FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Autonomous Navigation with Simultaneous Localization and Mapping in/outdoor

Carlos Miguel da Silva de Freitas

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Armando Jorge Miranda de Sousa

Second Supervisor: Eurico Pedrosa

Third Supervisor: Luís Paulo Reis

July 23, 2020

Autonomous Navigation with Simultaneous Localization and Mapping in/outdoor

Carlos Miguel da Silva de Freitas

Mestrado Integrado em Engenharia Informática e Computação

Approved in an oral examination by the committee:

Chair: Prof. Hugo Sereno Ferreira
External Examiner: Prof. Artur Pereira
Supervisor: Prof. Armando Jorge Sousa

July 23, 2020

Abstract

The necessity and use of wheelchairs are increasing with the constant rise of the world population and human needs. Admittedly, the advancements in technology over the years have helped to push forward from manual to electrical powered wheelchairs. However, these are still inaccessible to many individuals with specific disabilities. With the technological improvements over the years, electrical powered wheelchairs have helped filled the restrictions that come with manual wheelchairs, but only to an extent.

Furthermore, the impact of intelligent wheelchairs can be recognized by allowing access to them to a more significant number of users, as they integrate smart and autonomous systems capable of behaviors such as obstacle detection and avoidance, apprehension of objects and autonomous navigation. Besides, given the complexity intelligent wheelchairs can obtain, it reaches a point where it is not feasible to properly test the system completely without any risks involved.

One of the most studied areas in the field of mobile robotics is the SLAM problem, where the robot achieves a certain level of perception of its surroundings with the use of external sensors like cameras and lasers. Therefore, this dissertation explores a platform for a wheelchair capable of simultaneous localization and mapping (SLAM) with autonomous navigation, applied on a simulation environment. The simulation allowed to test the wheelchair in different situations with a distinct set of sensors, as to conclude what achieves better results for indoor and outdoor environments. With the tests done it was possible to conclude that only sensors like the ZED 2 or the RPLidar A3 are capable of obtaining reliable results in all use case scenarios studied.

Additionally, the system is part of the IntellWheels 2.0 project, an intelligent wheelchair platform that strives to be capable of being easily adapted to any commercial electric wheelchair, aiding any person with any disabilities with different methods of inputs and a robust multimodal interface.

Keywords: autonomous navigation, IntellWheels, intelligent wheelchair, mobile robots, navigation, simultaneous localization and mapping, SLAM

ACM Categories: Computer systems organization → Embedded and cyber-physical systems → Robotics → Robotics autonomy; Computing methodologies → Modeling and simulation → Simulation support systems → Simulation environments

Resumo

A necessidade e uso de cadeira de rodas tem vindo a aumentar com o constante incremento da população mundial e das necessidades humanas. De facto, os avanços na tecnologia ao longo dos anos ajudou no avanço de cadeiras de rodas manuais para cadeiras de rodas elétricas. Todavia, ainda existem muitos indivíduos com certas deficiências que não conseguem facilmente usufruir destas cadeiras. Com os avanços tecnológicos ao longo dos anos, as cadeiras motorizadas ajudaram a combater as restrições das cadeiras de rodas manuais, mas apenas até certo ponto.

Inclusivamente, o impacto das cadeiras de rodas inteligentes pode ser reconhecido por permitirem acesso a um maior número de indivíduos, pelo facto de integrarem sistemas autónomos e inteligentes capazes de comportamentos complexos como deteção e desvio de obstáculos, perceção de objetos e navegação autónoma. Além do mais, dado o nível de complexidade que algumas cadeiras inteligentes podem atingir, chega-se a um ponto em que não é possível testar propriamente o sistema completamente sem quaisquer riscos envolvidos.

Uma das áreas estudadas em robôs móveis relaciona-se com o problema de SLAM, em que o robô atinge um certo nível de perceção dos seus arredores com ajuda de sensores externos como câmaras ou "lasers". Portanto, esta dissertação explora uma plataforma para uma cadeira de rodas com navegação autónoma, capaz de realizar o mapeamento e localização dos seus arredores simultaneamente, aplicada a um ambiente de simulação. A simulação permitiu testar essa mesma cadeira de rodas em diferentes situações com distintas séries de sensores, de forma a identificar os que obtém melhores resultados no interior e exterior. Com os testes realizados, foi possível concluir que com apenas sensores como o ZED 2 e RPLidar A3 são capazes de obter resultados de confiança em todos os casos estudados.

Adicionalmente, o sistema faz parte do projeto IntellWheels 2.0, uma plataforma para cadeiras de rodas inteligentes, com o intuito de ser facilmente aplicável em qualquer cadeira de rodas elétrica no mercado, ajudando qualquer pessoa com deficiências a usufruir de diferentes métodos de input e de uma interface multimodal robusta.

Keywords: navegação autónoma, IntellWheels, cadeiras de rodas inteligentes, robôs móveis, navegação, localização e mapeamento simultâneo, SLAM

ACM Categories: Computer systems organization → Embedded and cyber-physical systems → Robotics → Robotics autonomy; Computing methodologies → Modeling and simulation → Simulation support systems → Simulation environments

Acknowledgements

I would first like to thank my main supervisor Prof. Armando Sousa of the Faculty of Engineering at the University of Porto. Prof. Armando was always extremely nice and open-minded, allowing me to work on this project at my own pace while still guiding me on what I should prioritize and not during the development of the project.

I would like to acknowledge the help of my co-supervisors Luís Reis from the Faculty of Engineering at the University of Porto and Eurico Pedrosa from the University of Aveiro. They were helpful whenever I had any problems during the development of the dissertation.

Finally, I must express my very profound gratitude to my lovely parents and my close friends for providing me with continuous support and laughter throughout my years of study at FEUP and specially during these hard times of self-isolation during a worldwide pandemic. Thank you for not making my self-isolation as depressing as it could have been, this project would not be accomplished without you.

I would also like to thank the IntellWheels 2.0 (POCI-01-0247-FEDER-39898) project and all people involved in it. It was an honor having an impact, if not just a little, in the evolution of this ambitious project.

Carlos Freitas

"One day, in retrospect, the years of struggle
will strike you as the most beautiful."

Sigmund Freud

Contents

List of Figures

List of Tables

Abbreviations

CPU	Central Processing Unit
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
FOV	Field of View
FPS	Frames per second
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
IR	Infrared
IW	Intelligent Wheelchair
LIACC	Laboratório de Inteligência Artificial e Ciência de Computadores
LIDAR	Light Detection and Ranging
MAS	Multiagent system
MMI	Multimodal interface
PCL	Point Cloud Library
PW	Powered Wheelchair
RnD	Research and Development
RBPF	Rao-Blackwellized Particle Filter
ROS	Robot Operating System
SDM	Sparse-Dense Mapping
SLAM	Simultaneous Localization and Mapping
TBF	Triangulation Based Fusion
TOF	Time of Flight
VIO	Visual Inertial Odometry
VO	Visual Odometry

Chapter 1

Introduction

Wheelchairs have been around for centuries as a means of independent mobility for the aged and impaired. With the constant increase in the average life expectancy in most developed countries within the last few decades, the necessity for the use of wheelchairs by the elderly and helpless has consequently increased. Moreover, with further technological improvements in the last century, wheelchairs have evolved from simple manually powered wheelchairs to electric powered ones. Manual or electrical wheelchairs can be acceptable for most users with a low-level disability, allowing them to use the wheelchair independently. However, for patients with more severe cases of impediments, it is difficult or even impossible for them to use wheelchairs autonomously, making them rely on some other individual for assistance [1]. Furthermore, many studies have shown that different aspects of the configuration and design of manual wheelchairs have some consequences on the efficiency and challenge of mobility for the user [36, 48]. Multiple researchers have started studying and proposing various prototypes of intelligent wheelchairs to solve the problems within the use of standard ones. Intelligent wheelchairs strive to promote a new technological shift in the wheelchair market, by developing smart systems and autonomous behaviors, like wall following and obstacle avoidance, that improve the quality of life of the user.

As such, one of the main capabilities of an Intelligent Wheelchair (IW) is the aptitude toward autonomous navigation. For the numerous proposed prototypes over the last decades, navigation systems in Intelligent Wheelchairs have varied from a quite simple obstacle detection and avoidance system to a more elaborate, thoroughly autonomous navigation system in an environment known a priori. In the field of robotics, many studies have been made on proposing methods for improving the autonomy on a robot [43]. More recently, applying the capability of simultaneous localization and mapping on a mobile robot for a full navigation system has been heavily studied as a way to improve the 'intelligence' of the system.

So, this dissertation intends to produce a simulation environment of an intelligent wheelchair capable of autonomous navigation and Simultaneous Localization and Mapping (SLAM). The system will be part of a project named IntellWheels, more precisely its second iteration IntellWheels 2.0, a platform for developing IWs, with the intent of improving its perception and navigation systems.

1.1 Motivation

Current improvements in the computational power of GPUs and CPUs, and computer vision algorithms have made visual sensors a viable cheap way to apply SLAM in mobile robots. SLAM is a broad concept and a problem thoroughly studied in probabilistic robotics. Recently, many studies have been made on numerous methods of applying SLAM for different situations like in underseas and aerial environments [19, 27]. Being the concept of Visual SLAM viable only recently, most of the prototypes regarding IWs have not taken into account applying this type of methodologies for their navigation systems.

Therefore, this system is important as a new mark and shift in the navigation systems of IWs. Developing a system like this can be complex, but the impact that it can have in improving the wheelchair market and the people who need it can't be denied.

1.2 Problem statement

With the technological advances of the last few decades, there has been a push to make intelligent wheelchairs as a viable product to the people who might need them. The project IntellWheels is a platform for developing new IWs, which makes use of modular architectures that facilitate the integration of new methods and hardware. One of the most significant characteristics of an IW is the capability of providing the user with autonomous behaviors. There have been various navigation modes applied to intelligent wheelchairs, from semi-autonomous to completely autonomous ones. Notwithstanding, to make use of autonomous navigation with path planning, most intelligent wheelchairs systems require some restrictions, like knowing the environment it is moving beforehand. IntellWheels makes use of visual cameras to apply methods studied in probabilistic robotics, allowing the platform to navigate autonomously in a known environment.

Recent improvements in the area of mobile robotics have allowed the implementation of simultaneous mapping and localization using cheap visual sensors. SLAM implementation on an intelligent wheelchair can improve the navigation of the system considerably without the need for any previous knowledge of the map. Additionally, the application of SLAM methods using semantics can also help the IW perceive like a human and improve the life quality of its users with more personalized navigation methods. Moreover, the complexity of systems like these can make it challenging to fully test the system in real life without possible consequences. Thus, the main objective of this dissertation was to implement a system for a wheelchair capable of autonomous navigation and simultaneous localization and mapping for both indoor and outdoor environments. The system was implemented in a simulation environment with a wheelchair making use of different sets of sensors, as to perceive the best setup needed.

Given a wheelchair with a set of sensors simulated in a robot simulator such as Gazebo[1], the system implemented in ROS[2] processes the data from the sensors with the aid of the Point Cloud

[1] http://gazebosim.org/
[2] www.ros.org

Library[3] (PCL) and makes uses of appropriate methods for SLAM and autonomous navigation. Furthermore, the simulated wheelchair was planted in different situations with different kinds of sensors that try to represent real life ones, enabling the possibility of interpreting the minimum setup needed for the wheelchair to work in both indoor and outdoor environments.

1.3 Objectives

The dissertation strives to develop a simulation environment of an intelligent wheelchair capable of path planning and simultaneous localization and mapping, using SLAM methodologies, for both indoor and outdoor environments. As such, the following objectives were proposed:

- Gather knowledge of the SLAM concept and the social impact of applying it on intelligent wheelchairs.
- Develop a simulation environment capable of:
  - Making a wheelchair model capable of manual navigation through any given world.
  - Applying different types of simulated sensors with different characteristics in distinct positions in the wheelchair.
  - Applying different SLAM methods on the simulated wheelchair in order to map the environment.
  - Showing the user a map of the environment and its current location within the map.
  - Applying different path planning methods with different configurations.
  - Allowing path planning to run simultaneously with SLAM.
  - Allowing the user, given the map and the wheelchair current position, to set a point to where the wheelchair should navigate to and to observe the wheelchair trying to get to that point if possible .
  - Allowing the wheelchair to apply SLAM and path planning in either indoor or outdoor environments.
  - Being easily applicable to a real wheelchair.
- Test the simulation with different sensors and configurations, to identify the best type of sensors to use for most cases.
- Gather an idea of what can be done better to improve the system in future work.

[3] https://pointclouds.org/

1.4 Document Structure

The dissertation starts by introducing the problem around the project, explaining the importance of wheelchairs in today's society and the impact that intelligent wheelchairs can have. Additionally, this initial chapter briefly describes the project, the motivations behind it, and clarifies the importance of the system on improving and innovating IWs. Furthermore, a record of the objectives expected to be achieved at the end of the project is explicitly declared.

The following chapter (Chapter 2) reviews some of the already made researches and projects about intelligent wheelchairs, what they are and distinguishes one from another and how important they are. Also, it takes a short look at the navigation systems on some proposed IW prototypes, following a more in-depth look at the 'IntellWheels' development platform, the importance of it and the innovations that it brings to the market.

The third chapter (Chapter 3) goes through the current state of the art and related studies concerning the problem of Simultaneous Localization and Mapping in robotics, revealing the history behind it and explaining some of its main concepts and proposed solutions. It also furthers the knowledge and implementation methods used on the SLAM system of the project.

The next chapter (Chapter 4), goes through some different types of sensors used on SLAM systems. Additionally, it talks in detail about some of the sensors that were analyzed and tested in simulation, as to understand the minimum viable sensor capacity to be applied on IntellWheels.

The fifth chapter (Chapter 5) explains the development process of the simulation environment. It goes through the various stages of the simulation, from the wheelchair model to the path-planning component. It explains in detail how the system works as a whole and the packages used.

The sixth chapter (Chapter 6) shows the experiments done in the simulation made to judge potential problems on the system, counting the best parameters and sensor configurations to be used for the simulated wheelchair to behave the best in most situations.

Finally, the last chapter (Chapter 7) states the conclusions regarding the importance of the simulation in correlation with the IntellWheels 2.0 project, along with potential future work on improvements on the current system, or on new features.

Chapter 2

Intelligent Wheelchairs

The elderly and people with impairments, disabilities or some other serious injuries, need to rely on wheelchairs to achieve a higher degree of autonomous mobility. However, manual wheelchairs are often hard to handle, making it extremely difficult for users with some impairment in their upper limb function. Many studies around manual wheelchairs have found that most users exhibit problems in propelling their wheelchairs in their daily lives, even if they do have minimal limb function required to use the wheelchair [48]. Also, the efficiency of the mobility of a manual wheelchair can be intrinsically related to the design and configuration of the chair. Variations in the wheelchair design such as backrest height, backrest and seat angle, and leg and foot support, can hold serious effects on propulsion forces, the range of motion of the upper limb joints and, more importantly, in the efficiency of the system [36]. Overall, it is complex to design a wheelchair that would satisfy the needs and necessities of all users.

With the constant shift in technology, the surface of Powered Wheelchairs (PW) in the market brought immediate improvements to the quality of life of its users. Nonetheless, most PWs only enhance the way to handle the chair, from manual propulsion forces to controlling a joystick, but overall, they provide the same or similar functions as the manual ones. Users with severe injuries and impairments, for instance vision problems, Tetraplegia or Parkison, still find it impossible to handle PWs independently, making it necessary for the assistance of another individual [1].

In conclusion, the emergence of intelligent wheelchairs is important for overcoming the shortcomings of manual and powered wheelchairs. Intelligent wheelchairs are designed to assist the user in various ways, by reducing or even removing the users' responsibility for handling the chair. They can also be devised to work for a variety of user types according to their restrictions.

2.1 Background

Commercially available wheelchairs, due to the reduced capacity of maneuvering and handling can still be pretty difficult to be used by people with severe disabilities, not only in terms of moving the chair but also in recognition of obstacle detection and avoidance. Intelligent wheelchairs try to solve this problem by improving on PWs technology by adding several layers of intelligent

systems that can help improve the independence and quality of life of the user. An Intelligent Wheelchair (IW) is a Powered Wheelchair (PW) to which computers and sensors have been added or a mobile robot base to which a seat has been attached [33]. Therefore, IWs strive mainly to aid in maneuvers like navigation with obstacle avoidance and wall following. These systems are supported by special hardware, being sensors such as cameras or lasers and other types of inputs imaginable. Across the last few decades, there has been an effort in research and development (RnD) on intelligent wheelchairs and also in assistive technology for people with necessities, with a lot of prototypes of IWs proposed making improvements in the field of inputs, sensors, operation modes, and human factors [33, 51].

2.1.1 Input Methods

The best choice of user input for IWs is not completely obvious, because it is mainly user-specific. Intelligent wheelchairs have been improving the capacity of input methods over the past few years. From the simple joystick controller used in PWs, most IWs have focused on creating inputs that would benefit people with severe injuries by controlling the wheelchairs with voice control, fingertip control, head movements and even more recently, with brain thoughts [33]. The diversity of inputs has helped making IWs stand out. However, with the increasing number of inputs, the complexity of the system also grows. Therefore, for IWs that make use of a variety of inputs or very complex ones, research studies had to be made for determining ways to make the interaction between the human and the system seemingly natural. As such, studies on improving multimodal interfaces for these systems have been made over the years [54, 58, 50].

2.1.2 Operation Modes

Operation modes in IWs can range between fully autonomous to semi-autonomous, depending on the task and necessity of the user. For example, if a user is unable to plan and execute a path to a destination, he would benefit from an autonomous operation. On the other hand, if the user can plan and execute a path it may be more useful to have the system stick to only collision avoidance, giving him some sense of control. The operation modes should be adequately designed around peoples' abilities and desires and also the environment they are in, giving them as much control as possible while still aiding them in their task. Some of the subtopics in the field of IWs operating modes research are [33, 51]:

- *Machine Learning* - use of neural network algorithms to detect obstacles and make pre-taught routes.
- *Following* - make the IWs be able to move alongside a companion, tracking it using mostly laser sensors based on Kalman filter.
- *Localization and Mapping* - make use of various sensors to gather data effectively to map the environment and help calculate the visual-inertial odometry (VIO) through the extended Kalman filter (EKF).

- *Navigational assistance*

Moreover, other prototypes like IntellWheels has also improved on the idea of test driving by allowing operation modes between simulation in virtual, or even mixed reality [9]. A more detailed look on operating modes can be seen in chapter 2.3.

2.2 Related Work

The emergence of intelligent wheelchairs can be traced back to over thirty years ago. Since then, a lot of ideas and prototypes have been introduced. This segment will focus on revealing some of the work done on the field of intelligent wheelchairs.

- **Mr. Ed (1990)** [21, 51]

  One of the earliest works on the development of intelligent wheelchairs was a system made by Connell and Viola, at the IBM T.J. Watson Research Center. The system, eventually named Mr. Ed, consisted of a chair on top of a robot. The wheelchair could be controlled using a joystick mounted on the arm of the chair and connected to the robot. Additionally, the system was able to perform certain navigation tasks like avoiding obstacles or pursue other moving objects with the help of eight infrared proximity sensors and two sonar sensors.

- **TinMan I and II (1994-1999)** [21, 51]

  TinMan I and II were developed by David Miller and Marc Slack at the KISS Institute for Practical Robotics (KIPR). Differently to other approaches, instead of using mobile robots, TinMan took advantage of already available powered wheelchairs like Dynamics and Penny and Giles. The original prototype used a mechanical interface for the wheelchair joystick, but in TinMan II (figure 2.1), a supplementary wheelchair controller was added between the joystick and the standard motor controller. With its sensors, the chair could go through openings, avoid obstacles, wall following, backtracking, and docking with minimum effort needed.

Figure 2.1: The TinMan II wheelchair [21]

- **NavChair (1991)** [51, 21]

  The development of NavChair (figure 2.2) started in 1991 by Simon Levine at the University of Michigan. The main idea of the project was to port the Vector Field Histogram (VFH) method used in autonomous robots to avoid obstacles [6]. However, the differences in the power base between robots and wheelchairs forced the method to be modified. As such, the Minimum VFH was born, giving the user a more variable control in manipulating the wheelchair.

  Nonetheless, NavChair was capable of avoiding obstacles, follow walls and travel safely in cluttered environments. It used twelve ultrasonic sensors and an on-board computer, with a control system where the user could plan the route, navigate and indicate the direction and speed of the travel. The chair also had various modes of autonomy, so the system could easily adapt to the users' behavior in real time.

Figure 2.2: The NavChair wheelchair [21]

- **CALL Centre Smart Wheelchair (1994)** [21, 40]

  The CALL Centre IW was originally developed as a motivating educational and therapeutic resource for severely disabled children. The chair was made to initially assist in the assessment of physical, cognitive, social and communicative skills.

  Moreover, the chair could be controlled through various inputs such as switches, joysticks, laptop computers, and voice-output. The architecture of the system was modular enough to simplify the addition of new features. It was one of the first wheelchairs with a report submodule called *The Observer* that was tasked to notify the user in detail everything the system was doing.

- **Wheelesley (1998)** [21, 60]

  Wheelesley (shown in figure 2.3) was an experimental standard powered wheelchair, developed by Holly Yanco, with an onboard computer, sensors and a graphical user interface. With the GUI, the user could point to in which direction he wanted to move to, specify the requirement of some tasks like going up a ramp, and keep a record of an environment and important details of said environment.

  Besides, the Wheelesley could turn in any place and easily avoid obstacles. It had twelve proximity sensors, six ultrasonic range sensors, two shaft encoders and a front bumper with sensors. The main purpose of this chair was to facilitate the control of a chair for those that were unable to use the standard joystick interface, so, a system for controlling the computer using the angle of the eyes was also implemented.

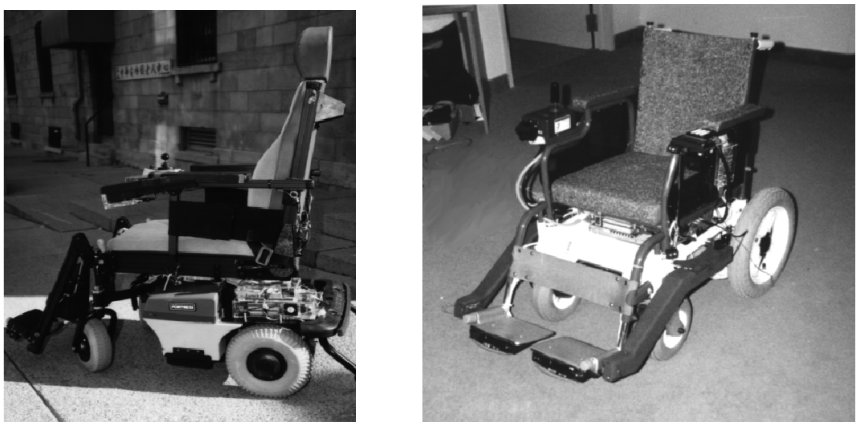
Figure 2.3: Wheelesley prototype [60]

- **TAO-1 and TAO-2 (1995-1998)** [51, 21]

  The TAO-1 and TAO-2 (figure 2.4) are wheelchairs based on powered ones with minimal modification required. The TAO family of wheelchairs, with the help of various sensors controlled by a micro-controller, such as cameras, bump sensors and infrared sensors, was capable of:

  - Avoid basic obstacles
  - Pass through narrow corridors
  - Enter through a narrow doorway
  - Maneuver in a tight controller
  - Landmark-based navigation

  The second iteration of the TAO wheelchair TAO-2 had the same functionalities and setup of the first one but was mostly done on a Suzuki powered wheelchair from Japan [51]. The project done on the Suzuki MC-13P did not perform as good as TAO-1 out of the box, so, some adjustments had to be made.



(a) TAO-1 (b) TAO-2

Figure 2.4: The TAO-1 and TAO-2 prototypes [21]

- **IntellWheels (2007)** [9, 8]

  IntellWheels is a platform for developing intelligent wheelchairs, the project started in 2007 at the Faculty of Engineering of the University of Porto. Since the project done in this dissertation is part of IntellWheels, a more detailed look of the platform is discussed in section 2.4.

- **Modern robotic intelligent wheelchair (2009)** [37]

  It is one of the most recent studies about developing a wheelchair that is safe, low-cost and intelligent. According to [37] the proposed chair has three main technological necessities:

- The motion of the chair to a place needs to be completely safe and comfortable for the passenger.
- The surrounding environment should be accurately mapped in 3D so the system can easily identify traversable and non-traversable obstacles.
- The spatial representation of the robot should facilitate infrequent requests for directions and allow natural commands while being as cost-efficient as possible.

What makes this project important is that it is one of the prototypes that try to be low cost while using well-studied concepts in robotics like 3D mapping and LIDAR simultaneous navigation and mapping.

Figure 2.5: An intelligent wheelchair prototype [37]

Overall, many more intelligent wheelchairs have been studied and developed over the last years [35]. Some of them try to bring something new and completely different from anything seen in the area, like Wellman's proposed hybrid wheelchair with two legs equipped to help climb stairs and move through rough terrain [8].

With the current evolution of technology and more into robotics, the intelligent systems applied to wheelchairs can be constantly improved.

2.3 Navigation on Intelligent Wheelchairs

Intelligent wheelchair's navigation mode can be fully autonomous or semi-autonomous, depending on the users' necessities. Over the years, the capacity of navigation done by IWs has evolved considerably, with different navigation modes like [51]:

- *Autonomous Navigation with Obstacle Avoidance* - Wheelchair can travel from its current location to a given destination based on a known a priori internal map. Some of these systems do not have obstacle avoidance when navigating autonomously.

- *Collision Avoidance* - The user is responsible for planing the path to the desired destination. The chair would automatically avoid or stop at incoming obstacles.
- *Wall Following* - IW maintains a fixed distance from the wall it is following.
- *Door passage* - IW can transverse through doorways.
- *Docking* - IW allows the user to approach an object very closely.
- *Trajectory Playback* - IW can reproduce a programmed path, that was previously added by demonstration.
- *Reverse Trajectory* - IW can return to its starting position by undoing its actions.
- *Target Tracking* - IW can track and move to a stationary or moving object.
- *Line Following* - IW can follow the track that is physically marked on the environment.
- *Turn around* - IW can reverse direction.
- *Bump and backup* - IW stops and backs up when the contact sensor is activated

Furthermore, with the improvement of the sensors, algorithms and computational power of computers, these navigation modes can evolve to more complex methods like simultaneous localization and mapping navigation [35, 37].

2.4 IntellWheels

IntellWheels is a project that originated in 2007 at the Faculty of Engineering of the University of Porto. Unlike the other prototypes of IWs mentioned in the previous sections, IntellWheels is not explicitly an intelligent wheelchair, but more of a platform that strives to facilitate the development of new intelligent wheelchairs. In essence, IntellWheels tries to create a platform that is modular not only on the software infrastructure, with a multi agent system, but also on the hardware one, making it capable of being applied on any powered wheelchair available on the market with the minimum amount of effort needed. Also, IntellWheels tries to fix the problem of some IWs being only operational for a specific type of people with particular impairments, by improving the human-machine interaction [8].

Being designed through a multi agent system (MAS), IntellWheels easily allows the integration of distinct sensors, actuators, user inputs methods, navigation methods and algorithms, intelligent planning procedures and cooperation methodologies. Moreover, the platform architecture was divided into several modules (figure 2.6), each module responsible for an integral function of the system.

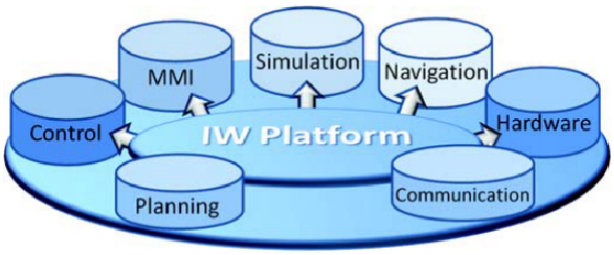


Figure 2.6: IntellWheels modules [9]

Additionally, the IntellWheels MAS architecture follows the standards of the Foundation for Intelligent Physical Agents (FIPA), with four main agents that can communicate between themselves through the agent communication language (figure 2.7) [9].

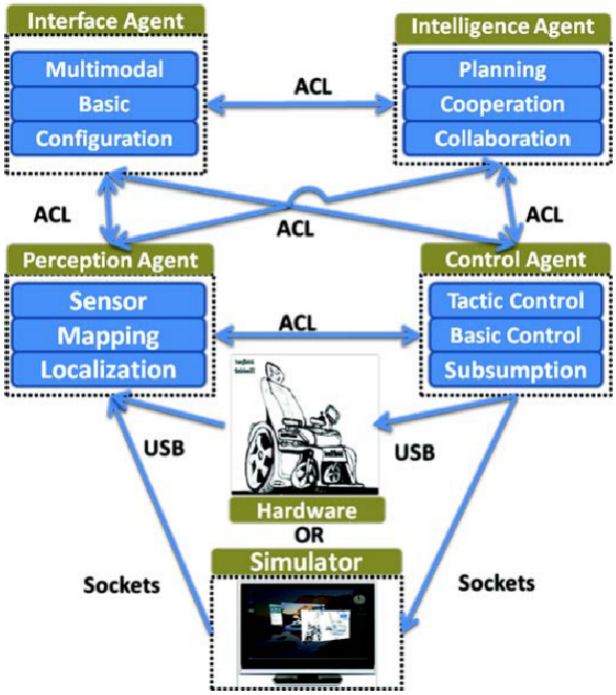


Figure 2.7: IntellWheels software architecture [9]

Besides the four main agents, the platform makes use of other agents to perform more specific tasks. Furthermore, another one of the IntellWheels innovative features is the simulation module, which allows the user to test-drive the platform in virtual or mixed reality, facilitating as well the test of new features and algorithms on the development of the wheelchair.

The navigation and control modules make use of the hardware architecture of IntellWheels to perform autonomous navigation with obstacle avoidance on a map known a priori. The first iterations of the project used a visual sensor, like a camera, to perform the visual odometry of the chair on the known environment, by making use of studied methods on probabilistic robotics [8].

Overall, the IntellWheels project allowed for the creation of a functional simple prototype of an intelligent wheelchair, capable of various things like voice command, obstacle avoidance, intelligent planning of tasks, and others. More recently, a second iteration of the original IntellWheels project, namely IntellWheels 2.0 has been worked on, following the previous work and research done as a way to improve it and obtain a more complete and robust prototype. Therefore, the IntellWheels 2.0 project focus on developing four fully functional products, that will aid in achieving the main goals of the original project:

- Creation of an Intelligent Wheelchair framework, that allows to transform motorized wheelchairs into intelligent ones with minimum effort and costs. It should include all hardware and software necessary to solve the problems of sensing, mapping, localization and navigation planning.
- Development of a realistic Intelligent Wheelchair simulator, with a 3D interface, virtual reality and games that will allow to learn and train users to drive the wheelchair and related tasks.
- Creation of a more complete and configurable multimodal interface with bi-directional multimodality and control capabilities not only of the wheelchair itself but also of other systems.
- Complete Intelligent Wheelchair prototypes with full integration of the before mentioned framework and multimodal interface.

This dissertation focuses on the IW framework, by studying and evaluating the possible sensors that could be used for SLAM and path-planning methods, on a simulation environment.

Chapter 3

Simultaneous Localization and Mapping

Navigation is a central requirement for achieving a fully autonomous mobile robot [15]. To be able to navigate through an environment, a mobile robot needs to know its location with the aid of a map. If the task of the robot is as simple as to reach a determined goal point on the same surroundings, the mission goal can be determined a priori with an already mapped environment. However, if the task is more complex, such as exploring an unknown environment, the robot can not make use of a previously examined description of the surroundings. A wheelchair is steadily making through different locations (either indoor or outdoor), as such, having predetermined maps of the environments is not a viable solution for achieving a fully autonomous wheelchair.

Implementation of navigation systems that makes use of already known maps of the environment is pretty forthright for today's robots, even for dynamic environments. Also, the task of building a map knowing the exact location of the robot is mostly an already solved problem in robotics. Yet, it is much harder if a robot is to do both simultaneously. This is where the problem known as Simultaneous Localization and Mapping [57] comes in.

3.1 Introductory Concepts

Simultaneous Localization and Mapping, also known as SLAM, is an open concept on mobile robotics where the robot, equipped with onboard sensors, tries to estimate its state while at the same time construct a representation of the environment the sensors are perceiving. The robot state estimation can be as simple as perceiving its pose (position and orientation) to something more complex such as robot velocity, sensor biases, and calibration parameters. Also, the map can be a simple representation of points of interest, like landmarks and obstacles, but various methods also try to achieve full metric map representation on either 2D or 3D models, and more recently, with semantics involved.

SLAM is a significant problem in mobile robotics, that has been studied for more than thirty years. The SLAM problem is often compared to the question of what came first, the chicken or the egg? To move precisely, a robot needs an accurate depiction of the environment. Yet, to produce an exact portrayal of its surroundings, the robot needs to know precisely its location [12]. Despite that, in practice, the localization procedure can be done using the map built so far, and consequently, the map can be updated using the calculated pose estimation. Thus, although localization and mapping are theoretically dependent on each other, they can be handled independently in spite of possible sub-optimal results [46].

In general, there are two main approaches to the SLAM problem, *Online SLAM* and *Full SLAM*. Where, at each step, in *Online SLAM* only the current state of the robot is updated along with the map, and in *Full SLAM*, the full trajectory made up until now is updated along with the map.

3.1.1 Full SLAM

The full SLAM approach, tries to estimate the map and entire path transversed by the robot. As such, the pose of the robot is only calculated at end of the trajectory made by the robot. A probabilistic definition of the full SLAM approach can be given by [57]:

$$u_{\{1:t\}} = \{u_1, u_2, u_3, ..., u_t\} \tag{3.1}$$

where u represents the robot control at time t (odometry),

$$z_{\{z1:t\}} = \{z_1, z_2, z_3, ..., z_t\} \tag{3.2}$$

where z represents robot observations at time t,

$$m \tag{3.3}$$

$$x_{\{1:t\}} = \{x_1, x_2, x_3, ..., x_t\} \tag{3.4}$$

where m represents the calculated map and x the obtained robot pose at time t. As such, for full SLAM the robot trajectory estimates can be represented by (figure 3.1):

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}) \tag{3.5}$$

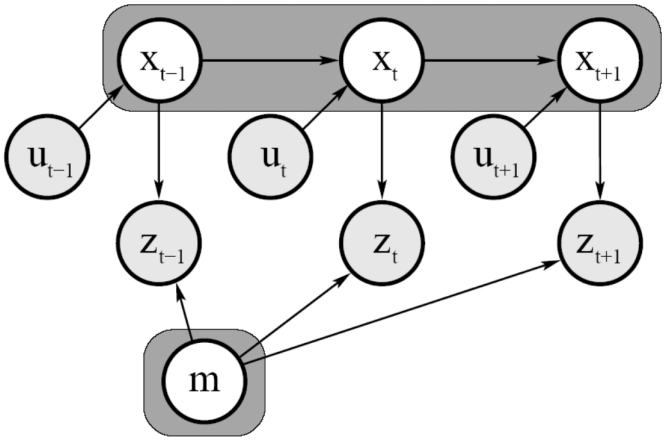


Figure 3.1: Graphical model of full SLAM approach [11]

3.1.2 Online SLAM

On contrary to the full SLAM approach, online SLAM tries to estimate the posterior of the current pose along with the construction of the map. Given the variables shown in the previous subsection (*equations* 3.1, 3.2, 3.3 and 3.4), similarly to *equation* 3.5, the robot position and map at time t can be represented as (figure 3.2):

$$p(x_t, m | z_{1:t}, u_{1:t}) \tag{3.6}$$

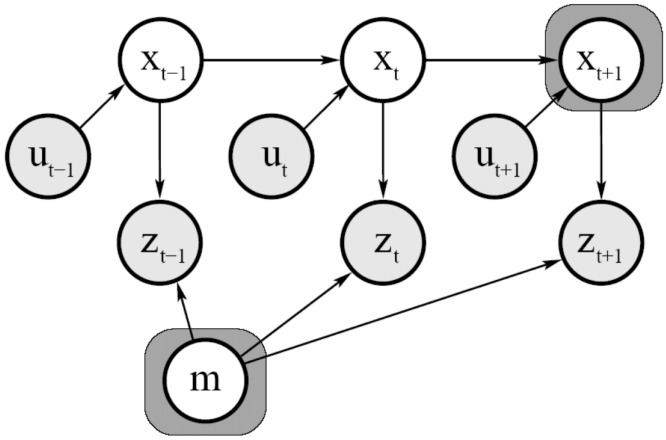


Figure 3.2: Graphical model of online SLAM approach [11]

Online SLAM only considers variables that persist at time t. Therefore, solutions for this approach are incremental: they discard past measurements and control data after they have been handled. Overall, the online SLAM problem is the result of integrating out past poses from the full SLAM approach [57]:

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int\int ... \int p(x_{1:t}, m|z_{1:t}, u_{1:t}) dx_1 dx_2 ... dx_{t-1} \tag{3.7}$$

3.2 Representation and estimation

When applying SLAM, mapping can be either in 2D or 3D. Mapping geometry in 3D is more complex and depends on the hardware used and wanted results. Moreover, 3D representation can be demanding of memory and do not provide any high-level understanding of the surroundings, as such, 2D mapping is enough for most applications. Beyond 2D and 3D representations, there are various SLAM approaches for map representations, like dense and sparse depictions of the environment [12, 45].

Therefore, there are various approaches for SLAM representation, being *grid-based* for dense portrayals, *feature-based* and *graph-based* for more sparse depictions, *topological* for recognition of locations types, and *semantic* for a more high-level understanding of the environment.

3.2.1 Feature-based SLAM

Feature-based approaches to SLAM make use of easily identifiable elements in the environment, and build an internal representation with the location of these landmarks. Historically, the earliest and most influential SLAM algorithm is based on the Extended Kalman filter (EKF). EKF SLAM makes use of the EKF to online SLAM by using maximum likelihood data association [57].

3.2.1.1 EKF SLAM

The goal of the SLAM process is to use the perceived environment to update the position of the robot. Given that the odometry of the robot can be erroneous, sensor measurements of the surroundings can be used to correct the position of the robot. This is done by extracting landmarks from the environment and re-observing them when the robot moves around. The Extended Kalman Filter is responsible for updating the robots beliefs of where it is based on the extracted landmarks. A brief outline of the SLAM process can be shown as in figure 3.3.

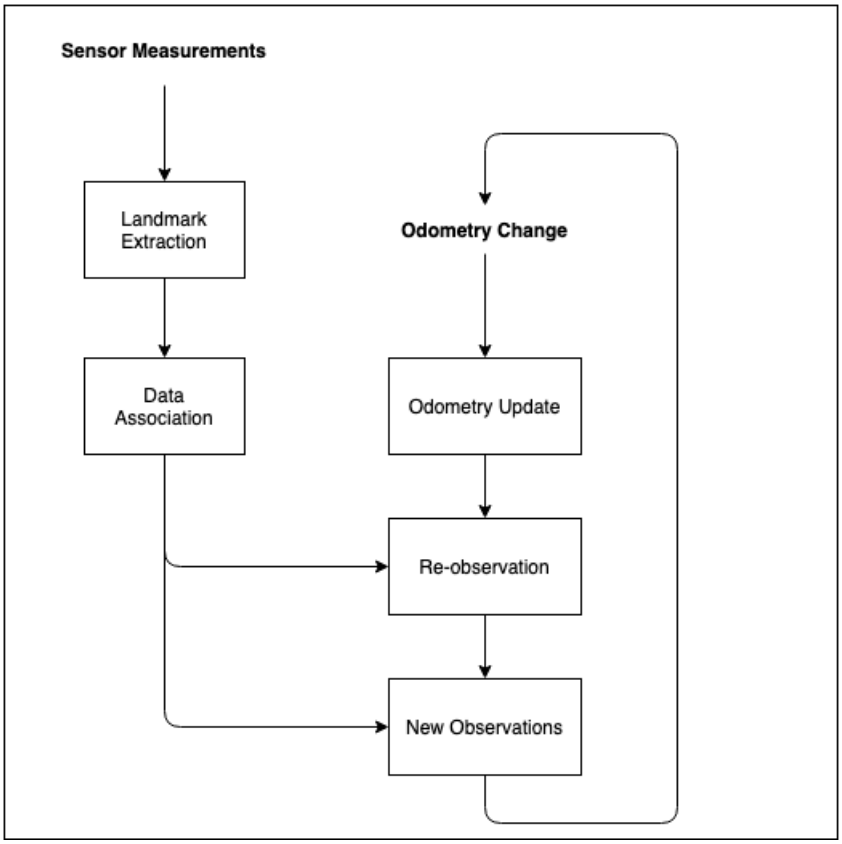


Figure 3.3: Overview model of feature-based SLAM, adapted from [49]

When the odometry changes, the uncertainty pertaining to the robots new position is updated with the EKF using Odometry Update. Landmarks are then extracted from the sensors measurements and from the robots new position. After that, the robot tries to associate these extracted landmarks to previous observed landmarks. Re-detected landmarks are then used to renew the robots position in the EKF. New discovered landmarks are added to the EKF as new observation so that they can possibly be re-observed later on.

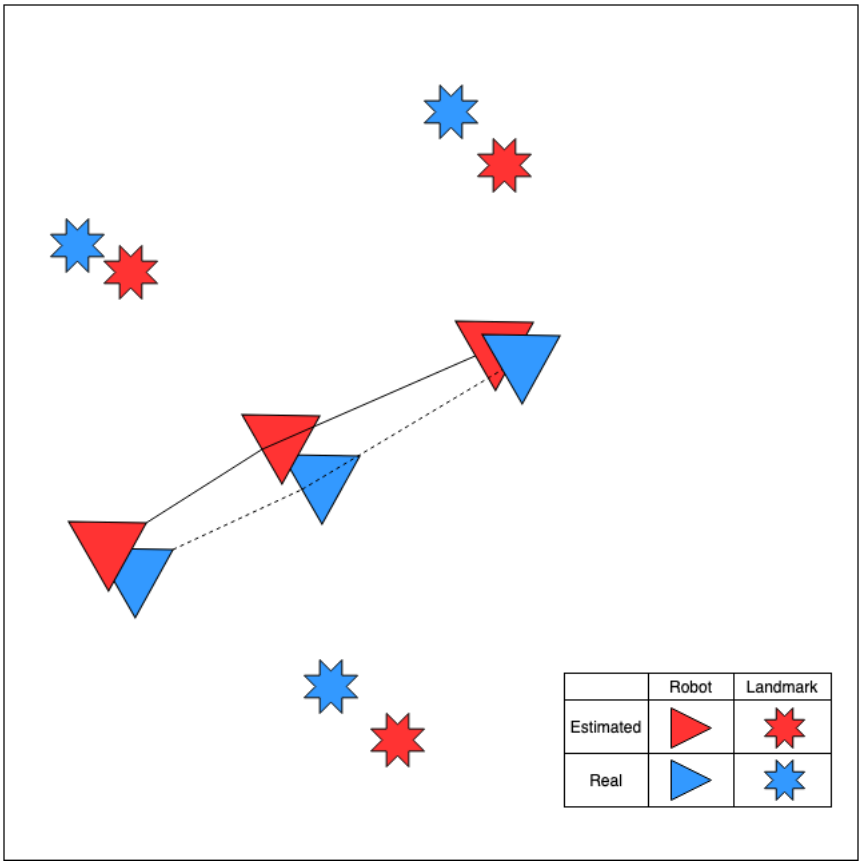


Figure 3.4: Feature-based map with estimated positions vs real positions

Overall, in feature-based SLAM, representations are composed of point landmarks (figure 3.4). More precisely, EKF SLAM makes use of a "stochastic map" of spatial relationships. *Stochastic mapping* provides metrically accurate navigation robustly and the possibility of incorporating false returns, drop-out and data association ambiguities [62]. A good stochastic mapping needs a good set of extracted landmarks, and reliable data association. In its naive form, the computational complexity of the EKF SLAM scales quadratically with the number of landmarks. As such, for computational reasons, the number of landmarks extracted tend to be small [4]. Furthermore, EKF SLAM makes a Gaussian noise assumption for the robot motion and perception.

3.2.1.2 Landmarks

The Extended Kalman Filter approach, tends to work well the less ambiguous the landmarks are. As such, it is essential for the SLAM method to correctly associate observations of landmarks with landmarks held in the map. Incorrect association can lead to catastrophic failure [57].

Landmarks are features which can be easily recognized and differentiated from the environment. Moreover, landmarks are used by the robot to localize itself. The type of landmarks a robot uses depends on the type of environment it is situated in. In general, the main points for good landmarks' extraction are as follows [49]:

- **Re-observable**- landmarks should be re-observable by allowing them to be detected from different positions and angles.
- **Unique**- landmarks should be unique so that they can be identified from one time-step to another without mixing them up. If the robot re-detects two landmarks at a specific time, it should be easy for the robot to identify which landmark is which. Landmarks should not be very close to each other.
- **Plentiful**- although feature-based SLAM can become computationally demanding with higher numbers of landmarks, the number of extracted landmarks should not be scanty as to not allow the robot to spend an extended amount of time without enough visible landmarks as it may get lost.
- **Stationary**- a chosen landmark should be static. Using a moving object as a landmark is a bad idea since it makes impossible for the robot to calculate, given the moving landmark, in which place it is.

There is a discrete amount of methods for extracting landmarks. The methods used can depend on the type of landmarks needed as well as the sensors used. Some first applications of SLAM made use of laser scanners, for these type of sensors, Spikes and RANSAC methods are some of the extract algorithms used [49]. Both of these extraction methods are fitting for indoor environments.

Spikes makes use of *extrema* to find landmarks. They are identified by finding values in the range of a laser scan where two values differ by a certain threshold. This method has the problem of identifying people as good landmarks, therefore it is not suitable for a lot of situations. Opposite to Spikes, RANSAC (Random Sampling Consensus) is used to extract lines from a scan, and these lines can be turned into landmarks. As such, it will not pick people as viable landmarks [49].

3.2.1.3 Data association

Data association refers to the robot matching observed landmarks from different sensor readings with each other. It is particularly important when a robot returns to a previously mapped region after a long tour, the alleged loop-closure problem. Before fusing data into the map, new

measurements are associated with existing map landmarks, and, after merging, these associations can not be revised. Thus, a single incorrect data association can cause divergence into the map estimate, generating catastrophic failure of the localization algorithm.

There are various methods for data association, like, batch-validation methods that exploit constraints inherent in the SLAM formulation, appearance-based methods, multi hypothesis techniques and the nearest-neighbour approach [4, 49].

In practice there are some problems that can, but should not, arise in data association:

- It is possible for the robot not recognize landmarks every time step.
- The robot might see something as being a landmark but fails to ever come in contact with it again.
- The robot might wrongfully associate a landmark to a previously extracted one.

Given that the landmarks should be easy to observe (see subsection 3.2.1.2). The first two problems above are not acceptable for a landmark. Even so, with a good landmark extraction method these problems can still arise, so it is best to define a suitable data-association policy.

3.2.2 Grid-based SLAM

Theoretically the simplest method is the grid-based approach. In this approach, the environment is split into a grid of cells of a specific size (figure 3.5). Each cell has the likelihood of being occupied by an object. For example, a cell with a value of 1 would be considered occupied, and another with a value of 0 would be completely free.

There are various approaches to grid-based SLAM. One the most used is with the help of Rao-Blackwellized particle filters (RBPF) [17], where each particle carries an individual map of the environment.

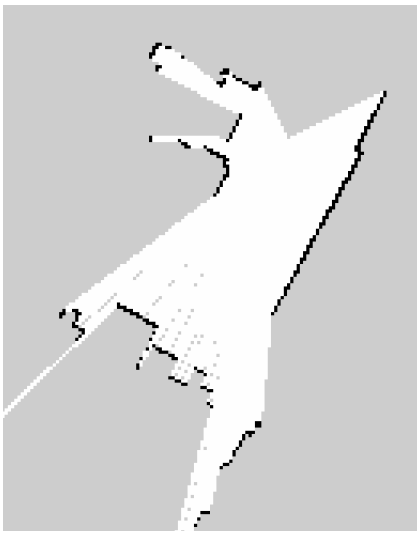


Figure 3.5: A grid-based map

3.2.2.1 RBPF SLAM

Particle filters are powerful sampling-based learning algorithms for dynamic Bayesian networks. One the most used for grid-based SLAM is the Rao-Blackwellized particle filter that exploits the dynamic Bayesian networks to obtain better efficiency compared to other particle filters [17].

The main idea of the Rao-Blackwellized SLAM is to estimate the posterior $p(x_{1:t}|z_{1:t},u_{1:t})$ where $x_{1:t}$ is the potential trajectory done by the robot given its observations $z_{1:t}$ and its odometry measures $u_{1:t}$, and then, use this posterior to calculate the posterior over trajectories and maps [23]:

$$p(x_{1:t},m|z_{1:t},u_{0:t}) = p(m|x_{1:t},z_{1:t})p(x_{1:t}|z_{1:t},u_{1:t}) \tag{3.8}$$

The mapping posterior $p(m|x_{1:t})$ can be calculated efficiently given the knowledge of $x_{1:t}$ and $z_{1:t}$, while the localization posterior $p(x_{1:t}|z_{1:t},u_{1:t})$ can be estimated using a particle filter in which to every sample is associated an individual map. Each map is built given the robot's measurements $z_{1:t}$ and trajectory $x_{1:t}$ defined by the corresponding particle. The robot's trajectory is dependent on its motion, therefore, the proposal distribution is considered equivalent to the probabilistic odometry motion model [23].

A prevalent particle filtering algorithm is the Sampling Importance Resampling (SIR) filter, which approximates the filtering probability density by a weighted set of N samples

$$\{(w_t^{(i)},x_t^{(i)} : i \in \{1,...,N\}\} \tag{3.9}$$

where $w_t^{(i)}$ is the weight of the sample $x_t^{(i)}$. By updating a set of samples representing the posterior about the map and the trajectory of the mobile robot, it is possible for a Rao-Blackwellized SIR filter for mapping to incrementally process the odometry and measurements readings as they are available. This can be done with the following four steps [23]:

1. *Sampling*: Where the next set of particles $\{x_t^{(i)}\}$ is obtained from the current one $\{x_{t-1}^{(i)}\}$, by sampling from a distribution $\pi(x_t|z_{1:t},u_{1:t})$.

2. *Importance Weighting*: Where an importance weight $w^{(i)}$ is assigned to each sample, defined by

$$w^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t},u_{1:t})}{\pi(x_{1:t}^{(i)}|z_{1:t,u_{1:t}})}. \tag{3.10}$$

   The weights account for the matter that the distribution π normally is not equal to the true distribution of future states.

3. *Resampling*: Where samples with a low importance weight w are probably renewed by particles with a high weight. This step is important since only a finite number of particles are used to approximate a continuous distribution, granting the application of the filter where the proposal distribution is different from the real one.

4. *Map estimating*: Where for each pose filter $x_t^{(i)}$, the corresponding map $m_t^{(i)}$ is calculated based on the trajectory and history of observations according to $p(m_t^{(i)}|x_{1:t}^{(i)}, z_{1:t})$.

This generic algorithm specifies a framework for Rao-Blackwellized mapping, thus, the methods on how the proposal distribution should be computed and when the resampling should be done are up to the different proposed solutions for SLAM based on Rao-Blackwellized particle filters.

The main problem from most grid-based RBPF SLAM approaches is the complexity given by the number of particles. Some approaches need a high number of particles to obtain consistent good results. However, there have been approaches that strived to combat this complexity by reducing the necessary samples needed. The most notorious proposal that improved immensely the RBPF SLAM was GMapping [23, 24].

To increase the performance of RBPF SLAM, GMapping presents two main approaches. Firstly, it computes the proposal distribution by assessing the likelihood around a particle-dependent pose obtained by a scan-matching process. Therefore, the last reading is considered while formulating the new particle, allowing to estimate the progression of the system according to a more informed model compared to others that just make only use of the last odometry measurement. This allowed for a significant reduction of the required particles. Secondly, it applies a resampling strategy that only performs resampling when it is needed, keeping a reasonable particle diversity [23].

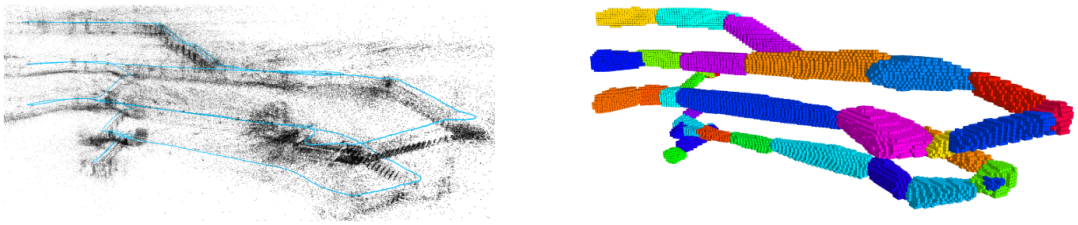
Despite this, even with the use of low number of samples, GMapping can still be computationally complex. Thus, there have also been other solutions that further improve RBPF SLAM, such as LaMa's particle filter SLAM approach [47]. This approach computes the proposal distribution by firstly sampling from a motion model and then refining the pose of the particle with scan matching. Moreover, it uses an adaptive resampling technique with smoothed likelihood in order to avoid particle depletion of good samples. Additionally, LaMa's RBPF SLAM takes advantage of the Rao-Blackwellized particle filters assumed independence between particles by parallelizing the scan matching process and concurrent map update procedures, allowing for multi-threaded use. Besides the computational efficiency, this approach also makes use of online data compression for an improved space efficiency [47, 45].

3.2.3 Topological SLAM

The topological representation to the SLAM problem seeks to create a graph-like description of the environment rather than a precise metric map. In the *topological* description, nodes can correspond to significant places which are easy to distinguish. Thus, arcs connecting the nodes correspond to sequences of actions that connect neighboring places. The motivation behind this approach was the belief that humans and animals do not produce accurate maps of the environments they are in. Typically, these methods are appropriate for navigation in simple environments, applying it in more complex and larger environments is harder.

There have been many proposals and studies making use of topological representation over the past decades [14, 52]. In more recent years, [5] tried to bridge the gap between feature and

topological representation with *Topomap*. *Topomap* is a framework capable of transforming a sparse feature-based map from a visual SLAM system into a three-dimensional topological map (figure 3.6).



(a) Visual SLAM map (b) 3D Topological map

Figure 3.6: Example of Topomap transformation from visual map to a topological one, adapted from [5]

3.2.4 Graph-based SLAM

Graph-based or network-based SLAM also tries to formulate the environment using a graph, whose nodes correspond to the poses of the robot at different points in time and whose edges represent spatial constraints relating two robot poses. The constraints consist in a probability distribution over the relative transformation between poses. With the graph built out of the sensor measurements, the system formerly tries to determine the most likely configuration of the poses given the edges of the graph [22].

One of the most popular approaches to Graph-based SLAM is the Real-Time Appearance-Based Mapping (RTAB-Map) method based on an incremental appearance-based loop closure detector that uses a memory management approach that only considers portions of the map [31, 32].

3.2.5 Semantic SLAM

The representation can also be in semantic map models. Unlike topological approaches, where mapping filters the metric information and only leverages place recognition to distinguish places, semantic mapping associates semantic concepts to the objects in the environment. Semantic mapping is still in the early stages of development, as such, depending on the level of semantic capabilities needed it can be very elaborate to implement, despite that, many studies around semantic mapping and semantic SLAM have been formed [2, 7, 61, 59, 41, 26].

Chapter 4

Sensors and features

For most mobile robots, knowing a set of parameters like radii and distance between the wheels, cheap sensors such as optical encoders can be used to compute the robots' odometry. The main problem with odometry information is that it is exposed to drift. The drift problem can come from two types of errors:

- *Systematic errors*- errors which are mainly due to imperfection in the structure design of the robot.

- *Non-systematic errors*- errors due to external causes, for instance, wheel slippage when the robot is moving on wet ground, human intervention, etc. This type of error is harder to deal due to its random nature.

Moreover, the use of more advanced *proprioceptive sensors* equally gyroscopes or accelerometers can aid in reducing the drift, but it can not completely eliminate it. The best way to correct it is through the use of *exteroceptive sensors*, which perform measurements corresponding to bodies external to the robot.

The methods used when applying SLAM can heavily depend on the hardware used. The possibilities when implementing SLAM are endless, a system can make use of just one to many types of sensors. However, the used sensors depend on the objectives of the system. Operating with various sensors can make the system complex to manage, therefore it should only be adopted if necessary. Early SLAM implementations mainly use sonar or lasers sensors, but nowadays, with the progress on computing power and computer vision algorithms, many systems are making use of visual cameras to apply SLAM [34, 62, 39, 37].

4.1 Sonar Sensors for SLAM

Sonar sensors are widely used in mobile robotics for being one of the most affordable sensors that can be used for SLAM [10]. Normally, sonars measure the distance between themselves and an object, based on the time difference between the emission of the sound wave and its return, after being reflected by an object. This method is adequately termed Time of Flight (TOF). The success of applying sonar measured data is dependent on the approach used to classify the data. These type of sensors are recommended mostly for underwater applications because of their affinity to how some aquatic beings communicate.

Besides being affordable, sonar sensors have the convenience of not being dependent on brightness conditions and being capable of high range precision, albeit they can have a very limited range sensing, specially for very small obstacles.

Even though they have their advantages, sonar sensors have problems that make considering a single sonar measurement alone for classifying the environment difficult. One of the main problems with sonars is the occurrence of weak echos, where it takes a long time for reaching the detection threshold, resulting in too long range estimates. Additionally, there is the possibility of false readings when the sound wave is reflected by more than one object before reaching the receiver. Furthermore, there are no guarantees that the received sound wave is not from other sources that emit identical sound waves. Also, the fire rate of the sensor is bottlenecked by the slow speed of sound (at least compared to the speed of light used in other sensors such as laser scanners).

Despite its problems, there are techniques that combine several readings from different sensors to extract simple features from the environment. Moreover, with favored techniques such as Triangulation Based Fusion (TBF) for extracting point features regarding the environment from sonar readings, SLAM with sonar sensors can be viable for more simple situations [62].

4.2 LIDAR Sensors for SLAM

Before the rise of visual sensors, LIDAR (Light detection and ranging) sensors were and still are one of the most used hardware to apply SLAM on mobile robots. Overall, LIDAR sensors can be divided into 2D and 3D LIDAR, depending on the number of LIDAR beams. Similarly to sonar sensors, TOF techniques are widely used. Most systems that use these kinds of sensors are reliable and use probabilistic methods for either 2D or 3D environments. Additionally, lately, the idea of using deep learning algorithms for feature detection, localization and recognition, and segmentation has been reviewed [27]. Despite being one of the most applied sensors, there are still some challenges when doing SLAM with them.

LIDAR has the advantage of being able to provide precise information fast with a large angle of view without being affected by night and light changes. Nonetheless, the technological threshold of LIDAR can be high making it not the most cost-effective method. Furthermore, low-texture environments and transparent objects can cause problems when using LIDAR sensors, making the

need for incorporating other complex methods to achieve viability. Also, deep neural networks are easily attacked by adversarial samples making deep learning LIDAR SLAM vulnerable [27].

In general, 3D LIDAR sensors are of high cost, however 2D LIDAR can be more affordable. Therefore, sensors like the RPLidar A3 can be good and widely available choices to apply viable SLAM methods on mobile robots.

4.2.1 RPLidar A3

The RPLidar A3 (figure 4.1) is a low cost, low profile 360 degree 2D laser scanner based on the laser triangulation ranging principle, developed by SLAMTEC. It is able to take up to 16000 samples of laser ranging per second with high rotation speeds. Furthermore, it can achieve a 2D 360-degree scan within a 25 meters range. SLAMTEC advertises a more stable performance when detecting objects in long distance, objects in white or black alternatively and objects under direct sunlight, which can be useful for outdoor use [30]. Moreover, A3 supports two modes alternatively [30]:

- *Enhanced mode*- uses the maximum ranging radius and sampling rate to realize an optimistic mapping performance in indoor environments.

- *Outdoor mode*- works with a more decent support against daylight interferences.

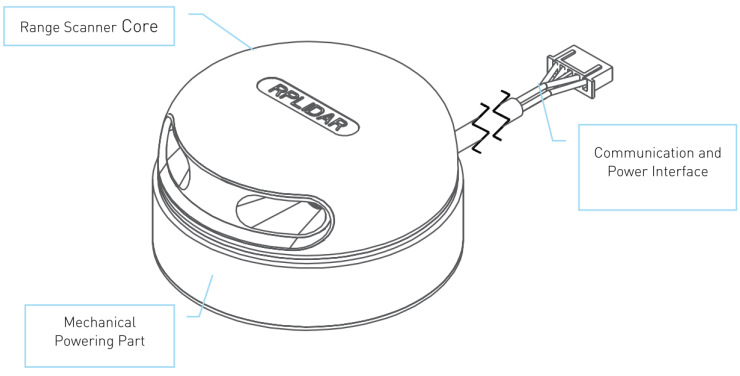


Figure 4.1: RPLidar A3 composition [30]

The A3 typically has a scan frequency of 10Hz, however the frequency can be freely adjusted from 5Hz to 20Hz. Withal, the sampling rate of 16kHz can only be achieved in 10Hz frequency, with an angular resolution of 0.225º. Additionally, SLAMTEC offers a fully fledged SDK that can help development process on systems made in ROS [53]. A more detailed look and comparison to other sensors can be seen in table 4.1.

4.3 Visual Sensors for SLAM

With the increasing computational power of CPUs and GPUs on the market, and the constant improvement in computer vision algorithms, the past decade has seen a rise of popularity on the development of visual SLAM. Unlike some LIDAR sensors, cameras are cheaper and smaller, making it now possible to apply SLAM on micro PCs or even smartphones. When speaking of Visual SLAM, the most common sensors used are cameras. Cameras can be mostly divided into monocular cameras, stereo cameras, RBG-D cameras, event cameras, etc [27].

Visual sensors are cheap and passive, unlike laser sensors, they do not need to emit light. Also, from an image, it is possible to get way more information than a laser normally can. However, visuals sensors can struggle on low-texture and dynamic environments, and, opposite to LIDAR sensors, they do not perform as well in low light conditions and are not as precise and have lower maximum range. Nonetheless, most visual sensors used on SLAM systems do not rely only on cameras but also can make use of other sensors, such as infrared sensors or inertial measurement units (IMU). Additionally, the possibility of making intelligent systems able to perceive environments like humans is more viable with the use of visual sensors [27].

4.3.1 Mynt Eye D1000-120/50

The Mynt Eye D1000 (figure 4.2) is a stereo depth camera, on which the depth data is computed within the camera, so there is no need to process the stereo images to obtain the depth data on an external system. It is capable of providing a max quality stereo resolution of 2560×720 at 60 FPS and, consequently, a max depth map resolution of 1280×720. Additionally, the Mynt Eye makes use of other sensors like IMU and infrared sensors. The IR module helps the sensor to work on different lighting conditions, making it more viable in dark or outdoor environments. Furthermore, the IMU is used to accommodate the odometry drift problem.

Figure 4.2: Mynt Eye D1000 [38]

The Mynt D1000 presents two different models that trade between detection range and field of view (FOV). On one side, the 120 model has a lower range with a maximum of 10 meters, but it presents a higher horizontal FOV of 105ª. On the other side, the 50 model has a maximum range of 15 meters, yet it has a lower horizontal FOV of 64ª [38]. A more detailed look and comparison to other sensors can be seen in table 4.1.

4.3.2 StereoLabs ZED 2

The ZED 2 (figure 4.3), developed by StereoLabs, is a stereo camera that provides high definition 3D video and neural depth perception of the environment. Unlike the Mynt Eye D1000 (section 4.3.1), the depth data is not computed within the camera, however , the Zed SDK provides a straightforward way to calibrate and obtain depth data from the sensor [56].

Furthermore, by mixing AI and 3D, the ZED 2 can detect and track objects with spatial context up to a 40 meters distance. Additionally, this stereo camera makes use of neural networks to reproduce human vision, adding a new level of perception to stereo cameras. The ZED 2 also features other sensors like an IMU, barometer and a magnetometer for gathering real time synchronized inertial, elevation and magnetic field data along image and depth [13].

Figure 4.3: StereoLabs ZED 2 [55]

The ZED 2 easily provides low level access to the device, providing control over all the camera parameters such as exposure, gain, sharpness, and others. Also, it has a higher depth range of up to 20 meters with a horizontal field of view of 110º. A more detailed look and comparison to other sensors can be seen in table 4.1.

4.3.3 Intel RealSense LiDAR Camera L515

The Intel RealSense L515 (figure 4.4) is a solid state LiDAR depth camera that enables highly accurate depth sensing in a small form factor, designed mostly for indoor environments. The L515 uses a proprietary MEMS mirror scanning technology, allowing for a more power efficient alternative to other time of flight technologies. Moreover, like the Mynt Eye D1000, all depth data is calculated within the device [28].

Figure 4.4: Intel RealSense LiDAR Camera L515 [29]

Furthermore, besides the RGB camera the IR laser, the appliance makes use of an IMU sensor to aid in odometry awareness. To achieve excellent performance of the camera, the L515 can operate in three preset modes [28]:

- *Max range*- useful when there is no ambient light in the scene. The laser power is set to maximum as well as the receiver gain, optimizing the depth quality in indoor applications.

- *No Ambient*- similar to the *Max range* preset, but used for when there is no sunlight in the environment. The power of the laser is lower in this mode to avoid false depth on objects that are distanced outside the ambiguity range.

- *Low Ambient*- used for where there is a low amount of ambient sunlight present. The laser power is set to maximum, but the receiver gain is reduced in order to avoid saturation due to the light. Useful for detecting close objects within 50 centimeters.

The L515 trades depth data maximum distance and horizontal field of view for consistent accuracy. In comparison to the other sensors mentioned, it has a lower maximum depth distance of 9 meters and a horizontal field of view of 70º.

4.4 Overview

Overall, there is a variety of sensors that can be used to apply SLAM in a mobile robot. Over the years, with the evolution of technology and computational power, visual based sensors seem to be the most favorable type of sensors since they can provide enough accurate 3D depth data for use in the most diverse type of scenes. Also, a lot of depth sensors try to bridge the gap between cameras and LiDAR only sensors, by making use of IR light sensors to achieve better depth accuracy. Nonetheless, these visual sensors can not still achieve the accuracy, field of view and operating distance of some laser scanners available. So, when developing a mobile robot that needs to apply SLAM, a thorough research of its necessities and use cases must be made in order to conclude the best sensor(s) to include. The table 4.1 shows a more detailed look and comparison between the sensors mentioned in the previous sections.

Table 4.1: Details and comparisons of various sensors available in the market

Sensor	RPLidar A3 (Enhanced mode)	RPLidar A3 (Outdoor mode)	Mynt Eye D1000-120/50	ZED 2	RealSense L515
Type	2D Lidar scanner	2D Lidar scanner	Stereo camera	Stereo camera	LiDAR camera
Dimensions	76 x 41 mm (D x H)	76 x 41 mm (D x H)	165 x 30 x 30 mm (L x H x W)	175 x 30 x 33 mm (L x H x W)	61 x 26 mm (D x H)
Depth/scan range	White object: 0.2-25m Black object: 0.2-10m	White object: 0.2-20m Black object: TBD	D1000-120: 0.3 - 10m D1000-50: 0.5 - 15m	0.3-20 m	0.25-9 m
Sample Rate	16 kHz	10 kHz or 16kHz	NA	NA	NA
Scan rate	5 Hz - 15 Hz	5 Hz - 15Hz	NA	NA	NA
Angular Resolution	0.225º	0.225º or 0.36º	NA	NA	NA
Camera resolution	NA	NA	2 x 1280 x 720 @ 60 FPS	2 x 2208 x 1242 @ 15 FPS 2 x 1920 x 1080 @ 15/30 FPS 2 x 1280 x 720 @ 15/30/60 FPS 2 x 672 x 376 @ 15/30/60/100 FPS	1920 x 1080 @ 30 FPS
Depth resolution	NA	NA	1280 x 720 @ 60 FPS	Native video resolution	1024 x 768 @ 30 FPS
Field of view (FOV)	360 degrees planar scan	360 degrees planar scan	D1000-120: 105º x 58º x 121º (H x V x D) D1000-50: 64º x 38º x 70º (H x V x D)	110º x 70º x 120ª (H x V x D)	70º x 55º (H x V)
Accuracy	TBD	TBD	TBD	<1% up to 3m <5% up to 15m	<5mm up to 1m <9mm up to 9m
Indoor use	Yes	Yes	Yes	Yes	Yes
Outdoor use	No	Yes	Yes	Yes	No
SDK Support	Yes	Yes	Yes	Yes	Yes
Other sensors	NA	NA	Stereo camera IMU (500 Hz) IR light	Stereo camera IMU (400 Hz) Magnetometer (50 Hz) Barometer (25 Hz)	RGB camera IMU (400 Hz) IR laser IR photodiode
Price	599€	599€	D1000-120: 389 € D1000-50: 299 €	449€	350€

Chapter 5

Simulation environment for an Intelligent Wheelchair

Whenever developing a mobile robot, it is hard to practically test the robot in different situations and configurations without putting the robot at risk of failure or possible damages. The same goes for an intelligent wheelchair, as it can be exposed to various different types of environments with distant degrees of conditions. It is also not feasible to test various sensors and SLAM methods arrangements in order to decide the best setup for most use conditions. Thus, the application of a simulation environment that runs a resembled wheelchair capable of doing autonomous navigation and SLAM with different sets of sensor configurations, aids in exposing the best setup for the IntellWheels 2.0 chair and its possible shortcomings. This chapter aims to explain the developed simulation environment from the application of the wheelchair in a simulated world to the point of the wheelchair being capable of navigating by itself while doing SLAM.

The main wheelchair system capable of doing simultaneous localization and mapping in conjuction with path planning was developed in ROS[1]. The robot operating system (ROS) is a modular framework for writing robot software. It provides a collection of tools, libraries and conventions that plan to simplify the task of developing complex and robust robot behavior. Furthermore, ROS provides seamless integration with other projects, one of which is Gazebo[2]. Gazebo is an open-source 3D robotics simulator that offers the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. It provides robust physics engines and convenient programmatic and graphical interfaces. In conclusion, the IntellWheels wheelchair and respective sensors, either camera or laser scanners, are completely running on Gazebo, while the main system runs in ROS. This allows for easy transition between simulated testing and deployment to the physical wheelchair with minimum effort.

The wheelchair ROS project is structured around various packages, at which, each package has a role to play in working with different aspects needed in the simulation. However, the project

[1] https://www.ros.org/
[2] http://gazebosim.org/

can be summarized in three main packages that make use of other packages in order to make the simulation run seamlessly:

- **wheelchair_world**- is the core package of the project. It is responsible for launching all necessary package nodes needed to spawn the wheelchair in any given Gazebo world. Furthermore, it handles all different types of data received by the distinct sensors simulated accordingly, in order to be used with the SLAM and navigation packages. This package also stores all models, either wheelchair or sensors, worlds and configurations used in the simulation.

- **wheelchair_slam**- contains all the configurable package nodes responsible for applying SLAM methods to the wheelchair according to the models launched in the wheelchair_-world package.

- **wheelchair_navigation**- it launches the nodes responsible for handling the path planning aspect of the wheelchair.

Overall, the project can be divided in three different segments. The simulated segment where the wheelchair model and respective simulated sensors are handled. The SLAM segment, where the wheelchair does simultaneous mapping and localization of its surroundings with the use of the simulated sensors. Finally, the navigation segment where the wheelchair applies path planning using the map and localization calculated by the SLAM segment. The first segment is essential for the other two segments to work properly. If the sensors simulated are not handled properly or some errors occur, consequently the system will not be able to do SLAM and provide a map of the surroundings. If there is no map and localization of the surroundings the system can not do path planning. Therefore, it is essential to make sure each segment is correctly configured in order to work properly in most case scenarios.

5.1 Wheelchair Modeling

The first important component to include in the simulation is the wheelchair itself. The developed work makes use of a previous modeled IntellWheels wheelchair, implemented in Gazebo and ROS by [16]. Moreover, the modeled wheelchair is based on the real wheelchair, and it has all the physics features to make it as close to the real model as possible. However, visually, the model is not a perfect recreation of the real chair, but it provides a close representation and footprint of it (as demonstrated in figure 5.1 and 5.2). Nonetheless, it is necessary to take in account the skeleton of the real wheelchair, in order to determine the spots on where sensors could realistically be placed.

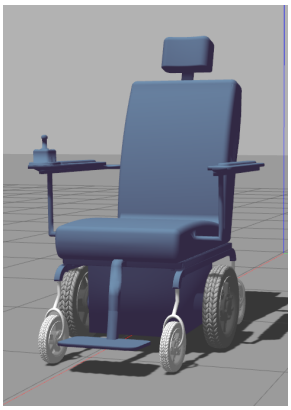


Figure 5.1: The wheelchair Gazebo model.

Figure 5.2: The real wheelchair model [16]

Gazebo uses the SDFormat to interpret and implement a robot in its engine. SDF is a format in XML that allows for an easy description of a robot within a world. In SDF, a robot and consequently the wheelchair can be represented around three main components:

- *Links*- are the skeleton of the robot. They are composed by visual elements, such as meshes, collision boxes and inertial factors.
- *Joints*- represent a parent-child relationship between links. It is useful for defining a connection between two links and setting how a link should move (or not) relative to another.
- *Plugins*- adds external elements that are needed for the robot operate to as desired. This can be a differential driver plugin to move the robot, or a sensor to gather information of its surroundings.

The implemented wheelchair model was described in the *.sdf* format. However, ROS does not support the SDFormat in order to get a description of the robot. Therefore, to be able to implement the wheelchair in ROS, most specifically in its 3D visualizer RVIZ, a URDF format description of the wheelchair was made. The Unified Robot Description Format, although a little different and more restrictive than the SDFormat, follows the same principles of links, joints and plugins, making the manual transformation from *.sdf* to *.urdf* seamless. The main difference between these two formats is that URDF can only specify the robot while SDF can also specify the world for the robot to live in. Automatic conversions from URDF format to SDF format are possible, as such, with some small tweaks, Gazebo can import robot models in the URDF format. Despite that, URDF is less flexible and scalable than SDF, making it harder for executing changes such as easily adding and removing new sensors. Therefore, SDF models of the wheelchair with different set of sensors were still used by Gazebo, while a simple URDF description of the main skeleton of the wheelchair was used by ROS to work with the *TF* tree state of the robot. Having a URDF version of the wheelchair is important for ROS to know the state of the wheelchair, and consequently for the other packages used, to work properly and seamlessly. Furthermore, it is essential to have the wheelchair represented in RVIZ so that the user can watch and order the wheelchair to navigate in the resulted map when doing SLAM (figure 5.3).

Figure 5.3: Wheelchair model in RVIZ.

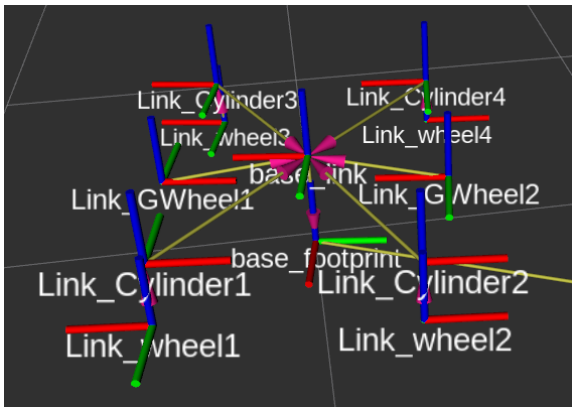
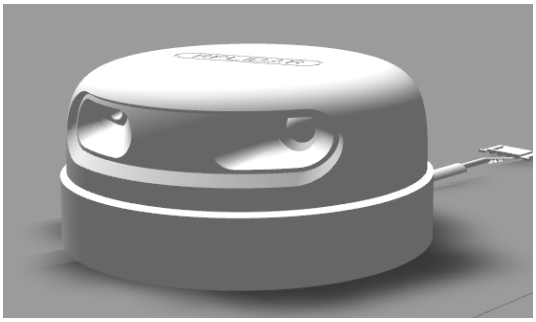


Figure 5.4: The state of the robot (links and joints) in RVIZ.

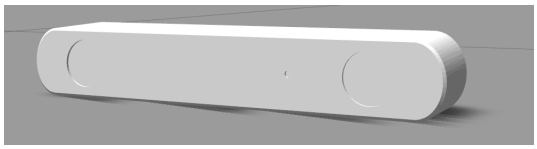
Additionally, [16] developed a differential drive controller for the real wheelchair which receives data from a multimodal interface, processes it and generates and sends the control commands to the real chair. The controller architecture follows ROS nodes, as such this controller can be used to navigate the wheelchair in simulation with the computer keyboard. This controller was adopted to assure and test that the SLAM methods used with different sensors were working properly.

5.2 Sensors Modeling

To do SLAM, the simulated wheelchair needs to be able to perceive its surroundings. For that, sensors need to be simulated in conjunction with the wheelchair. Gazebo has some predefined sensor plugins that can aid in simulating real life ones. Therefore, SDF models of some sensors mentioned in chapter 4 were created (see figure 5.5). Each sensor model has had to them a plugin linked. The plugin used for each model was the most appropriate to resemble its real life counterpart.



(a) RPLidar



(b) ZED2

Figure 5.5: Visualization models of some sensors in Gazebo

Overall, Gazebo has two types of sensors plugins that are favorable to implement SLAM: laser plugins and camera plugins.

5.2.1 Laser plugins

Gazebo laser plugins like the *libgazebo_ros_gpu_laser* are capable of simulating laser range sensors such as the RPLidar A3 (section 4.2.1) by publishing ROS compatible LaserScan messages to specified ROS topics. The plugin allows for setting some parameters that would help test different set of 2D like laser scanners:

- *update rate*- the frequency in Hz of the broadcast LaserScan messages.
- *samples*- the sample rate of the sensor.
- *resolution*- resolution of the laser scan.
- *min/max angle*- the angle view of the laser scan in radians.
- *range*- minimum and maximum range done by the laser.

The sensor sends a LaserScan message to a set topic (for example */scan*) at a specified rate. Despite that, the URDF description of the wheelchair does not have the model of the sensor, as such, a link and joint between the wheelchair and the sensor model must be manually added to the TF tree state of the chair, so that, RVIZ can seamlessly display the laser scan data from the wheelchair (figure 5.6).

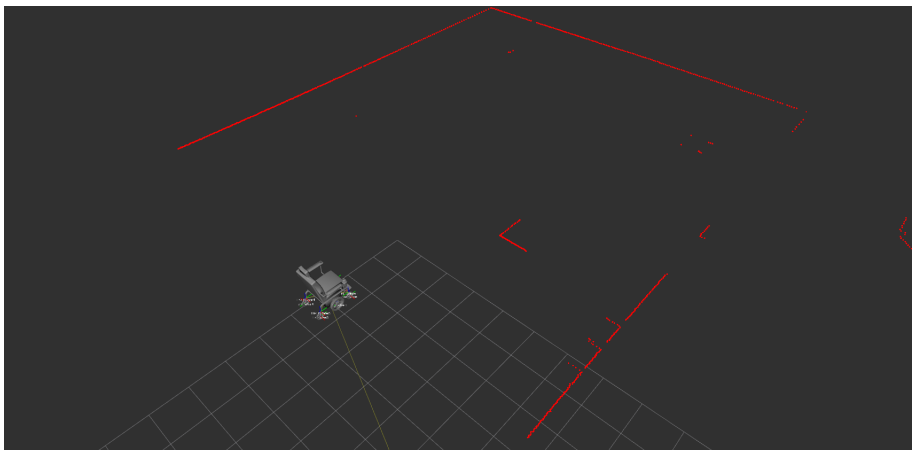


Figure 5.6: The resulted laser scan of a laser plugin in RVIZ.

5.2.2 Camera plugins

Gazebo provides a several camera like plugins. The most advantageous for SLAM testing is the depth camera plugin, which besides simulating the camera itself it produces a 3D point cloud of what it captures. The images and point cloud captured by the plugin depth camera are

appropriately published to a topic with ROS supported message types. Overall, camera plugins allows for a wide variety of parameters that would help test a set of different 3D like camera sensors:

- *horizontal fov*- sets the horizontal field of view captured by the camera, in radians.
- *image width/height*- it allows to set the resolution captured by the camera. It also lets you choose the image format (normally R8G8B8).
- *clip near/far*- the range the camera is allowed to capture from the environment.
- *update rate*- it sets the frequency the images (and pointcloud) are published, in Hz.
- *pointcloud cutoff* - it allows to set the minimum and maximum range of the 3D point cloud. It must be within the clip parameter range (only in depth camera plugin).
- *baseline*- it sets the baseline distance between two camera lenses (when applicable).
- *focal length*- sets the focal length of the camera.
- *distortion*- allows adding distortion to the image captured.

Besides using the depth camera sensor, it is feasible to try to depict a more direct representation of a stereo camera by making use of the multi camera plugin. However, with this plugin, only the images of the stereo cameras are published, external tools for processing are needed to obtain a 3D point cloud of the captured surroundings. ROS provides *stereo_image_proc*, a package that allows to process two images of a stereo camera into a disparity image, and consequently a point cloud.

Like the laser sensor models, a link and joint between the camera models need to be manually added for the ROS 3D visualizer be able to smoothly display the broadcasted data from the sensors.

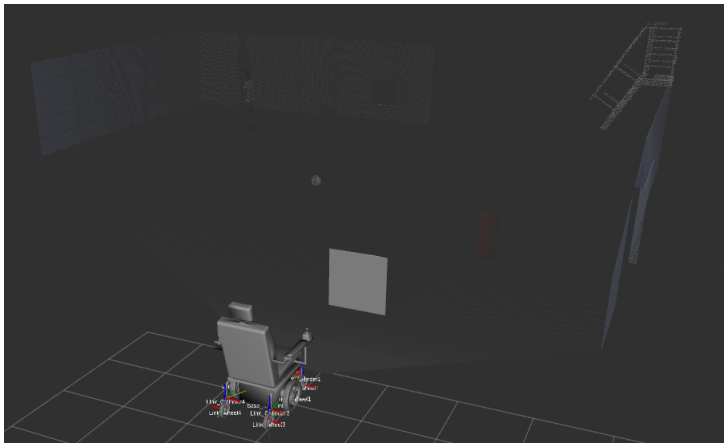


Figure 5.7: The resulted 3D point cloud of depth camera plugin in RVIZ.

5.2.3 Sensor data handling

The wheelchair adopted 2D grid-based methods of SLAM. Thus, most of the applied SLAM packages handle laser-based data. For laser plugins, the data published is already of laser type, so it can automatically be used by the SLAM packages. However, camera like plugins publish 3D point clouds. In order to make the camera sensors work with the adopted SLAM packages, the point cloud data had to be processed and converted to laser data type. For that, ROS provides the *pointcloud_to_laserscan* package that allows to smoothly convert point cloud messages to laser scan messages (figure 5.8).

Figure 5.8: Respective 3D point cloud and its laser scan conversion.

When changing to 2D laser data, a lot of information can be lost from the point cloud. Nonetheless, the *pointcloud_to_laserscan* provides parameters to set the minimum and maximum height to sample in the point cloud, allowing for the laser scan of an obstacle to be the maximum width the object takes in the environment.

Furthermore, the use of multiple sensors simultaneously was tested. When making use of multiple sensors, each sensor publishes its data in different ROS topics. For example, if the wheelchair has two laser sensors attached to it one laser will publish the laser scan data to */front_laser/scan*, while the other to the topic */back_laser/scan*. The SLAM packages take into account only data from one topic, therefore, the data from the multiple sensors need to be properly merged and published to a new topic. For merging data from the laser plugins, the *ira_laser_tools* package developed by [3] was applied. Meanwhile, merging two 3D point clouds into one was possible with the use of the Point Cloud Library (PCL).

5.3 Simultaneous Localization and Mapping

When it comes to SLAM, there are a variety of different methods and ROS packages available. Still, for that, the implemented simulation environment makes use of the *Localization and Mapping* (LaMa) library. LaMa is a software library for robotic localization and mapping developed at the Intelligent Robotics and Systems (IRIS) of the University of Aveiro. It includes an efficient

framework for mapping with 3D volumetric grids [45], a localization algorithm based on scan matching and a continuous likelihood field [44], and, most importantly, two SLAM solutions (an Online SLAM and a Particle Filter SLAM) [46, 47].

The **LaMa Online SLAM** method provides a fast and efficient scan matching method supported by a dynamic likelihood field based on an incremental Euclidean distance grid. Additionally, it presents the scan matching method as a non-linear least square problem that can be solved by popular algorithms such as Guass-Newton and Levenberg-Marquardt [46]. Moreover, it is a low computational solution that can be run in real time on a low-spec system. It is mostly ideal for outdoor environments where there is a considerable lack of close loops.

The **LaMa Particle Filter SLAM** method is an extension of the Online method. It gives a solution based on Rao-Blackwellized particle filters with the support of a similar fast scanning method to the Online SLAM approach. This method contests the main drawback of high complexity in RPBF SLAM by taking advantage of the independence between particles to provide a multi threading approach. Additionally, with the aid of online data compression algorithms used by the mapping approach [45], LaMa's Particle Filter SLAM is not only capable of achieving better computational efficiency than others Rao-Blackwellized solutions like GMapping [25], but also better memory allocation as well. This method provides superior map (quality) than the Online solution, more specifically in indoor environments where loop closures are more frequent [47].

The project makes use of the ROS LaMa packages that easily allow the integration of the SLAM solutions with the simulated wheelchair. Each package subscribes to two important topics, a topic related to the TF state of the world and another one where the laser data is properly published. From those topics the packages return a grid-based map and the location of the wheelchair within that map that can be visualized in RVIZ (figure 5.9).

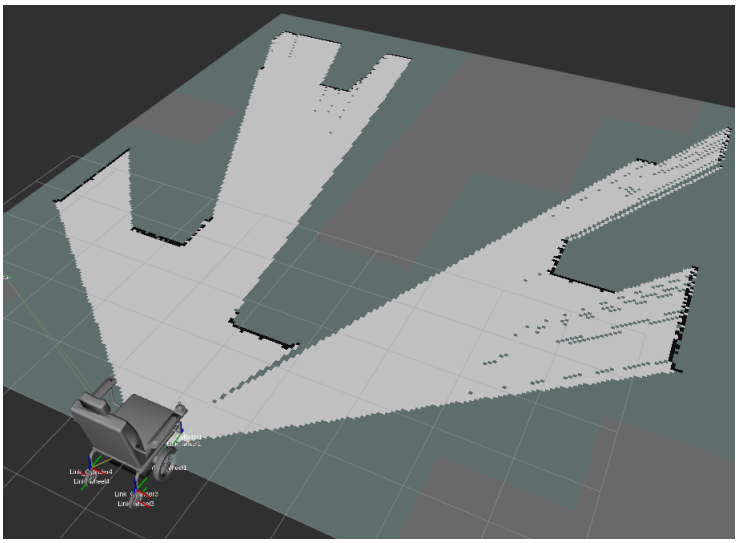


Figure 5.9: The result map and location given by the LaMa SLAM method.

As mentioned before, the RPBF SLAM method provides better results when dealing with environments where loops closures are frequent. Although it is probable that, when a wheelchair

user goes outside there are lower chances of loop closure, when it comes to moving around the house or other indoor environments, loop closures are more recurrent. Therefore, most of the testing and use cases on the simulation were run strictly on LaMas Particle Filter node, with the parameters tuned to obtain the best results [47].

5.4 Path Planning

With the map and localization of the wheelchair obtained with SLAM, the wheelchair can now be able to autonomously navigate through the perceived surroundings (figuere 5.13). To do that, the *move_base* node of the ROS 2D Navigation Stack was used. The *move_base* package provides an implementation of tasks, that, when given a goal position in the world, will try to command the wheelchair to reach it. This package adopts a global and a local planner to accomplish its desired goal. Each planner makes use of a cost map, that is maintained by *move_base*, as to determine the best path to take to the desired goal. Furthermore, the *move_base* package is modular as it supports any given planner that follows a specific interface of the navigation stack. A high-level diagram of the *move_base* node and its interaction with different components can be seen in figure 5.10.

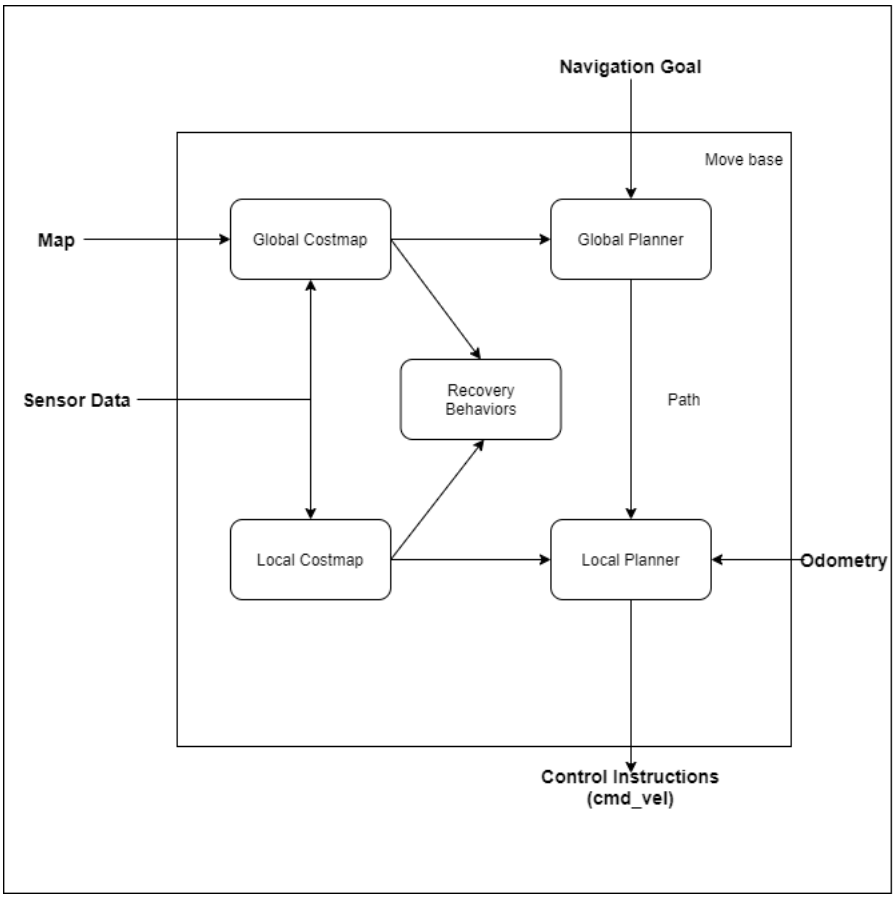


Figure 5.10: ROS move base components, adapted from [42].

From the map obtained by using SLAM and the sensors' data, the move_base package creates two 2D grid cost maps. A global cost map that represents a vast proportion of the environment, and a local cost map that embodies the close surroundings of the robot. When a user, or other external system, sends a goal position on the map, a global planner creates a path plan by analyzing the global cost map. Afterwards, a local planner evaluates the path created by the global planner, and, given the current wheelchair position and local cost map, tries to send commands (mainly in the *cmv_vel* topic) that makes the wheelchair follow the previously calculated path as close as possible as to not collide with obstacles. Furthermore, the *move_base* packages allows the wheelchair to perform any recovery behaviors given whenever the wheelchair fails to achieve the desired goal.

5.4.1 Cost maps

A cost map is a 2D occupancy grid that maintains information of where the wheelchair can and can not navigate through. Normally, the cost map uses sensor data and/or information from a map given from an external source to store and update information about obstacles in the environment. Normally, the sensor data is used to dynamically update new or removed obstacles from a previously scanned static map. Since the map obtained from the wheelchair is constantly being updated by the SLAM method, which already makes use of the sensors' data, the use of sensor data to construct the cost maps is more of a complement than a necessity.

Each cell in the cost map can have any value between 0 and 255 to represent the cost value of passing through that cell. Nonetheless, despite the possible range of values, each cell is normally categorized in three types. A free cell, an occupied cell and an unknown cell. The cost of the cells is used by planners to calculate the shortest path to the desired goal, as well as to determine if the wheelchair footprint has possibly collided or not.

The *costmap_2d* package of ROS provide a lot of parameters that can be tuned in order to obtain the most desirable cost maps. Furthermore, cost maps can be structured in different layers, in which each layer adds new configurations and parameters to the cost map. There are three main layers that the cost map package uses [42]:

- *Static Map Layer*- is the layer responsible for receiving a map from an external source. In this case would be the map obtained while doing SLAM. The map given does not need to be static, it can be dynamically changed and will still be used.
- *Obstacle Layer*- uses sensor data to track and add obstacles to the cost map.
- *Inflation Layer*- allows inflating obstacles in the cost map, by propagating cost values around them.

The *costmap_2d* package does not necessarily need to use all three layers. However, either the static layer or the obstacle layer must be configured as to assure that there is data that can be used to create the cost map at all.

Moreover, the inflation layer is needed to create cost maps that represent the configuration space of the wheelchair, helping the planners to calculate paths that better avoid obstacles. The

inflation layer allows for propagating cost values out from occupied cells that decrease with distance (figure 5.11). This allows for a more variety of cost values that help planners understand how close the wheelchair is to an obstacle.

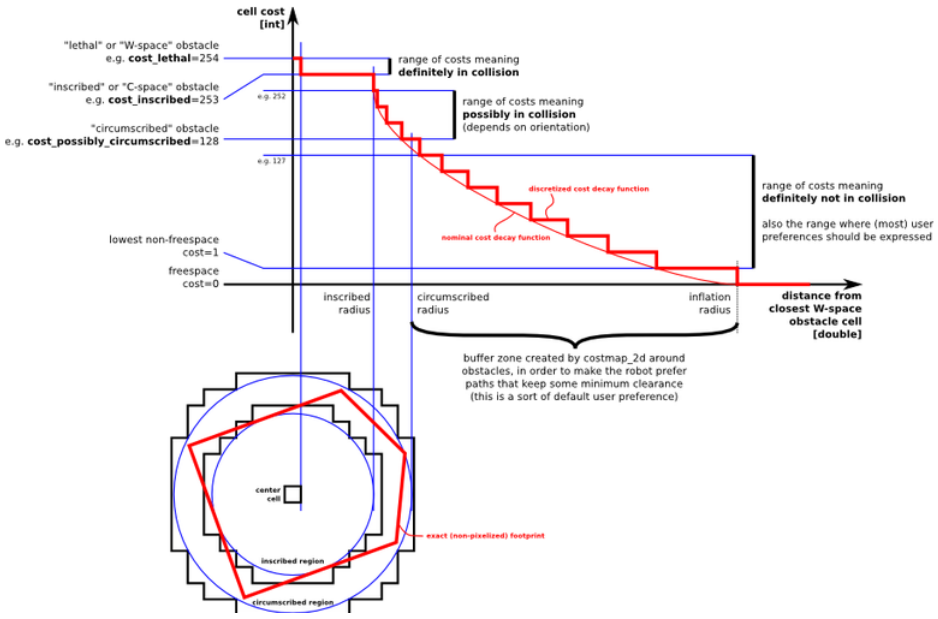


Figure 5.11: ROS cost map cost given obstacle inflation [42].

According to [42], the application of inflation in each obstacle defines five specific type of cost cells (each with a specific range of costs) instead of the standard three mentioned before:

- *Lethal/occupied cell*- one of the standard cells. Means that there is an actual obstacle in the cell.

- *Inscribed cell*- specifies a cell that is less than the wheelchair inscribed footprint away from an actual obstacle. If the center of the wheelchair is in this cell, a collision is detected.

- *Possibly circumscribed cell*- similar to the *inscribed cell*, but the wheelchair's footprint represent a cutoff distance from the obstacle. The collision or not depends on the orientation of the wheelchair.

- *Free cell*- represents a completely free cell of cost 0.

- *Unknown cell*- represents a cell which there is no information about it.

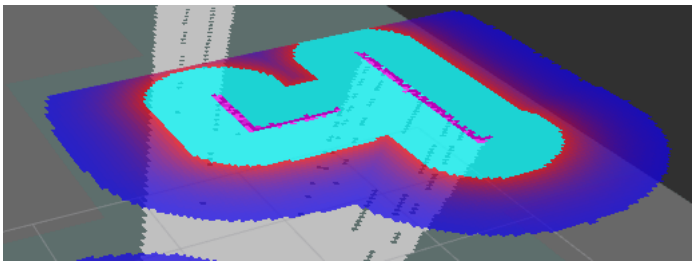


Figure 5.12: Example of cost map inflation shown in RVIZ.

The *move_base* package makes use of two cost maps. A global cost map, and a local one. Each cost map is used by the correspondent planner and can be configured without any restrictions. For the simulation environment created, the global cost map uses the three main layers, while the local cost map, since it is small, does not take into account an external map. Different configurations of the inflation parameters were tested as to assure the best cost maps that would allow the planners to achieve the most desired paths (more on Chapter 6).

5.4.2 Navigation Planners

Planners are used for calculating the best paths to a specific goal given the cost values of each cell of the used cost map. When implementing autonomous navigation with the Navigation Stack, there are two type of planners. The global planner and the local planner.

5.4.2.1 Global planner

The global planner has the job of creating the specific plan from the origin of the wheelchair to the desired goal. The planner applies a shortest path algorithm like Dijkstra or A* on the given cost map to find the minimum cost plan from a start point to the end goal. ROS provides packages for global planning such as *navfn* and *global_planner*. The *global_planner* is an improved version of *navfn* that allows to use either Dijkstra or A*, and change the cost values that are used in conjunction with the cost map cell values [42].

5.4.2.2 Local Planner

Given a plan calculated by a global planner and a cost map, the local planner is responsible for producing velocity commands to send to the wheelchair to make it follow the path as close as possible. ROS provides two local planners, the *base_local_planner* and the *dwa_local_planner*. The *base_local_planner* provides implementations of the Trajectory Rollout [20] and Dynamic Window [18] approaches to local navigation on a plane for both holonomic and non-holonomic robots. While the *dwa_local_planner* only provides implementation of the Dynamic Window approach for local navigation, but with more configuration capabilities compared to the *base_loca_planner*, for holonomic or pseudo-holonomic robots.

Local planners make use of a cost map and a precalculated path to produce a kinematic trajectory for the wheelchair to follow. Additionally, the planners create, locally around the wheelchair,

a value function which encodes the costs of traversing through the grid cells. It is the job of the planner, given the value function, to determine the most appropriate velocity commands to send to the wheelchair.

Thus, the main idea behind the local planners behaviors is as follows [42]:

1. Sample the wheelchair's control space (dx, dy, dtheta).

2. For each sample, perform a forward simulation from the wheelchair's current state to predict what would happen if the sampled velocity were applied for some time.

3. Score and evaluate each trajectory obtained from the forward simulation, using a metric that incorporates components like proximity to obstacles, proximity to the goal, proximity to the global path and speed. Also, impossible trajectories that collide with obstacles are discarded.

4. Select the trajectory with the highest score and publish the correspondent velocity commands to the wheelchair.

5. Repeat until the wheelchair achieves the end goal of the global path.

The main difference between the two approaches aforementioned, is how they handle the wheelchair's control space. The Trajectory Rollout approach samples from the achievable velocities over the entire forward simulation period, while the Dynamic Window approach only samples the velocities for one single simulation step [42].

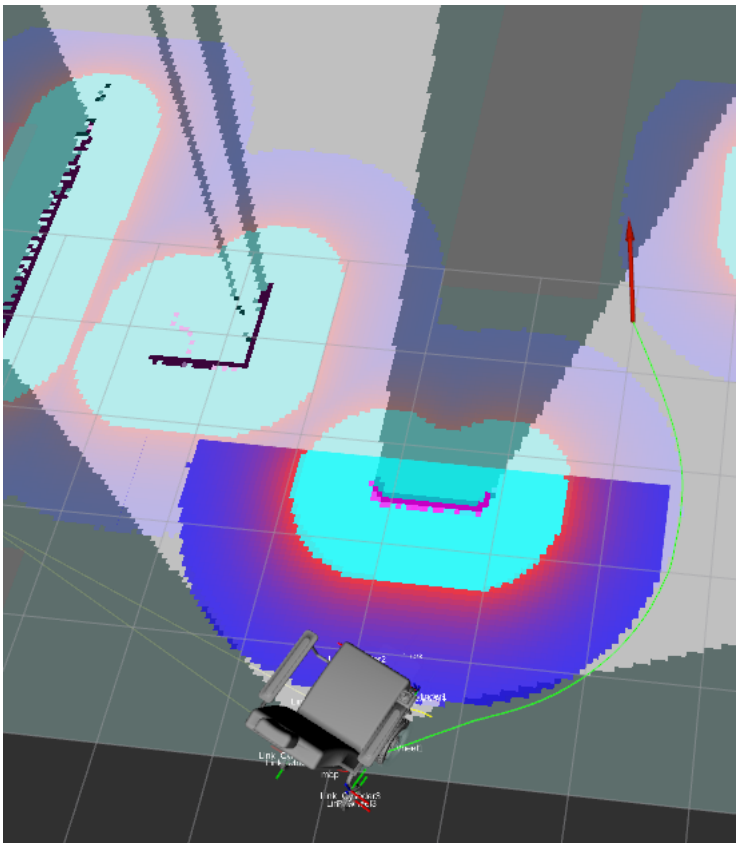


Figure 5.13: Wheelchair using the move base package to navigate through the environment. The green line represents the path obtained by the global planner, and the yellow line the path that the wheelchair will follow from the local planner.

5.4.3 Recovery Behaviors

Whenever the wheelchair is perceived as stuck or that it can not obtain a path to the desired location, the *move_base* package allows for the implementation of recovery behaviors that may try to fix the situation instead of just quitting in getting to the objective. The *move_base* package can make use of numerous amount of behaviors that are implemented in a specific order. The wheelchair tries to apply the first recovery behavior on the list. If the problem is fixed, the wheelchair continues the navigation towards the goal. Otherwise, the wheelchair will try the next behavior on the list. When all behaviors have been unsuccessfully executed, the wheelchair gives up in trying to achieve the end point.

By default, the *move_base* package applies two recovery behaviors. The first behavior attempts to clear out space in the cost maps by reverting to the original map obtained by the cost map static layer, outside a given radius away from the robot. Since the map used by the wheelchair is not static, this behavior has little to no impact in resolving the problem. The second behavior clears out space in the cost maps by rotating the robot 360 degrees, if allowed. Other behaviors can be manually created and added to the navigation stack if needed.

Chapter 6

Experiments and results

Given the simulation environment properly working with the wheelchair doing SLAM and autonomously moving through the map, it is necessary to test different configurations as to obtain the best results in both indoor and outdoor environments. This chapter will focus on demonstrating the different configurations tested on the wheelchair and the possible shortcomings of the system. Furthermore, this chapter will be divided in three main sections, that are not strictly independent.

The first section will focus on the various configurations of the sensors. More precisely, it will state the different possible specifications and types of sensors used in order to test the system.

The second will focus on the testing of the previous mentioned sensor configurations while performing the SLAM method. This allows for an understanding of which setup gives the best map and localization results, in both vast or restrictive environments.

Finally, the third section will focus on the path planning testing of the wheelchair, by evaluating mostly different configurations of cost maps that provides for better path results.

6.1 Sensors

The core components of the wheelchair are the sensors it takes advantage of, without them, SLAM and navigation are not possible. When building the wheelchair, it is crucial to test different configurations of either different types of sensors or sensors with various specifications. Additionally, the position of the sensors within the wheelchair is important as to understand spots where the sensor could realistically be placed and at same time obtain the most valuable amount of data.

6.1.1 Sensor Placement

The two types of sensors used for testing were mainly LIDAR type and visual camera type. Commonly, laser sensors have smaller footprints than visual cameras that are substantially wider, specially stereo cameras. As such, it was necessary to analyze where each type of sensor could be placed on the wheelchair. Furthermore, as previously mentioned, the model of the wheelchair used in Gazebo is not a perfect recreation of the original. Therefore, the sensors spots tested were in accordance with the real skeleton of the wheelchair .

6.1.1.1 Camera models

The camera sensors simulated used exact models of real cameras such as the ZED 2 (4.3.2) and an Intel RealSense Camera. These models present dimensions that are pretty standard for most stereo or depth cameras available in the market. For the front of the wheelchair it was tested two possible positions.

The first position attempts to place the camera model in the main chassis of the chair, as depicted in figure 6.1. The main problem of this spot is that the footrest and the wheel supports can substantially obstruct the field of view of the sensor. To minimize the obstructions, the camera could receive a little offset to the left or to the right as shown. Despite that, offsetting the camera would only be viable for smaller camera sensors such as the one shown in figure 6.1, or the L515 (4.3.3). For wider stereo cameras like the ZED 2 this spot would not be feasible. Moreover, leg rests of the real chair would possibly need to be removed for sensors to be placed in.

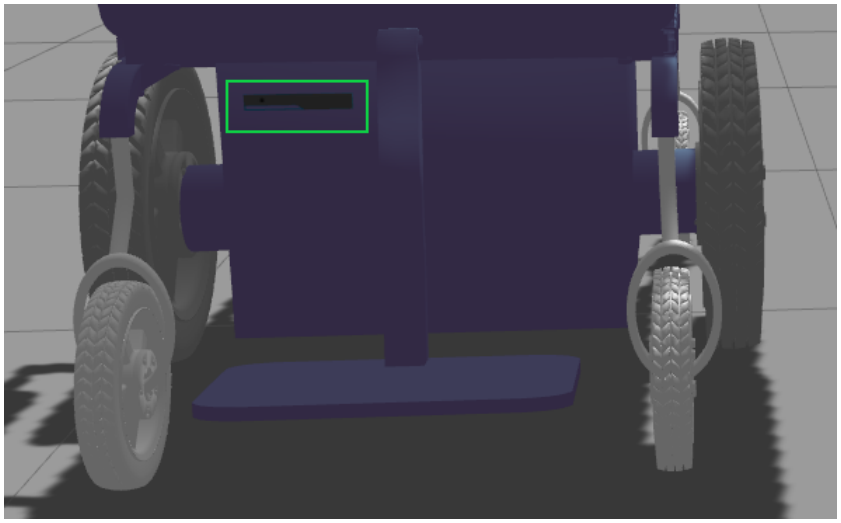


Figure 6.1: Example of the sensor in the main chassis of the wheelchair.

The other possible option would be to place the sensor directly under the footrest, as demonstrated in figure 6.2. This is a better position, because, opposite to other spot, there are no obstructions to the camera. Furthermore, it easily allows for the application of wider cameras. Nevertheless, the real wheelchair has the possibility of height adjustment, therefore, the sensor could move on the Z axis. Normally, small changes in height will not have any effect on the results obtained from the camera, since most of them have high enough vertical field of view. Additionally, since the map obtained is in 2D the height of the sensor is not a main issue. If this caused problems, the system would probably have to make use of the IMUs frequently built within most visual sensors to compensate.

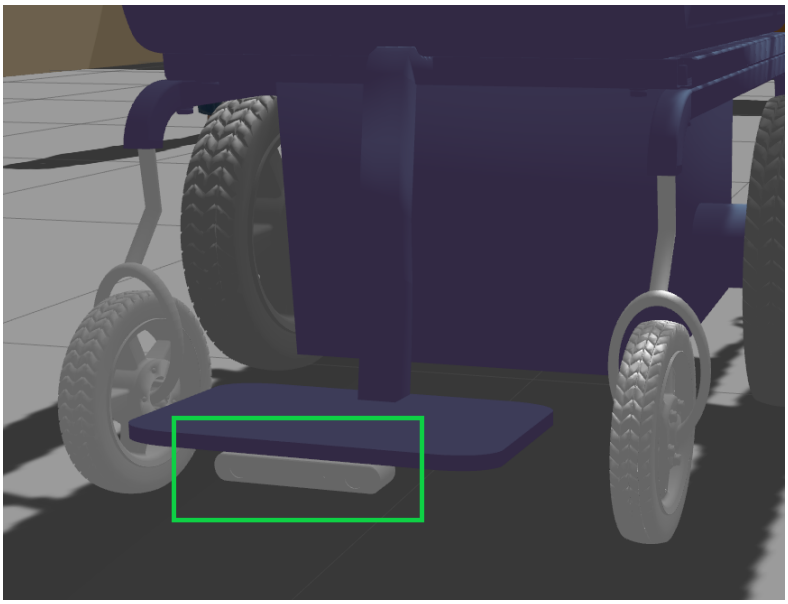


Figure 6.2: Example of the ZED 2 model under the footrest.

In case there is the necessity of adding a sensor on the back of the wheelchair, it could be easily placed on the main frame as shown in figure 6.3. Adverse to the respective front placement, as long as it is centered, the camera is not obstructed by anything even for high horizontal field of views. In terms of height placement, there are no significant differences, but for higher FOV cameras, a little obstruction from the wheel supports can occur. Thus, it is best to be always positioned on the bottom.

Figure 6.3: Example of the ZED 2 model on the back frame.

6.1.1.2 Laser models

Unlike visual sensors, laser scanners are consistently more compact. Thus, it might be easier to find spots where to place them. However, the laser scanners tested are purely 2D scanners. As

such, the height of the scanner must be considered. The laser scanner simulated used the real life model of the RPLidar A3 (4.2.1).

Despite all the spots available to put the laser scanner, it is impossible to find one where it can take advantage of a 360º degree scan. Nonetheless, achieving 180º is good enough and better than any visual sensor.

Firstly, the possibility of setting the scanner like the visual sensor, under the footrest, was pondered. In the simulation, this position is viable, allowing for a good 180ª laser scan of the environment. However, as already mentioned, the real chair has height adjustment, and since laser data is purely planar differences in height can have impact on the data obtained.

Therefore, the best position found to place the laser scanner would be above the small wheels, as shown in figure 6.4. In the simulated chair model, the position above the wheels seems impossible, but, the real model has the wheel supports placed in a manner that would allow it. Moreover, this placement can be followed both in the front and the back of the chair with the same height in both lasers (that would not be possible if the laser was stationed under the footrest). Additionally, the position allows, if needed, to scans above the 180ª radius.

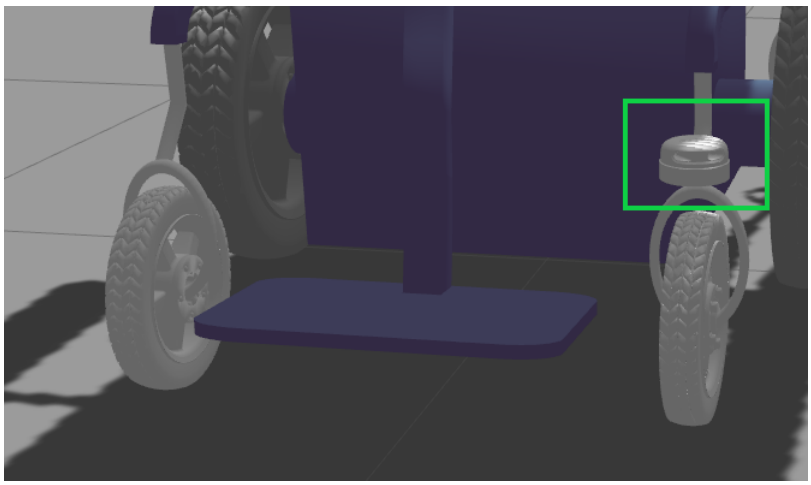


Figure 6.4: Placement of RPLidar models over the wheels

6.1.2 Sensor Configurations

As stated in Chapter 5, the sensor plugins are highly configurable. As such, it is possible to test a variety of different configurations of sensors that would try to resemble specifications close to real life ones, like the ones referenced in Chapter 4.

For the laser model used, the plugin followed closely an RPLidar A3 that had a scan radius of 180ª (instead of 360ª), and a laser range of 0.2-25 meters. Furthermore, the wheelchair is capable of using one laser model in the front, or two in total. Where one is placed in the front and the other in the back of the wheelchair. Whenever tests were made with the two laser scanners the *ira_laser_tools* was used to merge both laser data from each sensor into one, appropriately.

The same goes for the camera models. The wheelchair can make use of at most two cameras at the same time. If both sensors were to be used, the point clouds of each sensor are merged together into one with the help of an auxiliary created package (*wheelchair_pcl*) that makes use of the Point Cloud Library (figure 6.5).

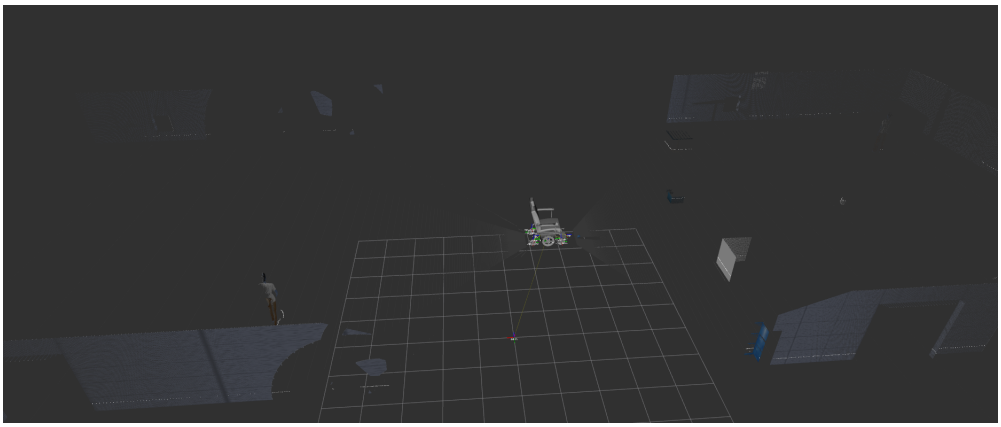


Figure 6.5: Example of a merged point cloud from two depth cameras.

All camera configurations used the depth camera plugin. The multicamera plugin was also tested as to demonstrate a stereo camera where the process of the images would be done externally with the help of packages like *stereo_image_proc*. However, the use of that plugin resulted in point clouds that were unusable because of bad quality and scaling. Normally, stereo sensors with processing outside the camera, like the ZED 2, provide a fully fledged SDK that provides easy means to obtain a point cloud from the images captured.

It is not in fact possible to easily and precisely simulate any real sensor. Nevertheless, it is possible to test the most important aspects of the sensors, as to perceive a minimum viable product, like the update rate, the field of view and the maximum range. Therefore, configurations of camera sensors that would approximate the ones in Chapter 4 were tested. The sensors would have a FOV in the range of 64-110[a] and a maximum range of 9-20 meters.

Besides using both types of sensors separately, tests were also done with two sensors of each type. A laser scanner in conjunction with a depth camera on the front of the wheelchair. The laser scanner could still represent something like the RPLidar A3 while the depth camera served as a low range (4 meters) but very precise visual sensor. This configuration tries to give a 3D view of the environment on a very precise, planar, long range scanner. Whenever this configuration was used, the system would firstly convert the point cloud to a laser scan with the aid of the *pointcloud_to_laserscan* package, and consequently, merge the resulting laser scan with the one from the laser sensor by using again the *ira_laser_tools*.

6.2 Simultaneous Localization and Mapping

The quality of the data obtained from the sensors can have a great impact on the results of SLAM. As a means to determine the best viable setup for the wheelchair, three use cases were

tested. For each use case, all sensor configurations previously mentioned were analyzed and the resulted maps and wheelchair locations were compared.

Use case 1: Indoor Loop Closure

The first use case places the wheelchair inside a sizable building with considerable vast rooms as shown in figure 6.6. This use case tests the robustness of loop closures for each setup. The wheelchair follows roughly the same path for each test. The resulted maps and current locations of the wheelchair on runtime were compared between each other.

Figure 6.6: Wheelchair first use case.

The first test made, used a sensor similar to an L515 (4.3.3). Of all the sensors researched, the L515 is the one with the lowest maximum range (9 meters) and one of the lowest horizontal FOVs (around 70º). Therefore, given the wide and big rooms of the building, this setup failed on obtaining a desirable map, as shown in figure 6.7. Furthermore, while the wheelchair was moving through the building the setup struggled to calculate the real location of the chair, causing a lot of pose jumps.

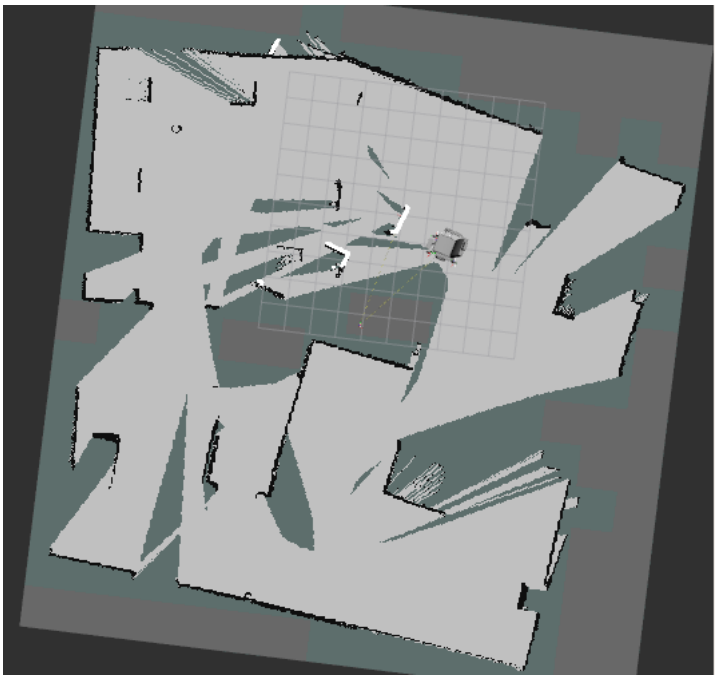
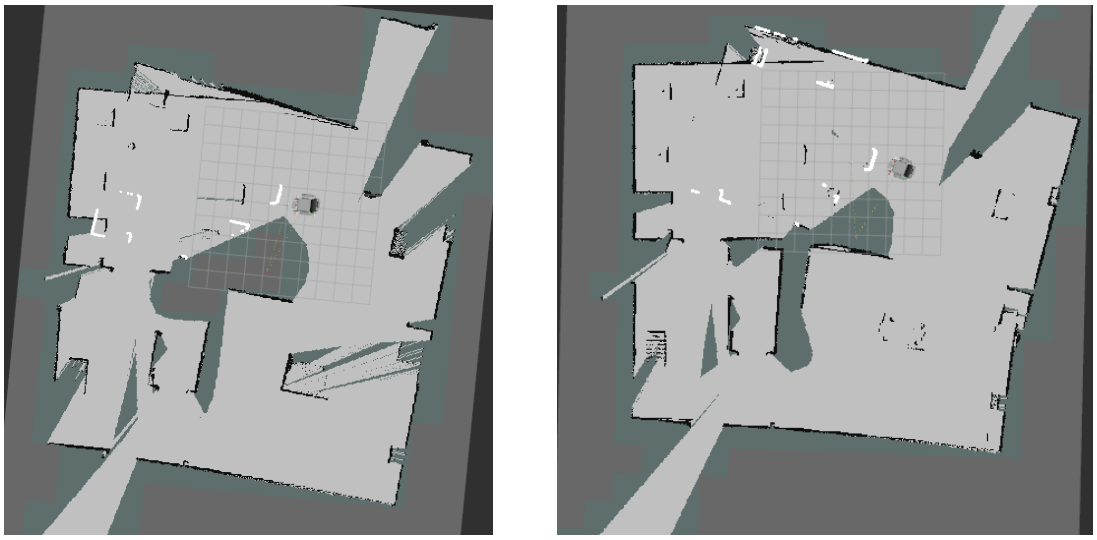


Figure 6.7: Resulting map and position with the L515 configuration.

The following test, used Mynt EYE D1000 (4.3.1) sensors. As previously mentioned, the Mynt EYE D1000 provides two available models that makes trades between the FOV and maximum range. A model has a higher horizontal FOV of 105º and a lower range of 10 meters (D1000-120), while the other uses a lower horizontal FOV of 64º but a higher 15 meters range (D1000-50).

The D1000-50 model's long scanning range helped it to obtain somewhat better results than the L515 even though it presents a slightly lower FOV. However, the results presented were still not ideal, the system still struggled to continuously obtain the current position of the wheelchair, causing the map layout to not be perfectly identical to the real environment (figure 6.8).

Due to the higher FOV, the other Mynt model was capable of extracting more details about the surroundings, yet in the end, it still failed to obtain a good current position even after loop closure (figure 6.8).



(a) D1000-50 (b) D1000-120

Figure 6.8: Resulting maps using Mynt Eye D1000 models.

Given the results obtained from the previously tested sensors, it can be derived that both horizontal FOV and maximum range are important for achieving good results in this use case. Thus, because of their higher FOV and maximum range, both the ZED 2 and RPLidar sensors were capable of obtaining satisfactory results without any problems in attaining the chair's position.

(a) ZED 2 (b) RPLidar A3

Figure 6.9: Resulting maps using the ZED 2 and RPLidar sensors respectively.

As it can be seen in figure 6.9, compared between each other, both the laser scanner and ZED 2 present similar findings. Moreover, even though the RPLidar was capable of scanning more area than the ZED 2 due to the higher specs, it failed to obtain the real footprint of some obstacles. This

is where the 2D nature of LIDAR struggles, as it is not capable of collecting a real representation of the environment. Anything that is lower or higher than the height set by the laser scan, will not be detected by it and consequently will not be added to the map. Also, some spotted objects might be misrepresented as their point of detection might not represent their real perimeter. For example, the tables in this testing were basically miss identified on the map, due to the scan only "hitting" their feet (figure 6.10).

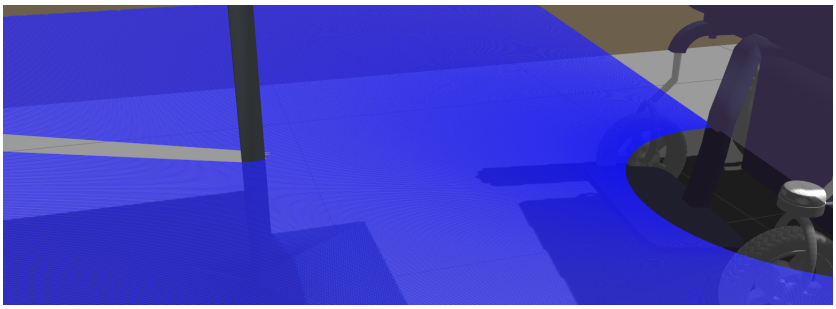
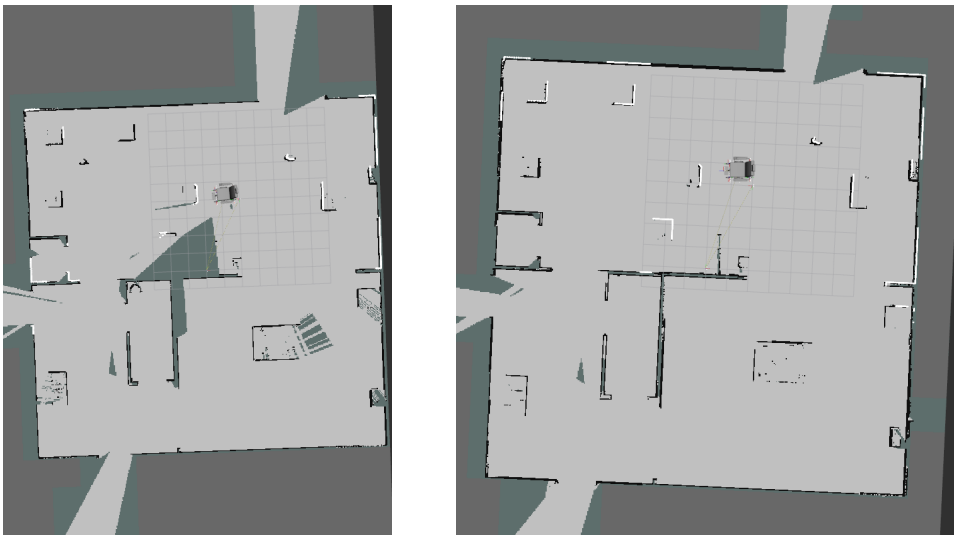


Figure 6.10: Laser scan only "seeing" a tables feet.

The setup that used both the laser scanner and a depth camera, somewhat fixed this problem by merging the data from both sensors. As such, the wheelchair could take advantage of the higher FOV and range of a LIDAR while still obtaining a 3D view of the environment. However, with a low range but high precision camera, the system could only be capable of figuring the real footprint of some obstacles when it got close enough for it. Therefore, in order to achieve as good results as the ZED 2 in terms of detail, a higher range and FOV depth camera would be needed, however, in this case there are no justifiable reasons to not just solely use the ZED 2 instead.

Tests with the use of two of the same sensors were also done, where, one sensor was added to the back of the chair. Given that the RPLidar and ZED 2 gave perfectly good results by themselves, adding another sensor to the back brought no justifiable improvements to the achieved map. It only allowed the system to "complete" the map of the building faster. Nevertheless, for the sensors that failed individually the use of two of them fixed their problems, helping them to achieve pretty reliable results (figure 6.11).



(a) Two Mynt Eye D1000-50 (b) Two ZED 2

Figure 6.11: Resulting map of using two Mynt Eye D1000-50 and two ZED 2 respectively.

Overall, this use case showed that, given the importance of the range and FOV of the sensors, only sensors like the ZED 2 and RPLidar were capable of achieving good results. The L515 and D1000 were only capable of being useful when at least two of them were used simultaneously.

Use case 2: Outdoor Riding

The second use case places the wheelchair at the same spot as the previous one, but instead of moving around the building, the wheelchair tries to go outside and travel through the outdoor environment without any loop closure as shown in figure 6.12.

Figure 6.12: Wheelchair second use case traveling through the outdoors.

As expected, given the obtained findings in the first case, with even bigger areas, sensors like the L515 and the Mynt Eye once again failed in achieving acceptable results. With these sensors pose jumps were way more frequent, resulting in the system failing to correctly localize the wheelchair within the map (figure 6.13).

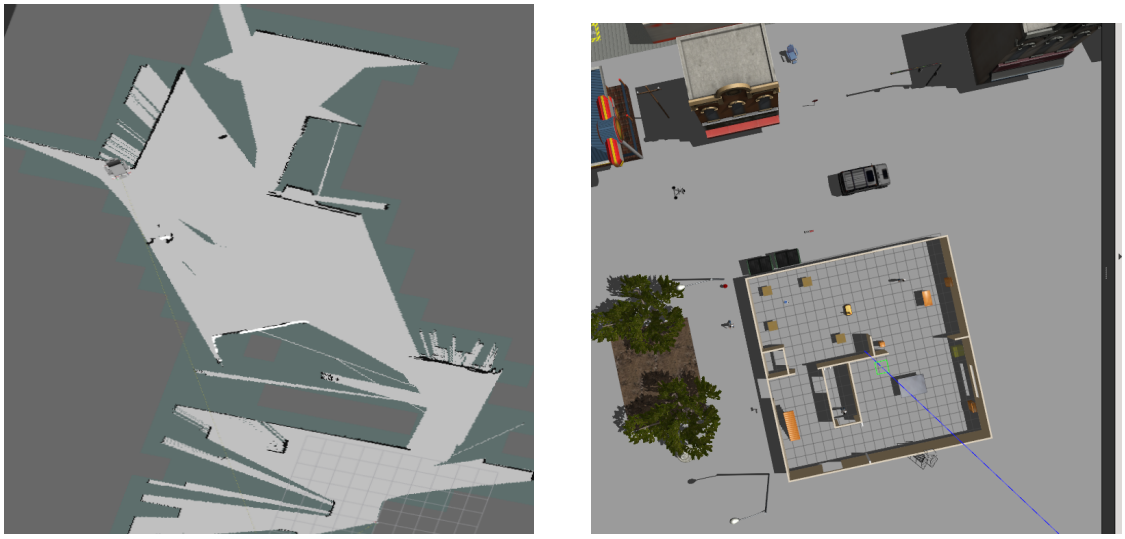


Figure 6.13: Example of the Mynt D1000-50 failing to obtain the wheelchair correct position.

However, once more, making use of another sensor on the back makes the Mynt Eye and L515 reliable options (6.14).

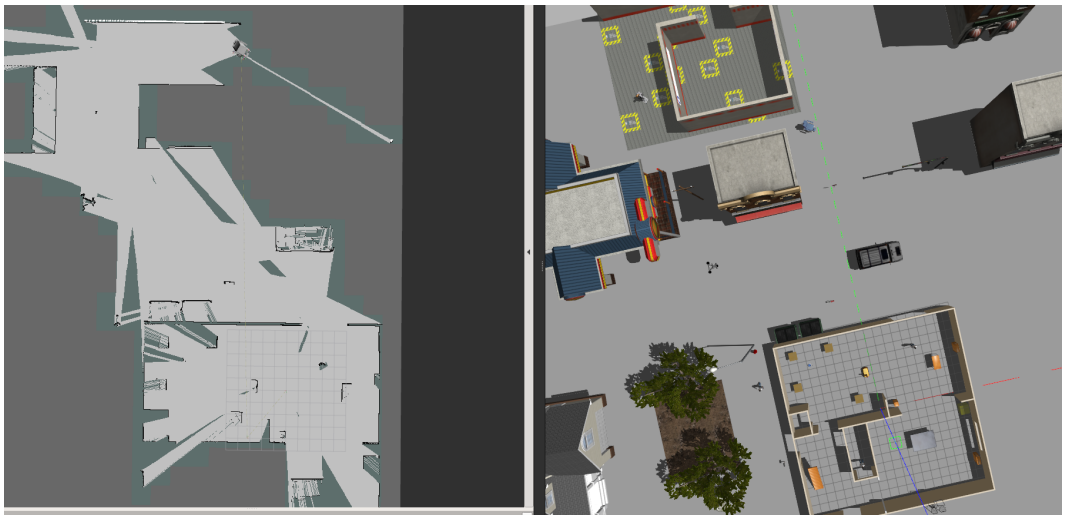


Figure 6.14: Example of using two Mynt Eye D1000-50 outside.

Overall, the ZED 2 obtained pretty good results, but at the last segment where it drives through the two buildings the lack of a higher FOV and any other obstacles besides the two building adjacent to him, makes the system struggle a little, causing some pose jumps and inaccurate location readings. As such, in the last situation the 180º FOV helps the RPLidar to achieve the best results for this use case (figure 6.15).

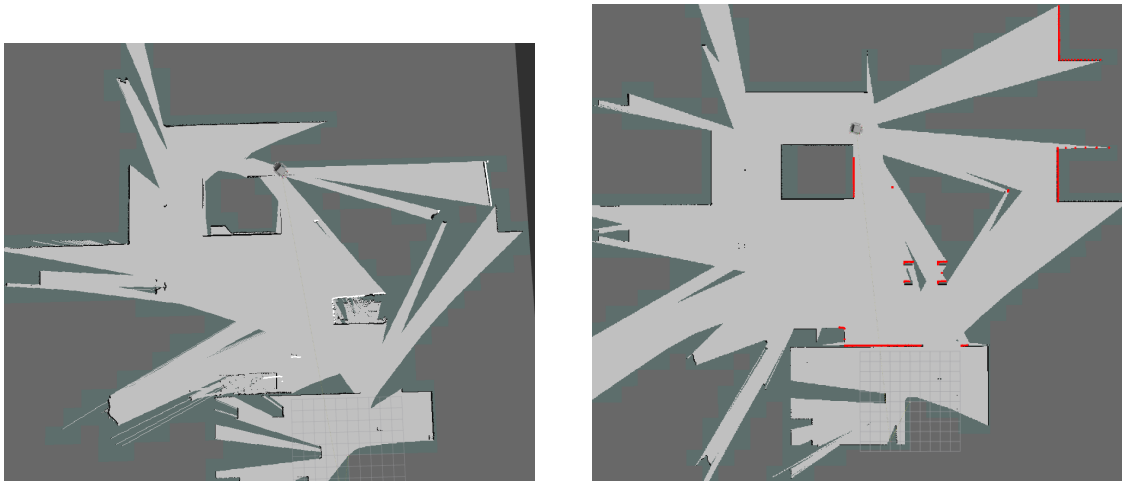


Figure 6.15: Resulting maps of using a ZED 2 and RPLidar A3 respectively in outdoor environment.

In conclusion, for this test case, the best results are achieved by the RPLidar. For obtaining similar or better results with the depth cameras, a setup of at least two of them would be needed. Nonetheless, the same problem of misrepresenting obstacles can still occur with the laser. Making use of the visual sensor in conjunction with the laser, once again, only helped when the wheelchair got close enough to the obstacle.

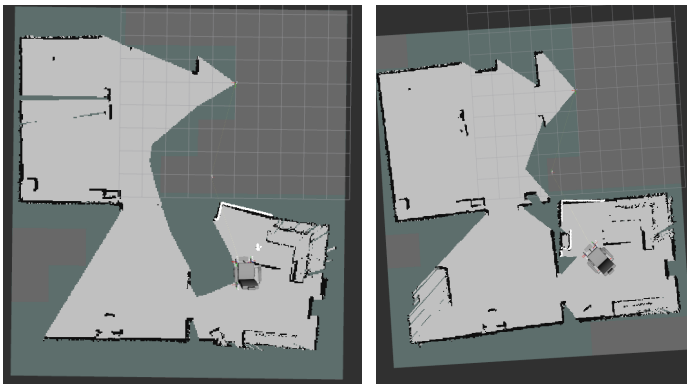
Use case 3: Indoor Home

The first use case places the wheelchair in an indoor environment, however, the building used a lot of wide rooms. Most situations inside a building with a wheelchair is in a house. Normally, houses do not have a lot of big rooms. As such, the third use case tries to test the sensors while the wheelchair travels through three small rooms of a house, as seen in figure 6.16.

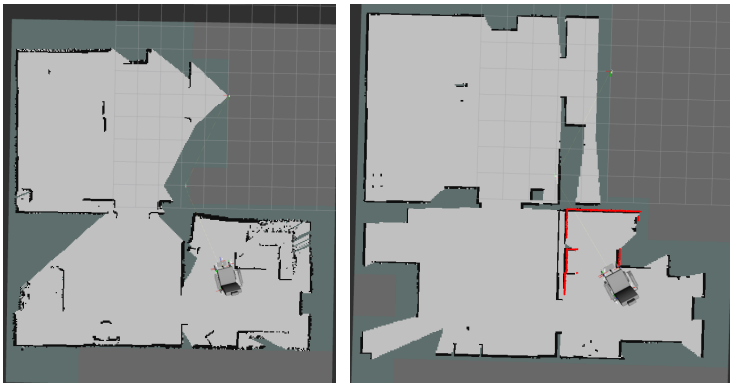
Figure 6.16: The third use case of the wheelchair going through a house.

In this situation, since the wheelchair is placed in smaller areas, the difference in the maximum depth range between the sensors are mostly meaningless. Therefore, all camera sensors by themselves attained very similar results between each other. Interestingly enough, all of them struggled a little in the last room when rotating around themselves, causing a slightly uneven portrayal of the room layout. As it can be seen in figure 6.17, the ZED 2 and D1000-120 present very identical outcomes derived from their mostly equal horizontal FOV, while the D1000-50 struggled more due to the low FOV radius.

The much wider FOV of the RPLidar makes it obtain the best results, without any struggles in any room. Yet again the system fails to portray properly some obstacles, but in this case, the setup with both a LIDAR and a depth camera is more reliable since the short range of the depth camera is enough in this case.



(a) Mynt Eye D1000-50 (b) Mynt Eye D1000-120



(c) ZED 2 (d) RPLidar A3

Figure 6.17: Mapping results of the sensors inside a house.

When using two deth sensors at the same time the problem of failing to properly map well when rotating around itself in the third room is fixed, and the system is capable of portraying perfect results.

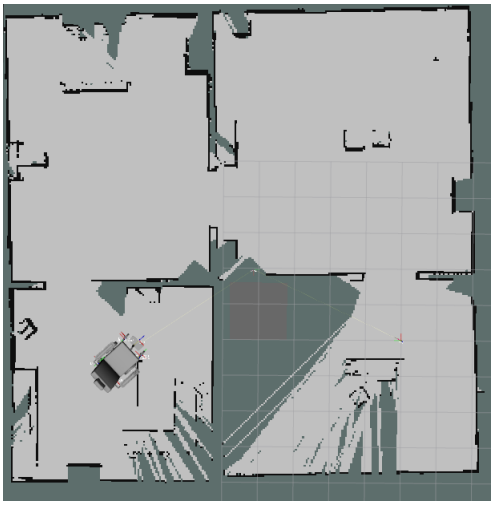


Figure 6.18: Result of the third use case with two ZED 2 sensors.

Furthermore, this use case was useful to demonstrate one of the struggles of 2D grid-based SLAM. The lack of height representation on the map makes it impossible for the system to portray something like a hole. As seen in figure 6.16, the staircase that goes towards a lower level of the house is not portrayed in the resulting map. Even though depth sensors are indeed capable of detecting those stairs, since the SLAM approach depends on a 2D laser scan it is not feasible to correctly transform the point cloud to a laser with negative obstacles.

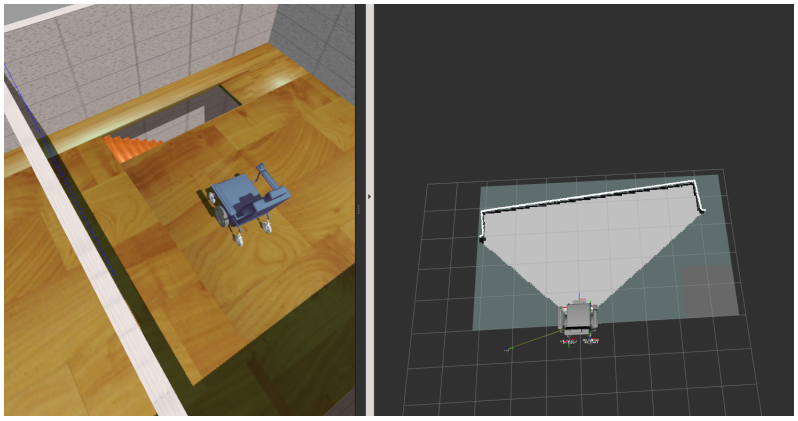


Figure 6.19: The wheelchair fails to perceive the stairs.

Summary

In general, for all the use cases tested, the ZED 2 and RPLidar A3 were the only sensors capable of achieving reliable results in all tested situations. The other sensors were only viable if at least two of them were used. In table 6.1 it is shown the reliability of each sensor for each use case. If the result is *not reliable*, it means those sensors were not capable of achieving results that were in any way viable to use in the real chair. The *most reliable* results correspond to the sensors capable of achieving the results that most correctly portray the environment without any problems, while the *reliable* ones, correspond to the sensors that, although did not give the best portrayal or struggled a bit, are still considered applicable for real use.

In the first use case scenario, only the ZED 2 and the RPLidar A3 achieved reliable results, with the ZED 2 achieving a better representation of some obstacles. The other sensors were not capable of portraying the map properly, unless another sensor was used in conjunction.

Moreover, in the second case, the Intel and Mynt sensors failed completely to portray the correct position of the wheelchair within the map. Thus, once again, these sensors were only viable when at least two of them were used simultaneously. Additionally, ZED 2 was capable of achieving decent results but at some point it briefly failed to compute exactly the location of the wheelchair, causing some small location inaccuracies. With the higher FOV and range, the RPLidar was the only sensor capable of achieving good results without any struggles.

In the third situation, all visual sensors performed similarly with decent results. However, the Intel and Mynt D1000-50 struggled a little more because of the lack of FOV, nonetheless they were still viable. The Mynt D1000-120 and the ZED 2 with the higher FOV were capable of a

more correct portrayal of the house the wheelchair was in, but at some point they also failed to correctly depict the layout of the environment. Again, the RPLidar A3 was the only sensor that most correctly demonstrate the layout of the house, yet, because of its nature it failed to correctly portray the footprint of some objects in the house, which could impact the performance of the navigation system.

Table 6.1: Overview of the overall results of each sensor in each use case

Reliability using SLAM	**Intel L515**	**Mynt D1000-50**	**Mynt D1000-120**	**ZED 2**	**RPLidar A3**
Use case 1:	Not reliable	Not reliable	Not reliable	Most reliable	Reliable
Use case 2:	Not reliable	Not reliable	Not reliable	Reliable	Most reliable[1]
Use case 3:	Reliable	Reliable	Reliable	Reliable	Most reliable[1]

[1] It performed better compared to the other sensors, but misrepresents/ignores some objects

6.3 Navigation

For the system to properly be able to navigate through the environment without any struggles, it is necessary for it to continuously provide a good map and localization of the wheelchair. The *move_base* package allows for a vast amount of configurations, with parameters that can have a big impact on the results. As such, this subsection tests the path planning in situations similar to the ones done in the previous section as to evaluate the performance of the autonomous navigation component of the system.

From the various tests done to the SLAM component in the previous section, the sensors that received the best results were the ZED 2 and the RPLidar, where the last one performed a little better in some situations. However, the problem of the RPLidar misrepresenting some objects can be really impactful to the navigation system as it can misjudge some obstacles and send the wheelchair in a path that can lead to collisions. Hence, most testing done with the path planning component made use of two ZED 2 sensor as to avoid the sensor data and the SLAM results to bottleneck the performance of the navigation system.

Cost maps can have the most impact on the path calculated by the planners. More specifically, the *inflation_radius* parameter is responsible for setting the inflation radius done to cells around an obstacle in the map (figure 6.20). The bigger the radius, the paths obtained are more prone to keep a safe distance from the obstacles, however, with that, the system can be more redundant to traverse through doors or small areas that do not have a lot of free space. The smaller the radius better paths may be found when going through small areas, however the system follows paths that get to close to some obstacles and occasionally collides with them.

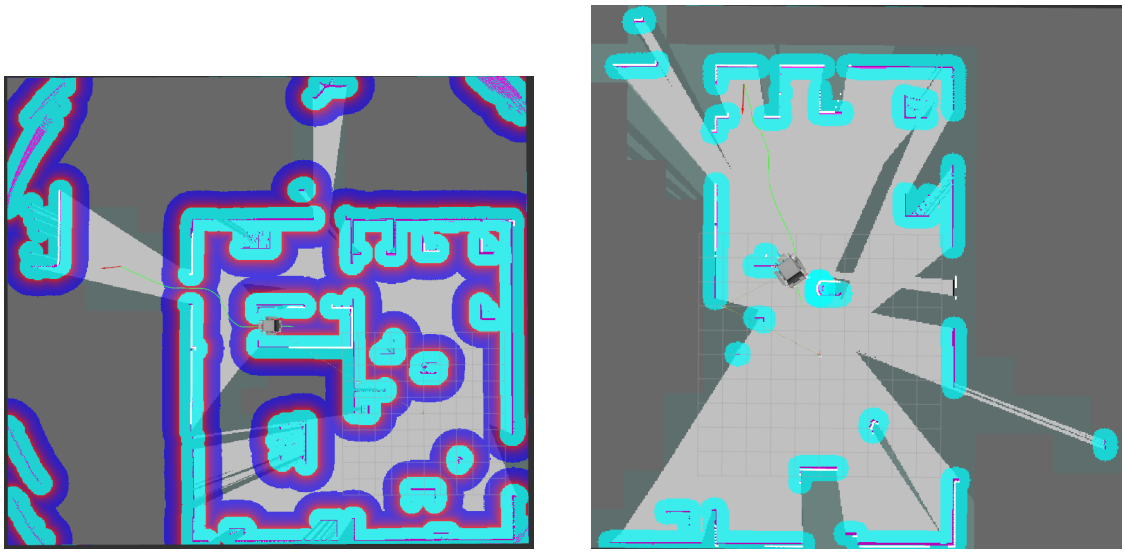


Figure 6.20: Resulting cost maps from a big and small inflation radius, respectively.

The *move_base* package makes use of two costs maps, therefore each one of them can have different settings. Through testing the wheelchair navigating through the world maps used in the test cases with the SLAM method, it was found that having different inflation radius between each cost map helped fill the gap between moving safely and still going through restrict spaces. Thus, the system would make use of a global cost map with a bigger inflation radius (around 1.5 meters) as for the global planner to retrieve a path to the end goal that tries to keeps a safe distance to obstacles, and a local cost map with a smaller radius (around 0.75 meters) as to allow the local planner to make the wheelchair go through small areas when needed. This setting helped for most situations, but whenever the wheelchair was in places where the area of movement was extremely limited, it still struggled a little.

Furthermore, the global cost map tries to fill a big or the entire portion of the environment, while the local cost map represents only a small area around the wheelchair. The *costmap_2d* package recognizes two different types of cost maps. One that is static and fills and matches the size of the map given by an external source (in this case the map resulted from SLAM). And another, that, given a specific width and height, keeps the wheelchair always in the center of it, as such, the cost map only keeps information about obstacles that are within the given area around the wheelchair, as the wheelchair moves, obstacle information that leaves the designated area is dropped and forgotten. Therefore, the designated wheelchair system makes use of a small dynamic local cost map as it only needs to know information about the current surroundings of the wheelchair, while the global cost map is static and fills all the map information given by using SLAM. Having the global cost map fill the entire map scanned, in long term use, might cause memory problems. Therefore, it is also feasible to make the global cost map only save information around a specific big area (like the local one), yet, this would not allow the user to demand the wheelchair to go to a spot outside the specified area.

The global planner used does not allow for too much configuration. It can either use the Dijkstra or A* algorithm to find the shortest path to the destination. The A* algorithm usually resulted in unpredictable and undesirable results compared to the Dijkstra so only the latter was used. More importantly, the *global_planner* has a variable nominated *allow_unknown*, that sets if the planner can create paths that go through unknown space. The system had this parameter disabled as it would cause the wheelchair to find and follow paths that were impossible, making the wheelchair sometimes go to some place and then stopping and changing direction as it finds it is not possible to go through that path. Having this setting disable avoids the system making unnecessary movements, however, it makes it harder for the user to set end goals in areas that were not fully mapped (figure 6.21).

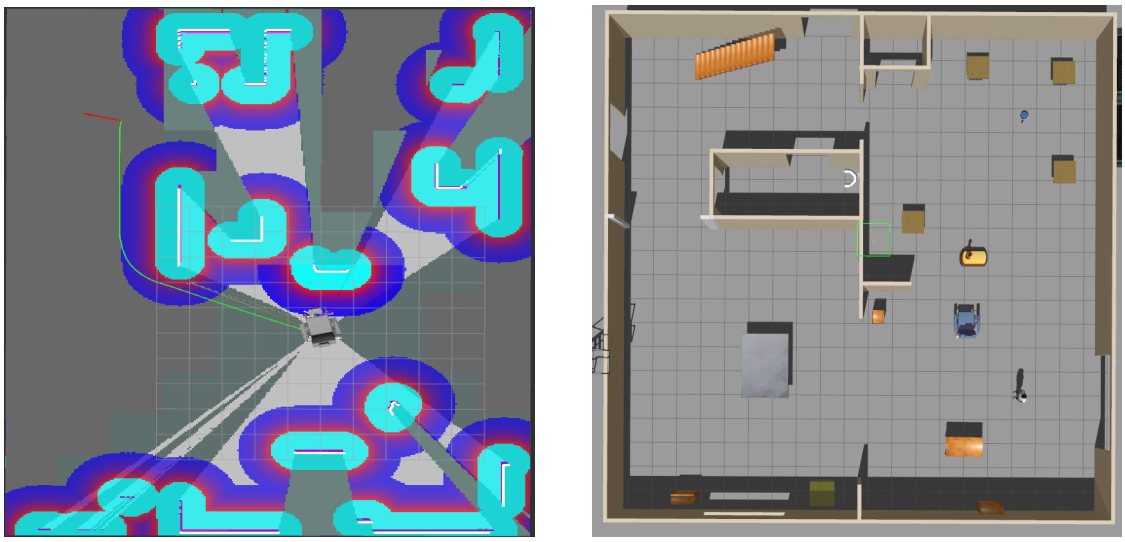


Figure 6.21: The wheelchair calculating a path that in reality is not possible.

The *dwa_local_planner* and *base_local_planner* were used. Overall, the *base_local_planner* gave better results specially in areas where the move area was more restrict. Furthermore, the *base_local_planner* allows to use the Direct Window Approach method as well. As to the use of the Trajectory Rollout and the DWA approaches delivered identical results between each other, and, since technically DWA is more efficient, it was the method used by the planner.

Overall the system was able to easily navigate through big rooms and outdoor areas without any noticeable problems. Nevertheless, although it can find paths and go through small areas as well, the system struggles to go to a specified spot in the opposite direction it is facing when it is in spaces where the area of movement is very strict, like the small house world used in the third use case (figure 6.16).

Chapter 7

Conclusion

Wheelchairs are a crucial element in today's society, as a means of giving independent mobility for those who need it like the elderly and the impaired. With the constant improvement in technology, the shift from manual wheelchairs to powered ones have helped improve the usability of them for most users. However, most powered wheelchairs are not enough to satisfy and assist individuals with severe physical or mental impairments.

Intelligent wheelchairs strive to help those with severe impairments by adding autonomous systems that try to minimize the work needed by the user. Many prototypes of IWs have been proposed over the last decades, despite that, most of them are not easily accessible to the market because of either the cost or the limitations they may have. IntellWheels is different from most other prototypes, by making it viable to easily and cost-effectively develop IWs, while still providing them for most people.

When developing a complex system like an intelligent wheelchair, it starts to be hard to practically test it in all different kind of situations and setups without any costs and risks of damage. The developed system and simulation environment allowed for simply testing the capabilities of autonomous navigation and SLAM of the wheelchair without any need to apply it to the real chair. Furthermore, since the system was developed in ROS, technically the system can be applied to the real wheelchair with not much effort. As such, the simulation allows for an iterative development of the intelligent wheelchair where changes to the system can firstly be tested in simulation and then formerly applied to the real chair.

Therefore, the purpose of this dissertation was to implement a simulation environment for an intelligent wheelchair. With all the designated objectives accomplished, the wheelchair was able to perform the task of Simultaneous Localization and Mapping with path planning in any simulated world, making use of simulated sensors. Moreover, the simulated wheelchair was tested around environments that portrayed a broad understanding of indoors and outdoors. With these tests it was possible to obtain conclusions in which real life sensors might be more useful and what are their advantages and disadvantages in specific situations. Overall, while doing SLAM with only using one sensor in all situations the conclusion would give the victory to a 2D laser scanner like the RPLidar A3. Despite that, the lack of 3D view gives a strain in the system and the possibilities

of further improvement in something like adding semantics. As such, it is more feasible to decide to use a good depth camera like the ZED 2 that still gives good results despite not being as precise as the laser sensor. To prevent the shortcomings given by the ZED 2 compared to the RPLidar A3, the use of two depth cameras in total made the system capable of achieving the best overall results without any substantial disadvantages, despite the possible increase of the complexity of the system.

Additionally, the implemented path-planning component of the system was tested and able to achieve, with the proper sensor configuration, a robust navigation system in outdoor and indoor environments whenever there was a wide space of movement. However, the wheelchair could still struggle to properly navigate the system whenever it moved around strict areas, where there was not a lot of space for maneuvering.

This simulation environment creates a new step on improving the IntellWheels 2.0 making a viable project that can have a real impact in the future.

7.1 Contributions

Overall, in this dissertation, a wheelchair system capable of Simultaneous Localization and Mapping and autonomous navigation on a simulation environment was implemented. From this simulated environment, the main contributions of this project are:

- A better understanding of the state of the art of intelligent wheelchairs and SLAM methods, and consequently the possible impact this project can bring.
- The proper implementation of the system capable of using SLAM and path-planning in a simulated environment.
- With the test scenarios created, an understanding of the impact the FOV and maximum range can have in the proficiency of the SLAM methods in certain situations.
- Demonstrate from the sensors tested in simulation, which ones were able to obtain reliable results (like the ZED 2 and RPLidar A3) in the tested situations, and which ones were not.
- Understand the impact the type of sensor can have in the achieved results from the SLAM method (LIDAR vs visual).
- Demonstrate the possible shortcomings of 2D based SLAM methods.
- Define a decent enough path planning system capable of making use of SLAM to easily navigate through the environment.
- Understand the shortcomings of the navigation system, where it is strictly dependent of the results of SLAM and struggles in strict environments.

7.2 Future Work

The tests done on the simulation environment allowed to gather a broad view of the necessary setup for the wheelchair achieve good results while doing SLAM. The results come from some certain situations in order to conclude what is enough for indoor and outdoor environments. Nevertheless, most of these test case scenarios are not portraying environments identically as they are in real life. Furthermore, the simulated sensors could not fully represent the real life counterparts, it was not possible to simulate how those sensors would work in different light systems and how precise they would be in the advertised maximum range. For that, the real sensors would need to be tested and calibrated.

There are other improvements that could be made to the system in the future, such as:

- Add methods to allow the system to perceive height of the environment such as holes and ramps.
- Improve the perception of the system with the addition concept of semantics, like chairs, doors and other important objects.
- Fully test the system in more detailed and realistic situations, with more dynamic environments.
- Implement the system to the real life wheelchair.
- Test and study more deeply the autonomous navigation component of the wheelchair and its possible improvements.

References

[1] Mohammad Al Sibai and Sulastri Abdul Manap. A Study on Smart Wheelchair Systems. *International Journal of Engineering Technology And Sciences (IJETS)*, 4(1):25–35, 2015.

[2] Nikolay Atanasov, Sean L. Bowman, Kostas Daniilidis, and George J. Pappas. A unifying view of geometry, semantics, and data association in SLAM. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:5204–5208, 2018.

[3] Augusto Luis Ballardini, Simone Fontana, Axel Furlan, and Domenico G. Sorrenti. ira_laser_tools: a ROS LaserScan manipulation toolbox. pages 1–6, 2014.

[4] Karsten Berns and Ewald von Puttkamer. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, (September):108–117, 2006.

[5] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Rol Siegwart. *Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps*. Number c. 2018.

[6] Johann Borenstein and Yoram Koren. The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 7(3):278–288, 1991.

[7] Sean L. Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J. Pappas. Probabilistic Data Association for Semantic SLAM. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729, 2017.

[8] Rodrigo A.M. Braga, Marcelo Petry, Antonio Paulo Moreira, and Luis Paulo Reis. Concept and design of the intellwheels platform for developing intelligent wheelchairs. *Lecture Notes in Electrical Engineering*, 37 LNEE:191–203, 2009.

[9] Rodrigo Antonio Marques Braga, Marcelo Petry, Luis Paulo Reis, and António Paulo Moreira. Intellwheels: Modular development platform for intelligent wheelchairs. *Journal of Rehabilitation Research and Development*, 48(9):1061–1076, 2011.

[10] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The Mobile Robot Rhino. *AI Magazine*, 16(2):31–38, 1995.

[11] Wolfram Burgard, Cyrill Stachniss, Kai Arras, and Maren Bennewitz. Introduction to Mobile Robotics SLAM : Simultaneous Localization and Mapping What is SLAM ? (June), 2010.

[12] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[13] Z E D Camera, S D K Overview, Z E D Overview, and All-aluminium Case. ZED 2 Camera and SDK Overview. pages 3–6.

[14] Howie Choset and Keiji Nagatani. Topological Simultaneous Localization and Mapping (SLAM): Towards Exact Localization Without Explicit Localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

[15] Ingemar J. Cox. Blanche—An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

[16] Ana Beatriz Cruz, Armando Sousa, and Luís Paulo Reis. Controller for Real and Simulated Wheelchair With a Multimodal Interface Using Gazebo and ROS. *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 164–169, 2020.

[17] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. *Sequential Monte Carlo Methods in Practice*, pages 499–515, 2001.

[18] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[19] Ana Rita Gaspar. Navegação e mapeamento subaquáticos simultâneos. *Master Thesis, Faculdade de Engenharia da Universidade do Porto*, pages 1–97, 2017.

[20] Brian P Gerkey and Kurt Konolige. Planning and Control in Unstructured Terrain. *In Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[21] Takashi Gomi and Ann Griffith. Developing intelligent wheelchairs for the handicapped. *Assistive Technology and Artificial Intelligence. Lecture Notes in Computer Science*, 1458:150–178, 1998.

[22] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 12 2010.

[23] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2432–2437, 2005.

[24] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. Fast and accurate SLAM with Rao-Blackwellized particle filters. *Robotics and Autonomous Systems*, 55(1):30–38, 2007.

[25] Giorgo 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.

[26] Peiyu Guan, Zhiqiang Cao, Erkui Chen, Shuang Liang, Min Tan, and Junzhi Yu. A real-time semantic visual SLAM approach with points and objects. *International Journal of Advanced Robotic Systems*, 17(1):1–10, 2020.

[27] Baichuan Huang, Jun Zhao, and Jingbin Liu. A Survey of Simultaneous Localization and Mapping. pages 1–16, 2019.

[28] Intel. LiDAR Camera L515. (December), 2019.

[29] Intel. Realsense lidar camera l515 — Intel. = https://www.intelrealsense.com/lidar-camera-l515/, [Online; Accessed: 18-06-2020].

[30] Development Kit and User Manual. Rplidar a3. *Slamtec*, pages 1–17.

[31] Mathieu Labbé and François Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, 2014.

[32] Mathieu Labbé and François Michaud. RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation. *Journal of Field Robotics*, 35(2):416–446, 2019.

[33] Jesse Leaman and Hung Manh La. A Comprehensive Review of Smart Wheelchairs: Past, Present, and Future. *IEEE Transactions on Human-Machine Systems*, 47(4):486–489, 2017.

[34] John J. Leonard and Hugh F. Durrant-White. Simultaneous Map Building and Localization for an Autonomous Mobile Robot. *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, 3:1442–1447, 1991.

[35] Ruijiao Li, Mohammadreza A. Oskoei, Klaus D. McDonald-Maier, and Huosheng Hu. ROS Based Multi-sensor Navigation of Intelligent Wheelchair. *2013 Fourth International Conference on Emerging Security Technologies*, pages 83–88, 2013.

[36] Fausto Orsi Medola, Valeria Meirelles Carril Elui, Carla da Silva Santana, and Carlos Alberto Fortulan. Aspects of Manual Wheelchair Configuration Affecting Mobility: A Review. *Journal of Physical Therapy Science*, 26(2):313–318, 2014.

[37] Aniket Murarka, Shilpa Gulati, Patrick Beeson, and Benjamin Kuipers. Towards a Safe, Low-Cost, Intelligent Wheelchair. *IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV)*, pages 42–49, 2009.

[38] MYNT Eye. Stereo depth camera mynt eye d1000 — MYNT Eye. = https://www.mynteye.com/pages/mynt-eye-d. [Online; Accessed: 18-06-2020].

[39] Alexandra Pereira Nunes. Mapeamento e odometria visual em ambiente subaquático. *Master Thesis, Faculdade de Engenharia da Universidade do Porto*, 2017.

[40] Phil Odor and Marison Watsson. Learning through Smart Wheelchairs: A formative evaluation of the effective use of the CALL Centre's Smart Wheelchairs as part of the children's emerging mobility, communication, education and personal development. *CALL Centre*, May 1994.

[41] Jean Oh, Martial Hebert, Hae-Gon Jeon, Xavier Perez, Chia Dai, and Yeeho Song. Explainable Semantic Mapping for First Responders. *ArXiv*, abs/1910.0, 2019.

[42] Open Source Robotics Foundation. Robot operating system documentation — Open Source Robotics Foundation. https://wiki.ros.org/, [Online; Accessed: 18-06-2020].

[43] Anish Pandey, Shalini Pandey, and DR Parhi. Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review. *International Robotics & Automation Journal*, 2(3):1–12, 2017.

[44] Eurico Pedrosa, Artur Pereira, and Nuno Lau. Efficient localization based on scan matching with a continuous likelihood field. *2017 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2017*, (4):61–66, 2017.

[45] Eurico Pedrosa, Artur Pereira, and Nuno Lau. A sparse-dense approach for efficient grid mapping. *18th IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2018*, pages 136–141, 2018.

[46] Eurico Pedrosa, Artur Pereira, and Nuno Lau. A Non-Linear Least Squares Approach to SLAM using a Dynamic Likelihood Field. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 93(3-4):519–532, 2019.

[47] Eurico Pedrosa, Artur Pereira, and Nuno Lau. Fast Grid SLAM Based on Particle Filter with Scan Matching and Multithreading. *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 194–199, 2020.

[48] Barnaby A. Perks, Rosalind Mackintosh, Colin P.U. Stewart, and Geoff I. Bardsley. A survey of marginal wheelchair users. *Journal of Rehabilitation Research and Development*, 31(4):297–302, 1994.

[49] Soren Riisgaard and Morten Rufus Blas. SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping. pages 1–127.

[50] Amberlay Ruíz Serrano, Ruben Posada Gómez, Albino Martínez Sibaja, Alberto Alfonso Aguilar Lasserre, Giner Alor Hernández, and Guillermo Cortes Robles. Performance Evaluation for a Multimodal Interface of a Smart Wheelchair with a Simulation Software. *Research in Computing Science*, 96(1):75–84, 2015.

[51] Richard C. Simpson. Smart wheelchairs: A literature review. *Journal of Rehabilitation Research and Development*, 42(4):423–435, 2005.

[52] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural Topological SLAM for Visual Navigation. 2020.

[53] Slamtec. Introduction to Standard SDK. 2016.

[54] Márcio Miguel Sousa. Multimodal interface for an Intelligent Wheelchair. *Master Thesis,Faculdade de Engenharia da Universidade do Porto*, (July), 2008.

[55] StereoLabs. Zed 2 stereo camera — StereoLabs. = https://www.stereolabs.com/zed-2/, [Online; Accessed: 18-06-2020].

[56] StereoLabs. Zed sdk documentation — StereoLabs. = https://www.stereolabs.com/docs/, [Online; Accessed: 18-06-2020].

[57] Sebastian Thrun. PROBABILISTIC ROBOTICS. 2000.

[58] Sérgio Vasconcelos. Multimodal interface for an intelligent wheelchair. *Master Thesis, Faculdade de Engenharia da Universidade do Porto*, 2019.

[59] Zemin Wang, Qian Zhang, Jiansheng Li, Shuming Zhang, and Jingbin Liu. A computationally efficient semantic SLAM solution for dynamic scenes. *Remote Sensing*, 11(11):1–19, 2019.

[60] Holly A. Yanco. Wheelesley: A Robotic Wheelchair System: Indoor Navigation and User Interface. *Assistive Technology and Artificial Intelligence*, 1458:256–268, 1998.

[61] Chao Yu, Zuxin Liu, Xin-jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments Chao. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1168–1174, 2018.

[62] Guido Zunino. *Simultaneous Localization and Mapping for Navigation in Realistic Environments*. 2002.