Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2019

José Nogueira

Sistema robótico para medição de áreas ocupadas em armazéns

A robotic system for measuring occupied areas in indoor warehouses

Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2019

José
Nogueira

Sistema robótico para medição de áreas ocupadas em armazéns

A robotic system for measuring occupied areas in indoor warehouses

José
Nogueira

Sistema robótico para medição de áreas ocupadas em armazéns

A robotic system for measuring occupied areas in indoor warehouses

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Miguel Armando Riem de Oliveira, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor Artur José Carneiro Pereira** Professor auxiliar da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)
vogais / examiners committee	**Doutor Andry Maykol Pinto** Investigador Sénior do INESC TEC - Porto (arguente)
	Professor Doutor Miguel Armando Riem de Oliveira Professor auxiliar da Universidade de Aveiro (co-orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que me ajudaram durante estes anos.

Resumo

Cada vez mais surge a necessidade de automatizar os processos na cadeia de produção com o objetivo de melhorar resultados e executar tarefas de forma mais rápida. Esta dissertação pretende desenvolver um sistema robótico autônomo que permita medir as áreas ocupadas num armazém, um problema comum em situações onde é necessário armazenar cargas não contententorizadas. Para o desenvolvimento de software para um agente robótico deste tipo é necessário desenvolver soluções para vários problemas complexos em robótica: mapeamento, localização e navegação.

Na solução proposta, o mapeamento é efetuado em dois momentos distintos. Inicialmente é feito um mapeamento com o objetivo de reconhecer o ambiente onde o robô vai circular posteriormente, utilizando o mapa gerado nesta fase. Neste mapeamento inicial foi utilizado o algoritmo Gmapping. Numa segunda etapa, durante o funcionamento em produção do robô, é feito outro mapeamento com o objetivo de localizar e representar no mapa os locais ocupados. Neste caso foram desenvolvidos alguns algoritmos com recurso às bibliotecas Point Cloud Library e Octomap.

O processo de localização é um requisito para que o robô possa navegar autonomamente no espaço. Foram consideradas várias abordagens, tendo sido utilizada uma abordagem adaptativa de localização de Monte Carlo. Este processo de localização ativa é importante para minimizar os erros que possam surgir na localização do robô, minimizando os erros da segunda etapa de mapeamento.

Este processo de localização também é fundamental para que o robô possa navegar de forma autônoma até um determinado ponto no mapa. No entanto é necessário traçar um percurso para o robô seguir. Para gerar o percurso é aplicado o algoritmo de planeamento A*.

No final desta dissertação foi desenvolvido um sistema robótico capaz de circular em determinados ambientes e verificar que areas foram ocupadas. O acesso aos dados recolhidos pelo robô são disponibilizados através de uma interface web, que permite consultar o mapa gerado, as áreas ocupadas com dados referentes a cada uma, a visualização em tempo real da área frontal do robô e a possibilidade de interagir com o robô podendo deslocá-lo.

Abstract

Nowadays is increasing the need to automate processes in the production chain to improve results and perform tasks more quickly. This dissertation intends to develop software to an autonomous robotic system that allows measuring the occupied areas in a warehouse, a common problem in situations where it is necessary to store uncontainable loads. For software development to this type of robotic agent, it is necessary to develop solutions for several complex problems in robotics: mapping, localization and navigation.

In the proposed solution, the mapping is performed in two different moments. Initially, a mapping is done with the objective of recognizing the environment where the robot will circulate, using the map generated in this phase. In this initial mapping, it is used the Gmapping algorithm. In a second step, during the robot operation, another mapping is done to locate and map the occupied places. In this case, we develop some algorithms using the Point Cloud Library and Octomap libraries.

The localization process is a requirement for the robot to operate autonomously in a defined space. Several approaches were considered, using an adaptive Monte Carlo localization approach. This active localization process is important to minimize the errors that may arise in the location of the robot, reducing also the errors of the second stage of mapping.

This localization process is also fundamental so that the robot can navigate autonomously up to a certain point on the map. However, it is necessary to plot a route for the robot to follow. To generate the route, the planning algorithm A* is applied.

At the end of this dissertation was developed a robotic system capable of circulating in certain environments and verifying which areas were occupied. Access to the data collected by the robot is possible through a web interface, which allows consulting the map generated, the areas occupied with data referring to each one, the real-time visualization of the front area of the robot and the possibility of interacting with the robot and can move it.

Contents

List of Figures

List of Tables

Glossary

AMCL Adaptive-Monte Carlo Localization. i, iii, iv, 6, 12–15, 31, 44–46

AMR Autonomous Mobile Robot. iii, 21

LiDAR Light Detection And Ranging. iii, 2, 5, 11, 21–24, 26, 29, 31, 33, 46

PCL Point Cloud Library. 26, 27, 35

PF Particle filter. 10–12

ROS Robot Operating System. 22, 23, 25, 26, 29

SLAM Simultaneous Localization and Mapping. i, 5, 6, 10, 29

urdf Unified Robot Description Format. iii, iv, 23, 24

WSGI Web Server Gateway Interface. 28

Chapter 1

Introduction

From the early beginnings, it is noticeable the mankind strive to evolve and develop ways of making life easier. Initially, advances were approached by handcraft, however, the industry eventually emerged with the goal of making these processes faster and more efficient. At that time, the industry was small and very limited, and as such only tried to do things in series. However, the industry evolved over time and so bigger and more complex machines began to be built in order to automate the simplest tasks. Eventually, a problem arose from the vicious cycle created, since industry always want to automate something else. This happens because processes when they are automated, they become more efficient and economically viable. This yield can be in faster and steady productions or services analysing several aspects.

With the current industry emerging, there is an increased need for multiple services to be automated. For example, stock management or even space management. In these cases, having an autonomous robot that moves through a warehouse and does an analysis of the occupied space can be extremely efficient, since it will be possible to present data in a faster and more accurate way without humans required to this process.

Looking at this increasingly recurrent problem, a software was developed for a robot along this dissertation with the goal of making the tasks of space management more simpler, in storage places of loads.

1.1 Motivation and Objectives

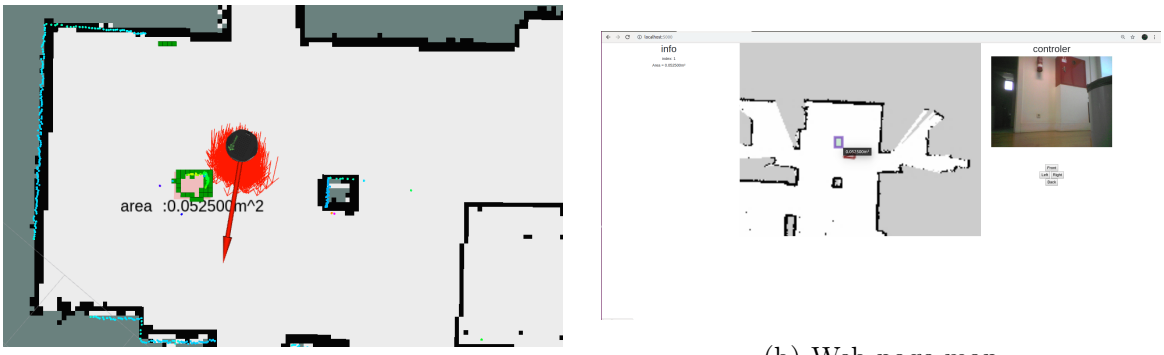
Currently, there is a high demand to automate several processes in different situations and environments with the purpose of assisting the community, in this case, a possibility to measure occupied areas autonomously. This evolution can not only support the workers but also improve the working conditions, as all information is easily and quickly recognized. It can also present information collected in a faster, more coherent, and constant way.

In this context, it was developed a solution to be apply in a robot to move in a real environment. First, the environment is recognised, then the robot can be either autonomously or manually navigated in the space to find places that have been occupied. The collected data can be accessed remotely through a web page, where the environment map is displayed containing the information about the occupied areas, with visual information and real measures of the areas occupied. It is also possible to control the robot movement and visualise the area in front of it, using camera on board.

The first objective of this dissertation is the development of a robotic platform and software that allows the robot to be able to locate itself in a certain environment and to be able to navigate autonomously. This task considers the need to analyse and interpret the robot's surroundings, resorting solely to the signal of a LiDAR.

The second objective is having the ability to identify the locations that have been occupied, based on the sensor data that is collected. In this task, it is necessary to keep in mind that there are objects that remain static and these are the ones that should be analysed and the corresponding occupancy area calculated. However, other interactions may also occur in this environment, as well as people moving around or other objects that are constantly changing locations that should be processed accordingly.

The third objective is the development of a web interface, for a user to control the robot and see the map with the various areas of interest and the analysis performed on each one. Figure 1.1 shows an example of the information calculated by the robot and the web interface available for the user.



(a) Occupied area on rviz

(b) Web page map

Figure 1.1: On the left, an example of a scenario where it is possible to see the map generated by the robot (black dots), the localisation of the robot on the map (red arrows), the measure of the LiDAR sensor (blue dots), and the calculated occupied area (green region). On the right, an example of the web interface where the user can access the robot's data in real time.

1.2 Document structure

This dissertation is divided into four chapters:

Chapter 2 presents an analysis of the state of the art. The main problems related to mobile robotics and autonomous navigation are reviewed. To solve these problems, several possibilities are presented that currently exist, and in a more detailed way the most appropriate solution to this implementation.

In Chapter 3 we present the case that was studied. We present a brief description of the tools used.

In Chapter 4 we describes the implementation that was developed, as well as the presented results that demonstrate its operation.

In Chapter 5 we present an analysis of what was developed, as well as the reference of aspects that can be improved in the future.

Chapter 2

Mapping and localization approaches

In robots with autonomous movement, one of the most important problems to solve is the ability to locate the robot in an unknown space and at the same time to create a map of the space. There are several reasons for this, being one of them the uncertainty on the measurements, since the sensors used by the robot are not completely accurate, having always some associated error.

For example, a LiDAR may not be accurate and may have small variations between equal measures, depending on their quality. On the other hand, we can also use odometry, which when calculated using encoders, the error is cumulative and may be due to small errors of measurement of the wheels or distance between the axes, as along the movement some drift occurs.

In certain situations, there are also objects and people that move on the scene, and this is usually data that we do not intend to represent on a map for future use, as they are merely temporary. In this way, we have algorithms that are robust and able to represent the map correctly and to locate quickly in diverse environments and conditions.

With the presence of cumulative errors, such as is the case of odometry, which accumulates error about the position in which the robot is localized, it is necessary to have other localization mechanisms at the same time to reduce this type of errors. Thus, while generating a map, the projection is the closest to reality. For this reason, simultaneous localization and mapping (SLAM) algorithms are used, because the two interconnected ones allow better results.

When the map is already known, the demands on the location are slightly different. It is necessary to have the ability to position the robot on the map, using the data that it is being collected. For this, it is common to use probabilistic models, such as the adaptive

Monte Carlo localization approach (AMCL), which uses particle filters to converge to the position with a higher likelihood of similarity.

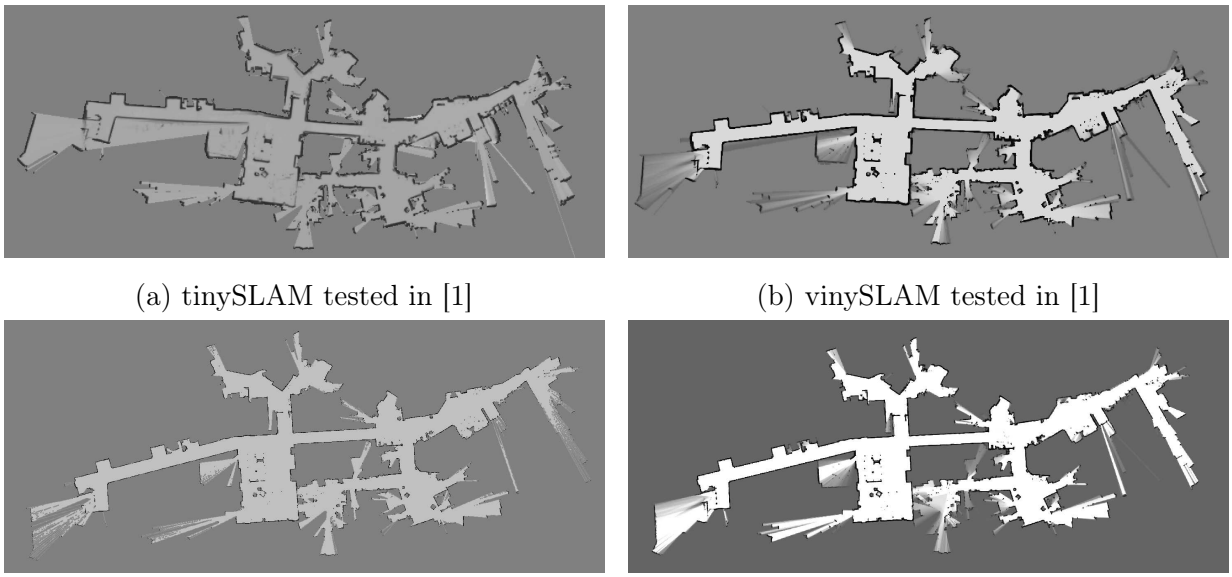
2.1 SLAM

In SLAM, different algorithms can be used. These may have different requirements, such as required hardware, and required processing, ultimately resulting in better or worse results. However, always these results also depend on the environment in which the robot is used. Some algorithms that we can use are the following:

- tinySLAM [1]
- vinySLAM [1]
- HectorSLAM [2]
- KartoSLAM [2]
- CoreSLAM [2]
- LagoSLAM [2]
- GMapping [1]
- Cartographer [1]

A more detailed comparative analysis of these algorithms can be seen in [1] and [2]. However, from these articles, we can make some conclusions to verify which algorithm adapts more to the solution that is desired.

With the use of tinySLAM, as can be seen in Figure 2.1a, it is noticeable that the map is poorly constructed, with some overlaps of some spaces occurring. This type of deformation makes the use of this method impracticable to generate a map for future use, in the autonomous location.



(a) tinySLAM tested in [1]

(b) vinySLAM tested in [1]

(c) GMapping tested in [1]

(d) Cartographer tested in [1]

Figure 2.1: tinySLAM, vinySLAM, GMapping and Cartographer

In Figures 2.1b, 2.1c and 2.1d, we can see an example of a map obtained respectively by vinySLAM, GMapping and Cartographer. Visually these three figures (2.1b, 2.1c and 2.1d) are more consistent than Figure 2.1a, but the three are similar. The most noticeable differences are at the boundaries of the mappings and this may be due to small differences in the path, during mapping, at these extremities. However, with the tests made in [1], it can be seen that vinySLAM has higher errors. GMapping and the Cartographer have very small errors, and it should be noted that the Cartographer is faster.

Figure 2.2 presents an example regarding the use of HectorSLAM in [2]. In this case, since the odometry is used, does not present sufficiently precise results when compared with others. It is possible to verify that the localization mechanisms are not very precise and cause distortions, such as unwanted curvatures in the walls. However, it can present good results in other types of implementations.

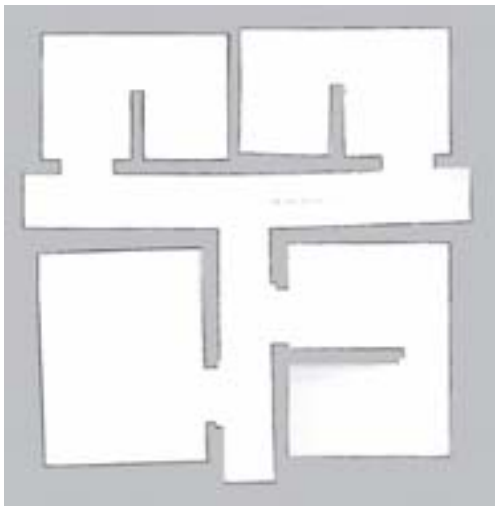


Figure 2.2: HectorSLAM tested in [2]

As HectorSLAM, KartoSLAM in Figure 2.3, also presents the same type of error, an inaccurate location correction, which results in mappings with some errors.

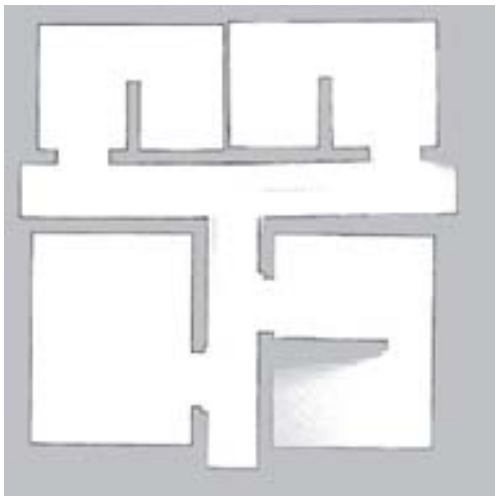


Figure 2.3: KartoSLAM tested in [2]

CoreSLAM, Figure 2.4, becomes completely unusable for larger areas. It is probable that the mapping started in the upper left area because it is better defined, however, as the mapping evolves and there is more need to take into account the errors that accumulate, a good handling of this type of error is not done, building a rather distorted map.

Figure 2.4: CoreSLAM tested in [2]

In the case of LagoSLAM, Figure 2.5, in the realization of the mapping presents constant deformations in the corners. It is to be noted that the upper areas are almost connected, this may be due to the accumulation of errors when travelling in the hall, following a certain offset.

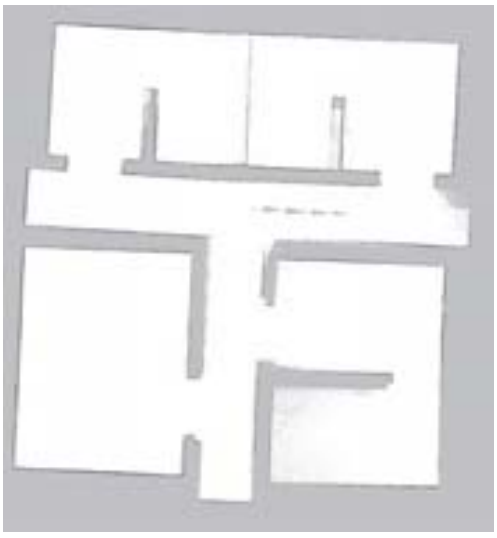


Figure 2.5: LagoSLAM tested in [2]

As we can see again GMapping, Figure 2.6, gives good results when compared to the other SLAM algorithms. This algorithm builds a more consistent, well-demarcated and well-framed map.

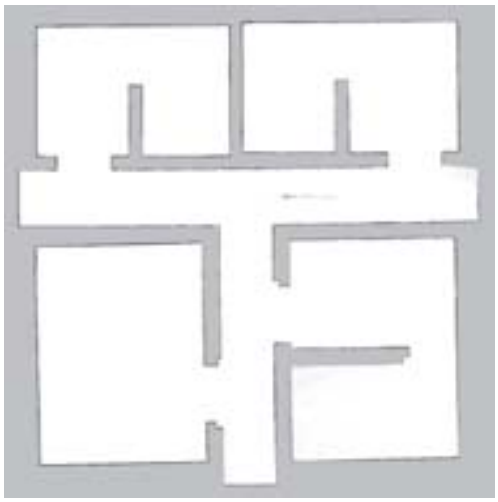


Figure 2.6: GMapping tested in [2]

According to the demonstration in [2], GMapping presents better results for the implementation that we intend, when compared with, HectorSLAM, KartoSLAM, CoreSLAM and LagoSLAM. In [1], we can see comparisons between tinySLAM, vinySLAM, GMapping and Cartographer, the latter two of which present better results.

Given the options of SLAM algorithms already developed and tested, GMapping has good results in several studies [1] [2]. For this reason, and the compatibility with the hardware used, it was decided to use this algorithm for the initial acquisition of the map. However, this algorithm have a high computational cost, which could be reduced by using a approach like this [3], but for this case we need a very accurate approach to minimize the erros later and in this task the runtime is not a critical factor.

GMapping is one of the most used algorithms in SLAM, and as it shows good results and was already being used in similar previous work it was chosen to maintain this use, although as we can see in [1], the Cartographer could be faster, however, the execution time is not a critical factor in this implementation, so we opted to maintain the use of GMapping.

2.1.1 GMapping

Regarding the operation of GMapping, we can find detailed information in [4]. This is one of the most used algorithms in robots. It uses methods proposed in Rao-Blackwellized PF SLAM[5]. For best results at the level of particle filter (PF), it is necessary to provide a large number of points. However, this, in turn, will increase the computational effort,

since it is necessary to process more data, as referred to in [5]. The depletion problem associated with the PF re-sampling process also decreases the accuracy of the algorithm. This is because the points less likely to be correct are eliminated.

The GMapping proposal in minimizes the depletion problem, making adaptive re-sampling, so it is only done when necessary [4]. It was also proposed by the authors to take into account the latest observations of the robot and not only its movement. This allows reducing the error in the position of the robots, and with the use of scan matching, the uncertainty is reduced so that fewer particles are needed to have the same accuracy, thus less computational effort[4].

Figure 2.7: Map generated by GMapping in the ROS environment using this robot on the 1st floor of the IEETA

2.2 Localization

The location is a very important factor for an autonomous robot. To be able to move between two different points, in a known map, it is necessary that the robot has the capability to localize autonomously in the same map and in real time. For this to be possible, it is necessary to use algorithms capable of analyzing the data coming from the sensors, in this case a 2D LiDAR and encoders, and find the most probable position on the map.

Some examples of algorithms used to handle this type of task are:

- Markov localization [6];
- Monte Carlo localization [7].

However, often these algorithms only serve as a base, because some adjustments are made to be possible to obtain better performances.

In [8], we can see some tests done on these algorithms and other adapted versions. Thus, it can be seen that the AMCL presents fairly assertable execution times, and it has good location precision even in the presence of noise and for an environment with some changes.

2.2.1 AMCL

The AMCL is a probabilistic algorithm based on PF. It aims to solve the problem of localizing a robot that moves on the map already known. It is also known by the Kullback-Leibler Distance (KLD) sampling because it measures the Kullback-Leibler distance approximation error to reduce the approximation error caused by the sample-based representation of the PF.

The main difference compared to the traditional Monte Carlo location (MCL) is the possibility of real-time implementation of the sample set size [9]. With this adaptive approach, a larger number of samples is used for higher levels of uncertainty and a smaller number when the uncertainty is lower [10]. This method greatly reduces computational effort. In the Figures 2.8, 2.9, 2.10, 2.11 can be see the progression of the AMCL, it can be seen that new readings of the sensors are obtained, the precision in the location increases considerably.

Initially, as we can see in Figure 2.8, points are scattered all over the map, where it is possible that the robot is. At the moment there is a very high uncertainty because it is a random distribution.

Figure 2.8: AMCL starting algorithm, with the red arrows representing possible positions of the robot scattered on the map

As the robot moves, LiDAR data is accumulated and periodically compared to the map, for each dispersed point on the map. Like this comparison, the points whith less similarity are discarded and more points are generated around the more similar points, as one can begin to see in the Figure 2.9, in which the points begin to group into small groups.

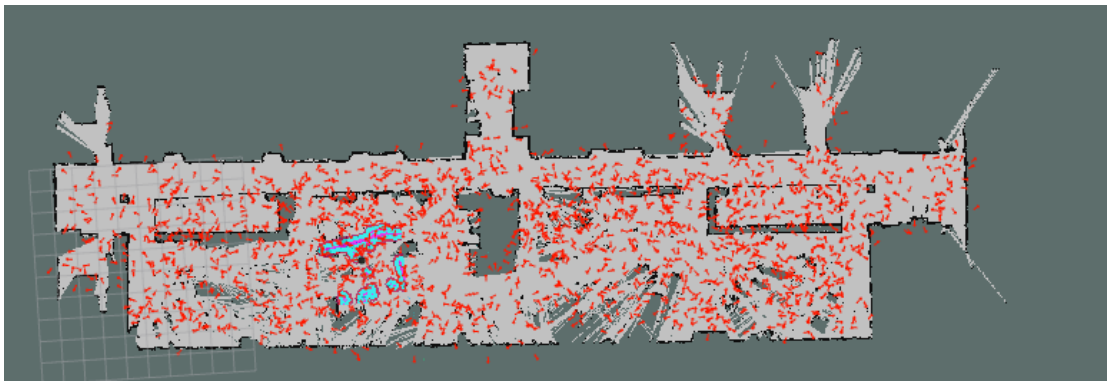


Figure 2.9: AMCL after moving a 1 meter, with the red arrows representing possible positions of the robot on the map, starting to nest

The more we move, the more data we are using to compare, so there are some groups of points where the robot is likely to be. As can be seen in Figure 2.10, where the points are grouped more, getting less but larger groups.

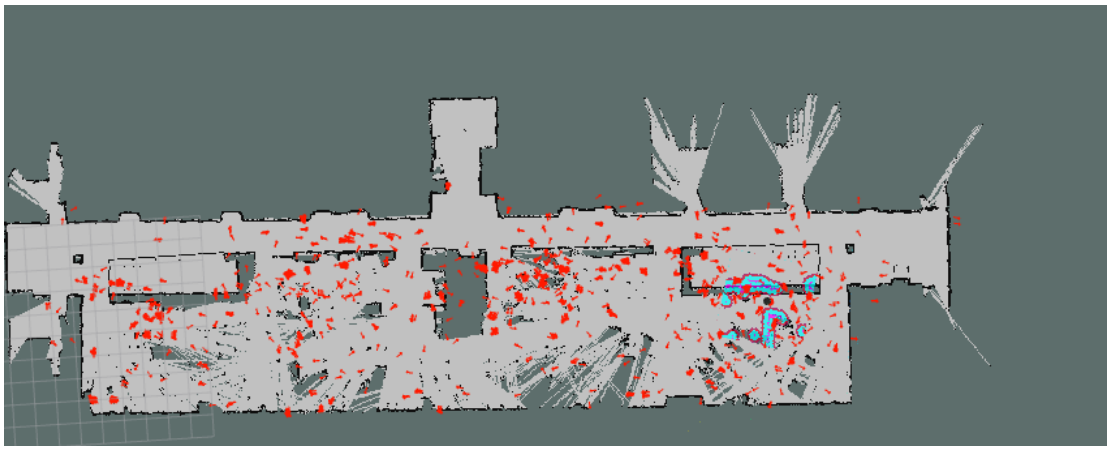


Figure 2.10: AMCL after moving a 4 meters, with the red arrows representing possible positions of the robot on the map, with the black circle representing the most probable position of the robot until this moment

With this process, considering the accumulation of data, it is possible to converge to a point where the robot is most likely to be located, as can be seen in the Figure 2.11. However, there is still a cloud of points around to periodically correct the position and prevent the drift errors that may occur in those intervals.

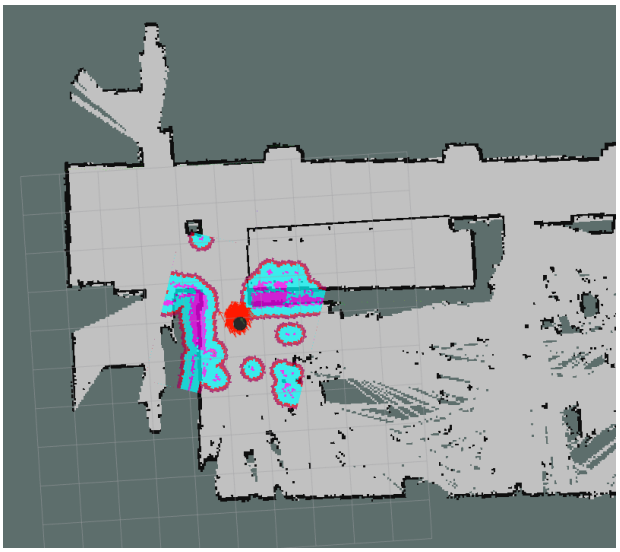


Figure 2.11: AMCL after some movement, with the red arrows representing possible positions of the robot on the map and the black circle the most likely position of the robot. The cloud (red, blue, and purple) is the representation of the cost_path used by A* to inflate obstacles.

To better understand the operation of the AMCL at a more practical level, it can be considered only one dimension. In the Figures (2.12, 2.13, 2.14), we can see how the algorithm converges to a certain location, based on the movement of the robot and the vision.

Figure 2.12: AMCL algorithm in t=0, example in 1D

Figure xxx, refers to an initial moment and the position of the robot is unknown. However, it is in front of the door. For this reason at the end of this first interaction, three possible locations for the robot appear, referring to the three doors.

Figure 2.13: AMCL algorithm in t=1, example in 1D

In a second interaction, figure xxx, the robot shifted, also shifting the frame of its position. As in this interaction, the robot has in front of a wall, the possible location begins to converge, leaving two places where it is possible for the robot to be located.

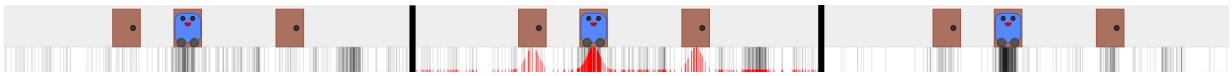


Figure 2.14: AMCL algorithm in t=2, example in 1D

After another interaction, figure xxx, where the robot moves forward of a door, the position that is estimated converges only to a single location.

But, in this type of approach can appear similar or symmetrical places. This sometimes causes the position to be ambiguous. We can verify this effect to occur in the Figure 2.10. In these situations, it is necessary for the robot to move greater distances to reduce this problem. An example of symmetry case can be seen in Figure 2.15.

Figure 2.15: Ambiguity due to symmetry, which can result in two distinct positions with near probabilities [9]

However, other active localization mechanisms can also be used [11], for example:

- WLan [12] [13];
- GPS;
- GSM [14];
- Ultrasound beacons [15].

With using this mechanisms, the system they become extremely dependent on the environment in which they are developed. Since the mere fact of having doors closed or open, we can change the accuracy of the location. This is because certain obstacles can act as insulators or reflectors by being positioned at different times. In this situation, what we want to prove is the possibility of detecting and analyzing places that have been occupied. As such we need some mechanisms that keep us in a precise location like AMCL. However, due to some limitations of this algorithm, as verified, in certain situations the position will have to be valid by hand, for example in case of very symmetrical maps. In future work, one of these methods may be included, according to the best-suited ones.

2.3 Path planning

Path planning consists of algorithms to structure a route for the robot to arrive at a certain location, avoiding collisions with known obstacles. These pre-requisites, as a rule, meant that they had a low cost, that is, the shortest and the fastest way to final goal. To solve this problem there are algorithms, mostly based on Dijkstra's algorithm and A* search algorithm.

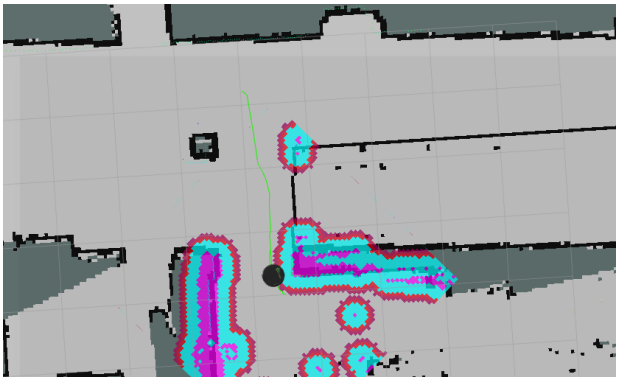


Figure 2.16: Path planning in IEETA using A*, with the black circle the most likely position of the robot and the cloud (red, blue, and purple) is the representation of the cost_path used by A* to inflate obstacles and generate the path, the green line

2.3.1 Dijkstra's

In the case of the Dijkstra's algorithm, it starts to explore from the initial node and continues until it reaches the final node. It then identifies the nodes that connect the end node to the start node. The Dijkstra algorithm explores the nodes by their distance to the initial node. It also explores the nodes that are further from the goal as well as the nodes closest to the goal as it expands outward from the starting node. This process can cause a significant increase in data to be processed, whichever the environment in which it is operating.

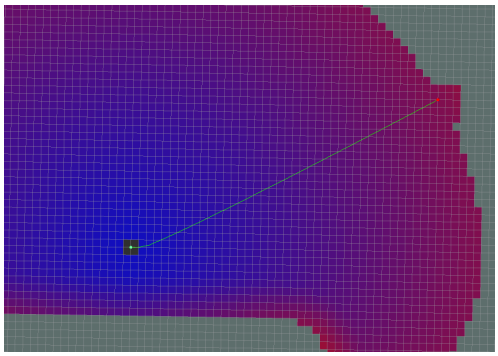


Figure 2.17: Dijkstra's algorithm. All blocks with colour represent the calculations performed by this algorithm of the cost of crossing each block. The black dot is the starting point, and the red dot is the endpoint. The green line is the path calculated based on this map. [16]

2.3.2 A*

The A* does this route search to a destination through a heuristic function that guides the expansion of the node toward the end node. That is, heuristic functions must be used to create a map node correlation for non-negative cost value. In this way, A* improves Dijkstra's Algorithm by focusing only on exploring the nodes approaching the final node. This greatly improves the speed of finding the shortest and most direct path, since this allows a significant reduction in computational effort. [17]

In this implementation, we opted for the use of the algorithm A*, as it can be seen that good planning with a performance far superior to Dijkstra's algorithms.

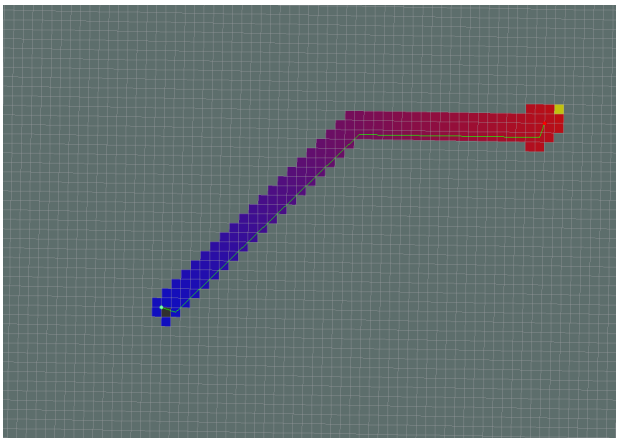


Figure 2.18: A* algorithm. All blocks with colour represent the calculations performed by this algorithm of the cost of crossing each block. The black dot is the starting point, and the red dot is the endpoint. The green line is the path calculated based on this map[16]

Chapter 3

Case study and system architecture

The main case study of this dissertation is to develop software for a robot already built, able to move in a known environment and analyze the space. This system will have as an objective to verify which places are being occupied and the corresponding area. It is also intended that this collected data be easily accessed and adequately represented to system users. The goal is to provide to these users a better view of the occupied or free space so that they can be managed more easily.

3.1 Hardware

There are several types of solutions, based on AMR, to assist people in different types of activities, such as for freight (Figure 3.1), floor cleaning, and some others.

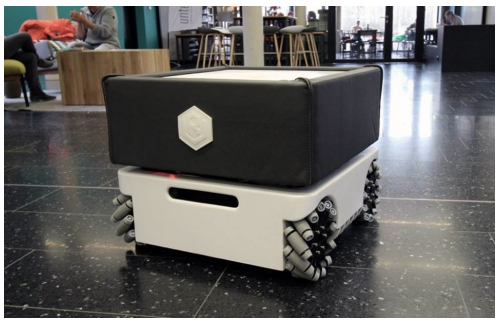


Figure 3.1: AMR, made by robotise

In this work, it will be used Turtlebot (Figure 3.2) with a 2D LiDAR (an Lms1xx, Figure 3.6), as shown in Figure 3.3. This hardware implementation is compatible with

the ROS framework, which already has drivers implemented to communicate with those devices.

Figure 3.2: Turtelbot 2. Commercial version, without changes [18]

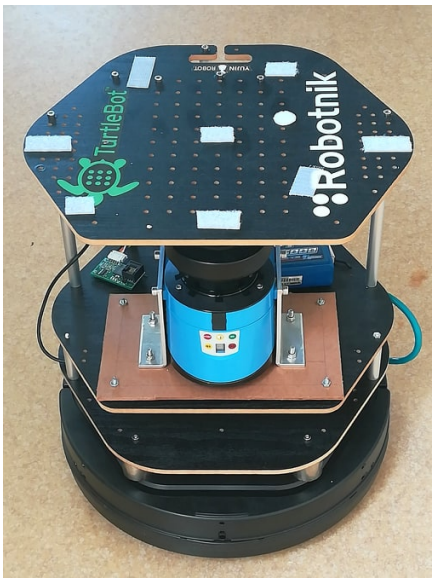


Figure 3.3: Turtlebot, used in this dissertation with the LiDAR at the front

3.1.1 Turtelbot 2

TurtleBot is a low cost hardware robot with open source software. TurtleBot was created in the Willow Garage by Melonee Wise and Tully Foote in November 2010. Being the base a Yujin Kobuki. This hardware is designed to be used in a development environment ROS. It has a very modular structure, and is easy to customize. In this case a 2D Lms100 LiDAR was placed. Figures 3.5 and 3.4 shows the graphical representation of the Urdf file (Unified Robot Description Format).

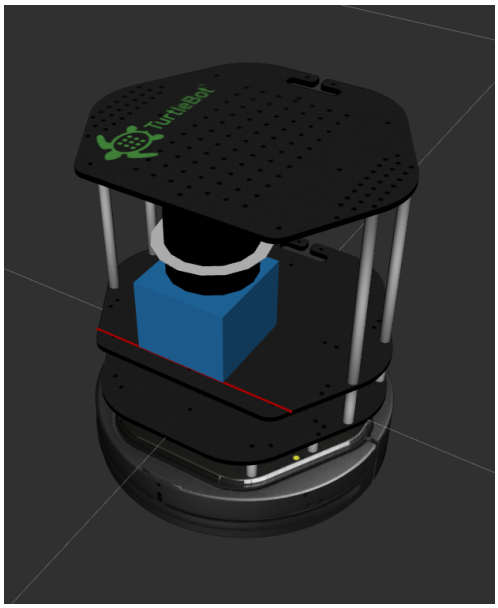


Figure 3.4: Urdf of TurtleBot with Lms100, the blue box with the black cylinder and the white disk representing the origin of the reading zone.

The entire structure of the robot is defined in a file, following a structure of Unified Robot Description Format (Urdf) and using Xacro. In this type of structure it is also possible to associate graphic objects, so using Rviz we can see if the description in Urdf corresponds to reality. In the Figure 3.5 the Turtlebot is represented, has already been described. However, the LiDAR was added in a custom position. In Figure 3.4, we can also see the axis corresponding to each link thus allowing to see if the description corresponds to the reality of the robot.

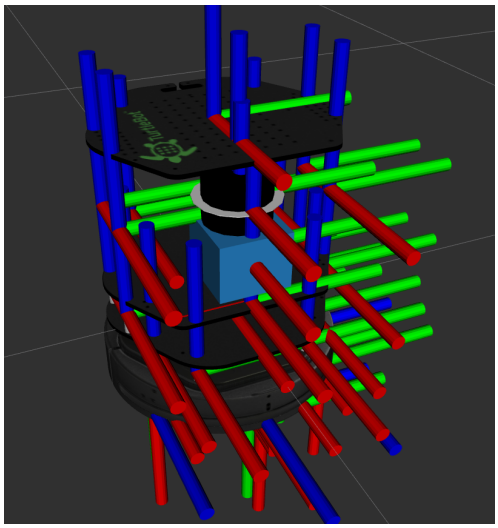


Figure 3.5: Urdf of TurtleBot with Lms100 showing the axis of each join

The final position of the LiDAR in this description was defined in the first phase, measuring the distance in the robot (physical). These measurements were then adjusted, placing the robot in motion, and analyzing the points collected by LiDAR and represented by Rviz in a global frame, until they remained as static as possible.

3.1.2 LiDAR Lms1xx

The most important sensor used in this application is a 2D LiDAR, capable of measuring distances between 0.5m and 20m, with an angular aperture of 270°, as can be seen in Figure 3.7 [19].

Figure 3.6: Sick Lms100 [19]

For this implementation, it was configured to operate at a resolution of 0.25° and perform scannings at a frequency of 50Hz. The data is being sent via ethernet, to the computer that controls the turtlebot. This computer is using ROS and in this framework, there are already drivers developed for this sensor, which allows we to convert the data received via ethernet to the ROS (sensor_msgs/LaserScan) message standard. In this way, it is possible to use several tools or processes that ROS has available to analyze or visualize this data.

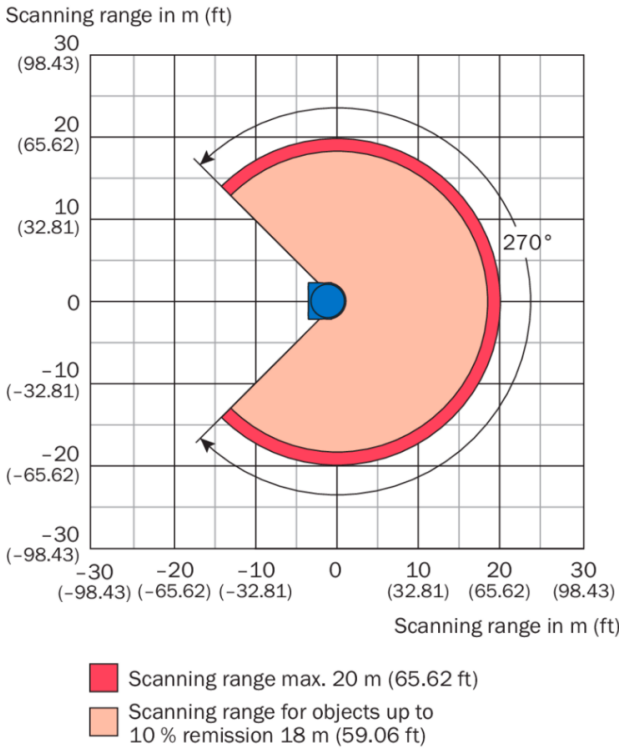


Figure 3.7: Working range diagram [19]

3.2 Software

With the increasing complexity of the systems being developed, it becomes crucial to use other tools already developed and tested. This type of approach makes the system more robust and faster to be developed. With this objective was used:

- ROS to assist in the development;
- Rviz allows simple and fast way to debug the system;

– PCL and Octomap are libraries that help us manage the data from the LiDAR;

– Flask simplifies the development of a web server.

3.2.1 ROS

Robot Operating System (ROS) is an extremely versatile framework for software development for robots. This contains a set of tools, libraries and conventions that make development easier for a wide variety of robots. As it is a collaborative platform, there is a great variety of software developed for different types of applications. This framework works similarly to an operating system, with the abstraction of hardware, allowing management and communication between processes. This adds the possibility of being implemented as a distributed system [20].

3.2.2 Rviz

Rviz is a ROS environment tool. This allows the visualization of the messages generated by the different processes, in a graphical 2D / 3D environment. In this way, we have a graphical visualization, namely, the map, the robot, the camera, new occupied area and respective area, etc. This graphical interface can also be used to generate messages, such as indicating the current position of the robot or a position to which it is intended to move.

3.2.3 Octomap

Octomap implements a 3D grid-based probabilistic mapping structure based on Octree. This allows a full representation of 3D models, which can be updated easily at any time, being very flexible with the map to expand dynamically. This creates a very compact structure. Since grid upgrade is done in a probabilistic way, errors in measurements due to noise are greatly attenuated. This implementation also can have several robots contribute to the construction of the mapping grid.

At the level of visualization or access, it is very efficient thanks to its octree structure, that is, each node can expand to eight children. This type of structure is represented in the figure 3.8, and the way we can visualize the grid at different depths is represented in the Figure 3.9 [21].

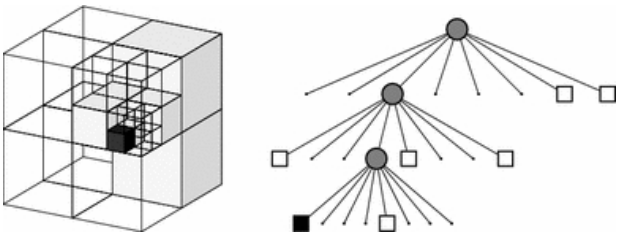


Figure 3.8: Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corre- sponding tree representation on the right.[21]

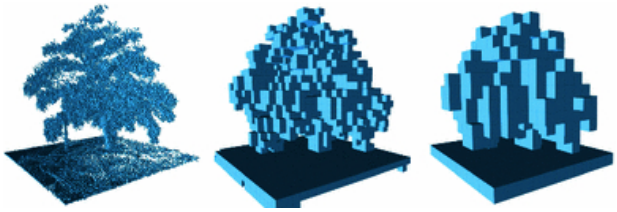


Figure 3.9: By limiting the depth of a query, multiple resolutions of the same map can be obtained at any time. Occupied voxels are displayed in resolutions 0.08 m, 0.64 , and 1.28 m. [21]

In this case, the representation is only 2D. Because Octomap is very flexible with the dimensions of the map, the use of the only 2D is equally well optimized and compact. Since the points sent to Octomap are always in the same plane, it will not have to expand in the third dimension.

3.2.4 PointCloud

For the processing of the cloud of points is used PCL, which is a library developed in C++ and dedicated to this type of processing. It is developed to be efficient and has good performance in modern CPUs. At the algorithm level, PCL, it also includes several implementations for the process this data, such as filters, segmentation, etc. [22]

3.2.5 Flask

Flask is a microframework for development in Python, and is based on Werkzeug and Jinja2 [23]. This is dedicated to developing web applications.

Jinja2 is a language for developing Python templates. With this, it is facilitated the process of having a page described in HTML with places where the information to present is a dynamic information. This information is dependent on the execution of the server itself (developed using Werkzeug) using the templates.[24]

Werkzeug is an extensive WSGI web application library. It is up to the developer to choose a template engine, database adapter, and even how to handle requests [25].

Chapter 4

Proposed solution

The robot developed, should navigate in a certain environment, to verify which places were occupied and to carry out the measurement of that area. To make this possible, it was necessary to tackle some problems such as mapping, location and navigation. Thus it is necessary to create a map to be used by the robot to locate, navigate autonomously and detect occupied areas. So this mapping done initially is very important. For this task, as already mentioned will be used GMapping, to generate a map with a resolution of 0.05 meters.

As we can see in Figure 4.1, while the map is being generated, the robot is controlled by a joystick. However, the output value of the joystick is not applied directly to the motors of the robot but passes through a process `teleop_velocity_smoother`, so that this value is filtered in order to the transition between movements became more smoother. With this, the wheels will be less likely to slip, with encoders values closer to reality, minimizing the probability of errors on the map being generated.

At the end is the GMapping, which uses the transformation trees of the ROS (TF) and the measurements of the LiDAR to generate the map. Also, we can notice that between GMapping and TF the communication is bidirectional, this because it deals with a SLAM algorithm that uses the position calculated by the robot using the encoders, and estimates the position comparing with the signal of the LiDAR, sending these estimates to the tree of transformations.

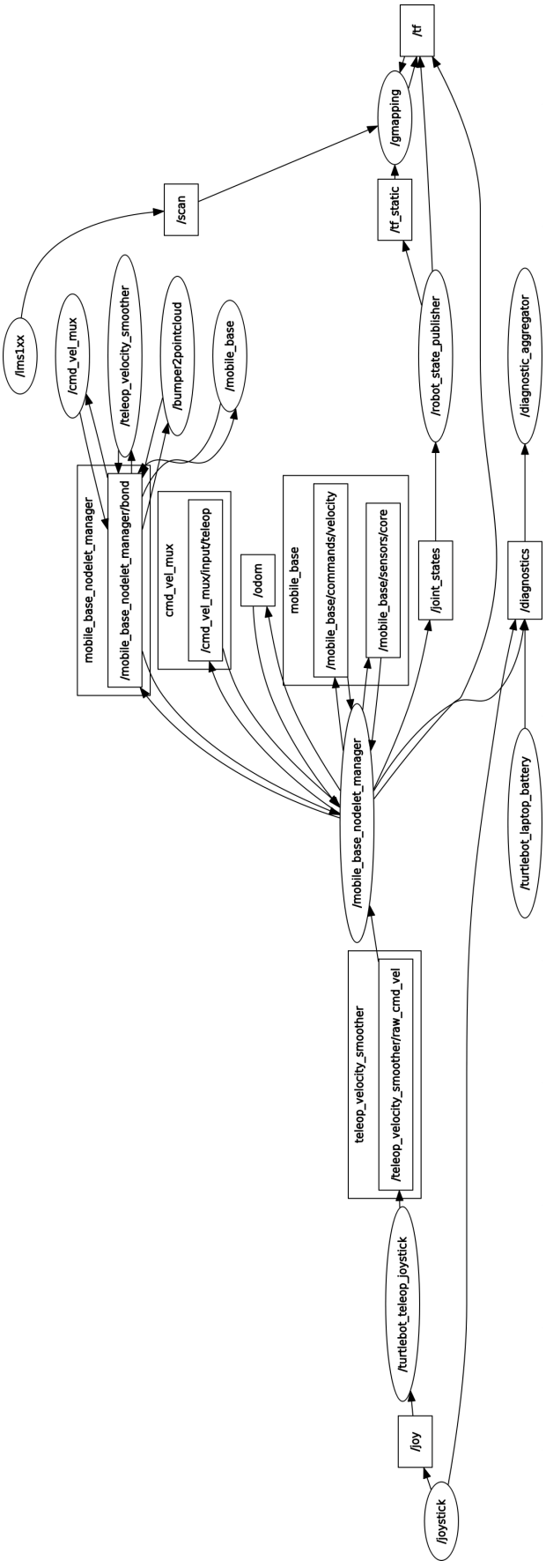


Figure 4.1: ROS graph of Gmapping

Having the map generated, it is possible to start the robot in an unknown position within the map. After moving a certain distance, and with the application of AMCL, it is found the most likely position where the robot is on the map. Depending on whether we continue to move the robot, AMCL continues to estimate the position, using encoder and LiDAR data.

As we can see in Figure 4.2, in a more detailed way, we no longer use GMapping because we already have a map built and we now essentially add two processes (AMCL and `move_base`).

The AMCL, based on the map that is initially loaded and using the transformation trees and the signal of the LiDAR, as already explained (section 2.2.1) and calculates the most probable position of the robot is on the map. After this calculation, it predicts the offset of error that the odometry is having and changes the origin of the reference of the same one so that it is corrected, being this change published in the tree of transformations.

The `move_base` receives the data from the transform trees, the odometry, the map, and the LiDAR signal. All these data are important for the cost map to be calculated. This process is also waiting for a goal message to be published (a point for which the robot is intended to move). When this goal is published a route it is plotted using the generated cost map and using A* algorithms. This creates a plan for the robot to transverse and begins to move it. The `navigation_velocity_smoother` works in a similar way to `teleop_velocity_smoother`, meaning that the velocity variations are smooth.

In both processes, the LiDAR signal undergoes another process, the `laser_odom_rate _normalize`. This is because the laser signal and the odometry signal (of the encoders) were out of phase and when the robot moved the position of the LiDAR associated with the temporal level to the respective scan is incorrect. This was a high source of error, in rotational movements, causing an error in the estimation of the AMCL, and later in the calculation of the areas of occupied places, because it expands all the occupied areas. However, since this error is a constant delay, it was possible to correct it by developing this process that publishes the scan with the respective delay to coincide with the odometry. After some tests and conducting a binary search, it was possible to conclude that this delay was 72 ms.

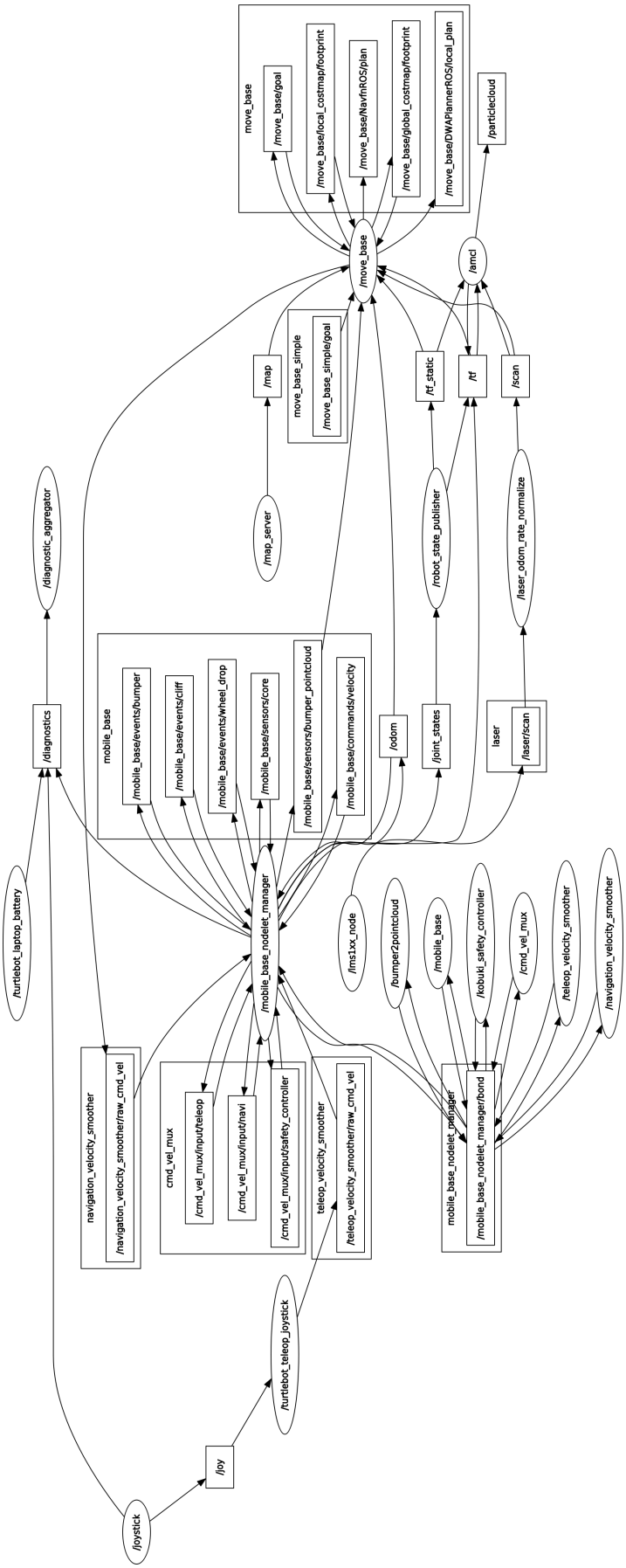


Figure 4.2: ROS graph of AMCL

This is extremely useful because it prevents the accumulating of errors. Since with errors in the location, the data collected by the LiDAR to verify the occupied areas would not be viable since they would not be aligned with the map to be validated with the following implementation.

In this implementation, a process similar to that described in [26] is used, except that one represents 2D. This consists of having a map already created (the map that is used for autonomous navigation), and performing a new mapping, discarding the areas that were not mapped or are occupied in the existing map.

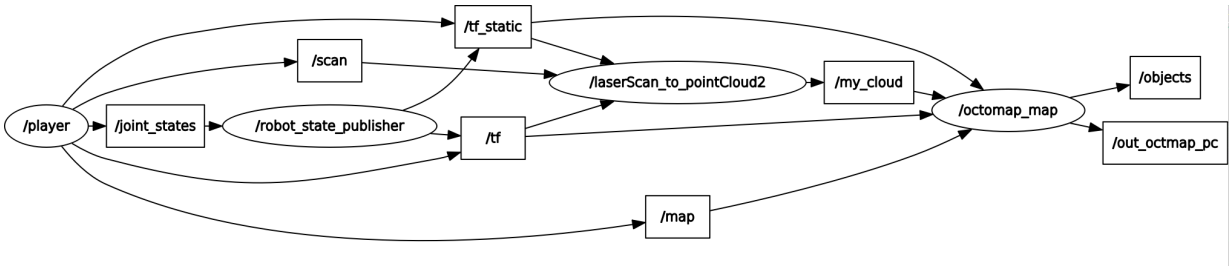


Figure 4.3: ROS node graph of object detection

In Figure 4.3, the process called `player` consists of the entire structure represented in Figure 4.2, there is also a new process called `octomap_map`. It uses data from the LiDAR, map and tree of transformations to generate an occupation grid as the new occupied places and publishes the `out_octomap_pc`, in a point cloud structure, where each point has a resolution of 5 cm and also generates markers for each object that is detected, the `objects`. The internal operation is explained in more detail in the following sub-section.

4.1 Find occupied areas

To find new occupied areas on the map, it is necessary to perform a new mapping using the odometer and the LiDAR signal. This mapping is done by comparing it with the initial map to verify where it was occupied. In Figure 4.4 we can see several white dots that refer to the data collected from LiDAR. However, in this situation, all the data that is presented must be discarded.

4.1.1 Filter points

In Figure 4.4, we have points that coincide with the occupied zones already displayed on the map, which are not zones of interest. For this reason, we apply morphological operations on the map to expand all occupied areas twice the resolution that it presents, this to prevent that due to some error in the location the already mapped walls are again considered. All the points contained in these areas, as well as the points contained in unexplored areas, are discarded because it is a person moving around.

However, as already mentioned, all points in this Figure 4.4 must be discarded. This is because the points marked, immediately in front of the robot, are due to the movement of a person. It is something that is in motion and this does not show us interest because it is not always in the same position. To solve these types of problems, we used a solution based on probabilities, the Octomap. With this solution, if the point is occupied the Octomap increases the probability of the respective cell. If it is free by the Lidar measurement, the probability of the cell decreases. Only when certain thresholds are reached, the cell change to the occupied state. Octomap is quite versatile in this management, so several parameters can be changed to define the variations of probabilities and set the thresholds for the cell to be marked free or occupied. In this way only if the object remains in the same place more than a certain time is that it is considered as a possible object of interest to be considered as an occupied area.

Figure 4.4: The black dots and the light grey areas represent the map already created. The white dots is a stack of some readings of the LiDAR and the black circle the most probable position of the robot.

4.1.2 Object detection

In this implementation, the data in an octomap are periodically converted to PCL point cloud. Using PCL we have some methods available to assist in data processing as well as filters. First we filter based on the number of neighbours of each point. The purpose of this filtering is to eliminate isolated points, mostly noise related to refraction effects in places such as corners. After this filtering, an Euclidean Cluster Extraction is done to group all points close enough to delimit an occupied area. In this way, it is possible to obtain individually each occupied place so that it can be analyzed. As can be seen in Figure 4.5, each set of green marked blocks that are grouped are separated for said analysis.

Figure 4.5: Occupied areas. The black dots and the light grey areas represent the map already created. The green blocks delimit the occupied area and the calculated area is displayed.

In the result shown in Figure 4.5, the space marked as occupied refers to a circular object of radius 12.75 cm, that is, 0.051 m^2. Comparing both values, the real and the estimated value are shown in Figure 4.5, we have an error of 7.8%.

However, if we look at the fact that the resolution we are using is 5 cm, it is noticeable that in the limit (without considering any errors that may exist in the position of the robot) the area expands 5 cm in all the periphery of the occupied place and this case, would consider this object with 17.5cm radius. In this way, the calculation of the area could reach 0.099 m^2. This error factor is being attenuated using the method explained below. Which consists of each grid cell having different weights as a function of their neighbourhood free or occupied.

4.1.3 Area estimation from a grid

Regarding the calculation of the areas, a method was developed to minimize the error caused by the grid resolution. In Figure 4.6 we can see the evolution carried out in equations A) and B).

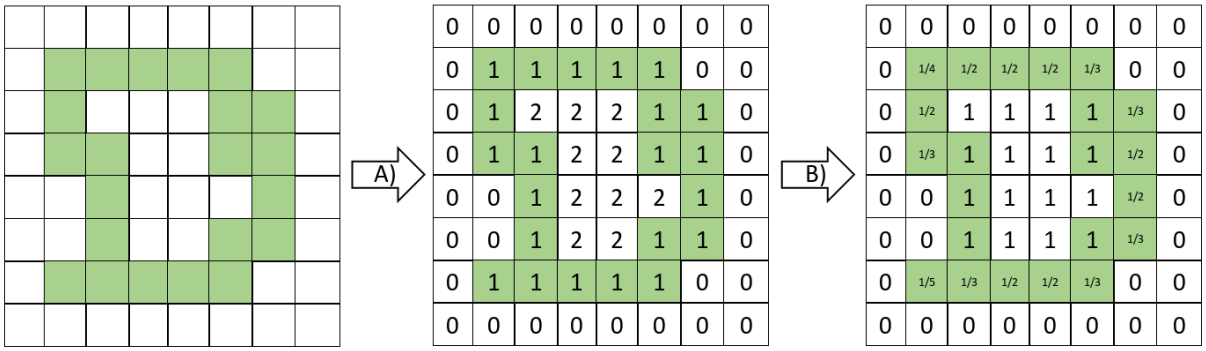


Figure 4.6: Calculation of areas. The green blocks delimit the occupied area. The methods A) and B) are the steps performed by the system in order to calculate the area.

In A), a grid representing the object is generated. All points marked as occupied (green) are set to 1, the free points with 0 and the internal area with 2.

V>0	V>0	V>0
V>0	**0**	V>0
V>0	V>0	V>0

Figure 4.7: Convolution matrix. The boolean condition, true is 1 and false is 0.

In B), the convolution is initially done with the matrix presented in 4.7. In this operation, V is the value of each point in the grid and as a result of the boolean condition, true is 1 and false is 0. Being the resulting convolution value represented by, CONV, and the final value assigned to the point by, X. Then

$$X = \frac{1}{max(7 - CONV, 1)} \tag{4.1}$$

Finally, the area is calculated by the sum of all the cells in the grid after B) multiplied by the square of the resolution.

4.1.4 Web page

For a user of the system to access the data gathered by the robot remotely, we implemented a Web server. This implementation is done using flask, a web development microframework. Thus a user can access this site and can control the robot.

In the exposed web page we can see several information such as a transmission of the robot front camera's images and a map of the environment where it is navigating with markers indicating the location of the robot as well as the locations that have been detected as occupied. When we click on any of the areas we can consult information related to this area, such as the area estimated value. In Figure 4.8, we can see an image of this interface as in the map we also see a red marker, referring to the position of the robot, this marker is triangular to be able to indicate the direction of the robot on the map.

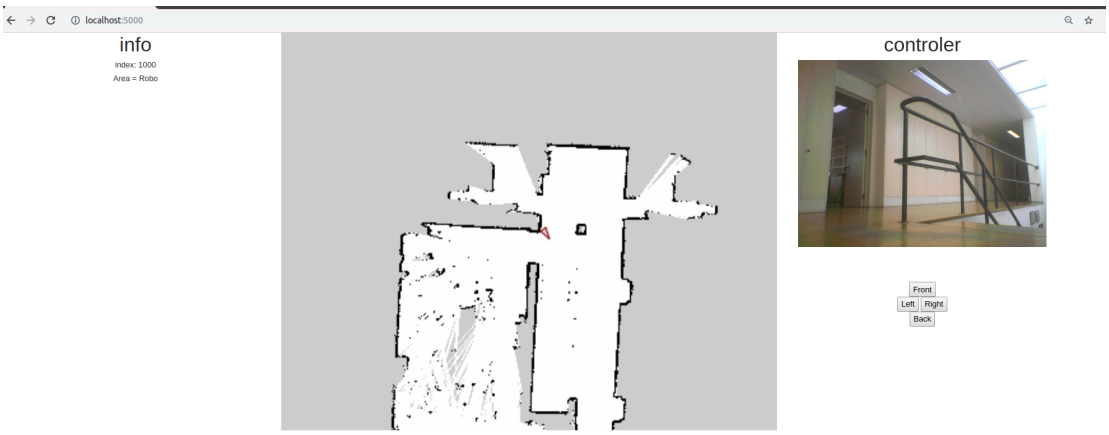


Figure 4.8: With the map in the center and with a red marker indicating the position of the robot. On the right are the real-time preview of the robot front zone and the controls. On the left, the information about the occupied area referring to the selected area marker on the map.

4.1.5 Final tests

To test what was developed, some tests were done and compared with the actual dimensions of the objects placed in the test scene. In the first test, a box was placed in an open area, as we can see in Figure 4.9, which occupies an area of 0.314m x 0.222m. However, the probabilistic grid constructed in octree and managed using octomap has a resolution of 5cm. This implies that in this tests, as well in the following ones, the object being measured may have an error that expands 5cm in all directions, i.e. in the limit, the occupied area is considered to be (0.314m+2x0.05m)x(0.222m+2x0.05m). However, the method described above is to be used to minimize this estimation error. Other inaccuracies related to the location may occur, which may cause a greater effect at a percentage level on smaller areas of measurement.

Figure 4.9: Object in the occupied area 1 (image from a camera onboard the robot)

In this first test, we then have an occupied area of,

$$0.314 \times 0.222 = 0.070m^2 \tag{4.2}$$

That limit due to the resolution of 5cm would be,

$$(0.314 + 2 \times 0.05) \times (0.222 + 2 \times 0.05) = 0.133m^2 \tag{4.3}$$

We would expect an area within the range $[0.070m^2; 0.133m^2[$. As we can see in the Figure 4.10, the area calculated by the system is $0.158m^2$. This difference is related to errors in estimating the position of the robot along with the movement and the 12mm error of the LiDAR (however this value may depend on the ambient conditions) [19]. It can be seen clearly two demarcated zones in Figure 4.10, one at the top that is too small and is discarded, this it is due to a door being slightly open and at the bottom is seen another occupied area referring to a door that was closed.

Figure 4.10: Occupied area 1. The green blocks delimit the occupied area and the calculated area is displayed.

In the Figure 4.11, we can see how this data are presented to the user on the web page.

Figure 4.11: Web page of the map of the first test. The blue squares represent the occupied areas.

In a second test, a circular object with a radius of 0.128m was used, as we can see in the Figure 4.12, whereby it occupies an area of,

$$\pi \times 0.128^2 = 0.051m^2 \tag{4.4}$$

That limit due to the resolution of 5cm would be,

$$\pi \times (0.128 + 0.05)^2 = 0.099m^2 \tag{4.5}$$

Figure 4.12: Object in the occupied area 2 (image from a camera onboard the robot)

So we would expect an area within the range $[0.051m^2; 0.099m^2[$. As we can see in the Figure 4.13, the area calculated by the system is $0.053m^2$. In this case, we can see that, unlike the previous situation, the door is open as the map was generated and this can influence the accuracy of the location, because the more differences that emerge on the map, the lower the probabilities to have a more accurate location.

Figure 4.13: Occupied area 2. The green blocks delimit the occupied area and the calculated area is displayed.

In the Figure 4.14, we can see how this data are presented to the user on the web page.

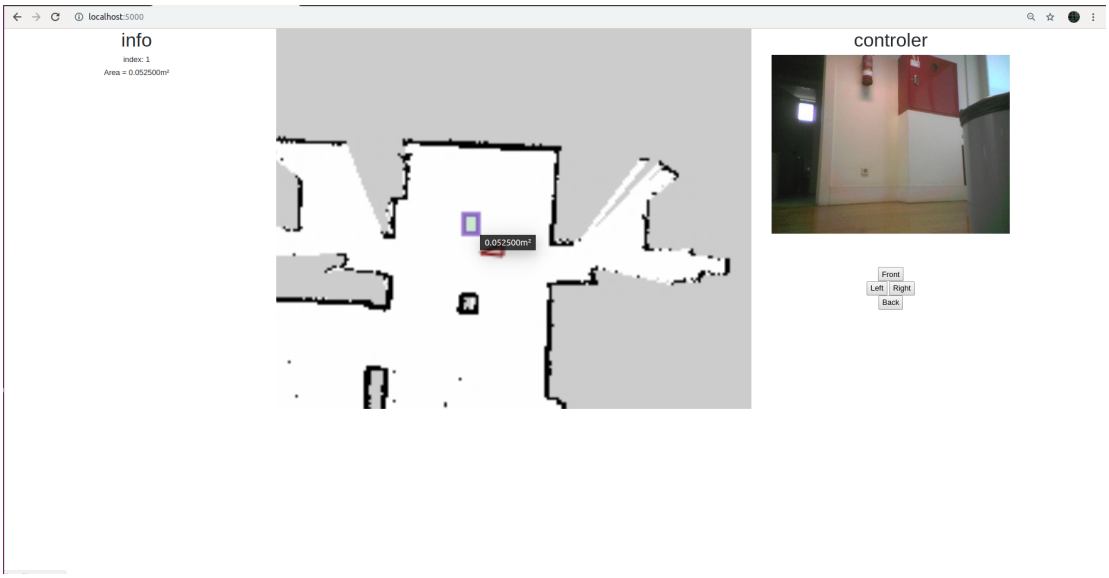


Figure 4.14: Web page of the map of the second test. The blue squares represent the occupied areas.

In a third test, the two previous objects were used, as we can see in Figure 4.15, so it occupies an area of,

$$\pi \times 0.128^2 + 0.314 \times 0.222 = 0.121m^2 \tag{4.6}$$

Figure 4.15: Object in the occupied area 3 (image from a camera onboard the robot)

That limit due to the resolution of 5cm would be,

$$\pi \times 0.178^2 - \left[\int_0^{0.1} 2 \times \sqrt{0.1775^2 - (0.1775 - x)^2} dx \right] + 0.414 \times 0.322 = 0.209m^2 \tag{4.7}$$

The integral part in this last calculation corresponds to the area of overlap between both objects when they are expanding 5cm all around.

For what it would be to look for an area comprised in the interval $[0.121m^2; 0.206m^2[$. As we can see in the figure 4.16, the area calculated by the system is $0.183m^2$. In this case, we can see that the conditions are the same as in the first test, but the occupied area is larger.

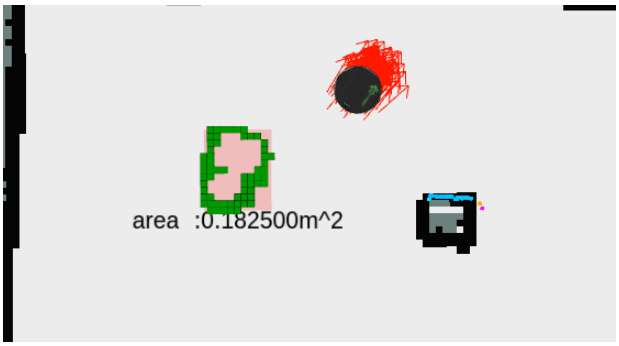


Figure 4.16: Occupied area 3. The green blocks delimit the occupied area and the calculated area is displayed.

In the figure 4.17, we can see how this data are presented to the user on the web page.

Figure 4.17: Web page of the map of the third test. The blue squares represent the occupied areas.

4.2 Localization Results

To understand the error contribution that can arise from the localization methods, a more detailed approach was made. Starting with an analysis of the encoders and the locomotion system in which they are integrated and that can cause some drift. Followed by an analysis of the system with the localization mechanisms, in this case, AMCL.

4.2.1 Encoders analyze

The possibility of using a methodology based only on encoders for the location of the robot generates some problems. Even if they are precise, there is always a possibility that along with the various movements some drift of the wheels with the ground may appear. And as such, this is an error that tends to accumulate. In this case, since we are using AMCL, to help position the robot on the map, errors that can be accumulated by encoders over longer distances are compensated by this localization algorithm. However, for small movements, the encoders are fundamental to have a greater accuracy. In this way, the following tests were performed to verify the amount of error that arises from these sensors:

- Ten rotations in the same direction

  the robot has moved 14º from the initial position

- Ten rotations alternating the direction with each rotation

  the robot finished displaced 7º of the initial position

- Displacement of five meters and return the starting position always in a straight line and without doing any rotation

  the robot finished displaced 0.038 m from the initial position

Odometry: 52 ticks/enc rev, 2578.33 ticks/wheel rev, 11.7 ticks/mm [27].

4.2.2 AMCL vs encoders alone vs reality

The odometry (calculated using relative encoders) already present an error that accumulates along with the movement of the robot, however, if this error can be feasible, the use of the encoders alone would have other problems, as the case presented in the Figure 4.18. For this to be possible, the initial position of the robot would have to be estimated with

extremely high precision, since the rotation with which the robot was initially extremely important. For example, an error of 1º in the initial position of the robot at the end of 5m causes an error in the position of 8.7cm, after 15m this error increases to 26.2cm and an error of 2º causes respectively 17.5cm and 52.4cm of error. As the robot has 23cm of the distance between the wheels, it means that a rotation of 1º the wheels must rotate in opposite directions 2mm each and to 2º, 4mm each. That is, they are relatively small dimensions to position manually, originating what is seen in figure 4.18.

Figure 4.18: Location by odometry (red) compared to AMCL (green) for small distances, with initial position error

By introducing the AMCL into the system, it is possible to have a considerable improvement. Thus the encoders continue to be used to calculate positioned the robot in a given coordinate system (odom), and the origin of this system is changed by the AMCL so that the position of the robot is in the correct position on the map reference. In this way, in the Figure 4.19, we see that the green signal is away from the position of the robot (red trajectory), this is because the signal corresponds to the one that AMCL is compensating so that in the coordinate system odom the robot is positioned correctly. We can also see the deviation of this signal (green), as the error that the position would have without using a AMCL.

In the images presented in Figure 4.20b, we can see what position of the robot presented in the figure 4.19, is very close to reality comparing the position in which the obstacles seen and the position on the map.

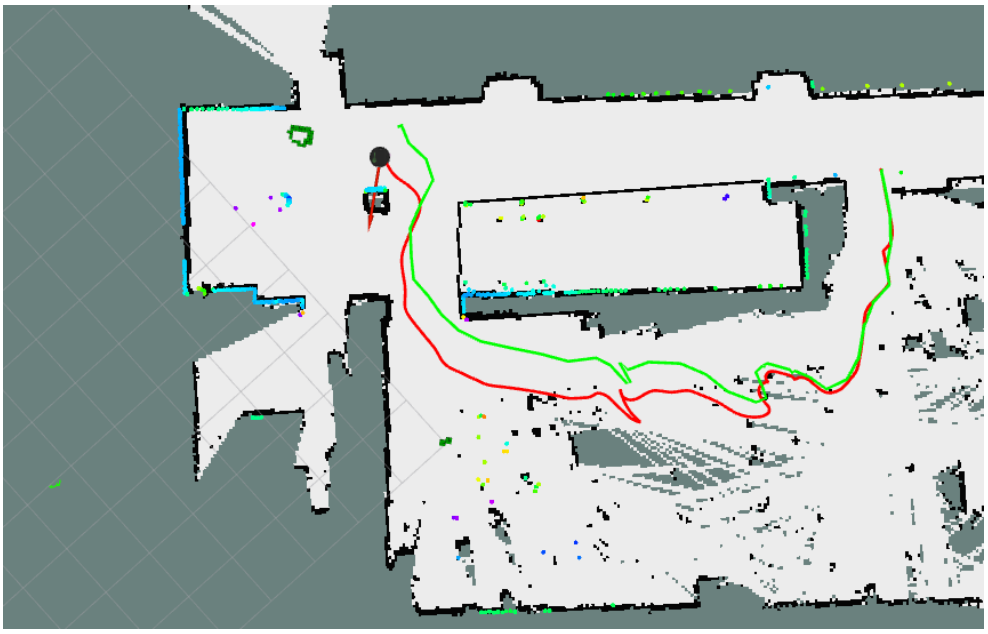


Figure 4.19: Location compensated by odometry with AMCL (red) and compensation made by AMCL (green)

In the Figure 4.19, we can also see the signal of the LiDAR, which coincides with the walls shown on the map. As this LiDAR presents a statistical error of 12mm, although this value may depend on environmental conditions [19], we can conclude that the robot is correctly positioned on the map.

(a) Initial position (robot viewer)

(b) Final position (robot viewer)

Figure 4.20: View of the front of the robot in the initial and final position in the path represented in 4.19

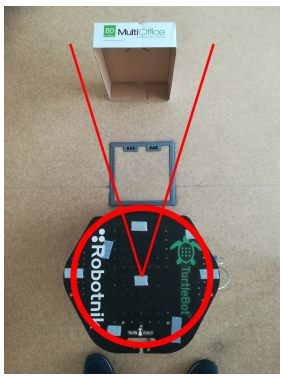
4.2.3 Path planning test

To test the possibility of the robot being able to move autonomously on the map, using the `move_base` node, was initially placed at a certain point as shown in Figure 4.21. This position was recorded to be the goal in the following tests. In Table 4.1, there are also some parameters that define the tolerances for goal achieving.

xy_goal_tolerance	0.15m
yaw_goal_tolerance	0.3rad (17.2°)
robot_radius	0.20m
inflation_radius	0.3m
cost_scaling_factor	5.0

Table 4.1: Some parameters for the move_base node

(a) Manually set of the end position (robot viewer)

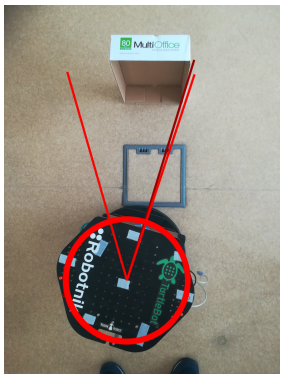


(b) Manually set of the end position (top viewer)

Figure 4.21: Set final position. The red markers represent the defined tolerance.

In the following tests, the robot was moved to different locations on the map. By moving autonomously, we can reach the goal. However, the end position is not exactly the position that was indicated, but a very close position. This difference of positions is related to the parameters of the Table 4.1, and as can be seen in Figures 4.22, 4.23, 4.24, 4.25, 4.26), the final position is reached within parameters. We could see that the angle never exceeded a maximum deviation of 17.2° (defined in the parameters) and that the position always approaches the final position within the maximum distance indicated of 0.15m.

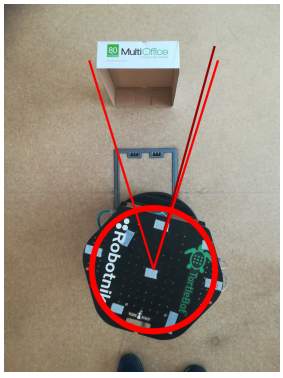
(a) Final position (robot viewer)



(b) Final position (top viewer)

Figure 4.22: 1º test. The light red markers represent the defined tolerance and the dark red the final rotation of the robot.

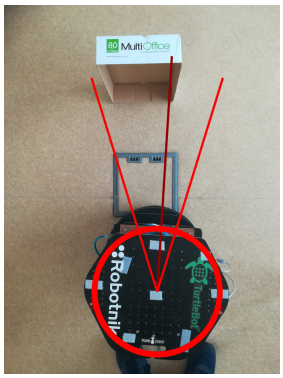
(a) Final position (robot viewer)



(b) Final position (top viewer)

Figure 4.23: 2º test. The light red markers represent the defined tolerance and the dark red the final rotation of the robot.

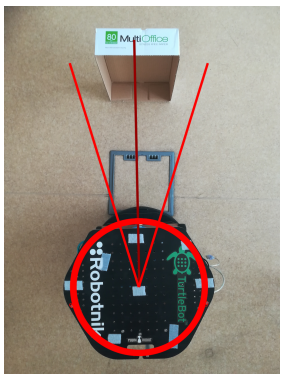
(a) Final position (robot viewer)



(b) Final position (top viewer)

Figure 4.24: 3º test. The light red markers represent the defined tolerance and the dark red the final rotation of the robot.

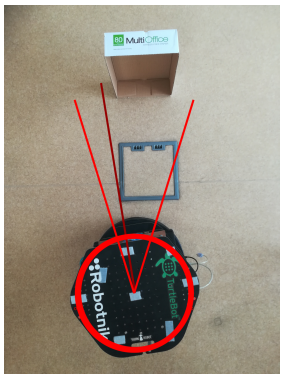
(a) Final position (robot viewer)



(b) Final position (top viewer)

Figure 4.25: 4º test. The light red markers represent the defined tolerance and the dark red the final rotation of the robot.

(a) Final position (robot viewer)



(b) Final position (top viewer)

Figure 4.26: 5º test. The light red markers represent the defined tolerance and the dark red the final rotation of the robot.

With the parameters that were used we can verify that the final position is very close to the desired position. However, these parameters could be adjusted for greater accuracy when reaching the goal, but this would imply that the robot made more moves taking more time. In the case of a list of points by which the robot is intended to move, it is preferable to use more versatile parameters such as these, which allows it to be faster and the transition to the next path more smoother.

In Table 4.1, the values that define the cost map (robot_radius, inflation_radius and cost_scaling_factor) are also indicated. The radius of the robot is 0.18m, however, a radius of 0.20m is used to create a small area of security around the robot. Obstacles are also expanded by the indicated values, generating a cost map such as can be seen in Figure 4.27.

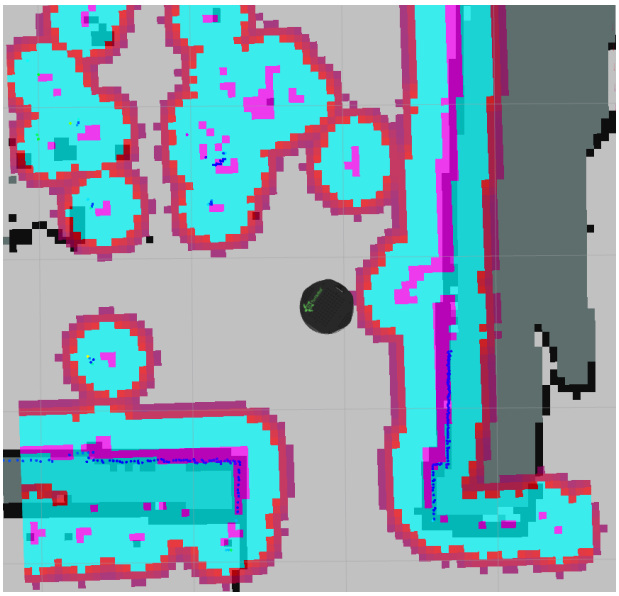


Figure 4.27: Cost map example. The cloud (red, blue, and purple) is the representation of the cost_path used by A* to inflate obstacles to generate the path.

The mode of operation of this node can be seen in Figure 4.28. Here we can see the data needed for its operation, as well as the interactions that are made until reaching a final solution. Thus it can be seen that the data collected from LiDAR will be used to create the local cost map as what we can see in Figure 4.27.

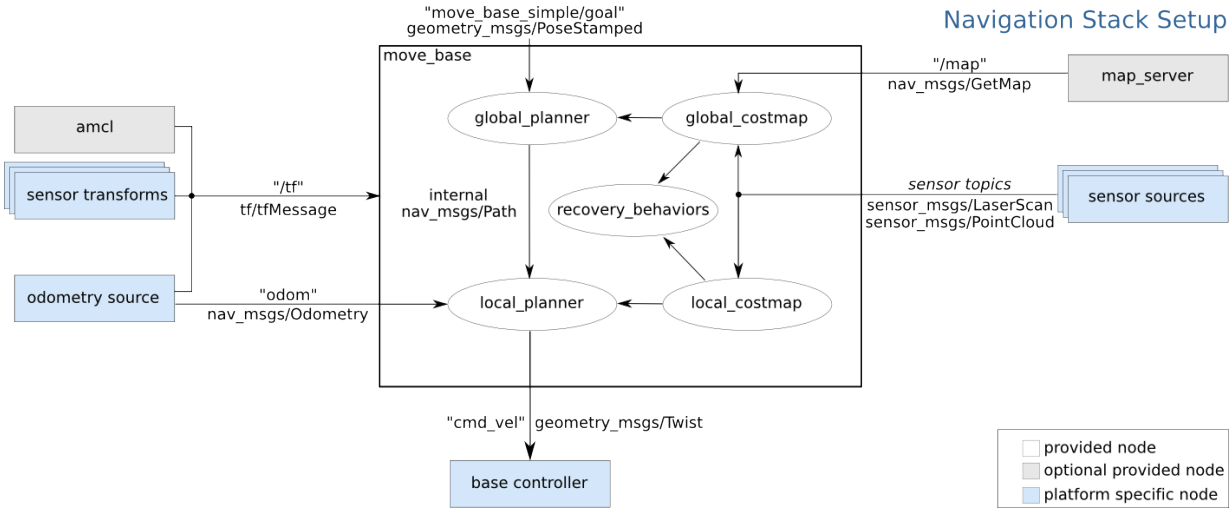


Figure 4.28: move_base node setup. A graphical representation of the steps and data used by this node to control the movement of the robot by the trajectory that is calculated. [28]

Having the cost maps, it is calculated the path to reach the goal. Sometimes new obstacles can arise that can not be passed and in these situations, the `recovery_behaviors` is executed, which follows the sequence of steps presented in the Figure 4.29, as the objective to reanalyze the space around it a find a new path without causing any collision. But as we can see from the diagram 4.29, after performing the sequence of all methods and none have found an alternative that avoids collisions this method is aborted, and the robot remains still and waiting for new instructions.

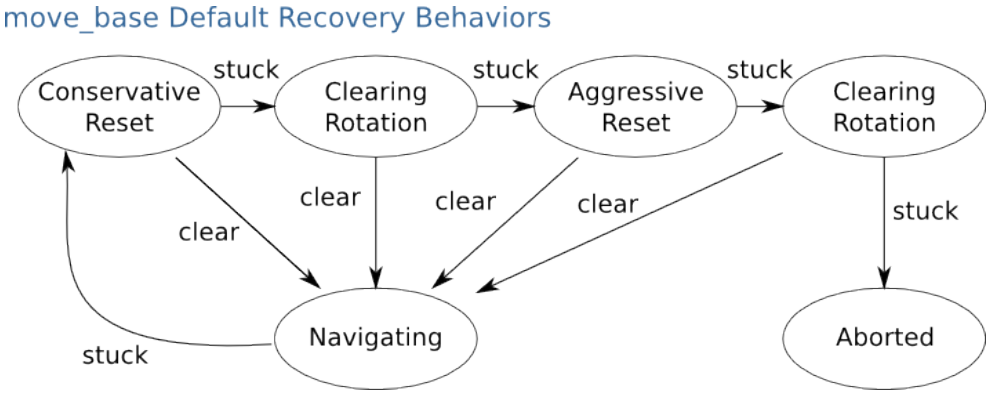


Figure 4.29: Robot recovery behavior [28]

Chapter 5

Conclusions

In this work, a software was developed that can be applied to a robot. Here it has been tested on a turtelbot with a LiDAR, allowing this robot to map the spaces it moves on so that it can later locate and navigate autonomously. Also was developed a software capable of mapping new obstacles at the same time the robot navigates the previously mapped space. This collected data is presented to the system user through a web page that has been developed.

With the this developed and tested, it is concluded that it is possible to analyze the occupied areas in indoor warehouses, using a mobile robot. In this case, using a robot capable of navigates manually or autonomously by the already mapped spaces, it is possible to detect the areas that are being occupied and can calculate the space occupied by each load on the map. This developed tool can become an extremely useful tool for analyzing and optimizing the occupation of spaces, to take advantage of the available space. When compared to traditional methods such as measuring the square surrounding the occupied area using the measuring wheel, the developed method presents a lower error.These methods also become much slower to present data and may not always be obtained in a similar way, making analysis more complicated. With this system, the data is always represented similarly and coherently, and the robot can constantly update the map, making more simple to read and accessible to all users.

Future work could be the implementation of an algorithm that generates points by which the robot should move to autonomously, reanalyze the entire map ensuring that all occupied areas are seen from different points around it so that it is possible to generate an outline of the entire occupied area. This ensures that the remaining system can have the data to calculate and correctly display the occupied space.

However, in an industrial environment, the existence of more mobile units like this robot, to move actively in the work areas can cause certain impediments in the flow of the works. This system can be implemented in an existing mobile unit, as in other robots that already perform some other task or even in forklift trucks that move loads. This may be a future implementation, since the need to integrate some hardware into a specific system, and the development of some software specific to the hardware used.

Another future work can be the integration of an active location system, to avoid positioning errors, which can be essential in places where there are several symmetries. As discussed in section 2.2, for indoor use.

Another possibility would be to use the system in an outdoor environment. For this, it would be necessary to implement tough hardware for this type of environment, thus developing a robot or a system to integrate into an existing mobile unit. However, at the location level, there may be more efficient methods such as possibly using GPS.

Bibliography

[1] A. Huletski, D. Kartashov, and K. Krinkin. Vinyslam: An indoor slam method for low-cost platforms based on the transferable belief model. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6770–6776, Sep 2017.

[2] J. M. Santos, D. Portugal, and R. P. Rocha. An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6, Oct 2013.

[3] E. Pedrosa, A. Pereira, and N. Lau. A scan matching approach to slam with a dynamic likelihood field. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 35–40, May 2016.

[4] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques forGrid Mapping With Rao-Blackwellized Particle Filters. *InTrans. onRobotics*, 23(1), February 2007.

[5] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian. A quantitative study of tuning ros gmapping parameters and their effect on performing indoor 2d slam. In *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 1–6, Sep. 2016.

[6] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3-4):195–207, 1998.

[7] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999.

[8] J. . Gutmann and D. Fox. An experimental comparison of localization methods continued. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 454–459 vol.1, Sep. 2002.

[9] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99 – 141, 2001.

[10] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003.

[11] Hiam M. Khoury and Vineet R. Kamat. Evaluation of position tracking technologies for user localization in indoor construction environments. *Automation in Construction*, 18(4):444 – 457, 2009.

[12] C. Lim, Y. Wan, B. Ng, and C. S. See. A real-time indoor wifi localization system utilizing smart antennas. *IEEE Transactions on Consumer Electronics*, 53(2):618–622, May 2007.

[13] Widyawan, M. Klepal, and D. Pesch. Influence of predicted and measured fingerprint on the accuracy of rssi-based indoor location systems. In *2007 4th Workshop on Positioning, Navigation and Communication*, pages 145–151, March 2007.

[14] Veljo Otsason, Alex Varshavsky, Anthony LaMarca, and Eyal de Lara. Accurate gsm indoor localization. In Michael Beigl, Stephen Intille, Jun Rekimoto, and Hideyuki Tokuda, editors, *UbiComp 2005: Ubiquitous Computing*, pages 141–158, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] Nelson Almeida Reverendo. Indoor autonomous navigation for service robots using beacons, 2018, Available at `http://hdl.handle.net/10773/25892` [Online accessed 1-June-2019].

[16] ROS. Global planner, 2018. Available at `http://wiki.ros.org/global_planner?distro=melodic` [Online; accessed 13-June-2019].

[17] Atomoclast. Graph-based path planning: A*, 2018. Available at `https://realitybytes.blog/2018/08/17/graph-based-path-planning-a/` [Online; accessed 13-June-2019].

[18] Turtlebot. Turtlebot 2. Available at `https://www.turtlebot.com/` [Online; accessed 10-June-2019].

[19] Sick. Lms1xx. Available at `https://cdn.sick.com/media/docs/5/15/415/Product_information_LMS1xx_2D_laser_scanners_en_IM0026415.PDF` [Online; accessed 3-June-2019].

[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Mg. Ros: an open-source Robot Operating System. 3:6, 2009.

[21] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[22] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, May 9-13 2011.

[23] Armin Ronacher. Flask, 2018. Available at `http://flask.pocoo.org/` [Online; accessed 6-June-2019].

[24] Armin Ronacher. Jinja2, 2008. Available at `http://jinja.pocoo.org/docs/2.10/` [Online; accessed 6-June-2019].

[25] Werkzeug, 2007. Available at `http://jinja.pocoo.org/docs/2.10/` [Online; accessed 6-June-2019].

[26] Rafael Arrais, Miguel Oliveira, César Toscano, and Germano Veiga. A mobile robot based sensing approach for assessing spatial inconsistencies of a logistic system. *Journal of Manufacturing Systems*, 43:129 – 138, 2017.

[27] yujinrobot. Kobuki. Available at `http://kobuki.yujinrobot.com/about2/`Sick web page [Online; accessed 10-June-2019].

[28] Eitan Marder-Eppstein. move base. Available at `http://wiki.ros.org/move_base` [Online; accessed 10-June-2019].