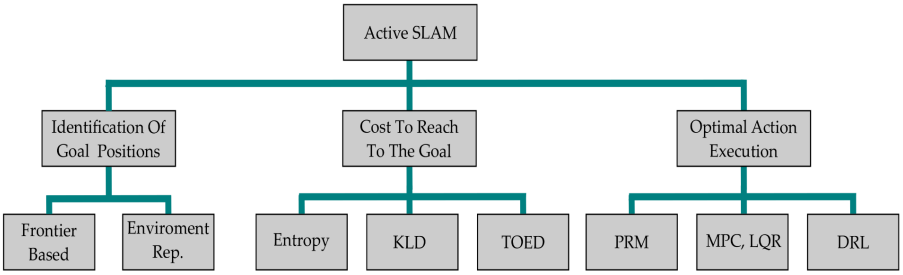


Figure 4.6. A-SLAM Components [1]

After creating a map using one or more of these methods, the robot can seek for appropriate target or goal locations to explore, taking into account aspects such as map ambiguity, undiscovered regions, and the possible benefits of revisiting previously visited places.

4.3.2 Cartographer

So, as already mentioned, Active SLAM is a sophisticated technique to mobile robot navigation that involves developing a map of the environment while also locating the robot inside it. ROS provides several packages for this purpose. It is possible choose between 2D ($gmapping, hectorslam, cartographer$) and 3D ($rgbdslam, lsdslam, rtabmap$), but often 2D SLAM is enough for ground-based robots to navigate around their surroundings. With real-time indoor surroundings in mind, Google Cartographer is a powerful SLAM system. It creates intricate 2D grid maps that facilitate precise navigation and localization. Cartographer uses a two-tiered approach: local SLAM and global SLAM. Local SLAM focuses on short-term mapping and localization by aligning sequential laser scans and estimating the robot's pose inside a local submap. Global SLAM ensures long-term consistency by identifying loop closures and optimizing the entire map.

Cartographer uses a sliding window approach to maximize computing efficiency, matching just the most current scans. This lowers computing load while retaining reasonable accuracy. Regular pose optimization actions are carried out to improve the predicted robot pose and reduce error buildup. Cartographer relies heavily on the detection of loop closures. When a submap is completed, it is compared to the other submaps on the global map. If a sufficient overlap is discovered, a loop closure constraint is imposed, compelling the optimizer to resolve any discrepancies between the two submaps. This helps to fix accumulated inaccuracies and improves overall map accuracy.

Cartographer's modular design and efficient algorithms make it an excellent candidate for a variety of robotic applications, including autonomous navigation, exploration, and mapping.

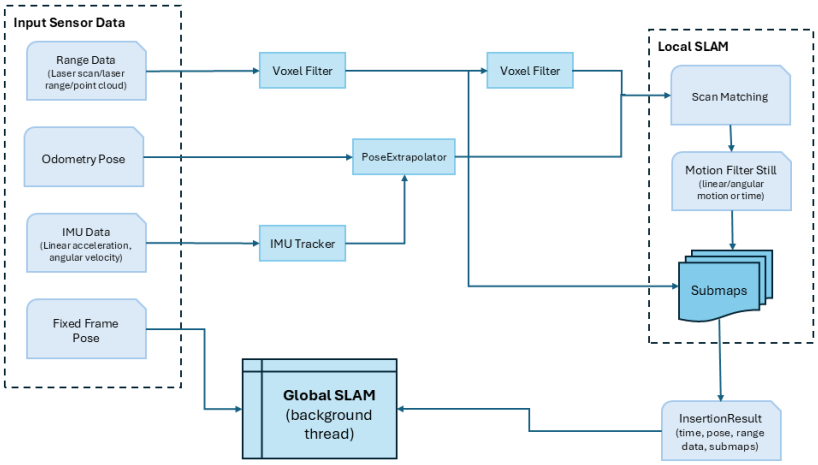


Figure 4.7. Overview of Cartographer

Chapter 5

Software in-the-Loop

This chapter will detail the implementation of the robot's autonomous navigation capabilities and accompanying tools. To begin, let's clarify the concept of autonomous navigation.

Autonomous navigation is the capability of a system to navigate in unfamiliar, unstructured, and unpredictable environments without human intervention. This requires the system to acquire information about its surroundings using sensors and make decisions based on that information. To ensure accurate perception and reduce errors, it is often beneficial to employ a different sensors, combining data from various sensors to obtain a more comprehensive understanding of the environment.

A key component of this system is the Simultaneous Localization and Mapping (SLAM) algorithm, which enables the robot to construct a map of its environment while simultaneously determining its own location. By employing SLAM, the robot can explore unknown spaces, build a map, and subsequently navigate autonomously.

The Robot Operating System (ROS), coupled with the main packages, provides a robust framework for implementing these functionalities. The RViz visualization tool is invaluable for real-time monitoring and debugging, allowing operators to observe the robot's behavior and map construction process. The system was initially developed and simulated in Gazebo to validate its functionality under controlled conditions.

5.1 Conceptual design

This robot is a differential-powered robot, which means that the robot has two drive wheels, one on the left and one on the right. These two wheels control all the movement, and all the other wheels are there just to keep it stable and can turn freely in all directions. The machines with differentiated propulsion can be of various shapes and sizes, we will display the measurements and model of the rover mini, as described in Chapter 6. Since it is a mobile robot, we will use some of the conventions established in the REPs ROS (i.e. standards):

- The main coordinate frame for the robot will be called *baselink* (REP 105 - Coordinate Frames for Mobile Platforms)
- The orientation of this coordinate frame will be X-forward, Y-left, Z-up (REP 103 - Standard units and coordination conventions)

To visualize the robot within a simulation environment, we must first define its configuration using URDF (Unified Robot Description Format). A set of URDF files, each describing a specific component of the robot, is combined into a single, comprehensive URDF file using the *xacro* tool.

This combined URDF file is then provided to the *robot state publisher* node, which broadcasts the robot's description on the *robot description* topic. Additionally, the node publishes the necessary transformations between the robot's various frames of reference.

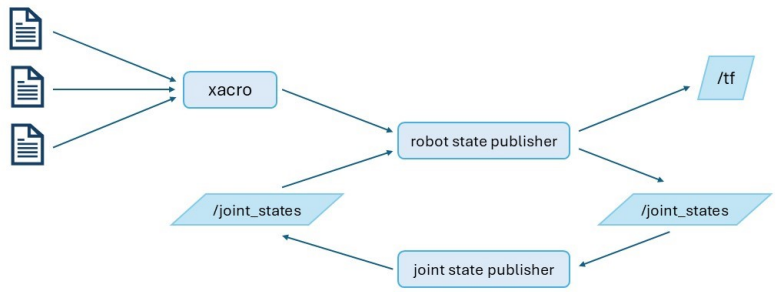


Figure 5.1. URDF file

5.1.1 URDF

In the Robot Operating System (ROS) ecosystem, it's a common convention to designate the primary reference frame of a mobile robot as *base link*. For differential drive robots, the halfway between the two drive wheels is a more sensible location for *base link*, despite the natural preference to place it at the geometric center of the robot's chassis. Because the robot's rotation is centered at this point, kinematic modeling is made simpler. The remaining parts of the robot can be precisely positioned in relation to *base link* by setting it up at this point. Thus, we construct an empty link named *base link* as the first step in building the robot's URDF model, continuing with the *base footprint link*.

```
1     <!-- BASE LINK -->
2
3     <link name="base_link">
4     </link>
5
6     <!-- BASE_FOOTPRINT LINK -->
7
8     <joint name="base_footprint_joint" type="fixed">
9         <parent link="base_link"/>
10        <child link="base_footprint"/>
11        <origin xyz="0 0 0" rpy="0 0 0"/>
12    </joint>
13
14    <link name="base_footprint">
15    </link>
```

```
<!-- CHASSIS LINK -->

<joint name="chassis_joint" type="fixed">
    <parent link="base_link"/>
    <child link="chassis"/>
    <origin xyz="${-wheel_offset_x} 0 ${-wheel_offset_z}"/>
</joint>

<link name="chassis">
    <visual>
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}"/>
        <geometry>
            <box size="${chassis_length} ${chassis_width} ${chassis_height}"/>
        </geometry>
        <material name="orange"/>
    </visual>
    <collision>
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}"/>
        <geometry>
            <box size="${chassis_length} ${chassis_width} ${chassis_height}"/>
        </geometry>
    </collision>
    <xacro:inertial_box mass="0.5" x="${chassis_length}" y="${chassis_width}" z="${chassis_height}">
        <origin xyz="${chassis_length/2} 0 ${chassis_height/2}" rpy="0 0 0"/>
    </xacro:inertial_box>
</link>

<gazebo reference="chassis">
    <material>Gazebo/Orange</material>
</gazebo>
```

To add the driving wheels into our robot model, we will connect them to *base ink* via continuous joints. These joints allow for rotational movement, demonstrating the wheels' capacity to spin. While we could join the wheels to the chassis, placing *base link* at the center of rotation allows for more direct connections to *base link*.

We visualize our wheels as cylinders aligned along the Y-axis (left to right). However, ROS's default cylinder orientation is Z-axis (up-down). To fix this, we will "roll" the cylinders 90 degrees ($\pi/2$ radians) around the X-axis. To keep the Z-axis pointed outward,

rotate the left wheel clockwise (negative) and the right wheel counterclockwise (positive).

In terms of the rotation axis, because our left wheel's Z-axis is facing outward, a "forward" motion equates to a counterclockwise (positive) revolution around it. As a result, the axis of rotation for the left wheel should be +1 in Z. The complete URDF elements for two wheels are listed below, and the other two will be the same with the necessary logical changes.

```
1   <!-- FRONT LEFT WHEEL LINK -->
2
3   <joint name="front_left_wheel_joint" type="continuous">
4       <parent link="base_link"/>
5       <child link="front_left_wheel"/>
6       <origin xyz="${-chassis_length/2} ${wheel_offset_y} 0"
            rpy="-${pi/2} 0 0" />
7       <axis xyz="0 0 1"/>
8   </joint>
9
10  <link name="front_left_wheel">
11      <visual>
12          <geometry>
13              <cylinder radius="${wheel_radius}" length="${
                    wheel_thickness}"/>
14          </geometry>
15          <material name="blue"/>
16      </visual>
17      <collision>
18          <geometry>
19              <sphere radius="${wheel_radius}"/>
20          </geometry>
21      </collision>
22      <xacro:inertial_cylinder mass="${wheel_mass}" length="${
            wheel_thickness}" radius="${wheel_radius}">
23          <origin xyz="0 0 0" rpy="0 0 0"/>
24      </xacro:inertial_cylinder>
25  </link>
26
27  <gazebo reference="front_left_wheel">
28      <material>Gazebo/Blue</material>
29  </gazebo>
30
31  <!-- REAR LEFT WHEEL LINK -->
32
33  <joint name="rear_left_wheel_joint" type="continuous">
34      <parent link="base_link"/>
35      <child link="rear_left_wheel"/>
```

```
36          <origin xyz="0 ${wheel_offset_y} 0" rpy="-${pi/2} 0 0"
                />
37          <axis xyz="0 0 1"/>
38      </joint>
39
40      <link name="rear_left_wheel">
41          <visual>
42              <geometry>
43                  <cylinder radius="${wheel_radius}" length="${
                        wheel_thickness}"/>
44              </geometry>
45              <material name="blue"/>
46          </visual>
47          <collision>
48              <geometry>
49                  <sphere radius="${wheel_radius}"/>
50              </geometry>
51          </collision>
52          <xacro:inertial_cylinder mass="${wheel_mass}" length="${
                wheel_thickness}" radius="${wheel_radius}">
53              <origin xyz="0 0 0" rpy="0 0 0"/>
54          </xacro:inertial_cylinder>
55      </link>
56
57      <gazebo reference="rear_left_wheel">
58          <material>Gazebo/Blue</material>
59      </gazebo>
```

5.1.2 Gazebo

The control system's primary role is to process a desired velocity command, translate it into appropriate motor commands for the motor drivers, monitor actual motor speeds, and determine the robot's true velocity. The required velocity is transmitted in the ROS framework via the */cmdvel* topic, which uses the Twist message type. This message comprises six values: linear velocities along the x, y, and z axes, as well as angular velocities around the same three axes. Only two of these variables apply to a differential drive robot: linear velocity in the x direction (forward/backward motion) and angular velocity around the z-axis (turning). The remaining four values are usually set to zero.

While accurate velocity is important, the control system frequently favors predicting the robot's position. This estimate is obtained by integrating the velocity across time with short time steps. This procedure, known as dead reckoning, results in an odometry estimate.

The Gazebo robot model is generated using the */robotdescription* parameter. The control plugin publishes joint states, which include wheel speeds. In addition, the plugin

broadcasts a transformation between the odom frame (which represents the world origin and the robot's initial position) and the *base link* frame. This transform updates other software components with the robot's current estimated position.

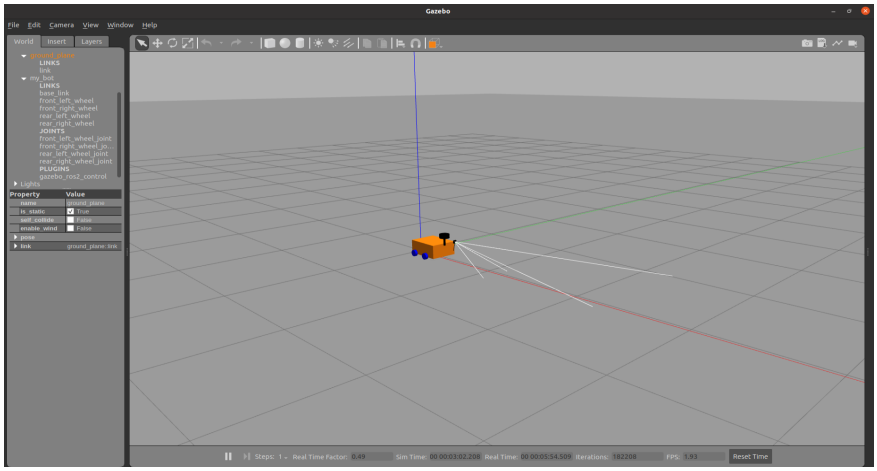


Figure 5.2. Gazebo environment

Gazebo provides the ability to create customized settings; it is possible design a globe filled with randomly placed obstacles. This will present new hurdles and complexities for the robot to manage. To accomplish this, use Gazebo's world-building capabilities to define the dimensions of the environment and insert various obstacle types such as walls, boxes, and cylinders. By randomly placing these barriers, we will create a dynamic and unexpected simulation setting.

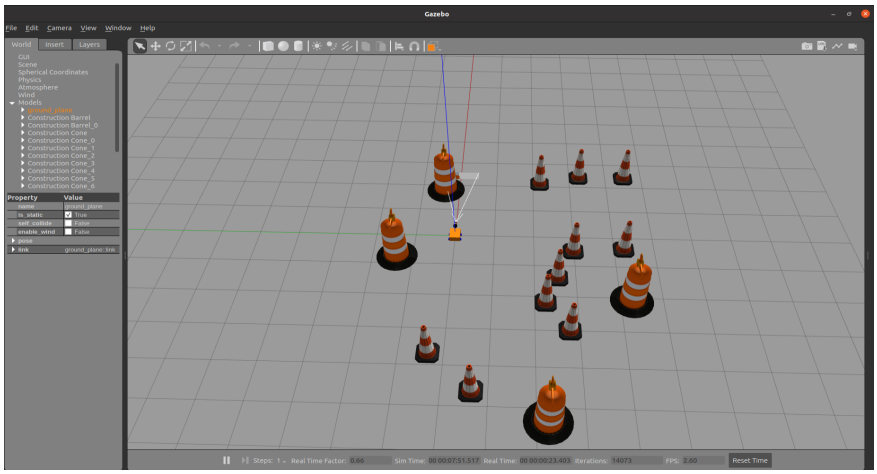


Figure 5.3. Gazebo with different obstacles created

The Gazebo 'Warehouse World' [23] simulation environment was used as the basis for designing and testing SLAM and navigation algorithms. The simulated environment's rich detail and complexity allow for the evaluation of algorithm performance under realistic operational conditions, with a variety of static and dynamic obstacles, such as pallets, shelves, and simulated agents, creating a challenging environment for robot navigation. The inclusion of multiple static and dynamic obstacles allows for testing the durability of localization and navigation algorithms in the face of uncertainties and disturbances.

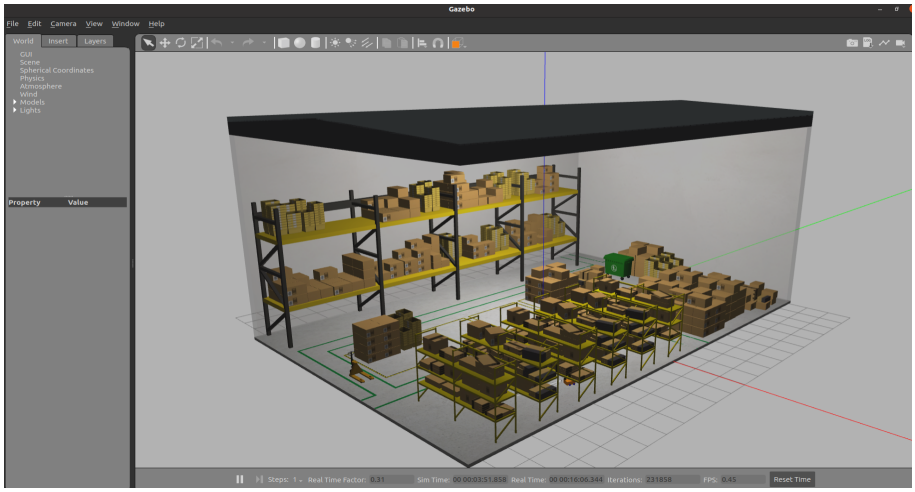


Figure 5.4. Gazebo Warehouse World

5.2 Slam toolbox

Slam Toolbox is a set of tools and capabilities for 2D SLAM built by Steve Macenski while at Simbe Robotics, maintained while at Samsung Research, and largely in his free time. An overview of how the map was generated is presented below [6]:

1. ROS Node: When the SLAM toolbox is run in synchronous mode, it creates a ROS node. This node subscribes to laser scan and odometry topics, as well as publishing map to odom transform and map.

2. Obtain odometry and LIDAR data: A laser topic callback will generate an odometry posture as well as a laser scan related to that node. These PosedScan objects are organized into a queue and processed by the algorithm.

3. Process Data: A pose graph is constructed from the queue of PosedScan objects, and odometry is optimized via laser scan matching. This pose graph is used to compute robot poses and identify loop closures. If a loop closure is discovered, the posture graph is optimized and pose estimates are revised. Pose estimates are used to generate and publish a map to Odom transform for the robot.

4. Mapping: Laser scans of each pose in the pose graph are used to create and publish a map.

In the construction of an environment map, two main approaches are opposed: feature-based SLAM and grid-based SLAM. The first one identifies distinctive points in the environment (feature) and traces their position relative to the robot. The latter, on the other hand, divides the space into a grid and assigns to each cell a probability of occupation value. For our system, we have opted for the grid map approach. This choice is motivated by the need for a detailed and intuitive representation of the environment, particularly useful for path planning and obstacle detection. Although feature-based maps offer a more compact representation, they are less accurate in areas with uniform textures and less suitable for detecting small obstacles. Also, the slam mode we will use is

- Online : Refers to the processing of live data streams as they are received, rather than relying on pre-recorded data. Ensures real-time responsiveness and enables immediate decision-making based on the latest information.

- Asynchronous : A technique that prioritizes the processing of the most recent data point, even if it means temporarily skipping older data. Helps to maintain a low latency between data acquisition and processing, preventing delays that could impact performance or decision-making.

So, thanks to the aid of this package we will see how to create a map and locate the robot inside.

```
    # ROS Parameters
odom_frame: odom
map_frame: map
base_frame: base_footprint
scan_topic: /scan
mode: mapping
```

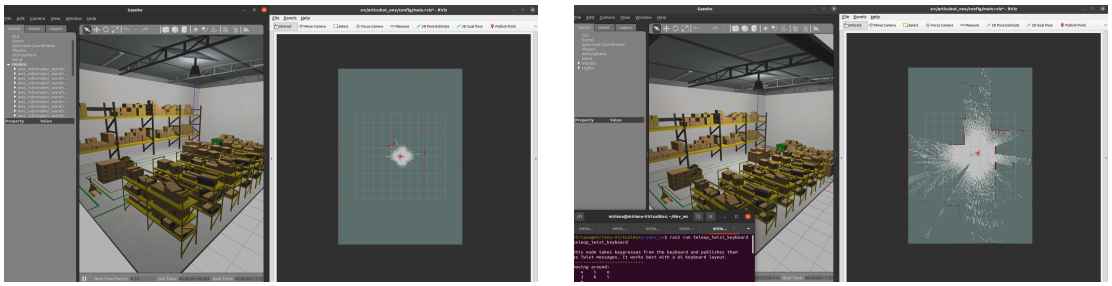
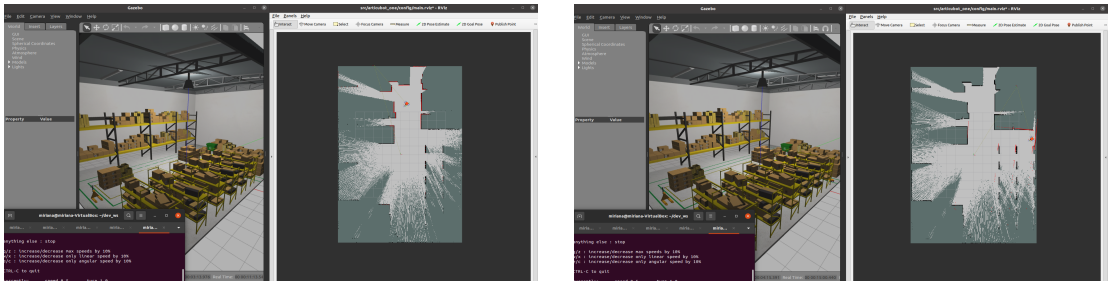


Figure 5.5. Start the mapping



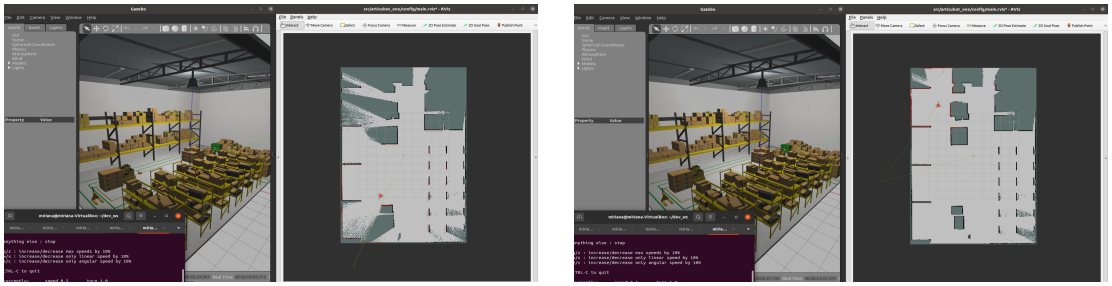


Figure 5.6. Evolution of the mapping

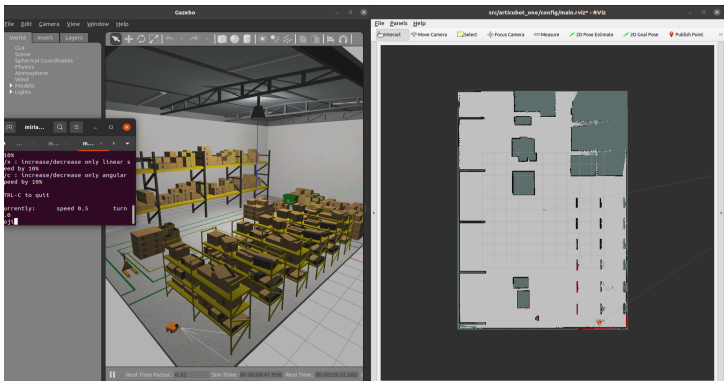


Figure 5.7. Mapping

Then changing the mapping mode from localization and adding the map with *map start at dock*, which means that it starts at the point where it started when the map was created, unlike the alternative *map start pose* i.e. in the initial pose, may begin tracking.

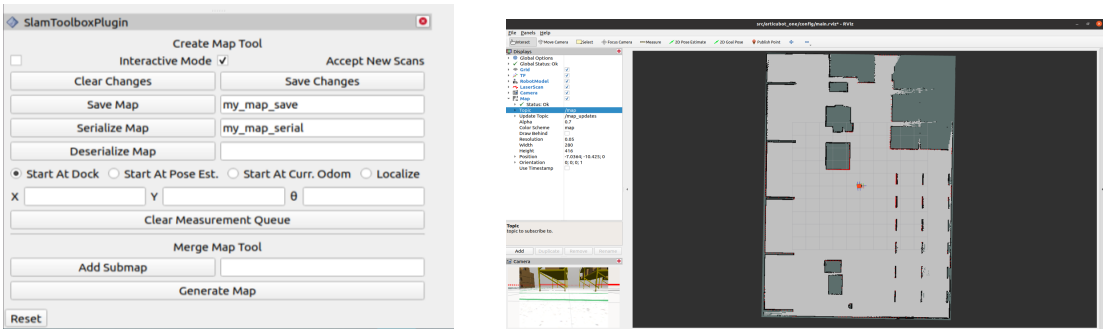


Figure 5.8. Save and adding the map

Once an accurate map of the environment has been generated using SLAM Toolbox, it can be used as a foundation for implementing more sophisticated localization algorithms that are suitable for specific scenarios. Adaptive Monte Carlo Localization (AML) is an excellent choice in this context, due to its ability to operate in dynamic environments and provide precise estimates of the robot's position, even in the presence of sensor uncertainties. By integrating the map generated by SLAM Toolbox with an AML system, it is possible to obtain a more robust and flexible navigation system. AML uses a set of particles to represent the probability distribution of the robot's position, and updates this distribution based on sensor measurements and the map of the environment. This integration allows us to leverage the advantages of both algorithms, improving the robot's ability to operate in complex environments and adapt to dynamic changes.

5.3 Nav2

Leveraging the information obtained through SLAM Toolbox, such as the map of the environment and the estimated position, it is possible to equip the robot with autonomous navigation capabilities. ROS Navigation (Nav2) provides a robust framework for implementing these functionalities, allowing the robot to autonomously plan and execute paths to specified goals. The combination of SLAM and Nav2 enables a flexible and adaptable navigation system that can handle dynamic and unforeseen situations. Path planning considers obstacles present in the map, ensuring the robot's safety during navigation. Continuous localization, provided by SLAM, allows for constant updates of the robot's position estimate and replanning of the path if necessary.

To ensure effective autonomous navigation, a robot requires an accurate perception of its surrounding environment. This involves a precise estimation of its own position, provided by SLAM algorithms, and awareness of obstacles. This obstacle information can be obtained from a static map, previously generated by SLAM, or from sensors like lidar, which provides real-time data on the presence of obstacles.

```
1         # ROS Parameters
2     odom_frame: odom
3     map_frame: map
4     base_frame: base_footprint
5     scan_topic: /scan
6     mode: localization
7
8     # if you'd like to immediately start continuing a map
9     # at a given pose or at the dock, but they are mutually
10    # exclusive, if pose is given will use pose
11    map_file_name: /home/miriana/dev_ws/map/my_map_serial
12    # map_start_pose: [0.0, 0.0, 0.0]
13    map_start_at_dock: true
```

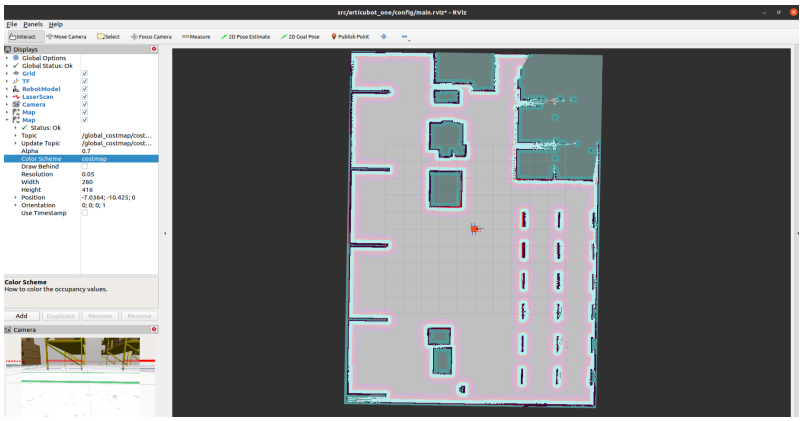


Figure 5.9. Navigation with an already created map

A static map, typically represented as a grid map, divides the space into cells, labeling each cell as occupied (with a safety margin to prevent collisions), free, or unknown. This map can be used as a basis for path planning but does not account for dynamic changes in the environment.

Alternatively, or in addition to the static map, a lidar (Light Detection and Ranging) can be used to create a dynamic map. The lidar continuously scans the environment and updates the map in real-time, allowing the robot to avoid moving obstacles. However, path planning based only on a dynamic map may be limited as it does not allow for long-term trajectory planning.

The ideal approach combines both modalities. A static map provides a global representation of the environment, allowing for long-term planning, while lidar data enables local map updates and avoidance of unexpected obstacles. All this information is integrated into a *costmap*, a map representation used for path planning, where the cost of a cell is related to the probability of collision.

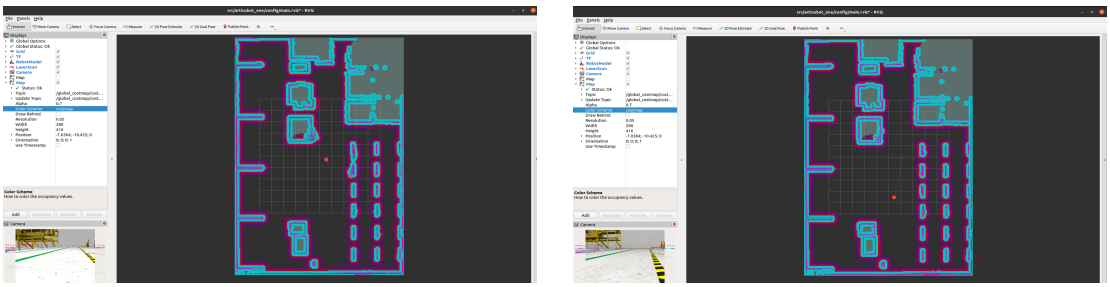


Figure 5.10. Passive SLAM

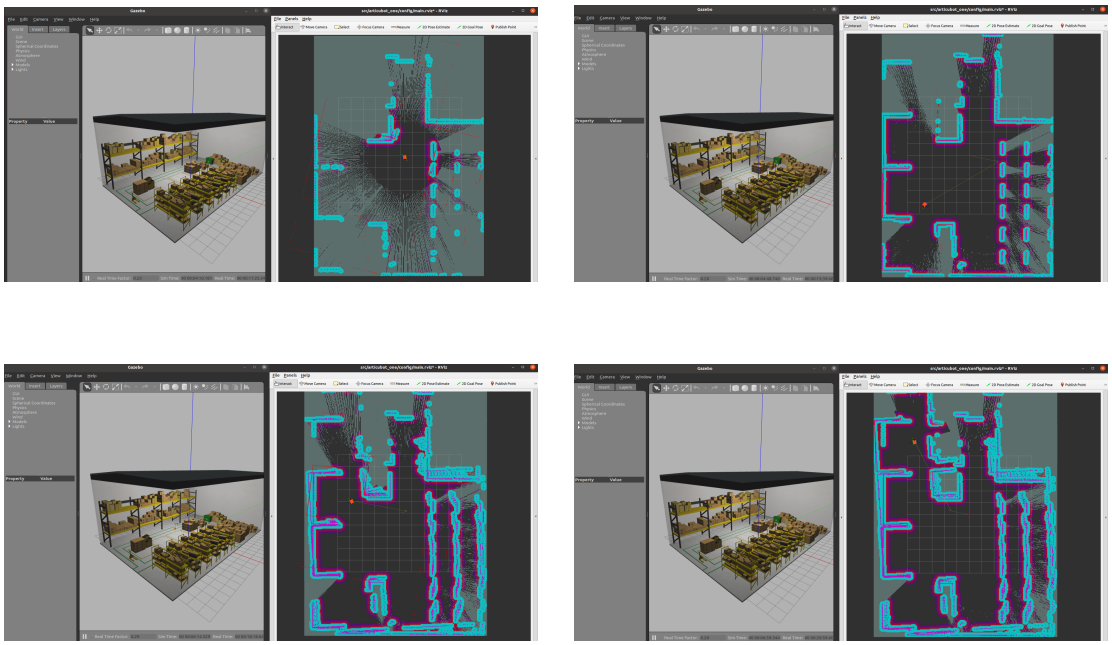


Figure 5.11. Active SLAM

5.3.1 Nav2 parameters

In order to assess the proposed navigation system's capabilities, these simulations are conducted in a precisely modeled virtual environment. The *nav*2 *params* file has been set up to optimize the parameters of the system's various components. Specifically, the VPC, Velocity Obstacle, parameters are carefully calibrated to assess the vitality of the obstacles in relation to the navigational velocity. The Dynamic Window Approach (DWB) parameters are set to adjust the robot's speed and acceleration to the environmental conditions in order to achieve a fluid and reactive navigational behavior.
The simulations have demonstrated the significance of accurately calibrating the VPC and DWB parameters. A precise calibration of the VPC parameters has made it possible to significantly reduce the number of collisions with obstacles, ensuring at this time an appropriate navigational speed. The optimization of DWB parameters has enabled the robot to precisely follow predetermined paths, even in the presence of dynamic obstacles.

In parallel, the parameters of the AMCL (Adaptive Monte Carlo Localization) algorithm have been optimized to ensure an accurate robot position inside the map. The choice of AMCL parameters is critical for the robustness of the localization system, especially in environments with challenging characteristics like variable illumination or the presence of specular surfaces. The use of an accurate rumor model and the adoption of an accurate map have the potential to obtain a reasonable position score even in complex environments. The analysis of the results has shown that, in comparison to the initial configuration, the medial localization error has decreased.

By using an iterative simulation and tuning procedure, it is possible to determine the ideal parameter combination for the navigation system, ensuring safe and effective robot behavior.

5.3.2 Rotation Shim Controller

The *nav*2 *rotation shim controller* was created to solve flaws in existing motion planning and control algorithms, such as Timed-Elastic-Band (TEB) and Dynamic Window Approach (DWA), which can display unwanted behavior, especially when dealing with pathways that need major heading changes. TEB's elastic band technique can result in excessive rotational motions, whereas DWB, while very adaptable, frequently confronts trade-offs between precise path tracking and seamless transitions to new paths. The *nav*2 *rotation shim controller* can substantially simplify tuning and improve overall performance by performing a preliminary rotation to align the robot's heading with the desired path before commencing path tracking. This method is especially useful for robots that may spin in place, such as differential-drive or omnidirectional robots, as well as for situations in which the initial path heading changes significantly from the robot's present position. By comparing the robot's current heading with the heading of the recently received path, the *nav*2 *rotation shim controller* operates. If the discrepancy exceeds a predetermined level, the controller will perform a spin in place to align the robot's heading with the path. Once alignment is accomplished within a specified tolerance, control is transferred to the principal controller plugin, providing for a more seamless transition into the route tracking task.

5.3.3 Regulated Pure Pursuit Controller

The Regulated Pure Pursuit controller is a customized version of the classic Pure Pursuit algorithm that is intended to meet the particular needs of industrial and service robots. This controller effectively reduces overshoot problems at high speeds, especially around blind corners, by controlling linear velocities based on path curvature, improving operational safety. The use of adaptive lookahead points, scaled by velocities, ensures stable behavior across a wider range of translational speeds, and the Regulated Pure Pursuit controller's superior path-following capabilities over other Pure Pursuit variations, as well as its ability to automatically slow down in proximity to obstacles, further improving safety and reliability in dynamic environments. These improvements make the Regulated Pure Pursuit controller an invaluable tool for a variety of robotic applications, especially those that require precise navigation and safe operation in complex environments. This controller stands out for being user-friendly and affordable in comparatively straightforward scenarios. Its ability to directly follow a predetermined path makes it an excellent choice for applications that require linear navigation behavior. Its intuitive configuration with limited settings makes it accessible to less experienced users as well.

5.3.4 Predictive Path Integral Controller

An important development in path planning and control algorithms is the MPPI Controller, a complex version of Model Predictive Control (MPC). MPPI is an iterative optimization technique that uses a sampling-based method to choose the best trajectories. It is a replacement for TEB and pure path tracking MPC controllers. It may be tailored to a variety of behaviors and qualities thanks to its plugin-based goal functions. Impressive performance is displayed by the MPPI controller, which can run at 50+ Hz on a low-end Intel processor (4th gen i5). Its iterative methodology, which uses the control solution from the previous time step as a foundation for later iterations, is responsible for this efficiency. MPPI efficiently chooses the best course of action by introducing random perturbations and analyzing the ensuing trajectories using critic functions. One of MPPI's main benefits is its capacity to handle non-convex and non-differentiable objective functions, which gives designers more freedom when creating robot behaviors. This capability enables more intricate and personalized navigation tactics and broadens the scope of possible applications.

The predictive path integral controller model represents the solution of the considered controllers. Their ability to forecast the future and optimize actions based on a system model enables them to obtain greater precision and robustness capabilities. However, it has powerful hardware and accurate model calibration due to its computational complexity.

5.3.5 Theta Star Planner

Theta Star Planner is a search algorithm that extends the A Star algorithm to allow the robot to move down a line drawn between the grid nodes, with the caveat that it is only able to move along the adjacent cells. This feature enables Theta* Planner to find more natural and concise courses compared to A*. Theta Star Planner works especially well in open spaces with few obstructions since it can use its ability to move along a straight line to find more direct courses. However, in very congested environments, the Theta* Planner may be slower than the A* since it takes longer to determine if a line drawn between two nodes is free from collisions.

5.3.6 SMAC Planner

Search-based Motion Planning, or SMAC Planner, is a motion prediction algorithm that uses a frontal search to find the best path in a static environment. This planner is very effective in environments with many obstacles because it can quickly explore the robot's configuration space and choose a course that minimizes a specific cost parameter, such as distance or time required. One of SMAC Planner's unique features is its ability to manage environments with free space features, such as narrow corridors or small apertures. This makes it especially suitable for use in indoor environments where space is limited. Furthermore, the SMAC Planner can be easily used to include additional constraints, such as speed or robot acceleration limits.

Feature	SMAC Planner	Theta Star Planner
Ideal Environment	Complex environments with many obstacles	Open environments with few obstacles
Handled Constraints	Free space	Less efficient with complex constraints
Computation Speed	Depends on environment complexity	Can be slower in congested environments
Path Quality	Optimal paths in terms of defined cost	More natural and often shorter paths

Table 5.1. Comparison of SMAC and Theta Star Planners

5.3.7 Navigation with Keepout Zones

The *KeepoutFilter* plugin is a useful feature in ROS2 navigation that allows robots to move safely and efficiently in complex settings by specifying which areas to avoid and which paths to take. Users can designate regions as no-go zones, favored lanes, or areas of changing permissibility by creating a filter mask, which is often an image file, such as Figure in this study-case. The plugin then overlays this data on the robot's costmap, impacting the path planner's decisions. This adaptability enables personalized navigation behaviors such as avoiding impediments, sticking to predetermined paths, and improving job completion. The *KeepoutFilter* is especially beneficial in industrial and warehouse environments where precision navigation and safety are critical. Key features include the ability to handle many map types (trinary, scale, and raw), support for dynamic environments via real-time updates, and connection with other navigation plugins. Developers can improve the autonomy and reliability of their robotic systems by implementing the *KeepoutFilter* properly.

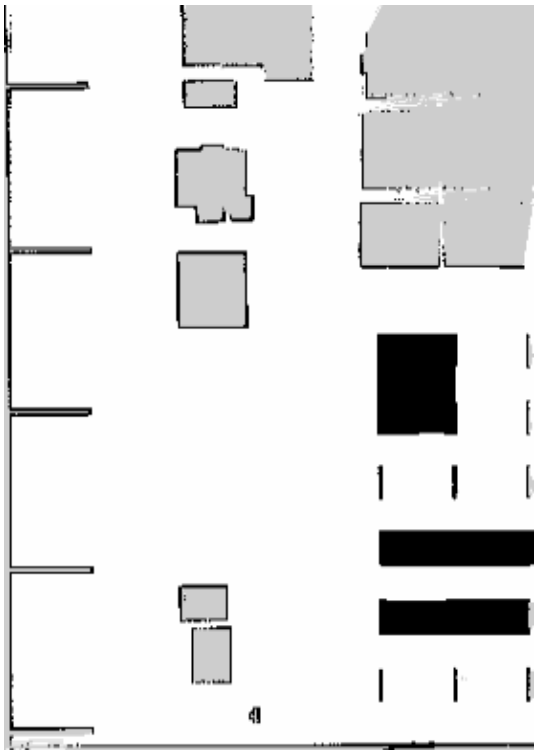


Figure 5.12. Map with the keepout zone

Chapter 6

Hardware

6.1 FIXIT Project

In the era of Industry 4.0, when digitization and automation are altering manufacturing processes, the FIXIT project is at the forefront of innovation. It is a ground-breaking drone-rover system designed to aid with inspection and maintenance activities in industrial contexts. FIXIT is a synthesis of the most sophisticated Industry 4.0 technologies, including mobile robots, augmented reality, the Internet of Things, drones, and additive manufacturing. This one-of-a-kind device can operate in complex and difficult-to-reach situations, allowing for more efficient maintenance interventions and operations. FIXIT's lightweight and adaptable construction, made possible by additive manufacturing, makes it appropriate for a wide range of interventions. The combination of drone and rover, along with modern communication systems and augmented reality, allows for detailed inspections and intuitive operator assistance. The on-board edge computing device enables real-time processing of acquired data, which improves operational efficiency.

Figure 6.1. FIXIT structure

The part of the project that concerns us is the rover. The Scout Mini mobile base from AgileX Robotics, a compact and agile platform, served as the foundation for this study. Its interoperability with the ROS framework and extensive capabilities make it an attractive option for a variety of applications. The Scout Mini's Mecanum wheels allow for omnidirectional movement and a larger cargo capacity than conventional models. The robot's small size, combined with powerful motors and sophisticated control systems, allows it to reach great speeds and mobility. The open-source software development kit (SDK) includes a C++ interface for easy interaction with other systems, and the CAN bus interface enables communication with other devices. The Scout Mini provides real-time feedback on motor status, encoder readings, and temperature, ensuring dependable operation and monitoring.

Figure 6.2. Agilex Scout Mini

6.1.1 Mecanum wheeled mobile robot

Wheeled mobile robots can be classified into two primary categories based on their wheel configurations: Conventional Wheeled Mobile Robots and Mecanum Wheeled Mobile Robots. Conventional wheels encompass three distinct types: fixed wheels, steerable wheels, and caster wheels. These types exhibit varying characteristics due to their unique rotational axes. Each wheel type offers specific advantages, enabling the robot to achieve desired behaviors in different applications. In contrast, Mecanum Wheels consist of multiple rollers arranged around the wheel at a 45-degree angle. This distinctive configuration allows the robot to move not only forward and backward but also sideways and diagonally. This capability is crucial for navigating narrow spaces and effectively avoiding obstacles. As a result, Mecanum-wheeled robots are particularly well-suited for dynamic environments, such as industrial workplaces.

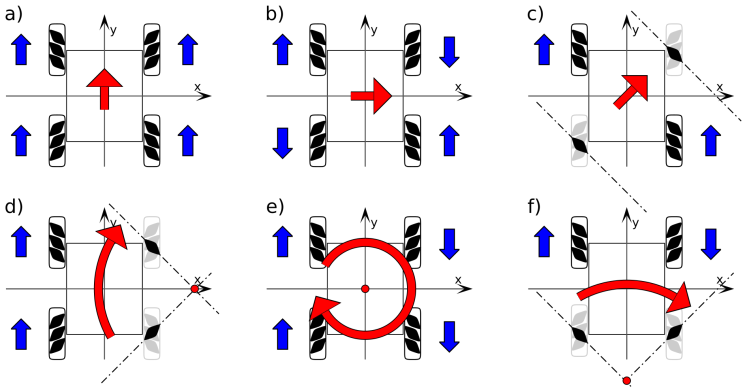


Figure 6.3. AMR movements allowed by mecanum wheels [27]

6.2 Sensors

The mobile platform incorporates a suite of sensors designed to facilitate efficient SLAM algorithm and ensure safe autonomous exploration. Beyond the encoders and Inertial IMU primarily used for localization, the platform is equipped with a RP-LIDAR A1 and an Intel RealSense Depth Camera D435i. These sensors actively perceive the environment, providing essential data for navigation and obstacle avoidance.

6.2.1 RP-Lidar

A RP-LIDAR A1 laser scanner was employed in the system. This low-cost, compact device, commonly used in robotics applications, especially for autonomous exploration, localization, and mapping, is capable of scanning a 360-degree environment. Manufactured by SLAMTEC, the 2D laser scanner features a rotating range scanner driven by a motor. It utilizes high-speed vision acquisition and operates based on the principle of laser triangulation. Specifically, the RPLIDAR emits an infrared laser beam, and the vision module detects and samples the reflected signal. The sensor can scan a distance of up to 12 meters at a configurable rate ranging from 2Hz to 10Hz. Detailed specifications of the RP-LIDAR A1 can be found in Table 6.1

Figure 6.4. RP-LIDAR A1 [24]

	Parameters	Description
Physical	width x length x height:	96.8 x 70.3 x 55 mm
	weight:	170 g
Features	use:	indoor/outdoor
	measuring range:	0.15-12 m
	range resolution:	8 K
	sampling frequency:	5.5 Hz
	system voltage:	5 V
	system current:	100 mA
	temperature range:	0° -40°^{0}C
	angular range:	360Â°
	accurancy:	1% for $\leq$3m, 2% for 3-5m

Table 6.1. RP-LIDAR A1 Datasheet

6.2.2 Intel RealSense Depth Camera

The system also incorporate a depth camera and an Inertial Measurement Unit (IMU). The IMU, comprising an accelerometer and gyroscope, measures linear acceleration and angular velocity, respectively, providing 3D spatial orientation. This sensor combination is commonly used in robotics due to its performance and cost-effectiveness. The depth camera's wide field of view and low light sensitivity enable navigation in diverse environments, including indoor and outdoor spaces. A multi-camera configuration ensures uninterrupted operation and precise navigation within a few meters, even in low-light conditions.

Figure 6.5. Intel RealSense D435i [14]

The Intel RealSense d435i, as depicted in Figure 6.4, consists of an RGB module (1920x1080 resolution), two infrared modules, and an IR projector that enhances depth camera performance using active stereo techniques. The device captures RGB and depth images simultaneously, with a depth resolution of 1280x720 at 90 frames per second and a depth field of view of 87Â°x58Â°. The integrated Intel RealSense Vision Processor D4 enables detailed environment reconstruction. The maximum visual range is 10 meters, though accuracy may vary based on factors like calibration and lighting conditions. The official specifications provided by the manufacturer are summarized in Table 6.2

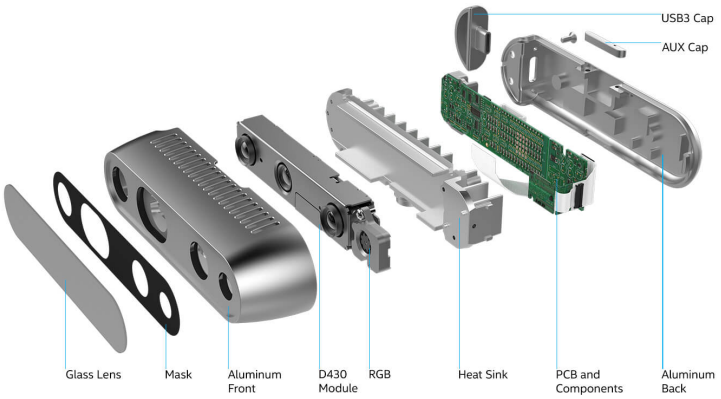


Figure 6.6. Intel RealSense D435i components [14]

	Parameters	Description
Physical	width x length x height:	90 x 25 x 25 mm
	connectors:	USB-C 3.1
Components	camera module:	module D430 + RGB Camera
	vision processor:	vision processor D4
Features	use:	indoor/outdoor
	ideal range:	0.3 to 3m
RGB	RGB frame resolution:	1920 x 1080
	RGB sensor FOV (H x V):	69° x 42°
	RGB sensor technology:	rolling shutter
	RGB frame rate:	30 fps
Depth	depth technology:	stereoscopic
	depth output resolution:	1280 x 720
	depth field of view (FOV):	87° x 58°
	depth frame rate:	90 fps
	depth accuracy:	<2% at 2m

Table 6.2. Intel Realsense Camera D435i datashhet

6.3 On board computer

The rover utilized two on-board computers, an Nvidia Jetson Xavier NX and an Nvidia Jetson Nano, to process and execute the developed Active SLAM solution. These computers were connected to the deployed sensors, allowing them to gather environmental data during the exploration phase.

6.3.1 Nvidia Jetson Xavier NX

The on-board computer is part of the NVIDIA Jetson platform, a high-performance, low-power embedded system. The Jetson Xavier NX, measuring approximately 70 mm by 45 mm, integrates a CPU and GPU onto a single chip. Its integrated software libraries, like CUDA, are well-suited for real-time applications. Designed for robotics and autonomous tasks, the Jetson Xavier NX features four USB 3.1 ports, requires a MicroSD card for operation, and supports various power modes. Detailed specifications of the board are provided in Table 6.3.

Figure 6.7. Nvidia Jetson Xavier NX [15]

Parameters	Description
Width x length x height:	103 x 90,5 x 34 mm
GPU:	Nvidia Volta architecture with 384 Nvidia Cuda cores and 48 tensor case
CPU:	6-core Nvidia Carmel ARM v8.2 64-bit CPU 6 MB L2 + 4 MB L3
Memory:	8 GB 128-bit LPDDR4x
Connectivity:	gigabit Ethernet Wi-Fi module
Display:	HDMI and display port
USB:	4x USB 3.1, USB 2.0 micro-B

Table 6.3. Jetson Xavier NX Datasheet

6.3.2 FIXIT-M board

The FIXIT-M main board is designed to power the entire FIXIT system using a battery-based approach. It features a charging system for the drone when it lands on the case and provides power to the boards required by the Automated Mobile Robot (AMR). The board offers 12V and 5V connectors to supply power to the Jetson Xavier NX and Jetson Nano, respectively. By utilizing an external battery to power the AMR's peripherals, the overall system achieves a significant increase in battery life compared to a solely internal battery system.

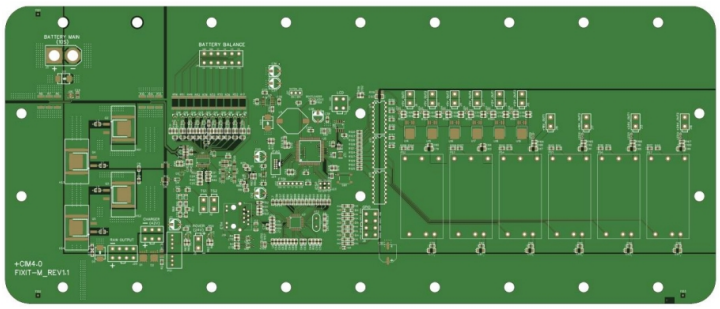


Figure 6.8. FIXIT-M PCB top view

6.4 Bunker

The mobile platform Agilex Bunker was used to conduct experiments. This robot, with a sturdy and compact structure, was chosen for its ability to operate in complex and hazardous environments. The Bunker is equipped with an advanced control system that allows for precise and autonomous navigation, even in restricted spaces. The platform is also equipped with a number of sensors and actuators, making it perfect for the implementation of advanced perception and control algorithms. Its modular design enables customization and expansion, making it an adaptable platform for research and development. The robot's tough chassis, paired with high-torque brushless motors and a sophisticated suspension system, offers dependable performance in harsh settings. A variety of sensors, including a 2D LiDAR, RGB camera, IMU, encoders, and force sensors, give detailed perception of the robot's environment. The integrated control system, which is built on a high-performance microcontroller, handles sensor data, actuator control, and navigation algorithms. The Bunker's open-source software development kit allows for easy customization and interaction with external systems, making it an excellent platform for a variety of applications such as research, teaching, and industrial automation.

Chapter 7

Conclusion

The major goal of this thesis was to develop and evaluate an Active SLAM algorithm that might enable autonomous exploration and navigation within the FIXIT project's AMR. This study effectively met its objectives by conducting a thorough review of cutting-edge SLAM algorithms and rigorous testing in both simulated and real-world situations. The developed Active SLAM system performed well in unfamiliar locations, successfully overcoming obstacles, dynamic conditions, and variable topography. The AMR's capacity to explore independently, avoid risks, and adapt to changing situations demonstrates the system's strength and dependability. Key findings from this study include

- Autonomous Navigation: The AMR can explore and navigate complicated landscapes without requiring human intervention.
- Obstacle Avoidance: The integrated obstacle avoidance algorithms enable safe and efficient navigation by reducing collisions.
- Adaptability: The system is highly adaptable, able to function in a variety of contexts and handle a wide range of jobs.
- Remote Operation: The AMR's cameras and sensors allow for remote monitoring and operation, even in dangerous settings.

The effective implementation of this Active SLAM system expands the potential for autonomous mobile robots in industrial and commercial applications. In tough or dangerous environments, potential use cases include inspection, surveillance, data collecting, and search and rescue operations. Future study could look into advances in SLAM algorithms, sensor fusion techniques, and human-robot interaction to improve the capabilities of AMRs and broaden their applications.

7.1 Future works

While the built autonomous exploration system effectively accomplishes its core goals, there are places where further improvements are possible. Although the current sensor setup ensures a high level of safety, certain impediments, particularly those near to the ground, may not be properly identified. Furthermore, the limitations of 2D LiDARs in outdoor contexts, such as their susceptibility to sunlight reflection, can degrade the system's performance.

To address these restrictions, the use of 3D LiDARs that can operate both indoors and outside is strongly encouraged. This would considerably improve the system's robustness and adaptability to different situations. Furthermore, enabling communication and collaboration between AMRs and UAVs may open up new avenues for difficult tasks like combined maintenance operations or cooperative exploration in large-scale industrial settings.

The results obtained in this thesis represent a great starting point for future robotics research. One area of particular interest is the incorporation of more advanced artificial intelligence techniques, such as deep learning and reinforcement learning, to allow the robot to acquire more sophisticated reasoning and adaptation abilities. The use of generated models may also make it easier to create more realistic simulation environments, accelerating the development of new algorithms and evaluating their performance. Similarly, the exploration of more sophisticated perception systems based on multimodal sensors and sensor data fusion techniques has the potential to significantly improve the robot's ability to interact with complex and dynamic environments.

Another promising research area is the development of robots that can effectively collaborate with humans. The use of intuitive interfaces, including as gestural and vocal interfaces, in conjunction with emotion recognition techniques, may facilitate the development of social robots capable of assisting the elderly, the disabled, and people with special needs. Furthermore, the exploration of collaborative robotics could lead to the development of robots capable of working side by side with human operators in a safe and efficient manner, increasing the productivity and flexibility of production systems.

It is important to evaluate the ethical implications of introducing these technologies. The development of clear guidelines and the establishment of security standards are critical for ensuring responsible robot use and mitigating potential risks. Finally, it is important to note that robotics is a constantly evolving field, and future research will have to address ever-more complex challenges, such as the development of robots capable of operating in uncertain and dynamic environments, adapting to new situations, and making autonomous and responsible decisions.

Bibliography

[1] Muhammad Farhan Ahmed, Khayyam Masood, Vincent Fremont, and Isabelle Fantoni. Active slam: A review on last decade. *Sensors*, 23(19), 2023.

[2] Automaticaddison. Topics vs. services vs. actions in ros2-based projects. *Robotics*, 2021.

[3] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks, 2013.

[4] Elisa Tosello,. Motion planning for a robot with many degrees of freedom, 2013.

[5] Omar Elmofty. Ros2vs.ros1-key differences and which one is better? *Medium*, 2022.

[6] Macwnski et al. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, page 2783, 2021.

[7] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, pages 23 – 33, 1997.

[8] Kristin Glass, Richard Colbaugh, David Lim, and Homayoun Seraji. Real-time collision avoidance for redundant manipulators. *IEEE Trans. Robotics Autom.*, pages 448–457, 1995.

[9] Gregory Antonovsky. Bidirectional rrt* fnd algorithm, 2018.

[10] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. A tutorial on graph-based slam. *IEEE Intelligent Transportation System Magazine*, pages 30–38, 20010.

[11] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. Fast and accurate slam with raoâblackwellized particle filters. *Robotics and Autonomous Systems*, pages 30–38, 2007. Simultaneous Localisation and Map Building.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968.

[13] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23:1036 – 1049, 2007.

[14] Intel RealSense Depth Camera. https://www.intelrealsense.com/depth-camera-d435/.

[15] Jetson Xavier NX. https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-xavier-nx/.

[16] Akshay Kumar. An introduction to simultaneous localization and mapping (slam) for robots. *Control Automation*, 2020.

[17] Agnieszka Lazarowska. A discrete artificial potential field for ship trajectory planning. *Journal of Navigation*, pages 1–19, 2019.

[18] Andrea Macario Barros, Maugan Michel, Yoann Moline, Gwenole Corre, and Frederick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 11, 2022.

[19] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 2022.

[20] MathWorks. Simultaneous localization and mapping (slam).

[21] MathWorks. Vector field histogram, 2024.

[22] Ricardo Tellez. A history of ros (robot operating system), 2019.

[23] AWS Robotics. Aws robomaker small warehouse world, 2023. GitHub repository.

[24] RPLIDAR A1. https://www.slamtec.ai/product/slamtec-rplidar-a1/.

[25] Kenta Takaya, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. Simulation environment for mobile robots testing using ros and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101, 2016.

[26] Bryan Thibodeau, Stephen Hart, Deepak Karuppiah, John Sweeney, and Oliver Brock. Cascaded filter approach to multi-objective control. volume 4, pages 3877–3882, 2004.

[27] Wikipedia contributors. Mecanum wheel — Wikipedia, the free encyclopedia.

[28] Wikipedia contributors. Motion planning — Wikipedia, the free encyclopedia, 2024.

[29] Zhang Xuexi, Lu Guokun, Fu Genping, Xu Dongliang, and Liang Shiliu. Slam algorithm analysis of mobile robot based on lidar. In *2019 Chinese Control Conference (CCC)*, pages 4739–4745, 2019.

[30] Yang Zhaofeng and Zhang Ruizhe. Path planning of multi-robot cooperation for avoiding obstacle based on improved artificial potential field method. *Sensors and Transducers*, 165:221–226, 2014.