POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Application of Autonomous Robotic Vehicles for Wireless Charging of Power Wheelchairs

Supervisors

Prof. Marina INDRI

Prof. Zeljko PANTIC

External Prof. Kevin HAN

Candidate

Giampietro NIGRO

December 2022

Summary

This work aims to provide a baseline for a new indoor robotic application, an assistive robot able to recharge a Power Wheelchair (PW) using Wireless Power Transfer (WPT). As such, I had to devise a plan from scratch and develop the first-generation prototype to test the application following strict price and size constraints, as the robot must fit under a wheelchair. Create 3, from iRobot, was used for quick prototyping, as it was both inexpensive and small. The classical Roomba shape has proven to be an effective hardware solution for the intended application. Its software is entirely based on ROS 2, publishing all its sensor information on easily accessible topics, while ROS 2 servers and subscriptions are used to control actuators and respond to users' requests.

While the WPT will be developed by other team members in a later stage, my focus was on the overall system, developing the mock-up wheelchair, communication, wheelchair detection, and path planning. The main technologies used are:

- **ROS 2**: used as a middle-ware, detecting information from the robot in an asyncronous way and running my application based on the received data.
- **Python**: used to code the desired behavior and implement it using ROS 2 framework.
- **BLE Beacons**: used both for coarse detection and communication.
- **IR LEDs**: used to achieve fine detection.
- **Micro controllers**: Arduino and Raspberry Pi 4 were employed for different reasons, the former mounted on the wheelchair, while the latter on the robot.

Assistive robotics is a rapidly expanding field, as the average age in most developed countries is growing increasingly fast. The technology to improve the quality of life of elders and people with disabilities already exists, and in the last five years, many new products have been introduced to the market. The *Chapter I* of this thesis is devoted to the introduction, discussing some of these products

and providing a clear understanding of where the idea of this project came from, what problems it fixes, and what the exact goals and constraints are. A brief introduction of the Create 3 robot and ROS 2 will also be given, as they are critical tools used in this project.

The robot path planning is split into two macro areas: **fine detection** and **coarse detection**. The fine detection procedure, which is described in the *Chapter II*, was the crucial part of the project. As the robot had to approach a wheelchair and go under it, particular care was taken to make the algorithm and detection method robust. Even an occasional failure of this process would not be acceptable, as it undermines the actualization of the product. This was achieved by making use of Infrared (IR) LEDs. The mock-up wheelchair setup was built using constraints slightly stricter than in an average real system, assuring that the robot can perform its task in most practical situations.

The coarse detection procedure is described in *Chapter III*. Indoor operation eliminated GPS as a way to perform the orientation, and alternative ways to bring the robot inside the range of IR LEDs had to be found. Moreover, simultaneous localization and mapping (SLAM) was not an available option either due to the cost and size constraints. The capabilities of BLE (Bluetooth Low Energy) technologies were tested. Using RSSI (Received Signal Strenght Indication), it is possible to estimate the distance to BLE beacons. The precision and limits of this method will be discussed, as it was determined to be too unreliable. Still, beacons could be used to determine the room location of the wheelchair and communicate with the robot, making it possible to start and stop the robot at a distance of about 15 meters. A trilateration algorithm was proposed based on RSSI measurements, demonstrating how such technologies could be used despite their limitation.

Once the hardware section and the main working principle were explained in the previous chapters, the main software is described in *Chapter IV*. Particular attention will be given to the new nodes written and how they fit the overall system, highlighting the subscribed and publishing topics and the ROS 2 action service used. Quality of Service (QoS) configurations, available on ROS 2, will be discussed. The robot is programmed to work through different states, in a particular sequence responding to precise inputs, and each state will be discussed. Emphasis will be on the **fine state** and **approaching state**, the two states that are the most developed and fully functional.

Finally, in *Chapter V* gives final digressions and suggestions for future research. Results will be discussed critically, as this is the first step of the process. When building the second generation prototype, which will feature a custom-designed

robot, it is essential to use this newly built knowledge, included what worked well and what did not and what type of extra hardware is needed to improve the results.

Acknowledgements

ACKNOWLEDGMENTS

"One original thought is worth a thousand mindless quotings."
Diogenes

In retrospect, deciding to work and write my thesis abroad was a bold choice, even if at first it did not feel like one. Having family, friends and good food six timezones away during this period was not easy, especially considering that most of my time was spent working on this project rather than on meeting new people or cooking. However, regardless of the obtained results, it is hard to argue that it was not worth it.
While the intended goal of this work is still far away, the amount of knowledge I gained during this period is more than I expected. Before coming to the US I even had trouble running Ubuntu on my laptop, and my knowledge of ROS and, in general, programming languages was very limited; now I can say that I was able to run a robot with a RaspberryPi, running ROS 2 and custom nodes made in Python, without a graphical interface.
The level of passion and ambition of the people in the group I was part of has driven me to these results. The amount of responsibility and trust I was immediately awarded was honestly overwhelming, and, from the start, I had to make some very important choices. The goal was the only certain objective: I had the freedom to pick which robot to use, which technologies, how to communicate with it and everything else. The research group I was part of was specialized in electrical engineering, and, while they surely provided good feedback and tips, they could not help me with technical decisions.
I hope that, somehow, I managed to live up to these expectations in their eyes. Starting this project from scratch with very little practical experience was a daunting

task, and I wish that what I have achieved and the steps that I took will be useful to fulfil their vision.
When I first thought of this acknowledgement section, I was convinced that I would fill it up with some kind of inside jokes. Now, after five months away from home, spent working hard on these few papers, I fully understand why this part is usually so unoriginal. Because, even if cliche, the love that was received across the ocean from family and friends deserves to be highlighted. Surely more than my jokes. Without naming any names, a sincere Thanks to you all, physically close or distant, than never made me feel alone in this journey. From offering sofas to sleep on when I did not have a house, to long calls or late night games, the possible examples are really endless. And for that I am glad. Even more than being done with this thesis.

Table of Contents

List of Tables

List of Figures

Acronyms

AoA
Angle of arrival

AoD
Angle of departure

BLE
Bluotooth low energy

DDS
Data distribution service

GPS
Global positioning system

GUI
Graphic user interface

IMU
Inertial mass unit

IoT
Internet of Things

IR
Infrared

LED
Light-emitting diode

LIDAR

Laser detection and ranging

LOS

Line of sight

PW

Powered wheelchair

PWM

Pulse width modulation

QoS

Quality of service

RCON

Room confinement

RFID

Radio frequency identification

RGB

Red green blue

ROC

Radius of curvature

ROS

Robotic operating system

RSSI

Received signal strength indication

SLAM

Simultaneous localization and mapping

UGV

Unmanned ground vehicle

UUID

Universally unique identifier

USB

Universal serial bus

VSLAM

Visual simultaneous localization and mapping

WLAN

Wireless local area network

WPS

Wi-Fi-Based positioning system

WPT

Wireless power transfer

Chapter 1

Introduction

The project presented in this thesis was developed between the 20th of July and the 5th of December at the FREEDM Systems Center in Raleigh, North Carolina. This research center is specialized in power electronics and smart grid developments. Research priorities include wide bandgap power electronics, electric transportation, modern power systems and renewable energy systems.

1.1 Assistive robotics

Human necessities have historically been the most important catalyst of robotic research [1]. Sadly, this has been highlighted during the recent pandemic [2] which caused a surge of interest in the medical robotics field [3]; intriguingly, robots could have been the perfect solution for providing suitable medical treatment and supplies without risking the lives of so many nurses and doctors.

In this increasingly aging world, providing appropriate care and comfortable life to the elders will soon become a challenge [4], which is why the field of service robotics is now gaining such traction. From path planning walkers [5] to robot therapy, used to improve the brain activity of patients suffering from dementia [6], the possibilities of drastically enhancing the quality of life and independence of senior citizens are endless, and the technology for non-invasive, easy to use robots is rapidly developing.

Among the plethora of different new high-tech applications, wheelchairs are getting a lot of attention; they are quickly becoming more and more advanced, and it is easy to understand why: only in the US the wheelchair user population has grown from 10% in the 1990s to 17% in 2010[7] and, due to the increased life expectancy, this number is only bound to grow. While most still prefer to use a manual wheelchair compared to a powered wheelchair (PW), around 75% of the

individuals that upgraded from a manual wheelchair to a PW reported improvements in their occupational performance [8]. However, even though the available research confirms that PWs empower users and positively impact their quality of life, still the majority of users rely on manual wheelchairs. One of the reasons for this phenomenon may very well be that PWs are not inclusive enough, which is especially peculiar when considering that this is a product intended the elders and people with disabilities. For instance, a common practise to charge a PW is connecting it to the utility outlet using a power adapter, defeating the purpose of making the user more self-reliant, as many PW users need assistance in this very critical task[9]. To eliminate this problem, a new charging paradigm is needed; an unmanned ground vehicle (UGV) indoor robot able to perform wireless power transfer (WPT) is proposed in this thesis.

1.2 WPT for wheelchair

Using WPT to charge a wheelchair is not a new idea, as the same research group at FREEDM Systems Center at NC State University has already developed a successful prototype [10]. The core idea is based on an autonomous PW capable of locating, reaching, and situating on a charging platform plugged into the power outlet. The platform is detected by cameras on the PW and LEDs mounted on the transmitter pad. The pad has an area of 0.75 m^2 and contains 48 coils divided into 3 different layers in a hexagonal pattern. When the wheelchair arrives on top of the pad, each coil is activated one by one and, based on the current received by the secondary coil on the wheelchair, the best-aligned one is selected to perform the charging procedure.

At a later stage, the pad was redesigned since users requested a less bulky mat, reducing the number of coils from 48 to 27, but still ensuring high efficiency and a WPT of around 800 W[11].
Although working in the right direction and providing a quality new product, users' feedback highlighted some limits of such a system:

- **Space**: when using this system, the user has to predispose a space for WPT. While Gen2 addressed this problem, making the area of the mat 0.4 m^2, the PW charging is still constrained to the exact same location.

- **Invasive modification of wheelchairs**: a PW must undergo severe modification to perform such a task. Besides adding a secondary coil under the wheelchair, it had to be equipped with sensors for autonomous localization, navigation and motion, and safety features, such as E-stop buttons, cameras or even WiFi. While this, in theory, is not a problem, it causes medical insurance companies to be unwilling to pay for such changes.

- **Perceived safety**: as we are discovering with autonomous car driving recently, letting go of the wheel and putting our lives in the hand of an algorithm, no matter how safe it is, is not an easy decision [12]. This is especially true if the user is a vulnerable individual. While more of a perceived problem than a real one, this could nonetheless be a difficult roadblock to overcome in the near future.

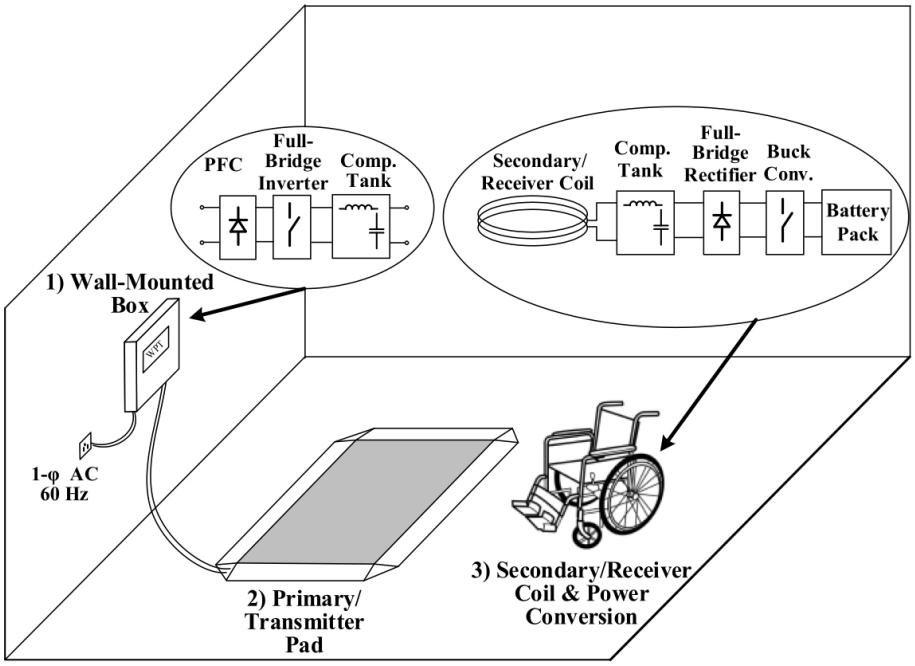

Figure 1.1: Previous research on WPT for a wheelchair.

Using a UGV performing WPT operations eliminates all these problems, and it is already proved to be a valuable option [13]. The robot could recharge the wheelchair anywhere inside the house or even outdoors. It needs just a coil on the charger mounted on top of the robot, which will align in the proper position. It does not require heavy wheelchair modifications; thus, it can readily be accepted by the user, and treated as another technological gadget, similar to other indoor robots. While limited in the battery compartment, a UGV could be designed to go back and forth from its recharging station, requiring no maintenance from the user. Future development of the final vision depends on available funds and users' interest in this new technology.

1.3 Prefixed objectives

Designing a new product from scratch with such ambitious scope requires a well-thought plan with clear goals. The discussion in this thesis will mainly focus on the robot hardware, path planning logic and communication with the user, while other team members will develop the WPT hardware. Keeping that in mind, the following goals and constraints are identified as critical for accomplishing the research goals:

- **Small dimensions**: to charge the wheelchair efficiently using a WPT, the robot must be able to fit under the wheelchair. The only way to achieve this is by approaching the wheelchair from the back. Dimensions must be as compact as possible to make this prototype compatible with a wide range of EWs. The exact dimensions chosen will be discussed in the following subsection.

- **Extra space on board**: even though the robot must be small, there should be enough space to accommodate the primary charging coil on top, so the robot's surface must be flat with no extra hardware. While not crucial in this prototype, even an internal free space is highly desirable, since it can be used to deploy more battery energy storage.

- **Path planning**: the robot must be able to reach the wheelchair and go under. In practice, this means that the robot should detect the wheelchair when nearby with precision and accurately approach it with no fault. This mechanism was called *fine detection*. When the robot is far away, high precision is not demanded; instead, the path planning algorithm should be able to bring the robot close enough to the wheelchair, so that the fine detection can perform the rest of the task. This procedure is named accordingly *coarse detection*.

- **Range**: the robot should be able to work robustly in a medium-sized apartment. To achieve this, the goal was set for the robot to reach the wheelchair from a neighboring room in the FREEDM System Center.

- **Lean design**: the robot should be easy to deploy and command by senior citizens and people with disabilities. This means that the final prototype must be an intuitive and straightforward product.

To achieve the intended goals, I had to improve my knowledge and gain skills in Python, ROS 2, and micro-controller (Arduino and Rapsberry Pi) programming. This was accomplished through research papers and other online resources, e.g, Github and the available ROS 2 and Raspberry Pi documentation.

1.3.1 Wheelchair dimensions

PWs come in many different shapes and forms [14]. Therefore, deciding which dimensional constraints should be considered was not a trivial task. The first PW considered was the model used in [10] for the first prototype of the WPT charging system, Quickie Pulse 6 power wheelchair. For the intended application, the ground clearance and minimum width of the underneath space were considered; the measured values are illustrated in figure 1.2.

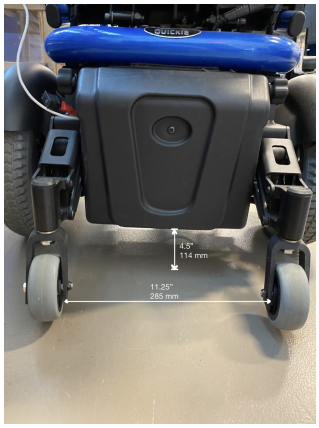

Figure 1.2: Main constraints on Quickie pulse 6.

While ground clearance was unambiguously determined to be 114 mm, the distance between back wheels heavily depends on their position, as shown in figures 1.3 and 1.4. Quickie Pulse utilizes 6 wheels: front wheels for steering direction of motion, propulsion rear wheels, and two anti-tip small caster wheels positioned on the back of the wheelchair.

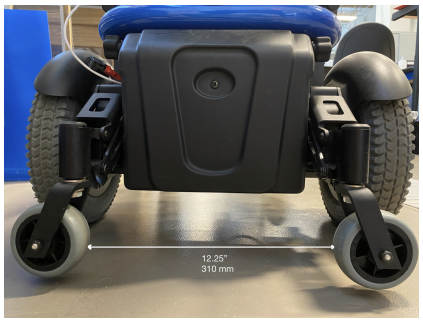

Figure 1.3: Max distance condition.

Figure 1.4: Min distance condition.

These small caster wheels are not present in most PW available on the market and vastly reduce the already minimal space underneath the unit. Therefore, Quickie Pulse 6 PW has been deemed an outlier due to the presence of such wheels and excluded from further analysis.
Unfortunately, the smallest distance between back wheels is not a well-documented measurement between PW sellers since it is crucial information that the user needs to pick his PW of choice. However, both overall and seat dimensions are commonly reported. In particular, the average seat width was estimated to be around 600 mm. At last, 420 mm was chosen as an appropriate width for the space underneath the wheelchair. While still very stringent, it reasonably estimated the space available in the average PW.

1.3.2 Mock-up wheelchair

To emulate an actual wheelchair with selected dimensional constraints, a mock-up one was built. The-mock up model was chosen to be as simple as possible, to make adding any extra hardware needed for the application easy. A galvanized metal sheet was mounted on the bottom to imitate parasitic effects during WPT in a practical PW.

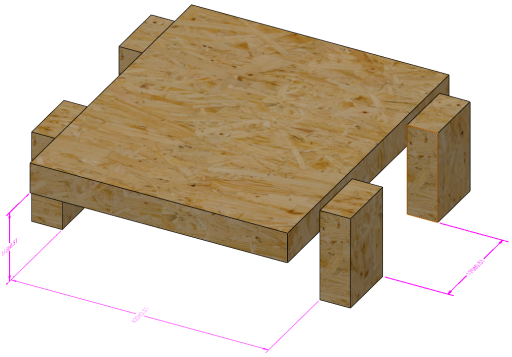

Figure 1.5: Solidworks mock-up wheelchair.

Figure 1.6: Mock-up wheelchair.

1.3.3 Robot selection based on constraints

Once all the required constraints were defined, the robot's hardware had to be chosen. Since the main goal for this first-generation prototype was to provide a good enough working basis, on top of which the future development will be constructed on, building hardware from scratch was not an option.

Previous research has shown that the Roomba platform is a perfect candidate for the rapid development of indoor mobile robots, as they provide a good balance between practicality, accessibility and low cost [15]. The Roomba platform is easy to integrate into a Python-based environment [16], which will be used extensively to program the desired behavior in this project. The Roomba has already proven to be a powerful machine, able to perform complex operations like SLAM (Simultaneous localization and mapping) after adding extra hardware [17]. Finally, the perfect form factor combined with intelligent on-board space management (especially because the trash bin and brushes are not required for the intended application) validated even further the choice of a Roomba robot.

Looking through the iRobot Roombas options one was clearly the best candidate: *Create 3*.

1.4 Create 3 dimensions and capabilities

Using the same hardware as a Roomba i3, Create 3 is a repurposed vacuum cleaning UGV. As such, it came hosting on board the sensors and actuator shown in table 1.1. It moves at around 0.2 m/s, mounting two independently driven wheels and a third caster wheel for balance. The two wheels permit the robot to move, turning at particular radius of curvature (ROC) with precision, allowing the UGV to rotate in place and navigate crowded environments relatively easily. It can carry about 9 kg on its center of mass without compromising its acceleration. It came with the traditional Roomba dock for recharging and booting up the robot. It presented a cargo bay, shown in figures 1.7 and 1.8, where the trash bag is usually hosted. The removable top plate and the cargo bay present holes to easily attach extra hardware.

Sensors	**Actuators and communication**
2x Front bumper zones	2x Drive motors
2x Wheel encoders	6x RGB LED ring
4x IR cliff sensors	1x Speaker
7x IR obstacle sensors	1x Docking port
1x Downward optical flow sensor	1x USB-C port
1x IMU (3D gyroscope and 3D accelerometer)	WiFi
1x RCON sensor	Bluetooth 5.0

Table 1.1: Sensors, actuators, and communication means present on Create 3.

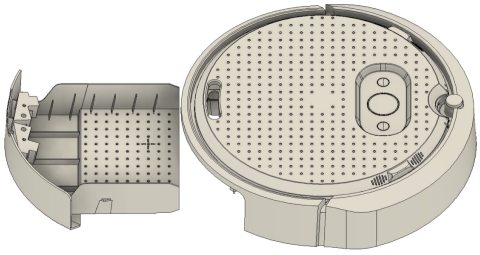

Figure 1.7: Solidworks Create 3 cargo bay model.

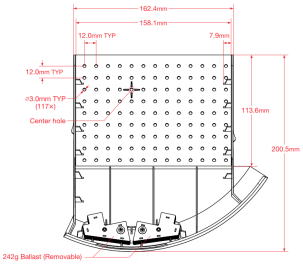

Figure 1.8: Cargo bay dimensions.

The robot's dimensions fit perfectly within the desired constraints, having a maximum height of 93.4 mm and maximum width of 336.7 mm, as shown in figure 1.9 and 1.10.

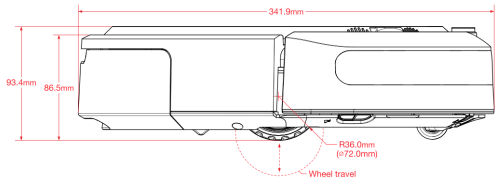

Figure 1.9: Create 3 maximum height.

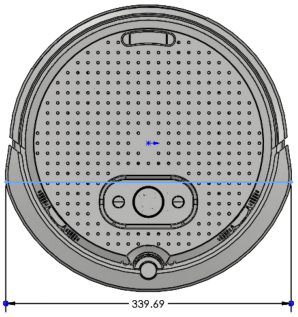

Figure 1.10: Create 3 maximum width.

The robot comes with a completely integrated ROS 2 framework. All the data collected by each sensor is published on easy-to-access ROS 2 topics, while it is possible to command the robot either by publishing particular data formats on ROS 2 topics or by calling some already programmed ROS 2 actions. A fully developed iRobot Create 3 simulator is provided, which runs on Gazebo. This was an invaluable tool for software development, especially before acquiring the robot.

1.5 ROS 2 framework

ROS is an open-source robotic middleware developed in 2007. Used in countless projects due to its ease of implementation, it is entirely based on independent executable programs called *nodes*, communicating through a publisher/subscriber

model: a ROS message is published on a *topic* by the publisher node. Moreover, every node subscribed to that particular topic receives the same message asynchronously. A node can also call another node functionality through a ROS *action*, making both algorithms run simultaneously in different scripts [18]. These nodes can be written either in Cpp using the roscpp library or in Python through the rospy library.

ROS became the industry standard since the node structure provided fault isolation, code reusability and modularity, which resulted in faster software development, especially considering the open-source nature of ROS. However, the original ROS release was imperfect, as it could not be used in real-time systems. This problem was later fixed with ROS 2, which uses DDS (Data Distribution Service) to communicate rather than the old master/slave concept of the original ROS. This makes the improved version suitable for real-time applications [19].

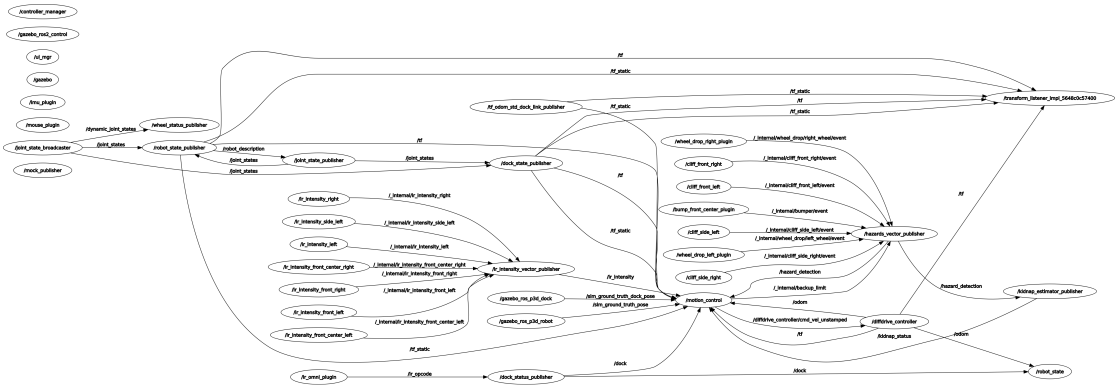

Figure 1.11: Create 3 rosgraph.

Create 3 ROS 2 graph can be seen in Figure 1.11. As it can be easily observed, due to the easy accessibility to all the different sensors' and actuators' data, there is a lot of information to unpack. The following two sections present the most useful topics and actions available on Create 3.

1.5.1 Useful topics on Create 3

- **/odom**: this topic receives information from the inertial mass unit (IMU), optical flow sensors, and wheel encoders. They are fused to produce a dead reckoning of its pose and then published. Due to the sensors on the wheels, the robot is able to distinguish if it is in slip condition or not, as well as if the robot is lifted, making this estimate accurate, especially in short applications. Position x=0, y=0 corresponds to the last position where the robot booted up, which usually corresponds with the dock location.

- **/dock**: in this topic, two crucial statements about the docking status of the robot are published, just using two booleans. In particular *is_docked* is TRUE if the robot is docked and FALSE otherwise, *dock_visible* is TRUE if the robot can detect the IR LEDs flashing at a particular rate on the dock, FALSE if not.

- **/ir_opcode**: on this topic, each time the dock's infrared (IR) LEDs are sensed, a particular code is published, which corresponds to a different combination of sensed LEDs. Each code is time-stamped and the robot's IR sensor that detected the LED is communicated, with a 1 for the single room confinement (RCON) sensor on top, or a 0 for the seven front IR sensors.

- **/ir_intensity**: information about any hazard detected by the seven front IR obstacle sensors is published here in the header field, time-stamped, with a particular identifier *frame_id*; this specifies the exact location where the sensor that performed the detection is. The raw data are also readable, a 7x1 vector reporting numbers from 0 (nothing detected) to about 2000 (touching). The first element of this vector corresponds to the most left sensor and the last one to the most right. The range of such sensors heavily relies on the obstacle material and color; overall, they are not very precise when the obstacle is distant. Accurate readings can be obtained when the obstacle is around 10 cm or less away from the sensors.

- **/hazard_detection**: on this topic, more information can be found about the obstacles detected by the various sensors. In particular, if the published integer number is 0, the robot reached its back up limit, which is hard coded on the robot as safety protocol and will be turned off; if 1 is reported the robot bumped into an object; if 2 is reported the robot detected a cliff; if 3 indicated that the robot wheels are stuck; 4 means that the wheels are lifted and, finally, if 5 is reported the robot is in close proximity of an obstacle.

- **/cmd_vel**: normally nothing is published on this topic. This topic is handy when programming a custom application on the robot since linear and angular speeds can be published here, and the information is used to activate the wheels' motors accordingly. For angular speed to turn clockwise, *angular.z* must be negative; a positive value is instead needed to turn counterclockwise. It's maximum linear speed is 0.306 m/s with an acceleration limit of 0.9 m/s^2.

- **/cmd_lightring**: this topic is used for custom applications, only. The light ring present on top of the robot is composed of 6 LEDs, and each color can be independently defined. While mostly an aesthetic feature, it is an important tool both for error fixing and improving robot communication with the user,

making it easier to understand and use.

1.5.2 Useful actions on Create 3

As seen in figure 1.12, ROS 2 actions use a client/server model. A client node sends a goal and receives steady feedback through a topic from the server, which executes the goal and sends back the result. Goal and result are communicated by means of two ROS 2 Service, which works similarly to a ROS 2 topic; in fact, the main difference between a ROS 2 service and a topic is that while a topic publishes a continuous data stream, service data is sent only if requested. ROS 2 actions are intended for long-running tasks and can be preempted.

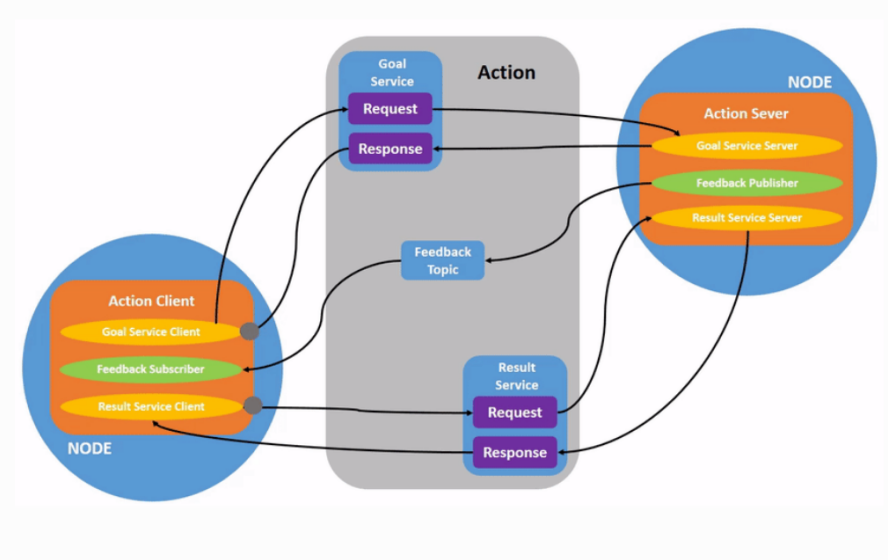

Figure 1.12: ROS 2 action work graph, taken from ROS 2 documentation.

While the user will design the action client following the correct data structure for goal requests, the action services listed below are already built-in on Create 3:

- **/undock**: this action does not require an objective; instead an empty goal must be sent to the action server to activate an undocking operation. This action will fail if *is_docked* is FALSE, which will be communicated through the result service. If the robot is docked, it will move 15 cm backward while still facing the dock, after which it will turn 180°.
- **/dock**: this action does not need a goal either. The robot will rotate 360° to check its immediate surroundings; if *dock_visible* is always FALSE during this scan the action will fail. Whenever it becomes true, the robot will move

towards the dock following the indication given by the */ir_opcode* topic. This results in a zigzag pattern until, when very close, the robot slows down significantly and moves forward, finally docking itself and ending the action.

- **/rotate_angle**: this action requires a goal composed of two numbers: angle, which is relative to the current robot heading and must be passed in radians, and max_rotation_speed, which has rad/s as a unit. If the angle is positive, the robot will rotate clockwise; if negative counterclockwise. It will saturate if a rotational speed higher than its maximum allowed 1.3 rad/s is requested.

- **/wall_follow**: the goal sent to this action is composed of two items: follow_side, which can be equal to either 1 or -1 (1 means the robot will move along an obstacle on its left, and -1 means the robot should move along its right), and max_runtime, which is again divided into two integers, sec and nanosec. When this behavior is activated, the robot will engage with the closest obstacle, moving in a spiraling clockwise motion if follow_side is 1 or counterclockwise if it is -1. After that, the robot moves along the obstacle until the timer, corresponding to max_runtime, expires. This action uses information passed by ir_intensity to detect an obstacle before bumping into it.

- **/navigate_to_position**: to request this action properly, a goal composed of two elements must be sent: position, which comprises three numbers corresponding to x, y and z coordinate, and orientation, which is then divided into four numbers corresponding to the x,y and z rotations, and the last one, w, is used as a proportional value and is usually set to 1.0. Position values are in meters, while orientation ranges from -1.0 to 1.0. Only the x and y positions and the z orientation should be passed as a goal. Positions are defined using the same value passed by the */odom* topic. The robot will first rotate, facing the goal, then move forward until the preassigned position is reached; finally, it will rotate again until its pose corresponds to the goal. If the robot bumps into an obstacle, the action is aborted.

- **/audio_note_sequence**: the robot is equipped with a speaker, and a note sequence can be played through this action. The sent goal is composed of iterations, an integer defining how many times the note sequence should be played, and the note_sequence. The note_sequence is composed of append, which if FALSE overwrites any currently playing sequence, and notes, defined by the frequency and the max_runtime, divided into sec and nanosec.

1.5.3 Nodes design approach

While most of these actions may not be useful in their current form, understanding how they operate and which information from published topics is used provides understanding to code the new proposed application, and defining clearly the potential and limits of the Create 3 hardware.

To communicate with Create 3, a ROS 2 machine connected to the same WiFi must send the command to either run a custom node, a ROS2 action or read published data using the *echo* functionality. While suitable for testing, WLAN (Wireless local area network) is not recommended [20] since the whole code would be running on a computer based on information sent by the robot; once received, the computer has to communicate back the proper response. This is not ideal, as it severely limits robot's responsiveness and risks severe faults just due to connection problems. To avoid this latency issue, code should be running on the same shared memory, avoiding different detached hardware as much as possible. A micro-controller will then be used and placed on the robot's cargo bay, connected through a cabled USB connection. Furthermore, a single node will be programmed to run all the intended robot behavior. At the same time, asynchronous data will be passed using ROS topics, either from the robot sensors or from custom-made publishing nodes. The primary node running the code will be called **detection**. All the nodes will be written using Python and the rcply library.

Chapter 2

Fine detection

2.1 Fine detection algorithm goals

Fine detection is the most critical part of the overall path planning algorithm. In fact, the most fundamental functionality that must be developed to make a UGV WPT possible is being able to detect the wheelchair and approach it following a particular pattern; the robot must be able to reach under it effortlessly and stop when required. If properly executed, the fine detection algorithm makes the requirements on all the other project segments less strict. To properly function at a commercial level of quality, this algorithm should achieve the following characteristics:

- **Robust**: fine detection must work without failure, given the appropriate circumstances. If the robot finds the wheelchair and does not manage to get under it, while there is an appropriate space in the back, the whole system fails.

- **Directional**: since the robot can approach just from the back, fine detection must be able to distinguish this side from the others. As the width constraint is stringent, the robot must approach the wheelchair straight, meaning that the orientation of the UGV and the one of the PW must coincide as much as possible.

- **Smart operation**: based on the received information, the robot must decide its course, adjusting as it gets closer. If the algorithm is advanced enough, the robot will reach the proper position regardless of the angle of approach and distance from the wheelchair.

- **Suitable range**: when close to the back of the wheelchair, the robot must execute fine detection. A proper range was defined to be at least 1 meter.

- **Minimal hardware requirements**: as the space on the cargo bay is very limited, the robot must detect the wheelchair using what is already on board. Moreover, the wheelchair itself must undergo as little modification as possible to limit the cost and unnecessary complexity.

All these requirements are already satisfied by the */dock* action. When activated, it performs precisely the same behavior as expected from the fine detection algorithm, reaching the charging dock reliably.

2.2 Dock reverse engineering

The dock communicates its position to the robot through the means of three IR LEDs, two of them positioned on the front window while the third is located on the top and it is called by the iRobot company room confinement (RCON) sensor. The two LEDs on the forepart work in a line-of-sight (LOS) mode of operation, which means they can be seen just in a cone area in front of the dock. The top LED works in a diffuse mode of operation, sending its signal in all directions around it. All three of them are sensed by the front seven IR sensors present on the robot, making this type of communication extremely pose dependent: if the robot faces towards the dock, it can detect it from 1.8 meters in a straight line distance; however, it will not see at all if turned 180°, regardless of how close they are.

Figure 2.1: Create 3 dock.

The two LEDs on the front window are positioned side by side at a slight angle. The robot differentiates between LEDs based on the LED flashing sequence; in

particular, the *green buoy* (the LED position on the right side of the dock) will flash the byte 1010 0100; the *red buoy* (the LED on the left side of the dock) will flash the byte 1010 1000; finally, the RCON LED present on the top, which is called *force field*, will signal a 1010 0001 pattern. If the robot senses more than one LED, the flashing sequences merge and becomes a new unique byte. For example, if both the red buoy and green buoy are detected, the robot will receive the sequence 1010 1100. All this information can be read on the */ir_opcode* topic, which publishes the decimal value of the detected code, as shown in table 2.1.

IR opocde	**Sensed LEDs**
160	Reserved
161	Force field
164	Green buoy
165	Green buoy and force field
168	Red buoy
169	Red buoy and force field
172	Red buoy and green buoy
173	Red buoy, green buoy and force field

Table 2.1: Ir opcodes.

2.2.1 IR communication principle

As it is easily detected and less susceptible to interference than visible light [21], IR communication has been used to pass information between different devices in many applications. Its flexibility made this technology appropriate both for simple docking applications and complex swarm self-assembling robots [22].

Typically, IR LEDs used in consumer applications have a wavelength of 960 nm. To differentiate this signal from the ambient IR light of the same wavelength, eg., sunlight or incandescent light, this signal is modulated, as illustrated in figure 2.2. The usual carrier frequency is between 36 and 40 kHz, with 38 kHz being the most common. Through the means of a band pass filter, the receiver then acquires the information and demodulates the data sent to it.
Regarding the information bearing signal, which contains the useful data sent in bits, there is no clear standard. In the case of Roomba's dock, bit 0 is described by the LED being ON for 1 ms and then OFF for 3 ms; bit is 1 instead if the LED is ON for 3 ms and then OFF for 1 ms. This means a byte is received once every 32 ms, setting the communication speed to 31.25 Hz. This is sufficient for docking

purposes, but other applications may use different conventions to achieve faster response.

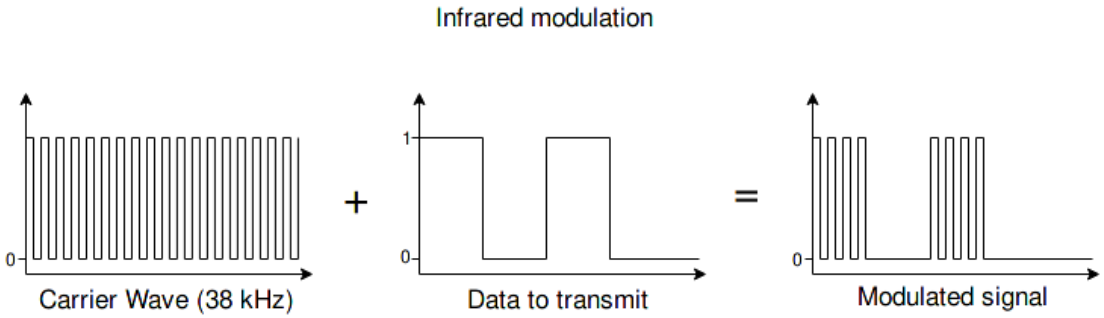

Figure 2.2: IR modulation; the image is taken from the Eclectic Koala blog.

2.3 Dock replica approach

At first, some tests had to be carried out to validate the use of a mechanism similar to the docking procedure for fine detection. The main idea is to develop hardware working similarly to the original Create's 3 dock so that the */dock* topic would recognize it. Once this is achieved, using the information published on the */ir_opcode* topic, the robot can move towards the wheelchair. Some slight differences were implemented, so that the robot can distinguish the real dock from the wheelchair's one. This method was named the **dock replica approach**.

Firstly, it was essential to demonstrate that this type of technology would be reliable, even if there is a height difference between the robot's IR sensors and the dock's IR LEDs. To implement a duplicate of the dock on the back of the wheelchair without reducing the entrance area, the LEDs could only be positioned on the side of the top plane. That increased their height from the ground from 5 cm to around 12 cm. Preliminary tests were carried out at the height of 17 cm, just elevating the dock's position: results are shown in figure 2.3 and figure 2.4. Due to the height difference, the maximum detection range increased from 180 cm to 230 cm. However, if the robot gets closer than 8 cm, it loses track of the dock.

Figure 2.3: Maximum distance of 230 cm.

Figure 2.4: Minimum distance of 8 cm.

This behavior is desirable for our application, as the goal of positioning the LEDs on the side of the wheelchair is to make the back of it visible and recognizable by Create 3, and approachable with the proper orientation. Once the capabilities of the sensors on Create 3 were established, the detection algorithm was devised.

The main idea is to write a node subscribed to both */ir_opcode* and */dock* topics, to recognize when the wheelchair is visible and make the robot approach the wheelchair slowly. The robot will use the information given by the sensed LEDs: if the opcode is 164 the robot must steer to the right; if the opcode is 168 it must steer to the left, while when 172 is detected it must go straight. Every time 172 is detected, the robot's orientation, taken from the */odom* topic, is saved. When the robot is close enough to the wheelchair, the dock replica will no longer be visible. The robot will turn first left and then right, and if the LEDs are not sensed during this motion, the robot will turn to match the last straight orientation, corresponding to the last time it could detect a 172, and go under the wheelchair. To ensure that the robot, without any more information from the dock replica, would safely reach its position under the wheelchair, information from both */ir_intensity* and */hazard_detection* will be used to prevent bumps or react appropriately, if they occur. Sounds and lights will be used during this process to communicate robot's state. Finally, to get out, the */rotate_angle* action will be used to turn 180°, followed by similar controls to avoid obstacles. The graph of this conceptual node

design can be seen in figure 2.5.

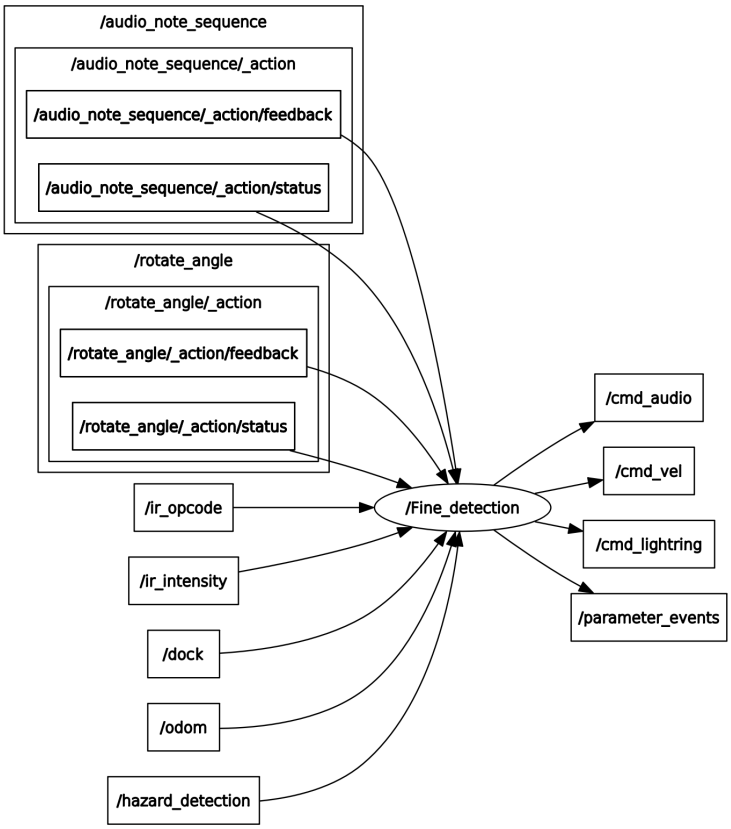

Figure 2.5: Fine detection node graph.

2.3.1 Arduino logic and measurements

An Arduino Mega was used to develop the dock replica's hardware. The circuit is shown in figure 2.6. Port 7 and 6 are connected to the two front-facing LEDs, the red and green buoys.
No field LED is used because the actual dock never flashes both buoy LEDs and RCON LED simultaneously, but rather alternates between them. That means that, even if very close, the */ir_opcode* will never register 173 solely. Instead, an alternating sequence combining 161 and 172 is registered, as shown in figure 2.7. The robot is then programmed to determine if it is seeing a dock or not, based on this interpreted opcode; so, if 173 is detected the value of */dock_visible* in the */ir_opcode* topic is still false.

While making the field and buoy LEDs flash alternatively would be easy, the intended implementation would not benefit from the field LED. Moreover, there is no physical space to put a diffuse LED on the bottom of a wheelchair. Another very important consequence of this choice was that the dock replica and the actual one can now be easily differentiated: if, at any point, while following the intended trajectory, a 161 opcode is detected, it can only mean that the robot is approaching the actual dock; if not, the robot is approaching the wheelchair.

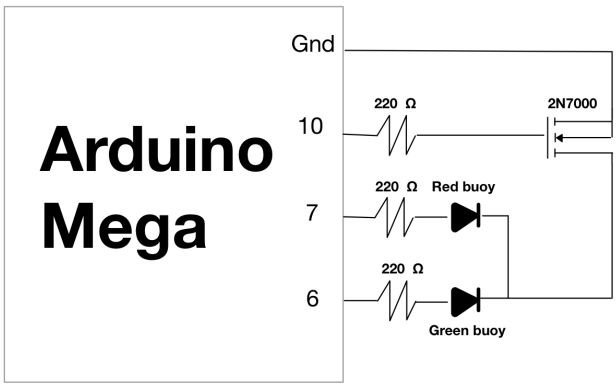

Figure 2.6: Dock replica circuit.

Figure 2.7: Comparison between the actual dock opcodes and the dock replica mounting three LEDs.

To build the dock replica, the Arduino Mega was set up in the following way: port 7 and 6 are used as digital output ports, connected to the LEDs; the LEDs

are then connected to a 2N7000 Mosfet, which has its gate connected to port 10, that will be powered making use of pulse width modulation (PWM) present on the Arduino Mega, and its drain to the ground.
As resistance 220 Ω was picked, since Arduino Mega has 20 mA as maximum current and supplied around 5 V as an output, while the used Gikfun IR LEDs support a maximum current of 30 mA and needed at least 1.2 V to work properly, and the 2N7000 mosfet has a maximum Vgs of 3 V. Using a relatively low resistance ensures that the brightness of the IR LEDs is close to their maximum potential, which in practice implies a higher range.

The Arduino code is relatively simple. Since a bit 1 is defined as 3 ms ON and 1 ms OFF, while a bit 0 is 1 ms ON and 3 ms OFF, the first two vectors are defined, each containing 32 numbers. Each bit is described by four numbers.

```
int red[] = {1,1,1,0, 1,0,0,0, 1,1,1,0, 1,0,0,0, 1,1,1,0, 1,0,0,0,
    1,0,0,0, 1,0,0,0};
int green[] = {1,1,1,0, 1,0,0,0, 1,1,1,0, 1,0,0,0, 1,0,0,0, 1,1,1,0,
    1,0,0,0, 1,0,0,0};
```

If this is translated in bits it corresponds to 1010 1000 for the red vector, which is 168 in decimal, and 1010 0100 for the green one, which corresponds to 164.
To make the LEDs flash with the correct timing a simple for loop was used.

```
void loop() {
  for (int i = 0; i < 32; i++)
  {
    digitalWrite(7,red[i]);
    digitalWrite(6, green[i]);
    delayMicroseconds(1000);
  }
  delay(100);
}
```

Between each instance there is a 1 ms delay, while, when it finishes, there is a delay of 100 ms before starting again. Port 7 and 6 are then turned ON or OFF, according to the predefined vectors. While this reduces the speed of the communication to about 7.5 Hz, it ensures the correctness of the information as well, avoiding completely the possibility of a missed signal if in the correct range.
Finally, port 10 was defined to flash at 38 kHz, as it can be seen in figure 2.8, modulating the LEDs signal and acting as a carrier wave. In figure 2.9, the signal for the green buoy can be observed, first just the information bearing signal, without the use of a mosfet, then modulated.

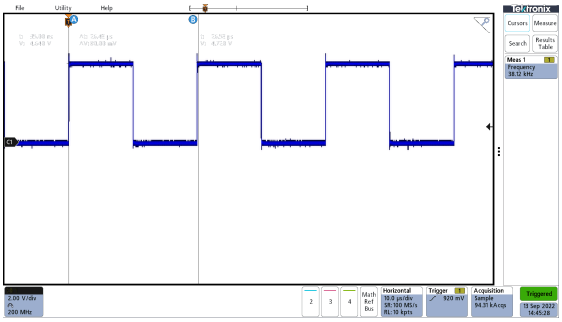

Figure 2.8: Output of pin 10.

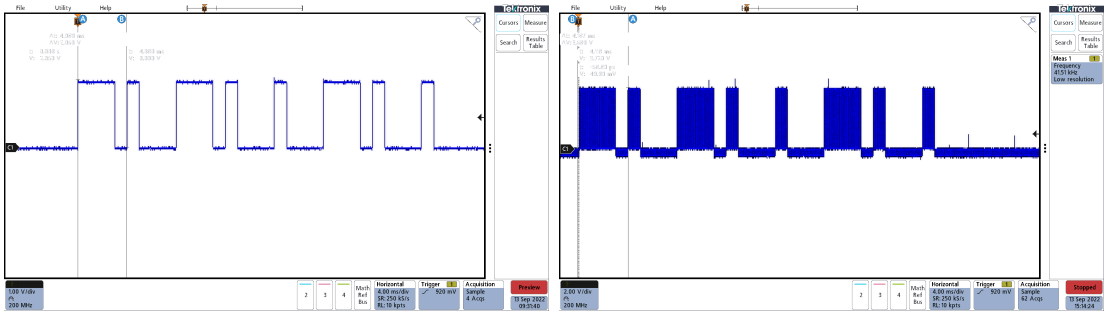

Figure 2.9: 1010 0100 signal, first without the carrier wave, then modulated.

2.3.2 Board design on mock-up wheelchair

The circuit was built on a breadboard, attached utilizing the sticky surface present on the back. It was positioned in the center of the back side top surface, as it can be seen in figure 2.10. The Arduino was powered by a power bank and positioned on top, all the wiring was made through the use of jumpers. The IR LEDs were positioned exactly at the center of the breadboard, 2 cm apart from each other and at a slight angle, around 5°. They face forward, with a 10° tilt downwards, and they stand at 11.5 cm from the ground.

During the assessing process of this design, it proved to be solid enough except for a single problem: due to the IR LEDs lack of directionality, working in a diffusive way, the robot would detect an opcode of 172 even from the side. While inherently not a problem, it severely limits the functionality of our algorithm: in fact, it relies on the last 172 detected, before losing sight of the dock, to occur when the robot is positioned straight, facing the dock and in the middle of the two IR LEDs. To fix this problem, some shielding was introduced on the LEDs, as it can be seen in figure 2.11.

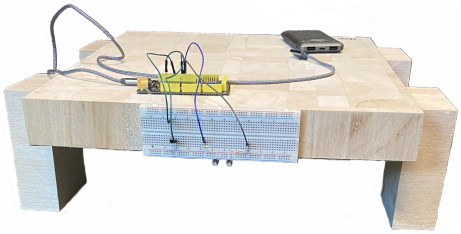

Figure 2.10: Dock replica on wheelchair mock-up.

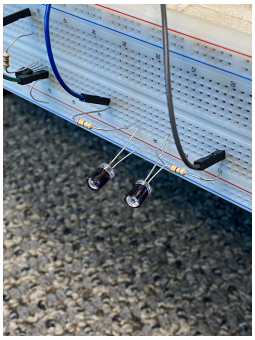

Figure 2.11: IR LEDs design on dock replica.

To prevent even more this problem, a flap was designed and 3D printed to fit in the breadboard holes. When mounted in the middle of the 2 IR LEDs, it avoids that a 172 code is detected when close and on the side, while still maintaining the same functionality as before when further away.

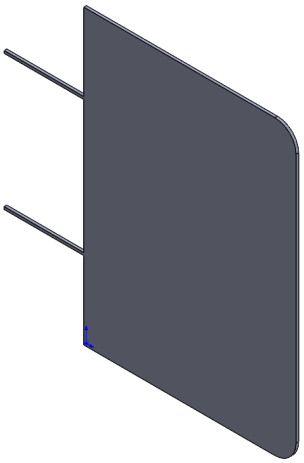

Figure 2.12: Solidworks model of the flap.

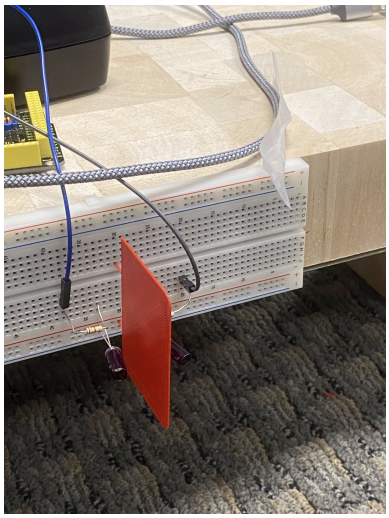

Figure 2.13: Mounted flap on breadboard.

Although surely a temporary configuration, it made testing easy and it is functionally a close representation of the behavior of the intended final product.

2.3.3 Range discussion

Finally, tests were carried out to define the range of such a configuration. Firstly, the side ranges were defined, positioning the robot as far on the left and right of the wheelchair, while still maintaining the sight of the dock. Results are shown in figure 2.14. Unsurprisingly they were almost the same, with the robot being able to detect the dock on the left side at a 50 cm distance, at an angle of 39°, and on the right side at a distance of 45 cm and an angle of 41°.

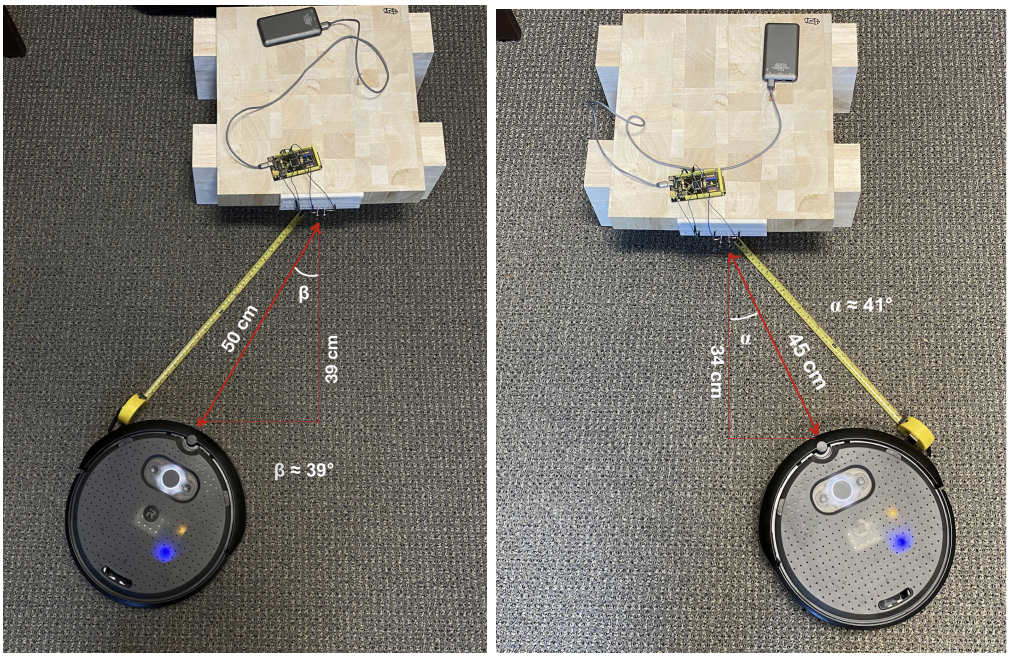

Figure 2.14: Side range limits of the dock replica.

From the very front instead, maximum range was measured to be 140 cm, as it can be seen in figure 2.15. While these values are slightly less than the one from the original dock, they could be easily extended by picking brighter LEDs; still, during most of the development, the range was deemed appropriate for the application. The final objective of fine detection was to detect and approach the wheelchair when close enough, especially if in front of it: 1.4 meters is both far enough to make the coarse detection stage not too precise, and close enough to ensure that, when detected, using the developed algorithm the robot reaches under the wheelchair with no faults. In fact, 1.4 meters is a good threshold to pass from a more focused on obstacle avoidance path planning to a more precise and accurate one. The detection zone can be seen in figure 2.16.

During testing on the coarse detection algorithm, this range was enhanced greatly by just changing the LEDs. In fact, despite the initial evaluation, increasing the

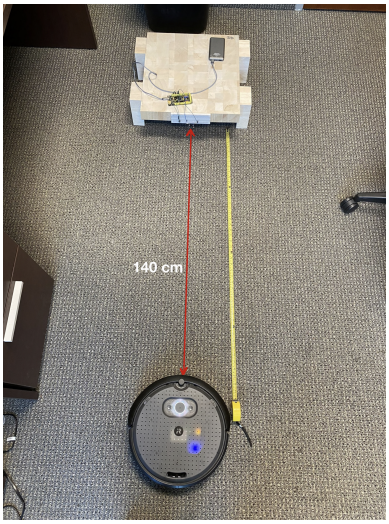

Figure 2.15: Dock replica front range.

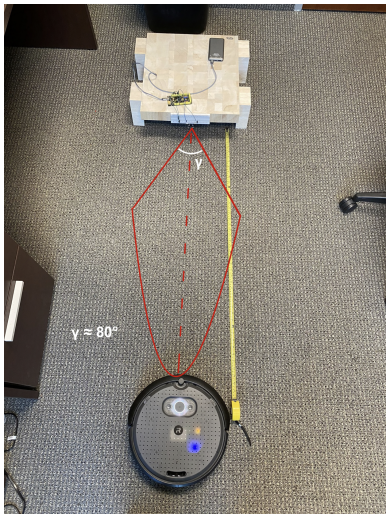

Figure 2.16: Dock replica zone of detection.

range to 5 meters resulted in a more robust algorithm. To make this possible, some obstacle avoidance protocols had to be implemented in the fine detection stage as well.

Chapter 3

Coarse detection

3.1 Indoor localization principle

While the objectives for fine detection were very specific and strict, requiring a more refined algorithm, the intended final goal for the **coarse detection** is apparently less ambitious: the robot must reach the wheelchair closely enough to engage in the fine detection procedure in any way possible.
In reality this is a much harder problem compared to fine detection, for a variety of reasons, namely:

- **Spatial constraints**: space is the most limiting parameter for the UGV WPT. The most refined and precise algorithms for indoor positioning rely on precise sensors with high range, such as cameras or LIDARs; but, on Create 3, space on top will be used to mount the primary coil for WPT and the front is already occupied by the seven IR sensors, which have a range of just 15 cm. Not only that, but most algorithms are computationally heavy, and, since the robot is designed to be self-reliant, cloud computing must be avoided. Powerful micro-controllers are too large for our application, limiting the amount of free space on the cargo bay too much or not fitting at all.

- **Unpredictable environment**: coarse detection must work at a range of about 15 meters in a standard flat. However, there is no such thing as a standard flat, as even the same house will present very different environmental challenges at different times of the day, such as: signal reflection due to walls and furniture, fast obstacle movements due to the presence of people or pets, signal scattering or high attenuation due to high number of objects cluttered in a small space. Still, indoor localization requires high accuracy and precision due to the relatively small coverage area [23].

- **Lack of generality**: while outdoor positioning has found in GPS (Global

positioning system) a standard highly efficient and inexpensive approach, indoor positioning is still an open problem. In fact GPS does not work reliably indoors, especially in apartment buildings, since it requires LOS between the device that needs to be detected and the satellite [24].

- **Heavily situation dependent**: the lack of a standard method has led to a huge variety of different hardware and software solutions to make indoor localization possible. While some are more developed than others, there is, still to this day, no clear best method, since each approach has different cons and pros, and it can be more or less preferable depending on the intended application.

For these reasons, the choice of the most fitting procedure was crucial. The most common methods currently used for indoor application are:

- **SLAM**: undeniably the most popular indoor localization algorithm used in robotics, it can fix the position of a robot while simultaneously constructing the map of the environment. Due to its popularity, many different SLAM methods were developed over the course of the last 20 years. Generally speaking SLAM works by combining an IMU with a precise sensor to detect obstacles, usually either 2D or 3D LIDARs [25], which are very efficient and computationally light, or cameras, which in the last 10 years have gained more and more traction, creating a branch of SLAM called VSLAM [26]. Even if VSLAM methods are more computationally demanding and slow, as the mapping of a medium sized apartment could take several hours compared to about 20 minutes required by LIDAR SLAM, it is considered more future proof: someday in the near future machine learning algorithms for visual recognition of objects could be implemented to augment the applications of VSLAM.

- **RFID**: Radio Frequency Identification is another very common method, as it is heavily employed in shops as anti-theft. While readers are relatively bulky, tags are encased in a compact plastic shell. Tags are usually passive, which means they have no battery; using a tiny fraction of energy sent by the reader, making use of a resonant circuit, they are able to communicate their unique ID. While passive RFID tags are used primarily for detecting proximity, working at a range of around 10 meters, active tags can be detected at a maximum range of 100 meters. Using RSSI it is possible to estimate the distance to such tags, even though not suitable for sub-meter accuracy.

- **WPS**: Wi-Fi positioning system is based on Wi-Fi technologies. This method has the big advantage of leveraging the widespread distribution of Wi-Fi access points, making the cost of implementation practically null for any

Wi-Fi employing technology. For this reason smartphones already use this technology as a substitute for GPS when using map applications indoors. Localization is computed as a fusion of RSSI, which estimates the distance from detectable routers with known position, and fingerprinting, which makes use of a database of previously tested positions where the list of accessible Wi-Fis and their signal strength are collected.

- **BLE**: Bluetooth Low Energy is becoming very popular in the last few years, with the newest smartphone coming out having integrated either the protocol iBeacon, patented by Apple, or Eddystone, patented by Google. These different types of protocol do not change the functionality of BLE, which is just a highly efficient, low power and secure method of communication based on Bluetooth, as the name suggested. Beacons and Tags are based on BLE and have relatively no difference in the way they function, as they just broadcast small Bluetooth packets, utilizing radio waves at frequencies between 2.402 GHz and 2.48 GHz, containing all the needed information relative to their application. The difference is more for marketing than anything else: in fact, while beacons are intended to be fixed in a point and send information to a moving receiver, such as a person using a phone in a public environment (airports, museums, shops, etc...), tags are tuned to be small and are used to retrieve the position of a moving object or pet. Airtags, Samsung tags and Tile tags are all based mostly on BLE. Other than the very low power consumption, as some tags can work up to three years without changing the battery, the security and potential range are the main selling points: a beacon/tag does not receive any information, which means that a receiver cannot be tracked in any way. Moreover, Bluetooth protocol can avoid connecting to personal tags without permission, and the range can reach up to 200 meters based on the transmission power of the sender[27].

- **UWB**: Ultra Wide Bandwidth is currently the most precise indoor localization wave based method, achieving a localization error of just 20 cm. Due to this high resolution it is commonly used with multiple antennas, to not only communicate the distance through received signal strenght (RSSI), but also the direction. Using very fast pulses, as the carrier wave has a frequency of 2.5 GHz, and a large bandwidth, of about 500 MHz, this way of communication is able to send data at a very fast rate while also being less susceptible against interference and fading. Due to this very large bandwidth it is actually banned in some countries, such as Russia, Pakistan and Argentina, due to the possibility of obstructing military communication. It is most commonly used in combination with other longer range communication methods, such as BLE and Wi-Fi, to give a more precise position estimation when close. Airtags and Estimote beacons, for example, use both BLE and UWB.

SLAM is clearly the best method for navigation, but, if a LIDAR is used, it would be difficult to identify the wheelchair by just using this type of sensor data, as, even if highly efficient, it is tailored to just identify the presence of obstacles; using VSLAM, instead, would require expensive and larger than feasible micro-controllers for our application. Even if, on a new generation of the prototype, space could be found to mount a LIDAR, another additional method needs to be implemented for wheelchair recognition.

Some wave based methods had to be used, and BLE tags were chosen. They provided an inexpensive option and are easily available on the market, while also being capable of a good enough range for a medium flat, even considering the many non-idealities of such an environment. Precision was initially estimated to be good for our application, as the objective of coarse detection is just to get in range of fine detection, which is 5 meters from the back of the wheelchair.

3.2 BLE technologies

When Bluetooth 5.0 was released, in July 2016, BLE beacons and tags truly became a reality. In fact, while some implementations were already tested using Bluetooth 4.2, with the advent of Bluetooth 5.0, the speed of communication was more than doubled, going from 24 Mb/s to 50 Mb/s, and, most importantly, the range was increased four-fold, from a maximum range of 60 meters to 240 meters.

All Bluetooth 5.0 beacons work using what is called **RSSI**, which stands for received signal strength indication, to estimate distance following the relation reported in equation 3.1. The **TxPower**, which can also be called Calibration power and is measured in dB, defines the signal strength that a receiver would detect at a distance of 1 meter from the beacon. This value can usually be set up, in order to have longer range in exchange for battery life. The **n** factor is called **Free space factor**: it is an environmental value which is set up during testing, and it varies from 2, when the connection is very weak to 4, indicating a very strong connection.

$$d = 10^{\frac{txPower - RSSI}{10 \cdot n}} \tag{3.1}$$

Beacons and tags are usually capable of using different communication protocols. They are mostly equal, as they all transmit the same information but with different standards. For our application the iBeacon protocol was used, where data is formatted as shown in figure 3.1.

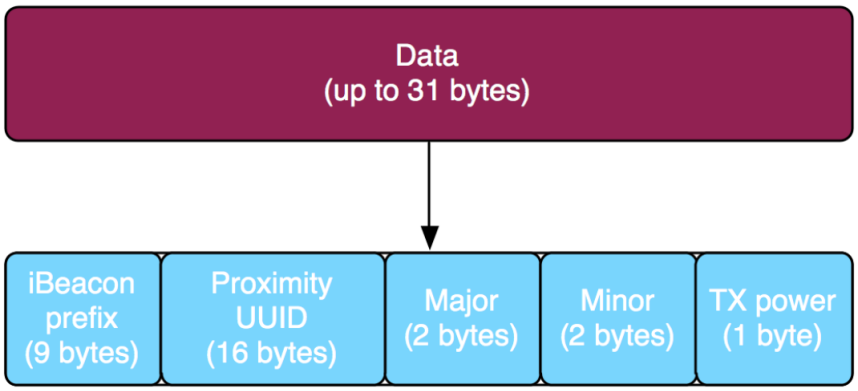

Figure 3.1: iBeacon format, image taken from Bleeks documentation.

The first 9 bytes are used to communicate the iBeacon prefix, which defines the type of protocol and, at the same, communicating that the signal is used just for broadcasting and not for connection. The proximity UUID (universally unique identifier) corresponds to the name of the beacon, composed of 16 or less hexadecimal characters. Some UUIDs are used by companies to identify their products, meaning that all the beacons produced by such a company have the same UUID. To distinguish between them, major and minor information is used, which are two unsigned integers, having values from 0 to 65535. This ensures that each beacon has a unique combination of UUID, major and minor values. Finally, the TxPower is communicated in the final byte. The RSSI is defined by the receiver.

While RSSI is an appropriate and flexible method, as it is currently used by most radio wave indoor localization methods, it has some clear faults. In fact, as we can see from equation 3.1, this method is heavily reliant on the space free factor which can change easily, even in the same environment, depending on the amount of interference due to other radio waves or obstacles. Not only that: with this method just distance can be estimated. In order to get direction as well, three beacons or more must be used, applying triangulation or trilateration techniques.

This problem was solved when the more recent Bluetooth 5.1 came out, in January of 2019. While the distance is still estimated using RSSI methods, the direction can be defined by using multiple antennas, as it can be seen in figure 3.2. This is actually very similar to the way UWB is able to tell direction. When working with Bluetooth 5.1 just a single device needs to have multiple antennas, as both AoA (Angle of Arrival) or AoD (Angle of Departure) methods can be used in combination with wavelength and phase of the received signal to estimate the

exact direction. With Bluetooth 5.1 it is possible to pinpoint the exact location of a transmitter or receiver with an error of just a few centimeter.

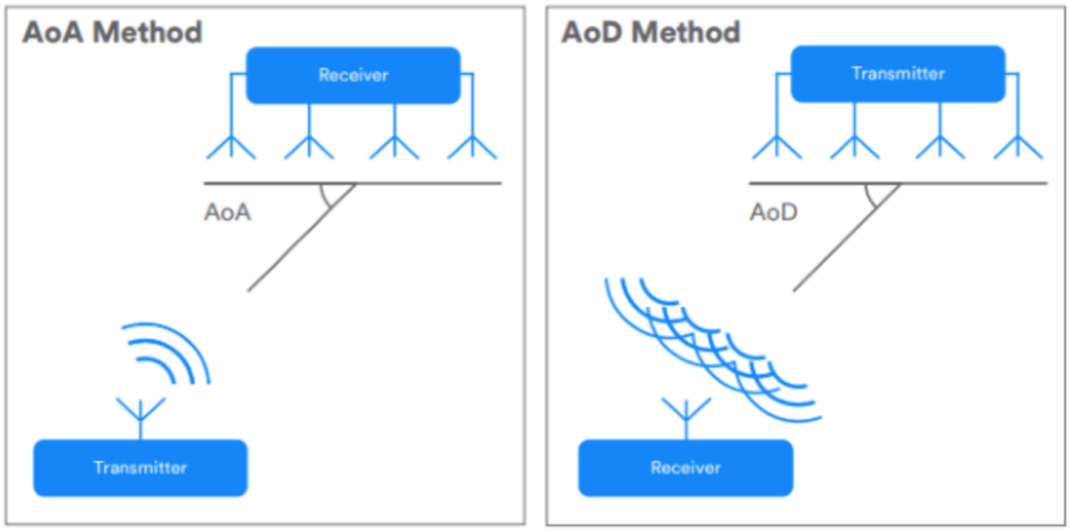

Figure 3.2: Bluetooth 5.1 AoA and AoD method, taken from howtogeek blog.

Still, currently beacons working with Bluetooth 5.1 are just in the development stage and not available on the market.
BLE beacons based on Bluetooth 5.0 will be used for our application, but, as the BLE technology will develop during the following years, the functionality of our UGV could be easily augmented by switching to Bluetooth 5.1.

3.2.1 How BLE can be used as a foundation for coarse Detection

BLE beacons have already been used for robotics to find small objects in a known environment. Either by using multiple receivers[28], or multiple beacons, the application of triangulation or trilateration techniques allows to locate objects with accuracy of about 3 meters. The use of multiple beacons in a known environment is becoming increasingly popular in hospitals [29], where tracking the position of nurses or particular medicines/equipment could save lives.
Our implementation differs from these applications, since the robot should be designed to work in any flat; so, an a priori map can not be considered, and set up should be minimal. All the testing and results obtained were done by making use of just 3 beacons, with one positioned on the wheelchair. This number can be considered very low, as to achieve good results, in a relatively small space, more than 20 beacons with Bluetooth 4.0 were used in conditions similar to ours in

previous research papers[30].
Anyway, the first step was to pick which beacons to use. The Blue charm beacon, shown in figure 3.3, was chosen for multiple reasons: while the compact size is common among modern tags, what set this beacon apart was the presence of a button, as shown in figure 3.4, and the ability to design its functionality as well as other important features using the Blue charm app.

Figure 3.3: Blue charm beacon.

Figure 3.4: Beacon's button.

If the beacon is placed on the wheelchair, it is possible to use a receiver on the robot side to detect the distance between the two using RSSI. While the robot is capable of Bluetooth 5.0 natively, its use is limited to either communicating with the iRobot coding app for smartphones or the iRobot web server. Not only that, there is even an extra caveat as the Create 3 adapter board, which is shown in figure 3.5, can either receive information through BLE or USB. This can be configured by a little switch on the board itself; but, if somehow BLE from the robot is used directly to receive information from the beacon, it means that no other hardware of any type can be connected with the robot.
Rather than relying on this option, which severely limits further development, it was decided to make use of a micro-controller attached and powered over a USB-c cable, as modern micro-controllers are equipped with Bluetooth 5.0 receivers. Another benefit of this approach, as mentioned in Chapter 1, is that it makes possible to run a program through USB communication, which is significantly less error prone when using ROS 2, as well as providing different possible connections with new devices for future development.

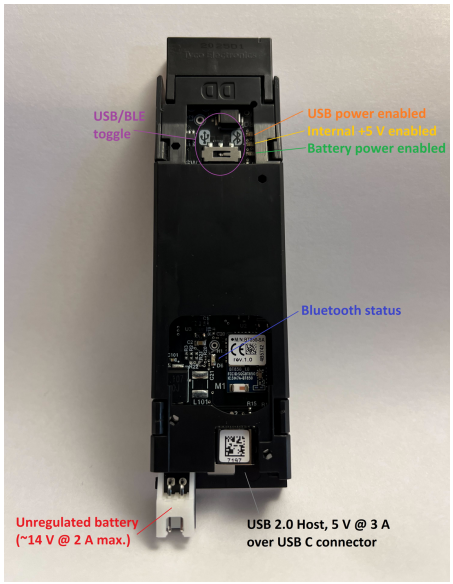

Figure 3.5: Create 3 adapter board, from iRobot documentation.

3.3 Raspberry Pi 4 deployment

Raspberry Pi 4 was the micro-controller of choice, as it satisfied all the needed requirements in a 56.5 mm by 85.6 mm figure and at a low price of 40 $. In fact, while being one the most inexpensive micro-controller, the ease of integration with a Linux OS (operating system) and a ROS 2 environment has made it one of the most popular hardware, not only for robotics application but for any IoT (internet of things) implementation. Considering also the extensive documentation available [31], as well as the integration of common hardware, e.g, monitors, keyboards and USB sticks, and the presence of Bluetooth 5.0 and Wi-Fi capabilities, it was the perfect candidate for our application.

The RasbperryPi was configured with Ubuntu 20.04.5 as the robot, which now with more recent software updates can also run the latest ROS 2 version Humble, was running ROS 2 Galactic, which is not available in the latest Ubuntu release. No GUI (graphical user interface) was installed, to avoid wasting computational resources on non required functions. Just a monitor, a keyboard and the command line were used to program our application.
To make testing and the robot work in a smoother way, extra effort was put in making the application continuously running, using the two extra buttons present on the face of the robot to each side of the power button as well as other external inputs to interact with the code.

Our micro-controller was installed on Create 3 by four plastic M2 hand screws and nuts, making use of the holes present on the cargo bay. As it can be seen in figure 3.6, by making use of a 10 cm USB-c cable, just about 25% of the space on the cargo bay was used, causing no changes on the external figure of the robot.

Figure 3.6: Raspberry Pi 4 installation on Create 3.

3.3.1 DDS protocol

When running simultaneously more than two ROS 2 machines, multicasting can be problematic if not addressed properly. Multicasting can be described as a communication between a single sender and multiple receivers, and it is one of the key improvements that distinguish ROS 2 to ROS. Making use of DDS protocol, which is employed as a middleware to deal with the transportation of information between multiple nodes, it supports the so called "distributed discovery", rather than the old centralized master used on the original release of ROS: each message is not anymore discovered by a central node and then sent to each receiver but rather discovered by each receiver independently.

ROS 2 supports a plethora of different DDS implementations, as each has their pros and cons, but Create 3 supports just two of them: CycloneDDS, which is the default one, and FastDDS. When using multiple network interfaces, such as Wi-Fi and USB connection, multicasting can fail, as the distributed discovery can be limited to a single network by default. This is exactly what happened in our implementation: when using Raspberry Pi 4 to run our code on Create 3, connected via USB, it was impossible to read information from the robot's sensors by using a ROS 2 computer, connected to the same Wi-Fi as the robot. This caused problems during testing, and it was due to CycloneDDS. It was then decided to switch to FastDDS, as it supports multiple networks by default while also providing a faster rate of communication.

Changing DDS protocol has obviously other implications, as another characteristic of this middleware is the control available to quality of service (QoS) options: it is possible to set parameters, such as the reliability and buffer size of sent or received messages, and the behaviour is slightly different depending on the chosen DDS. This will be later discussed in paragraph 4.1.

3.3.2 Parameters tuning

Once communication was established, the Free space factor n from equation 3.1 needed to be tuned to make accurate estimation. To do so, firstly $TxPower$ or measured powered value needed to be verified: in fact, the given -63 dB by the iBeacon last byte was a precise enough estimation when working on a laptop, using Bluetooth 4.0, and the Blue charm beacon; this value proved to be incorrect when detecting the beacon with the Raspberry Pi inside the robot. In fact, due to Bluetooth 5.0, the signal at 1 meter was received at a much higher power of **-55 dB**.

Once that value was fixed, to tune parameter n equation 3.2 was used.

$$n = \frac{\sum(RSSI - TxPower)}{10 \log_{10} d_{mes}} \tag{3.2}$$

All the measurements were made in the same room: the Raspberry Pi would be positioned at a given measured distance d_{mes} from the beacon, and each estimation of the n parameter would be calculated using 100 different measures for $RSSI$. This procedure was repeated 5 times for each distance and the results are shown in table 3.1.

Distance	Measurements for n
0.3 m	3.58, 3.77, 3.89, 3.79, 3,84
1.45 m	4.33, 3.65, 3.77, 3.22, 4.08
2 m	2.32, 2.14, 2.28, 3.23, 2.80
3 m	2.80, 2.80, 2.63, 2.66, 2.84
4 m	2.60, 2.59, 2.52, 2.30, 2.35
5 m	3.40, 3.27, 2.96, 3.07, 3.09
6 m	2.95, 2.88, 2.58, 2.54, 2.63
7 m	2.45, 2.59, 2.46, 2.20, 2.18
8 m	3.06, 2.84, 2.61, 2.72, 2.73
9 m	2.36, 2.42, 2.46, 2.34, 2.41
10 m	2.60, 2.78, 2.70, 2.79, 2.61

Table 3.1: Measurements for Free space factor.

Remembering that the n parameter should vary between 2, for a weak connection, and 4, for a strong one, some considerations could be clearly stated: distances close to 1 meter are not good for tuning, as the logarithm tends to zero, making the overestimation of n common. Also, smaller distances could have a bigger relative error with the measured distance, and, due to the resolution of $RSSI$, which is measured as a whole number, small changes may result in big differences in the n estimation. For these reasons, just the measurements from 3 meters on were considered. Taking an average of all the calculated n from that distance on, the most accurate n estimation can be found, equal to **2.67**.

The found model is described by the graph shown in figure 3.7. Even if properly tuned, it is important to understand that this is just a good fit for the room in which testing was conducted, and for the condition of that particular room at that particular time. Different circumstances could influence the validity of this model, and while some disturbing factors can be mitigated, such as interference caused by the presence of too many obstacles, some can be very unpredictable. Still this model proved to be a solid base to evaluate the potential of BLE RSSI to estimate distance, and while the precision can vary, the accuracy is decent, as it will be

discussed in the next paragraph.

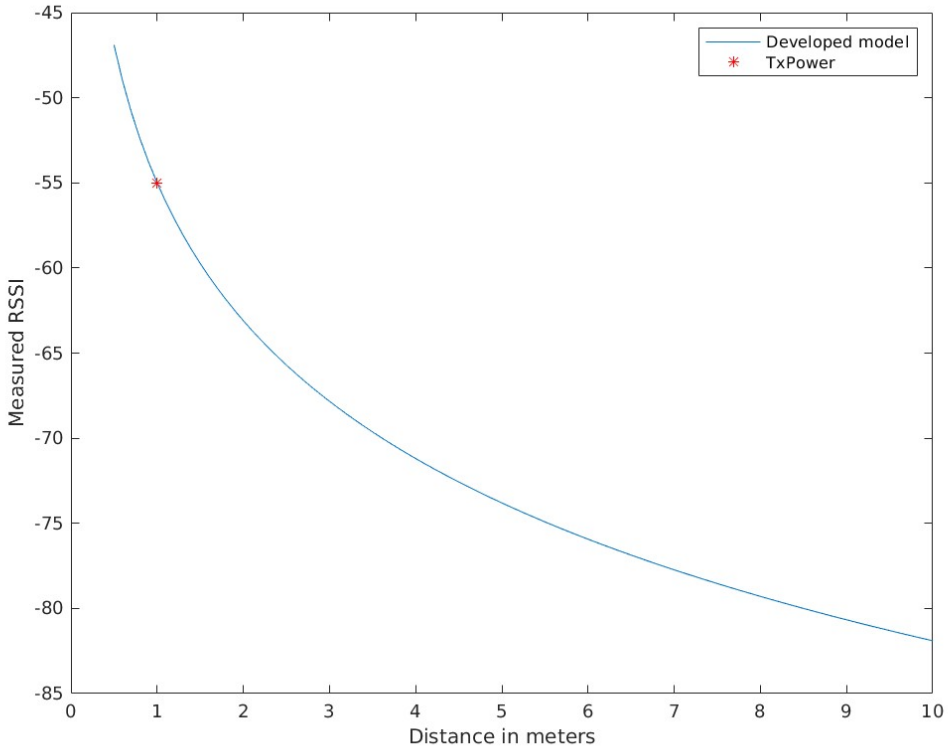

Figure 3.7: Plot for RSSI and distance from beacon.

3.3.3 BLE detection tests

To make the computed distance through the previously computed model some more adjustments had to be implemented. The developed algorithm will send a *'No beacon'* message if the beacon is inactive, and it is tuned so that just the beacon present on the wheelchair will be detected, making use of its unique UUID. Once the beacon is activated, eight distance measurements are taken and their median is stored. After four of these medians calculations the *final estimate* will be computed as an average of them.
Results for tests done at close, medium and far range are shown in figure 3.8. Although this may seem like a lengthy procedure, it is important to address that the beacon was set up to send packets at maximum available frequency of 10 Hz. This means that, if all the packets are read, a single estimate takes just 3.2 seconds, but this is seldom the case. The results shown are all taken in the same

time window of about 50 seconds. As it can be seen there is more delay when the distance increases, and even when the beacon is at a very close range a single final estimate takes on average 7 seconds.

```
No beacon
Measure 1 = 0.37874426846697995 meters
Measure 2 = 0.4410059454176737 meters
Measure 3 = 0.38271427666741836 meters
Measure 4 = 0.38271427666741836 meters
Final estimate = 0.3962946918048726 meters
Measure 1 = 0.4410059454176737 meters
Measure 2 = 0.4410059454176737 meters
Measure 3 = 0.32442260791716304 meters
Measure 4 = 0.5883240999386575 meters
Final estimate = 0.448689649672792 meters
Measure 1 = 0.37874426846697995 meters
Measure 2 = 0.40647498253455094 meters
Measure 3 = 0.3981071705534972 meters
Measure 4 = 0.35938136638046275 meters
Final estimate = 0.3856769469838727 meters
Measure 1 = 0.30864353218984336 meters
Measure 2 = 0.32442260791716304 meters
Measure 3 = 0.32442260791716304 meters
Measure 4 = 0.32442260791716304 meters
Final estimate = 0.3204778389853331 meters
Measure 1 = 0.2928644564625237 meters
Measure 2 = 0.2928644564625237 meters
Measure 3 = 0.35938136638046275 meters
Measure 4 = 0.35938136638046275 meters
Final estimate = 0.32612291142149324 meters
Measure 1 = 0.4001936558990682 meters
Measure 2 = 0.41955655798558544 meters
Measure 3 = 0.3981071705534972 meters
Measure 4 = 0.4410059454176737 meters
Final estimate = 0.41471583246395616 meters
Measure 1 = 0.41955655798558544 meters
Measure 2 = 0.4410059454176737 meters
Measure 3 = 0.4410059454176737 meters
Measure 4 = 0.4433172638527181 meters
Final estimate = 0.43622142816841275 meters
```

```
No beacon
Measure 1 = 7.742636826811269 meters
Measure 2 = 7.3660550170423775 meters
Measure 3 = 5.418781319210806 meters
Measure 4 = 11.092073432287822 meters
Final estimate = 7.904886648838069 meters
Measure 1 = 6.065612517478706 meters
Measure 2 = 6.309573444801933 meters
Measure 3 = 8.159797906360104 meters
Measure 4 = 9.039072029545189 meters
Final estimate = 7.393513974546483 meters
Measure 1 = 7.026105135806601 meters
Measure 2 = 8.576958985908941 meters
Measure 3 = 7.3660550170423775 meters
Measure 4 = 5.695810810737687 meters
Final estimate = 7.166232487373901 meters
Measure 1 = 7.3660550170423775 meters
Measure 2 = 7.742636826811269 meters
Measure 3 = 7.742636826811269 meters
Measure 4 = 8.159797906360104 meters
Final estimate = 7.752781644256255 meters
Measure 1 = 7.742636826811269 meters
Measure 2 = 7.3660550170423775 meters
Measure 3 = 6.989473207273485 meters
Measure 4 = 8.159797906360104 meters
Final estimate = 7.564490739371809 meters
Measure 1 = 7.3660550170423775 meters
Measure 2 = 7.3660550170423775 meters
Measure 3 = 6.989473207273485 meters
Measure 4 = 8.576958985908941 meters
Final estimate = 7.574635556816795 meters
```

```
No beacon
Measure 1 = 11.659144011798316 meters
Measure 2 = 13.611363271043206 meters
Measure 3 = 13.611363271043206 meters
Measure 4 = 12.41611637235745 meters
Final estimate = 12.824496731560544 meters
Measure 1 = 11.092073432287822 meters
Measure 2 = 10.525002852777327 meters
Measure 3 = 11.092073432287822 meters
Measure 4 = 12.287320330973579 meters
Final estimate = 11.249117512081638 meters
Measure 1 = 12.41611637235745 meters
Measure 2 = 12.287320330973579 meters
Measure 3 = 11.659144011798316 meters
Measure 4 = 14.382214287379988 meters
Final estimate = 12.686198750627334 meters
Measure 1 = 14.307229891937572 meters
Measure 2 = 13.754037968204724 meters
Measure 3 = 13.611363271043206 meters
Measure 4 = 13.611363271043206 meters
Final estimate = 13.820998600557175 meters
Measure 1 = 12.983186951867944 meters
Measure 2 = 11.659144011798316 meters
Measure 3 = 10.013093962979381 meters
Measure 4 = 11.092073432287822 meters
```

Figure 3.8: Obtained measurements at close, mid and far distances.

Even if this speed is not optimal, this fixed some problems that arise when using RSSI, especially at medium range. At close range, results were always accurate, regardless of the method, because due to the low distance interference was fairly improbable. When working at higher range, results were surprisingly accurate, since if the signal faced some type of obstacle it would simply not reach the receiver. At medium range, between 2 and 8 meters, things were very unpredictable. As it can be seen on the shown measurements even the medians value have significant variance, having lowest estimate of 5.4 meters and highest of 11.09 meters. Still, by taking the average of the four medians measurements, this error is lowered in the final estimation.

To fully understand why the variance is so high, further tests were carried out, but this time just raw RSSI data was collected. Results shown in figure 3.9 come from measurements carried out in a regular apartment; instead, the graph in figure 3.10 depicts RSSI value taken outside, where the influence of wave reflection and interference from obstacles should be minimal.

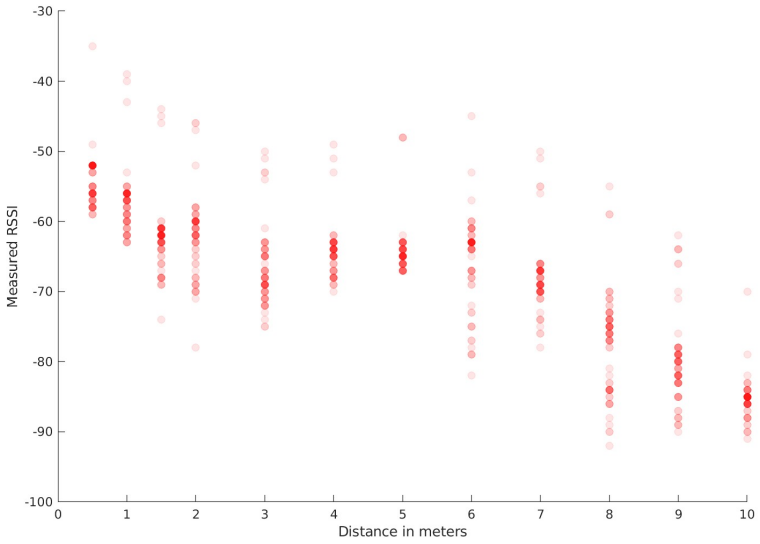

Figure 3.9: RSSI values in a flat.

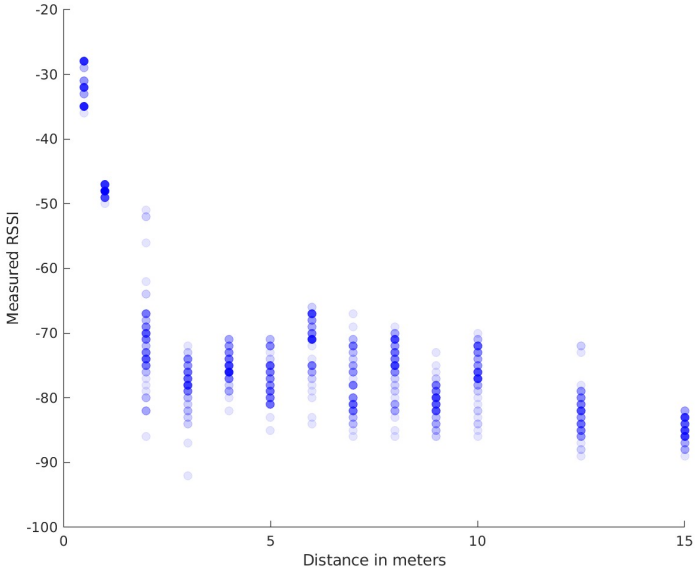

Figure 3.10: RSSI values outside.

While outside measurements show a lower average standard deviation of 3.23 in the RSSI values, compared to the 4.87 measured indoors, the precision did not have

any significant improvement. When compared to the model, the average absolute error recorded was 5.28 in a flat and 5.15 outside. But even this average values do not depict the full picture, as both show significant differences depending on the measure. It is clear that the RSSI is not a good enough method when trying to estimate distances in the range of 2 to 10 meters outdoors, and 1.5 to 7 meters indoors. In fact, RSSI values tend to be too similar in these ranges, and the trend is not even clear, as even the median values do not monotonically decrease. In figure 3.11, the median values are plotted against the computed model.

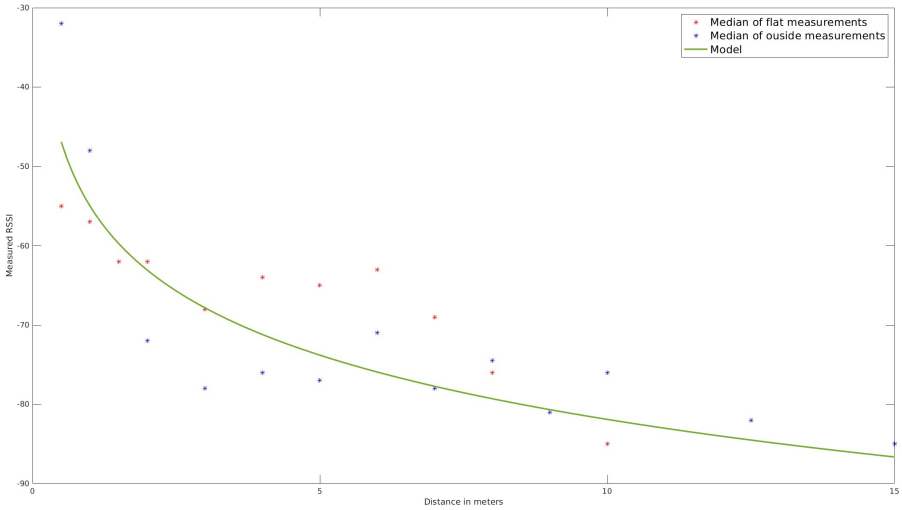

Figure 3.11: Model RSSI behaviour compared to collected data.

Independently of the correctness of the model, from the collected data the unreliability of the RSSI method is clearly depicted. Still, BLE beacons are a new technology, and more advanced ones are currently being developed, using other data such as the phase of the received wave or ToF (Time of flight). Even in its current state, BLE beacons can be used both because of the future development prospective and because RSSI estimation can be still useful. Ultimately, the goal was to the have a way to coarsely localize the wheelchair, and making use of trilaretation techniques, with some adjustments, this could be achieved. Other functionalities, not depending on RSSI, were also tested.

3.4 Room identification

While RSSI measurements proved to be somewhat unpredictable, from previous experiments one thing was clear: if the beacon is closer, data is received at a faster

rate. It may be possible to build a distance model from this knowledge, but the speed at which information from a beacon is received can vary depending on other factors, such as the speed and usage of the receiver's CPU or interference.
However, this can be an invaluable tool when using multiple beacons; in fact, even if in the medium range RSSI values tend to blur together, a closer beacon will always communicate with the receiver at a faster rate. This can be exploited to determine in which room the mock-up wheelchair is. To make use of this knowledge, a map of the environment is needed. Even if, on this first prototype, SLAM is not used, due to the lack of appropriate sensors, it will be implemented in the future. To demonstrate that room identification using this method is possible, a receiver had to be mounted on the mock-up wheelchair. While the perfect candidate would again be a Raspberry Pi 4, due to its ability to run Ubuntu and ROS, due to the current chip shortage there were none available on the market. A laptop placed on top of the wheelchair was used instead, communicating results trough Wi-Fi using ROS 2 to the Raspberry Pi 4 inside the robot. Two beacons were placed in two communicating rooms, depicted by blue cirlces in figure 3.12. The blue square corresponds to position x=0, y=0, which is the dock's location.

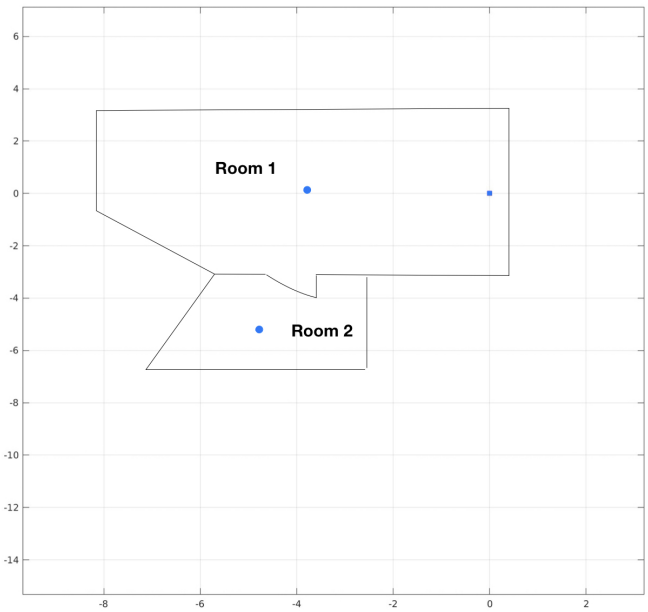

Figure 3.12: Testing environment for room selection.

The developed code is very simple: two counters are initialized to 0, one for each beacon. Once a beacon is detected, UUID is checked and the counter of the corresponding beacon is augmented by one. The first counter that reaches 35

determines in which room the receiver is. This number was picked experimentally. The room can then be communicated to the robot through Wi-Fi, making use of ROS topics.
Surprisingly, this proved to be very precise, even in the worst testing conditions, which would coincide with the receiver placed close to the opened door.

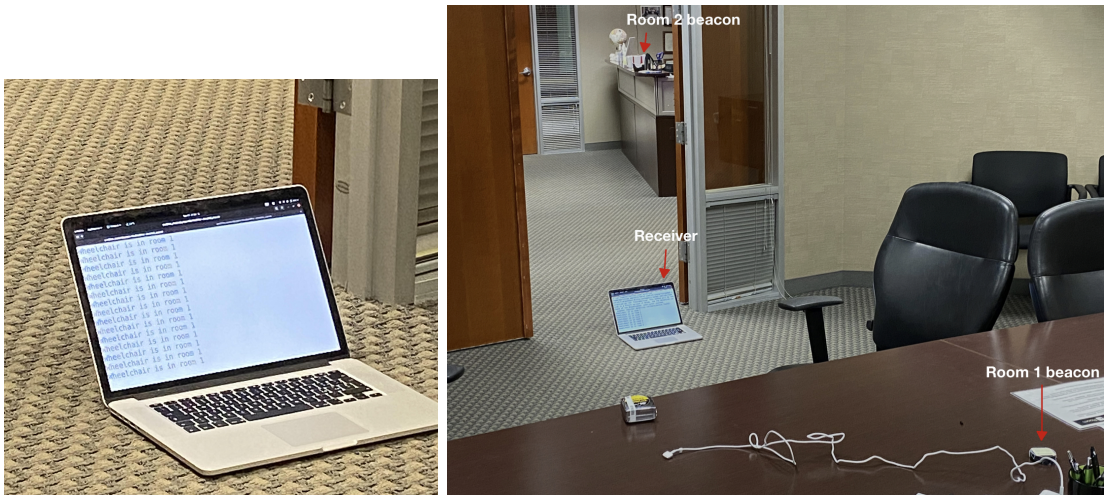

Figure 3.13: Room 1 localization.

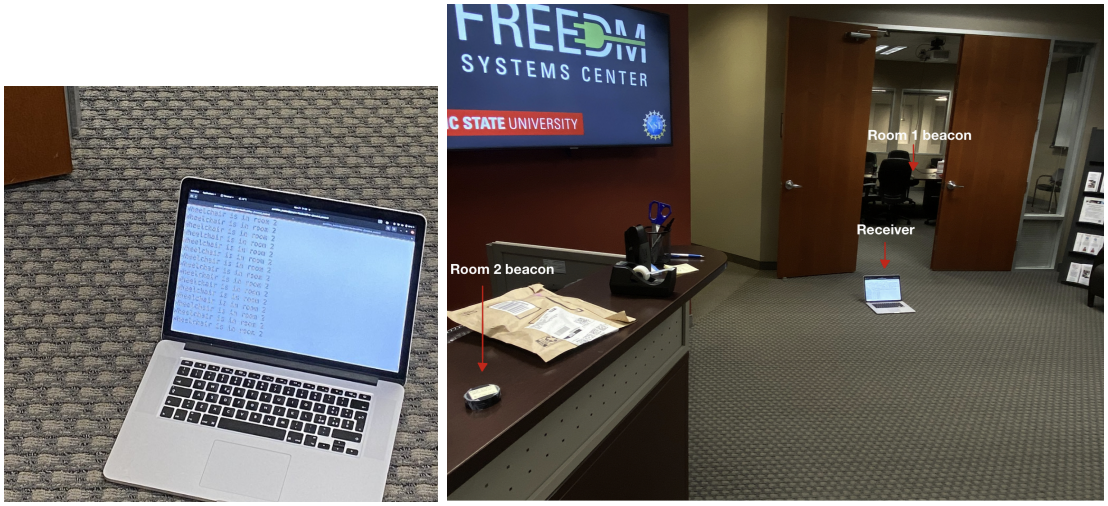

Figure 3.14: Room 2 localization.

3.5 Trilateration

Given the appropriate circumstances, a tweaked trilateration method can still be used to achieve some rough estimation of the position of the wheelchair, even considering the RSSI problems shown before. In fact, even if inaccurate at medium ranges, RSSI distance estimation can be trusted when close. When having a high number of beacons, it is possible to pick the 3 closest to the receiver, which are the 3 having the lowest median RSSI with repsect to the others, to trilaterate the position. This implies a higher cost, as a single beacon is approximately 25 $, as well as a more involved set up procedure, as each beacon position must be exactly known and registered.
In our application just 3 beacons were used to test this procedure and 2 receivers, one on the wheelchair and one on the robot. A beacon will be placed on the wheelchair, while the other 2 beacons were placed in a priori known positions, which were estimated by making use of the dead reckoning of the robot, published on the */odom* topic. When the robot returned to the dock, which should correspond to position x=0 y=0, the dead reckoning estimated a position of x=0.05 y=-0.1, which means that the error accumulate on these measurements is relatively low. Tests were conducted in what was called "room 1" in the previous chapter. The configuration can be seen in figure 3.15.

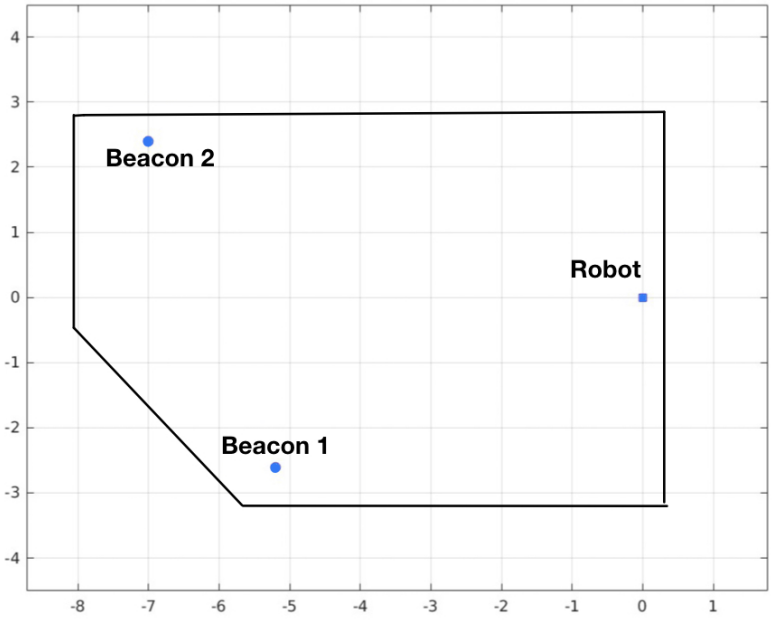

Figure 3.15: Test environment for trilateration.

To make proper trilateration possible, 3 distances are needed. In our case,

one distance could be obtained between the robot and the beacon present on the wheelchair; two more could be estimated from the wheelchair's receiver to beacon 1 and 2 and then sent to the robot. As it will be shown, in our peculiar trilateration technique, the distance between the robot and the wheelchair will have little influence.
The distances between the wheelchair and the two beacons are purposely underestimated, following the model in figure 3.16, which has TxPower = -75 dB. All distances are calculated as four medians of eight measurements, and then averaged.

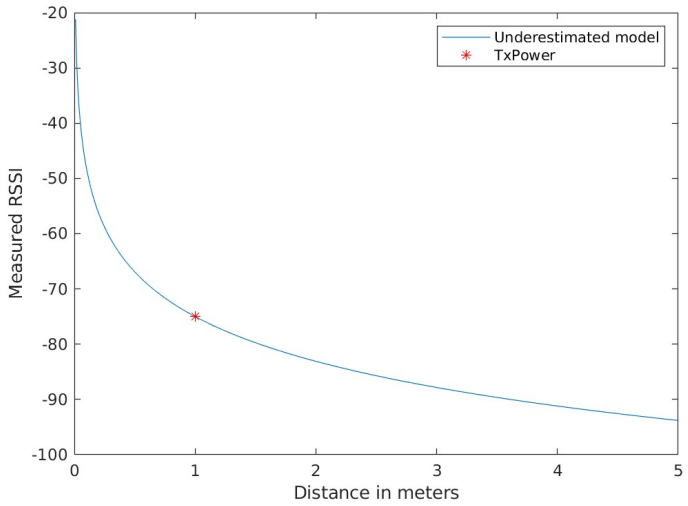

Figure 3.16: Underestimated model for wheelchair receiver.

While this underestimation hinders the potential of such a method, it also limits the potential for errors, as RSSI below -83 dB are all estimated to be below 2 meters. The two obtained measures, even if incorrect, can be compared: due to the size of the room the wheelchair will always be close to at least one of the beacons, which will give rise to a lower estimate for distance with respect to the other. If the wheelchair is instead at medium range between both of the beacons, results are similar. This can be exploited to estimate position of the wheelchair: knowing the position of the two beacons, two circles are calculated using as radius the underestimated distance from the wheelchair. If these circles intersect, one of the two intersection points is selected as an estimate of the wheelchair position, making use of the third distance, which is the one between the robot distance and the wheelchair, as it can be seen in figure 3.17. If not, the radius are scaled up until they intersect in a point. Four different positions were tested, as it can be seen in figure 3.18. Clearly this method is flawed, as there is a bias in the center of the two beacons. But, by making use of this underestimated model, the estimated

positions are more predictable.

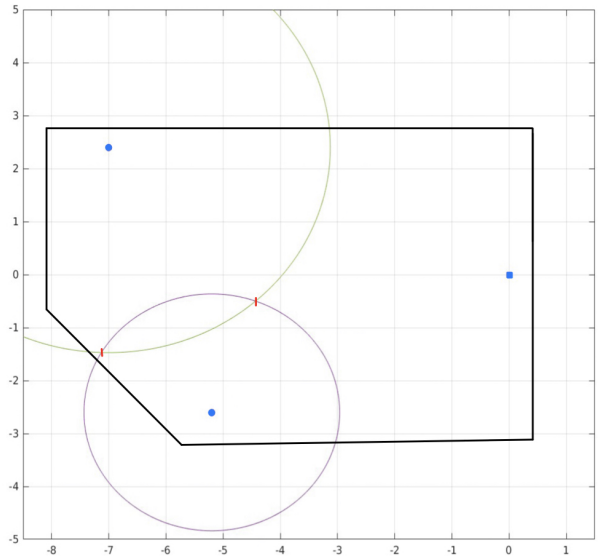

Figure 3.17: In red, the intersection between the two circles, which have radius equal to the estimated distance from the two beacons. The position closer to the value of distance estimated from the robot (blue square) will picked.

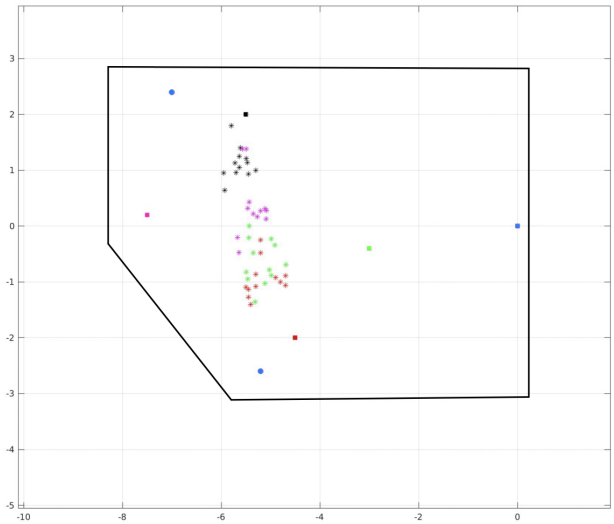

Figure 3.18: Triangulation estimation results. The blue square indicates the robot, while the blue circles are the two beacons. The other squares correspond to the real wheelchair position in which the triangulation was tested, while asterisks of the same color are the corresponding estimated positions.

Due to the way beacons are positioned, the maximum average error on y position was less than a 1 meter, while the total maximum average error is 2.19 meters. This error will surely increase if the wheelchair is positioned further away, on the very right side of the room, or in an other room entirely. It is clear that this method can be trusted only if the wheelchair is already somewhere in the middle of the two beacons, having a limited number of positions that can be estimated correctly. But, increasing the number of beacons, it can be scaled up, without worrying about the unreliable nature of RSSI estimation, as there will be always be beacons close enough to have a sound wheelchair position evaluation.
From these experiments it was clear that achieving a good estimation using RSSI trilateration at an apartment scale was too unreasonably expensive and complex.

3.6 Activating detection through beacon

Finally, in contrast to the way most beacons function, the Blue charm beacon present on the wheelchair was programmed to be inactive until the beacon's button is pressed twice. One reason is that, while BLE technologies and the bought beacon can accomplish battery life of a couple of years, when sending data every 100 ms battery life quickly lowers to a couple of months, and there is no reason to have the beacon active if the robot is inactive.
But more importantly, by making use of the feedback sent to the robot when the beacon is active, it is possible to use this as a start and stop method for our application. In fact, while the algorithm is always running, the robot will be mostly idle at the dock. When the first estimate of distance is received the charging mission is activated, the robot undocks and the coarse detection algorithm is started. If, during the coarse detection stage, a *No beacon* message is received, the robot stops, as it has failed in reaching the wheelchair in what was deemed acceptable time and goes back to the dock. As it was set up during testing, once activated through the double click, the beacon sent data packets for 5 minutes.

Chapter 4

Developed algorithm

4.1 Developed nodes and QoS discussion

For our application just three nodes were developed: **detection** and **beacon scanner**, running on the robot, and **distances**, which runs on the laptop positioned on top of the wheelchair. The first 2 can be launched together, making use of a ROS 2 launch file. The *beacon scanner* node, as the name suggests, is used for scanning for BLE data sent by the beacon present on the wheelchair, whereas *detection* takes care of everything else. *distances* is used just to pass the estimated distances of the wheelchair to the other beacons present in the environment. It is usually suggested to avoid promoting clarity and modularity by separating code, due to latency related problems. Still, the reasons why the algorithm was decided to be split into these parts was that the *beacon scanner* seldom publish data, and when it does it is at low frequency too, making lag non problematic. Also, when separated, it is possible to monitor the data published from *beacon scanner*, which was very useful during testing. The message interface sent by *beacon scanner* is made of two components: name, which is just a string containing the UUID, and distance, a float64.

To make communication between these three nodes effective, ad hoc QoS policies had to be set up. The main policies that can be customized are:

- **Reliability**: changes to this option influence the strength of communication. If set to *best effort*, the sender/receiver will attempt to exchange all the data; it may lose some instances, especially if communicating at high speeds, if the network is not stable. Instead, if *reliable* is picked, it is guaranteed that all sent samples are not lost, as it can try multiple times until received. By default FastDDS has reliability=best_effort for receivers and reliability=reliable for data writers.

- **History**: it defines the queue type. Can be either set on *keep all*, which tries to store all the samples until the middleware limits are reached, or *keep last*, which is the default option for FastDDS, which saves up to N instances.

- **Depth**: queue size can be defined by the depth policy only if history is set to *keep last*. By default it is set to 1 in FastDDS.

- **Durability**: data writers can publish information even if no data reader is present yet. If durability is set to *transient local* the information is saved and, once a subscriber joins the topic, it stores previous samples in its history; if durability is set to *volatile*, it ignores them and only new samples will be considered. On FastDDS publishers are set to *transient local* and receivers to *volatile* by default.

- **Deadline**: in deadline the maximum expected duration between samples is defined. This is useful to detect possible malfunctioning in the communication. On FastDDS it is set to infinite by default.

- **Lifespan**: again useful for debugging purposes, lifespan defines after how long a messaged should expire. By default on FastDDS it is set to infinite.

- **Liveliness**: this defines how a publishing node can be considered active after sending a message. If set on *automatic*, which is the default option, it will be alive for another lease duration; instead, if set on *manual by topic*, the publisher has to send a specific message to the topic to be considered alive for another lease duration.

- **Lease duration**: defined to be infinite in FastDDS by default, it specifies for how long a publisher should be considered alive.

To make sure publisher and subscriber are able to communicate trough a topic, it is important to avoid compatibility errors. On table 4.1, 4.2 and 4.3 the most common sources of incompatibility are reported.

Publisher	**Subscription**	**Compatible**
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

Table 4.1: Compatibility for Reliability policies

Publisher	Subscription	Compatible
Automatic	Automatic	Yes
Automatic	Manual by topic	No
Manual by topic	Automatic	Yes
Manual by topic	Manual by topic	Yes

Table 4.2: Compatibility for Liveliness policies

Publisher	Subscription	Compatible
Volatile	Volatile	Yes
Volatile	Transient local	No
Transient local	Volatile	Yes
Transient local	Transient local	Yes

Table 4.3: Compatibility for Durability policies

During testing, it was clear that using the default options for FastDDS to publish information on the robot had the best results, as all the actuators on the robot also receive data from these topics by a system default QoS profile.

When, instead, reading information from the robot's sensors, the default QoS showed its limits: as the robot publishes information at a relatively fast rate (by making use of the *ros2 topic hz*, data was estimated to be sent at 60 Hz for the faster sensors), a lot of information was lost due to the minimal history depth. To fix this, the QoS profile for sensors' data, imported from the rclpy.qos library, was used: while almost identical to the default FastDDS policies for subscribers, the change in depth from 1 to 5 made a significant improvement.

Finally, the volatile nature of information was not desirable when communicating from the *beacon scanner* or *distances* to the *detection* node. In fact, due to slight timing differences, this resulted in just about 50% of the sent message actually being read by the receiver. This was a big problem, because of the overall infrequency of the sent message (about every 10 seconds when the beacon is turned on). To fix this, a QoS profile was defined from scratch, for both the receiver and sender, having reliability=reliable, history=keep_all and durability=transient_local. Below is shown the subscription to the 'beacon_name' topic present on the *detection* node.

```
from rclpy.qos import QoSProfile, QoSReliabilityPolicy,
    QoSHistoryPolicy, QoSDurabilityPolicy
from beacon_interface.msg import Beacon

qos_profile = QoSProfile(
        reliability=QoSReliabilityPolicy.RELIABLE,
        history=QoSHistoryPolicy.KEEP_ALL,
        durability=QoSDurabilityPolicy.TRANSIENT_LOCAL
)

self.subscription7 = self.create_subscription(Beacon, 'beacon_name',
    self.listener_callback_switch, qos_profile)
self.subscription7
```

4.2 Beacon scanner

Writing the code for the scanner was not trivial, as it presented multiple snippets of the algorithm that needed to be running in an asynchronous way. To do so, the *asyncio* library had to be used, which is not supported by ROS. This meant that it was impossible to spin the node concurrently to the execution of code outside of the node. This problem was circumvented by running the node only at very specific instances.
The library used for Bluetooth detection is called Bleak. In the main, which is defined as an async function and continuously running, BleackScanner and its callback, *device_found*, are defined. Despite the name, this callback will be run every time the scanner is started. Finally, in an infinite loop, the scanner is started, run for a single second and then stopped.

```
async def main():
    scanner = BleakScanner()
    scanner.register_detection_callback(device_found)

    while True:
        await scanner.start()
        await asyncio.sleep(1.0)
        await scanner.stop()

asyncio.run(main())
```

In the callback, the calculation of distance is performed. Firstly, both global variables are set: UUID to 'No beacon found yet' and final_estimate to 0.0. If an iBeacon is found, defined by the code 0x004C communicated in the first 9 bits, it is possible to enter the try condition. The iBeacon parse is defined, to check the

correctness of the continuous information stream received, and the new UUID is stored. If it coincides with the one of our beacon, than the distance is estimated eight times and, after that, the median of these eight values is calculated, printed and stored. Once four of these median values are collected, an average of them is computed and **BeaconNode** is spinned, or runned, once.

```
@dataclass
class distance_class:
    measure=[0]*8
    counter1: int = 0
    counter2: int = 0
    counter3: int = 0
    final=[0]*4

def device_found(
    device: BLEDevice, advertisement_data: AdvertisementData
):
    distance=distance_class
    global final_estimate
    global uuid
    uuid='No beacon found yet'
    final_estimate=0.0
    try:
        apple_data = advertisement_data.manufacturer_data[0x004C]
        ibeacon = ibeacon_format.parse(apple_data)
        uuid= UUID(bytes=bytes(ibeacon.uuid))
        if str(uuid)=='426c7565-4368-6172-6d42-6561636f6e79':
            if device.rssi!=0:
                distance.counter3=0
                distance.measure[distance.counter1]=10**((-55 -
    device.rssi)/(10*2.67))
                distance.counter1+=1
                final_estimate=0.0
            if distance.counter1==8:
                distance.final[distance.counter2]=statistics.median(
    distance.measure)
                distance.measure=[0]*8
                distance.counter2+=1
                print(f"Measure {distance.counter2} = {distance.final
    [distance.counter2-1]} meters")
                if distance.counter2==4:
                    final_estimate=sum(distance.final)/4
                    print(f"Final estimate = {final_estimate} meters"
    )
                    rclpy.init(args=None)
                    beacon = BeaconNode()
                    rclpy.spin_once(beacon)
```

```
38                          rclpy.shutdown()
39                          distance.counter2=0
40                      distance.counter1=0
```

If it is not possible to run the try statement, we go into our except condition. KeyError just means that no Apple company ID (0x004C) could be found. Here a counter is updated and, once it reaches 100, all the previous collected values are reset and the node is spun once. Since the code did not enter in the try condition, actually for 100 times in a row, the values of UUID and final_estimate are still the same as defined just before it.

```
1     except KeyError:
2         distance.counter3+=1
3         if distance.counter3>100:
4             print('No beacon')
5             distance.measure=[0]*8
6             distance.counter1 = 0
7             distance.final=[0]*4
8             distance.counter2 = 0
9             distance.counter3 = 0
10            rclpy.init(args=None)
11            beacon = BeaconNode()
12            rclpy.spin_once(beacon, timeout_sec=10.0)
13            rclpy.shutdown()
14            final_estimate=0.0
```

Finally, the node itself is a simple publisher node, making use of the previously designed QoS profile and the custom beacon message. The message, which contains either information about distance or absence of a beacon, is published and then the node is shut down.

```
1  class BeaconNode(Node):
2
3      def __init__(self):
4          qos_profile = QoSProfile(
5              reliability=QoSReliabilityPolicy.RELIABLE,
6              history=QoSHistoryPolicy.KEEP_ALL,
7              durability=QoSDurabilityPolicy.TRANSIENT_LOCAL
8          )
9          super().__init__('beacon')
10         self.publisher = self.create_publisher(Beacon, 'beacon_name',
       qos_profile)
11         timer_period =0.0001 # seconds
12         self.timer = self.create_timer(timer_period, self.
     timer_callback)
```

```
13
14      def timer_callback(self):
15          msg = Beacon()
16          msg.name=str(uuid)
17          msg.distance=final_estimate
18          self.publisher.publish(msg)
```

4.2.1 Distances

This node will be running on the receiver placed on the wheelchair. Based on the same logic of the *beacon scanner*, the *distances* node presents just a few key differences. Firstly, the number of median estimations before averaging was increased to ten. The reason for this is that rapidness of this part of the algorithm is not required; as it will always be running, the robot will receive information even when idle at the dock. Thus, having a faster estimation does not imply any functional improvement.
Then, before sending information to the robot, all the distance estimations from all the different coded beacons have first to be computed. The speed of communication will correspond to the lowest beacon's speed. In our case, just two beacons are used. So, the custom message is composed of two floats: distances.beacon1 and distances.beacon2.
Finally, the TxPower was set to -75 dB. The reason for this choice is that, as explained previously, underestimating distances has a beneficial effect, if the wheelchair is located somewhere in the middle of the two beacons.

4.3 Detection

To program the *detection* node, which for obvious reasons is the core of our application, some rules had to be followed, to keep a clean, modular and readable code. Firstly, all the code written runs simultaneously inside the class *detection(Node)*. By doing so, it is possible to easily interact between different parts of the code, by making use of class attributes. Thus, all the information that is intended to be shared is defined as an attribute of self, a parameter used to refer to the current instance of the class, which is the detection node itself.

The node will publish into three topics, *cmd_vel*, *cmd_lightring* and *cmd_audio*, at a frequency of 20 Hz. Each callback will take care of different behaviors, namely path planning, light ring color and audio sequences played by the robot's speaker. It is subscribed to eight topics, and their callbacks are called as soon as new information is received. These callbacks can have various applications: they can either directly control the robot for safety movements, pass information on other

topics by storing received data into class attributes, or define the state of the robot. This last function is probably the most important one, as each part of this algorithm is coded to have completely different behaviors depending on the state of the robot. Six states were defined, **idle**, **stop**, **coarse**, **approaching**, **entering** and **returning**, each of them uniquely identified by a different light ring color.

4.3.1 Idle and stop states

Figure 4.1: Idle state.

Figure 4.2: Stop state.

As soon as the application is running, the robot will start in idle state. The light ring will be white, as shown in figure 4.1. While in this state the robot will have no safety protocols active, making it possible to stay docked and charge, as in this position the front bumper is pressed. While the robot seems to be completely inactive, it is still storing all the information fed by the other sensors and nodes. There are 2 ways the robot can become idle:

- If a 0.0 distance to the wheelchair is detected when in the coarse state. This means that the robot has failed to reach the wheelchair in time and the wheelchair's beacon deactivated.

- If the button 1, on the left of the power button, is pressed. This acts as soft reset of the algorithm, clearing all the previously detected data and being virtually identical to relaunching the application again.

The stop state is instead identified by the a red light ring color, as shown in figure 4.2. There is only one way to enter this state, which is by pressing button 2, on the right of the power button; it is possible to exit this state by pressing button 1. In the stop condition the robot will be unresponsive to any input from sensors, except button 1.

4.3.2 Coarse state

Figure 4.3: Coarse detection ring color.

Due to the unreliable nature of the chosen BLE technology, combined with the inability to perform SLAM, the coarse state is surely the most unpolished and incomplete part of the algorithm. It is the most complicated part of the project, due to the endless possible error inducing situations, and it will need to be developed further in the future.
Nevertheless, even if missing a robust method for path planning and localization of the wheelchair, it is important to acknowledge what was tried, and what are the most promising or limiting possibilities.
Before the coarse state is activated, some information must be first be gathered: distance of the wheelchair from the 2 beacons in the room, communicated from the *distances* node and stored in the self.distance_beacon1 and self.distance_beacon2 parameters, and distance from the robot to the wheelchair, which is estimated by the *beacon scanner* node and stored in the self.distance parameter.

```
if self.distance!=0.0 and self.distance_beacon1!=0.0 and self.distance_beacon2!=0.0 and self.idle==1:
    x1, x2= -5.2, -7.0
    y1, y2= -2.6, 2.4
    dx=x2-x1
    dy=y2-y1
    d=5.5
    if (self.distance_beacon1+self.distance_beacon2)<d:
        self.distance_beacon1=1.1*self.distance_beacon1
        self.distance_beacon2=1.1*self.distance_beacon2
    elif d < math.fabs(self.distance_beacon2 - self.distance_beacon1):
        print("ERROR: cirlce within circle")
    else:
        r1=self.distance_beacon1
        r2=self.distance_beacon2
        distance_center = (r1**2 - r2**2 + d**2)/(2*d)
        halfheight = math.sqrt(r1**2 - distance_center**2)
        center_x = x1 + (distance_center*dx)/d
        center_y = y1 + (distance_center*dy)/d
        intersec1_x=center_x+(halfheight*dy)/d
        intersec1_y=center_y-(halfheight*dx)/d
        intersec2_x=center_x-(halfheight*dy)/d
        intersec2_y=center_y+(halfheight*dx)/d
        diff_distance_origin1=math.fabs((intersec1_x**2+intersec1_y**2)**(0.5) - self.distance)
        diff_distance_origin2=math.fabs((intersec2_x**2+intersec2_y**2)**(0.5) - self.distance)
        if diff_distance_origin1<diff_distance_origin2:
            self.wheelchair_posx=intersec1_x
            self.wheelchair_posy=intersec1_y
        else:
            self.wheelchair_posx=intersec2_x
            self.wheelchair_posy=intersec2_y
        print(f"Position wheelchair: x= {self.wheelchair_posx}, y= {self.wheelchair_posy}")
        self.send_goal_undocking()
        self.coarse_movement=1
        self.idle=0
```

This piece of code is contained in the odometry topic callback. Beacon 1 and 2 position are known a priori, stored in respectively in the x1, y1 and x2, y2 components. At this point, a trilateration algorithm is executed, where, if the two circles build around the beacon position do no not intersect, estimated measurements are increased by 10%. Using some simple math, position of the two intersection points are calculated. Distances between the intersection points and origin, which corresponds to the dock's position, are calculated and compared to the one obtained

by the robot. The closest one is picked as wheelchair position. After this, the robot undocks and coarse movement starts.
From this point, some assumption had to develop the first draft of a path planning algorithm:

- **Wheelchair is in the same room as the robot**: while this requirement is definitely stringent, it was already proved that, with the developed algorithm for room localization, wheelchair room can be defined without error. If the robot has information about the map of the flat, which is planned to be developed in the future, it can easily move from its starting room to ensure this condition.

- **Wheelchair is somewhere between the 2 beacons**: the position of the wheelchair was defined using the underestimated model for trialateration, presented in chapter 3.5, with the 2 beacons in the same positions. It was shown that the maximum error was 2.1 meters, which is acceptable taking into consideration that the range of the IR LEDs present on the wheelchair is 5 meters. In particular, the y component of wheelchair position is estimated to be more accurate when in this configuration, as it presented a maximum error of just 1 meter. This condition can be ensured if a much higher number of beacons are implemented.

- **Orientation of the wheelchair is known**: knowing the orientation is extremely beneficial for this application, as the goal of the coarse detection algorithm is to enter in the cone zone of the wheelchair's IR LEDs. As of now, tests are been carried out to achieve this by mounting a gyroscope on the wheelchair.

The idea is that, once these 3 conditions are achieved, the robot can move towards the line passing through the wheelchair estimated position and having orientation equal to the one estimated by the gyroscope. A band is then defined by 2 other parallel lines, which are positioned at 0.5 meters from the original one. The robot can than move in a zig-zag pattern inside the band until the IR LEDs are sensed. This idea is shown in figure 4.4.

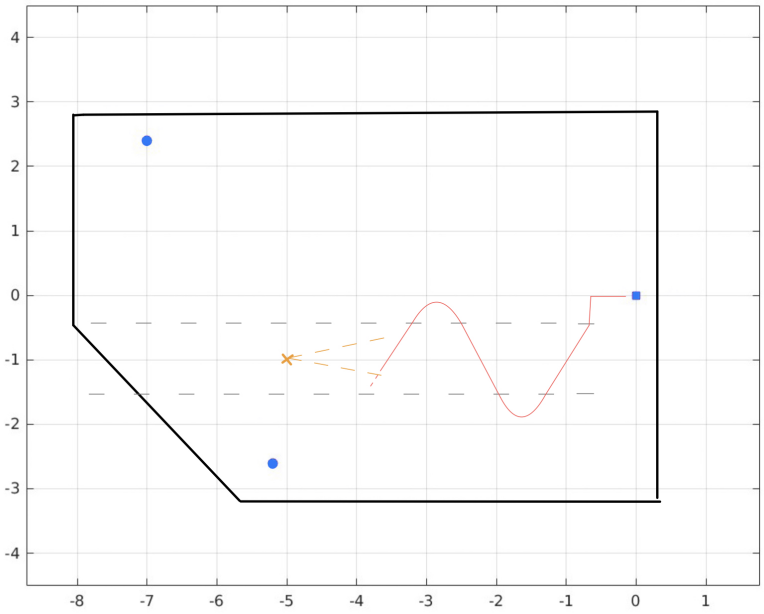

Figure 4.4: In blue, the robot original position and beacons are shown. The orange cross corresponds with the wheelchair estimated position and the orange dotted line indicates its orientation. The two black dotted lines define the band for path planning. Finally, in red movement of the robot is shown.

If the bumper is pressed, it means that during this procedure the robot hit an obstacle. A flag is then raised, depending on which side of the bumper was activated. Whenever that happens, a safety algorithm runs, stopping whatever the robot was doing.

```
if self.bumped_left==1:
    if self.bumped_counter<30:
        twist.angular.z=0.0
        twist.linear.x=-0.08
        self.bumped_counter+=1
    elif self.bumped_counter<50:
        twist.angular.z=-0.8
        twist.linear.x=0.0
        self.bumped_counter+=1
    elif self.bumped_counter==50:
        self.bumped_counter=0
        self.bumped_left=0
    self.publisher1.publish(twist)
elif self.bumped_right==1:
    if self.bumped_counter<30:
        twist.angular.z=0.0
```

```
17            twist.linear.x=-0.08
18            self.bumped_counter+=1
19        elif  self.bumped_counter<55:
20            twist.angular.z=0.8
21            twist.linear.x=0.0
22            self.bumped_counter+=1
23        elif  self.bumped_counter==55:
24            self.bumped_counter=0
25            self.bumped_right=0
26        self.publisher1.publish(twist)
```

The robot backs up, then turns to the opposite direction of where the bump was detected before resetting the flag. This behaviour is shared with other states of the robot and, depending on what it was doing before the bump, the main algorithm is started again in different ways.

While this algorithm has shown some mild success, it is clear that it is still in its primordial state. It is easy to imagine possible situation where this algorithm can fail. At this moment, no extra data can be used to improve this situation. Even the distance from the wheelchair, which theoretically should be an important factor in this stage, can hardly be used, as BLE RSSI method are too inaccurate.

4.3.3 Approaching state

Figure 4.5: Approaching detection ring color.

Approaching state corresponds to the first part of our fine detection algorithm and it is identified by the blue light ring color. To understand how this state works, it is important to first have a look at what happens in the *dock*, *ir_opcode* and *odom* callbacks, which play a crucial role as anticipated in subchapter 2.3.

```
def listener_callback_dock(self, dock):
    if (dock.dock_visible == 1 and (self.distance < 8.0 and self.
    distance!=0) and self.stop==0 and self.coarse_movement_started==1
    and (self.distance_from_dock > 2.5)):
        self.coarse_movement = 0
        self.lost_wheelchair = 0
        self.coarse_movement_started = 0

        if dock.dock_visible == 1 and self.approaching == 0 and self.
    getting_out1==0 and self.getting_out2==0:
            self.get_logger().info('Wheelchair is visible, starting
    to approach!')
            self.approaching=1

    elif self.approaching==1 and self.entering==0:
        self.get_logger().info('Wheres the wheelchair again?')
        self.lost_wheelchair=1
```

In the dock callback, if the dock is visible and all conditions are met, the approaching state is started and the coarse stopped. If for any reason, the dock is no longer visible while in this state, a flag, called *lost_wheelchair* is raised.

```
def listener_callback_iropcode (self, ir_opcode):
    if ir_opcode.opcode == 168:
        self.ir_opcode=168
        self.lost_wheelchair=0
    if ir_opcode.opcode == 172:
        self.ir_opcode=172
        self.lost_wheelchair=0
    if ir_opcode.opcode == 164:
        self.ir_opcode=164
        self.lost_wheelchair=0
    if ir_opcode.opcode == 160:
        self.ir_opcode=160
        self.lost_wheelchair=0
    if ir_opcode.opcode== 161:
        self.approaching=0
        self.coarse_movement=1
```

Whenever an opcode is detected, this flag is reset and the opcode is saved in a class attribute called *ir_opcode*. Even if there is a check on the distance from the real

dock, when, for any reason, an opcode of 161 is detected, the approaching state is stopped and the robot returns to the coarse stage. An opcode of 161, in fact, corresponds to the single detection of the field LED, which is not present on the dock replica, mounted on the wheelchair, and can only be detected if in the near proximity of the real one. While, in theory, this should never happen, it was still kept in the code to increase robustness.

```
def listener_callback_odom (self, odom):
    self.current_pose=odom.pose.pose.orientation.z
    if self.ir_opcode == 172:
        self.entering_pose = self.current_pose
    self.distance_from_dock = ((odom.pose.pose.position.x)^2+(odom.
   pose.pose.position.y)^2)^(0.5)
```

In the odom callback, three important values are stored in class attributes: the current pose, the pose where a 172 opcode was detected lastly and the distance from the dock. As our application is designed to start from the real dock, it is important to mention that position x=0.0, y=0.0 corresponds to the dock location. Having all these variables set up, it is possible to fully understand what happens in the *cmd_vel* callback when the approaching state is active, considering first what happens when the *lost_wheelchair*=0.

```
if self.bumped_left==1:
    if self.bumped_counter<30:
        twist.angular.z=0.0
        twist.linear.x=-0.08
        self.publisher1.publish(twist)
        self.bumped_counter+=1
    elif self.bumped_counter<70:
        twist.angular.z=-0.8
        twist.linear.x=0.0
        self.publisher1.publish(twist)
        self.bumped_counter+=1
    elif self.bumped_counter==70:
        self.bumped_counter=0
        self.bumped_left=0
        self.publisher1.publish(twist)
elif self.bumped_right==1:
    if self.bumped_counter<30:
        twist.angular.z=0.0
        twist.linear.x=-0.08
        self.bumped_counter+=1
        self.publisher1.publish(twist)
    elif self.bumped_counter<65:
        twist.angular.z=0.8
```

```
24            twist.linear.x=0.0
25            self.bumped_counter+=1
26            self.publisher1.publish(twist)
27        elif self.bumped_counter==65:
28            self.bumped_counter=0
29            self.bumped_right=0
30            self.publisher1.publish(twist)
31    elif self.ir_opcode == 172 and self.entering == 0 and self.
      getting_out1==0:
32            self.counter=0
33            twist.linear.x = 0.08
34            twist.angular.z = 0.0
35            self.publisher1.publish(twist)
36            self.ir_opcode=0
37    elif self.ir_opcode == 168 and self.entering == 0 and self.
      getting_out1==0:
38            self.counter=0
39            twist.linear.x = 0.08
40            twist.angular.z = 0.3
41            self.publisher1.publish(twist)
42            self.ir_opcode=0
43    elif self.ir_opcode == 164 and self.entering == 0 and self.
      getting_out1==0:
44            self.counter=0
45            twist.linear.x = 0.08
46            twist.angular.z = -0.3
47            self.publisher1.publish(twist)
48            self.ir_opcode=0
49    elif self.ir_opcode == 160 and self.entering == 0 and self.
      getting_out1==0:
50            twist.linear.x = 0.08
51            twist.angular.z = 0.0
52            self.publisher1.publish(twist)
53            self.ir_opcode=0
```

Firstly, safety operations are introduced when a bump is detected, by using information from the *hazard_detection* callback in exactly the same way as in the coarse state. Then, depending on the detected opcode, linear and angular speeds are defined. It is important to note that, once velocities are published, ir_opcode information is reset to zero. By making this adjustment, we avoid being stuck in a situation in which the dock is no longer visible, but the robot behaves as it still was. This fix does not compromise the efficiency of the algorithm in any way, as the sensor data from the opcode are sensed at a rate faster than 20 Hz.
Finally, another fix that was introduced during testing is the behavior of the robot when an opcode of 160 is detected. While theoretically meaning nothing in particular, when a 160 is sensed it means that the robot is able to understand the first four bits of the sent data, which correspond to 1010, but not to define clearly

the last four, which are set to 0000. This situation happens when the robot is in the limits of the LEDs range. By making the robot move forward, better data can be collected and the robustness of the algorithm was greatly increased.

The flag for lost wheelchair will be raised essentially for two main reasons: either the robot lost line of sight with the dock replica during its movement, or it became close enough to the wheelchair and it can no longer see them. Distinguishing between these two situations is extremely important.

```
if self.lost_wheelchair==1:
    twist.linear.x = 0.0
    twist.angular.z = 1.0
    self.counter+=1
    if self.counter < 20:
        self.publisher1.publish(twist)
    elif self.counter < 60:
        twist.linear.x = 0.0
        twist.angular.z = -1.0
        self.publisher1.publish(twist)
    elif (self.current_pose > (self.entering_pose-0.03) and self.
    current_pose < (self.entering_pose+0.03) and (self.entering == 0
    and self.distance < 3.0)):
        self.entering=1
        twist.linear.x = 0.0
        twist.angular.z = 0.0
        self.publisher1.publish(twist)

    #Here other elif condition are present, which correspond to the
    entering stage

    else:
        twist.linear.x = 0.0
        twist.angular.z=(self.entering_pose-self.current_pose)*2
        self.publisher1.publish(twist)
```

Once in the lost wheelchair condition, the robot will turn first 45° to the left and then 90° degrees to the right. If, during this movement, an opcode is detected, the *ir_opcode* callback will reset the flag and the approaching stage is continued.
If, instead, no opcodes are detected, it means that the robot reached the correct position under the wheelchair. It then rotates to match its current pose to the entering orientation, which was defined to be equal to the pose the last 172 opcode was detected. Once this position is reached with a maximum error of 3%, the entering state is set and will overwrite the approaching one.

4.3.4 Entering state

Figure 4.6: Entering ring color

The entering stage corresponds to the second and final part of the fine detection algorithm, and when active the light ring shines a purple color. Once in this stage, the robot should be positioned in the middle of the backside of the wheelchair and face straight into it. The callback for the *ir_intensity* topic is then used to fix slight miss positioning.

```
def listener_callback_ir_intensity(self, ir_intensity):
    self.side_left=ir_intensity.readings[0].value
    self.left=ir_intensity.readings[1].value
    self.front_left=ir_intensity.readings[2].value
    self.front_center_left=ir_intensity.readings[3].value
    self.front_center_right=ir_intensity.readings[4].value
    self.front_right=ir_intensity.readings[5].value
    self.right=ir_intensity.readings[6].value
    twist=Twist()
    if (self.entering==1 or self.getting_out2==1) and self.idle==0:
        if self.side_left>1200:
            twist.linear.x=0.04
            twist.angular.z = -(float(self.side_left)*0.0004)
            self.publisher1.publish(twist)
        if self.left>1200:
            twist.linear.x=0.04
            twist.angular.z = -(float(self.left)*0.0004)
```

```
18                self.publisher1.publish(twist)
19            if self.front_left>600:
20                twist.linear.x=0.04
21                twist.angular.z = -(float(self.front_left)*0.0004)
22                self.publisher1.publish(twist)
23            if self.front_center_left>600:
24                twist.linear.x=0.04
25                twist.angular.z = -(float(self.front_center_left)*0.0004)
26                self.publisher1.publish(twist)
27            if self.front_center_right>600:
28                twist.linear.x=0.04
29                twist.angular.z = float(self.front_center_right)*0.0004
30                self.publisher1.publish(twist)
31            if self.front_right>600:
32                twist.linear.x=0.04
33                twist.angular.z = float(self.front_right)*0.0004
34                self.publisher1.publish(twist)
35            if self.right>1200:
36                twist.linear.x=0.04
37                twist.angular.z = float(self.right)*0.0004
38                self.publisher1.publish(twist)
```

The values of the sensors are first stored in class variables to improve readability of the code. From all the given names, which were picked from the header of the sensor received data, it is already possible to note a slight asymmetry, as there is extra IR sensor on the left side.
If the value of each particular sensor surpasses a certain threshold, the robot stirs away from the obstacle proportionally to the strength of the signal. All the thresholds and proportional constants values were picked experimentally. As it can be seen from the code, side sensors need to have way higher readings compared to the front ones to change direction. The reason for this is that, since the robot is entering into a not so wide environment, side readings will always be high and, most of the time, they will not correspond to an imminent bump or a wrong misalignment.

On top of this safety method, again a bump safety protocol was introduced but with some small tweaks.

```
1 elif self.entering == 1:
2     if self.bumped_left==1:
3         if self.bumped_counter<30:
4             twist.angular.z=0.0
5             twist.linear.x=-0.04
6             self.publisher1.publish(twist)
7             self.bumped_counter+=1
8         elif self.bumped_counter<45:
9             twist.angular.z=-0.2
```

```
            twist.linear.x=0.0
            self.publisher1.publish(twist)
            self.bumped_counter+=1
        elif self.bumped_counter==45:
            self.going_under-=20
            self.bumped_counter=0
            self.bumped_left=0
            self.publisher1.publish(twist)
    elif self.bumped_right==1:
        if self.bumped_counter<30:
            twist.angular.z=0.0
            twist.linear.x=-0.04
            self.bumped_counter+=1
            self.publisher1.publish(twist)
        elif self.bumped_counter<45:
            twist.angular.z=0.2
            twist.linear.x=0.0
            self.bumped_counter+=1
            self.publisher1.publish(twist)
        elif self.bumped_counter==45:
            self.going_under-=20
            self.bumped_counter=0
            self.bumped_right=0
            self.publisher1.publish(twist)
    else:
        twist.linear.x = 0.04
        twist.angular.z=0.0
        self.publisher1.publish(twist)
    self.going_under+=1
    if self.going_under==200:
        self.get_logger().info('Im under the wheelchair!')
        self.coarse_movement=0
        self.ir_opcode=0
        self.counter=0
        self.lost_wheelchair=0
        self.entering=0
        self.going_under=0
        self.approaching=0
        self.entering=0
        self.under_wheelchair=1
```

In fact, both the backward and turning movement were sensibly reduced. Another difference is that now the left and right turning motion, when bumps happen, is symmetric, as there is no longer the risk of getting stuck in a corner in this stage. Finally, when a bump happens, the self_going under counter is reduced by 20. In fact, while in a later stage of the project magnetic field presence will be used as a stop condition for the robot, right now a simple counter is implemented. If the robot bumps slightly more time than normal is necessary to reach the exact

intended position.
Unless the robot bumps or detects obstacles in close proximity, it will just move slowly forward. Once the counter reaches a value of 200, the robot stops. At this point the under_wheelchair flag is raised, the robot plays a short tune, and WPT would begin. Working videos of the approaching, entering and first part of the returning state can be seen in the following links:
https://www.youtube.com/watch?v=1wctWv9RnpI
https://www.youtube.com/watch?v=2dxQak075mU
https://www.youtube.com/watch?v=WkQtsRsomug

4.3.5 Returning state

Figure 4.7: Returning state ring color

The goal of the returning state is to go back to the dock after the WPT operation is done. As for the coarse state, this stage is suffering from the absence of a map or more advanced sensors. Still, compared to what the coarse state must achieve, here the situation is less complex, as the dock will always be at position x=0, y=0. Making use of the dead reckoning estimation of the position of the robot, it was possible to reach the dock.

```
1 if self.under_wheelchair == 1:
2     self.counter_for_now+=1
```

```
3       if self.counter_for_now > 250:
4           self.getting_out1=1
5           twist.linear.x = 0.0
6           if self.counter_for_now==252:
7               self.get_logger().info('Getting out now, back to the dock
    ')
8           if self.getting_out2==0:
9               self.angle=3.1415
10              self.send_goal_rotation()
11              self.getting_out2=1
12          if self.getting_out2==1:
13              if self.bumped_left==1:
14                  if self.bumped_counter<30:
15                      twist.angular.z=0.0
16                      twist.linear.x=-0.08
17                      self.bumped_counter+=1
18                  elif self.bumped_counter<45:
19                      twist.angular.z=-0.8
20                      twist.linear.x=0.0
21                      self.bumped_counter+=1
22                  elif self.bumped_counter==45:
23                      self.bumped_counter=0
24                      self.bumped_left=0
25                  self.publisher1.publish(twist)
26              elif self.bumped_right==1:
27                  if self.bumped_counter<30:
28                      twist.angular.z=0.0
29                      twist.linear.x=-0.08
30                      self.bumped_counter+=1
31                  elif self.bumped_counter<45:
32                      twist.angular.z=0.8
33                      twist.linear.x=0.0
34                      self.bumped_counter+=1
35                  elif self.bumped_counter==45:
36                      self.bumped_counter=0
37                      self.bumped_right=0
38                  self.publisher1.publish(twist)
39              else:
40                  if self.dockvisible==1 and self.distance_from_dock
    <2.5:
41                      self.send_goal_docking()
42                  elif self.posx<-1.5 and self.posx>-0.5:
43                      twist.linear.x=0.08
44                      twist.angular.z=0.0
45                      self.publisher1.publish(twist)
46                  else:
47                      if (self.current_pose >(-math.copysign(0.7,self.
    posy)-0.03)) and (self.current_pose <(-math.copysign(0.7,self.posy
    )+0.03)):
```

```
48                          twist.linear.x=0.08
49                          twist.angular.z=0.0
50                      else:
51                          twist.angular.z= -math.copysign(0.8, self.posy
    )
52                          twist.linear.x=0.0
53                      self.publisher1.publish(twist)
```

At this stage, as the WPT mechanism is still in development, the robot just waits for about 10 seconds after getting under the wheelchair before entering in this stage. Once the counter reaches 252, the robot spins 180° and the returning stage is initiated, which is identified by the flag self.getting_out2=1.
To get out of the wheelchair, the robot uses the same type of obstacle proximity algorithm which was shown in the previous stage. After that the robot moves forward, until the x variable of position of the robot enters in the range -0.5, -1.5. This solution is not a general one, as it is surely just an ad hoc good approximation of the behaviour due to the particular room configuration it was tested in. After that the robot turns to face the dock: positive Y direction corresponds with a self.current_pose=-0.7, while the opposite direction implies self.current_pose=0.7. After that, the robot continues moving forward.
Finally, whenever the dock is visible and the distance to x=0, y=0 is less than 2.5 meters, the robot will dock.

Chapter 5

Conclusion

5.1 Accomplished objectives

In this thesis the first steps to develop a UGV able to wirelessly charge PWs were taken.
In the beginning, the constraints of the problem were defined clearly, and a robot model was picked for the first prototype. Using this model, software was developed using ROS 2 and Python.
Communication with the robot, through cabled USB or Wi-Fi, was established, making use of the QoS options in ROS 2. Both a Rasperry Pi and a laptop were used to simultaneously exchange data with the robot.
The fine detection algorithm works reliably, even if the range sometimes varies due to some parasitic effect in the board on which the IR LEDs are mounted on. The algorithm idea is effective and, with some minor polishing, optimal. The robot can reach autonomously under the wheelchair once the IR LEDs of the dock's replica are detected.
The WPT primary coil, which will be mounted on top of the robot, is still in development, so no integration of charging procedures in the behavior of the robot could be developed.
Coarse detection and navigation are still an open problem: using BLE beacons the robot can start and stop, and provide accurate estimation of which room the wheelchair is in. Other than that, using RSSI methods to estimate location yielded unsatisfactory results.
However, considering that this is the starting point of the development and its ambitious scope, it is not a surprise that complete fulfilment of this project's goals was not achieved in five months. Still, the work discussed on this thesis shows clearly the techniques that have worked and gives very specific indication on what should be developed next.

5.2 Possible future improvements

The list of possible improvements is vast, as while for fine detection the use of IR LEDs proved to be a good approach, BLE technologies were not as useful as expected. In fact, while room localization and communication can be easily achieved with BLE beacons, RSSI distance estimation is too unreliable and the underestimated trilateration technique is too limited.
The most important objectives that the new prototype of the robot must achieve are:

- **SLAM**: without a doubt, the lack of mapping capabilities was the most limiting factor. When a new robot is built from scratch, space must be found to add a LIDAR or a camera as it will make the use of SLAM possible, as well as significantly increasing the path planning capabilities.
- **More precise localization**: at room level, some way a localizing the wheelchair more precisely when not in line of sight with the IR LEDs should be found. This could be achieved by using UWB or this problem can be circumvented entirely by the new robot capabilities.
- **App implementation**: an App can be used to control the robot. It could start the operation, stop it as well as showing useful information about the state of charge of the robot and wheelchair. It could be used for localization as well: the user could indicate the position of the wheelchair in the map of the flat or data directly from the phone could be used to augment the precision.
- **New data received from the wheelchair**: the wheelchair should be able to communicate more information with the robot. The implementation of gyroscopes to transmit data about wheelchair's pose to the robot is currently being tested. A new way of estimating distance between robot and wheelchair would also be incredibly useful, as the BLE beacon was not precise or fast enough.

Bibliography

[1] Elena Garcia, Maria Antonia Jimenez, Pablo Gonzalez De Santos, and Manuel Armada. «The evolution of robotics research». In: *IEEE Robotics Automation Magazine* 14.1 (2007), pp. 90–103. DOI: 10.1109/MRA.2007.339608 (cit. on p. 1).

[2] Guang-Zhong Yang et al. «Combating COVID-19—The role of robotics in managing public health and infectious diseases». In: *Science Robotics* 5.40 (2020), eabb5589. DOI: 10.1126/scirobotics.abb5589. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abb5589. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abb5589 (cit. on p. 1).

[3] Ajmal Zemmar, Andres M Lozano, and Bradley J Nelson. «The rise of robots in surgical environments during COVID-19». In: *Nature Machine Intelligence* 2.10 (2020), pp. 566–572 (cit. on p. 1).

[4] Rie Fujisawa and Francesca Colombo. «The Long-Term Care Workforce: Overview and Strategies to Adapt Supply to a Growing Demand». In: 44 (2009). DOI: https://doi.org/https://doi.org/10.1787/225350638472. URL: https://www.oecd-ilibrary.org/content/paper/225350638472 (cit. on p. 1).

[5] Paolo Bevilacqua, Marco Frego, Daniele Fontanelli, and Luigi Palopoli. «Reactive Planning for Assistive Robots». In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1276–1283. DOI: 10.1109/LRA.2018.2795642 (cit. on p. 1).

[6] Kazuyoshi Wada, Takanori Shibata, Toshimitsu Musha, and Shin Kimura. «Robot therapy for elders affected by dementia». In: *IEEE Engineering in Medicine and Biology Magazine* 27.4 (2008), pp. 53–60. DOI: 10.1109/MEMB.2008.919496 (cit. on p. 1).

[7] Matthew W Brault et al. *Americans with disabilities: 2010*. US Department of Commerce, Economics and Statistics Administration, US ..., 2012 (cit. on p. 1).

[8] Mary Ellen Buning, Jennifer A Angelo, and Mark R Schmeler. «Occupational performance and the transition to powered mobility: A pilot study». In: *The American journal of occupational therapy* 55.3 (2001), pp. 339–344 (cit. on p. 2).

[9] Ahmed Aldousari, Abdulaziz Alghamdi, and Hassan Alwadei. «The 1991 Americans with Disabilities Act (ADA) standards for accessible design». In: *Am. Res. J. Humanit. Soc. Sci* 4 (2021), pp. 59–62 (cit. on p. 2).

[10] Ahmed Azad, Reza Tavakoli, Ujjwal Pratik, Benny Varghese, Calvin Coopmans, and Zeljko Pantic. «A Smart Autonomous WPT System for Electric Wheelchair Applications With Free-Positioning Charging Feature». In: *IEEE Journal of Emerging and Selected Topics in Power Electronics* 8.4 (2020), pp. 3516–3532. DOI: 10.1109/JESTPE.2018.2884887 (cit. on pp. 2, 5).

[11] Chakridhar R. Teeneti, Ujjwal Pratik, Gavin R. Philips, Ahmed Azad, Mark Greig, Regan Zane, Cathy Bodine, Calvin Coopmans, and Zeljko Pantic. «System-Level Approach to Designing a Smart Wireless Charging System for Power Wheelchairs». In: *IEEE Transactions on Industry Applications* 57.5 (2021), pp. 5128–5144. DOI: 10.1109/TIA.2021.3093843 (cit. on p. 2).

[12] Kornelia Lazanyi and Greta Maraczi. «Dispositional trust—Do we trust autonomous cars?» In: *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE. 2017, pp. 000135–000140 (cit. on p. 3).

[13] Ivan Cortes and Won-jong Kim. «Autonomous positioning of a mobile robot for wireless charging using computer vision and misalignment-sensing coils». In: *2018 Annual American Control Conference (ACC)*. IEEE. 2018, pp. 4324–4329 (cit. on p. 3).

[14] Andrew J Rentschler. «Analysis of the ANSI/RESNA Wheelchair Standards: A Comparison Study of Five Different Types of Electric Powered Wheelchairs». Sept. 2002. URL: http://d-scholarship.pitt.edu/8897/ (cit. on p. 5).

[15] Panus Nattharith and Mehmet Serdar Guzel. «An indoor mobile robot development: a low-cost platform for robotics research». In: *2014 International Electrical Engineering Congress (iEECON)*. IEEE. 2014, pp. 1–4 (cit. on p. 7).

[16] Ben Tribelhorn and Zachary Dodds. «Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education». In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 1393–1399 (cit. on p. 7).

[17] Elvis Ruiz, Raúl Acuña, Novel Certad, Angel Terrones, and Marıa Eugenia Cabrera. «Development of a control platform for the mobile robot Roomba using ROS and a Kinect sensor». In: *2013 Latin American Robotics Symposium and Competition*. IEEE. 2013, pp. 55–60 (cit. on p. 7).

[18] Lentin Joseph and Jonathan Cacace. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018 (cit. on p. 9).

[19] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. «Exploring the Performance of ROS2». In: *Proceedings of the 13th International Conference on Embedded Software*. EMSOFT '16. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2016. ISBN: 9781450344852. DOI: 10.1145/2968478.2968502. URL: https://doi.org/10.1145/2968478.2968502 (cit. on p. 9).

[20] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthé, Till Smejkal, and Gerhard Fettweis. «Latency Analysis of ROS2 Multi-Node Systems». In: *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. 2021, pp. 1–7. DOI: 10.1109/MFI52462.2021.9591166 (cit. on p. 13).

[21] Thai-Chien Bui and Suwit Kiravittaya. «Demonstration of using camera communication based infrared LED for uplink in indoor visible light communication». In: *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. 2016, pp. 71–76. DOI: 10.1109/CCE.2016.7562615 (cit. on p. 16).

[22] Wenguo Liu and Alan Winfield. «Implementation of an IR approach for autonomous docking in a self-configurable robotics system». In: *Proceedings of Towards Autonomous Robotic Systems* 251 (2009), p. 258 (cit. on p. 16).

[23] Paolo Barsocchi and Francesco Potortì. «Chapter 6.4 - Wireless Body Area Networks». In: *Wearable Sensors*. Ed. by Edward Sazonov and Michael R. Neuman. Oxford: Academic Press, 2014, pp. 493–516. ISBN: 978-0-12-418662-0. DOI: https://doi.org/10.1016/B978-0-12-418662-0.00012-X. URL: https://www.sciencedirect.com/science/article/pii/B978012418662000012X (cit. on p. 26).

[24] Huthaifa Obeidat, Wafa Shuaieb, Omar Obeidat, and Raed Abd-Alhameed. «A Review of Indoor Localization Techniques and Wireless Technologies». In: *Wireless Personal Communications* 119.1 (July 2021), pp. 289–327. ISSN: 1572-834X. DOI: 10.1007/s11277-021-08209-5. URL: https://doi.org/10.1007/s11277-021-08209-5 (cit. on p. 27).

[25] Zhang Xuexi, Lu Guokun, Fu Genping, Xu Dongliang, and Liang Shiliu. «SLAM algorithm analysis of mobile robot based on lidar». In: *2019 Chinese Control Conference (CCC)*. IEEE. 2019, pp. 4739–4745 (cit. on p. 27).

[26] Ilmir Z. Ibragimov and Ilya M. Afanasyev. «Comparison of ROS-based visual SLAM methods in homogeneous indoor environment». In: *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*. 2017, pp. 1–6. DOI: 10.1109/WPNC.2017.8250081 (cit. on p. 27).

[27] Umair Mujtaba Qureshi, Zuneera Umair, and Gerhard Petrus Hancke. «Evaluating the Implications of Varying Bluetooth Low Energy (BLE) Transmission Power Levels on Wireless Indoor Localization Accuracy and Precision». In: *Sensors* 19.15 (2019). ISSN: 1424-8220. DOI: 10.3390/s19153282. URL: https://www.mdpi.com/1424-8220/19/15/3282 (cit. on p. 28).

[28] David Schwarz, Max Schwarz, Jörg Stückler, and Sven Behnke. «Cosero, find my keys! Object localization and retrieval using Bluetooth Low Energy tags». In: *Robot Soccer World Cup*. Springer. 2014, pp. 195–206 (cit. on p. 31).

[29] Keiko Yamashita et al. «Smart hospital infrastructure: geomagnetic in-hospital medical worker tracking». In: *Journal of the American Medical Informatics Association* 28.3 (Dec. 2020), pp. 477–486. ISSN: 1527-974X. DOI: 10.1093/jamia/ocaa204. eprint: https://academic.oup.com/jamia/article-pdf/28/3/477/36428688/ocaa204.pdf. URL: https://doi.org/10.1093/jamia/ocaa204 (cit. on p. 31).

[30] Shinsuke Kajioka, Tomoya Mori, Takahiro Uchiya, Ichi Takumi, and Hiroshi Matsuo. «Experiment of indoor position presumption based on RSSI of Bluetooth LE beacon». In: *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*. 2014, pp. 337–339. DOI: 10.1109/GCCE.2014.7031308 (cit. on p. 32).

[31] Peter Membrey and David Hows. *Learn Raspberry Pi with Linux*. Springer, 2013 (cit. on p. 33).