POLITECNICO DI TORINO

Master Degree in Mechatronic Engineering

Master Degree Thesis

Multi-robot frontier-based exploration strategies for mapping unknown environments

Supervisors:

Prof. Alessandro Rizzo

Dr. Stefano Primatesta

Candidate:

Alex Vellucci

Academic Year 2019/2020

Acknowledgements

The 2020 has been a really difficult year for everyone because of pandemic that in different ways has touched everyone. For this reason the greatest thought goes to all the victims of the pandemic and all people who suffered in first person because of Covid-19. Nevertheless in this situations it is possible to really appreciate the value of many people. In my case I could appreciate the huge humanity and kindness of my supervisor, Professor Alessandro Rizzo, who accepted to help me in finishing the thesis when I had to come back from Erasmus, for this I want to deeply thank him. Moreover I want to show all my gratitude to Stefano Primatesta who always has helped me with commitment and professionalism. Even if my Erasmus experience has been prejudiced because of pandemic, I want to thank Prof. Lino Marques of University of Coimbra who let me work on a really interesting topic, that really fascinated me and I hope to be able to continue in this field for future works. I want to thank the Polytechnic of Turin for all the opportunities that gave me in the two years and for their professionalism in providing high level formation. Moreover I want to thank all the other people who supported me, Roberta first of all, who has always been by my side and never backing down. I want to dedicate a thank also to my colleague Giuseppe Aiello, loyal friend and constant companion of this university path. Last but not least I want to express my deep gratitude to my family

who always supported me and helped me in chasing my dreams, I will never end to say you "Thank you".

"To my Mother and my Father,

worlds will never be enough

to express how much you are important for me"

Abstract

With the recent developments in robotics, mobile robots are gaining momentum and are increasingly involved in our lives. Robot exploration and mapping are essential tasks for robot navigation in unknown environments. In the last three decades the challenge has grown and many researches have tried to accomplish these tasks through a team of robots. The use of multi-robot systems represents a big challenge because they require robots able to collaborate each other and, then, to create coordination techniques to make the multi-robot system efficient. Multi-robot mapping introduces many advantages, providing a faster and efficient map building even in high dimensional unknown environments. In a multi-robot system, each robot has to be able to move autonomously in the map, avoiding obstacle and reaching desired goals to explore the environment. Furthermore, in order to explore optimally the environment, e.g. minimizing the exploration time, it is necessary to define an efficient technique to determine optimal target points to each robot, considering the multi-robot context, that makes the exploration fast. In this work five algorithms for multi-robot exploration have been evaluated; four of these are defined and used in other works and represent the state-of-art in this field, whereas the fifth strategy is a novel solution proposed in this work as a valid and effective multi-robot exploration strategy. Each of these algorithms is frontier-based, i.e., exploiting the

frontier-detection approach, which is one of the most common and efficient exploration strategy. These algorithms are tested and analyzed, as well as compared with each other. Moreover, evaluation criteria are presented and discussed to efficiently evaluate the proposed frontier-based algorithm. Finally the outcome of the tests are provided with a discussion emphasizing the difference between different strategies and the improvements of the proposed technique.

Contents

Chapter 1

Introduction

Building a map of an unknown environment is a critical problem in autonomous robotics. The generation of a map can be done for various reasons, for example, to find an object of interest in a rescue environment, as happen in many military operations, where it is needed to find a victim as quickly as possible without jeopardizing soldiers. Learning maps has therefore been a major research issue in the robotics community over the last decades. Map building methods can be *passive* or *active*. The *passive* ones only perceive information about the environment to build a map. On the other hand, the *active* ones additionally plan the motion of the vehicle in order to guide it through the environment.

Maps are [21]:

> "an inexhaustible fund on interest for any man with eyes to see or with two pence worth of imagination to understand with"
>
> -R.L. Stevenson, Treasure Island

and the mapping problem can be referred to as "What does the world look like?".

Since the robot does not have eyes to see the world, the only solution for it to build a map of the environment is to gather all the information it can obtain by its sensors into a given representation. The mapping problem becomes more difficult in the case where the environment changes over time. Most mapping techniques assume that the environment is static and does not change over time even if, it is clear that this is an unrealistic assumption since most places where robots are used are populated by humans. In general, learning maps with single-robot systems requires the solution of three tasks, which are mapping, localization and path planning. Building a map without being able to estimate the position of the robot in the environment is useless and the wrong interpretation of it would produce a fake result in the mapping process. For this reason, it is fundamental to be able to localize the robot time by time in order to efficiently answer to the question "Where am I?".

In the area of localization there exist two cases: (*i*) the initial position of the robot is known a priori, also know as *pose tracking*, and (*ii*) no prior knowledge about the robot position is given, also called *global localization* problem [61]. Finally there is the path planning or motion control problem, i.e., the techniques used to move the robot from a starting point to a target point, which takes into account the question of "How can I reach a certain point in the map?". The interaction of these tasks in a robotic system is also known as Simultaneous localization and mapping, simply called SLAM, that is the problem of building a map, while, at the same time, localizing the robot within that map.

SLAM has been widely studied in the last decades and several techniques and improvements have been introduced. The huge interest in this area, has lead several researches to think about what could improve using a multi-robot approach. This

consideration has brought the born of a new robotics field that considers all the problems previously mentioned using a multi-robot approach, sometimes described as a *swarm*, as a *colony* or a *collective*. In general the problem can be addressed as the cooperation of multiple robots all aiming at solving the same objective [21]. The use of multiple robots has several advantages and at the same time arises the problem of choosing the most appropriate design to let the robots to cooperate effectively, without increasing too much the complexity of the solution. The definition of an appropriate technique to solve this problem introduces many improvements, such as the use of a fleet of simple robots, rather than a complex and more expensive one able to efficiently work alone. Moreover, the use of multiple robots makes the system less susceptible to failures, that, in many cases, just thinking of a medical field, it is a primary requirement.

In order to summarize, the basic questions this projects tries to answer to are:

- How to estimate the position of the robotic?
- How to guide the robot during autonomous exploration?
- How to create a map of an unknown environmental?
- How to efficiently coordinate a team of mobile robots?

In the following chapters some solutions to solve these problems will be discussed. Chapter 5 defines the evaluation criteria used to evaluate the performances of the five exploration strategies compared in this thesis.

The comparison is done by performing several simulations. Anyway it is clear that the effective feedback about the efficiency of an algorithm has to be checked in a real environment, where there exist many variants and aspects that don't take place in a

simulated environment. Nevertheless, the employment and testing of algorithms is more difficult in a real environment rather than in simulations and also more time demanding. For this reason the use of simulations is a necessary step in order to optimize the test operation, provided that appropriate experimental methodology [5] that follows experimental principles well established in science [4] are applied.
Since the aim of this work is the investigation of methodologies for mobile robot exploration, it is needed to define how to compare different approaches. The simplest solution is to test the algorithms in the same framework and perform several trials in the same scenario. It is evident that this is a peculiarity of a simulation environment that provides a controllability of the scenario, differently from a real environment in which is quite impossible to reproduce the same framework for a number of tests [26].
In the Section 5.6.3, three performance indicators will be introduced. Finally tests will be executed in simulation performed using ROS (Robot Operating System) and Gazebo simulator.

1.1 Problem Statement

The goal of this thesis is the definition of a frontier-based algorithm able to manage a fleet of robots to explore autonomously an unknown structured environment, in order to create a grid map of the environment optimizing the exploration time. The map creation is performed using an occupancy grid mapping [42]. In the occupancy grid mapping the map is divided in cells and to each cell is assigned a value that can be "*free*", "*occupied*" or "*unknown*". Each robot moves in a $\Re^2$ space and is labeled as $R_1, R_2, ..., R_n$. The robot model is the differential drive and it is endowed

of two encoders, a GPS and a 360 degrees lidar with a limited sensing range ρ. Each robot is equipped of the following algorithms:

1. A localization algorithm
2. A path planning algorithm
3. A mapping algorithm

Finally a map-merge algorithms merges all the maps generated by each robot and a centralized algorithm takes care of assigning to each robot the best goal point to visit, basing on the merged-map.

1.2 Problem Specification

In this work, a frontier based exploration method is developed to create a grid map of the environment. The aim of robot exploration is exploring the environment in the shortest possible time, thus it's clear that the performance parameter is the exploration time which can be measured as the longest exploration path traveled by a single robot in the team [26]. Furthermore, another interesting parameter is the longest length applied to one robot between to consecutive exploration point. This can help to understand how the strategy assigns a goals given an initial pose and to quantify how one goal influences the overall length traveled by one robot and consequently the exploration time.
Exploration strategies for multi-robot systems are mainly divided into *centralized* and *decentralized*. In this paper only centralized algorithms are used to simplify the problem.

Many parameters influence the ability of a robot of gaining information, just thinking about the path the robot follows to go from one point to another, which is clearly related with the motion planner used. In this work three motion planning strategies have been taken into account which are the Artificial Potential Field (APF), the Vector Field Histogram (VFH) and finally the ROS Navigation stack.

Another parameter that leverage the algorithm outcome is the decision-making frequency [3], which is usually set as: (*a*) *goal replanning* (GR), in which the assignment of a new goal is assigned when a robot reaches its previously assigned goal, or (*b*) *immediate replanning* (IR), that means in the case of frontier based exploration algorithm, that a new goal position is assigned as soon as the current goal is no more a frontier point. Another important parameter is the sensing frequency, i.e. the frequency at which the robot gathers new information through its sensors. Authors of [3] experimentally confirm that, generally, a higher frequency provides better results, i.e., a shorter exploration path that consequently affect the exploration strategy result. On the other hand, it may not be the case of the computationally demanding methods because of limited computational power. Hence, a less sophisticated strategy may perform better than a more demanding approach on the same hardware because of a more frequent decision-making [26]. A problem in almost all the exploration algorithms is their high demanding computational burden, that can be decreased adopting approaches like [37].

Chapter 2

Robot Operating System (ROS)

2.1 Overview

"ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers" [54] . ROS is an abbreviation of Robot Operating System and is a Meta-Operating System, i.e., a system that performs processes such as scheduling, loading, monitoring and error handling by utilizing virtualization layer between applications and distributed computing resources.

This framework works on Linux distributions, even if a beta version is developed for Windows 10 and MacOS.

Figure 2.1: ROS as a Meta-Operating System [75]

One of the most important feature of ROS is that data communication is supported not only on one operating system, but by multiple operating systems, hardware, and programs. In fact in the execution of this work, three laptop have been used concurrently to perform experiments, each one executing certain tasks. The main characteristics of ROS are [75]:

1. Distributed Process: each process runs independently and exchanges data.

2. Package management: multiple processes having the same purpose are treated as packages

3. Public repository: every package is public and each developer can access to them in specific repository (e.g. GitHub, GitLab, etc.)

4. API: for each program, ROS calls an API and merges it into the code making possible every ROS product and service to communicate with the other products and services without knowing their implementation

5. Supporting various programming languages, such as C++, Pytohn and LISP

6. Supporting third-party libraries such as OpenCV, PCL, and many others

2.2 Concepts

ROS executable files are called **nodes** that are the smallest unit of processes running in ROS. This nodes can *publish* or *subscribe* to a **topic**, i.e., buses over which nodes exchange messages. Furthermore nodes can provide or use **services**, that are synchronous bidirectional communication between the service client, i.e., the entity that asks for a service, and a service server, the provider of the service. Another type of communication present in ROS is the **action** used when a requested task (or action) takes long time to respond after receiving a request and intermediate responses are required until the result is returned. ROS nodes use a ROS client library to communicate with other nodes. The communication between nodes occurs in a network. The use of a network make possible and simple the communication between multiple machines connected to the same ROS network, in which each machine is identified with an URI (Uniform Resource Identifier). In any ROS communication it must be specified the **master**, that is the server for node-to-node connections and message communication. Without the master registration to the network, nodes can't communicate and no topics or services are provided. If the master is registered, then the communication with the slaves happen through XMLRPC (XML-Remote Pro-

cedure Call) that is an HTTP-based protocol and also nodes uses this protocol for the communication with the master. Figure 2.2 provides an example of interaction among nodes, exchanging messages, and the master.

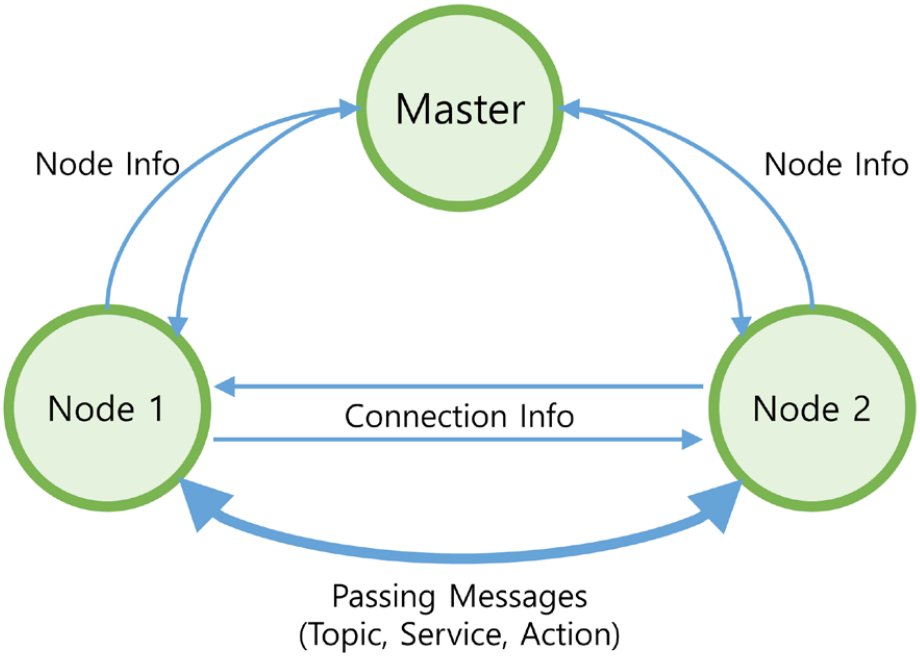


Figure 2.2: Ros nodes [75]

To have a graphical interpretation of all the message types listed so far and to describe how nodes communicate, the figure 2.3 is depicted below:

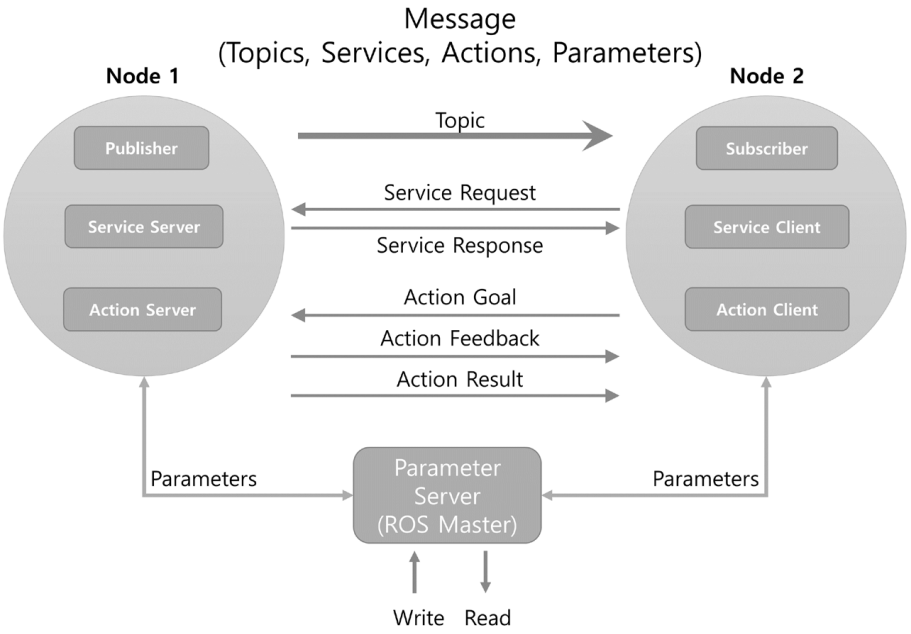


Figure 2.3: Message Communication between Nodes [75]

2.3 Tools

Since ROS is a robotic tool, it disposes about 180 types of robots in the Wiki page. Anyway it is possible to create customized robots and add all the functionality a robot could have to the customization. In this work, since there was not a specific requirement in the typology of robot, one of the most simple and used robot in the ROS environment has been used. In particular, the choice is the TurtleBot3. "TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping." [49] Turtlebot3 is a differential drive mobile robot, which mathematical model is given in section 2.1, and one of the main characteristics of this robot is the significantly reduced size and the good accuracy. Turtlebot3 has many sensors integrated, anyway it is possible to

customize the payload for the specific needs.

There exist three models of TurtleBot3 which are TurtleBot3 Burger, Waffle and Waffle Pi that differ each other for the shape and dimension. The model used in this thesis is the Burger, depicted in figure 2.4:

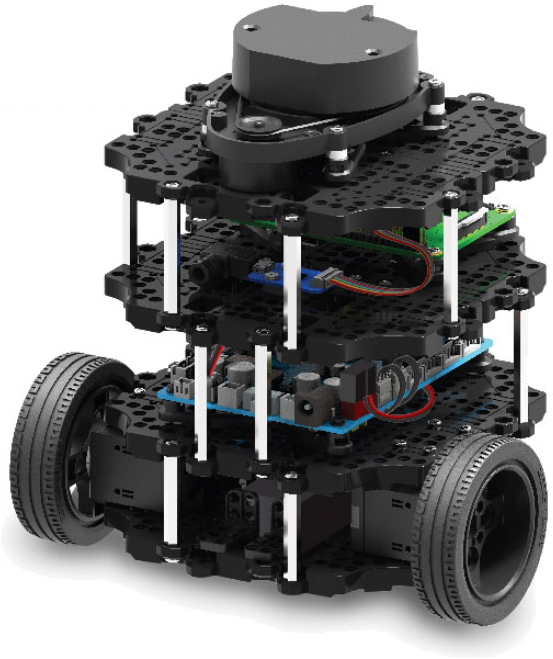


Figure 2.4: TurtleBot3 Burger

Sensor plugged with the Turtlebot3 model are: two encoders, one 360 degree lidar (LDS-01) with modifiable range distance (max $10m$) and a GPS.

Among all the ROS tools, a really useful one is *Gazebo*, which is a 3D robotic simulator environment that provides robots, sensors, environment models for 3D simulation required for robot development, and offers realistic simulation with its physics engine.

Another important tool that can be used together with the Gazebo simulator is

Rviz (ROS Visualizer). Rviz is a 3D robot topic visualizer software that makes easy to see the Gazebo simulation topics. The utility of Rviz consists for example in the possibility to visualize the map that a robot is creating, plot topics published by a certain robot in order to have a graphical interpretation of its results or, for example, examine the coordinate transformations between robot links. Coordinate transformation between robot's links is essential to define the geometrical relation between all the part of a robots (see figure 2.5). If the transform between all the part of a robots are not well defined, it is difficult to execute simple computations. For example when the robot finds the distance to an object, this distance is evaluated by the lidar sensors that are positioned in a certain point on the robot. Clearly the distance from the lidar to an object and the distance between the base footprint of the robot are not the same because have different position. Then, to define this distance it is needed to define the geometrical relation between the coordinate of the lidar with the one of the base of the robot. These geometrical relations are defined through transformations, supported by the TF package.

The TF package allows also to extract the full tree of robot coordinate transforms; an example of it is provided by figure 2.6.

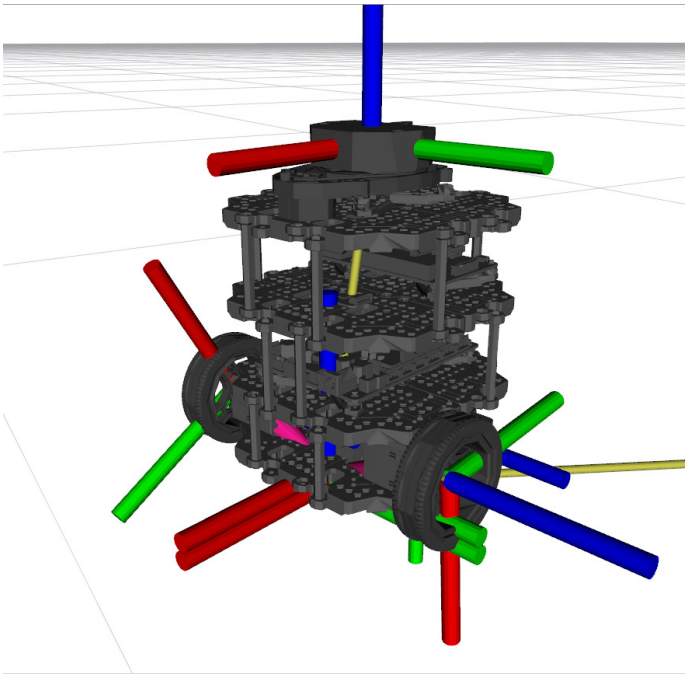


Figure 2.5: Link Transforms in Rviz

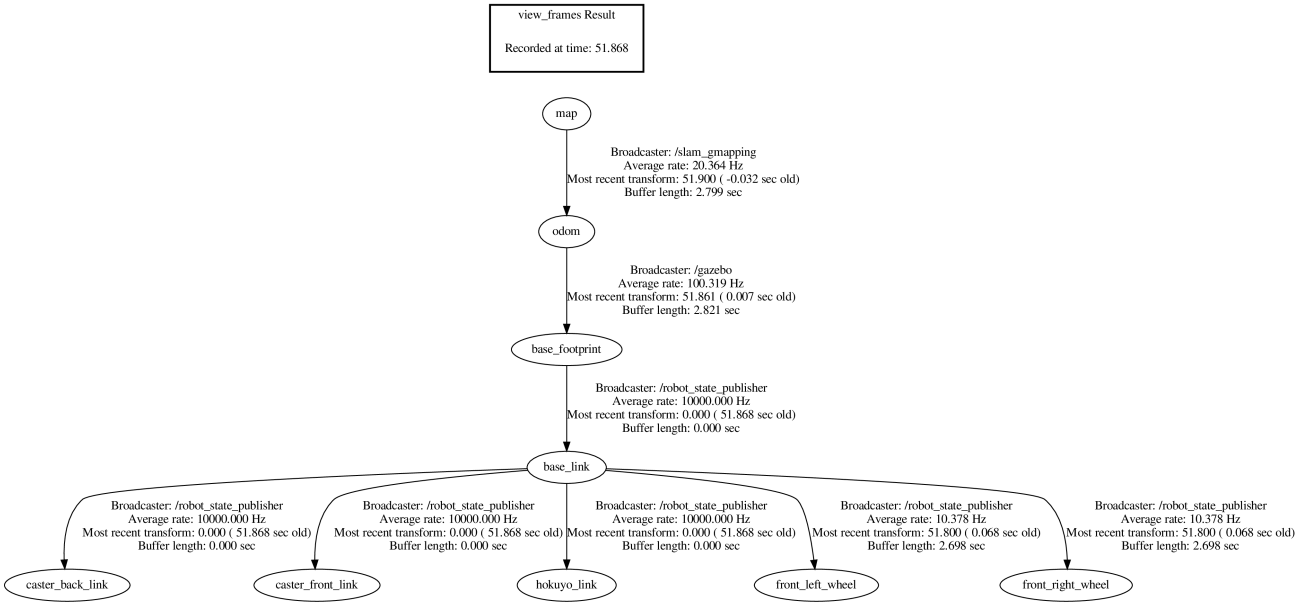


Figure 2.6: Tf view Turtlebot

Chapter 3

Mobile robot

3.1 Mobile Robot Model

"A mobile robot is the combination of various physical and computational components" and can be described as a collection of four subsystems [21]:

- *Locomotion*: How the robot moves through the environment
- *Sensing*: How the robot measures properties of itself and the environment
- *Reasoning*: How the robot converts measurements into actions
- *Communication*: How the robot communicates with an outside operator or with other entities

"Locomotion is the process by which an autonomous robot or vehicle moves" [21]. The analysis of the motion of a mobile robot is performed through the study of **dynamics**, so the study of the motion in relation to the forces applied to the robot, and the **kinematics**, i.e., the study of the motion without considering the forces

affecting the motion. The strategy through which a robot performs a locomotion task depends on the type of robot we are focusing on. In case of wheeled mobile robots these strategies can be effectively described only considering the kinematics of them, whereas in case of legged, space, aquatic and flying robots, it is usually necessary to consider the dynamics of the robot [21].
Depending on the application domain in which the robot operates, four categories of mobile robots can be listed [21]:

- *Terrestrial*, i.e., robots that operates on the ground. This kind of robots, also known as ground-contact robots, are usually wheeled even if there are a huge variety of robots that can walk, climb, roll, use tracks or crawl.

- *Aquatic*, robots that are able to navigate in the water, travel over its surface or dive into it.

- *Airborne*, so all the robots able to freely move in the aire, such as helicopters, fixed-wings aircraft or dirigibles. Usually these kinds of robot share many issues of the aquatic ones.

- *Space*, probably the robots that have to overcome the greatest issue of operating in the microgravity of outer the space, usually for space station maintenance. Typically this kind of robots are divided into climbing robots and free flyers.

In this work a terrestrial mobile robot has been used and, in particular, a differential drive kinematic model robot has been exploited.
Differential drive is the simplest drive mechanism for a ground contact mobile robot. This robot consists of two wheels mounted on a common axis controlled by separate

motors in the backside of the robot. The rotation of the robot occurs by varying the relative velocity of the two wheels, greater is the velocity of one wheel with respect to another, greater will be the rotation of the robot. Due to the structure of the model, at each time instant, the point at which the robot rotates must have the property that the left and right wheel follow a path that moves around the Instantaneous Center of Curvature (ICC), that is a point around which each wheel on the vehicle follows a circular course, at the same angular rate ω. For this property the following equations hold [21]:

$$\omega \cdot (R + \frac{l}{2}) = v_r \tag{3.1}$$

$$\omega \cdot (R - \frac{l}{2}) = v_l \tag{3.2}$$

where l is the distance of the axle between the center of the two wheels, v_r and v_l are the velocity of the right and the left wheel, respectively, and R is the signed distance from the ICC of the midpoint to the two wheels.

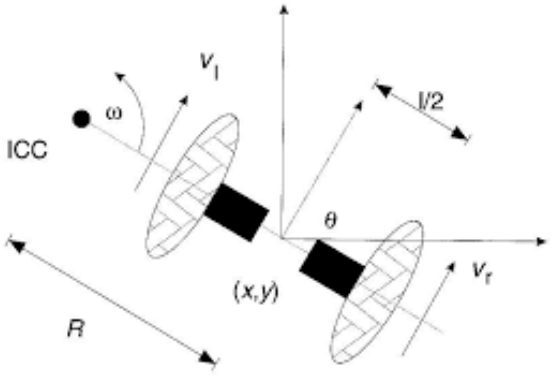


Figure 3.1: Differential Drive Model

Solving the equations (3.1) and (3.2) and considering that all those quantities are functions of the time we find:

$$R = \frac{l \cdot (v_r + v_l)}{2 \cdot (v_r - v_l)} \tag{3.3}$$

$$\omega = \frac{(v_r - v_l)}{l} \tag{3.4}$$

From these equations a number of observations can be drawn. For example, if the left and the right velocity are equal in module and direction, the consequent angular velocity is zero and the radius R is infinite, so the robot moves on a straight line. On the contrary, if the velocities are equal in module and with opposite direction, then the radius is null and the robot rotates around the mid point between the two wheels, i.e., rotates in place.

The ability of differential drive wheeled robot of rotating in place is the reason why this kind of robot is popular and is preferred to other types of robot, since it makes possible to effectively move in complex and cluttered spaces.

Anyway the kinematic structure has some limitations, for example the robot is not able to directly move along the wheel axle and the error in the trajectory is strongly dependent on the relative velocity between the two wheels, so it is quite difficult to have a perfectly straight trajectory since in many cases is not possible to have a complete identical velocity between the two wheels.

By manipulating the control parameters v_r and v_l we can get the robot to move to different positions and orientations. Knowing velocities v_r and v_l and using equations (3.3) and (3.4) , we can find the ICC location:

$$ICC = [x - R \cdot sin(\theta), y + R \cdot cos(\theta)] \tag{3.5}$$

and at time $t + \delta t$ the robot's pose will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{3.6}$$

Without enter in details, the motion equations, also known as forward kinematics problem, of a robot able to move at velocity $v(t)$ in a certain direction $\theta(t)$ are:

$$x(t) = \int_0^t v(t) \cdot cos(\theta(t))dt \tag{3.7}$$

$$y(t) = \int_0^t v(t) \cdot sin(\theta(t))dt \tag{3.8}$$

$$\theta(t) = \int_0^t \omega(t)dt \tag{3.9}$$

Characterizing these equations for the case of differential drive robot, equations 3.7, 3.8 and 3.9 can be written as:

$$x(t) = \frac{1}{2}\int_0^t (v_r(t) + v_l(t)) \cdot cos(\theta(t))dt \tag{3.10}$$

$$y(t) = \frac{1}{2}\int_0^t (v_r(t) + v_l(t)) \cdot sin(\theta(t))dt \tag{3.11}$$

$$\theta(t) = \frac{1}{l}\int_0^t (v_r(t) - v_l(t))dt \tag{3.12}$$

Since these equations describe a constraint on the velocity of the robot that cannot be integrated to get the positional constraint, they are known as non-holonomic constraints of the model. This kind on constraints make difficult to compute the inverse kinematics for a differential drive robot. Anyway, it is possible to study the

inverse kinematic problem in a simple way, considering particular classes of control functions $v_r(t)$ and $v_l(t)$ (e.g. $v_r(t) = v_r$ and $v_l(t) = v_l$).

The reason of the choice of this robot model in this work is due to its simplicity and effectiveness, since it would have been useless to use a more complex model which would not lead to more significant results.

3.2 Odometry motion model

Odometry motion models is part of the probabilistic motion model theory and is deeply described in [61].

The motion model can be defined as a conditional density:

$$p(x_t|u_t, x_{t-1}) \tag{3.13}$$

in which x_t and x_{t-1} are robot poses in certain time instant and u_t is the motion control at time t. This density function describes the posterior distribution of the kinematic states of a robot when a motion control u_t is applied at x_t. One way to calculate the robot motion over time is to exploit odometry measurements. Odometry combine wheel encoder information to estimate the robot pose in periodic time intervals. Even if odometry readings suffer from drift and slippage errors, it is often useful especially when used as parameters in localization algorithms. Furthermore this measurement is more used for filter algorithms such as localization and mapping algorithms rather than for accurate motion planning and control for its intrinsic characteristic to be available in retrospect so only when the robot moved. The odometry model uses the relative motion information measured by the robot's internal encoders. Considering the time interval $(t-1, t]$ let's assume that the robot

advances from a pose x_{t-1} to pose x_t. What we can observe from odometry are $\overline{x}_{t-1} = (\overline{x}\ \overline{y}\ \overline{\theta})^T$ and $\overline{x}_t = (\overline{x}'\ \overline{y}'\ \overline{\theta}')^T$, where the bars represent the measurements expressed in the robot coordinate whose relation to the global world coordinates is not known a priori. The difference, under an appropriate definition of the term, of these two values is a good estimator of the difference between x_{t-1} and x_t. Let's now define the motion information u_t as:

$$u_t = \begin{pmatrix} \overline{x}_{t-1} \\ \overline{x}_t \end{pmatrix} \tag{3.14}$$

To extract relative odometry, three steps are performed starting from u_t:

1. Rotation δ_{rot1}
2. Straight line motion (Translation) δ_{trans}
3. Rotation δ_{rot2}

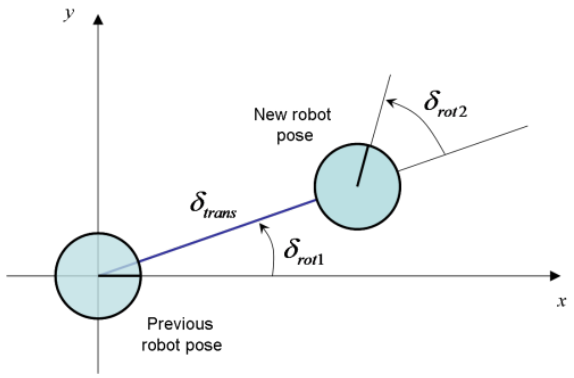


Figure 3.2: Odometry model

The assumption that probabilistic motion model assumes on these three parameter is that they are corrupted by independent noises. Finally the algorithm for computing $p(x_t|u_t, x_{t-1})$ from odometry is explained in Algorithm 1.

Algorithm 1 Odometry Motion Model Pseudo-Code

1: **function** ALGORITHM MOTION_MODEL_ODOMETRY(x_t, u_t, x_{t-1})

2: $\delta_{rot1} = atan2(\overline{y'} - \overline{y}, overline x' - \overline{x}) - \overline{\theta}$;

3: $\delta_{trans} = \sqrt{(\overline{x} - \overline{x'})^2 + (\overline{y} - \overline{y'})^2}$;

4: $\delta_{rot2} = \overline{\theta'} - \overline{\theta} - \overline{\delta_{rot1}}$;

5: $\hat{\delta}_{rot1} = atan2(y' - y, x' - x) - \theta$;

6: $\hat{\delta}_{trans} = \sqrt{(x - x')^2 + (y - y')^2}$;

7: $\hat{\delta}_{rot2} = \theta' - \theta - \hat{\delta}_{rot1}$;

8: $p_1 = \mathbf{prob}(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1\hat{\delta}_{rot1} + \alpha_2\hat{\delta}_{trans})$;

9: $p_2 = \mathbf{prob}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3\hat{\delta}_{trans} + \alpha_4(\hat{\delta}_{rot1} + \hat{\delta}_{rot2}))$;

10: $p_3 = \mathbf{prob}(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1\hat{\delta}_{rot2} + \alpha_2\hat{\delta}_{trans})$;

11: **return** p_1, p_2, p_3

3.3 Robot components

3.3.1 Lidars

Lidar is a sensor that allows to measure distances, or ranging, by illuminating the target with laser light and measuring the reflection with a specific sensor. The term lidar comes from the fusion of the worlds light and radar and it is recently used as acronym of "light detection and ranging" [70]. The main use in robotic applications is to create maps, detect obstacles and localize the robot. The main idea behind this tool is depicted in figure 3.3. This means that lidar measures the "time of flight" of the laser, and knowing the speed that the pulse travels, it computes the distance from the obstacle. Since light travels at 300 million meters per second (186,000 miles per

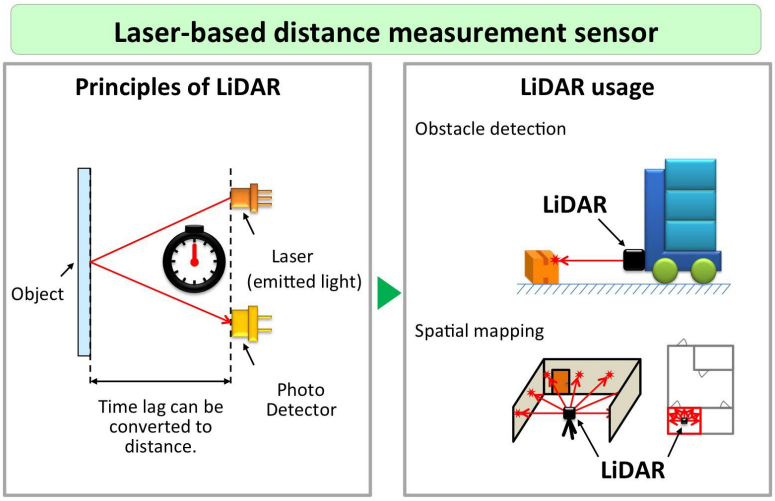


Figure 3.3: Basic time-of-flight principles applied to laser range-finding

second), very high precision equipment is needed to be able to generate data about distance. Lidar does not use only one laser usually, it is composed of a number of laser basing on the type of lidar we are using. Basing on the number of channels the lidar has, that is the number of couple emitter/receivers the instrument is equipped of, the number of beams is defined. In order to have a wider field of view many lidar systems exploit rotating assemblies, or rotating mirrors to enable the channels to sweep around the environment 360 degrees in order to avoid to build systems with too many channels that would have a higher cost. This instrument usually works at really high sampling rate on the individual emitters/receivers in order to produce a complete point cloud of the environment. Lidars are able to detect targets with different types of materials including non-metallic objects, rocks, rain, chemical compounds and many other. The lidar used by the TurtleBot3 is the LDS-01, which is a 2D laser scanner capable of sensing 360 degrees.

3.3.2 Wheel Encoders

An encoder is a sensor of mechanical motion that generates digital signals in response to motion. It is used to measure rotational speed of motors or wheels. Usually, encoders are divided in two types:

- Linear, that responds to motion along a path
- Angular, that responds to rotational motion

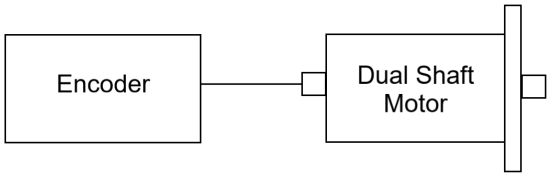


Figure 3.4: Encoder Block-Diagram

Furthermore, linear and rotatory encoders are ulteriorly spit in absolute encoder and the incremental encoder. Absolute encoders measure absolute or true angular position.This contrasts with incremental encoders which measure the change in angular position. The most famous types of encoders in the field of mobile robots are:

- Light based encoders, in which a series of markings or holes are made around the rotating disk that are used to break a light beam or change reflectivity that is detected using an infrared sensor.
- Magnetic based encoders, which use a disk with alternating magnetic orientation as markings around it that are read by a Hall effect sensors.

The type of encoder used by the TurtleBot3 Burger is the Dynamixel XL430-W250, which is a high performance networked actuator module endowed with a contactless absolute encoder.

3.3.3 Global Positioning System (GPS)

Global Positioning System, simply called GPS, is a satellite navigation system used to determine the ground position of an object. GPS works with multiple satellites, equipped with a high accuracy clock that continuously transmit their own position along with a transmission time. A receiver on the ground, by triangulating the signals sensed by at least four satellites, estimates its own position measuring the time of flight of each satellite signal. The receiver on ground has to solve the set of equations shown in compact way in 3.15:

$$c \cdot (t_{TOT,i} - t_{TOA,i} + t_s) = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \tag{3.15}$$

where:

- $c = 3 \cdot 10^8 \ \frac{m}{s}$ - speed of light
- $t_{TOT,i}$ - time of transmission of satellite i of its position (x_i, y_i, z_i)
- $t_{TOA,i}$ - time of reception of position information of satellite i
- t_s - time skew between transmitter clock-receiver clock

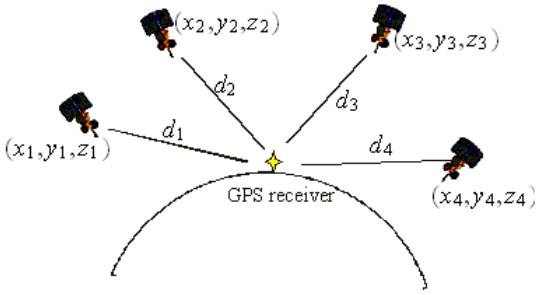


Figure 3.5: GPS Position Estimation [31]

Chapter 4

Autonomous Navigation

4.1 Localization Algorithm

"Using sensory information to locate the robot in its environment is the most fundamental problem to providing a mobile robot with autonomous capabilities." [64] Robots have to have the ability of localize themselves accurately in order to know where they are and consequently in order to know where they have to go to reach the assigned goal, because the goal information for a robot is meaningless if it does not know where it is currently located. The localization is usually referred to as *strong localization* if it is possible to estimate the position of the robot in a certain global representation of the space, or *weak localization* if we can only pronounce on the fact that a robot has already been in a certain problem, so it only answers to the question "have I been here before?" [21]. In the case of SLAM it is almost always needed a strong localization. The simplest approach for estimating the position of the robot is known as *dead reckoning*, which is the procedure of modeling the pose of a robot by updating a pose estimate obtained measuring internal quantities of

velocity acceleration and time, while the robot is moving. In most mobile robots this is achieved with the use of wheel encoders and is called odometric estimation. The estimate of the pose of the robot is usually corrupted with errors resulting from conditions such as: unequal wheel diameters, misalignment of wheels, finite encoder resolution (both space and time), wheel-slippage, travel over uneven surfaces [47]. Another way to localize the the robot can be obtained through Landmark Measurements, which consists in localizing the robot knowing the position of fixed sensor. This is a similar concept to the Global Positioning System (GPS), which, actually is a type of landmark. The landmarks send a certain signal over a certain frequency and the robot, receiving these signal is able to localize itself. In order to do so, the robot uses techniques like triangulation or trilateration. Anyway, also this technique is not optimal since it is really few robust. To exploit the methods discussed so far it is needed to filter out the error introduced by the readings and the most common way to do it in mobile robotics is through recursive filtering methods. These methods aim at estimating variables from noise observations over time. The main filter in this field is the Bayes filter, that in the localization problem is referred to as Markov localization [61]. The Bayes algorithm and the Markov localization are reported in Algorithm 2 and 3:

Algorithm 2 Bayes Filter Pseudo-Code

1: **function** ALGORITHM BAYES_FILTER($bel(x_{t-1})$, u_t, z_t):
2: **for all** x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$;
4: $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$;
5: **endfor**
6: **return** $bel(x_t)$

Algorithm 3 Bayes Localization Pseudo-Code

1: **function** ALGORITHM MARKOV_LOCALIZATION($bel(x_{t-1})$, u_t, z_t, m):
2: **for all** x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m)bel(x_{t-1})dx_{t-1}$;
4: $bel(x_t) = \eta\ p(z_t|x_t, m)\overline{bel}(x_t)$;
5: **endfor**
6: **return** $bel(x_t)$

In the Markov localization framework, the localization problem is described as estimating a posterior belief of the robot's pose at present moment conditioned on the whole history of available data and a given map as it can be seen in the above pseudo-code in the *line* 3, $p(x_t|u_t, x_{t-1}, m)$, and *line* 4 $p(z_t|x_t, m)$. Markov localization addresses the position tracking problem, the global localization problem and the kidnapped robot problem in static environments. Extended Kalman Filter (EKF) localization, grid localization and Monte Carlo localization (MCL) are three most common Markov localization algorithms.

4.1.1 Extended Kalman Filter

In this work, an EKF has been used. The EKF is the nonlinear version of the Kalman Filter (KF) algorithm and it exploits the multivariate Taylor series expansions in order to linearize the model about a working point to finally apply the Kalman filter algorithm. This because as assumption the Kalman Filter only works for Gaussian distributions and linear functions only. Since in the real world there don't exist linear systems, an extended version of the Kalman filter is needed.

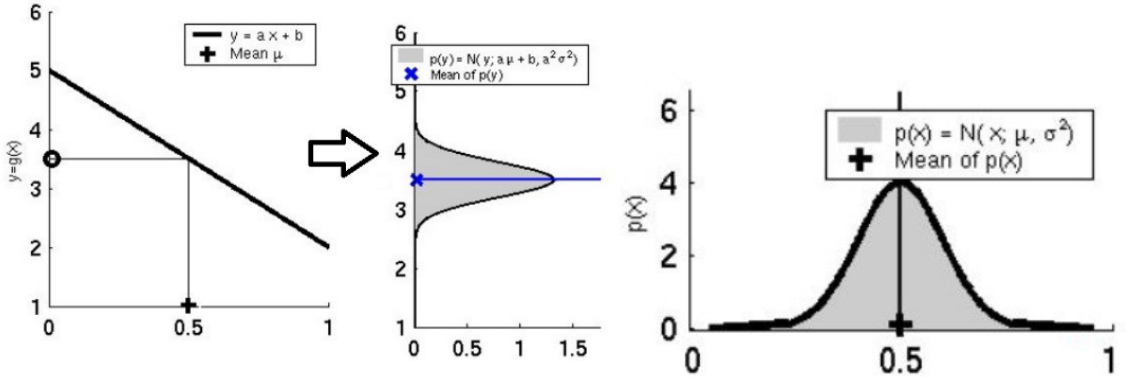


Figure 4.1: Linear Function

Anyway also the Extended Kalman Filter is based on Gaussian Distributions and the problem with non linear functions is that they lead to non Gaussian distributions. Thus it is necessary to linearize the input function and for doing it the EKF makes use of Taylor expansion. Taylor expansion approximate a g function starting from $g's$ value and slope [61]:

$$g'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{4.1}$$

The point by which the function is linearized is it median value μ_{t-1} (and μ_t) that is:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(\mu_t, \mu_{t-1}) \cdot (x_{t-1} - \mu_{t-1}) = g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{4.2}$$

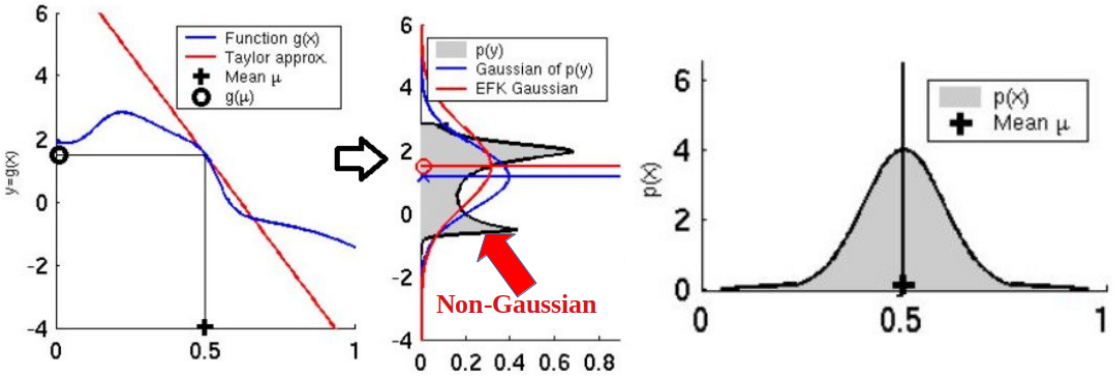


Figure 4.2: Function Linearization

Written in Gaussian form, the state probability is then:

$$p(x_t|u_t, x_{t-1}) \approx det(2\pi R_t)^{-\frac{1}{2}} \cdot$$
$$\cdot exp\left\{-\frac{1}{2} \cdot [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T R_t^{-1} [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]\right\} \tag{4.3}$$

$$p(z_t|x_t) \approx det(2\pi Q_t)^{-\frac{1}{2}} \cdot$$
$$\cdot exp\left\{-\frac{1}{2} \cdot [z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]^T Q_t^{-1} [z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu}_t)]\right\} \tag{4.4}$$

Where G_t is a $n \times n$ matrix, with n equal to the number of states, and it is known as Jacobian matrix, R_t describes the noise of the motion and Q_t describe the measurement noise. The reason for the choice of this algorithm is for its simplicity and

efficiency. Furthermore this algorithm presents the peculiarity to be really useful for sensor fusion, which makes the algorithm even more precise for the localization.

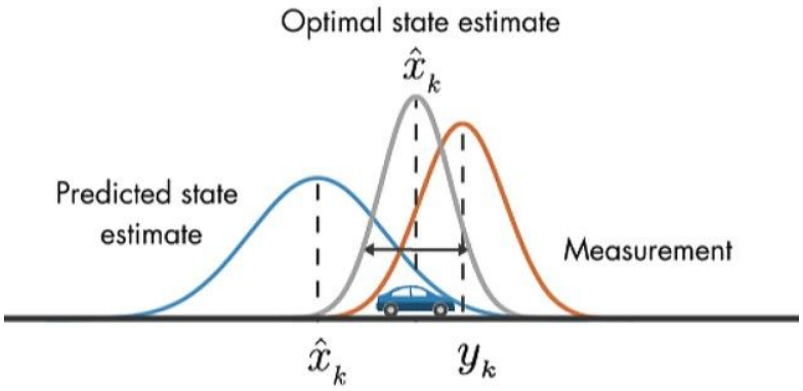


Figure 4.3: EKF example

Extended Kalman Filter is based on the Algorithm 4.

Algorithm 4 EKF Pseudo-Code

function EXTENDED_KALMAN_FILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

Prediction :

$\overline{\mu}_t = g(u_t, \mu_{t-1})$

$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ ▷ $G_t = \dfrac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$

Correction :

$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$ ▷ $H_t = \dfrac{\partial h(\mu_t)}{\partial x_t}$

$\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$

$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$

return μ_t, Σ_t

In this work the predicted state includes the pose and the velocity of the robot, the input vector is composed of linear and angular velocity of the robot taken from the encoder readings and the observation vector takes the position and orientation from GPS. Actually, the robot does not dispose of a real GPS sensor, in fact the

observation vector is composed of a fake localization system carried out by the simulation environment which describes the global position of the robot in the world it is simulated. Since conceptually the meaning of this measurements is equal to the concept of Global Positioning System, these quantities will be referred to as GPS measurements.

$$z_t = \begin{pmatrix} x_{GPS} \\ y_{GPS} \\ \phi_{GPS} \end{pmatrix} \qquad Observation\ Vector$$

$$u_t = \begin{pmatrix} v_{encoder} \\ \omega_{encoder} \end{pmatrix} \qquad Input\ Vector$$

$$G_t = \begin{pmatrix} 1 & 0 & -v_{encoder} \cdot sin(\phi_{encoder}) & cos(\phi_{encoder} \cdot dt) \\ 0 & 1 & -v_{encoder} \cdot cos(\phi_{encoder}) & sin(\phi_{encoder} \cdot dt) \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad Jacobian\ Matrix$$

Where $\phi_{encoder} = \phi_{encoder} + \omega_{encoder} \cdot dt$.

Important considerations have to be done for what concern R and Q matrices, which choice has to be done efficiently because they have repercussions on the accuracy of the algorithm.

1. R matrix is a covariance matrix associated with the errors in the state vector

2. Q matrix corresponds to the expect uncertainty in the state equations

$$R = \begin{pmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & \sigma^2 \end{pmatrix} \qquad Covariance\ Matrix$$

$$Q = \begin{pmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & \sigma^2 \end{pmatrix} \qquad State\ Uncertainty\ Matrix$$

The reason for the choice of an EKF Algorithm born due to the low accuracy of the odometric and GPS readings. GPS in simulation provide almost perfect results because clearly they don't have any type of interference, but thinking about the real world, many error sources may affect the GPS measurements. In particular, the most evident errors are due to:

- Propagation in troposphere and ionosphere; since the speed of light is not constant, due to weather conditions the traveling time of the light through troposphere and ionosphere may be different from the expected one and then causing errors in position evaluation
- Multipath effect, due to the signal bounce between buildings that then add traveling time lateness and so error in the position computation
- Satellite position inaccuracies

Also odometry position estimation has shown many inaccuracies. In particular

tests has shown that the odometry readings have a derive especially when the path followed by the robot is on straight line. In the figure 4.4 position measurements of odometry, GPS and EKF estimation are shown.

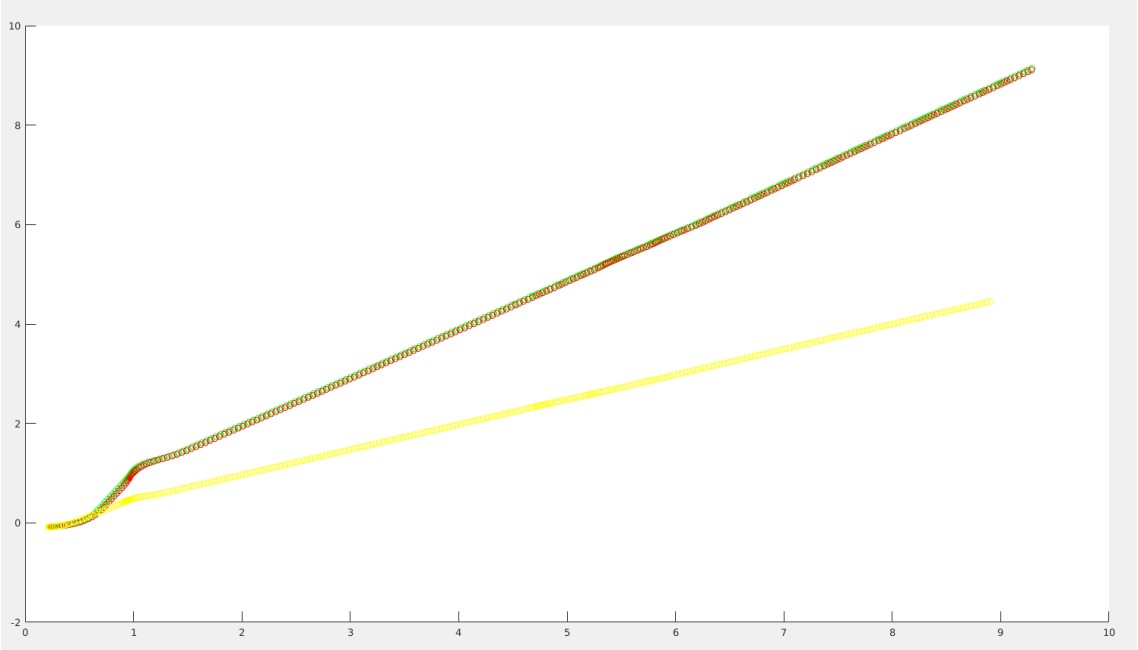


Figure 4.4: Noiseless EKF measurements

In the figure 4.4 green circles are GPS position estimations, yellow one are odometry measurements and red circles are EKS position estimations. As it can be noticed by the figure, odometry readings are really ineffective and GPS and EKF positions estimations are almost equal.

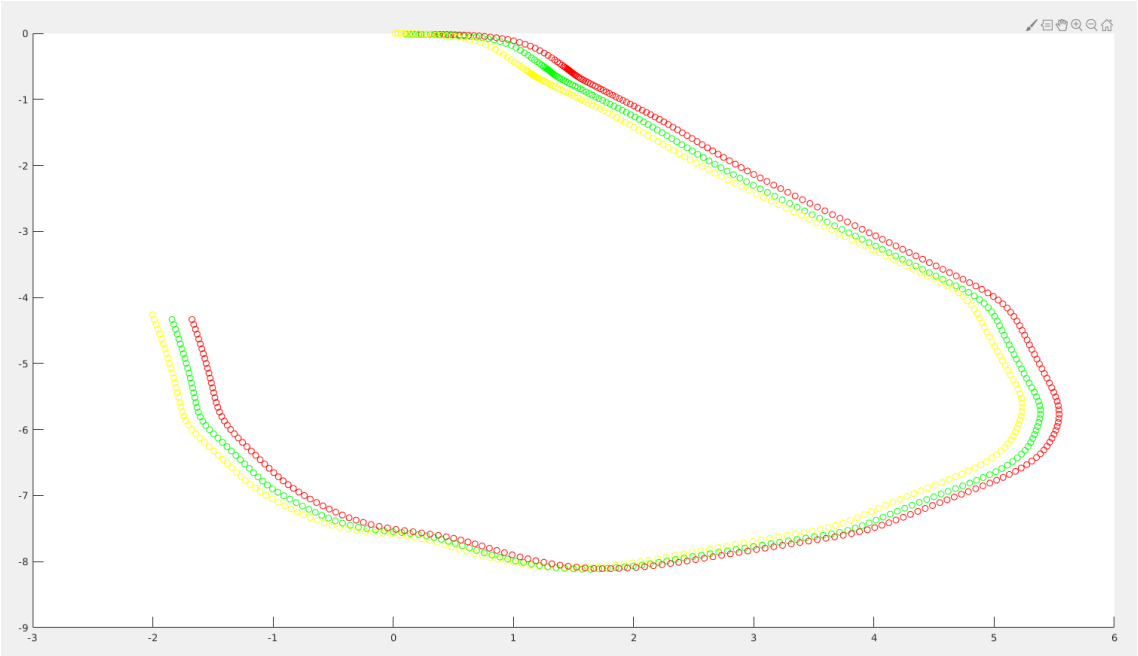


Figure 4.5: Noisy EKF measurements

In the figure 4.5 instead, 10 % of noise is added to GPS measurements to test the robustness of EKF simulating a situation similar to the real one where the GPS measurements are affected by the errors previously mentioned. From this figure it is possible to notice the robustness of the EKF even in a noisy situation. To conclude, the Extended Kalman Filter is a really useful tool for state estimation and has as key points the simplicity and computational efficiency due to the use of a multivariate Gaussian distribution. Nevertheless the most penalizing limitation is the fact that the EKF is based on an approximation state transitions and measurements using Taylor expansions. Another variation of the Kalman Filter is the Unscented Kalman Filter which performs a stochastic linearization with a weighted statistical linear regression process [61].

4.2 Collision avoidance approaches

Motion planning and collision avoidance approaches (also known as the navigation problem) compute a movement from a start configuration to a desired goal configuration that satisfies dynamic constraints and, at the same time, optimizes some aspects of the movement itself. In this way each robot is a specific entity, able to freely move in a structured environment, being able of reaching a desired point while avoiding obstacles. Collision avoidance approaches for mobile robots are usually divided in two categories: *global* and *local*. The global techniques generally assume that a complete model of the robot's environment is available and that the trajectory from the current position of the robot to the goal-point can be computed off-line [27]. Examples of global collision avoidance approaches are Voronoi graph, cell decomposition, Visibility graph and potential field methods. The main problem with these techniques arises when the goal-point is not reachable, as typically happen in a populated environment. Another problem of global motion planning systems is the inherent complexity that creates even greater problem in a simulation environment with multiple robots in which the simplicity of the algorithm and the preservation of the hardware resources is fundamental. On the contrary, local approaches make use of only a small portion of the world model to generate the robot control. The main improvement of local approaches consists in their low computational burden that allows the robot to move without problems and adapt quickly even in a fast changing scenario.

Local approaches create the path to be followed usually in two steps :

1. First the desired motion direction is determined

2. Then an appropriate combination of steering commands to accomplish the desired direction are sent to the robot

Anyway a problem that could arise is that the desired direction is not reachable to the robot for its own physical limitations. In this work three approaches have been evaluated:

1. Artificial Potential Field (APF)
2. Vector Field Histogram (VFH)
3. The *move_base* ROS Package (NavFN global planner and DWA local planner)

4.2.1 Artificial Potential Field

The Artificial Potential Field method was developed as a basis for generating smooth trajectories for both mobile and manipulator robotic systems. Many variations of the method have been developed and used [46] [48] [15] [35], but the basic idea is that the robot moves in an abstract artificial field force, in which the robot is a positive particle, as well as obstacles, and the goal has a negative charge. Thus the robot is attracted by the goal-point and repulsed from obstacles, as figure 4.6 shows.

Even if there exist a lot of versions and different implementation of this algorithm, the most used formulas are the ones reported from equation 4.5 to 4.14 [2].

$$U_{att}(q) = \frac{1}{2}\zeta\rho^2(q, q_{goal}) \tag{4.5}$$

where, $U_{att}(q)$ is the attractive potential, ζ is a positive scaling factor, $\rho(q, q_{goal})$ is the distance between the robot q and the goal q_{goal}. Since the force is nothing but

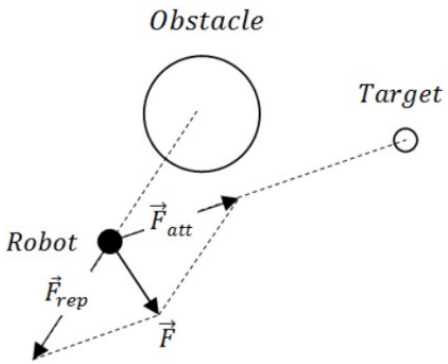


Figure 4.6: Attraction and repulsion by goal and obstacle [45]

the negative gradient of the potential function, the attractive force can easily be derived as:

$$F_{att}(q) = -\nabla U_{att}(q) = \zeta(q_{goal} - q) \tag{4.6}$$

So the attractive force is directly proportional to the distance and it becomes null as soon as the robot position equals the desired goal position. For what concerns the repulsive potential, it is described by the function in 4.7:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho(q, q_{obs})}\right)^2 & if\ \rho(q, q_{obs}) \leq \rho_0 \\ 0 & if\ \rho(q, q_{obs}) > \rho_0 \end{cases} \tag{4.7}$$

where η is a negative scaling factor, $\rho(q, q_{obs})$ denotes the shortest Euclidean distance from the robot q to the obstacle, ρ_0 is the largest impact distance of the obstacle. The negative gradient of the repulsive potential function:

$$F_{rep} = -\nabla U_{rep}(q) = \begin{cases} \eta \left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_0}\right) \cdot \frac{1}{\rho^2(q, q_{obs})} \bigtriangledown \rho(q, q_{obs}) & if\ \rho(q, q_{obs}) \leq \rho_0 \\ 0 & if\ \rho(q, q_{obs}) > \rho_0 \end{cases} \tag{4.8}$$

Equation 4.8 shows clearly that, when the robot is far from the obstacle with a distance greater than the threshold ρ_0 , the repulsive force is negligible and we set it to zero. On the contrary, the repulsive force is inversely proportional to the distance between the robot and the obstacle, creating repulsion. The applied total force to the robot will finally be: $F_{total} = F_{att}(q) + F_{rep}(q)$ which determines the robot motion. A graphical interpretation of the potential field is provided by the figure 4.7.

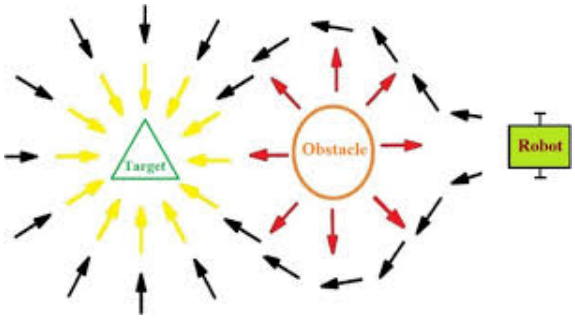


Figure 4.7: Potential field: field lines [45]

The reason why this method is so popular is because of its simplicity and effectiveness in almost every situation. Despite its utility, potential field has some intrinsic problems, for example the Local minima, that is the situation in which the robot faces to in particular spacing configuration in which the sum of repulsive and attractive force is zero, i.e:

$$||\vec{F}_{tot}|| = ||\vec{F}_{att} + \sum_{i=1}^{N} \vec{F}_{rep,i}|| = 0 \tag{4.9}$$

Anyway the Local minima problem can be overcome by determining an escaping strategy. A solution for escaping local minima is described by Wilschut [71]. According to Wilschut [71], a local minimum is identified when the following conditions

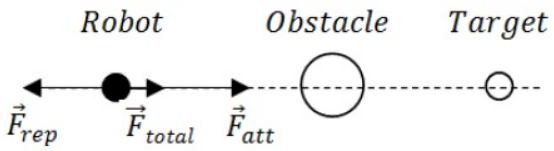


Figure 4.8: Local minima

are true, i.e. 4.10 and 4.11, where b and c are arbitrary chosen constants.

$$\frac{||\vec{F}_{tot}||}{||\sum_{i=1}^{N} \vec{F}_{rep,i}||} < b \tag{4.10}$$

$$cos(\angle\vec{F}_{att} - \angle\sum_{i=1}^{N} \vec{F}_{rep,i}) < -cos(c) \tag{4.11}$$

When a local minimum is identified, an escape force $\vec{F}_{escape}$ is introduced in the direction perpendicular to $\vec{F}_{rep}$. This steers away the robot from the local minimum since $\vec{F}_{tot}$ is not pointing to the local minimum anymore causing the robot to go around the obstacle.

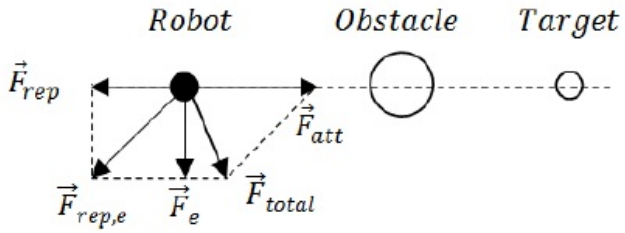


Figure 4.9: Escaping force for local minima [45]

The angle $\angle\vec{F}_{att} - \angle\sum_{i=1}^{N} \vec{F}_{rep,i}$ can be computed as:

$$\angle\vec{F}_{att} - \angle\sum_{i=1}^{N} \vec{F}_{rep,i} = acos(\frac{\vec{F}_{att}\sum_{i=1}^{N} \vec{F}_{rep,i}}{||\vec{F}_{att}|| \cdot \sum_{i=1}^{N} \vec{F}_{rep,i})} \tag{4.12}$$

One additional condition is added to assure the escape force does not steer the robot away from the target position which is explained by Wilschut [71]:

$$||\vec{p}_{tar} - \vec{p}_{r}|| > d \tag{4.13}$$

Where d is the minimum distance between the robot and the target for the escape force to be applied. Then the escaping force is evaluated as:

$$\vec{F_e} = \alpha_e \frac{1}{(\rho_s - \rho_r - \rho_m)^2} \cdot \vec{n}_{rep\perp} \tag{4.14}$$

where $\vec{n}_{rep\perp}$ is a unit vector perpendicular to the repulsive force $\vec{F_e}$ and $\alpha_e = 10^z$, with $z = log_{10}(F_{rep})$

4.2.2 Vectorial Field Histogram

The Vector Field Histogram (VFH) was originally proposed by Borenstein and Koren in 1991 as local obstacle avoidance method for mobile robot applications. In [63] the authors propose an improved version of the original technique applied on GuideCane,a robotic platform developed for blind people. This algorithm takes as input the map grid of the local environment, called histogram grid. The VFH+ works in a four stage data reduction processes in order to compute each time the new motion goals to be performed. In the first stage, the primary polar histogram is built. In this stage a specific region of the map grid around the robot called active region C_a, is mapped onto the primary polar histogram H^p. The active region has a circular shape with a diameter w_s which is set according to the specific application in which the robot is used. This region is used to create an obstacle vector. The vector direction $B_{i,j}$ is defined by the direction formed by the active cell and the robot center point (RCP) according to the following motion model:

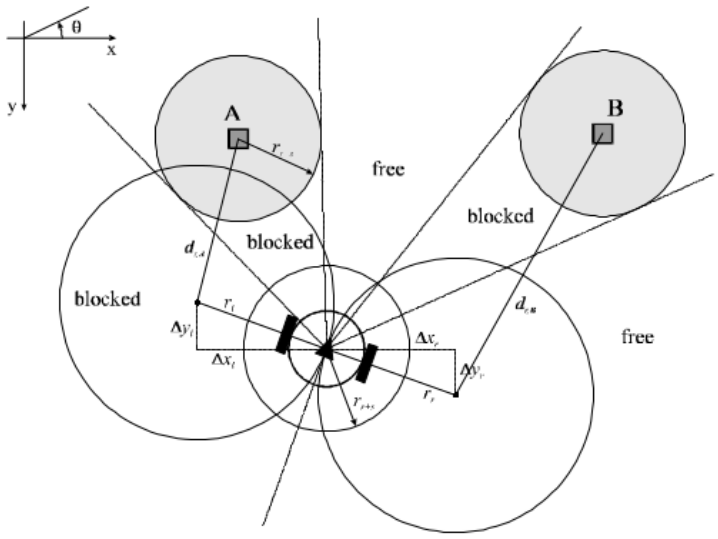


Figure 4.10: VFH Motion Model [63]

$$\beta_{i,j} = tan^{-1}(\frac{y_o - y_j}{x_i - x_o}) \tag{4.15}$$

where:

x_o, y_o : coordinates of the RCP

x_i, y_j : coordinates of the active cell $C_{i,j}$

The vector magnitude of $C_{i,j}$ is given by:

$$m_{i,j} = c_{i,j}^2 \cdot (a - bd_{i,j}^2) \tag{4.16}$$

where:

$c_{i,j}$: coordinates of the active cell $C_{i,j}$

$d_{i,j}$: distance from the active cell $C_{i,j}$ to the RCP

and the parameters a and b are chosen as:

$$a - b \cdot \left(\frac{w_s - 1}{2}\right)^2 \tag{4.17}$$

As it can be seen from formula 4.16 the vector magnitude is directly proportional to the certainty value $c_{i,j}$ and the distance $d_{i,j}$. Thus, as an obstacle is near to the robot, the vector magnitude becomes high. Having the obstacle vector, the primary polar histogram H^p can be built, with a resolution α such that $n = \frac{360^o}{\alpha}$, the number of angular sectors, is an integer. Each angular sector k corresponds to a discrete angle $\rho = k \cdot \alpha$. To get a more significant polar histogram, the obstacles in the map are enlarged of a quantity r_r equal to the radius of the robot. Moreover, in order to have a security distance, another parameter d_s is used to enlarge the obstacles cells, having as a result $r_{obst} = r_s + d_s$.

The enlarged angle is defined as:

$$\gamma_{i,j} = arcsin\frac{r_{obst}}{d_{i,j}} \tag{4.18}$$

For each sector k, the polar obstacle density is defined as:

$$H_k^p = \sum_{i,j \epsilon C_a} m_{i,j} \cdot h'_{i,j} \tag{4.19}$$

where:

$$h'_{i,j} \qquad = \qquad \begin{cases} 1 & if\ k \cdot \alpha\ \epsilon\ [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}] \\ 0 & otherwise \end{cases} \tag{4.20}$$

In the second stage, called the binary polar histogram creation, in which the aim is to avoid the steering oscillations to make the trajectory smooth and safe. To this aim, differently from the classic VFH which defines a fixed threshold τ, a two threshold selection is applied. Starting from the primary polar histogram H^p is filtered through two thresholds τ_{low} and τ_{high}, obtaining a binary polar histogram

H^b such that:

$$H^b_{k,i} = \begin{cases} 1 & if\ H^b_{k,i} > \tau_{high} \\ 0 & if\ H^b_{k,i} < \tau_{low} \\ H^b_{k,i-1} & otherwise \end{cases} \quad (4.21)$$

The third stage consists in the creation of a masked polar histogram. The basic idea under this stage is to consider the mobile robot trajectory composed of circular arcs, of curve $k = 1/r$, and straight lines. Using the definition of curve for both side of the robot we have $r_r = 1/k_r$ and $r_l = 1/k_l$. Using these definitions and with the map grid it is possible to say which sectors are blocked by obstacles. The trajectories positions for each wheel, with respect to the robot are then:

$$x_r = r_r \cdot sin(\theta) \qquad y_r = r_r \cdot cos(\theta)$$
$$x_l = -r_l \cdot sin(\theta) \qquad y_l = -r_l \cdot cos(\theta)$$

The distance from an active cell $C_{i,j}$ to the trajectory centers are:

$$d_r = \sqrt{(x_r - x(j))^2 + (y_r - y(i))^2} \quad (4.22)$$

$$d_l = \sqrt{(x_l - x(j))^2 + (y_l - y(i))^2} \quad (4.23)$$

So, an obstacle would block the right direction if $d_r^2 < (r_r + r_{obst})$ and the right one if $d_l^2 < (r_l + r_{obst})$. Then two limit angles ϕ_r and ϕ_l are computed. Furthermore a third angle, representing the backward direction with respect to the current direction of motion, is defined as $\phi_b = \theta + \pi$. The procedure to determine these angles is:

1. Determine ϕ_b. Set ϕ_r and ϕ_l equal to ϕ_b

2. For every active cell $C_{i,j}$ in the active window C_a with $c_{i,j} > \tau$:

- If $\beta_{i,j}$ is to the right of θ and to the left ϕ_r, check condition 1. If condition is satisfied, set ϕ_r equal to $\beta_{i,j}$.
- If $\beta_{i,j}$ is to the left of θ and to the right ϕ_l, check condition 2. If condition is satisfied, set ϕ_l equal to $\beta_{i,j}$.

Having ϕ_r and ϕ_l and the binary polar histogram found in the precedent stage, the masked polar histogram is found as:

$$H_k^m = \begin{cases} 0 & if\ H_k^b = 0 \quad and \quad k \cdot \alpha \epsilon \left\{[\phi_r, \theta], [\theta, \phi_l]\right\} \\ 1 & otherwise \end{cases} \tag{4.24}$$

At the last stage, the selection of the steering direction is managed. In this step, basing on the masked polar histogram that takes into account the free directions in the map, a set of possible direction candidates are selected. Then a cost function is applied to these candidates to evaluate the most opportune one. The openings in the masked polar histogram that are divided as wide and narrows. An opening is wide if the difference between its two borders is greater than s_{max} sectors, otherwise the opening is considered narrow. A narrow opening has only one candidate direction and is defined as:

$$c_n = \frac{k_r + k_l}{2} \tag{4.25}$$

For a wide opening the candidate are instead:

$$\begin{gathered} c_r = k_r + \frac{s_{max}}{2} \\ c_l = k_l - \frac{s_{max}}{2} \\ c_t = k_t \quad if\ k_t\ \epsilon\ [c_r, c_l] \end{gathered} \tag{4.26}$$

where c_t is the target direction. The candidate directions c_r and c_l make the robot follow an obstacle contour at a safe distance, while c_t leads the robot towards the target direction. To these candidates, the cost function to be applied is:

$$g(c) = \mu_1 \cdot (c, k_t) + \mu_2 \cdot (c, \frac{\theta_i}{\alpha}) + \mu_3 \cdot (c, k_{n,i-1}) \tag{4.27}$$

Where the function (c_1, c_2) is defined as:

$$(c_1, c_2) = min\left\{|c_1 - c_2|, |c_1 - c_2 - n|, |c_1 - c_2 + n|\right\}$$

μ_1, μ_2 and μ_3 are weighting parameters. Experimentally, the author in [63] shown that a good set of weights for a goal-oriented mobile robot is: $\mu_1 = 5$, $\mu_2 = 2$ and $\mu_3 = 2$.

The VFH is in general a good and simple obstacle avoidance algorithm for mobile robots. Anyway one of the main drawback is its local nature that often leads to bring the robot into dead-ends.

4.2.3 Move base

The move base node [52] is the core element of the ROS Navigation stack [53]. It consists in *global* and *local* planners to accomplish the global navigation task. This package handles two distinct costmaps, a global and a local one, that contains information that represent the projection of the obstacles in a 2D space, as well as a security inflation radius, an area around the obstacles to prevent the collision of the robot with obstacles. While the global costmap represents the whole environment (or a huge portion of it), the local costmap is, in general, a dynamic rolling window that moves in the global costmap in relation to the robot's current position. Inside

these two costmaps the global and the local planner generates a plan in the respective map. The interactions between the elements of the Navigation Stack discussed so far are shown in figure 4.11.

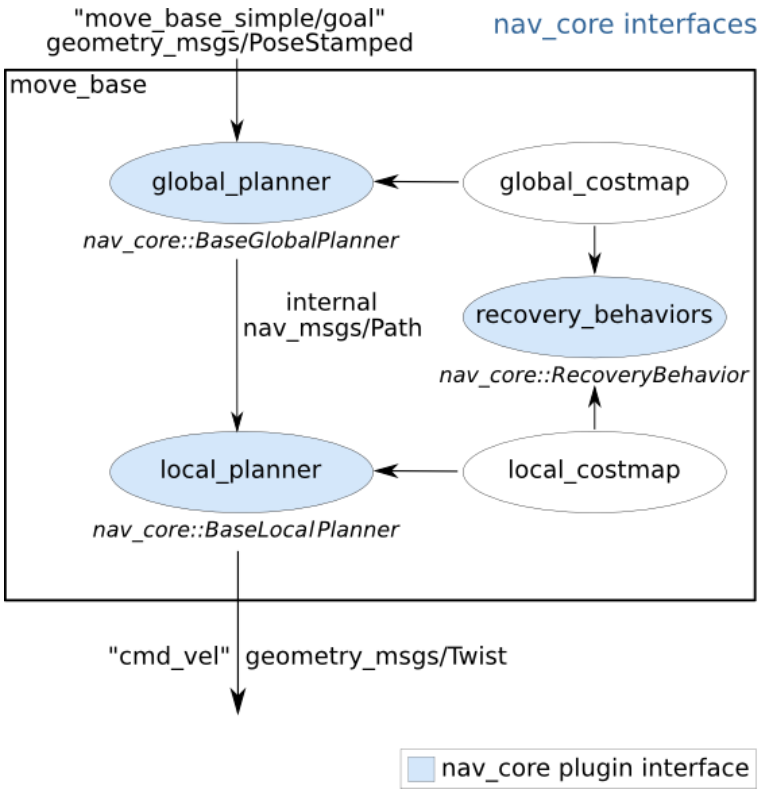


Figure 4.11: ROS Navigation Stack [52]

Both the global and local planner strategies can be defined by the user, determining the so called *plugins*. For the global planner there are three popular plugins commonly used by ROS users: the *Navfn*, a grid-based global planner that uses a navigation function to compute a path for a robot, and the *carrot_planner*, that attempts to find a legal place to put a carrot for the robot to follow by moving back along the vector between the robot and the goal point and finally *global_planner* which is similar to the *Navfn* but it has more flexible options, including [76]:

1. A^* algorithm

2. toggling quadratic approximation

3. toggling grid path

. In general, the global planner takes the current robot position and the goal and traces the trajectory of lower cost in respect to the global costmap. For what concern the choice of the local planning algorithm, popular local planner plugins are: the *dwa_local_planner*, which is the implementation of the Dynamic Window Approach, the *teb_local_planner*, that implements the Time-Elastic-Band (TEB) method for on-line trajectory implementation, and the *mpc_local_planner*, which collects a number of model predictive control (MPC) approaches. Since the local planner works in the local costmap that for definition is smaller than the global one, it allows to detect small obstacle with respect to the global costmap, because of its greater definition. The role of the local planner is to follow the path designed by the global planner avoiding obstacles. In order to synthesize the roles of the two planner it can be said that:

1. *Global motion planning*: used to create paths for a goal in the map or a far-off distance

2. *Local motion planning*: used to create paths in the nearby distances and avoid obstacles not considered with the global planner

Figure 4.12 shows all these elements of the Navigation Stack; in particular, it is possible to recognize the global costmap, which is the one in light blue with an almost circular shape, the local costmap, which is the squared blue map around the

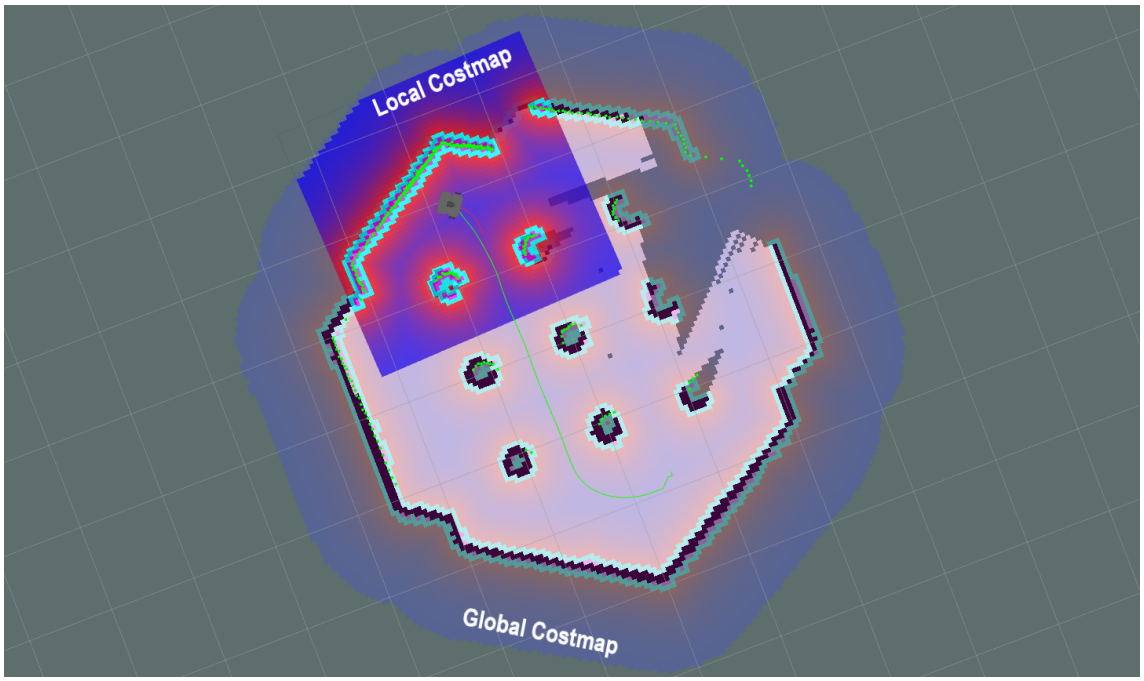


Figure 4.12: Example of ROS Navigation Stack

robot, the global motion planning trajectory which is the green line connecting the robot with the goal position, and the local planner which is the small semicircular arrow in red. Furthermore in figure 4.12 it is possible to see the so called *inflation* in both local and global costmaps, that usually it is set as the radius of the robot in order to prevent the robot collision with obstacles. When a goal is passed to the move base through, this will create a path and will try to bring the robot to that specific goal pose. The goal is achieved when the robot position is in the goal pose range with a tolerance specified by the user (usually $0.1 - 0.2\ m$). The expected robot behaviors are described in figure 4.13.

If for any reason the move_base fails in the production of the plan, a set of recovery behaviors intervenes. After each behavior completes, move_base will attempt to make a plan. If planning is successful, move_base will continue normal operation,

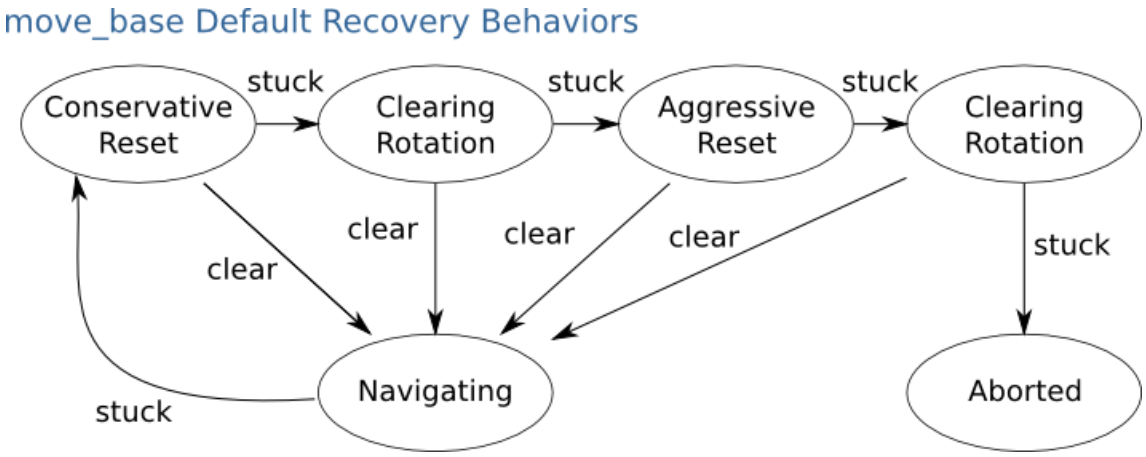


Figure 4.13: Robot behavior during a Move Base Navigation [52]

otherwise, the next recovery behavior in the list is executed. The execution practice in a stuck situation is divided in the following steps: first, obstacles outside of a user-specified region will be cleared from the robot's map. Next, if possible, the robot will perform an in-place rotation to clear out space. If this too fails, the robot will more aggressively clear its map, removing all obstacles outside of the rectangular region in which it can rotate in place. This will be followed by another in-place rotation. If all this fails, the robot will consider its goal infeasible and notify the user that it has aborted.

The overview of a typical system running the navigation stack is provided in figure 4.14.

The ROS move_base node requires a localization system to work. Usual choices for it is through the Gmapping ROS node or with the AMCL (Adaptive Monte Carlo localization). The former is able to localize the robot in unknown environment performing SLAM, while the latter can only work in a known map and it's based on Monte Carlo localization approach. In this work, even if the EKF has been

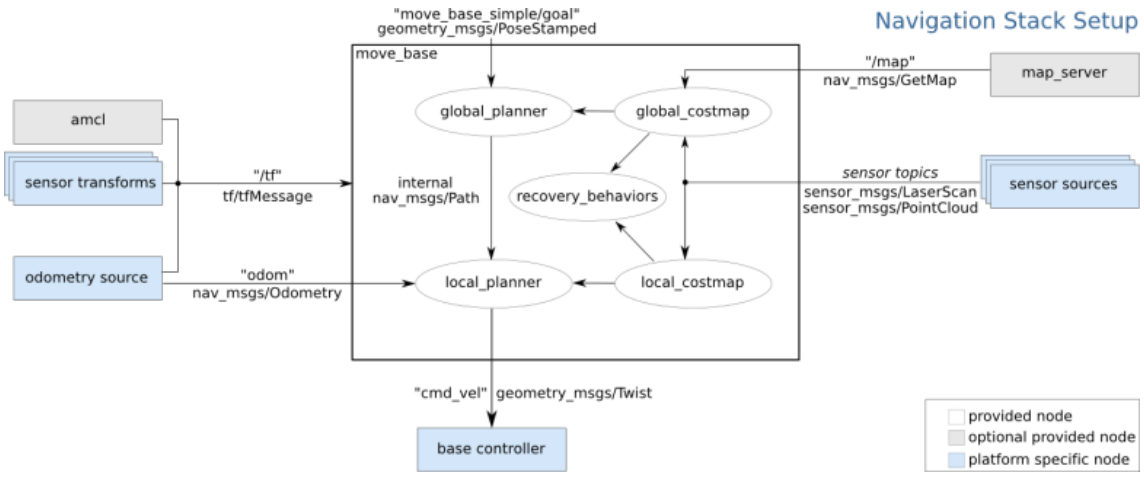


Figure 4.14: move_base configuration

created as mentioned in Section 4.1.1, the Gmapping has been preferred to be used as localization for the move_base because of simplicity.

Global Planner: Navfn

"Navfn provides a fast interpolated navigation function that can be used to create plans for a mobile robot" [56]. The robot shape is assumed to be circular and the planner works on a costmap trying to find a minimum cost plan from a start point to an end point in the grid. The ROS Navfn is based on [12]. The navigation function is computed with Dijkstra's algorithm, which is an algorithm for finding the shortest paths between nodes in a graph [68]. The algorithm was concieved by Edsger W. Dijkstra in 1956 and then published in 1959 in [17]. It generates a SPT (shortest path tree) with given source as root and maintains two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, a vertex in the other set (set of not yet included) with a minimum distance from the source is found.

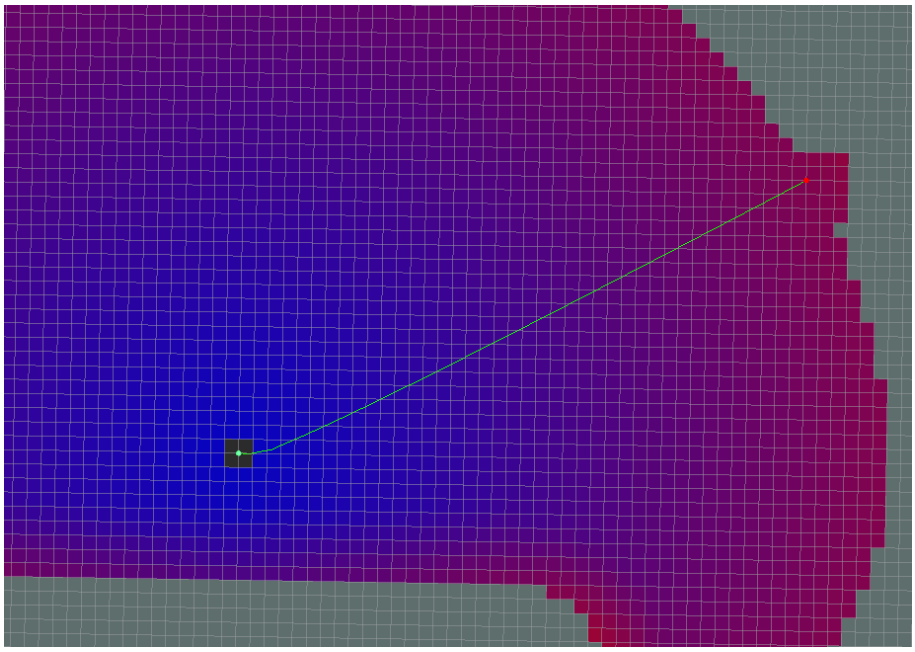


Figure 4.15: Dijkstra path [52]

Considering an initial node and the distance between one node and the starting one, the algorithm performs the following steps [68]:

1. find all the unvisited node and add them in a list called *unvisited set*.

2. Set to zero initial node distance and to infinity for all other nodes. Set the initial node as *current node*.[8]

3. For the *current node* calculate the tentative distance with all it neighbors. Compare the newly calculated tentative distances to the current assigned value and assign the smaller one.

4. Once all the neighbors of the *current node* are visited, this will considered visited. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route be-

tween two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm is finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

The Dijkstra's pseudo code is depicted in Algorithm 5 [68].

Algorithm 5 Dijkstra's Pseudo-Code

```
1: function DIJKSTRA(Graph, Source):
2:     create vertex set Q
3:     for all vertex v in Graph do
4:         dist[v] ← INFINITY;
5:         prev[v] ← UNDEFINED;
6:         add v to Q;
7:     dist[source] ← 0
8:     while Q is not empty : do
9:         u ← vertex in Q with min dist[u]
10:        remove u from Q
11:        for all neighbor v of u : do
12:            only v that are still in Q
13:            alt ← dist[u] + length(u, v)
14:            if alt < dist[v] : then
15:                dist[v] ← alt
16:                prev[v] ← u
17: return dist[ ], prev[ ]
```

The use of Dijkstra's algorithm as global planner for mobile robots is possible converting the problem into a graphic search method using the information of a

grid cell map. This method first considers nodes reachable by the robot, called *free spaces* and assigns a cost to each of them. From the starting point, this value is increased basing on the number of nodes the robot will pass through to reach each node [41].

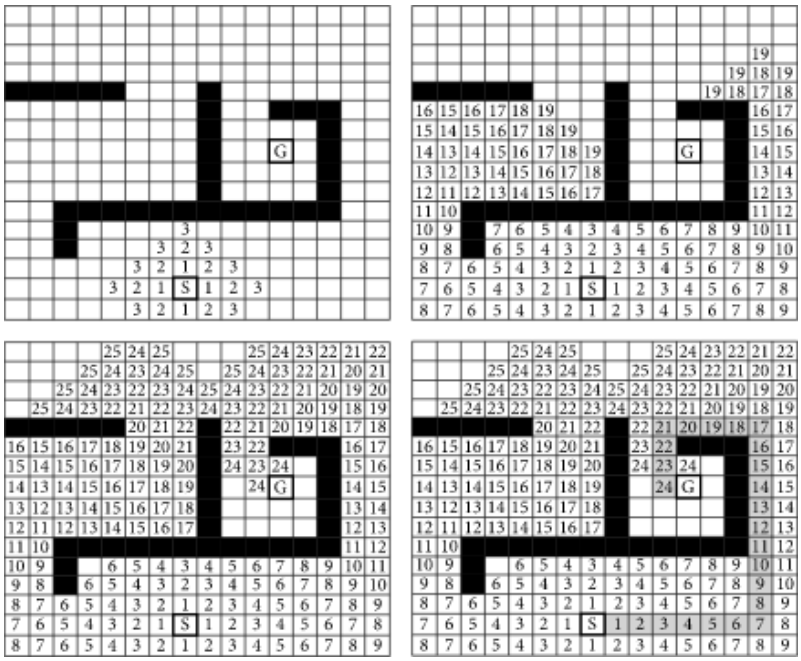


Figure 4.16: Dijkstra method to find the optimal path

Local Planner: Dynamic Window Approach

As suggested by the name, this approach makes use of *dynamic windows* in order to reduce search space into the reachable velocities within a predefined time interval. The created dynamic window only admissible velocities yielding a trajectory on which the robot is able to stop are considered. Among the admissible velocities, a combination of transnational and rotational velocities is computed in such a way an objective function is maximized. Let's define the kinematic configuration of the

robot as the triplet $\langle x, y, \theta \rangle$ where x and y describes the position of the robot and θ is its heading. Considering the translational ($v(t)$) and rotational velocity ($\omega(t)$) of the robot, it can be say that [27]:

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n} v(t) \cdot cos(\theta(t))dt \tag{4.28}$$

$$y(t_n) = y(t_0) + \int_{t_0}^{t_n} v(t) \cdot sin(\theta(t))dt \tag{4.29}$$

In order to take into account only the reachable point in the window, we have to consider that $v(t)$ and $\theta(t)$ cannot be chosen arbitrarily, but they depends on the initial value at time zero and on the admissible acceleration of the robot. Then the equation can be rewritten as:

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n} (v(t_0) + \int_{t_0}^{t} \dot{v}(\hat{t})d\hat{t}) \cdot cos(\theta(t_0) + \int_{t_0}^{t} (\omega(t_0) + \int_{t_0}^{\hat{t}} \dot{\omega}(\widetilde{t})d\widetilde{t})d\widetilde{t})dt \tag{4.30}$$

Considering that we are dealing with a digital system (microprocessor of the robot) and after some simplification, the above equation can be written as:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} (v_i + cos(\theta(t_i) + \omega_i \cdot (\hat{t} - t_i))d\widetilde{t}) \tag{4.31}$$

which by solving the integral, can be simplified to:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1} (F_x^i(t_{i+1})) \tag{4.32}$$

where

$$F_x^i(t) = \begin{cases} \frac{v_i}{\omega_i}(sin(\theta(t_i)) - sin(((\theta(t_i)) + \omega_i \cdot (t - t_i))) & if\ \omega_i \neq 0 \\ v_i cos(\theta(t_i)) \cdot t & if\ \omega_i = 0 \end{cases} \tag{4.33}$$

Similarly, for the y-coordinates equations are:

$$y(t_n) = y(t_0) + \sum_{i=0}^{n-1} (F_y^i(t_{i+1})) \tag{4.34}$$

where

$$F_y^i(t) = \begin{cases} -\frac{v_i}{\omega_i}(cos(\theta(t_i)) - cos(((\theta(t_i)) + \omega_i \cdot (t - t_i))) & if\ \omega_i \neq 0 \\ v_i sin(\theta(t_i)) \cdot t & if\ \omega_i = 0 \end{cases} \tag{4.35}$$

In the dynamic window approach the search for commands controlling the robot is carried out directly in the space of velocities. The dynamics of the robot is incorporated into the method by reducing the search space to those velocities which are reachable under the dynamic constraints. After the first skimming of the set of velocities, the definitive velocity is chosen in such a way to maximize an objective function. A nutshell of the dynamic window approach is shown in the next page.

1. **Search space:** The search of the possible velocities is reduced in three steps:

   (a) **Circular trajectories:** The dynamic window approach considers only circular trajectories (curvatures) uniquely determined by pairs (v, ω) of translational and rotational velocities. This result in a two dimensional velocity search space.

   (b) **Admissible velocities:** The restriction to admissible velocities ensures that only safe trajectories are considered. A pair (v, ω) is considered admissible, if the robot is able to stop before it reaches the closes obstacle on the corresponding curvature.

   (c) **Dynamic window:** The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the limited accelerations of the robot.

2. **Optimization:** The objective function:

$$G(v, \omega) = \sigma(\alpha \cdot angle(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega))$$

   is maximized. With respect to the current position and orientation of the robot this function trades off the following aspects:

   (a) **Target heading:** *angle* is a measure of progress towards the goal location. It is maximal if the robot moves directly towards the target.

   (b) **Clearance:** *dist* is the distance to the closest obstacle on the trajectory. The smaller the distance to an obstacle the higher is the robot's desire to move around it

   (c) **Velovity:** *vel* is the forward velocity of the robot and support fast movements.

   The function σ smoothes the weighted sum of the three components and results in more side clearance from obstacles

Then the algorithm draws the dynamic window in which only the points corresponding to the reachable velocities are contained, in order to take into account the accelerations that the robot motors can provide in a certain instant.Let's call the search space inside the window V_r.

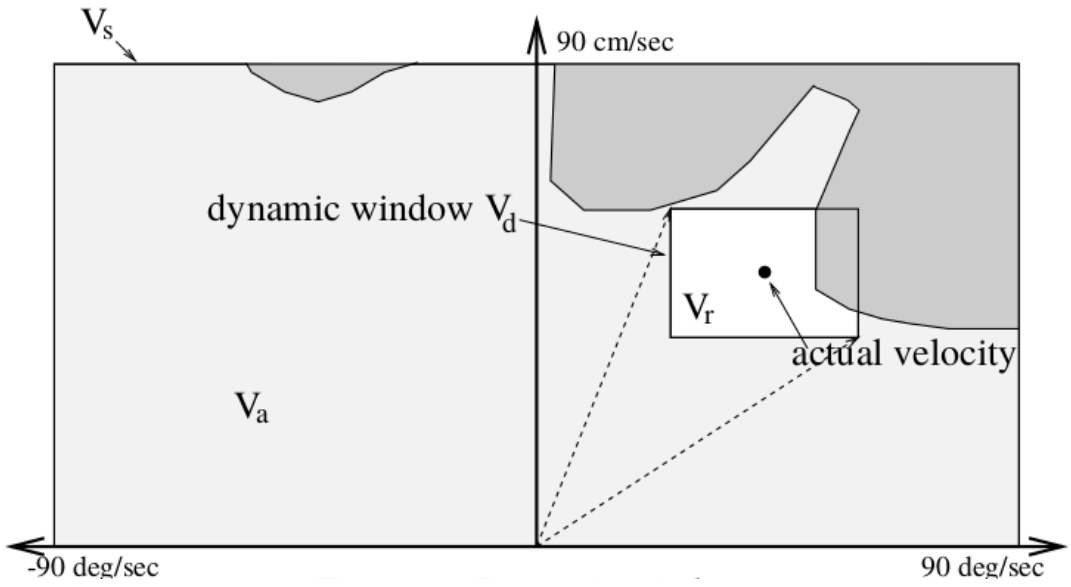


Figure 4.17: Dynamic Window [27]

The dynamic window is centered around the actual velocity and the extensions of it depend on the accelerations that can be exerted. Once determined V_r, the objective function $G(x, \omega)$ is computed over V_r as [27]:

$$G(v, \omega) = \sigma(\alpha \cdot angle(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot velocity(v, \omega)) \tag{4.36}$$

which takes into account the target heading, clearance and velocity. In the objective function we find:

- $angle(v, \omega)$, which measures how much the robot is aligned with the target direction

- $dist(v, \omega)$, which is the distance to the closest obstacle that intersects with the curvature. If there are not obstacles, then a high value is assigned to $dist(v, \omega)$

- $velocity(v, \omega)$, which measures the progress of the robot on the corresponding trajectory

These components are normalized in the range of $[0, 1]$ and then are weighted through the parameters α, β and γ.

4.3 Map Building

The creation of a map with a mobile robot is a challenging problems for many reasons. However, the two main problems are described in [61]:

1. The hypothesis space, that is the space of all the possible maps is really huge. Maps are described in a continuous space, so the space of all maps is infinite. Even discretizing the space, it can be shown that map cannot be described with less then 10^5 variables.

2. The map building is strictly related with the accuracy of the localization estimation and the knowledge of the initial pose of the robot.

Anyway the difficulty in the map building is not equal for all the environmental situations. Parameters that affect the hardness of the mapping problem are [61]:

- **Size**. The vastness as could be easily be understood is a parameter than increases the complexity of this job. The measurement of the vastness has to be compared with the robot's perceptual range in order to have a measure of the level of complexity.

- **Noise in perception and actuation**. Mapping is also really affected by the level of noise the robot's sensors are subjected to. Being sure about the sensor measurements makes things easier, but in a real world this condition is never verified

- **Perceptual Ambiguity**. That is the difficulty in understanding the correspondence between the sensed information about a same area of the map traversed at different points in time.

- **Cycles**. Mapping a corridor is simpler than mapping a path containing loops, where measurements about position can become less precise.

In the field of mobile robotics, different model for representing the environment have been introduced over the years. The most common ones are *feature maps*, *geometric maps*, and *grid maps* [58]. The feature maps creates the map representation through the features extracted by the robot sensors. Usually the features consist in lines and corners; an example of a feature map is depicted in figure 4.18.

Figure 4.18: Feature Map

Geometric map instead, represents all the obstacles detected by the robot sensors as geometric entities, like circles or polygons.

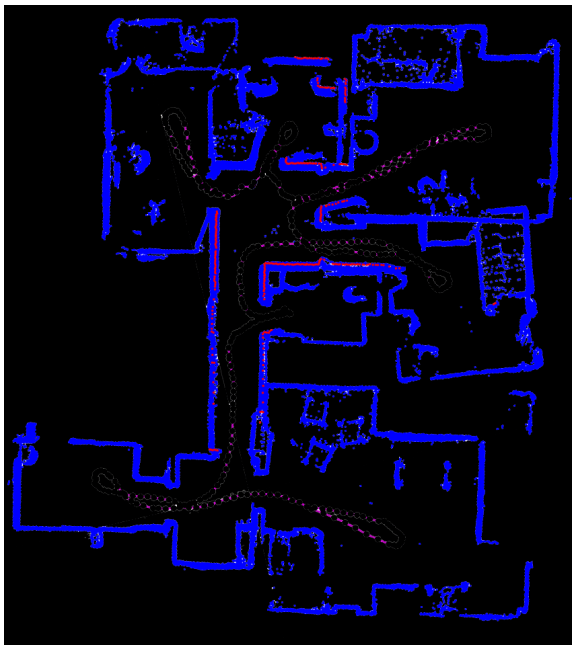


Figure 4.19: Geometric Map

Finally, occupancy grid maps that represent environment by a grid.

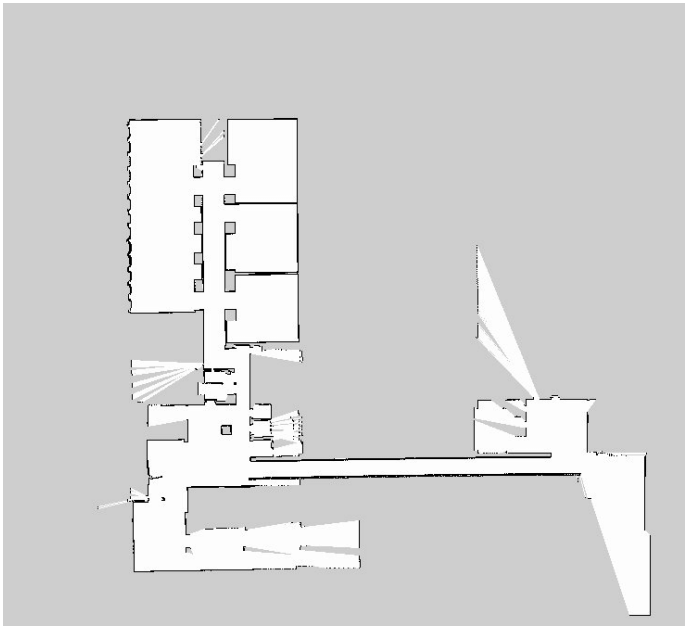


Figure 4.20: Occupancy Grid Map

The model used in this work to generate the map is the occupancy grid map. The reasons for this choice are described in the following section.

4.3.1 Occupancy Grid Mapping

Occupancy grids were first proposed by H. Movarec and A. Elves in 1985 [39] and today is one of the most used mapping algorithm because of its robustness from uncertain sensor measurements, assuming known robot pose. The occupancy grid defines the map as a field of random variables, arranged in a equally spaced grid, each one with a binary value corresponding to the occupancy of the location it covers.

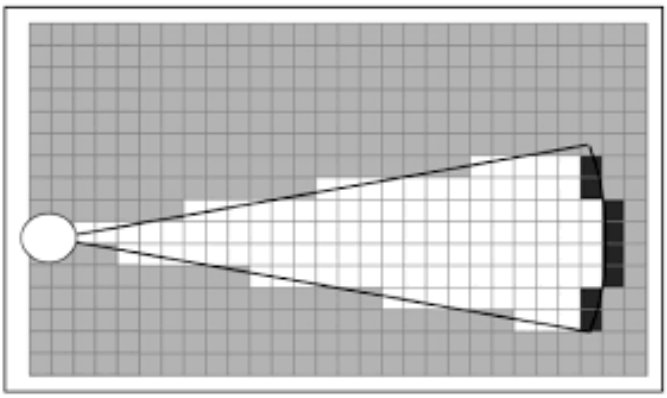


Figure 4.21: Grid Mapping Example

The algorithm takes into account a sequence of sensor observations $z_{i:t}$ obtained by the robot at the positions $x_{i:t}$ and seeks to maximize the occupancy probability for the grid map [61]. The goal of an occupancy mapping algorithm is to estimate the posterior probability over maps given the data:

$$p(m|z_{1:t}, x_{1:t}) \tag{4.37}$$

where m is the map, $z_{1:t}$ is the set of all measurements until time t and $x_{1:t}$ is the path followed by the robot. An occupancy grid mapping algorithm provides a partitioned configuration of the map m such that $m = \{m_i\}$ where m_i is the $i - th$ grid cell of the map. At each grid cell it is associated a binary value such that "1" denotes occupied cells and "0" denotes free cells. Then 4.37 can be decomposed as $p(m_i|z_{1:t}, x_{1:t})$, for each grid cell m_i. This decomposition is useful but it does not allow to represent the dependencies between neighboring cells [61]. Thus the following approximation is needed:

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t}) \tag{4.38}$$

The formula 4.38 is a binary estimation problem with static state which can be solved with a binary Bayes filer. A binary estimation problem with static state is a problem where states don't change over time and so its belief is a function only of the measurement:

$$bel_t(x) = p(x|z_{1:t}, u_{1:t}) = p(x|z_{1:t}) \tag{4.39}$$

Considering $p(\neg x) = 1 - p(x)$ and defining the *log odds* $l(x) := log\frac{p(x)}{1-p(x)}$ the final equation of the Bayes filter in log odds form, after some equations is:

$$l_t(x) = log\frac{p(x|z_t)}{1 - p(x|z_t)} - log\frac{p(x)}{1 - p(x)} + l_{t-1}(x) \tag{4.40}$$

The pseudo code of this estimation problem is shown in Algorithm 6 [61].

Algorithm 6 Occupancy Grid Pseudo-Code

1: **function** ALGORITHM OCCUPANCY_GRID_MAPPING($\{l_{t-1,i}\}$, x_t, z_t):
2: **for all** *cells* m_i **do**
3: **if** m_i *in perceptual field of* z_t **then**
4: $l_{t,i} = l_{t-1,i} + \textbf{inverse\_sensor\_model}(m_i, x_t, z_t) - l_0$;
5: **else**
6: $l_{t,i} = l_{t-1,i}$;
7: **endif**
8: **endfor**
9: **return** $\{l_{t,i}\}$

As it can be seen in the Algorithm 6, the Occupancy grid algorithm makes use of *log odds* representation of occupancy:

$$l_{t,i} = log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})} \tag{4.41}$$

This, inversely, can be written as:

$$p(m_i|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + exp\{l_{t,i}\}} \tag{4.42}$$

Moreover, the function *inverse_sensor_model* used in the algorithm 6 is the inverse measurement of $p(m_i|z_t, x_t)$ in its log odds forms:

$$inverse\_sensor\_model(m_i|z_t, x_t) = log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \tag{4.43}$$

The reason why this algorithm uses the log odds is to avoid numerical instabilities due to probabilities near zero or one.

Figure 4.22: Map through occupancy grid representation

4.3.2 Gmapping

To build the maps, the ROS gmapping package has been used [51]. This package, as mentioned in the ROS gmapping documentation, provides laser-based SLAM. This package does not just create a map of the environment surrounding a robot, but it also estimates the robot pose inside this map. Gmapping employs a Particle Filter (PF) called Rao-Blackwellized Particle Filter (RBPF), which is a technique for model-based estimation. RBPF used as solution of SLAM has been introduced by Murphy, Doucet and colleagues [18], [44]. As discussed in his work, Murphy affirms that the aim of Rao-Blackwellized Particle Filter in the SLAM context is to estimate the posterior probability $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$ about the map m and the trajectory $x_{1:t} = x_1, ..., x_t$ of the robot given the observations $z_{1:t}$ and the odometry measurements $u_{1:t-1}$ from the robots. The estimation $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$ is defined as:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{4.44}$$

Having this formula and assuming that we are mapping with known poses, then $p(m|x_{1:t}, z_{1:t})$ is known.

For what concerns $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ the computation is more difficult. For this reason a particle filter is applied in which each particle represents a potential trajectory of the robot. Usually the sampling importance resampling (SIR) filter [32] is used. In particular the Rao-Blackwellized SIR filter makes use of the sensor observations and the odometry readings when they are available. It updates the set of samples that represents the posterior about the map and the trajectory of the vehicle through four steps:

1. **Sampling**: the generation of particles at time t x_t^i is obtained from the generation x_{t-1}^i

2. **Importance Weighting**: a weight ω_t^i is assigned to each particle according to the following formula:

$$\omega_t^i = \frac{p(x_{1:t}^i|z_{i:t}, u_{1:t-1})}{\pi(x_{1:t}^i|z_{i:t}, u_{1:t-1})} \tag{4.45}$$

3. **Resampling**: particles are resampled according to their importance weight so that a less amount of particles will result important for the approximation of the distribution. After the resampling all the particles have the same weight.

4. **Map Estimation**: for each particle the corresponding map estimate $p(m^i|x_{1:t}^i, z_{1:t})$ is computed based on the trajectory $x_{1:t}^i$ of the particle and the history of observations $z_{1:t}$

Anyway as it is described the problem becomes really inefficient as the length of the trajectory, and so the observations grow over time. Then a reformulation is

proposed in [19], in which:

$$\pi(x_{1:t}|z_{1:t}, u_{1:t-1}) = \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t-1}) \cdot \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-2}) \tag{4.46}$$

and

$$\begin{aligned}\omega_t^i = \frac{p(x_{1:t}^i|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^i|z_{1:t}, u_{1:t-1})} &= \frac{\eta \cdot p(z_t|x_{1:t}^i, z_{1:t-1}) \cdot p(x_t^i|x_{t-1}^i, u_{t-1})}{\pi(x_t^i|x_{1:t-1}^i, z_{1:t}, u_{1:t-1})} \cdot \frac{p(x_{1:t-1}^i|z_{1:t-1}, u_{1:t-2})}{\pi(x_{1:t-1}^i|z_{1:t}, u_{1:t-2})} \\ &\propto \frac{p(z_t|m_{t-1}^i, x_t^i) \cdot p(x_t^i|x_{t-1}^i, u_{t-1})}{\pi(x_t|x_{1:t-1}^i, z_{1:t}, u_{1:t-1})} \cdot \omega_{t-1}^i\end{aligned} \tag{4.47}$$

where $\eta = \dfrac{1}{p(z_t|z_{1:t-1}, u_{1:t-1})}$ is a normalization factor. The authors in [32] describe techniques to compute accurate proposal distributions, and to adaptively determine when it is necessary to resample in order to improve the mapping technique. First of all, according to the optimal choice proposal distribution suggested in [16], it is possible to write:

$$p(x_t|m_{t-1}^i, x_{t-1}^i, z_t, u_t) = \frac{p(z_t|m_{t-1}^i, x_t) \cdot p(x_t|x_{t-1}^i, u_t)}{\int p(z_t|m_{t-1}^i, x') \cdot p(x'|x_{t-1}^i, u_t)dx'} \tag{4.48}$$

Furthermore the author in [32] asserts that the use of the odometry motion model $p(x_t|x_{t-1}, u_t)$ as the proposal distribution can be sub optimal when used with a mobile robots equipped with laser range finder. This because due to the accuracy of the laser range finder, the likelihood function is extremely peaked.
A graphical interpretation is given by the figure 4.23:

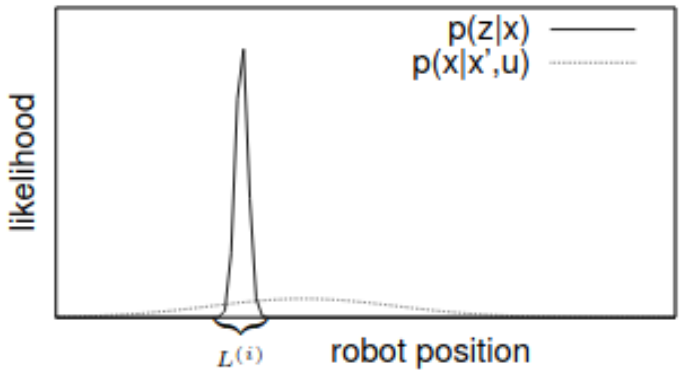


Figure 4.23: Robot Position vs Likelihood [32]

Thus the author proposes to approximate $p(x_t|x^i_{t-1}, u_t)$ by a constant k within the interval L^i defined as:

$$L^i = \left\{ x \mid p(z_t|m^i_{t-1}, x) > \varepsilon \right\} \tag{4.49}$$

Under this approximation, the equation 4.48 can be written as:

$$p(x_t|m^i_{t-1}, x^i_{t-1}, z_t, u_t) \simeq \frac{p(z_t|m^i_{t-1}, x_t)}{\int_{x' \epsilon L^i} p(z_t|m^i_{t-1}, x') \cdot dx'} \tag{4.50}$$

Furthermore, the author approximates the distribution around the maximum of the likelihood function as a Gaussian function:

$$p(x_t|m^i_{t-1}, x^i_{t-1}, z_t, u_t) \simeq \mathcal{N}(\mu^i_t, \Sigma^i_t) \tag{4.51}$$

For each particle i, parameters μ^i_t, Σ^i_t can be determined as:

$$\mu^i_t = \frac{1}{\eta} \cdot \sum_{j=1}^{k} x_j \cdot p(z_t|m^i_{t-1}, x_j) \tag{4.52}$$

$$\Sigma^i_t = \frac{1}{\eta} \cdot \sum_{j=1}^{k} p(z_t|m^i_{t-1}, x_j) \cdot (x_j - \mu^i_t) \cdot (x_j - \mu^i_t)^T \tag{4.53}$$

where $\eta = \sum_{j=1}^{k} p(z_t|m_{t-1}^i, x_j)$ is a normalizer.

Furthermore, without going into the mathematical process, the final formula for the importance weights ω^i is approximated as:

$$\omega_t^i = \omega_{t-1}^i k\eta \tag{4.54}$$

The proposal distribution described so far allows a robot with laser range finder good results in terms of particle accuracy.

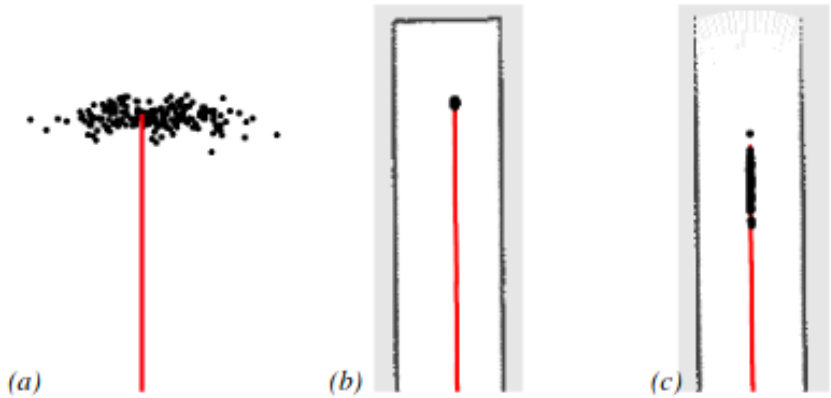


Figure 4.24: In a featureless open space the proposal distribution is the raw odometry motion model (a). In a dead end corridor the particles uncertainty is small in all of the directions (b). In an open corridor the particles are distributed along the corridor (c). [32]

Finally, the authors in [32] propose a selective resampling step, proposed in [40] where the so-called effective number of particles N_{eff} is defined as:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\omega^i)^2} \tag{4.55}$$

N_{eff} can be regarded as a measure of the dispersion of the importance weights and through the formula 4.55 it is possible to pronounce on the efficiency of the particle set approximation of the true posterior.

The technique proposed by [32] is to resample when N_{eff} gets lower of a certain threshold, that is set as $N/2$ where N is the number of particles.

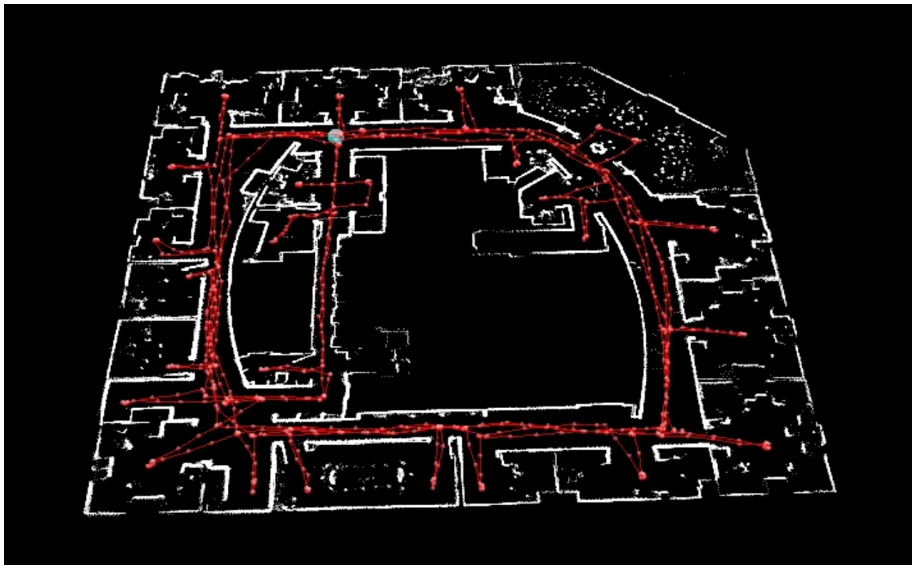


Figure 4.25: Particle mapping a Lab

As said so far, the gmapping more that just create a map of the environment, performs the Simultaneous Localization and Mapping (SLAM) it is also used as localization system [33] by the move_base node.

Costmap

The costmap is a map of the environment that divides the space into cells and assigns a cost to each cell. Each cell can be free, occupied or unknown, as for the occupancy grid representation. Anyway, the peculiarity of this map is to assign a further information around the occupied cells to which it is assigned a "lethal cost" and an inflation around the cell is applied. The inflation is the process of

propagating cost values out from occupied cells that usually decays as [25]:

$$e^{-k\cdot(d_{obst}-r_{infl})} \tag{4.56}$$

Where k is a cost scaling factor, d_{obst} is the distance from obstacle and r_{infl} is the inflation radius.

The inflation radius r_{infl}, i.e., the radius to which the cost scaling function is applied, is set by the user and usually it is choose as at least the radius of the robot, in order to prevent a collision with obstacles. Then the cost scaling factor k is just a parameter that sets the scaling factor that applies over the inflation in order to obtaining a more aggressive or conservative behavior near obstacles. Finally the distance from obstacle d_{obst} is the minimum distance dividing the robot with an obstacle. According to the definition of the inflation radius, five specific symbols for costmap values are defined [50]:

- "Lethal" cost means that there is an actual (workspace) obstacle in a cell. So if the robot's center were in that cell, the robot would obviously be in collision.

- "Inscribed" cost means that a cell is in a position with a distance, from the nearest obstacle, lower than the robot's inscribed radius. So the robot is certainly in collision with some obstacle if the robot center is in a cell that is at or above the inscribed cost.

- "Possibly circumscribed" cost is similar to "inscribed", but using the robot's circumscribed radius as cutoff distance. Thus, if the robot center lies in a cell with this mark, the robot may collide with an obstacle. However, since we are evaluating the circumscribed radius, the collision depends on the orientation of the robot. The term "possibly" is used because it might be that it is not

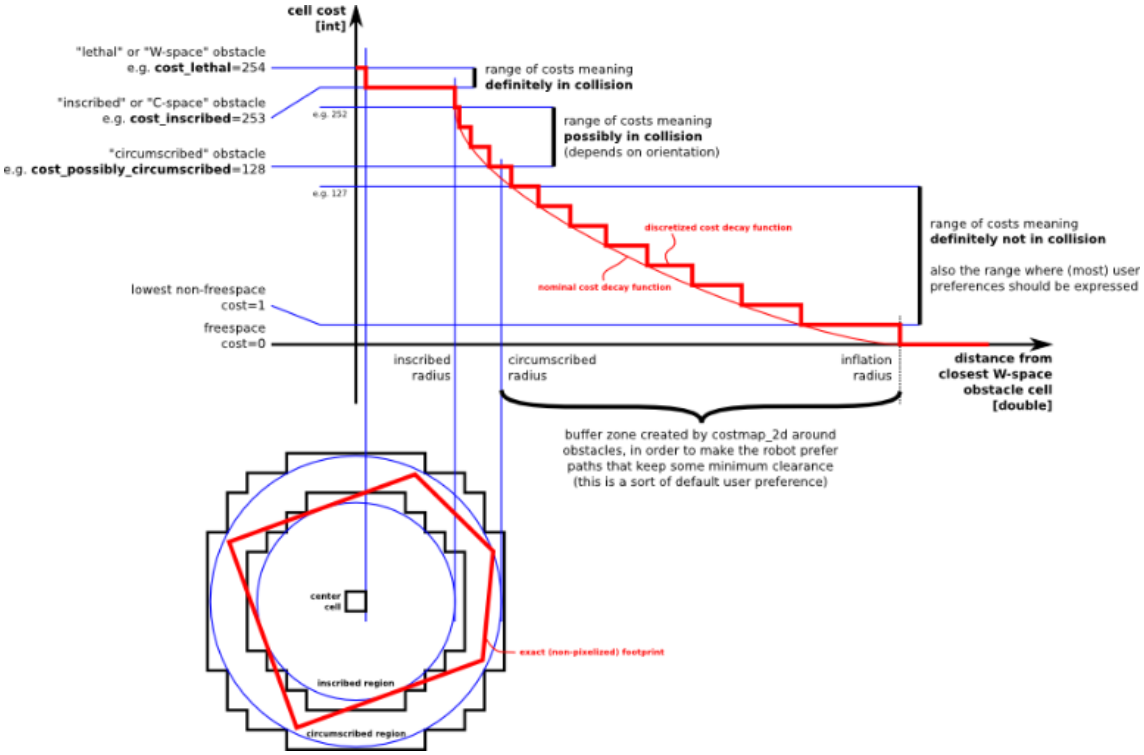


Figure 4.26: Inflation [50]

really an obstacle cell, but some user-preference, that put that particular cost value into the map. For example, if a user wants to express that a robot should attempt to avoid a particular area of a building, they may inset their own costs into the costmap for that region independent of any obstacles.

- "Free space" cost is assumed to be zero, and it means that there is nothing that should keep the robot from going there.

- "Unknown" cost means there is no information about a given cell. The user of the costmap can interpret this as they see fit.

- All other costs are assigned a value between "Freespace" and "Possibly circumscribed" depending on their distance from a "Lethal" cell and the decay

function provided by the user.

In figure 4.27 there is an example of a costmap, in which red cells represent the obstacles, the blue cells have a cost defined using the inflation radius of the robot, as previously described, and the red polygon is the footprint of the robot.

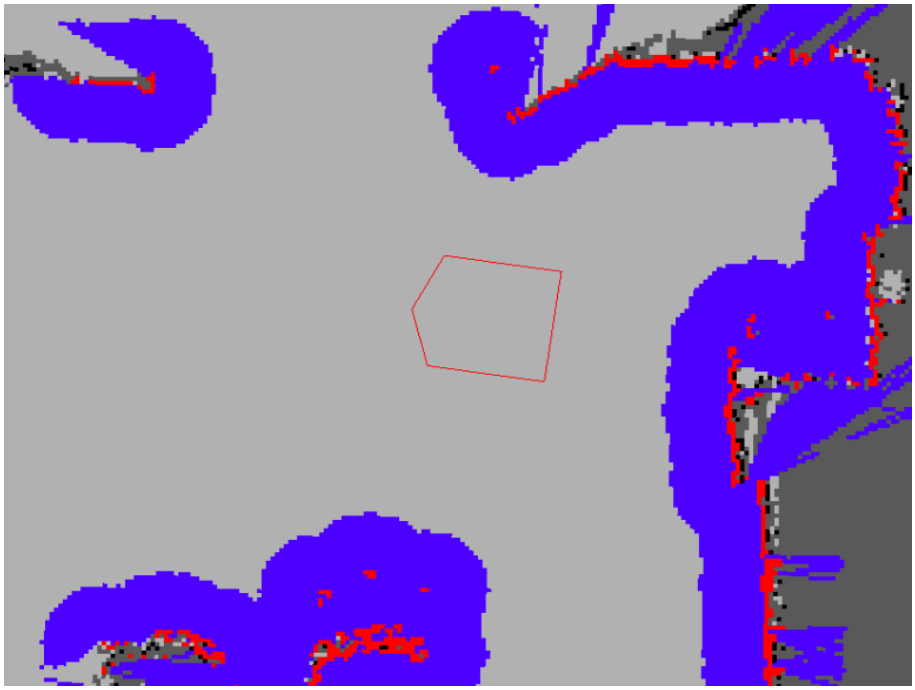


Figure 4.27: Costmap example [50]

4.4 Exploration

"Exploration is the problem of controlling a robot so as to maximize its knowledge about the external world" [61]. Given all the algorithms explained so far the robot is able to localize itself, move autonomously and map the environment surrounding the robot. The next question to answer to is: "Where should the robot go to increase its knowledge of its workspace?". Exploration problems are usually divided in three main cases [61]. The first one is the exploration of a static map, in which in the case of occupancy grid map , as the case of this thesis, the robot tries to maximize the cumulative information about each grid cell. An other exploration problem is the mapping of a dynamic environment populated of moving subjects. This is the case of finding a person in a known environment that can also moves in the environment making necessary for the robot to visit the map areas multiple times. Finally, another exploration type is the exploration of a map gaining as much information as possible in order to increase robot's information about its own pose, also called *active localization*. Even if a really simple choice to visit an unknown map could be to assign random-goal to a robot until the full map is available, this would lead to a really inefficient and non-deterministic approach. To this aim many researches over the years proposed always more efficient techniques. In [61] the authors explain some basic probabilistic exploration algorithms, that are:

- **Information Gain**, in which a specific formula is in charge of evaluating the expected information gain associated with an action u in belief b:

$$I_b(u) = H_p(x) - E_z[H_b(x'|z,u)] \tag{4.57}$$

H_p is the *entropy* and can be computed as:

$$H_p(x) = -\int p(x) \cdot logp(x)dx \qquad \left(or \ -\sum_x p(x)logp(x)\right) \tag{4.58}$$

and $H_b(x'|z,u)$ is the *conditional entropy*, such that:

$$H_b(x'|z,u) = -\int B(b,z,u)(x')logB(b,z,u)(x')dx' \tag{4.59}$$

- **Greedy Techniques**, that treats the exploration problem as a decision theoretic problem, takes into account both the information gain and the cost of applying a control action u in a state x. In particular the exploration of the optimal exploration for the belief b is the one that maximizes the following function:

$$\pi_b(b) = \underset{u}{argmax}\ \alpha \underbrace{(\ H_p(x) - E_z[H_b(x'|z,u)])}_{expected\ information\ gain} + \underbrace{\int r(x,u)b(x)dx}_{expected\ cost} \tag{4.60}$$

- **Monte Carlo Exploration**, that replaces the integrals in the greedy technique by sampling it. This in order to make the computational burden of the greedy approach faster. Anyway since in a real situation a robot gets many measurements z coming from, for example, lidars, the approximation introduced by this technique could not be enough to overcome the computational problems.

Another famous technique in the autonomous exploration area is the **topological exploration method**. This technique considers the world defined as an embedding on undirected graph G such that $G = (V, E)$, in which V are a set of vertices and E the edges that are enumerated in a systematic way. The robot moves from one vertex to another traversing an edge. In [20], author describes an algorithm for

exploring a graph using k markers. Every time a new vertex is encountered, it is added to the "explored graph" an its outgoing edges are added to a set of edges describing the edges to be traversed to reach unexplored areas.

In this work, the exploration technique proposed is the so called *Frontier Detection*.

4.4.1 Frontier Detection

The frontier detection strategy has been firstly introduced by Yamauchi in 1997 [72] and the name of the algorithm is based on the concept of *frontier*. The heart of the idea is that: "To gain the most new information, move to the boundary between open space and uncharted territory"[72]. This last sentence is exactly the concept of frontier, i.e., the segment that separates known regions from unknown regions. Formally, a frontier is a set of unknown points that each have at least one open-space neighbor [72].

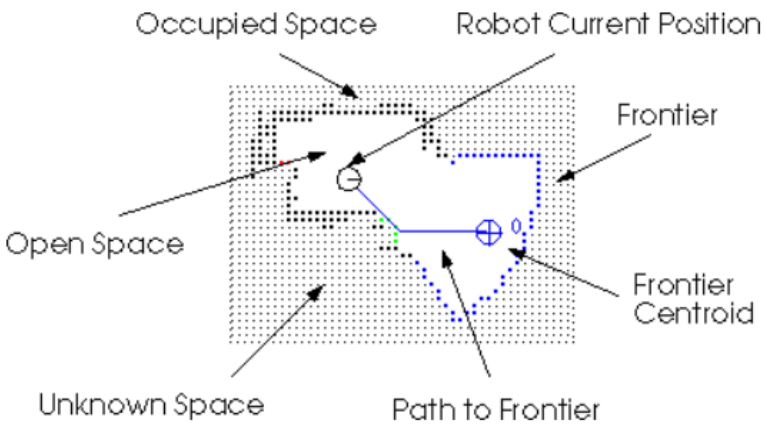
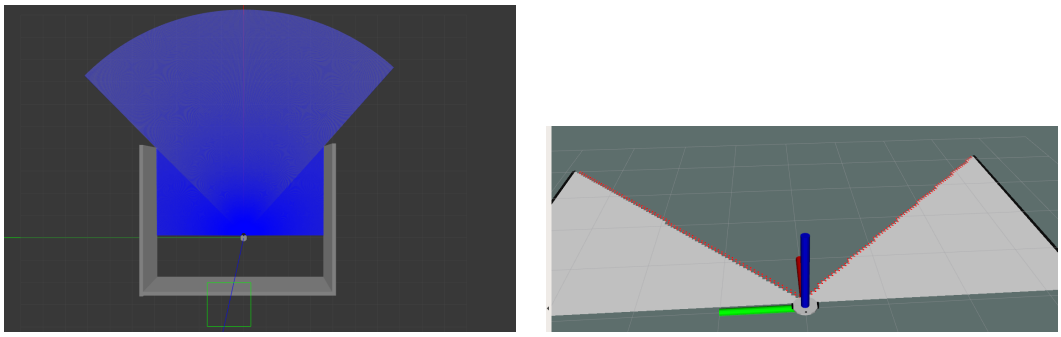


Figure 4.28: Frontier Definition [72]

A graphical interpretation of the concept of frontier is shown in figure 4.29.



(a) *Robot's surrounding* (b) *Resultin Frontier*

Figure 4.29: Example of frontier in a real situation

In the first figure, 4.29(a), a robot is depicted within a three-wall structure in Gazebo and its lidar beams painted in blue while in the second figure, 4.29(b), the mapped representation of this situation is shown in RVIZ. From these two images it can be visualized how a frontier is defined and evaluated. When the lidar detects an object, then the space between the robot and the object is revealed and selected as known area, whereas the open areas without obstacles are unknown parts. In 4.29(b), the two frontiers are shown by a number of red points that are, as previously said, the lines separating a known part (the light one) from unknown part (the dark one). There exist many implementations of Frontier Detection Algorithms, the one chosen in this work is named Wavefront Frontier Detector (WFD) [62], which is based on two nested Breadth-First Searches [14]. A Breadth-First Search is an algorithm for traversing a tree or graph data structures. The algorithm allows to traverse a layered graph starting from a selected node, that represents the root, and exploring the neighbor nodes. As the name suggests, the algorithm traverses the graph breadthwise as follows [1]:

1. First move horizontally and visit all the nodes of the current layer

2. Move to the next layer

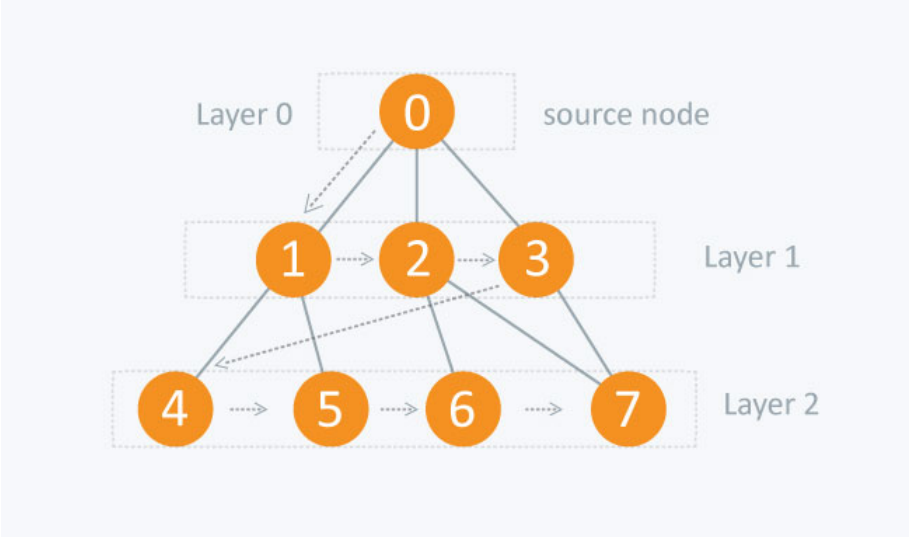


Figure 4.30: Layered Diagram [1]

The pseudo-code of the BFS is shown in Algorithm 7 [1].

Algorithm 7 BFS Pseudo-Code

```
1: function ALGORITHM BFS(G, s):    ▷ Where G is the graph and s the source node
2:     let Q be queue.
3:     mark s as visited.
4:     while Q is not empty do        ▷ Removing that vertex from queue, whose neighbor will be visited now
5:         Q.dequeue()
6:         for all neighbours w of v in Graph G do
7:             if w is not visited then
8:                 Q.enqueue(w);        ▷ Stores w in Q to further visit its neighbor
9:                 mark w as visited
```

Applying this pseudo code, it is possible to traverse a layered diagram, as it is depicted in figure 4.31.

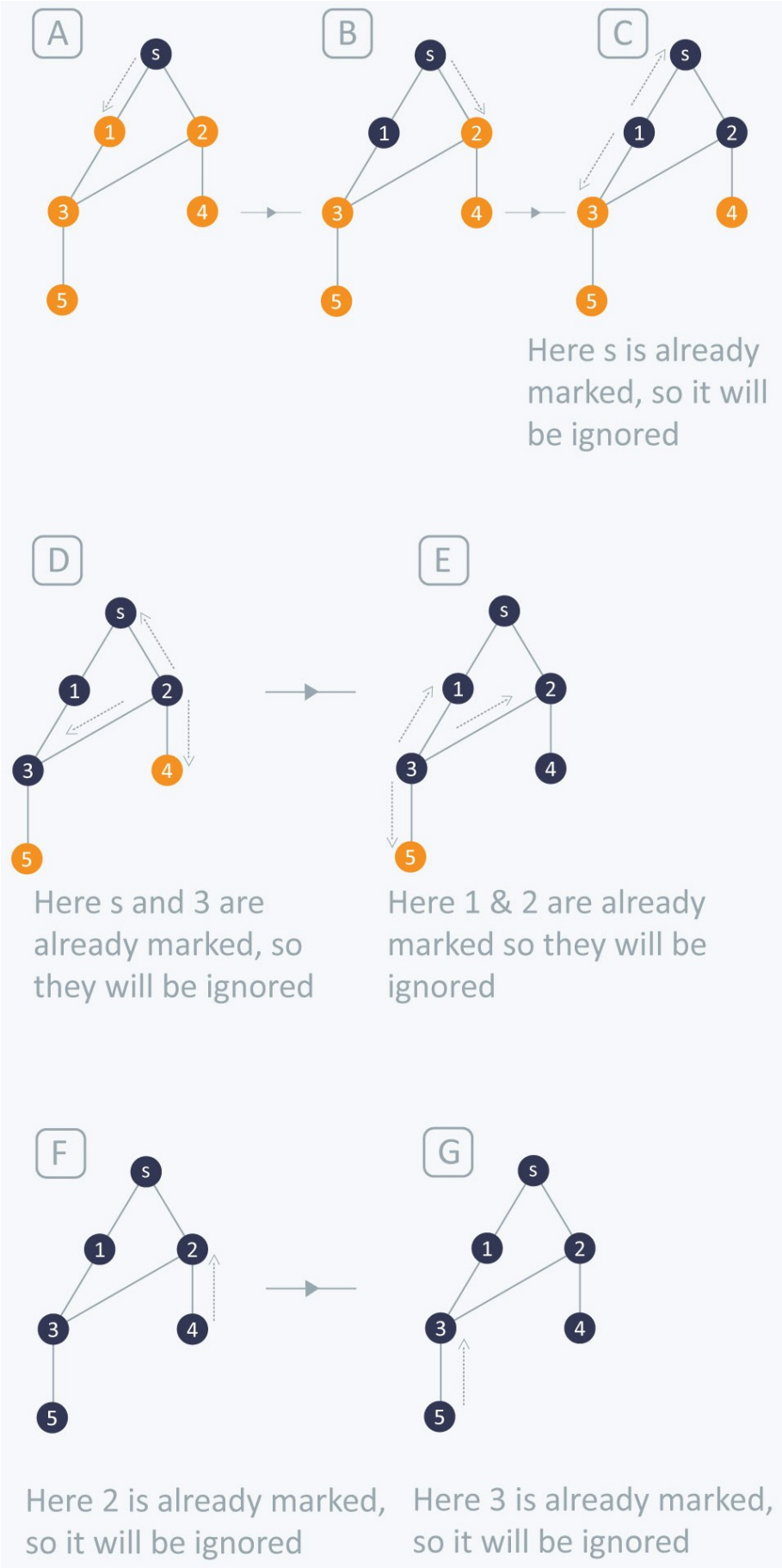


Figure 4.31: Algorithm application [1]

In the case of the Wavefront Frontier Detector, a first BFS is applied on the entire grid to find frontier points. Once a frontier point is found, another BFS is run to extract its frontier. As the authors states in [36], in this way we scan only the known regions of the occupancy grid as opposed to the original approach which scans the entire grid at every run of the algorithm. This has been possible by the use of appropriate point classifications based on four indications:

1. *Map-Open-List*: points that have been enqueued by the outer BFS

2. *Map-Close-List*: points that have been dequeued by the outer BFS

3. *Frontier-Open-List*: points that have been enqueued by the inner BFS

4. *Frontier-Close-List*: points that have been dequeued by the inner BFS

The full pseudo-code of the WFD algorithm is depicted in Algorithm 8 [36]. This approach proposes that moving to the frontiers the robots will gain as much information as possible. With this algorithm we obtain the list of all points locations that belongs to the lines separating known areas from unknown areas. Since this step is needed to propose a number of possible goal points for the robots, it is needed to group all the points belonging to a same frontier and then elect some representative points that represent potential goals for the robots. To this aim, after all the frontier points have been computed, a clustering algorithm is needed to groups these points in group of frontiers and, in particular, a $K - means$ strategy has been chosen. Through the $K - means$ algorithm all the frontiers are grouped and the information about how many points belong to them is stored. Finally the median point for each frontier is found so that it can be offered as goal points. A representation of

this process is depicted in figure 4.32 [74].

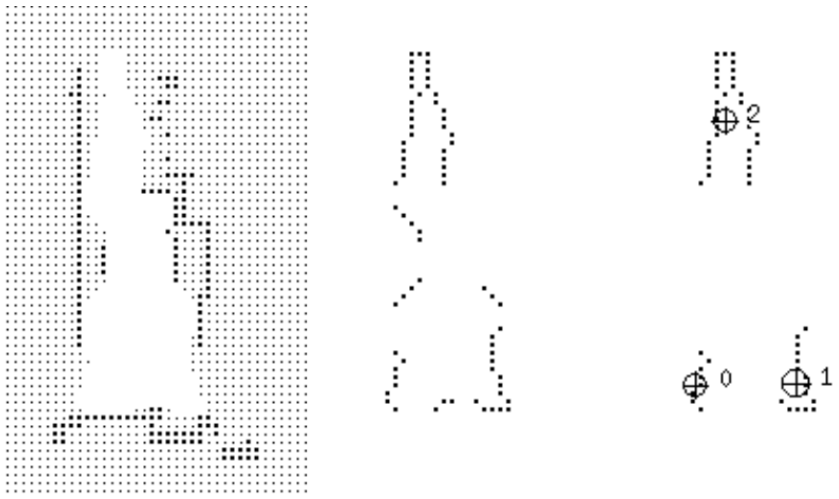


Figure 4.32: (a) Frontier detection in the evidence grid, (b) frontier edge clustering, (c) frontiers median points

Algorithm 8 WFD Pseudo-Code

Require: $queue_m$ || *queue, used for detectionfrontier points from a given map*
Require: $queue_f$ || *queue, used for extracting a frontier from a given frontier cell*
Require: *pose* || *current global position of the robot*

1: $queue_m \leftarrow \emptyset$
2: $ENQUEUE(queue_m, pose)$
3: *mark pose as "Map-Open-List"*

4: **while** $queue_m$ *is not empty* **do**
5: $p \leftarrow DEQUEUE(queue_m)$

6: **if** *p is marked as "Map-Close-List"* **then**
7: *continue*
8: **if** *p is a frontier point* **then**
9: $queue_f \leftarrow \emptyset$
10: *NewFrontier* $\leftarrow \emptyset$
11: $ENQUEUE(queue_f, p)$
12: *mark p as "Frontier-Open-List"*
13: **while** $queue_f$ *is not empty* **do**
14: $q \leftarrow DEQUEUE(queue_f)$
15: **if** *q is marked as* {*"Map-Close-List","Frontier-Close-List"*} **then**
16: *continue*
17: **if** *q is a frontier point* **then**
18: *add q toNewFrontier*
19: **for all** $w \epsilon\ adj(q)$ **do**
20: **if** *w not marked as "Frontier-Open-List", "Map-Close-List", "Frontier-Close-List"* **then**
21: $ENQUEUE(queue_f, w)$
22: *mark w as "Frontier-Open-List"*
23: *mark q as "Frontier-Close-List"*
24: *save data of NewFrontier*
25: *mark all points of NewFrontier as "Map-Close-List"*
26: **for all** $v \epsilon\ adj(p)$ **do**
27: **if** *v not marked as* {*"Map-Open-List","Map-Close-List"*}
28: **and** *v has at least one "Map-Open-Space" neighbor* **then**
29: $ENQUEUE(queue_m, v)$
30: *mark v as "Map-Open-List"*
31: *mark p as "Map-Close-List"*

Chapter 5

Multi robot Scenario

5.1 Multiple Robot System

For a Multi Robot System, many classifications can be done, for example *homogeneous* and *heterogeneous* systems can be distinguished; a system of multiple robots is said homogeneous if all the elements of the systems are of the same type and heterogeneous if it is composed of different types of robots. [24], [23] and independently [13] created a taxonomy or collection of axes to classify a swarm or collective or robot collaboration research. The classification is reported in [22]:

- Size of collective: the number of robots in the system. The greater the number of robot, the greater the complexity of the strategies that have to be designed to guarantee the effectiveness and the safety of the system.
- Communication range: the maximum distance between two robots that allows them to communicate.
- Communication topology: so the topology with which the communication hap-

pens within the communication range.

- Collective reconfigurability: the ability of the system to react if something goes wrong, for example one robot stops working of the communication falls down.

- Processing ability: the computational model utilized by individual elements of the collective.

- Collective composition: that describes if the system is heterogeneous of homogeneous.

Multi-robot systems may perform a given task faster and more robustly than a single robot system, but in order to take full benefit of using these systems, the work performed by each robot should be effectively allocated.
This allocation can be done in a different ways [22]:

- **Centralized** control structure has the coordination control of the elements of the collectives in one central entity. Each robot shares its informations, obtained from sensors, and its position to a central coordinator that, basing on certain reasonings, assigns a goal to each robot. This kind of control architecture is the simplest to be implemented and in many cases the most effective in terms of task assignment. Anyway the biggest problem of this kind of choice derives to the poor robustness since if the coordinator gets damaged the entire system crashes down.

- **Hierarchical** architecture, that assigns at each element of the collective a hierarchy with well-defined lines of communication and authority among the

members. In extreme cases the hierarchy is similar to the one used in military units. This kind of solution is more robust than the centralized one but it presents problems when some superior is damaged since it implies that all the lower robot in the hierarchy would not be able to work anymore too.

- **Decentralized** control structure, conversely, considers each robot in the multi robot system as an individual entity, able to make it own decision basing on the information it gets. This kind of control structure requires deeper design and computational needs might be higher on the robots' side that can be critical for platforms with limited resources, but it guarantees a significant redundancy and robustness that may be the most important requirements in many cases.

- **Hybrid** system, in which all the control structure described so far are combined opportunely to get the best system for specific cases.

Anyway in the fields of multi-robot exploration and mapping, the main control architectures are centralized and decentralized, which their graphical interpretation is provided by figure 5.1.

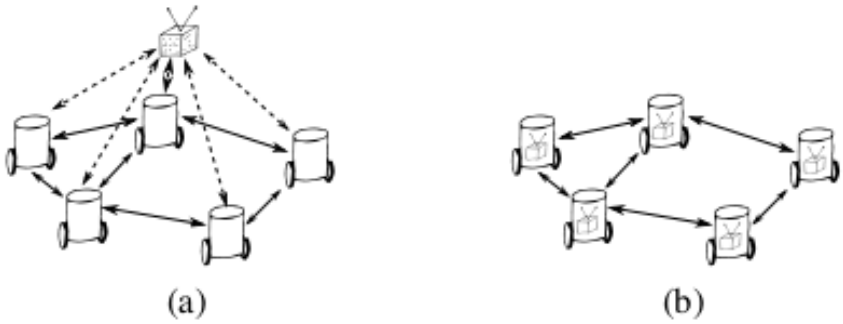


Figure 5.1: (a) Centralized system (b) Decentralized system [7]

A further decision that has to be taken when dealing with a multi robot system is about the level of coordination. In particular the multi-agent communication can be:

- **Full**, that means that each robot can communicate and exchange information with other robots
- **Limited**, that means that the communication is possible only in some cases
- **None**, so each agent don't directly communicate with other robots.

Full communication is usually very difficult to achieve. In real situations, the communication is possible when robots are within certain distances and when the communication terminal is free i.e. when a robot is not talking with some other agents yet.

5.2 Multi Robot Exploration Strategies: Background

Multi robot exploration strategies have been studied for almost three decades. Many researches in this field have been conducted due to its huge potentiality and many control policies for optimally explore unknown environments have been introduced. According to the different robot coordination approaches mentioned in the previous section, most algorithms are usually centralized or decentralized. Each coordination policies have advantages and disadvantages and the choice is really dependent on the application field. The simplest way to allocate goal to each robot can be in a random way. For sure this method will lead to fully explore a map but in this

way the exploration is inefficient and not deterministic. One of the most popular solutions in the area of centralized approaches is a solution based on auction among the communicating agents. The method assumes that each robot, disposing of a set of possible goals, shares this set among a central entity. Once all the goals are communicated, each robot bids on the goals according to a certain metric, that usually involves the distance between the robot and the goal. Finally robots share their bids and the central authority decides the winner for each goals basing on a first-price auction format. Anyway many variants of this idea have been done and also distributed versions of this algorithm have been introduced to overcome the centralization mechanism limitations. Many works on this approach have been conducted, such as [10], [59], [28]. Another strategy used in many researches is called Iterative Assignment (IA), based on the Broadcast of Local Eligibility (BLE) algorithm [65], in which after all the pairs $\langle robot, task \rangle$ are collected, they are ordered according to the highest utility, i.e. the distance cost, Ω such that $\Omega(p_1) \leq \Omega(p_2) \leq ... \leq \Omega(p_l)$; the ordered sequence is traversed starting from its first element and the first not already used goal is iteratively assigned to a robot without goal. Both centralized and decentralized versions of this method exist. In this work four state-of-art algorithms have been studied and tested. Moreover another novel strategy is introduced. The algorithms tested and discussed in the following sections are:

- Greedy Algorithm
- MinPos
- Hungarian Method
- Cost-Utility Function

5.2.1 Greedy Algorithm

One of the most used technique in the area of multi robot exploration is surely the Greedy algorithm, introduced by Brian Yamauchi, which in 1998 proposed a really simple but very effective frontier navigation strategy [73]. The idea of the Greedy algorithm is that, once frontiers are detected, each robot "attempts to navigate to the nearest accessible, unvisited frontier" [73]. Once the robot determines the nearest frontier, exploiting a path finding that returns the shortest obstacle-free path, the robot moves towards its destination for a certain amount of time, after that the robot declares that point inaccessible and that location is inserted in a list of inaccessible frontiers. The algorithm proposed by Yamauchi considers each robot as a central entity, which makes its own choice basing on the nearest frontier. While moving each robot builds a Global map. When a robot arrives at a frontier, it builds a local grids through its sensors and this grid is integrated with its global map and it's shared with the global grid of each robot (figure 5.2). This approach has the advantage of being both cooperative and decentralized. All of the information obtained by any robot is available to each robot. This allows robots to use the data from other robots to determine where to navigate. Yamauchi's method introduces a certain coordination among robots since each robot shares the gathered information to all the other robots. Anyway, the coordination is an "implicit coordination" that in many cases could not be enough. In fact, the main inefficiency of this approach is that since there is no shared information about the point that each robot is going to visit, so it may happen that two or more robots navigate to the same frontier. The pseudo-code of the Greedy Algorithm is shown in Algorithm 9.

Algorithm 9 Nearest frontier

Input: R_i, C_i $Cost\ Vector\ of\ R_i$

Output: α_{ij} $assignment\ of\ robot\ R_i\ to\ frontier\ F_j$

begin

$\alpha_{ij} = 1\ with\ j = \underset{\forall F_j \epsilon F}{argmin} C_{ij}$

end

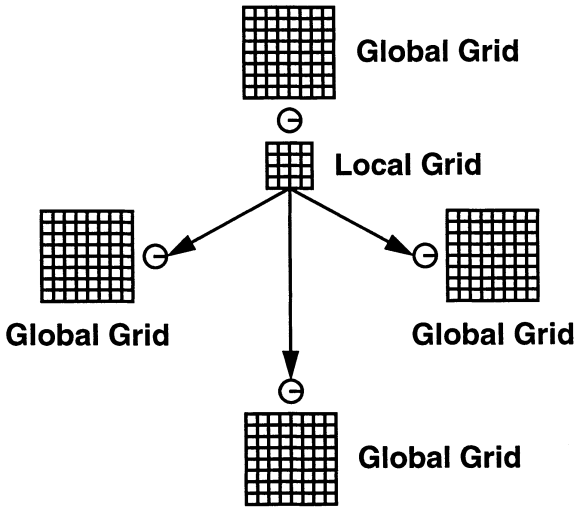


Figure 5.2: Information Sharing [73]

5.2.2 MinPos

Minimum Position (MinPos) algorithm is based on the distribution of robots among the frontiers [9]. This algorithm does not just try to assign the best goal for one robot with respect to its position, as Greedy algorithm does, but it tries to choose the most convenient goal point with respect to the other robots. In order to do this, the MinPos algorithm considers the notion of position or rank of a robot towards a frontier, by counting how many robots are closer to the frontier. The original paper [9] proposes an exploration strategy articulated as follow:

- Frontiers identification and clustering
- Computation of the distances to frontiers
- Assignment to a frontier
- Navigation towards the assigned frontiers for a fixed time period

In practice, this approach consists in assigning to a robot a frontier for which it is in best position, i.e. the frontier having the less robots closer than itself. Considering P_{ij} the position of the robot R_i towards a frontier F_j as:

$$\sum_{\forall R_k \epsilon R, k \neq i, C_{kj} < C_{ij}} 1 \tag{5.1}$$

which computes the cardinal of the set of robots closer to the considered frontier than the robot being assigned. This approach is based on the concept of positions and not of distances as Greedy algorithm does. With this approach, one robot will be assigned to one frontier only if it is the nearest one with respect to the other for that frontier, whatever the distances, thus favoring a better spatial distribution of robots on frontiers. Let's take as an example the situation depicted in figure 5.3.

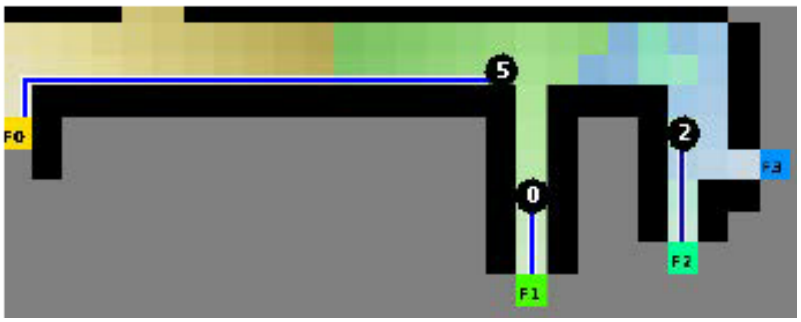


Figure 5.3: MinPos example [9]

As it can be seen in the figure 5.3, R_5 is the nearest robot to the frontier F_0 and is in second position for frontiers F_1, F_2 and F_3. The situation described in the figure 5.3 can be translated in the table 5.1:

Table 5.1: Table of positions

	F_0	F_1	F_2	F_3
R_0	$2nd$	$1st$	$3rd$	$3rd$
R_2	$3rd$	$3rd$	$1st$	$1st$
R_5	$1st$	$2nd$	$2nd$	$2nd$

Each robot will be assigned to the frontier in which they are in first position and in case a robot is first for more than one frontier, as for the *robot*2 in the example, the robot will be assigned to the nearest frontier (in the example to the frontier F_2). In case of the Greedy algorithm, in which each robot would consider only the nearest frontier to itself, both R_0 and R_5 would select as goal point the frontier F_1. In this terms, the MinPos method gives a better spatial distribution. Finally:

$$R_0 \rightarrow F_1$$

$$R_2 \rightarrow F_2$$

$$R_5 \rightarrow F_0$$

The MinPos pseudo code is given in Algorithm 10.

Algorithm 10 MinPos

Input: R_i, C *cost matrix*

Output: α_{ij} *assignment of robot* R_i

for all $F_j \epsilon F$ **do**

$$P_{i,j} = \sum_{\forall R_k \epsilon R,\, k \neq i,\, C_{k,j} < C_{i,j}} 1$$

$\alpha_{ij} = 1$ *with* $j = \underset{\forall F_j \epsilon F}{argmin} P_{i,j}$

In case of equality choose the minimum cost among min $P_{i,j}$

5.2.3 Hungarian Method

This multi-robot exploration strategy is based on the Hungarian algorithm, a *combinatorial optimization algorithm* that solves the assignment problem for m machines and m tasks. It was developed by Hatold Kuhn in 1955, an American mathematician, who dedicated the name "Hungarian method" to the two Hungarian scientists that started before him the basis of the algorithm. The original algorithm had a time complexity of $O(n^4)$ and Prof. James Munkres observed that the algorithm is strongly polynomial [43]. Nevertheless, the algorithm has undergone variations that brought it to be $O(n^3)$ [69].

The first step of the algorithm consists in creating a $n \times n$ table that describes the cost of a machine with respect to a certain work. In the case of goal assignment of multi robot system, the costs are the distance between robots and goals:

Table 5.2: Table of costs

	$Goal1$	$Goal2$	$Goal3$	$Goal4$	...	$Goaln$
R_0	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	$d_{1,4}$	...	$d_{1,n}$
R_2	$d_{2,1}$	$d_{2,2}$	$d_{2,3}$	$d_{2,4}$	...	$d_{2,n}$
R_5	$d_{3,1}$	$d_{3,2}$	$d_{2,3}$	$d_{3,4}$	...	$d_{3,n}$
R_4	$d_{4,1}$	$d_{4,2}$	$d_{4,3}$	$d_{4,4}$	...	$d_{4,n}$
...	...	...	...	...	...	...
R_n	$d_{n,1}$	$d_{n,2}$	$d_{n,3}$	$d_{n,4}$	...	$d_{n,n}$

The next step is to translate this table into a $n \times n$ cost matrix:

$$C = \begin{pmatrix} d_{1,1} & d_{1,2} & d_{1,3} & ... & d_{1,n} \\ d_{2,1} & d_{2,2} & d_{2,3} & ... & d_{2,n} \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & d_{n,n} \end{pmatrix}$$

The aim of the algorithm is to find the assignments goal−robot, such that the total cost of assignments is minimum. This can be expressed by permuting the rows and columns of a cost matrix C to minimize the trace of a matrix:
$\min_{L,R} Tr(LCR)$ where L and R are permutation matrices [69]. The method can be more easily be expressed in terms of bipartite graph:

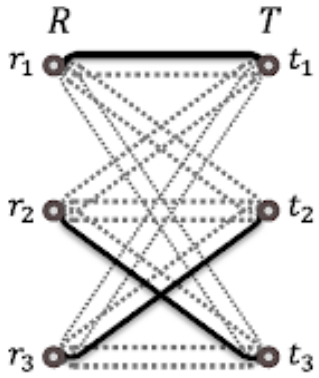


Figure 5.4: Hungarian Method: graph representation

in which vertices are represented by robots (R) and goals/tasks (T) and lines connecting them represents the weights, so the costs. Given the bipartite graph, the Hungarian method is able to find the maximum-weight matching. The Hungarian algorithm is divided into four steps. Let's apply the algorithm for the minimization problem shown in table 5.3.

Table 5.3: Minimization problem example

	*Goal*1	*Goal*2	*Goal*3	*Goal*4
R_0	20	25	22	28
R_2	15	18	23	17
R_5	19	17	21	24
R_4	25	23	24	24

1. Identify the minimum element in each row of the cost matrix C and subtract it from every element of that row.

Table 5.4: Minimization problem example: Step 1

	*Goal*1	*Goal*2	*Goal*3	*Goal*4
R_0	0	5	2	8
R_2	0	3	8	2
R_5	2	0	4	7
R_4	2	0	1	1

2. Identify the minimum element in each column of the cost matrix C and subtract it from every element of that column.

Table 5.5: Minimization problem example: Step 2

	*Goal*1	*Goal*2	*Goal*3	*Goal*4
R_0	0	5	1	7
R_2	0	3	7	1
R_5	2	0	3	6
R_4	2	0	0	0

3. Draw lines through appropriate rows and columns so that all the zero values of the cost matrix are covered and the minimum number of such lines is used.

Table 5.6: Minimization problem example: Step 3

	*Goal*1	*Goal*2	*Goal*3	*Goal*4
R_0	0	5	1	7
R_2	0	3	7	1
R_5	2	0	3	6
R_4	2	0	0	0

4. If the minimum number of lines required to cover all the zeros is exactly n, then choose the combination of robot-goal with zero to obtain the optimal solution. If the number of covering lines is less than n, then an optimal solution cannot be defined yet and step 5 has to be executed

5. Considering the minimum vertical and horizontal lines necessary to cover all zeros in the reduced matrix obtained in step 3, then considering the elements

not covered by these lines, choose the smallest one and subtract it from all the uncovered elements and add it to the elements lying on the intersection of two lines. Return to step 3.

Table 5.7: Minimization problem example: Check Step

	*Goal*1	*Goal*2	*Goal*3	*Goal*4
R_0	(0)	4	0	6
R_2	0	2	6	(0)
R_5	3	(0)	3	6
R_4	3	0	(0)	0

The final optimal solution for the example will be:

$$\text{Robot } 1 \longrightarrow \text{Goal } 1$$

$$\text{Robot } 2 \longrightarrow \text{Goal } 4$$

$$\text{Robot } 3 \longrightarrow \text{Goal } 2$$

$$\text{Robot } 4 \longrightarrow \text{Goal } 3$$

with a total cost of assignment equal to: $20 + 17 + 17 + 24 = 78$.

The Hungarian algorithm is based on the following theorem:

if a number is added to or subtracted from all of the entries of any one row or column of a cost matrix, then on optimal assignment for the resulting cost matrix is also an optimal assignment for the original cost matrix. Since the Hungarian Algorithm is based on a square matrix, some problems can arise in a real situations

in which a number m of goals are present in a certain instant and we are using n robots such that $n < m$. To overcome this problem it is possible to add a number $m - n$ of "dummy robots" with high costs that make the cost matrix C square and do not have any importance in the computation of the optimal goal points for the real robots. Since in the simulations four robots ($m = 4$) have been used, each time the frontier exploration algorithm gives a number n of frontier representatives, the algorithm finds the number of "dummy robots" to add as $p = n - m$ and assigns as costs to them 1000:

Table 5.8: Hungarian Algorithm: real application

	$Goal1$	$Goal2$	$Goal3$	$Goal4$	$...Goaln$
R_0	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	$d_{1,4}$	$d_{1,n}$
R_2	$d_{2,1}$	$d_{2,2}$	$d_{2,3}$	$d_{2,4}$	$d_{2,n}$
R_5	$d_{3,1}$	$d_{3,2}$	$d_{2,3}$	$d_{3,4}$	$d_{3,n}$
R_4	$d_{4,1}$	$d_{4,2}$	$d_{4,3}$	$d_{4,4}$	$d_{4,n}$
$Dummy_1$	1000	1000	1000	1000	1000
...	1000	1000	1000	1000	1000
$Dummy_n$	1000	1000	1000	1000	1000

In order to improve the algorithm efficiency, the distance between the robots and the goals is not just the Euclidean distance, but it is the distance statically evaluated by the path planner. This because the Euclidean distance does not take into account the obstacles, thus a point that is really near in terms of Euclidean distance, would mean a huge travel distance in case several obstacle divide the robot and that point.

5.2.4 Cost-Utility Function

Another navigation strategy tested is a reinterpretation of [30] for the case of frontier-based exploration strategy. Since there is not a real name for this implementation, the name Cost-Utility Function will be used to refer to this approach since it makes use of a cost utility function. The central point of [30] addresses is "where should the robot go next?". According to the authors, this problem is similar to the Next-Best View (NBV) problem studied in Computer Vision and Graphics. The paper also describes an NBV algorithm that uses the safe-region concept to select the next robot position at each step. The safe region is defined as the largest region guaranteed to be safe given the history of sensor readings. The new position is chosen within the safe region in order to maximize the expected gain of information under the constraint that the local model at this new position must have a minimal overlap with the current global map. Anyway, using a frontier-based algorithm the notion of safe region is implicitly considered as the proposed goal lays on the frontier which are the separation between the known and the unknown part of a map. Having the representatives of each frontier, let's identify them as $q \, \epsilon \, F_r$ where F_r is the set of all the frontier representatives present in a certain instant in the map, then the score of each candidate is defined by the following function:

$$g[q] = A(q) \cdot e^{-\lambda L(q)} \tag{5.2}$$

where λ is a positive constant, $L(q)$ is the length of the collision-free path computed by the planner, and $A(q)$ is, according to a frontier-based exploration, the width of the frontier. The idea that the formula wants to implement is that for a robot the best goal to visit is the one that allows the robot to visit a big area of the map and we know that a huge frontiers will offer a bigger range to acquire new information of

the environment. Furthermore, as it has been exploited in the Greedy algorithm, we want to visit the near points so that the robot can acquire new information about the map in the least possible time.

The constant λ is used for trading off between the two parameters $A(q)$ and $L(q)$. In particular, a small λ means that the motion is "cheap" and then also longer path can be chosen, contrary, with a high λ , the travel cost increases and closer points are preferred. So practically, when a robot computes the frontiers of its map, it evaluates for each frontier the equation aforementioned; the frontier that maximizes that function is the best frontier to visit and it's assigned as the next goal of the robot.

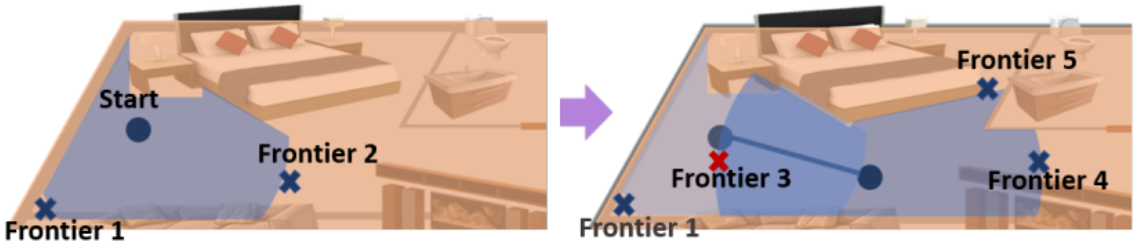


Figure 5.5: Cost Function Example [29]

For example, let's take into account the scenario in the figure 5.5. As it can be seen when a robot is in point start, tho frontier points are proposed as goal candidates by the frontier detection algorithm, which are $Frontier1$ and $Frontier2$. Assuming that the distance between the start and these two points is equal, what is important to maximize the cost-utility function $g[q]$ is the width of the frontier. This example let to visually understand why it is important to choose the bigger one. This is because a higher frontier will open the robot to visit a much higher area of the map and so this let the robot to improve the fastness of the exploration.

Assuming for example that $Frontier1$ is closer to the start point with respect to $Frontier2$, even if with a small quantity of $0.1m$, with a strategy like the Greedy algorithm that always chooses the nearest one, the robot would go to the $Frontier1$, loosing time in visiting a useless frontier. A similar and interesting variation of this algorithm is provided in [29] that uses a similar cost function with respect to the one described so far, but also mentions another parameter that takes into account the semantic utility of a frontier, preferring doors and transit area instead of corners and less useful frontier, with a cost-utility function equal to: $g[q] = A(q) \cdot S(q) \cdot e^{\{1/C(q)\}}$.

5.3 Proposed method

Each of the four algorithms proposed so far tries to answer to a basic question: "How can I exploit at best the use of multiple robots to make the exploration faster?". Each algorithm answers to this question in a different way, someone trying to make the algorithm as simpler as possible, avoiding the communication between robots to make them autonomous and accepting some defects in terms of efficiency, someone else aims at finding the most efficient task allocation having as drawbacks a huge computational burden and assuming the explicit coordination among robots. Anyway, also taking into account the experimental results it is evident that some algorithms have better outcome in terms of efficiency. Analyzing the results and make use of the results proposed in many papers, this work proposes a new strategy for the management of multiple robots for mapping an unknown environment.

How many times we heard in a detective film the sentence "Let's split up and sweep the area" when a task of policemen has to explore an unknown environment to find hostages in the shortest possible time? This is the main idea on which it insists

the proposed strategy, that is to make the robot to be far from each other in order to have a better spatial distribution of a map. Anyway as the other algorithms discussed sentence, it is also important for a robot to choose near points to visit in order to acquire new information as soon as possible. In order to balance between these two aspects a cost-utility functions is proposed in order to find a frontier point that is as near as possible to the robot that is searching its goal and as far as possible from the other robots in order to distribute the overall group of robot in the best way in the discovered part of the map. In order to maximize the distance from the other robot it is needed to find a metric able to maximize such a distance. Since in this thesis the work has been conducted with four robots, when a robot wants to find the best frontier point to visit, according to the idea mentioned before, it should evaluate the distance from each frontier point to the other three robot and then maximize the distance with these robots at the same time. Thus, the question to answer is "which is the point equidistant from the three robot locations?"; for this, a geometric definition comes in handy, which is the *circumcenter*. According to [66], the circumcenter of a polygon is the center of the circle, called *circumcircle*, that passes through all the vertices of the polygon. In the case of this work, having four robot, when a robot asks for a goal, the other three robot locations are considered and the circumcenter is the point where the perpendicular bisectors of the triangle formed by these locations intersect.

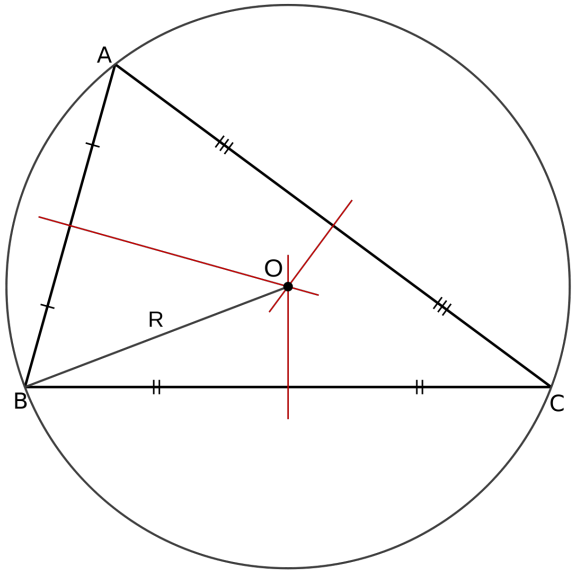


Figure 5.6: Circumcenter: A, B, C correspond to robots positions

The main property that makes this geometric definition perfect for the problem analyzed is that this point is equidistant from vertices of the triangle formed by the robots locations. In order to evaluate this point it is possible to exploit another mathematical theorem that states that through three non-collinear points passes one and only one circle. Then finding the equation of this circle and extracting it's center we have exactly the circumcenter.

The canonical equation of a circle is for definition:

$$x^2 + y^2 + ax + by + c = 0 \tag{5.3}$$

In order to find the circle passing through the three robot locations $[x_1, y_1]$, $[x_2, y_2]$

and $[x_3, y_3]$, the following system of equations has to be solved:

$$\begin{cases} x_1^2 + y_1^2 + ax_1 + by_1 + c = 0 \\ x_2^2 + y_2^2 + ax_2 + by_2 + c = 0 \\ x_3^2 + y_3^2 + ax_3 + by_3 + c = 0 \end{cases} \tag{5.4}$$

Since quantities $(x_i^2 + y_i^2)$ are known terms, the system in 5.4 can be rewritten as:

$$\begin{cases} ax_1 + by_1 + c = -(x_1^2 + y_1^2) \\ ax_2 + by_2 + c = -(x_2^2 + y_2^2) \\ ax_3 + by_3 + c = -(x_3^2 + y_3^2) \end{cases} \tag{5.5}$$

In order to summarize these three equations in just one it is possible to write:

$$ax_i + by_i + c = -(x_i^2 + y_i^2) \;\; where \;\; i = 1, 2, 3 \tag{5.6}$$

In which a, b and c are the unknowns, x_i is the coefficient of the unknown variable a, y_i is the coefficient of the unknown variable b, 1 is the coefficient of the unknown variable c and $-(x_i^2 + y_i^2)$ are the known terms. A possible way to solve this system of three equations in three unknowns is the Cramer's rule. The Cramer's rule is a linear algebra formula that allows to determine the solution of a system of linear equations with n equations in n unknowns [67]. The formula expresses the solution in terms of the determinant of the square coefficient matrices obtained replacing one column by the column vector of right-hand-sides of the equations. This, in the treated case, translates in:

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} -(x_1^2 + y_1^2) \\ -(x_2^2 + y_2^2) \\ -(x_3^2 + y_3^2) \end{pmatrix} \tag{5.7}$$

The Cramer's rule states that values a, b and c can be determined as follows [67]:

$$a = \frac{\begin{pmatrix} -(x_1^2+y_1^2) & y_1 & 1 \\ -(x_2^2+y_2^2) & y_2 & 1 \\ -(x_3^2+y_3^2) & y_3 & 1 \end{pmatrix}}{\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}} \tag{5.8}$$

$$b = \frac{\begin{pmatrix} x_1 & -(x_1^2+y_1^2) & 1 \\ x_2 & -(x_2^2+y_2^2) & 1 \\ x_3 & -(x_3^2+y_3^2) & 1 \end{pmatrix}}{\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}} \tag{5.9}$$

$$c = \frac{\begin{pmatrix} x_1 & y_1 & -(x_1^2+y_1^2) \\ x_2 & y_2 & -(x_2^2+y_2^2) \\ x_3 & y_3 & -(x_3^2+y_3^2) \end{pmatrix}}{\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}} \tag{5.10}$$

Knowing the equations of the circle, the center position is easily computed as:

$$O = (-\frac{a}{2}; -\frac{b}{2}) \tag{5.11}$$

Once computed the center of the circumference, we know the circumcenter of the triangle formed by the three robot positions. The position of point O depends of

the type of triangle formed by the three points.

In particular [66]:

- if the triangle is acute, the point will be inside it
- if the triangle is rectangle, the circumcenter is the median point of the hypotenues
- if the triangle is obtuse, the point is outside the triangle

Before computing the circumcenter is essential to verify that the three points are not aligned, because, if it happens, a circle passing through the three points does not exist. To check this condition it is needed to check if the following equation holds:

$$\frac{y_3 - y_1}{y_2 - y_1} = \frac{x_3 - x_1}{x_2 - x_1} \tag{5.12}$$

If this happens, the robot will choose the nearest frontier, behaving as the Greedy algorithm. If the three points are not aligned, and then it's possible to evaluate the circumcenter with coordinates $O = (-\frac{a}{2}; -\frac{b}{2})$, then the distance between each frontier representatives and this point is evaluated, let's call it $d_{cc}(q) \mid q \, \epsilon F_r$ where F_r is the set of frontier representatives.

The next goal that the robot will visit is the one the maximizes the following cost-utility function:

$$g[q] = \frac{d_{cc}(q)}{L(q)} \tag{5.13}$$

where $L(q)$ is the distance between the robot and the frontier q and $d_{cc}(q)$ is the distance between frontier q and the circumcenter found. The idea that the function tries to implement is to choose a frontier that is as near as possible to the robot

that is asking for a goal and as far as possible from the other robot. Let's imagine to have the four robots that form a square as shown in figure 5.7.

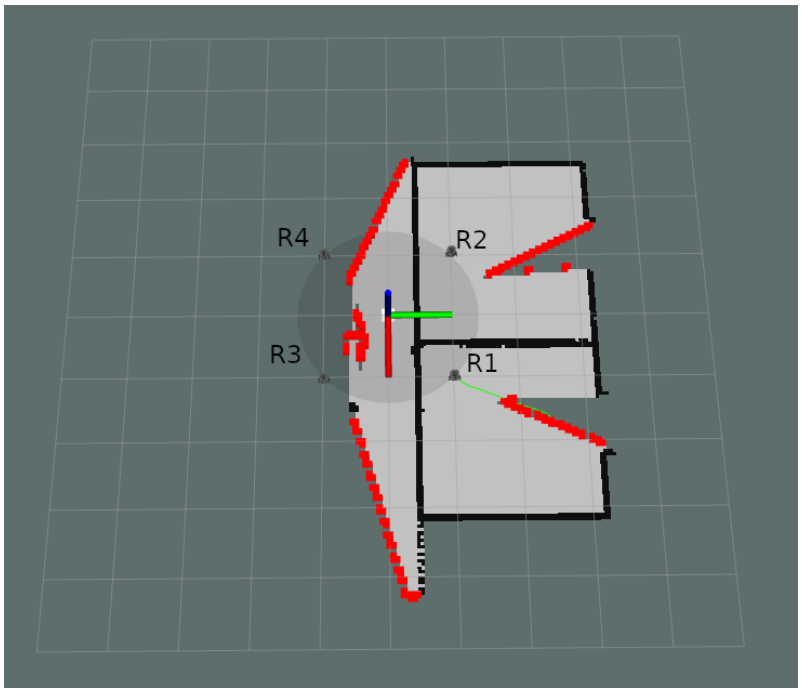


Figure 5.7: Circumcenter Rule Example1

In this example we have a robot $R1$ searching for a new goal. First of all a central unit takes the position of the other three robots evaluated from the Extended Kalman Filters. Then it computes the circle passing through their position. Since in this case the robots are positioned on the vertices of a square, clearly the triangle formed by any combination of three robots is a rectangular triangle. For the reasons explained before, the circumcenter is positioned on the middle of the hypotenuse of the triangle formed by the three robots locations. In the picture it is depicted as a white box in the center of the three axes. Then the robot evaluates the equation $g[q] = \frac{d_{cc}(q)}{L(q)}$ for each frontier median point that the frontier detection algorithm extracts and it chooses the one that maximize such function. In this case it the

point is the circular white one and the path the robot $R1$ will travel is the green one, as shown in figure 5.8.

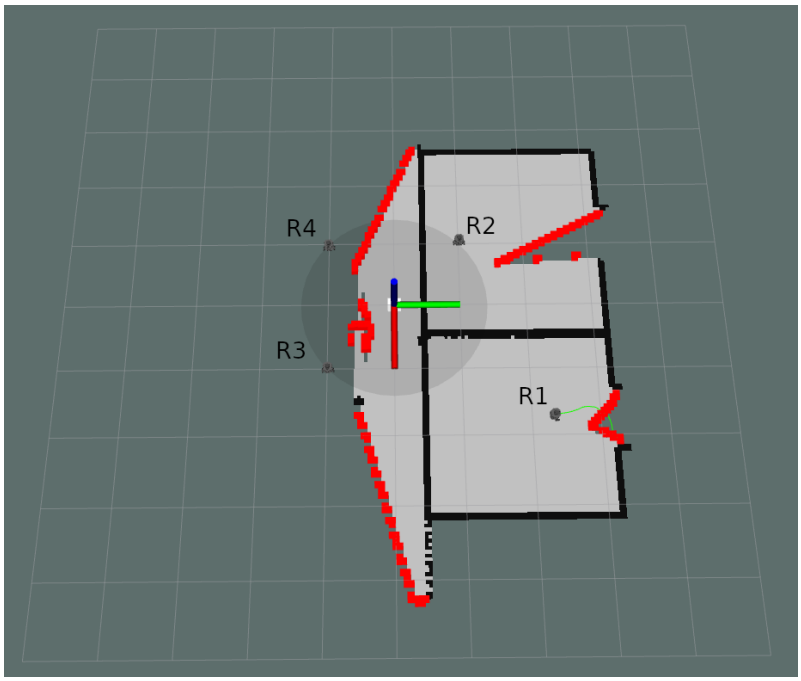


Figure 5.8: Circumcenter Rule Example2

5.4 Map Merge

Since in this work the aim is to build a map of an environment exploiting multiple robots, it is important that each robot shares its map with the other robot and viceversa, by merging all the maps in a unique one. The map merging problem , "is an interesting and difficult problem, which has not enjoyed the same attention that localization and map building have" as Konolidge et al. wrote in [38]. Techniques of map merging are usually divided into *direct map merging*, in which the algorithm relies on sensors to directly compute transformation between reference frame of

robots, and *indirect map merging*, that uses overlapping areas in maps to estimate transformation between maps. In this work an indirect approach has been used, in which a central entity is responsible for collecting local maps created by each robot and merge them into a global map which is used by the robots to navigate, explore and coordinate.

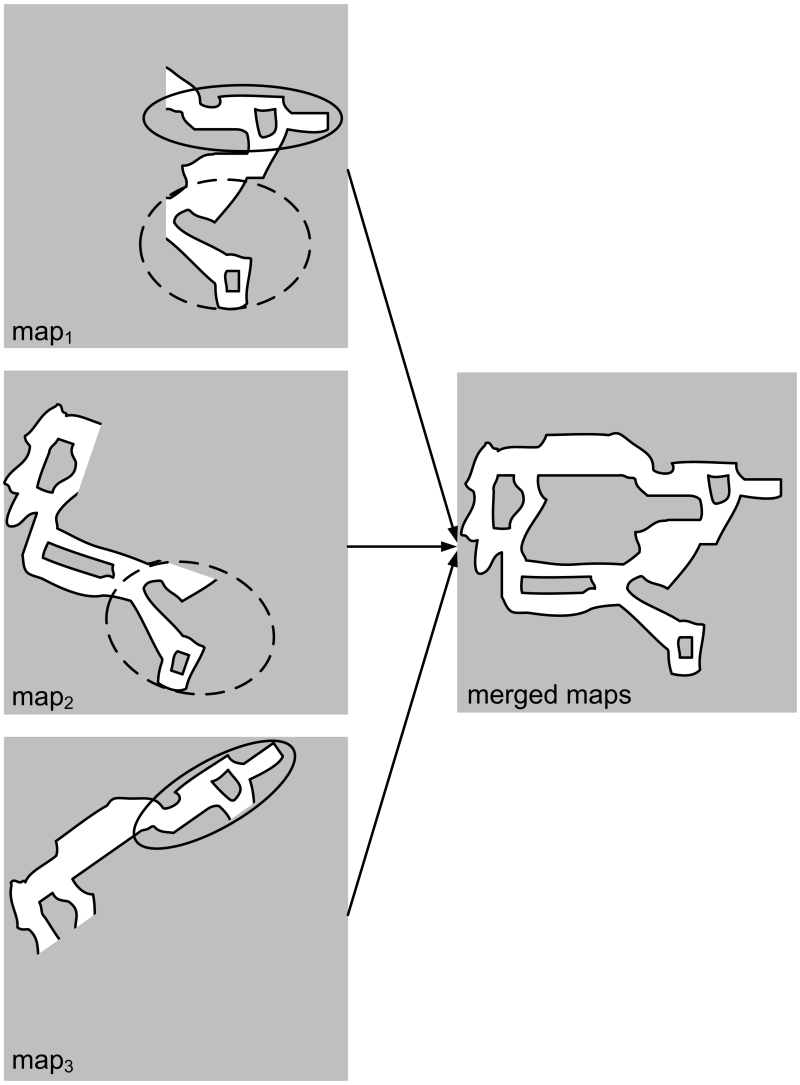


Figure 5.9: Map Merge [57]

In particular, a specific ROS package called *multirobot_map_merger* tries to

match patterns within the local maps from each robot to determine the transformation. The ROS *multirobot_map_merge* node was created by [34], anyway it is not the unique one implemented in ROS. For instance, it exists a package called *map_stitch* which has a strong limitation, since it can only produce a merged map given static maps. The *multirobot_map_merge* node, instead, is able to execute and update online during exploration. The map merger computes the transformation between two local maps $M1$ and $M2$ constructing the global map by joining the local maps maximizing the match over overlapping areas. The mathematical model of the map merging problem is described in [11]. As said before, the map merging problem has as objective to find the transformation between two maps that returns the best merging. Let m_1 and m_2 be two maps expressed in terms of matrices with dimensions $N \times M$. The overlapping between these two maps is defined as:

$$\omega(m_1, m_2) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} Eq(m_1[i,j], m_2[i,j]) \tag{5.14}$$

where:

$$Eq(m_1[i,j], m_2[i,j]) = \begin{cases} 1 & if\ m_1[i,j] = m_2[i,j] \\ 0 & otherwise \end{cases}$$

ω, called *overlapping function*, is a measure of how much two maps can be overlapped. In an ideal world, in which the map created by any single robots is equal to the real representation of the map, it exists a transformation which yields a perfect overlapping function. In the real applications this is not possible, so what it is possible to search is the transformation that maximizes the overlapping values. The problem can be then defined as finding the transformation $T(x, y, \theta)$ such that $\omega(m_1, T_{x,y,\theta}(m_2))$ is maximized.

Then, the problem is an optimization problem in a three dimensional space, which

goal function is the overlapping function ω. To perform this optimization problem a heuristic function δ is used to guide the search process. This function defines a metric that indicates the overlapping regions. Formulas introduced so far only deal with two merging maps. Anyway when more then two robots are used, a more structured model definition is required.

In particular, let $m_1, m_2, ..., m_n$ be the matrix representations of the maps produced by robots $r_1, r_2, ..., r_n$ respectively, with dimensions $N \times M$.

The overlapping between $m_1, m_2, ..., m_n$ is defined as:

$$\omega(m_1, m_2, ..., m_n) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} Eq(m_1[i,j], ..., m_n[i,j]) \tag{5.15}$$

where:

$$Eq(m_1[i,j], ..., m_n[i,j]) = \begin{cases} 1 & if\ m_1[i,j] = m_2[i,j] = ... = m_n[i,j] \\ 0 & otherwise \end{cases}$$

Also in this case, ω is called *overlapping function* and the map merging problem is solved finding $n-1$ transformations $T^1(x_1, y_1, \theta_1)$ up to $T^{n-1}(x_{n-1}, y_{n-1}, \theta_{n-1})$ such that $\omega(m_1, T^1_{x_1,y_1,\theta_1}(m_2), ..., T^{n-1}_{x_{n-1},y_{n-1},\theta_{n-1}}(m_n))$ is maximized.

The map merge node works in two modalities:

1. Merging with known initial positions: in which the user declares the initial position of its robots and make the merging problem easier

2. Merging without known initial positions: in which the algorithm itself has to find the merged map only observing the overlapping area of the various map. If the number of overlapping areas is not enough, the node won't find the merged map

The architecture of the ROS multirobot_map_merge is shown in figure 5.11 [55].

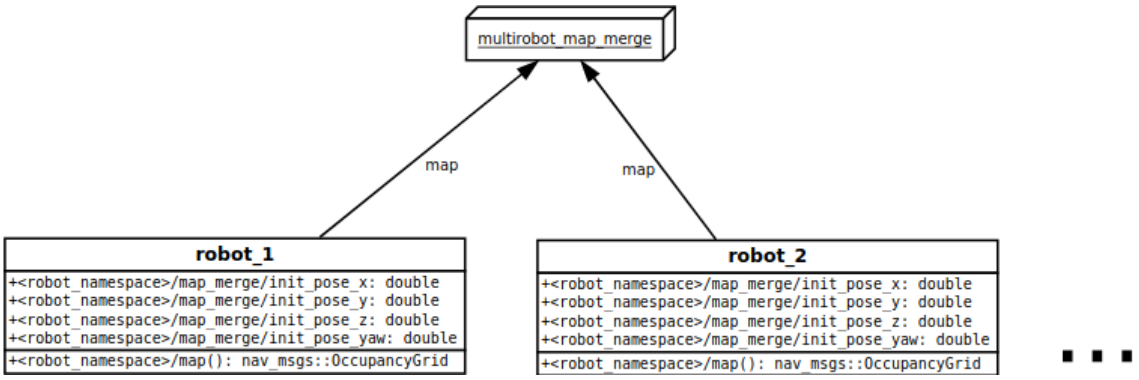


Figure 5.10: ROS Multirobot Map Merge Architecture

An example of the map_merge node output in a four robot application is depicted in figure 5.11.

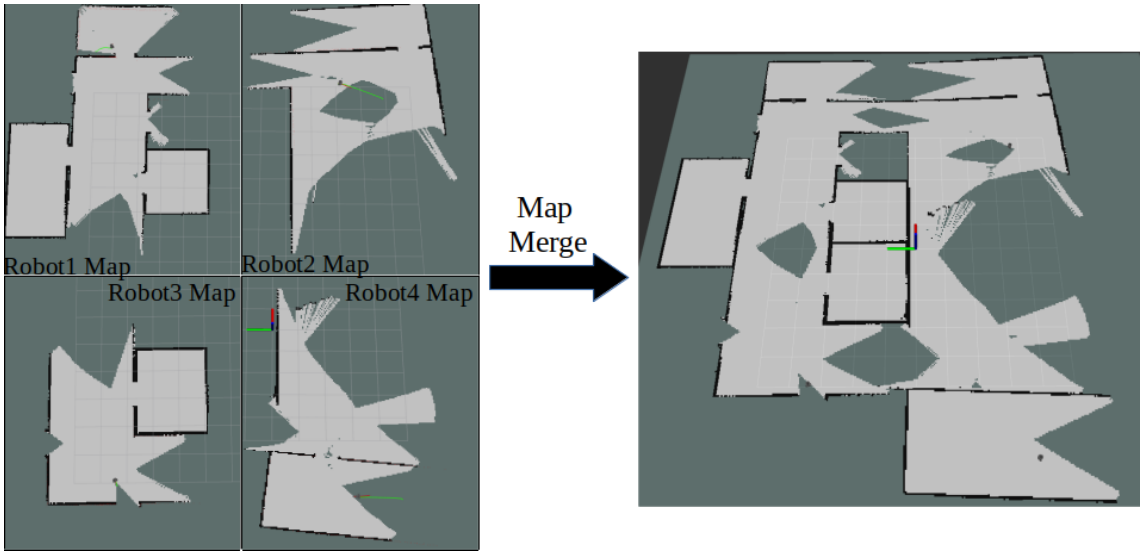
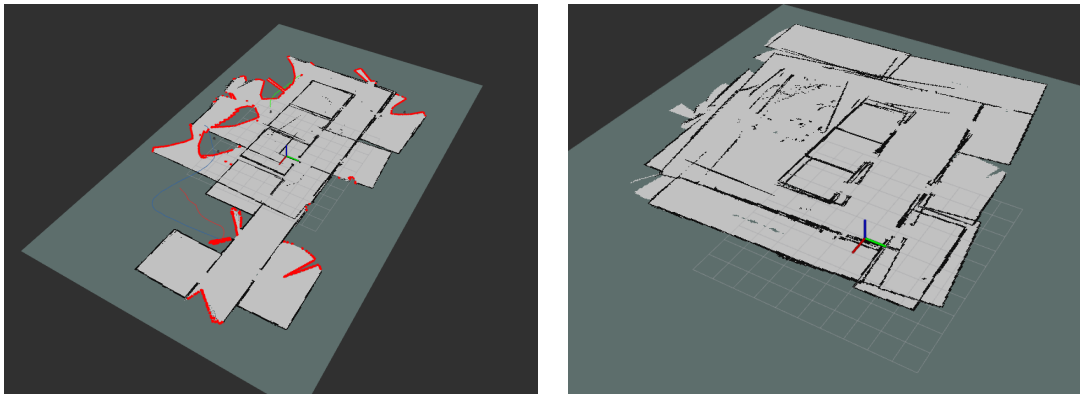


Figure 5.11: Map Merge example with 4 robots

The map merge node is a really useful node, anyway it presents some inefficiencies. Sometimes the map produced is not coherent with the actual map. This usually

happen when robot maps are not so equal each other making difficult the job of the merger to find enough fitting point to create a good transformation. In other cases, the merger computes wrongly the position of the map and shifts it with respect to the center. When these situations happen it is a problem for the simulation, then it is needed to chest such experiment and perform it again.



(a) *Wrong map* (b) *Real map, wrongly centered*

Figure 5.12: Problems of Map Merge node

As shown in figure 5.12(a), due to the wrong map created, the frontier detection computes frontiers that actually don't exist, but are a result of the fact that the maps that should be merged are not completely fused and the holes that born due to this inefficiency are considered as frontiers by the frontier detection node. Instead, in figure 5.12(b), the problem that arises is that robots get confused about the point to go since the merged map give them a certain position as goal, but it may happen that in their personal map that point lays on a wall thus creating conflict and impossibility to continue the experiment.

5.5 Experiment Outlines

5.5.1 Experimental Principles

In order to carry experiments in an efficient and proper way, the following principles, already treated in [26] and based on [4], are adopted:

- *Comparison* – is one of the main principles to measure and decide whether an algorithm is better than another basing on results. Anyway it is important to state in which conditions an algorithm provides better algorithms than another. To this aim three approaches are described in [6]:

  1. usage of the same implementations used in the previous experiments;
  2. development of a new implementation based on the available description;
  3. usage of the results reported by other authors in their publications.

- *Reproducibility* – concerns the level of detail of experiments to allow to other researches to replicate it

- *Repeatability* – that is related with the capacity of getting the same results among various trials, thus it is related to the controllability of the experiment

- *Justification and Explanation* – which deal with the general conclusions about some experiment and the motivation of results given the collected data. To efficiently execute this phase, it is useful to compare the obtained results with the one expected by the theory or reference result.

5.5.2 Views on Experimentation

Following the strict process used in [26], five views on experimentation are described [60]:

- *Feasibility Experiment*, which aims at demonstrating that a certain approach works, so it's feasible
- *Trial Experiment*, provides complex results in view of several experiment attempts providing performance indicators
- *Field Experiment*, which is the employment of the experiment into the real world in order to show validity and robustness of a certain solution
- *Comparison Experiment*, that is the demonstration of improvement brought by a solution rather than another by comparing different solutions and showing what the improvements consist of
- *Controlled Experiment*, i.e., testing of a solution controlling all the variable that could influence the response of the experiment, under specific protocol rules, so that the obtained result can be extended and generalized.

The previously mentioned techniques will then be employed to test algorithms cited in order to find out which algorithm is the most effective for multi-robot exploration. Anyway what mentioned so far does not take into account in which environment tests should be carried out. At this aim it is quite important to answer to this question proposing also for this problem opportune methods and techniques to efficiently create suitable environments for tests. According to [26] a suitable way to test multi-robotic exploration is to follow an approach based on three-pillars. The first

pillar consists in the description of the exploration framework, which gives a rough idea on what the exploration strategy is and what parameters could influence the outcome of that strategy. The second pillar is the benchmark that consists of environments, particular frameworks and methods, reference solutions, and statistical evaluation. The third pillar is the experimental protocol, i.e. the set of procedures and experiment methodology applied for a specific test. Furthermore, in order to check all the aspects of a certain experiment, tests can be divided into level of realism, in which going deeper in the levels we will have much more reality and so variable parameters to check for robustness and effectiveness of algorithms in a real world. In particular:

- **Level 0**, fixes all parameters and aspects that could influence the exploration strategy, in order to provide an absolute controller evaluation environment. At this level of realism we are more focused on the strategy outcome regardless of computational resources and influencing parameters. With these constraints it is not possible to use the real required time for the exploration as performance indicator because clearly it would be meaningless, due to the facilitating conditions; the traveled distance is instead in this case a good evaluating parameter since it is only related with the throughput of the algorithm.

- **Level 1**, differently from the *Level* 0, we aim at testing the algorithm taking into account also computational requirements and a physical simulation is used to test the actual exploration time. In this work the simulation environment used is the 3D Gazebo Simulator.

- **Level 2**, is the employment of the real robot in the real world. As stated

before, this part is the crucial part in which the algorithm is finally deployed to test its validity in real world where a lot of variable exists, both in the environment setting and for the physical limitations of the robot. Anyway in this work this level or realism is not applied.

Further level of realism can be the field experiment, in which not only the working of an algorithm is tested, but also its robustness and efficiency in always more unstructured and uncontrolled conditions are applied in order to check the robot outcome in emergency and dangerous situations. Anyway for sake of simplicity in this work the experiments have been only carried in a Level 1 of abstraction. Nevertheless the aim of this work out of the thesis is to tests all the strategies proposed in a real environment in a future work.

5.5.3 Exploration Framework

The exploration framework is a fundamental choice in the strategy testing since it influences the overall system performance. In the case of a multi-robot system $R = \{r_1, ..., r_m\}$, the exploration procedure has been stated in [26] and costs of the following steps:

1. Initialize the model of the environment and set the initial plans to
   $P = \{P_1, ..., P_m\}$, where $P_i = \{\emptyset\}$ for each robot $1 \leq i \leq m$

2. *Repeat*

   - Navigate robots using the plans P;
   - Collect new measurements;

- Update the navigation map M;

Until **replanning condition is met**.

3. Determine goal candidates G from M.

4. *If* $|G| > 0$ assign goals to the robots

   - $(\langle r_1, g_{r1}\rangle, ..., \langle r_m, g_r m\rangle) = assign(R, G, M),\ r_i \epsilon R,\ g_i \epsilon G$;;
   - Plan paths to the assigned goals $P = plan(\langle r_1, g_{r1}\rangle, ..., \langle r_m, g_{rm}\rangle, M)$;;
   - Go to Step 2;

5. Stop all robots and navigate them to the starting position.

5.6 Tests

5.6.1 Generalities about the tests

First of all, every algorithm has been managed as centralized algorithm for a matter of limitations in terms of resources. Frontiers are directly evaluated in the centralized map created by the map merge node, which takes the single map from each robots and create a unique map. The reason for this choice is multiple. First of all because of hardware limitations. Even if three laptop have been used concurrently to perform tests, the use of multiple frontier detection nodes running each one for one robot, has put a strain on the pc resources, instead, using a unique map, makes possible to have more efficient results from simulations because there are not slowdowns caused by the hardware performances. Furthermore, the use of the merged map allows to take into account the information gained by all the robots. Not having an

algorithms for the global map broadcast of each robot map, this choice resulted to be the most efficient compromise. Secondly, the collision avoidance algorithm chosen for performing tests is the move_base from ROS Navigation Stack. This because testing all three approaches described in Section 4.2, the most effective in terms of robustness and performances it is for sure the move_base. Moreover, the use of the move_base node made possible to exploit the *make_plane*, which is a service of the move_base node to evaluate the path length from robot position to one hypothetical goal point. The service is requested from a robot only when it does not have a goal to be reached in its current plan. Lastly the robots initial positions are fixed to simplify the simulation observations and always for sake of simplicity, the map merge node is informed about the initial positions of the robots. This is a really strong assumption because in rare cases, in a real situation, we know the position of a robot and, in general, it would be better that the robot localizes itself without any information. Nevertheless, the use of the map_merge node makes necessary to do this assumption because, even if it allows to merge maps without known positions, the outcome of the merge is not always positive. To lighten the map merge workload to focus only on the multi robot exploration strategies, robots will always start from fixed positions, that are:

- $Robot_1 \rightarrow (1, 1)$;
- $Robot_2 \rightarrow (-1, 1)$;
- $Robot_3 \rightarrow (1, -1)$;
- $Robot_4 \rightarrow (-1, -1)$;

5.6.2 Scenario Setup

Tests have been performed in two different maps, in order to get more information about the outcome of an algorithm. This because the behavior of an algorithm is not the same for every map, in fact there are maps in which an algorithm performs better and other maps in which the outcome of the algorithm is worse. The first map, let's call it *house*, is a home-like environment, with eight rooms, one corridor and some separation walls.

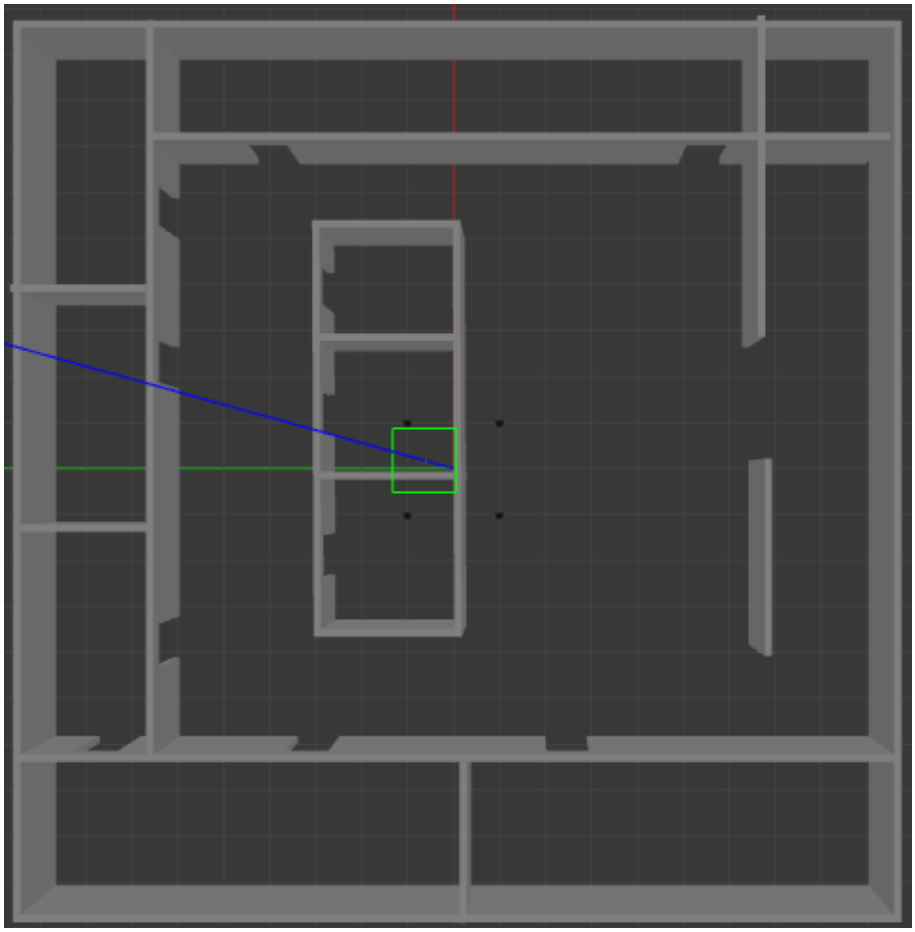


Figure 5.13: House map

Instead, the second map, called *boxed map*, is an open-space area with a number of boxes situated in a random order inside it.

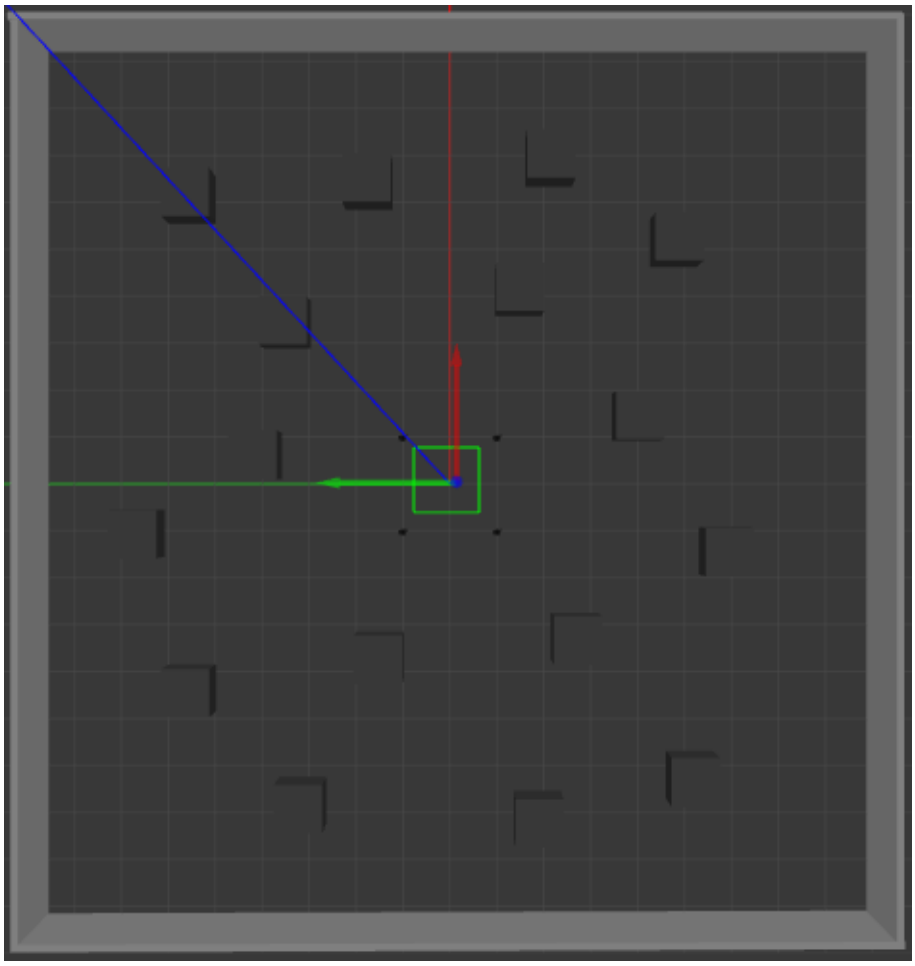


Figure 5.14: Boxed map

Both maps are $18m \times 18m$.

5.6.3 Test evaluation criteria

Using these two maps the five algorithms described have been tested. For each algorithm 20 tests have been conducted, 10 on the house map and 10 on the boxed map. During the tests three parameters have been collected:

1. The mapping time, i.e., the time needed to map the whole environment
2. The length of the longest path assigned between two consecutive goal points
3. The length of the longest path, i.e., the longest path traveled by any robot

5.6.4 Results

Results in the two maps are shown in the figures 5.15 and 5.16 where, to make they simpler to read, brief acronyms are used to identify the algorithm used.
In particular:

HM $\rightarrow$ Hungarian Method

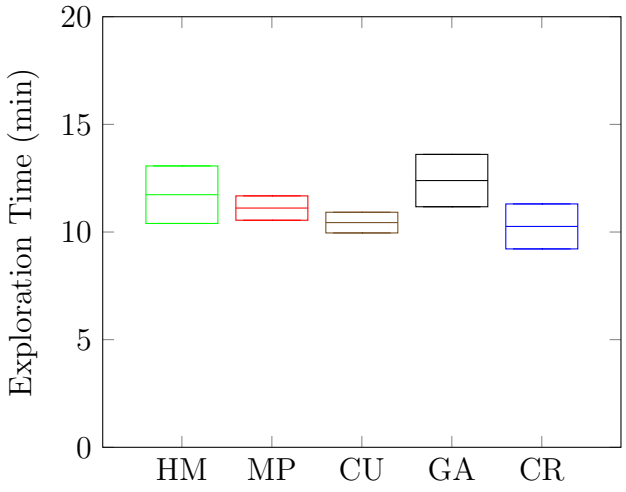
MP $\rightarrow$ MinPos

CU $\rightarrow$ Cost-Utility Function

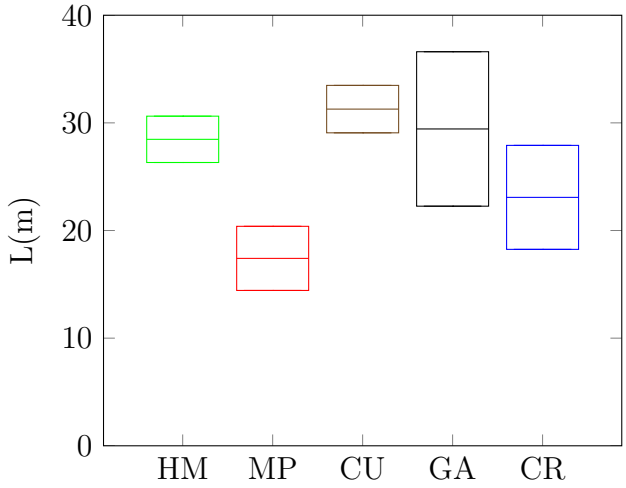
GA $\rightarrow$ Greedy Algorithm

CR $\rightarrow$ Circumcenter Rule

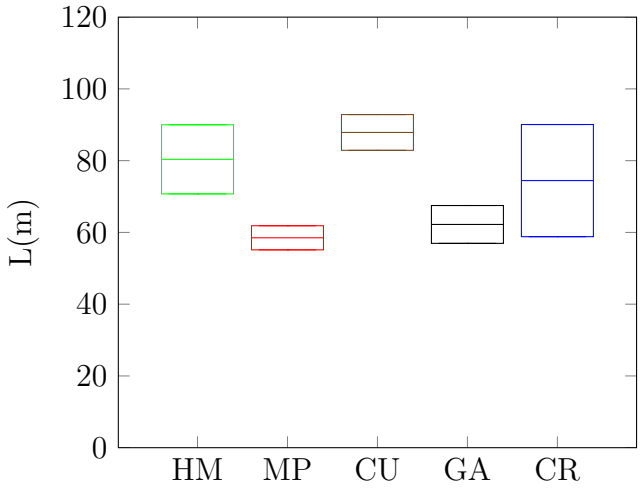
Results are shown in the following two pages.



(a) *Total Exploration Time*

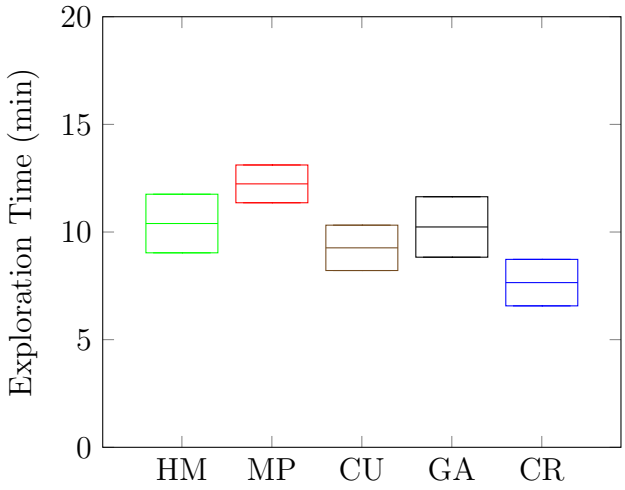


(b) *Maximum Length of two consecutive goal points*

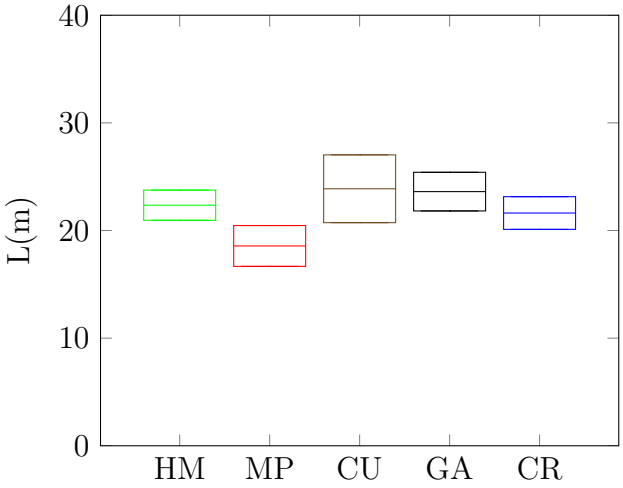


(c) *Length of the longest exploration path*

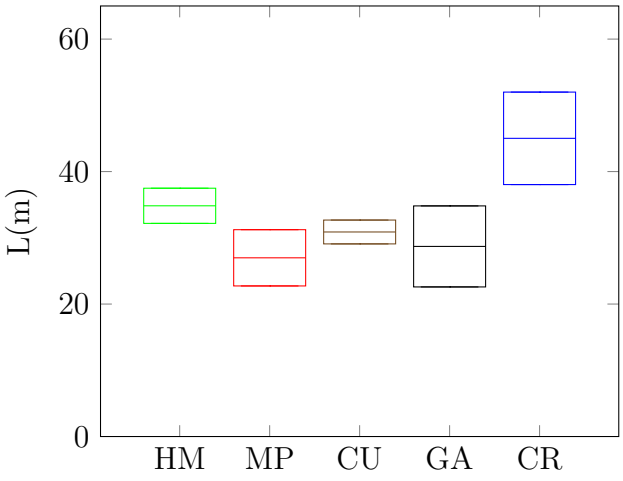
Figure 5.15: Tests on House Map



(a) *Total Exploration Time*



(b) *Maximum Length of two consecutive goal points*



(c) *Length of the longest exploration path*

Figure 5.16: Tests on Boxed Map

In these figures (5.15 and 5.16) both the mean value and the respective standard deviations are shown through boxes. In particular the central line of each box represents the mean value of tests that, as said before, are 10 per each algorithm. The width of the boxes represents the standard deviation around each mean value. In order to clarify this the following formula is shown to explain in analytical way the meaning of results:

$$\underset{mean\ value\ of\ algorithm\ i}{\mu_i} \pm \underset{standard\ deviation\ of\ algorithm\ i}{\sigma_i} \tag{5.16}$$

As it can be seen from results, Circumcenter rule has good results for both maps, especially in the case of the *boxed map*. Furthermore as anticipated before, tests in different maps provided different results in terms of efficiency. For example for the *house map*, the Hungarian method provides worse results with respect to the MinPos, whereas in the case of the boxed map we have the opposite behavior. This is the reason why more than one map has been exploited, so that now it is possible to assert for example that the Greedy algorithm has good results in a map in which there are small frontiers (i.e. *boxed map*), due to the nearness of the various boxes, instead of maps with more open areas, like the *house map*, in which it is important to take into account the hugeness of a frontier as the result of the Cost-Utility Function confirms. Anyway the outcome of the results is also influenced by some limitations of certain algorithms. For what concerns the Hungarian method, for example, its limitation is that it has to evaluate the next goal when all the robots require a goal. This means that if three robots out of four have already reached their goal, whereas a robot is still trying to reach its goal, the other three robot have to wait until the last one requires a new goal to have new positions to visit. To mitigate this inconvenient that may become very penalizing in the case a robot fails, two preventive measures

have been introduced. First a goal point remains so if and only if when the frontier detection algorithm updates new frontiers and deletes the old ones, the goal point is still a frontier point. For example, if a robot that was reaching a goal, moved near the goal point of another robot, updating the map and so the position of the new frontiers, that goal is deleted from the list of goal to visit and new goals are computed. This helps to not waste time for searching a frontier that does not exist anymore. This type of decision making frequency is known as immediate replanning (IR) that differs from the goal replanning (GR) in which the assignment of newly determined goals whenever a robot reaches its previously assigned goals. Another precautionary measure is to activate a timer when a robot stops before reaching its goal so that, if it remains stuck for a certain amount of time, it means that something did wrong in the planning task and it's better to eliminate the goal it has and, then, assign a new goal. The first security measure is present in all the other algorithms, in order to have more significant results. Anyway these checks and precautions in a certain way influence the efficiency of the code. That's why, another important parameter for the comparison of various algorithms is the computational burden and the simplicity of the code. For example even if the Hungarian method shown good results for the boxed world, the code to implementation is quite complex and it's really heavy to run. Even the MinPos needs a lot of calculations. Since the aim of the simulation is to evaluate the performance of an algorithm because we want to deploy it in a real robot in the future, it is really important to choose the most efficient algorithms also in terms of resources since physical robots have limited resources. The algorithms with computational simplicity are the Greedy algorithm, the Cost-Utility Function and the Circumcenter rule. Furthermore, these algorithms

can easily be transformed into decentralized algorithms. The difference about this observation is that, while the Greedy algorithm and the Cost-Utility Function do not require any communication between robots, the method proposed in this work, requires that each robot has to be able to send its communication to the other robots of the collective. This may cause problems in the real world, in which this communication can be complex in noisy situations and sometimes impossible for example when robots are very far from each other. Furthermore in a real situation in which one fundamental resource is the economical affordability of the project, the use of effective means of communication could be too exogenous. So, as usually happen for any engineering project, the choice of the strategy to use is a matter of cost-utility with respect to the application field. A resuming table to depict important parameters characterizing the exploration procedures is shown in table 5.9.

Table 5.9: Resuming table

Parameter	**Value**
Sensor model	360° Lidar with $10m$ sensing range
Environment map	Occupancy Grid map
Global path planner	Dijkstra's Algorithm (ROS *Navfn*)
Local path planner	*Dynamic Window Approach*
Path finder	*move_base/make_plan*
Coordination	Centralized
Decision-making	Immediate Replanning
Communication	Full without restriction
Initial Positions	Known

Chapter 6

Conclusions

In this thesis, five multi-robot frontier-based exploration strategies have been tested. The aim of each exploration strategy is to enable a team of robots to explore and map environments in the most efficient way. To accomplish the mapping task, robots have to be able to solve the SLAM problem. To this aim, each robot is equipped of:

- A localization algorithm, in particular, an Extended Kalman Filter has been implemented
- A collision avoidance package, the ROS *move_base* node has been exploited, with *Navfn* as global planner and *Dynamic Window Approach (DWA)* as local planner
- A mapping algorithm, the ROS *gmapping* package has been used, which exploits the occupancy grid map representation

When a group of robot explores an environment, the fundamental challenge to be addressed is the coordination of the team. In this work, only centralized coordination

approach has been tested because of resources limitation. The centralized algorithm makes use of the central map, created by the ROS *multirobot_map_node*. The map merge problem has been simplified assuming known initial position. This is a strong assumption because it may happen in real scenarios that initial positions of the robots are not known a priori. The centralized approach has the cons of being susceptible to failures, because if errors occur in the central entity, the whole multi-robot system fails. Tests have been conducted in the Gazebo simulation environment with four differential robots (TurtleBot3 Burger model), exploiting two different maps. Results show that the outcome of an exploration strategy depends on the map in which it is tested.

The five exploration strategies have been compared collecting three parameters:

- Mapping time, the time required by the team of robots to map the whole environment
- The length of the longest path assigned between two consecutive goal points
- The length of the longest path traveled by any robot

Finally, an Immediate Replanning (IR) decision-making frequency has been adopted, which means that as soon as the goal of a robot is no more a frontier, then a new goal is assigned to it.

6.1 Contributions

In Chapter 5, four state-of-art multi-robot exploration strategy have been analyzed and a novel exploration strategy approach, called *Circumcenter Rule*, has been presented. Circumcenter Rule tries to answer to the question: "How can I exploit at

best the use of multiple robots to make the exploration faster?". The proposed strategy provides the following answer: "Let's split up and sweep the area". To do this, a cost-utility function has been created in such a way the robot navigates through frontier points that are as close as possible to itself and as far as possible from the other robots. To this purpose, the concept of circumcenter has been introduced. When a robot asks for a goal, it finds the circle passing through the locations of the other three robots exploiting the Cramer's rule and then it extracts its center that, for definition, corresponds to the circumcenter. Finally, the robot will reach the frontier q that maximizes the quantity $g[q] = \frac{d_{cc}(q)}{L(q)}$, where $L(q)$ is the length of the collision-free path between the robot and the frontier q and $d_{cc}(q)$ is the distance between frontier q and the circumcenter found. The proposed method has shown positive results in terms of exploration time, classifying as the fastest exploration strategy among the other exploration strategies compared.

6.2 Future Work

Since the proposed multi robot strategy "Circumcenter Rule" has brought interesting result, future works will include tests of the multi-robot strategy in a real environment, as well as extend the formulation for more then four robot. Furthermore the implementation of this algorithm with physical robot would make possible to modify the centralized formulation into a decentralized one since, having performed the tests in a unique system didn't make possible to use decentralization in the simulations. Anyway a group of mobile robot has already been created but due to impossibility to be in lab because of Covid-19 emergency, the employment of these robot could not have been done. In particular the robot it would have been

used is shown in figure 6.1.

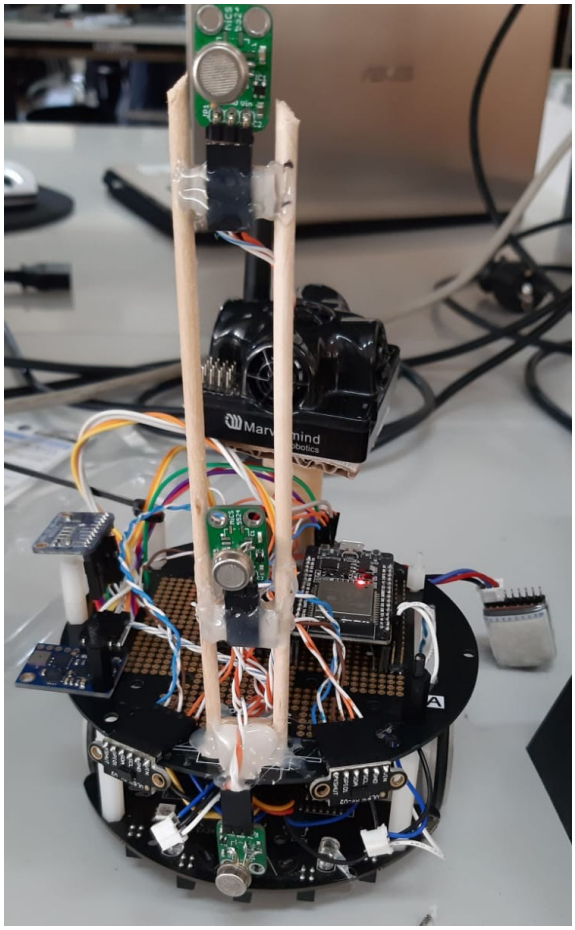


Figure 6.1: Real Robot

The schematic of all components and their interaction is depicted in figure 6.2.
In particular a brief list of the main components and their use is:

- ESP32: is a low-power system on a chip microcontroller equipped of Wi-Fi module and dual-mode Bluetooth. This MCU is used as high level layer and is the bridge between the user and the robot. ESP32 has 32-bit dual-core CPU operating at 160 or 240 MHz, 520 KB of SRAM and 448 KB of Rom. A huge utility of this instrument is that allows communication both through UART

and I^2C.

- STM32: is a microcontroller based on Arm® Cortex®- M7 core. These 32-bit MCUs have from 256 Kbytes to 2 Mbytes of Flash memory. In the robot configuration it is the lower level MCU that occupies of sending commands to motors and receiving information by encoders.

- ADS1115: is an 16-bit analog-to-digital converter, I^2C compatible and low-power device. It's scope is to convert the Mox sensor readings into digital information.

- IMU: not a specific model has been used as IMU. An example of IMU used is the GY-80, which is a 9-axis sensor, provided by a 3-axis gyroscope (L3G4200D 0x69), a 3-axis accelerometer (ADXL345 0x53), a 3-axis magnetometer (MC5883L 0x1E) and a Barometer + Thermometer (BMP085 0x77).

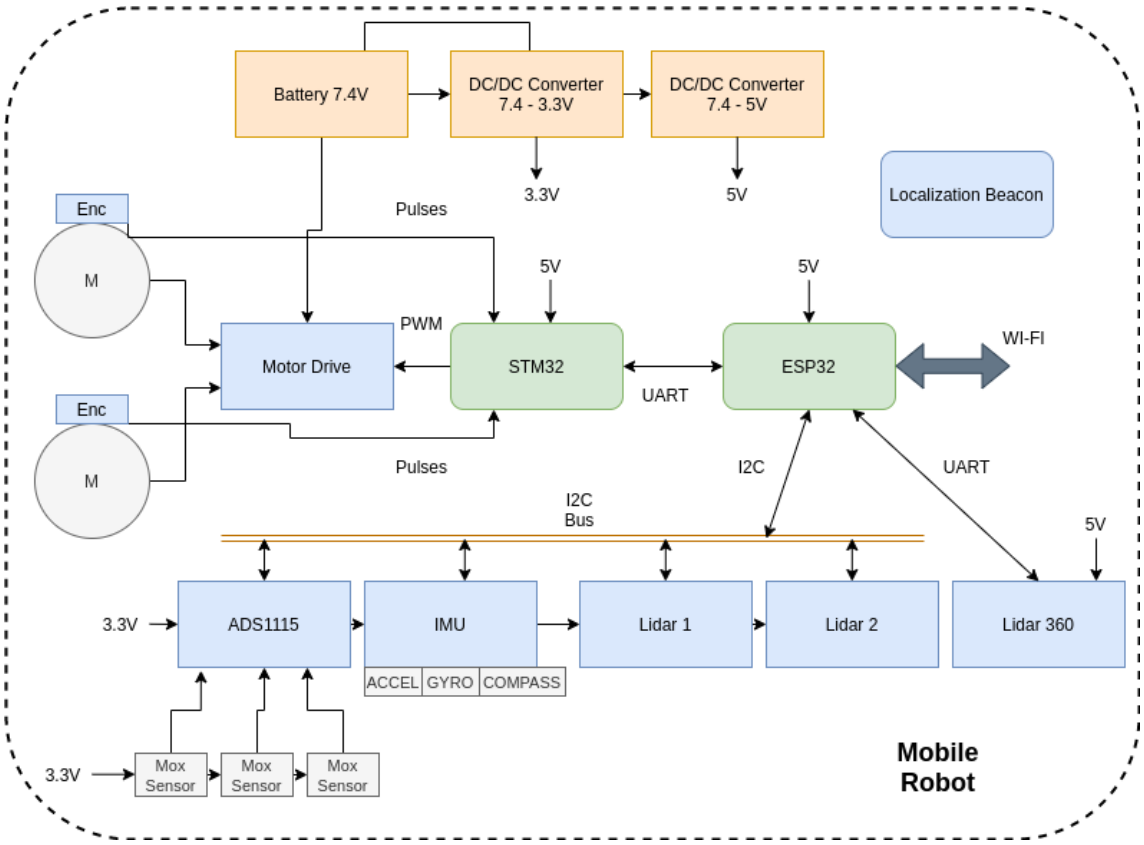


Figure 6.2: Hardware Components Interaction

List of Abbreviations

Abbreviation	Meaning
AMCL	Adaptive Monte Carlo Localization
APF	Artificial Potential Field
BLE	Broadcast of Local Eligibility
BFS	Breadth-First Search
CPU	Central Processing Unit
CR	Circumcenter Rule
CU	Cost-Utility Function
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
GA	Greedy Algorithm
GPS	Global Positioning System
GR	Goal Replanning
HM	Hungarian Method
IA	Iterative Assignment
ICC	Instantaneous Center of Curvature
IMU	Inertial Measurement Unit
IR	Immediate Replanning

Abbreviation	Meaning
KF	Kalman Filter
MCL	Monte Carlo Localization
MCU	Micro Controller Unit
MinPos	Minimum Position
MP	MinPos
MPC	Model Predictive Control
NBV	Artificial Potential Field
PF	Particle Filter
RBPF	Rao-Blackwellized Particle Filter
RCP	Robot Center Point
ROS	Robot Operating System
RVIZ	ROS Visualizer
SIR	Sampling Importance Resampling
SLAM	Simultaneous Localization And Mapping
SPT	Shortest Path Three
SRAM	Static Random Access Memory
TEB	Time-Elastic-Band
UART	Universal Asynchronous Receiver Transmitter
URI	Uniform Resource Identifier
VFH	Vector Field Histogram
WFD	Wavefront Frontier Detection
XMLRPC	XML-Remote Procedure Call
XML	eXtensible Markup Language

List of Figures

List of Tables

Bibliography

[1] Breadth first search, https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/.

[2] Alaa A.Ahmed, Turki Abdalla, and Ali Abed. Path planning of mobile robot by using modified optimized potential field method. *International Journal of Computer Applications*, 113:6–10, 03 2015.

[3] Francesco Amigoni, Alberto Quattrini Li, and Dirk Holz. Evaluating the impact of perception and decision timing on autonomous robotic exploration. In *2013 European Conference on Mobile Robots*, pages 68–73. IEEE, 2013.

[4] Francesco Amigoni, Monica Reggiani, and Viola Schiaffonati. An insightful comparison between experiments in mobile robotics and in science. *Autonomous Robots*, 27(4):313, 2009.

[5] Francesco Amigoni and Viola Schiaffonati. Good experimental methodologies and simulation in autonomous mobile robotics. In *Model-based reasoning in science and technology*, pages 315–332. Springer, 2010.

[6] Francesco Amigoni and Viola Schiaffonati. *Good Experimental Methodologies*

and Simulation in Autonomous Mobile Robotics, volume 314, pages 315–332. 09 2010.

[7] Torsten Andre, Daniel Neuhold, and Christian Bettstetter. Coordinated multi-robot exploration: Out of the box packages for ros.

[8] Osman Balci and SI Gass. Verification, validation and testing of models. *Encyclopedia of operations research and management science*, pages 1618–1627, 2013.

[9] Antoine Bautin, Olivier Simonin, and François Charpillet. Minpos : A novel frontier allocation algorithm for multi-robot exploration. pages 496–508, 10 2012.

[10] Curt Bererton, Geoffrey Gordon, and Sebastian Thrun. Auction mechanism design for multi-robot coordination. 01 2003.

[11] A. Birk and S. Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, 2006.

[12] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 341–346 vol.1, 1999.

[13] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[14] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[15] Robert Daily and David M Bevly. Harmonic potential field path planning for high speed vehicles. In *2008 American Control Conference*, pages 4609–4614. IEEE, 2008.

[16] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1322–1328. IEEE, 1999.

[17] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[18] Arnaud Doucet, Nando De Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. *arXiv preprint arXiv:1301.3853*, 2013.

[19] Arnaud Doucet and N.J Freitas, N.and Gordon. *Sequential Monte-Carlo Methods in Practice*, volume 1. 01 2001.

[20] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[21] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[22] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, USA, 2nd edition, 2010.

[23] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Experiments in sensing and communication for robot convoy navigation. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 2, pages 268–273. IEEE, 1995.

[24] Gregory Dudek, Michael RM Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, 1996.

[25] João Fabro, Rodrigo Guimarães, André Oliveira, Thiago Becker, and Vinícius Brenner. *ROS Navigation: Concepts and Tutorial*, volume 625, pages 121–160. 02 2016.

[26] Jan Faigl and Miroslav Kulich. On benchmarking of frontier-based multi-robot exploration strategies. pages 1–8, 09 2015.

[27] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4:23 – 33, 04 1997.

[28] Brian Gerkey and Maja Mataric. Sold!: Auction methods for multirobot coordination. pages 758–768, 10 2002.

[29] Clara Gomez, Alejandra Hernández, and Ramon Barber. Topological frontier-based exploration and map-building using semantic information. *Sensors*, 19:4595, 10 2019.

[30] Héctor González-Baños and Jean-Claude Latombe. Navigation strategies for exploring indoor environments. *I. J. Robotic Res.*, 21:829–848, 10 2002.

[31] The GPS. https://www.math.tamu.edu/ dallen/physics/gps/gps.htm, 2017. [Online; accessed 3-October-2020].

[32] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[33] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. Ros navigation: Concepts and tutorial. In *Robot Operating System (ROS)*, pages 121–160. Springer, 2016.

[34] Jiří Hörner. Map-merging for multi-robot system, 2016.

[35] Kwang-Min Jung, Hea-Jae Lee, and Kwee-Bo Sim. Study on path planning for autonomous mobile robot using potential field. *Journal of Korean Institute of Intelligent Systems*, 19(5):737–742, 2009.

[36] Matan Keidar and Gal Kaminka. Robot exploration with fast frontier detection: Theory and experiments. volume 1, pages 113–120, 06 2012.

[37] Matan Keidar and Gal A Kaminka. Robot exploration with fast frontier detection: theory and experiments. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 113–120, 2012.

[38] K. Konolige, Dieter Fox, B. Limketkai, J. Ko, and Benjamin Stewart. Map merging for distributed robot navigation. pages 212 – 217 vol.1, 11 2003.

[39] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by

tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382, 1991.

[40] Jun S Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and computing*, 6(2):113–119, 1996.

[41] Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo de la Escalera. Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018, 2018.

[42] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. In *Sensor devices and systems for robotics*, pages 253–276. Springer, 1989.

[43] James R. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[44] Kevin P Murphy. Bayesian map learning in dynamic environments. In *Advances in Neural Information Processing Systems*, pages 1015–1021, 2000.

[45] G Naus, S Coenen, and M van de Molengraft. Implementation of the potential field method for motion planning at the turtles.

[46] Luciano CA Pimenta, Alexandre R Fonseca, Guilherme AS Pereira, Renato C Mesquita, Elson J Silva, Walmir M Caminhas, and Mario FM Campos. Robot navigation based on electrostatic field computation. *IEEE Transactions on magnetics*, 42(4):1459–1462, 2006.

[47] Ioannis Rekleitis. A particle filter tutorial for mobile robot localization. 01 2004.

[48] Jing Ren, Kenneth A McIsaac, and Rajnikant V Patel. Modified newton's method applied to potential field-based navigation for mobile robots. *IEEE Transactions on Robotics*, 22(2):384–391, 2006.

[49] ROBOTIS e-Manual. Turtlebot3, 2017. [Online; accessed 2-October-2020].

[50] ROS Wiki. costmap_2d, 2015. [Online; accessed 14-September-2020].

[51] ROS Wiki. gmapping, 2015. [Online; accessed 14-September-2020].

[52] ROS Wiki. move_base, 2015. [Online; accessed 14-September-2020].

[53] ROS Wiki. navigation, 2015. [Online; accessed 14-September-2020].

[54] ROS Wiki. Ros/introduction, 2015. [Online; accessed 14-September-2020].

[55] ROS Wiki. Multirobot map merge, 2016. [Online; accessed 14-September-2020].

[56] ROS Wiki. navfn, 2016. [Online; accessed 25-September-2020].

[57] Sajad Saeedi, Liam Paull, Michael Trentini, and Howard Li. Occupancy grid map merging for multiple robot simultaneous localization and mapping, international journal of robotics & automation. *International journal of robotics and automation*, 30(2):149–157, 2015.

[58] Cyrill Stachniss. *Robotic Mapping and Exploration*, volume 55. 01 2009.

[59] Anthony Stentz and M. Dias. A free market architecture for coordinating multiple robots. 01 2000.

[60] Matti Tedre and Nella Moisseinen. Experiments in computing: A survey. *TheScientificWorldJournal*, 2014:549398, 02 2014.

[61] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[62] Anirudh Topiwala, Pranav Inani, and Abhishek Kathpal. Frontier based exploration for autonomous robot. *arXiv preprint arXiv:1806.03581*, 2018.

[63] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146)*, volume 2, pages 1572–1577. IEEE, 1998.

[64] Autonomous Robot Vehicles. Ij cox and gt wilfong, eds. *Spring-Verlang, New York*, 1990.

[65] Barry Werger and Maja Mataric. Broadcast of local eligibility: Behavior-based control for strongly cooperative robot teams. 04 2000.

[66] Wikipedia contributors. Circumscribed circle — Wikipedia, the free encyclopedia, 2020. [Online; accessed 14-September-2020].

[67] Wikipedia contributors. Cramer's rule — Wikipedia, the free encyclopedia, 2020. [Online; accessed 14-September-2020].

[68] Wikipedia contributors. Dijkstra's algorithm — Wikipedia, the free encyclopedia, 2020. [Online; accessed 13-September-2020].

[69] Wikipedia contributors. Hungarian algorithm — Wikipedia, the free encyclopedia, 2020. [Online; accessed 14-September-2020].

[70] Wikipedia contributors. Lidar — Wikipedia, the free encyclopedia, 2020. [Online; accessed 13-September-2020].

[71] T Wilschut. An obstacle avoidance algorithm for a mobile robot based upon the potential field method. *Eindhoven: University of Technology*, pages 6–8, 2011.

[72] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.

[73] Brian Yamauchi. Frontier-based exploration using multiple robots. pages 47–53, 01 1998.

[74] Brian Yamauchi. Decentralized coordination for multirobot exploration. *Robotics and Autonomous Systems*, 29(2-3):111–118, 1999.

[75] Leon Jung Darby Lim Yoonseok Pyo, Hancheol Cho. *ROS Robot Programming (English)*. ROBOTIS, 12 2017.

[76] Kaiyu Zheng. Ros navigation tuning guide. *arXiv preprint arXiv:1706.09068*, 2017.