POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Thesis

Autonomous Docking System with Obstacle Avoidance for a Differential Drive Robot

Thesis Supervisors
Prof. Marcello Chiaberge
Prof. Marina Mondin

Candidate
Simone Cavallera

Academic Year 2020-2021

Abstract

Nowadays, autonomous mobile robots are widely used in many different fields, from industries to home environments. To ensure autonomy from human intervention, a navigation system that includes obstacle avoidance, as well as an autonomous recharging system capable of guaranteeing long term activity, is essential. This work is part of a project carried out by the InnoTech System company in collaboration with the California State University of Los Angeles. Their goal is to develop a fully autonomous differential drive robot capable of executing various tasks such as helping people at the San Diego airport terminal or delivering food to students in the University Citadel. The aim of this thesis is the development and the implementation of an autonomous docking system that allows the mobile robot to reach the charging station avoiding potential obstacles. The introductory chapters present a review of the research solutions adopted to solve the autonomous charging issue and a summary of some of the actual state of the art obstacle avoidance algorithms. Subsequently, the project is divided into four main parts: the detection of the station identified by an augmented-reality tag, the docking process, the obstacle detection exploiting point cloud and the obstacle avoidance function. Lastly, after a brief description of the mobile robot that has been built in order to evaluate the developed system, the work concludes by analyzing the results obtained during the test phases.

Acknowledgements

I would like to thank my family for their support during my university experience at Politecnico di Torino. They have always encouraged me to do my best and I know that I can always count on them. A heartfelt thanks to my girlfriend Anna, for always being at my side despite the distance due to our university carriers.

I would also like to thank my thesis supervisor Marina Mondin, Dr. Fred Daneshgaran and the Innotech System CTO Kash Olia for the great opportunity to work with them at the California State University of Los Angeles.

Thinking of the last five years I would like to express my gratitude to all my friends and colleagues. Especially to Franco, Giro, Cesi, Giorgio and Rubi who helped me during the exam sessions and the long hours of lesson. A special thanks to all my housemates: Jack, Ambro, Nico, Yaya, Meme and Gianlu. I shared with them unforgettable experiences.

Then, a greeting to my long friends Mitch and Fill and Forno for their constant presence even during the pandemic period and for their important advises. I consider myself lucky to have met honest and sincere people like you. I also want to thank my friends with whom I spent my best weekend in Cuneo.

In the end, I want to thank Alessandro, my roommate and friend who shared with me this Erasmus project in California. I could not ask for a better colleague.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The spread of Autonomous mobile robots (AMR) has exponentially increased in the last decade thanks to their relevance and applications to the actual world. They can be distinguished from the other robots because they are capable of making decisions, moving autonomously and reacting based on the information they can obtain from the surrounding environments. A robot is defined as autonomous when it has the ability to determine the actions to be taken to perform a task, using a perception system and without human control or intervention [1]. It is a system that operates in an unpredictable and partially unknown environment having the capability to avoid obstacles while carrying out its tasks. Nowadays, mobile robots are employed in different fields such as medicine, sports, industry, distribution of goods and service robotics. Moreover, given their high level of reliability and safety they have begun to be used in a household environment. An example of this is the Roomba robot vacuum, an advanced cleaning system that takes action based on what it perceives. It can be placed in a room, left alone, and it will do its job without any human supervision. At the end of September 2021 Amazon introduced Astro, a household robot (Fig. 1.1). It is a three wheel robot with a camera that rises up on a 42-inch arm that is able to follow you around,

play music or display TV shows on its touchscreen display. It can create a map of your house and complete some simple tasks such as bringing two sodas to someone in another room without any supervision from a person.

Figure 1.1. Amazon's Astro robot.

In general the basics of mobile robotics consist of four main fields: locomotion, perception, cognition and navigation [2].

Locomotion

The locomotion topic is an important aspect of mobile robot design based on kinematics, dynamics and control theory. It does not rely only on the medium in which the robot is supposed to work but also on other technical criteria such as maneuverability, controllability, terrain condition, efficiency, stability and so on. According to the locomotion system there exist different mobile robot categories: stationary (arm/manipulator), land-based (wheeled mobile robot, walking mobile robot, tracked slip locomotion and hybrid), air-based, water-base and other (nanorobots,

snake-like robots, ecc. . .). Wheeled mobile robots can be further classified depending on the drive system: differential drive, car-type, omnidirectional and synchro drive. There does not exist an optimal drive configuration but depending on the requirements it is possible to choose the most convenient one. The advantages of wheeled mobile robots are efficiency and simplicity with respect to the other robot categories. Usually, they are easier and cheaper to build, design and program and they have a great stability having the wheels always in contact with the ground [3].

Perception

It is the main process that allows robotics systems to interact with the world. If a mobile robot is unable to observe the surrounding environment correctly or to perform tasks with high accuracy such as the obstacle detection and position estimation, it cannot operate in an autonomous way. Perception is achieved through sensor measurements and a subsequent analysis of the extracted information from the collected data. With sensors it is possible to perform localization, mapping and obstacle identification tasks which are the basis for the autonomous navigation and path planning topics.

Cognition

The cognitive level of a mobile robot is the decision-making and execution part that allows it to achieve high-level objectives. According to the information received by the sensors and the robot's targets the cognition and control system take the optimal actions in order to achieve its goals. The purpose of the cognition models is to represent the robot, the environment and the manner in which they interact. For instance, a motion planner, given the position of the robot and position of its target destination, can compute the optimal path to attain the task without

colliding with obstacles detected using sensor data.

Navigation

The goal of navigation is to drive the robot from its location to the destination in an known or unknown environment and to avoid possible obstacles taking into account the sensors measurements. It means that the robot has to rely on other aspects, such as perception, localization, cognition and motion control. The navigation field is divided in three main tasks:

- generating a map to model the real world.
- computing a free trajectory from starting point to a target point based on static information read in the map.
- according to the sensor data, modify the trajectory in order to avoid collision with obstacles and move the robot along the recalculated trajectory.

Mobile robots are widely used because they can be employed in a huge amount of applications such as transportation, surveillance, research, education and customer support. Moreover, the future innovations in the artificial intelligence and neural networks fields will increase their abilities allowing them to revolutionize the world of the automotive sector, logistics and distribution, oceanic exploration and household.

1.1 Autonomous Recharging Issue

As previously stated, an autonomous system is a certain type of system that is able to deal with the real environment in autonomy. In recent years, the great strides that have been made regarding autonomous navigation of mobile robots have highlighted the importance of the robot's long term autonomy. Robotic

systems, which have to operate for long duration, need to be recharged often without human intervention.

Successful ARP (Autonomous Recharging Problem) solutions have increased the efficiency of robotic systems reducing the human intervention and at the same time, increasing the operational capability. This issue can be divided in 4 sub-problems [4]:

- Energy Awareness: it has to choose when the robot should recharge itself in order to reach a recharging station before its batteries are drained.

- Static Recharging Stations: Fixed locations for mobile robots to replenish their energy. It requires a docking phase and a certain time to complete the recharging of batteries.

- Mobile Recharging Agents: they offer more flexibility for recharging a deployed team of robots, however, they increase the complexity of the system. In fact, recharging with mobile agents involves the worker robot and a mobile recharging agent that have to negotiate a location for recharging.

- Coordinating Teams of Workers and Rechargers: in a teams of robot scenario the recharging mechanism has to coordinate properly to optimize the performance.

Autonomous recharging brings with it the promise of extended runtime, reduced need for human intervention, and enhanced system performance.

So far, autonomous robots have been defined and attention has been focused on the relevance of the charging process in order to guarantee complete independence from human intervention. Now, in this introductory chapter, it will be presented the purpose of the thesis work and the entire project of which it belongs. Later, it will be described the following chapters structure.

1.2 The Goal of the Thesis

The investments made in research on autonomous robots and in particular on autonomous guidance systems have seen huge growth over the recent years. These trends have led the Department of Electrical and Computer Engineering of California State University of Los Angeles and the startup *Innotech Systems* company to invest and gain competence in these fields. The mission of the *Innotech Systems* is to develop and deploy secure and collaborative autonomous systems for a wide range of applications in the service of the public. The company, in the last few years, joined the incubator of the San Diego Airport with the aim of building an autonomous robot. The latter should be able to interact with users, help passengers navigate the complex structure of the airport, provide delivery services (foods, beverages, packages, ...) inside terminals and contribute to the security services by monitoring the various areas. This project is divided into subsystems, some of them are under developing and others have to be start from scratch:

- Mechanical design
- User interface
- Depth-SLAM based
- Path Planning
- GPS Localization
- Autonomous charging system

At the time of writing the robot is able to localize itself in the indoor environment thanks to the Depth-SLAM technology and in the outdoor scenarios using the GPS Localization with sensor fusion. It can navigate autonomously using the

A* algorithm for path planning purposes and a mechanical design of a differential drive prototype is under planning. As previously described, the autonomous charging field is composed of different sub problems. The goal of the thesis is the development of a docking system for a differential drive robot based on a static recharging station. More details about this topic will be provided in the next chapters.

1.3 Organization of the Thesis

The thesis is divided into 5 chapters. The second chapter describes the most used and famous sensors employed for object detection and the autonomous charging projects. Then, after an overview of the literature on docking strategies, the topic of obstacle avoidance is defined by going into detail of some of the most famous algorithms. The third chapter is totally dedicated to the thesis project, where it is explained why certain types of sensors have been chosen and the docking system features. The latter can be divided into 4 major phases: station detection, docking process, obstacle detection and obstacle avoidance. The fourth chapter shows the work done in the Gazebo simulation environment and the performance obtained during the tests in realistic scenarios. At the end the last chapter summarizes the goals achieved by this thesis and the possible and future improvements that can be integrated in order to enhance the autonomous charging system.

Chapter 2

State of the Art

Mobile robots which can guarantee a long term activity need a technology that supports the autonomous recharging and the corresponding docking phase. The docking and charging strategy are strongly related to the hardware and sensors employed to build the robotic systems. For this reason, in this chapter, the principal sensors mounted on mobile robots for autonomous navigation and docking applications are first introduced, explaining their advantages and disadvantages. Secondly is presented an overview of docking and charging strategies starting from one of the first algorithms and ending to the actual approaches focusing the attention on the different types of sensors used. To conclude, it addresses the obstacle avoidance theme providing the definition of the topic and some of the most popular methods to solve this fundamental problem for the development of autonomous systems.

2.1 Sensors

"A sensor is a component that measures some aspects of the environment"[5]. Sensors are divided in two classes:

- proprioceptive: they measure something internal to the robot. An example could be the car speed measured with the speedometer which counts the rotation of the wheels.
- exteroceptive: they measure something external to the robot and give the information about the surroundings allowing the vehicle to interact with the world. They are divided in two categories: active sensors (sonar, radar, laser-scanner, IR sensors) and passive sensors (visible and infrared spectrum camera). The former emit energy, for instance, the ultrasonic sensors emit sound waves and then use the reflected sound to measure the obstacles distance. The others, on the other hand, do not affect the environment such as a camera that records the light reflected by objects.

In many applications, such as obstacle avoidance and path planning for autonomous navigation, robots have to measure the distance from the possible surrounding obstacles. Nowadays, the most widespread technology for distance measurement are infrared (IR) sensors, ultrasonic sensors, light detection and ranging (LiDAR) sensors and stereo cameras. The first two are cheap and are often used on educational robots or in combination with more complex sensors. The remaining ones are more expensive and guarantee better performance, in fact, they are used in challenging applications such as self-driving cars. IR receivers, stereo cameras and cameras are a possible solution also for the docking and autonomous charging problem because they are able to detect the station using landmarks such as IR beacons or QR codes.

2.1.1 Infrared Sensors

An infrared (IR) sensor is an electronic device that measures and detects infrared radiation in its surrounding environment. There are two types of infrared sensors: active and passive. Passive infrared sensors (PIR) only detect infrared radiation

and do not emit it from a light emitting diode (LED). They are commonly used in motion-based detection such as in-home security systems. Instead, the active IR sensors are made up of two components: transmitter and receiver. The transmitter is a LED with a beam projection area. The information that is read by the receiver could be of two types: analog and digital. The analog information detects the intensity of the beam coming to the transmitter, instead the digital information gives us only if we have a detection or not. Active IR sensors could be used for different scope:

- Object detection: Transmitter and receiver are facing in the same direction. When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver (Fig.2.1).
- Line Detection: Transmitter and receiver are facing in the same direction. The light is reflected by the bright colored surface and absorbed by the dark surface, so if we draw a black line on the ground we are able to follow it.
- Motor/Wheel encoders: Transmitter and receiver are placed in such a way that the detection is interrupted by the wheel rotation in order to derive distance traveled by the robot. The resolution of the encoder depends on the number of detection per rotation.

Infrared sensors have good reliability both in daytime and nighttime for obstacle or motion detection. Furthermore, it can detect soft and small objects which are not easily detected by ultrasonic sensors. The disadvantages are related to the fact that this type of sensor is affected by smoke, dust, fog and sunlight.

2.1.2 Ultrasonic Sensors

An ultrasonic sensor is an electronic device that generates ultrasonic sound waves (typically between 40-50 kHz). It is able to measure the distance to an

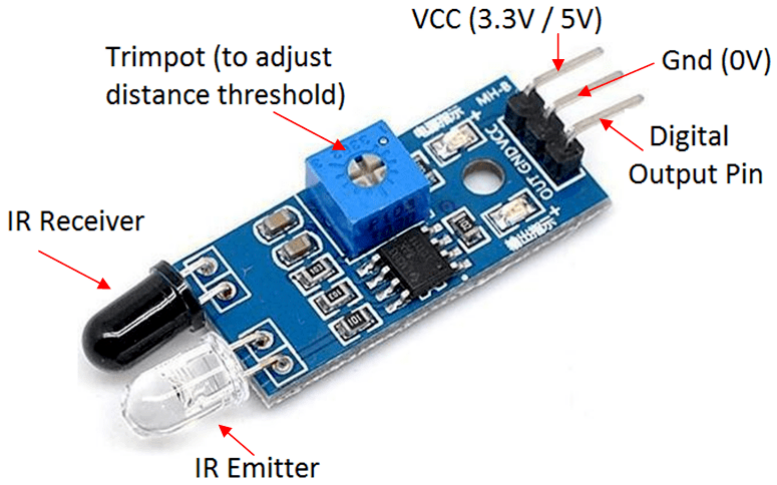

Figure 2.1. IR sensor for obstacle detection [6].

object converting the reflected sound waves into an electrical signal. Ultrasonic sensors have two principal components: the transmitter which is in charge to emit the sound using piezoelectric crystals and the receiver which receives the sound after it has traveled to and from the target (Fig. 2.2). The distance between the sensor and the object is calculated measuring the elapsed time between the sound emissions to its contact with the receiver.

$$D = \frac{1}{2} * T * C \tag{2.1}$$

Where D is the distance, T is the measured time and C is the speed of the sound (343 meters/second in the air at 20 °C). Ultrasonic sensors can work in any adverse conditions and they provide good readings when objects are large and with a hard surface. They are not affected by interference of smoke, gas, and other airborne particles or by the color or transparency of objects. However, they are strongly affected by the variation of temperature and they have some issues detecting soft, curved, thin and small obstacles.

Figure 2.2. Ultrasonic sensor.

2.1.3 LiDAR

Light Detection And Ranging (LiDAR) technology fires laser beams in all directions and then it catches the reflection measuring how long the beams take to return (Fig. 2.3). Its mode of operation is conceptually the same as radar and sonar. It is able to detect obstacles and figure out how far away they are exploiting the time taken for the laser pulse to return to the receiver. Usually it works using multiple lasers that rotate to scan the environment around in a 360-degree field. In fact, for each laser signal emitted the sensor receives a distance measurement, therefore the greater the number of signals the better the performances. LiDAR is capable of depth estimation and dense point cloud generation. It has a high precision in distance measurement (centimeters), it is stable, reliable and the detection is not influenced by temperature or light. However, it also has some weaknesses. Depending on weather performance it could be subject to false positives data created by the reflection caused by rain, fog and dust. Some dedicated algorithms have been developed to manage these issues but weather can still remain a problem for LiDAR-based systems. The laser strength has to be regulated in order

to avoid eyesight damages. This limitation introduces a trade-off between field of view (FOV), resolution and distance. To conclude, the efficacy of LiDAR measurements strongly depends on the reflectivity of the objects. If the laser signal encounters a reflective object, the majority of its energy rebounds back to the receiver and we have a successful detection. However, if the signal meets an obstacle with poor reflectivity the signal energy that returns could be only a fraction of the total energy and the reliability of detection can decrease [7].

Figure 2.3. LiDAR sensor.

2.1.4 Camera

Unlike other sensors, cameras can't measure distances. However they can provide much more detailed information. Digital cameras use an electronic component called a charge-coupled device which senses light waves and returns an array of picture elements (pixels) [5]. Their principal characteristics are the pixel number captured in each frame, the content of pixels and the lens field of view. For example the Raspberry Pi Camera Module v2 is capable of 3280 $\times$ 2464 pixels with static images (Fig. 2.4).

The images provided by the camera can be of three different types: black and

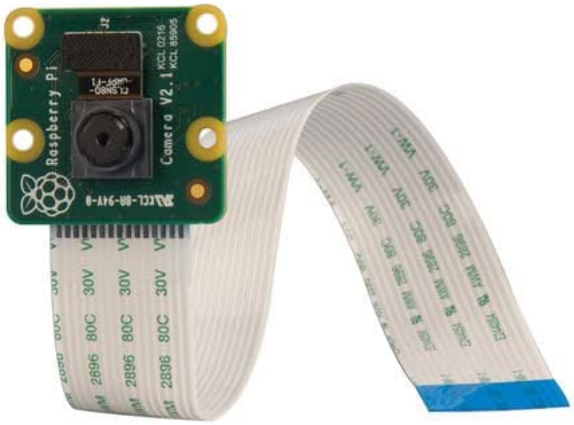

Figure 2.4. Raspberry Pi Camera Module v2.

white (1 bit per pixel), gray-scale (1 bytes per pixel) and full color RGB (3 bytes per pixel). In robotic applications cameras work at high frame rate (60-90fps), therefore they need a very large memory to store and analyze several images per second. Image analysis is fundamental in the autonomous navigation topics such as for example object detection in the environment and the interaction with people or other robots.

2.1.5 Stereo Camera

Stereo camera is a type of camera with two or more image sensors (Fig.2.5). This allows the camera to simulate human binocular vision and gives it the ability to capture three-dimensional images and perceive the depth. Unlike other sensors, it doesn't provide direct distance measurements but it provides distance estimations obtained by processing two brightness images of the same environment. Stereo camera is capable of depth estimation and dense point cloud generation. The effectiveness of measurements is tied to the resolution of the cameras and to the environment light. It requires high computational resources for computer vision

processing and it needs calibration if it becomes un-calibrated during driving. Nevertheless, significant developments have occurred that make stereovision a more attractive technology. For instance, the availability of low-cost high-resolution cameras, the production of embedded SoCs designed explicitly for real-time computer vision processing and the automatic on-the-fly camera calibration.

Figure 2.5. Stereo camera.

2.2 Autonomous Docking

The effectiveness of mobile robots is directly impacted by the amount of time they can spend executing tasks and the level of autonomy they can exert in remaining operational for long durations. One big issue of autonomous robots is not only the autonomy provided by their batteries, but also the docking process needed to reach the charging station. In this section it will be presented an overview of the most relevant researches about autonomous docking with charging stations. For example, one of the solutions proposed in 2013 [4], incorporates both static and mobile recharging stations and provides the necessary coordination for multiple worker robots to share a single mobile recharging station (Fig. 2.6).

However, in this thesis project, the aim is the development of an algorithm that can guarantee a correct docking of a single mobile robot with a static recharging station. For this reason, the next subsection will analyze some researches about

Figure 2.6. Mobile recharging agent [4].

docking strategies, but all the other subproblems of the autonomous charging issue such as energy awareness, mobile recharging and coordinating teams of workers and rechargers will not be furthered.

Docking, for our purposes, can be defined as moving the robot from the current position to a desired position and orientation following a safe trajectory [8]. In fact, autonomous recharging solutions imply that the mobile robot has to dock with the static or mobile station with a certain precision error. The autonomous docking issue has different solutions that depend on the task and the environment given to the robot, but also on the technologies and sensors employed.

2.2.1 Overview of Docking Strategies with Static Station

One of the first implementations was developed at the University of Tsukuba, in 1998, using a mobile robot named Yamabico-Liv (Fig. 2.7).

A laser beam projector was installed on the recharging station and a reflection detector was fixed on the robot body. Thanks to the optical reflection tapes present in the environment, which worked as landmarks, Yamabico-Liv was able to estimate its position in the previously installed map. Then, the robust navigation program guided it into its recharging station where its special hardware enabled the electric contact to the battery recharger [9].

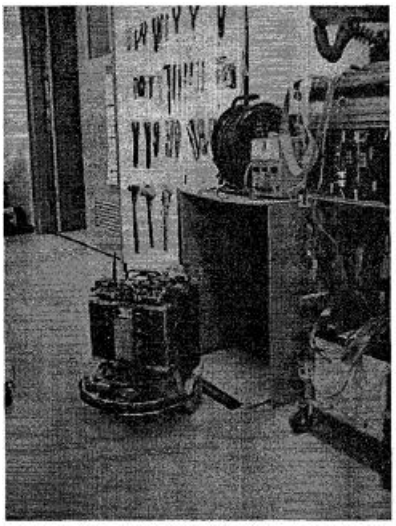

Figure 2.7. Yamabico Liv and recharging station [9].

Some years later at the Australian National University a more complex docking system was proposed. In this project was used the Nomadic Technologies™ Nomad XR4000 mobile robot equipped with 48 short-range infra-red sensors surrounding it, sonar sensors and a Sick LMS-200 laser range finder. The recharging station had an infra-red beacon and a target designed to be detected by the Sick Laser Range Finder and to be distinguished by the surrounding environment. The docking strategy was divided in two phases: in the first one the robot approached the recharging station thanks to the IR sensors that performed the long-range IR beacon detection. Unfortunately, the latter were not able to distinguish between the IR beacon and nearby objects. Therefore, sonar sensors and Sick laser were used in combination with them in order to be able to detect close obstacles (Fig. 2.8).

The second phase began when the laser target pattern was visible to the Sick laser. It provided guidance information in order to reach the perpendicular position of the plugging direction with an accuracy error of 1mm. [10].

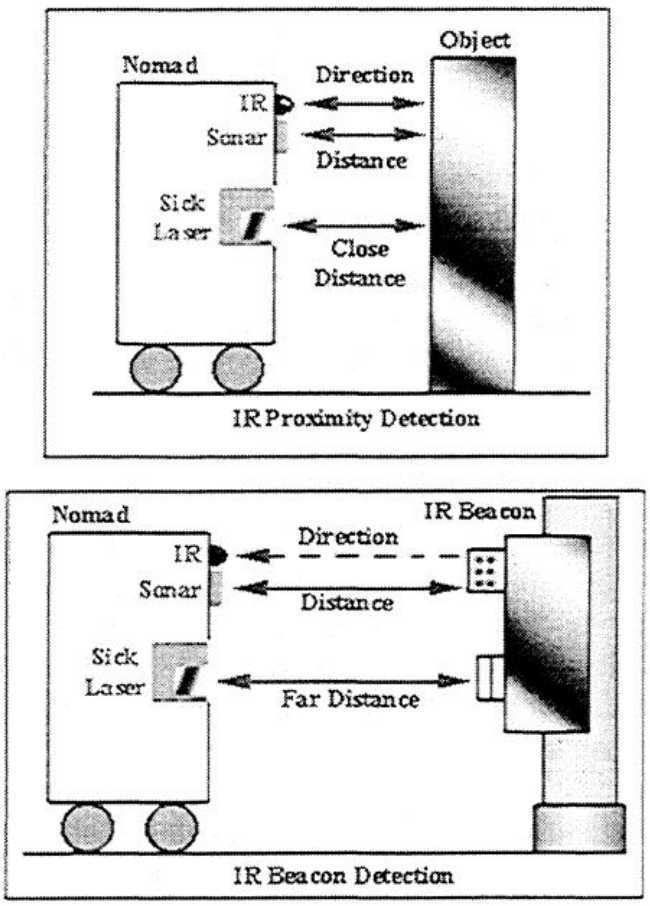

Figure 2.8. The distinction between the IR proximity detection and the IR beacon detection using the Sick laser range finder and Sonar sensors [10].

Another interesting project is shown in [11], where authors focus their efforts on developing a particular docking mechanism to allow a high angular and displacement error during the docking process. In Figure 2.9 we can see the specific hardware that was designed on the stationary docking station and on the Pioneer 2DX robot.

The battery voltage level was monitored by a software that decided when the recharging algorithm had to take control of the robot. In that case, firstly, the robot's pan-tilt-zoom (PTZ) camera guided it using a vision target that is an

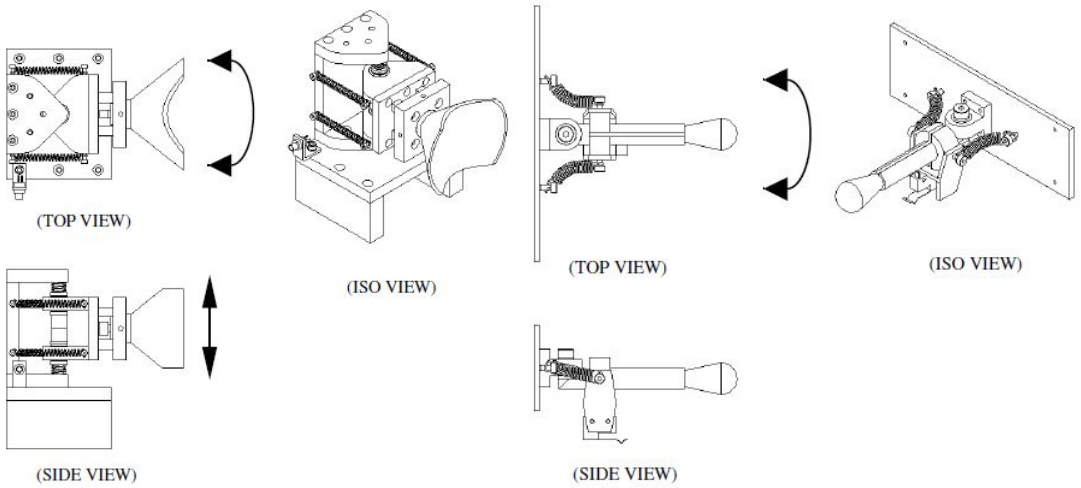

Figure 2.9. At left the docking station model and at right the robot docking mechanism model [11].

orange colored piece of paper mounted on the wall above the docking station. Secondly, the algorithm used a laser range-finder that provided the angle to the laser bacon mounted above the station. An interesting aspect of this work is the docking strategy, which was developed taking in consideration the fact that the docking mechanism was attached on the back of the robot where there were no sensors. The algorithm started with the robot headed towards its destination. When it reached a certain distance, it executed a turn until it faced away from the station. At this point, it began the blind mate with the docking station. It is important to point out that the vision target had to be in the robot's view, otherwise, it couldn't dock because it didn't have a map of the environment.

From these first articles, as mentioned at the beginning of the section, it is possible to understand that the autonomous docking strategies vary greatly according to the sensors and the charging hardware employed. Moreover, it would seem that the most used technology in the early 2000s was the laser range-finder which guarantees a high level of accuracy despite its high costs. In subsequent years, it

was gradually replaced with more accurate and powerful IR sensors. In fact, the docking strategies that are based on IR sensors combined with sonars, or in some case, exclusively on IR sensors are cheaper than the previous solutions.

For instance, this is one of the reasons that led to the birth of commercial cleaning robots equipped with cheap infrared sensors for the charging station docking. In 2005 an interesting research conducted by the Department of Mechanical Engineering of the Korea Advanced Institute of Science and Technology in South Korea showed a homing system that utilized cheap infrared sensors and a passive docking mechanism that could compensate for docking errors [12]. The robot was equipped with 6 receivers with a receipt angle of 45°. Two of them were located at the front of the robot at an interval of 10°, instead, the remaining ones were arranged at intervals of 40° on either side of it. The transmitter was built with 5 LEDs and was installed on the docking station. They were arranged at intervals of 30° in order to divide the area in front of the station of ± 75° in 9 regions (Fig. 2.10).

The robot was able to estimate its position thanks to the infrared signal that provided the information about which LED is emitting. Depending on where the robot was located among these nine regions, the docking algorithm executed the predetermined moving patterns in order to drive it in the central region.

The previous solution worked only with cheap infrared sensors but, however, a specific docking mechanism was necessary in order to compensate for the potential errors. In this article [13], the authors propose an algorithm based on the concept of triangulation to drive the robot toward the station that requires only infrared and ultrasonic sensors without a specific hardware. Three receivers were fixed on the robot respectively one at left, one at right and one in the center. The station position was provided with the aid of two beacons. The docking algorithm

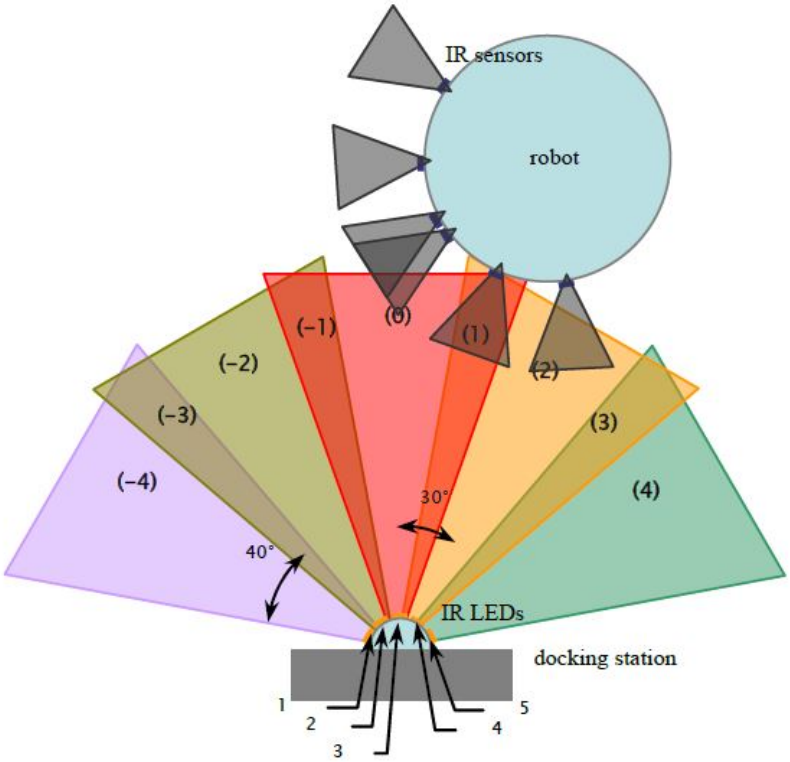

Figure 2.10. The homing system and the regions created by infrared beams [12].

consisted of two modes of operations: the seek source mode and the beacon recognition mode. During the first, the robot, based on the signal received by the IR receivers, aligns itself in the direction of any of the beacons and moves towards that beacon. The latter was detected when the robot was within 1m radius of it, the ultrasonic sensors detected an obstacle and the central IR receiver indicated a presence of source. In this situation, the robot shifted in the beacon recognition mode which was used to register the detection. In Figure 2.11 is illustrated a possible implementation of the algorithm.

When the robot was powered-on it shifted in seek source mode and it looked for beacon signals. If the IR receivers detected something the robot started moving towards the beacon (1) until the latter was detected. It entered in the beacon

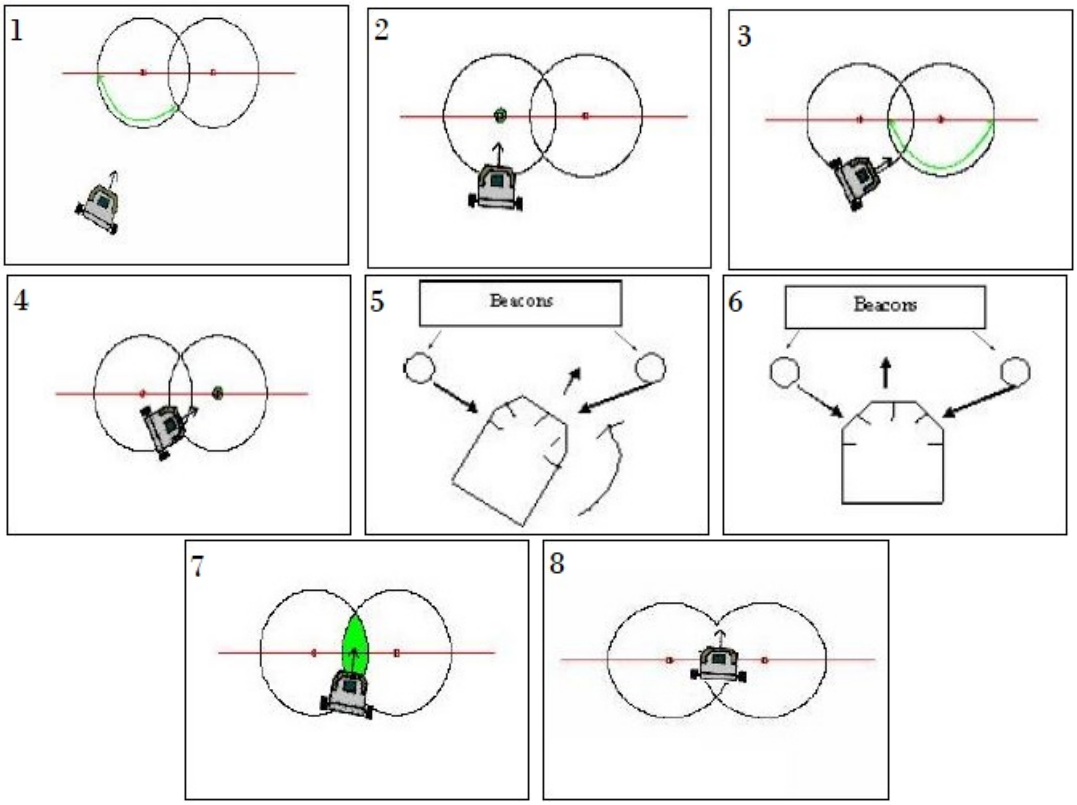

Figure 2.11. Example of the algorithm implementation divided in 8 steps [13].

recognition mode and registered the beacon (2). Then, it turned again in seek source mode and started moving towards the other beacon (3) till the recognition wasn't possible (4). At this point, it started to rotate about itself in order to be directed along the angular bisector of the triangle, formed by the beacons and the robot (5). Once the robot reached the correct orientation (6), it moved in a straight line (7) and concluded the docking (8). There exists another version of this implementation with the same modes of operation, but it was based on three beacons which were able to identify more precisely a unique point in the work environment.

In the last years the advances in camera technology that have dramatically reduced the cost of this type of hardware, combined with progresses in software

for image processing and miniaturization of computing power, have made them the sensors of choice for robotics and automation. Moreover, the innovation in the visual pattern recognition world led to the development of algorithms that can solve some fundamental problems in computer vision such as correspondence, pose estimation, and structure from motion.

In 2015, at the International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities in Vancouver was showed a docking projects which used QR (Quick Response) codes as recharging station landmarks and IR sensors to avoid possible obstacles [14]. QR code is a type of landmark increasingly used nowadays because it is easy to detect, to read and also to create. In addition, there are many libraries to decode it and it can store a suitable amount of information. The robot was called Wifibot and it was equipped with 2 front IR sensors that measured distances from obstacles and a web-cam which captured images for QR code detection. In Figure 2.12 is shown a possible docking path combined with the obstacle avoidance obtained thanks to the IR sensors.

Essentially, Wifibot turned around itself looking for a QR code. It took a picture, decoded it, and checked whether there was a QR code with the desired information encoded. If the QR code was not found, it turned, and tried again. Once the QR code was found, the robot determined its relative position and aligned itself to the QR code in such a way that the symbol is centered in the middle of its image.

Some years later, a research about autonomous charging based on computer vision was published by two engineers of the University Politehnica of Bucharest [15]. The approach is similar to the Wifibot solution, but in this case, they used a wireless recharger and the descriptor of the recharging base was a simple symbol printed on a post-it note (Fig. 2.13). The docking ended when the last snapshot

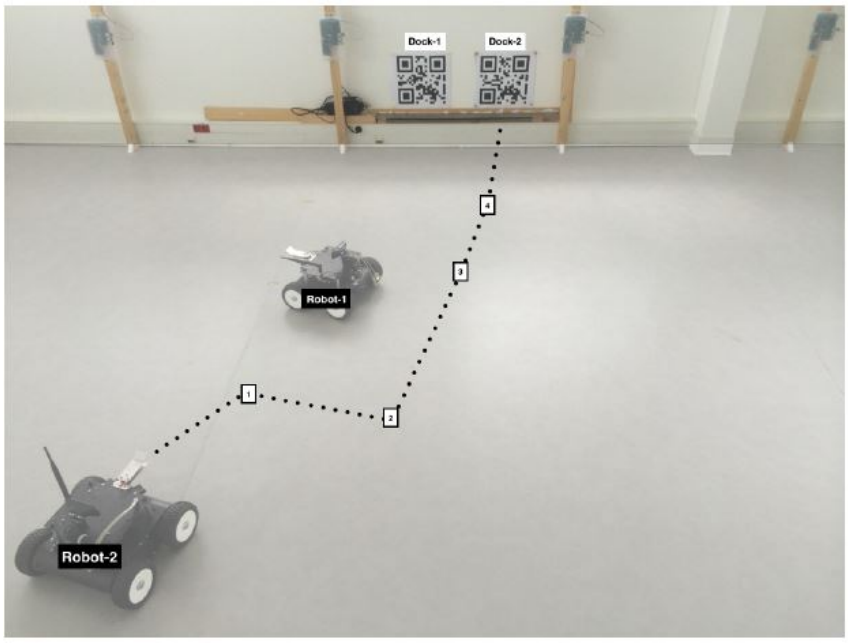

Figure 2.12. Sample docking path with obstacle [14].

had the descriptor of the base centered and the distance to it is smaller than 10 mm.

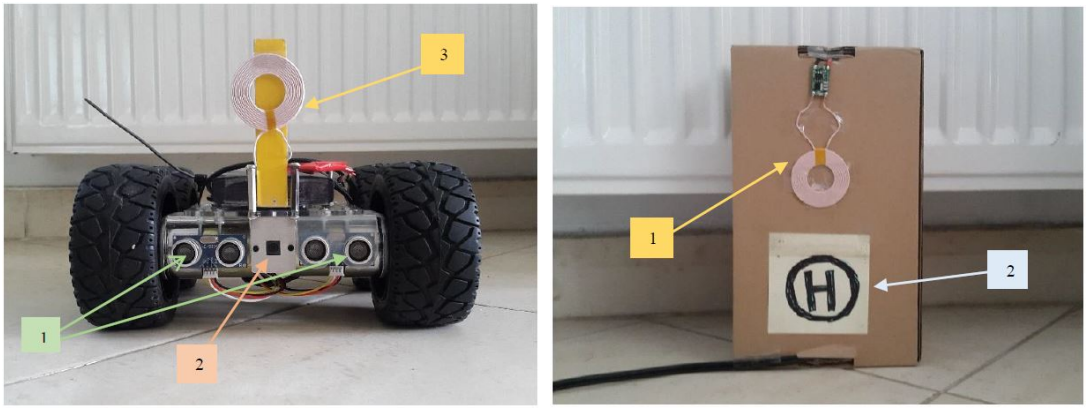

Figure 2.13. At right the frontal image of the robot: 1, ultrasonic sensors; 2, Raspberry Pi camera; 3, wireless power receiver. At left the charging base: 1, wireless power transmitter; 2, base descriptor [15].

Recent works about autonomous recharging topic would seem to suggest that the direction for the possible solutions of the autonomous docking issue is the employment of computer vision combined with proximity sensors.

2.3 Local motion planning

The concept at the base of the autonomous navigation in robotics is very simple: a mobile robot must navigate from one point to another, without human intervention, in a cluttered environment. The navigation of mobile robot topics can be divided into two tasks: the high-level task (global navigation) and the low level task (local navigation).

The goal of global navigation is to find the best path in order to reach the destination. Assuming a perfect scenario and a complete knowledge of the environment, this technique gives a complete solution for the problem (e.g. the A* Algorithm [16] [17]). Nevertheless, the real word is unknown and unpredictable, therefore, without the combination of a low level task the previous technique fails. Local navigation adapts the mobile robot behaviors to the changes of the surrounding environment. In fact, the high-level path finding can be done once (or every few minutes) but the low-level task of obstacle avoidance must be performed frequently exploiting the sensor's information. For example, in self-driving applications, the car never knows when a pedestrian will jump into the road or when the car it is following will suddenly brake but it can compute the best path to reach the destination using maps and traffic information without the aid of sensors [5].

2.3.1 Obstacle avoidance definition

The advantages of obstacle avoidance in the local navigation is to compute motion by introducing the sensor's information which allows it to take into account

the reality of the world. It is used to adapt the motion planning to any possible trap situation which is incompatible with initial global plans. An obstacle avoidance definition extracted from Motion Planning and Obstacle Avoidance book is set out below:

"Let A be the robot (a rigid object) moving in the workspace W, whose configuration space is CS. Let q be a configuration, q_t this configuration at time t, and $A(q_t) \in W$ the space occupied by the robot in this configuration. In the vehicle there is a sensor, that in q_t measures a portion of the space $S(q_t) \subset W$ identifying a set of obstacles $O(q_t) \subset W$. Let u be a constant control vector and $u(q_t)$ this control vector applied in q_t during time δt. Given $u(q_t)$, the vehicle describes a trajectory $q_t + \delta_t = f(u, q_t, \delta t)$, with $\delta_t \geq 0$. Let $Q_{t,T}$ be the set of configurations of the trajectory followed from q_t with $\delta_t \in [0, T]$, a given time interval. $T \geq 0$ is called the sampling period. Let $F : CS \times CS \rightarrow R^+$ be a function that evaluates the progress of one configuration to another.

Let q_{target} be a target configuration. Then, in time t_i the robot A is in q_{t_i} , where a sensor measurement is obtained $S(q_{t_i})$, and thus an obstacle description $O(q_{t_i})$. The objective is to compute a motion control u_i such that: (i) the trajectory generated is free of collisions with the obstacles $A(Q_{t_i}, T) \cap O(q_{t_i}) = \emptyset$; and (ii) it makes the vehicle progress to the target location $F(q_{t_i}, q_{target}) \leq F(qt_i + T, q_{target})$. The result of solving this problem at each sample time (Fig. 2.14a) is a sequence of motion controls $\{u1...u_n\}$ computed in execution time that avoids the obstacles gathered by the sensors, while making the vehicle progress towards the target location in each configuration $\{q_{t_1}...q_{target}\}$" (Fig. 2.14b) [18].

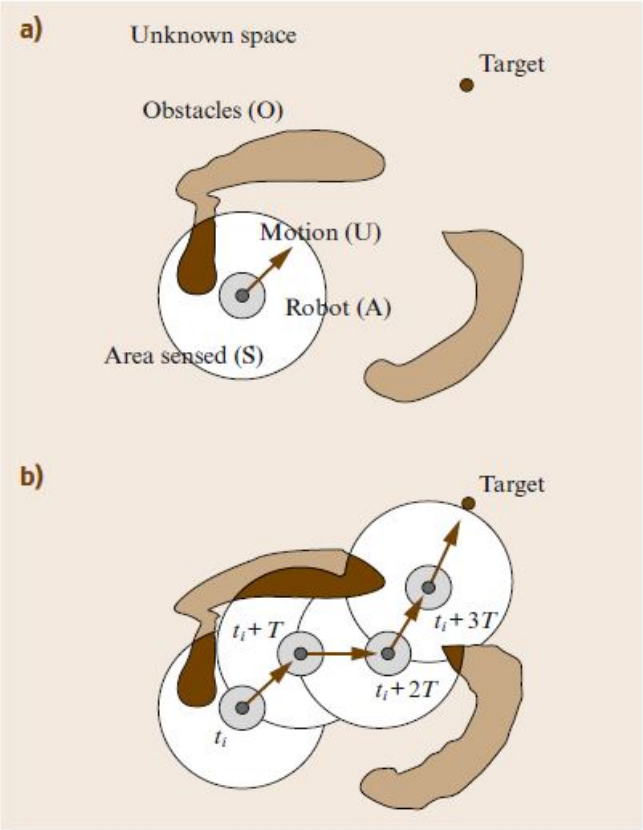

Figure 2.14. (a) The obstacle avoidance problem consists of computing a motion control that avoid collisions with the obstacles gathered by the sensors, whilst driving the robot towards the target location. (b) The result of applying this technique at each time is a sequence of motions that drive the vehicle free of collisions to the target. [18].

2.3.2 Obstacle avoidance techniques

In order to develop an obstacle avoidance method there are three main aspects to take in consideration: the obstacle avoidance technique, the type of robot sensors and the type of scenario (static or dynamic, unknown or known, structured or not). In this section are described the most common techniques used nowadays in the autonomous navigation field.

Potential Field Methods (PFM)

In the potential field method the robot is seen as a particle that moves in space under the influence of a force field. The destination exerts a force that attracts the robot [18]:

$$F_{att}(q_{t_i}) = K_{att} n_{q_{target}}, \tag{2.2}$$

where K_{att} is the constant of the force, q_{t_i} is the vehicle configuration and $n_{q_{target}}$ is the unitary vectors that point from q_{t_i} to the target. Instead, the obstacles exert a repulsive force in order to move away the particle.

$$F_{rep}(q_{t_i}) = \begin{cases} K_{rep} \sum_j \left(\dfrac{1}{d(q_{t_i}, p_j)} - \dfrac{1}{d_0} \right) n_{p_j}, & if \;\; d(q_{t_i}, p_j) < d_0 \\ 0, & otherwise \end{cases} \tag{2.3}$$

where K_{rep} is the constant of the force, d_0 is the influence distance of the obstacle p_j, q_{t_i} is the robot configuration and n_{p_j} is the unitary vectors that point from q_{t_i} to the obstacle p_j. The robot motion at each instant t_i is computed in order to follow the resulting force induce by the sum of $F_{att}(q_{t_i})$ and $F_{rep}(q_{t_i})$ [19] (Fig. 2.15).

$$F_{tot}(q_{t_i}) = F_{att}(q_{t_i}) + F_{rep}(q_{t_i}) \tag{2.4}$$

The Eq.2.3 is the classic formulation where the repulsive force depends only on the robot configuration. In the subsequent version, the potential depends also on the instantaneous velocity and acceleration of the vehicle.

$$F_{rep}(q_{t_i}) = \begin{cases} K_{rep} \sum_j \left(\dfrac{a\dot{q}_{t_i}}{(2ad(q_{t_i}, p_j) - \dot{q_{t_i}}^2)} \right) n_{p_j} \cdot n_{\dot{q}_{t_i}}, & if \;\; \dot{q}_{t_i} > 0 \\ 0, & otherwise \end{cases} \tag{2.5}$$

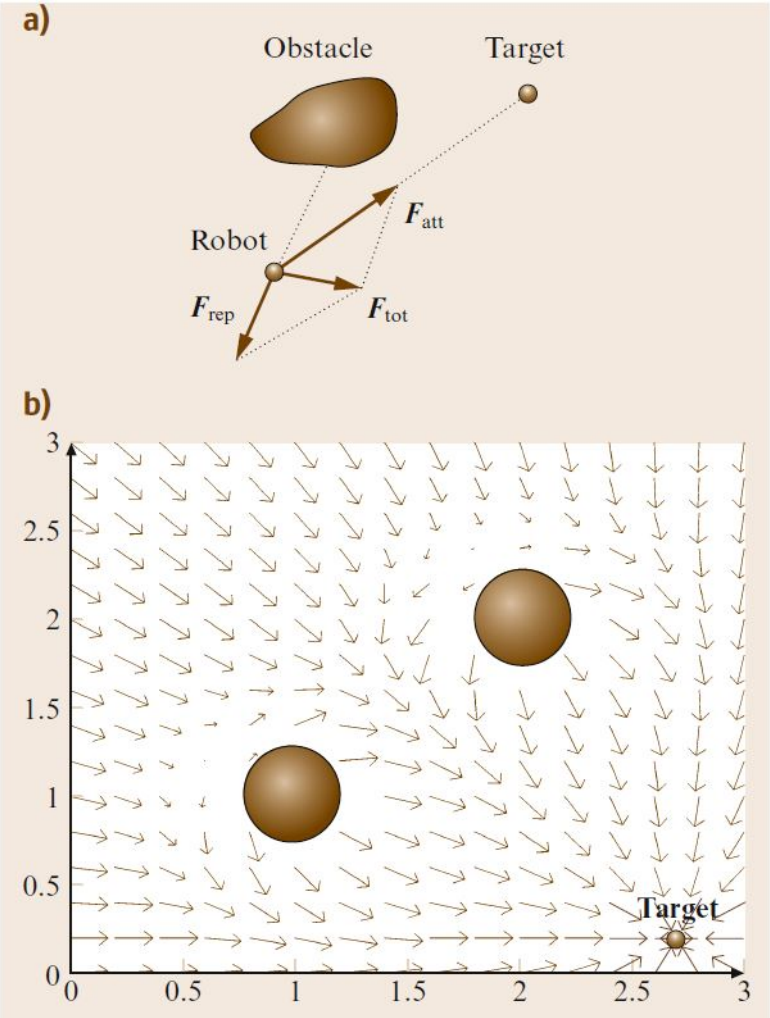

Figure 2.15. (a) Computation of the motion direction with a potential field method. The target attracts the particle F_{att} while the obstacle exerts a repulsive force F_{rep}. The resulting force F_{tot} is the most promising motion direction. (b) Motion directions computed in each point of the space with the classic method [18].

where $\dot{q}_{t_i}$ is the current robot velocity, $n_{\dot{q}_{t_i}}$ the unitary vector pointing in the direction of the robot velocity, and a is the maximum vehicle acceleration.

Vector Field Histogram (VFH)

The VFH computes the motion direction in two steps: firstly it computes the set of candidates, secondly it selects one of them [18] [20].

- Candidate Set of Directions

  The space around the robot location is divided into sectors. Then, the method builds a polar histogram where each component represents the obstacle polar density in the corresponding sector. The function

  $$h^k(q_{t_i}) = \int_{\Omega_k} P(p)^n \left(1 - \frac{d(q_{t_i}, p)}{d_{max}}\right)^r dp \tag{2.6}$$

  maps the obstacle distribution in sector k on the corresponding component of the histogram $h^k(q_{t_i})$. The density $h^k(q_{t_i})$ is proportional to the probability $P(r)$ that an obstacle occupies a point, and to a factor that increases as the distance to the point decreases (both functions powered by some integers n, r > 0). The domain of integration is $\Omega_k = \{p \in W \setminus p \in k \wedge d(q_{t_i}, p) < d_0\}$. The histogram is characterized by directions with high density (peak) and directions with low density (valley). The set of candidate directions is called selected valley. It is an histogram region where the directions are lower than a given density threshold and closest to the target direction (Fig.2.16).

- Motion Computation

  The second step consists in choosing the best direction from the selected valley using a strategy based on three rules:

  - Case 1: the goal sector (k_{target}) is in the selected valley. It means that $k_{sol} = k_{target}$.
  - Case 2: the goal sector is out of the select valley and the valley sectors are greater than m. The solution is $k_{sol} = k_i \pm \frac{m}{2}$, where m is a fixed number of sectors and k_i the sector of the valley closer to the k_{target}.

– Case 3: the goal sector is out of the select valley and the valley sectors are lower than m. The solution is $k_{sol} = \frac{k_i + k_j}{2}$, where m is a fixed number of sectors and k_i and k_j are the extreme sectors of the valley.

The direction solution θ_{sol} is the bisector of the k_{sol} sector and the velocity v_{sol} is inversely proportional to the closest obstacle distance.

Obstacle Restriction Method (ORM)

The ORM solves the obstacle avoidance problem in 3 steps. As in the previous method, it computes the motion selecting the best direction from a set of candidates extracted in the first two steps [18].

- Instantaneous Target Selection

  This step finds a set of possible sub-goals when it is not possible to directly reach the goal and it is better to direct the motion towards a given zone of the space (that ameliorates the situation to reach the goal later), rather than directly towards the goal itself. The sub-goals can be located at the edge of the obstacles or between obstacles. A local algorithm checks if the final destination is directly reachable. If not, the closest reachable sub-goal becomes the new destination (Fig.2.17).

  Let x_a and x_b be two locations of the space, R the robot radius, and L a list of obstacle points, where x_p^L is an obstacle of the list. Let be L' is the list of points of L that are in the rectangle with the height the segment $\overline{x_a x_b}$ and width $2R$. Let A and B be the two semi-planes divided by the line that joins x_a and x_b. If for all the points of L', $d(x_j^L,) > 2R$ (with $x_j^L \in A$ and $x_k^L \in B$), then the algorithm return positive if there is a collision-free path that joins both locations or negative if the final location cannot be reached [21].

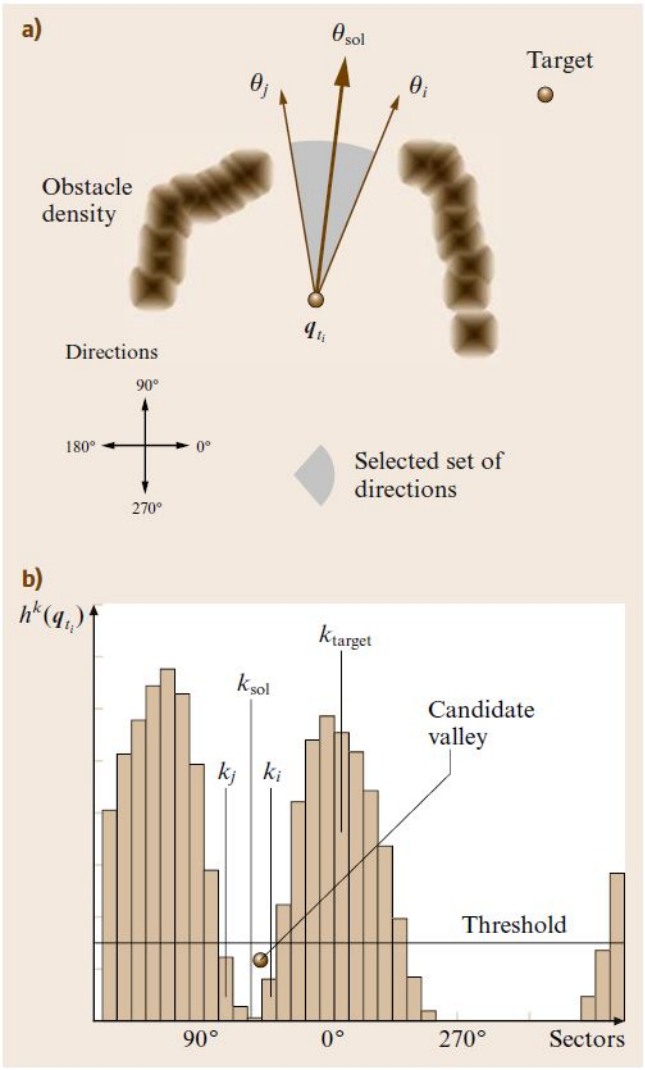

Figure 2.16. Computation of the motion direction θ_{sol} with the VFH. (a) Robot and obstacle occupancy distribution. (b) The candidate valley is the set of adjacent components with lower value than the threshold. The navigation case is case 3 since the sector of the target k_{target} is not in the valley and the number of sectors is lower than a fixed quantity m ($m = 8$, i. e., 45°). Thus the solution is $k_{sol} = \frac{k_i+k_j}{2}$, whose bisector is θ_{sol} in (a). The bisectors of k_i and k_j are θ_i and θ_j, respectively [18].

This process is standard and can be used also in the other methods as a prepossessing step to validate the final location.

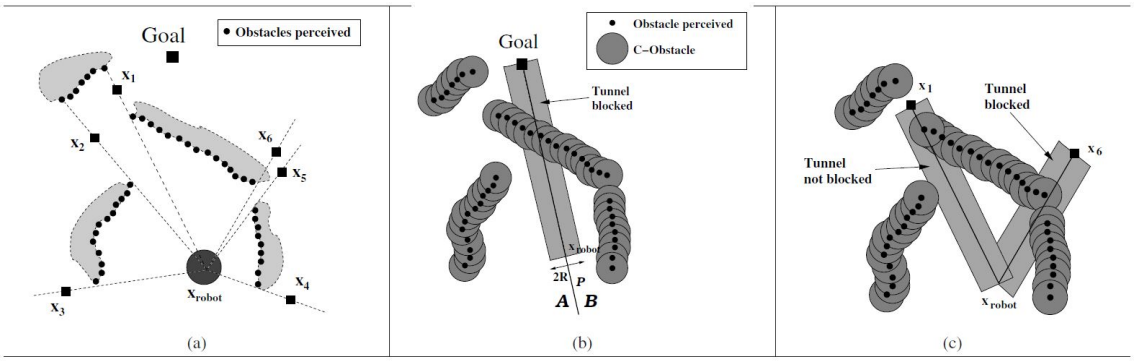

Figure 2.17. (a) This Figure illustrates the subgoal selector step of the ORM. (a) Robot, obstacle information perceived and the six candidate subgoals $x_1 ... x_6$. (b) The tunnel to the goal is blocked, thus there is no path within the tunnel. The C-Obstacles are the obstacle points enlarged with the robot radius. (c) The tunnel to x_6 is also blocked, but the one to x_1 is not because the distance between x_1 and x_2 is greater than robot diameter. Thus, there is a path that joins the current robot location and x_1. In this situation, x_1 is selected as the subgoal. [21].

- Candidate Set of Directions

  This step computes a set of motion constraints $S_{nD} \in [-\pi, \pi]$ in order to avoid obstacles. For each obstacle, it is computed the set of not desirable directions S_{nD} that is the union of two different subsets $S_{nD} = S_1 \cup S_2$ (Fig.2.18c). S_1 represent the side of the obstacle not suitable to achieve avoidance (Fig.2.18a), instead, S_2 is the exclusion region around the obstacle (Fig.2.18b). Let the reference frame be the robot frame, R the radius of the robot, D_s a security distance around the robot, θ_{target} the target direction and θ_{obst} the obstacle direction. When $\theta_{target} > \theta_{obst}$ (the set S_1 is on the right-hand side of the obstacle) $\phi_L = max(S_{nD})$ it is called left bound and in the opposite case $\phi_R = min(S_{nD})$ right bound.

- Motion Computation

  For each obstacle i, it is computed $S^i_{nD} = S^i_1 \cup S^i_2$. Thus, the final set is $S_{nD} = \cup_i S^i_{nD}$ and the set of desired direction for motion is the complementary

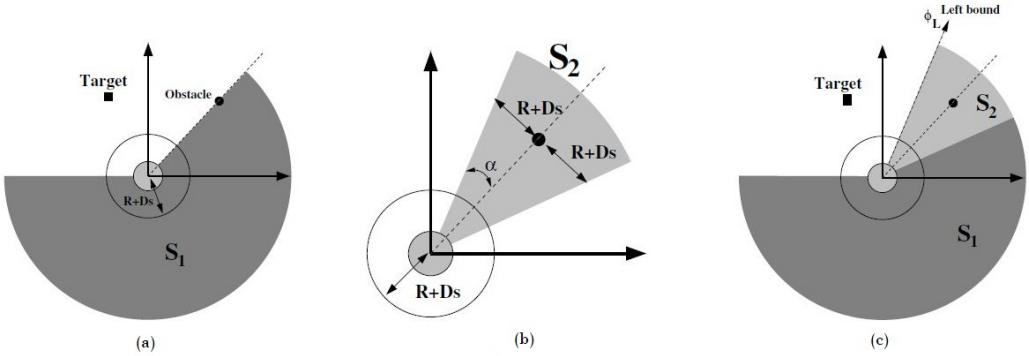

Figure 2.18. Set of motion constraints for an obstacle with the goal located in the left-hand side [21].

$S_d = \{[-\pi, \pi] \setminus S_{nD}\}$. After the definition of $\phi_L^{max} = max(\phi_L^i)$ and $\phi_R^{max} = max(\phi_R^i)$ we can compute the motion direction with the aid of three possible cases [21]:

- Case 1: $S_D \neq \emptyset$ and $\theta_{target} \in S_D$. Solution: $\theta_{sol} = \theta_{target}$ (Fig. 2.19a)
- Case 2: $S_D \neq \emptyset$ and $\theta_{target} \notin S_D$. Solution:

$$\theta_{sol} = \begin{cases} \phi_R^{max}, & if \ |\theta_{target} - \phi_R^{max}| < |\theta_{target} - \phi_L^{max}| \\ \phi_L^{max}, & otherwise \end{cases}$$

  The closest bound to the target direction is selected. For instance in Fig.2.19b the right bound is selected.

- Case 3: $S_D = \emptyset$. Solution: $\theta_{sol} = \frac{\phi_L^{max} + \phi_R^{max}}{2}$ (Fig.2.19c). The medium value between left and right bounds.

In the end this method can compute the most promising direction θ_{sol} to avoid obstacle collision and at the same time to drive the robot towards the target.

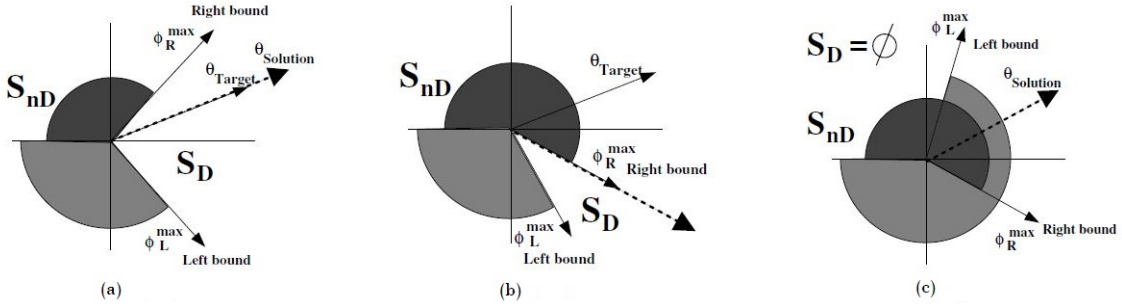

Figure 2.19. Computation of the direction solution in the three possible cases [21].

Dynamic Windows Approach (DWA)

The Dynamic Window Approach incorporates the dynamic of the robot reducing the number of admissible values of the velocity space in which we are looking for the optimal command control. It is a two steps method: firstly, it considers only the velocities which are safe with respect to the obstacle and then it chooses the velocity that maximizes the objective function [22]. For simplicity it is considered a motion control as translational and rotational velocity (v, w) [18]. U is defined by:

$$U = \{(v, w) \in R^2 \setminus v \in [-v_{max}, v_{max}] \wedge w \in [-w_{max}, w_{max}]\}. \tag{2.7}$$

- Candidate Set of Controls

  The candidate set of controls U_R is composed by the intersection of the maximum velocities of the vehicle U, the controls that generate safe trajectories U_A and the set of controls that are reachable in a short period of time given the vehicle acceleration U_D (Fig.2.20).

$$U_R = U \cap U_A \cap U_D. \tag{2.8}$$

  U_A is the set of admissible controls. A velocity is considered admissible if the robot can be stopped before collision by applying the maximum deceleration

(a_v, a_w).

$$U_A = \{(v, w) \in U | v \leq \sqrt{2d_{obs}a_v} \wedge w \leq \sqrt{2\theta_{obs}a_w}\} \tag{2.9}$$

where d_{obs} and θ_{obs} are respectively the distance to the obstacle and the orientation of the tangent to the trajectory over the obstacle. U_D is the dynamic window which contains only the controls reachable within the next time interval in order to take into account the limited motor acceleration.

$$U_D = \{(v, w) \in U \backslash v \in [v_0 - a_v T, v_0 + a_v T] \wedge w \in [w_0 - a_w T, w_0 + a_w T]\} \tag{2.10}$$

where T is the time interval and $\dot{q}_{t_i} = (v_0, w_0)$ is the current velocity.

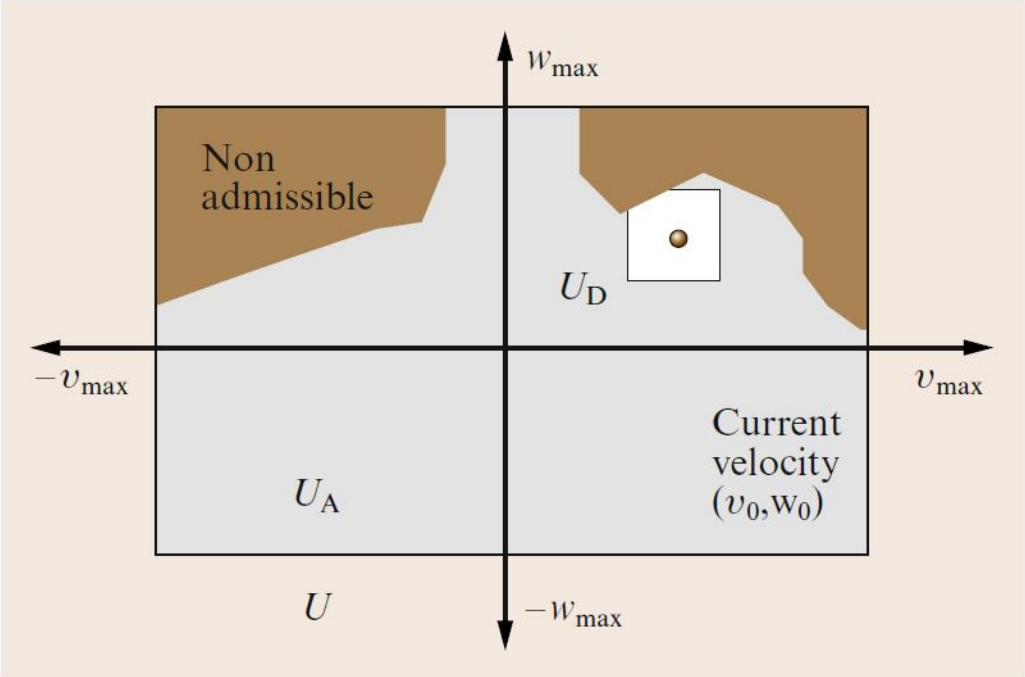

Figure 2.20. Subset of controls $U_R = U \cap U_A \cap U_D$, where U contains the controls within the maximum velocities, U_A the admissible controls, and U_D the controls reachable by a short period of time [18].

- Motion Computation

  In this phase it selects the control $u_i \in U_R$ which maximize the objective

function:

$$G(u) = \alpha_1 \cdot Goal(u) + \alpha_2 \cdot Clearance(u) + \alpha_3 \cdot Velocity(u), \tag{2.11}$$

where Goal(u) measures the robot alignment with the target direction and favors controls that offer progress to the goal, Clearance(u) favors velocities far from the obstacles and Velocity(u) evaluates the progress of the robot on the corresponding trajectory and favors high speeds.

This section shown some possible obstacle avoidance methods and their basic concepts. In general, on the one hand we have obstacle avoidance strategies that are local techniques to address the motion problem in an unknown environment. However, these methods can fall into local minima that translates in trap situations or cyclic motions. On the other hand, global path planning techniques compute a path free of collisions that guarantee global convergence but they fail in unknown scenarios. It seems clear that in order to build an autonomous system it is necessary to combine the best aspects of them.

Chapter 3

Autonomous Charging Project

The goal of the thesis is the development of an algorithm that allows a mobile robot to reach the recharging station and complete a correct docking. The contents of this chapter describes the whole work that led to the realization of the autonomous docking system. After a small introduction to the software platforms and a general description of the employed hardware, the projects is presented divided into its four stages: station detection, docking algorithm, obstacle detection and obstacle avoidance. In the first one it is described the process at the base of a precise and unique station identification, then the information provided by the latter is exploited by the docking algorithm to drive the robot toward the station following a predetermined strategy. The possible presence of obstructions during the docking has resulted in the development of the last two phases. Through the obstacle detection the objects are recognized and modeled in a certain data structure called point cloud which is used by the obstacle avoidance function to find the optimal direction to reach the station preventing collisions.

3.1 Software Tools

This section presents a small overview about the software tools that have been used to accomplish this project thesis. A larger part of the work is based on ROS, which is a collection of software libraries and tools that simplify the build of robot applications. Moreover, it takes care of all the development steps from the communication with hardware using drivers to the state-of-the-art algorithms. Any robotic project needs a simulation phase based on a certain software toolbox. For this purpose, the Gazebo environment has been chosen. It offers different services including the algorithm testing, the design of the robot and the ability to accurately and efficiently simulate the robot navigation in complex indoor and outdoor environments.

3.1.1 ROS Framework

ROS is the abbreviation for Robot Operating System and nowadays could be considered the standard for robot software platform.

"ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers." [23].

A Meta-Operating System is a system that performs scheduling, loading, monitoring and error handling by utilizing a virtualization layer between applications and distributed computing resources. In other words, ROS is not a conventional operating system such as Windows, Linux and Android but it runs only if an existing operating system has already been installed. This type of software is called

software framework because it is not only able to exploit all the features provided by the conventional operating system such as process management system, file system, user interface and program utility but it can also provide essential functions and libraries required for robot application programs such as data transmission/reception, scheduling and error handling.

In the following section will be presented the main features of the ROS framework and the principal concepts of the ROS computational graph.

Objectives of ROS

The ROS goal is to *"build the development environment that allows robotic software development to collaborate on a global level"* [24]. Therefore, the objective of this software framework is to support code reuse in robotics research and development. These are the main characteristics:

- Distributed framework of processes: the processes are called Nodes and enable the executable to be individually designed and run independently exchanging data systematically.

- Package management: processes with the same purpose can be grouped into packages in order to be easier to develop, share, modify and redistribute them.

- Repository: each package can be public to the developer's public repository in order to enable the collaboration.

- API: ROS is designed to simply call an API and insert it easily into the code being used.

- Language independence: ROS framework provides a client library to support various modern programming languages.

Computation Graph

The ROS run-time "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are processing data together [25]. The basic concepts are:

- Nodes: Process that performs computations. Usually a robot control system is based on many nodes and each of them has a specific goal. For example, one node controls the wheel motors and the other manages the depth camera sensor.

- Master: The ROS Master provides name registration. Without the Master, nodes aren't able to find other nodes, exchange messages or invoke services.

- Parameter Server: It is a part of the Master and allows data to be stored by key in a central location.

- Messages: The communication between nodes is possible thanks to these data structures. It supports standard primitive types (integer, floating point, boolean, etc...), arrays and also nested structures.

- Topics: The management of messages is based on a transport system with publish / subscribe semantics. At each topic is associated a specific type of message. When a node wants to publish something, it firstly registers its topic with the master and then it starts publishing messages on that topic. All the nodes that are interested in a certain type of data have to subscribe to that specific topic. In general, there can be multiple subscribers and multiple publishers and they are not aware of each other's existence. The ROS characteristic is the separation of the information production from its consumption.

- Services: They are a synchronous bidirectional communication based on a request / reply interaction. They need two types of message structure: one for the request and one for the reply. The service server node offers a service under a specific name, instead the service client sends a request and waits for the reply.

- Bags: They save and play back ROS message data. Bags are fundamental for developing and testing algorithms because they store data such as sensor readings that can be difficult to collect.

Summarizing the elementary operations of the ROS computational graph (Fig.3.1); the Master acts as a DNS server registering topics and services of the nodes and providing lookup information. Instead, nodes communicate with the Master to report their registration information and to ask the necessary parameters to establish appropriate connections with other nodes. The communication between nodes is direct, the Master provides only the correct parameters in order to agree upon a connection protocol. Usually in the ROS environment it is used the TCPROS protocol that works with standard TCP/PI sockets.

Rviz

ROS visualization (Rviz) is a powerful 3D robot visualization tool for ROS applications [26]. It provides a graphical interface to visualize the captured sensor data, the robot model and the environment maps in order to develop and debug robot controllers. It supports user specified polygons or markers and it allows to perform interactive movements with commands and data received from the user node. In Figure 3.2, it is shown a possible example of the robot model and the obstacles point cloud generated in the surrounding environment.

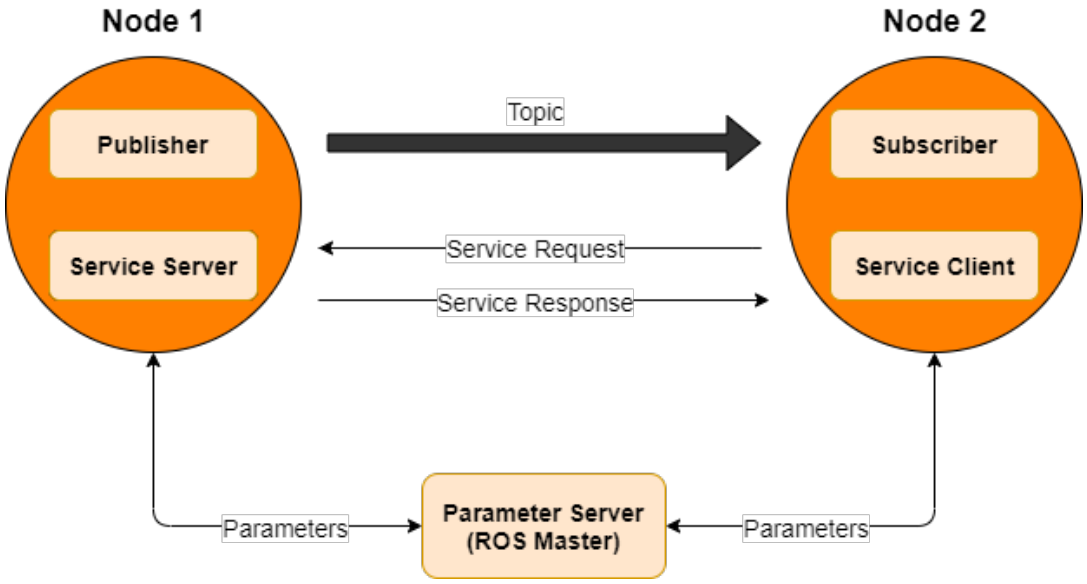

Figure 3.1. Message communication between Nodes.

3.1.2 Gazebo

Simulation is an essential phase to develop a robotic product. A well-designed simulator allows to test algorithms, design robots and model the sensors behaviour. Gazebo was born as a component in the *Payer Project* in 2004. Then in 2011 it became an independent project supported by Willow Garage, the same company that developed the open source software ROS. Gazebo permits to create 3D scenarios with robots, obstacles and other subjects and uses a physical engine for illumination, gravity and inertia (Fig. 3.2). As far as the robot design on Gazebo, the Unified Robotic Description Format (URDF) is employed. The URDF is an XML format used in ROS to describe all the elements of a robot. Unfortunately, it is not a universal description format because it can only specify the kinematics and dynamics of a single robot and it cannot specify the pose of the robot itself, the joint loops and many other properties such as friction property. This inflexibility of the URDF has led to the creation of a new format for the Gazebo environment

called the Simulation Description Format (SDF). It is a complete description for everything, from the world level down to the robot level. This is the reason why, in order to use a URDF file in Gazebo, some additional simulation-specific tags must be added. In fact, under the hood it will convert the URDF to SDF automatically [27]. Gazebo also supports several types of plugins that can be connected to ROS. They are chunks of code that are compiled as a shared library and inserted into the simulation. The most used plugins are the types which can be referenced through a URDF file [28]:

- *ModelPlugins*, to provide access to the physics of the model.
- *SensorPlugins*, to provide access to the sensor model.
- *VisualPlugins*, to provide access to the rendering of the model.

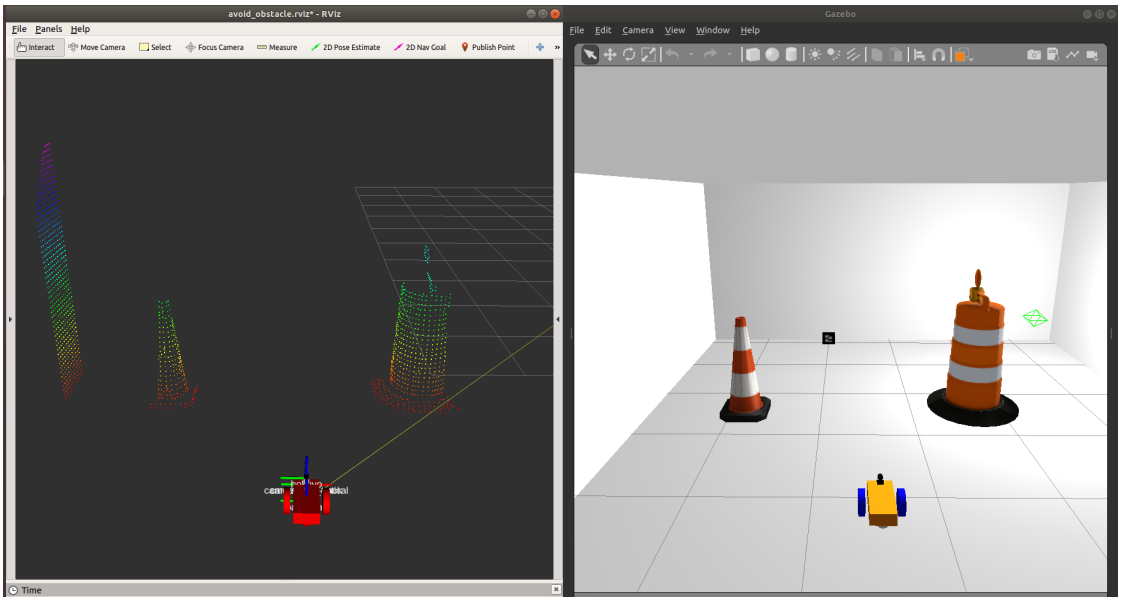

Figure 3.2. Rviz window example at left and at Gazebo window example at right.

3.2 Project Hardware

In this section, after a description of the main hardware employed for the autonomous recharging project, it is explained what are the reasons that led to the choice of these particular sensors.

NVIDIA Jetson Nano

NVIDIA Jetson Nano is an embedded system-on-module (SoM) and the latest developer kit released from the NVIDIA Jetson family (Fig. 3.3).

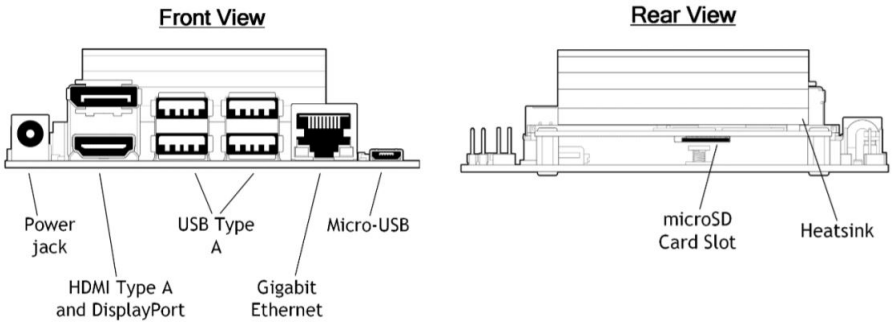

Figure 3.3. NVIDIA Jetson Nano front and rear view.

It includes an integrated 128-core Maxwell GPU, quad-core ARM A57 64-bit CPU, 4GB LPDDR4 memory, along with support for MIPI CSI-2 and PCIe Gen2 high-speed I/O [29] [30]. In the Table 3.1 are listed all the available ports and interfaces. It is an ideal fit for autonomous machines because it is tiny with a low power consumption (5-10w) and a high level of computational performance, thanks to which, it is possible to carry out real time computer vision tasks and to perform mobile-level deep learning operations. The module is released in the Jetson Nano Developer Kit, an easy way to get started using Jetson Nano that includes the software. It is a common solution for robotic applications because it runs the Linux operating system that allows the installation of the last version of

ROS, the natural choice for a multi-sensory autonomous robot.

	Jetson Nano
USB	(4x) USB 3.0 Type-A, USB 2.0 Micro-B
Camera	(2x) MIPI CSI-2 x2 (15-position Camera Flex Connector)
Display	HDMI 2.0, DisplayPort
Wireless	M.2 Key-E (PCIe x1)
Ethernet	Gigabit Ethernet (RJ45)
Storage	MicroSD card slot
Other	40-pin Header - (3x) I2C, (2x) SPI, UART, I2S, GPIOs
Power	Micro-USB (5V, 2.5A) or DC barrel jack (5V, 4A)

Table 3.1. NVIDIA Jetson Nano ports and interfaces.

ZED 2 Camera

ZED 2 is one of the most powerful stereo cameras on the market and the first that uses a neural network to reproduce human vision [31]. It is able to bring the stereo perception to a new level combining artificial intelligence and sensor hardware. The camera has an unrivaled field-of-view (FOV) and an excellent image quality (Table 3.2). The ZED 2 is equipped with a sensor stack (Table 3.3) that combined with the wide angle FOV greatly improves spatial perception. Moreover, it comes with an all-aluminum enclosure with thermal control that compensates focal length and motion sensor biases [32]. ZED 2 has been designed for the most challenging applications, from autonomous navigation and mapping to augmented reality and 3D analytics.

Camera	
Output Resolution	Side by Side 2x (2208x1242) @15fps 2x (1920x1080) @30fps 2x (1280x720) @60fps 2x (672x376) @100fps
Field of View Max.	110°(H) x 70°(V) x 120°(D)
Interface	USB 3.0/2.0 - Integrated 1.2m cable
Depth Range	0.3 m to 20 m (1 to 65.6 ft)
Depth Accuracy	$< 1\%$ up to 3m $< 5\%$ up to 15m

Table 3.2. ZED2 camera specifications.

Sensors	
Motion	Gyroscope, Accelerometer, Magnetometer
Environmental	Barometer Temperature

Table 3.3. ZED2 sensors.

Physical	
Dimensions	175 x 30 x 33 mm (6.89 x 1.18 x 1.3”)
Weight	166g (0.36 lb)
Operating Temp.	-10°C to +45°C (14°F to 113°F)
Power	380mA / 5V USB Powered

Table 3.4. ZED2 physical.

ZED 2 camera is the optimal solution for this project because it is cheaper than LiDAR sensor but, at the same time, it is possible to measure the distance from obstacles and computing the corresponding point cloud allowing us to implement autonomous navigation and mapping solutions. Furthermore, it is completely compatible with the NVIDIA Jetson Nano module and, given the fact that stereovision is based on images, it is able to exploit artificial intelligence algorithms in order to perform object detection and image classification.

3.3 Station Detection

In the section 2.2.1 it has been presented an overview of some possible recharging strategies. The fundamental concept behind all these techniques is the accurate and unique identification of the charging stations. It is obvious that the higher the accuracy of the station detection, the higher the docking precision. The station detection task can be implemented in different manners depending on the hardware. Firstly, a laser beam projector was used, then it was substituted with infrared beacons, until in recent years, computer vision techniques based on AR tags, QR codes or other types of landmarks have begun to be widely employed. For this project, it has been decided to identify the station with the AR tags and to use the ROS package ar_track_alvar to detect them in the surrounding environment.

3.3.1 AR Tag

Augmented Reality Tags or AR tags are commonly used for augmented reality applications. For instance, they are employed to generate virtual objects, games, and animations within the real world. There exist several types of AR tags generated by different algorithms. Each of them has its pros and cons, some are

computationally less intensive to generate than others, while some are more difficult to detect at distance. Usually they appeared as a black and white squared image with some patterns within a black border.

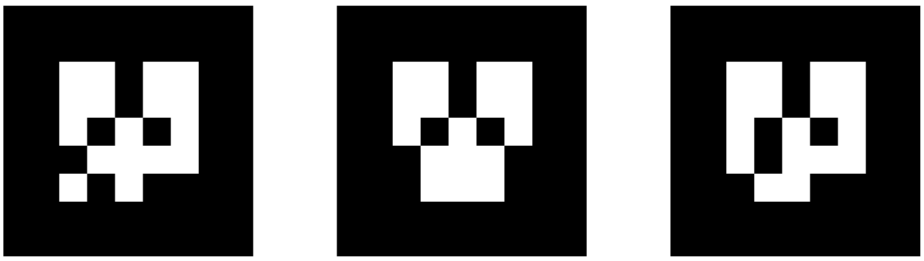

Figure 3.4. Examples of AR tags generated with ar_track_alvar package [33].

Besides their importance in augmented reality applications, they are also used in computer vision tasks such as landmarks to estimate the pose of a certain target.

3.3.2 Ar_Track_Alvar Package

Ar_track_alvar is a ROS package based on Alvar [34] , an open source AR tag tracking library. The latter has the goal of detecting and tracking 2D markers and keeps their pose estimation as accurate as possible [35]. Ar_track_alvar has 4 main functionalities [36]:

1. Generation of AR tags of varying size, resolution and data/ID encoding.
2. Identify and track the pose of individual AR tags.
3. Identify and track the pose of multiple tags (bundles).
4. Using camera image to automatically calculate spatial relationship between tags in a bundle.

This package, based on AR tags size and distortion in the captured image,

can estimate their position and orientation in relation to a camera frame knowing "a priori" their real dimensions. In this work, the station is identified by a single marker, so we exploited the second functionality of the package. For the identification and the tracking of the AR tags it is possible to combine the information provided to the package with depth data or point clouds in order to improve the pose estimates. Ar_track_alvar has two operating modes, individualMarkersNoKinect and individualMarkers which require different parameters for the depth data integration (Table. 3.5). Although the ZED 2 stereo camera can provide a point cloud of the surrounding environment, this specific type of data structure is not supported by the individualMarkers option.

Arguments	individualMarkersNoKinect	individualMarkers
marker_size	width in centimeters of one side of the black square marker border.	
max_new_marker_error	threshold that determines when new markers can be detected under uncertainty.	
max_track_error	threshold that determines how much tracking error can be observed before a tag is considered to have disappeared.	
camera_image	topic that provides camera frames.	topic that provides point cloud.
camera_info	topic that provides the camera calibration parameters in order to rectified the image	
output_frame	frame name that the published Cartesian locations of the AR tags will be relative to.	

Table 3.5. ar_track_alvar arguments.

When the ar_track_alvar node is launched it publishes its results on two separate topics: *ar_pose_marker* and *tf*. They transport respectively different type of message *ar_track_alvar_msgs/AlvarMarker.msg* [37] and *tf/tfMessage.msg* [38]. However, they provide similar information because they are based on the *geometry_msgs/PoseStamped.msg* [39] and *geometry_msgs/TransformStamped.msg* [40] types (Fig. 3.5).

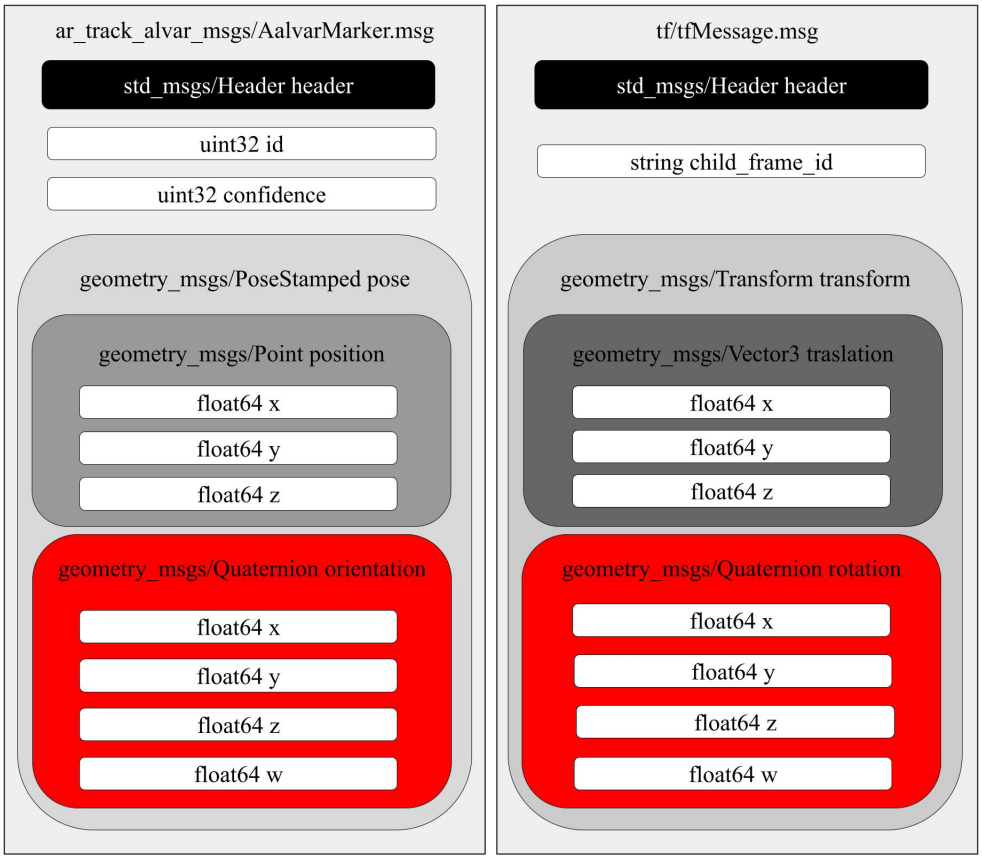

Figure 3.5. At left *ar_track_alvar_msgs/AlvarMarker.msg* and at right *tf/tfMessage.msg*.

In particular, it publishes the position and the orientation of the AR tag with respect to the output_frame through the 3 Cartesian coordinates x, y, z and the

quaternions representation x, y, z, w. A coordinate frame is a set of orthogonal axes attached to a body that serves to describe the position of points relative to that body [41]. In this case, usually, the output_frame is a frame attached to the camera or stereo camera sensor.

This information is sufficient to drive the robot towards the station, obviously, under the hypothesis that the AR tag is in the field of view of the camera. Otherwise, the AR tag is not detected and nothing is published.

During the execution, ar_track_alvar node publishes another topic called *visualization_marker*, which transports rviz messages for visualization purposes. It displays a green square block at the location of each detected tag not only in the rviz environment but also in the camera image (Fig. 3.6).

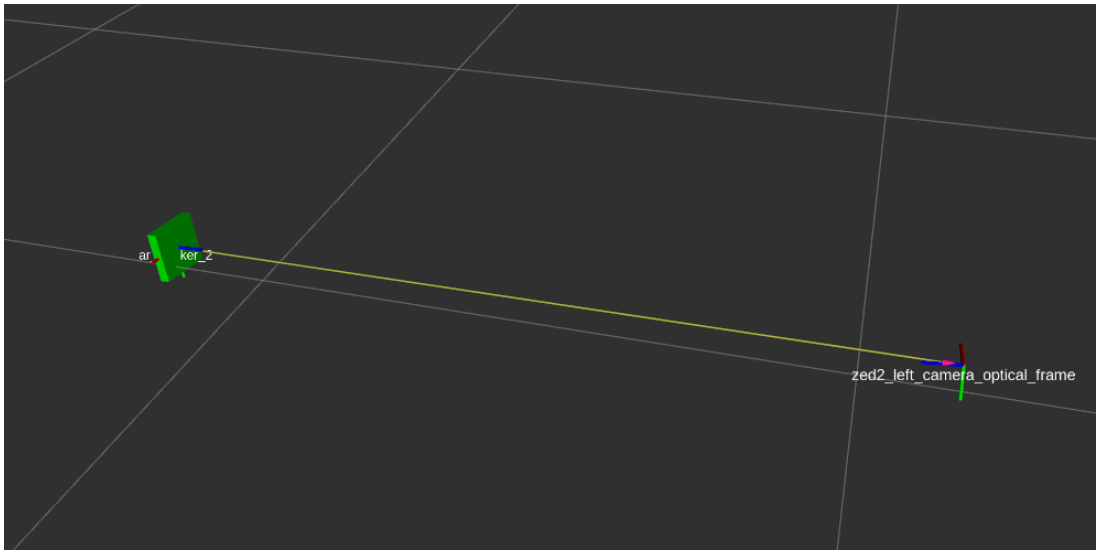

Figure 3.6. Example of an AR tag detection test in Rviz environment.

3.4 Docking Algorithm

In the previous section it has been addressed the problem of recharging station detection. Now, the obtained information are exploited in order to drive the robot towards its target location. Before going into detail of the docking algorithm two hypothesis has to be made:

1. The robot always knows its position with respect to the map frame. For example, it can be provided by the GPS in open areas or by the SLAM technique in enclosed spaces.

2. Before running the algorithm it is provided the position and the orientation of the recharging station that correspond to the AR tag location with respect to the map frame.

These two hypotheses allow the robot to drive towards its destination when the AR tag is not detected. Otherwise, when the AR tag is visible, the data provided by the ar_track_alvar node are sufficient. However, for a correct and precise docking, the station pose is not enough and the information generated by the ar_track_alvar package are required. The convention for the station orientation parameter relative to the map frame, which is passed to the docking algorithm, is shown in the Figure 3.7.

The distance d, the angle θ and the orientation ϵ are the only data that are needed for the docking phase. d is the distance between the AR tag and robot position, θ is the angle between robot and the perpendicular position to the station and ϵ is the orientation difference between the AR tag and the robot frames. The perfect docking is defined by the angle θ and the orientation ϵ equal to 0 at an established distance $d_{docking}$ to the AR tag. In other words, the robot has to be on the perpendicular line and in front with respect to the recharging station. In

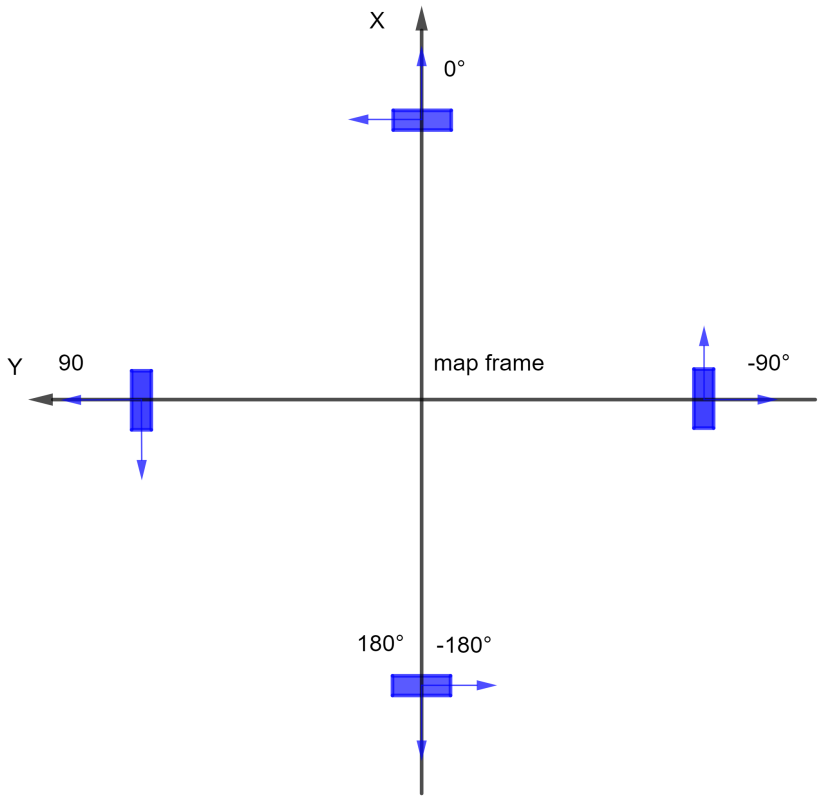

Figure 3.7. Convention for station orientation with respect to the map frame.

Figure 3.8 are illustrated the 4 possible combinations of angle θ and orientation ϵ. For instance, θ is positive when the robot is to the right of the station and negative in the other case.

The algorithm has two main functionalities: firstly it has to compute distance, angle and orientation through the data received by the sensors and secondly it has to use the previous extracted information to drive the robot toward its target. These functionalities are implemented by two different nodes: the distance_angle

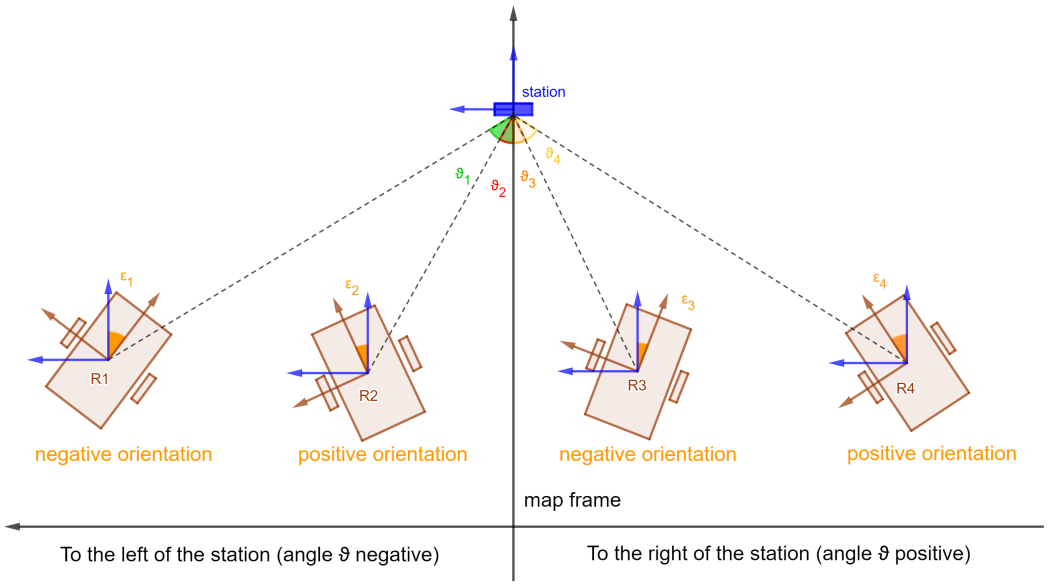

Figure 3.8. Convention for angles and orientations between robot and station.

node and the docking node.

3.4.1 Distance_angle Node

The main purpose of this node is to extract the distance, the angle and the orientation between robot and station from the sensor data. Later, it publishes the results on the *distance_angle/DistanceAngleOrientation* topic which transport the message type *DistanceAngleOrientation.msg* (Fig. 3.9).

There are two possible scenarios: the AR tag is in the field of view of the stereo camera sensor and it is visible or the AR tag is not detected due to obstacles or too high distances.

AR tag visible

In this scenario, the node, which is subscribed to the *tf* topic, reads the data that are published by the ar_track_alvar node. The *tfMessage.msg* type (Fig.

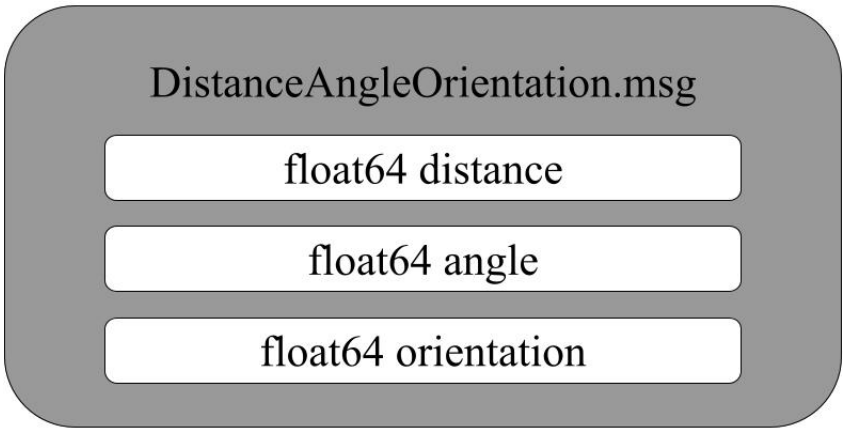

Figure 3.9. DistanceAngleOrientation.msg.

3.5) uses the quaternion convention for the AR tag orientation and provides the coordinate of the station distance with respect to the robot frame (x_p, y_p). In order to compute the angle θ it is necessary to have the robot distance coordinate (x_d, y_d) in the station reference system (Fig. 3.10). Thus, firstly, the orientation of the station is converted in the Roll, Pitch and Yaw angles.

$$\begin{cases} Roll = \arctan \frac{2(q_0 q_1 + q_2 q_3)}{1-2(q_1^2+q_2^2)} \\ Pitch = \arcsin 2(q_0 q_2 - q_3 q_1) \\ Yaw = \arctan \frac{2(q_0 q_3 + q_1 q_2)}{1-2(q_2^2+q_3^2)} \end{cases} \tag{3.1}$$

Then, a rotation matrix with the Yaw angle, which corresponds to the orientation ϵ that is the difference between the robot frame and the station frame, is applied.

$$\begin{cases} x_d = x_p \cos\epsilon - y_p \sin\epsilon \\ y_d = x_p \sin\epsilon + y_p \cos\epsilon \end{cases} \tag{3.2}$$

Obtained x_d and y_d, which are the coordinates of the station distance in the station reference system, it is possible to compute the distance d and, in particular,

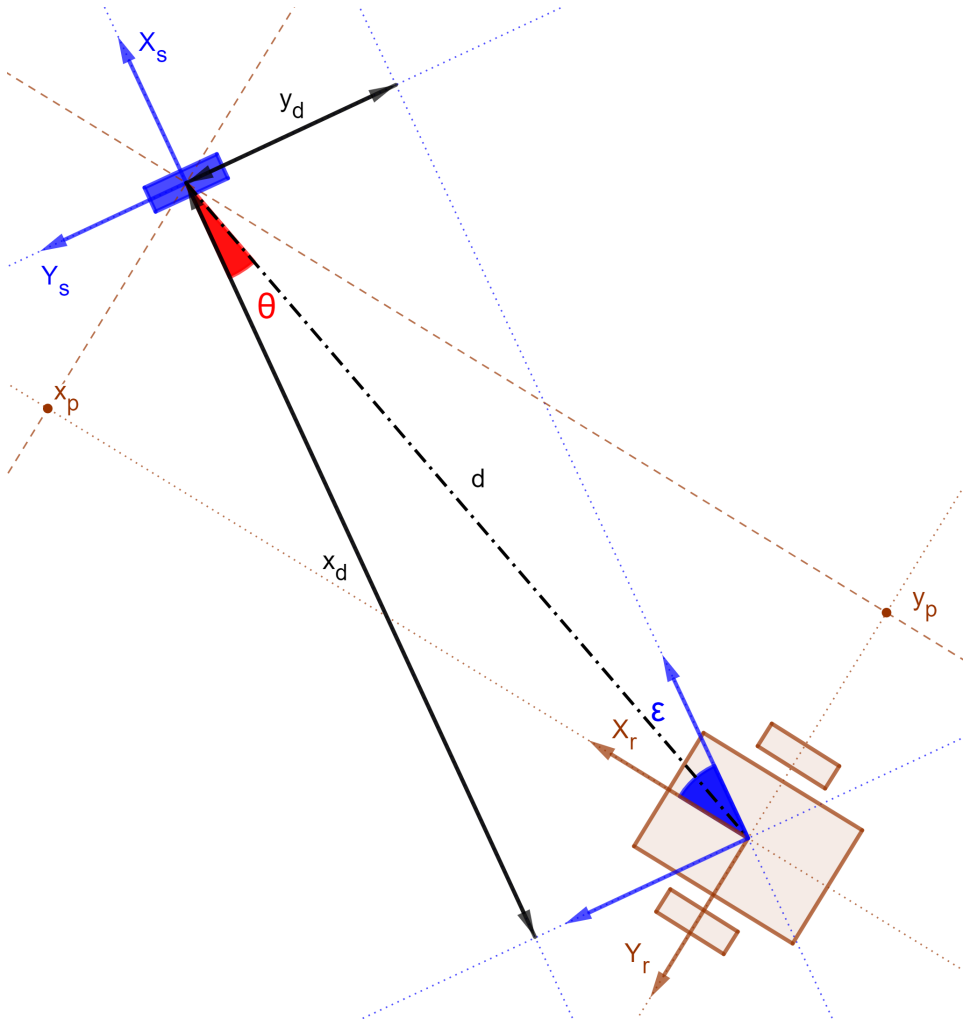

Figure 3.10. Distance, angle and orientation between robot and station computed through ar_track_alvar information. The station frame is in blue and the robot frame is in brown.

the angle θ between the robot position and the station perpendicular line.

$$d = \sqrt{x_d^2 + y_d^2} \tag{3.3}$$

$$\theta = \arctan(\frac{y_d}{x_d}) \tag{3.4}$$

AR tag not detected

When the AR tag is not visible, the distance_angle node uses the current robot position (*robot_pose_x*, *robot_pose_y*) and orientation ϵ_r and the station position (*station_pose_x*, *station_pose_y*) and orientation ϵ_s, which is provided as argument, to obtain the necessary information (Fig. 3.11). Both pose and orientation data are relative to the map frame reference system. Thus, as first thing, we compute the distance coordinates in the map reference system:

$$\begin{cases} \delta_x = |station\_pose\_x - robot\_pose\_x| \\ \delta_y = |station\_pose\_y - robot\_pose\_y| \end{cases} \tag{3.5}$$

Then, it is needed to apply a rotation by an angle equal to the station orientation ϵ_s with respect to the map frame to obtain the distance coordinates in the station reference system (x_d, y_d).

$$\begin{cases} x_d = \delta_x \cos\epsilon_s - \delta_y \sin\epsilon_s \\ y_d = \delta_x \sin\epsilon_s + \delta_y \cos\epsilon_s \end{cases} \tag{3.6}$$

Finally, it's possible to compute the distance d, the angle θ and the orientation ϵ.

$$d = \sqrt{x_d^2 + y_d^2} \tag{3.7}$$

$$\theta = \arctan(\frac{y_d}{x_d}) \tag{3.8}$$

$$\epsilon = \epsilon_r - \epsilon_s \tag{3.9}$$

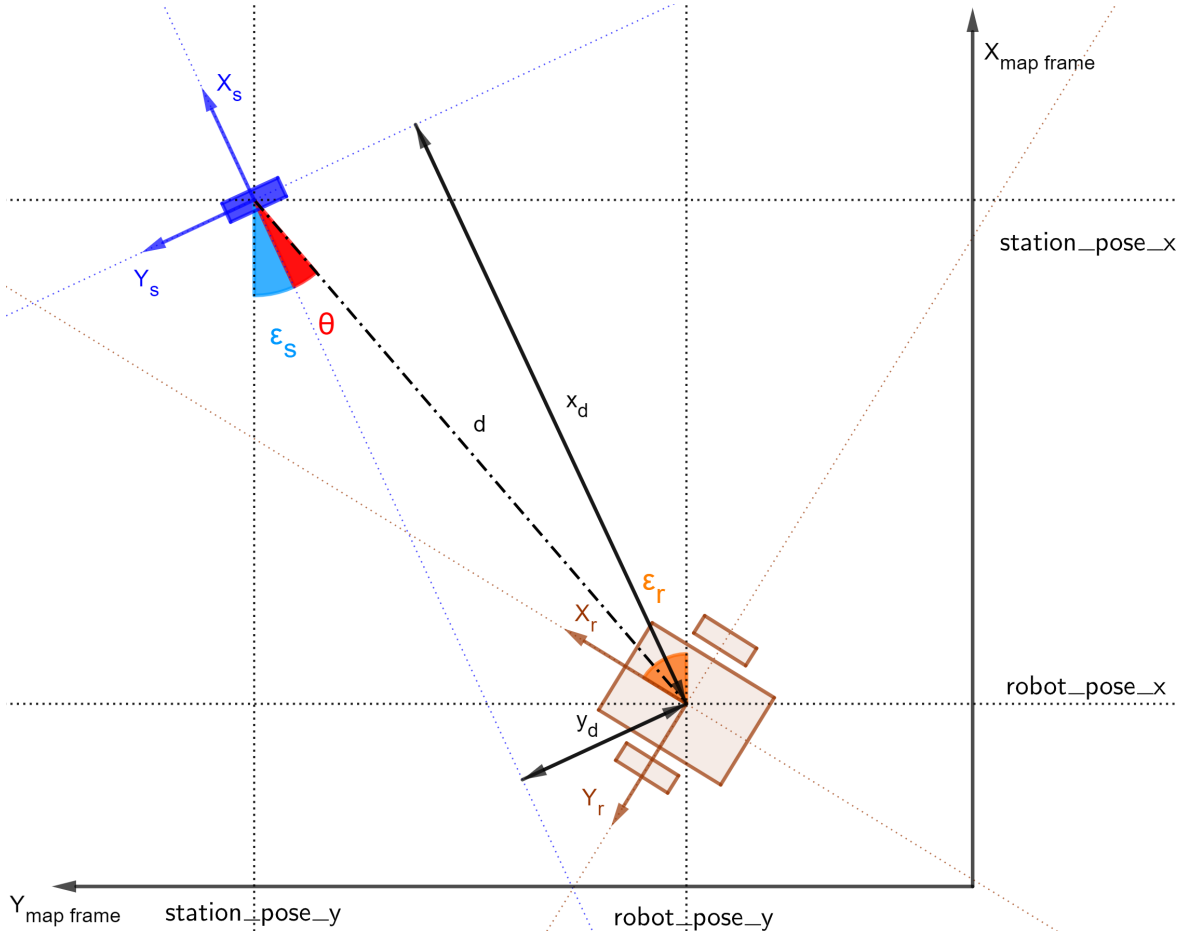

Figure 3.11. Distance, angle and orientation between robot and station computed through the position and orientation of robot and station relative to the map frame. The map frame is in black, the station frame in blue and the robot frame in brown.

The secondary goal of this node is to compute the same type of information (distance, angle and orientation) between the robot and a target point. These data are obtained in the same way as they are computed when the tag is not detected, but, in this case, the target is not the station but the location where the robot is driven when the docking fails. This package publishes the *DistanceAngleOrientation.msg* message type on the *distance_angle/DistanceAngleOrientation_retry* topic.

3.4.2 Docking Node

The data previously computed are exploited by the docking node which uses them to drive the robot toward the AR tag. The docking algorithm is composed of 3 main parts (Fig. 3.13).

- Docking: it is the core of the algorithm, the robot moves towards the station exploiting the distance, angle and orientation information read in the *distance_angle/DistanceAngleOrientation* topic.
- Docking failed: when the docking failed, because of not acceptable values, it is necessary to drive the robot in a position that is optimal in order to try again. This is the goal of this part, the robot is driven towards the target point exploiting the distance, angle and orientation information read in the *distance_angle/DistanceAngleOrientation_retry* topic.
- Docking end: in this case the docking process ended with acceptable parameters.

The general behavior of the entire algorithm is described in the Fig. 3.13. However, before getting into details, some useful constants and variables are introduced for a better understanding of the flowchart representation.

Useful constant:

- AR_DIST: maximum distance value at which the robot has to stop in front of the station.
- $MAXANGLE$: maximal angle that defines the beginning of the PHASE 2.
- $MAXANGLE\_DOCKING$: maximal angle that define a correct docking.
- $MAXORIENT\_DOCKING$: maximal orientation that define a correct docking.

- TAG_AREA: distance where the PHASE 1 changes the approaching angle.
- $DOCKING\_RETRY\_AREA$: distance to reach respect with the station in order to retry the docking.

Useful variable:

- $docking\_failed$: boolean variable that signals when the docking is failed.
- d: station distance.
- θ: station angle.
- ϵ: station orientation.

The base concept of the docking process is simple: reach the position perpendicular to the station as soon as possible and then adjust the orientation with the AR tag. The PHASE 1 has the goal to drive the robot on the perpendicular line, instead, the orientation is in charge of the PHASE 2. Theoretically, the optimal path to reach the perpendicularly is moving the robot with an angle of 90° with respect to the recharging station. However, in our scenario, it is fundamental that the ZED 2 camera sensor is able to detect the AR tag as soon as possible. This requirement has led to divide the PHASE 1 in two parts (Fig. 3.12):

- $d > TAG\_AREA$: When the robot is far from the station, it reaches PHASE 2 with an angle of 80°.
- $d <= TAG\_AREA$: When the robot is close to the station it reaches the PHASE 2 with an angle of 45°. This choice allows not only to detect the AR tag, but also not to lose it during its movements to reach the perpendicular pose.

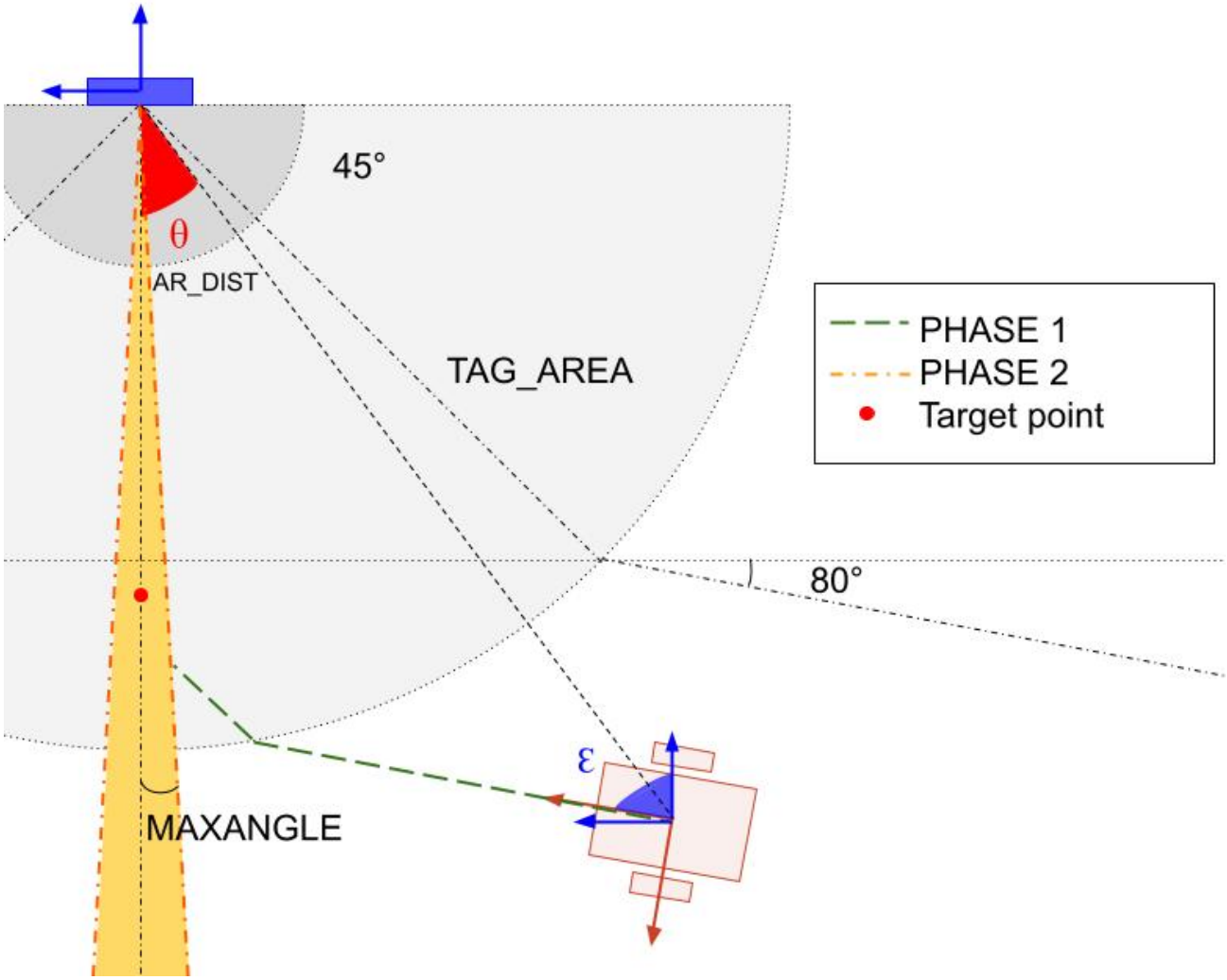

Figure 3.12. Docking phase.

PHASE 2 begins once the vehicle reaches an area in front of the AR tag that is defined by the MAXANGLE constant. It's evident that the larger the latter, the greater the area of the cone to start the second phase. But, the larger the cone area, the lower the accuracy of the docking. In the event of the robot leaves this cone area during the approach of the station, the PHASE 1 drives it inside.

Currently, the InnoTech project doesn't include a charging mechanism, not even the charging station. Therefore, it has been necessary to find a definition of successful docking. In this thesis the docking is supposed to be correct when the angle and the orientation respects two conditions:

- $-MAXANGLE\_DOCKING < \theta < MAXANGLE\_DOCKING$.
- $-MAXORIENT\_DOCKING < \epsilon < MAXORIENT\_DOCKING$.

Otherwise, it is considered a failure, and appropriate actions have to be taken by the docking failed part. The latter is in charge of moving the robot to a target point, which is located in front of the station at a distance of $DOCKING\_RETRY\_AREA$ meters.

3.5 Obstacle Detection

"Obstacle detection is the process of using sensors, data structures, and algorithms to detect objects or terrain types that impede motion" [42].

This topic has many aspects including the world models that represent sensor data in a convenient form, the mathematical model of the interaction between objects and robots and the algorithm that is able to process all in order to infer obstacles.

Hazard detection is often used as a synonymy of obstacle detection, but, sometimes, they are applied in different domains. For instance, obstacle detection is often used in ground vehicle navigation, instead, hazard detection is applied to the aircraft or spacecraft landing process.

An optimal obstacle detection system is obtained combining passive and active technologies. Generally, the best solution is composed of vision systems such as vision cameras or stereo cameras and distance sensors. Proximity sensors or distance sensors are some types of hardware such as IR sensors, sonars and lasers

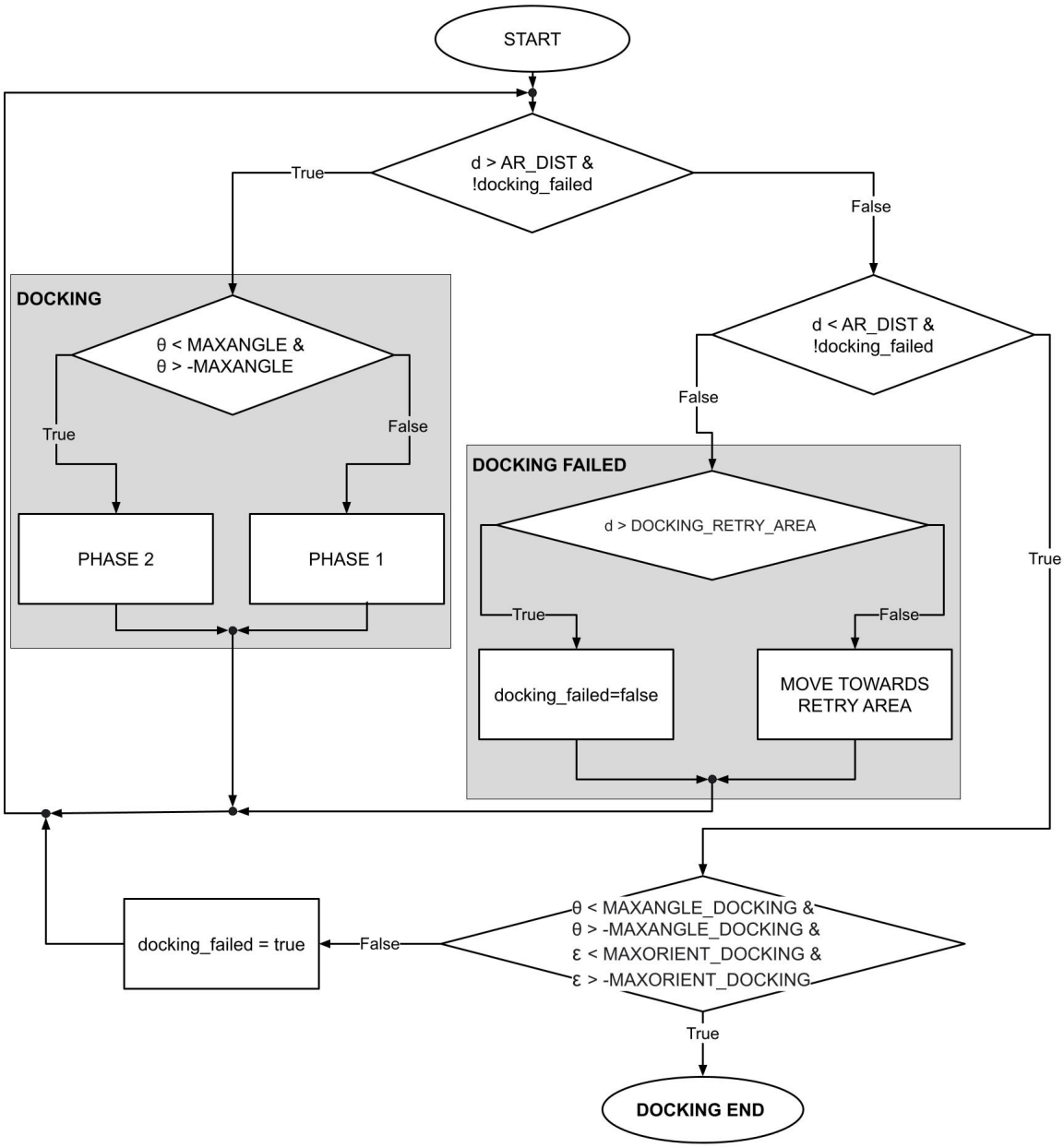

Figure 3.13. Docking algorithm flow chart.

which are used to measure the relative distance between the sensor and objects in the surrounding environment [43]. In other systems, this level of redundancy and sensor combination is not possible because of the high cost of technology.

For instance, our robotic system is equipped with a ZED 2 camera. However, the latter is not able to cover all the area in front of the robot having a 120° field of view and a minimal detection area of 0.3m as presented in the Table 3.2. These limitations have led to combining the stereo camera sensor with proximity sensors in order to be able to cover a 180° field of view (Fig. 3.14).

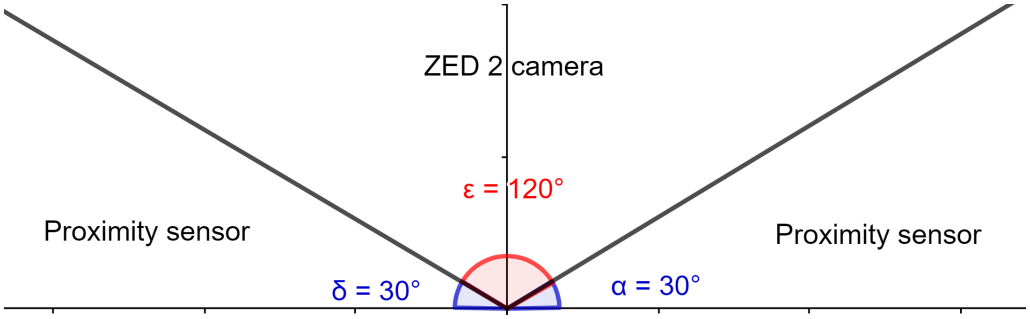

Figure 3.14. Obstacle detection algorithm field of view.

This type of sensor combination doesn't provide the redundancy that high level robotic systems have, because, the proximity sensors compute a distance measurement in an area where the ZED 2 camera cannot detect obstacles. Unlike proximity sensors, ZED 2 camera doesn't compute a direct measurement of the distance. In fact, one of the features of this sensor category is the capability of generating a more complex data structure of the surrounding environment which is called point cloud.

3.5.1 Point Cloud

Point cloud is a collection of multi-dimensional points which is commonly used to represent 3D information about the world (Fig. 3.15). Besides the geometric coordinates x, y and z each point can hold additional information such as RGB colors, intensity values, ... In the graphic and physics simulation world this type of data structure has been adopted for decades. However, nowadays, it is

widely used and more relevant for two main trends: the birth of many applications in the robotics field such as navigation, obstacle avoidance, object recognition, grasping and manipulation and the availability of cheap point cloud acquisition devices. The reason why point clouds became so popular in the autonomous machine world resides in the fact that they have a high level of depth sensing and they can provide direct and precise distance information that helps vehicles detect obstacles. Moreover, given the accuracy of these data structures, robotic systems rely on depth not just for navigation but also for localization purposes.

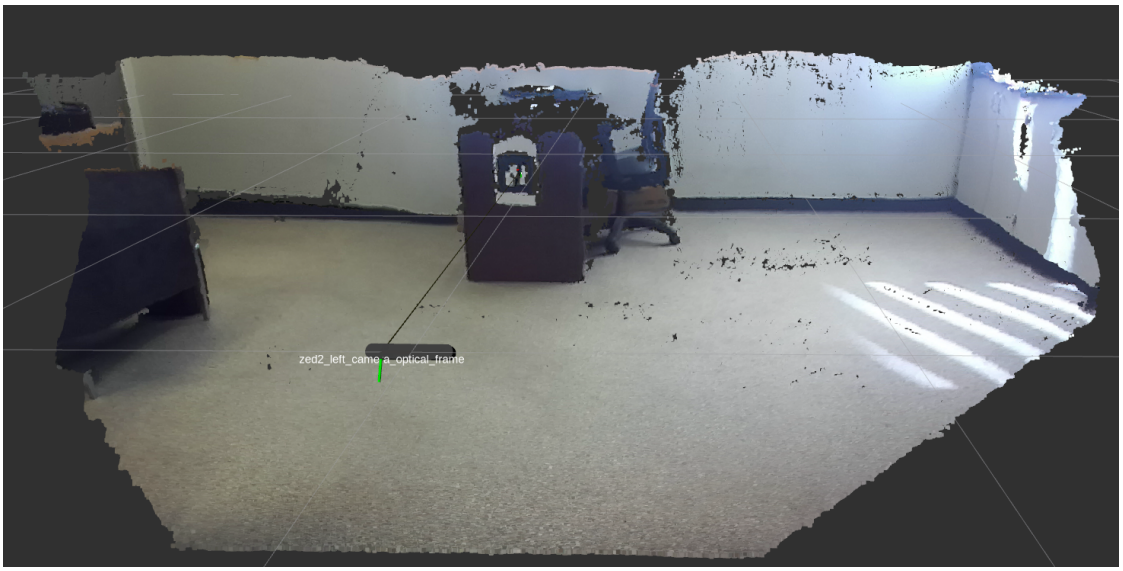

Figure 3.15. Point cloud example.

3.5.2 Obstacle Recognition inside Point Clouds

Once it has been obtained a point cloud representation of the world from the ZED 2 camera sensor, it is needed to extract obstacles from this data structure. As previously described, the latter is a model of the surrounding environment based on multi-dimensional points, however, not all the points represent possible obstacles. This is evident in the Figure 3.17 where only the white points identify

obstacles, while the others represent the ground. In order to handle point clouds a specific library called Point Cloud Library (PCL) has been used.

Point Cloud Library (PCL)

PCL is a standalone and open source library for 2D/3D images and point clouds processing [44]. The basic structure is a templated C++ class called PointCloud which basically contains the following data fields [45]:

- *width*: it specifies the width of the point cloud dataset in the number of points.
- *height*: it specifies the height of the point cloud dataset in the number of points.
- *points*: it contains the data array where all the points are stored.

This library is widely used because it allows the usage of different point types (PointXYZ, PointZYZI, PointXYZRGB, ...) and, above all, it handles point cloud data structure using smart pointers. However, the ROS environment uses a different structure called PointCloud2 message [46]. It is a general representation containing a header defining the point cloud structure. For instance, the ZED 2 camera publishes the point cloud data on appropriate topics exploiting this type of message. Fortunately, the PCL library also provides the functions that allow the conversion between these two frameworks: pcl::fromROSMsg and pcl::fromROSMsg.

Voxel Grid Approach

Given the high CPU resources consumption of the process in charge of handling point clouds, before working on them, it has been applied a downsampled function to the points dataset using a voxelized grid approach. A voxel identifies a value

on a regular grid in three-dimensional space. This function creates a 3D voxel grid and for all the points present in each voxel it approximates them with its centroid. It is slower than approximating them with the center of the voxel but this approach allows a more meaningful and accurate reduction in the number of points for the surface representation (Fig. 3.15).

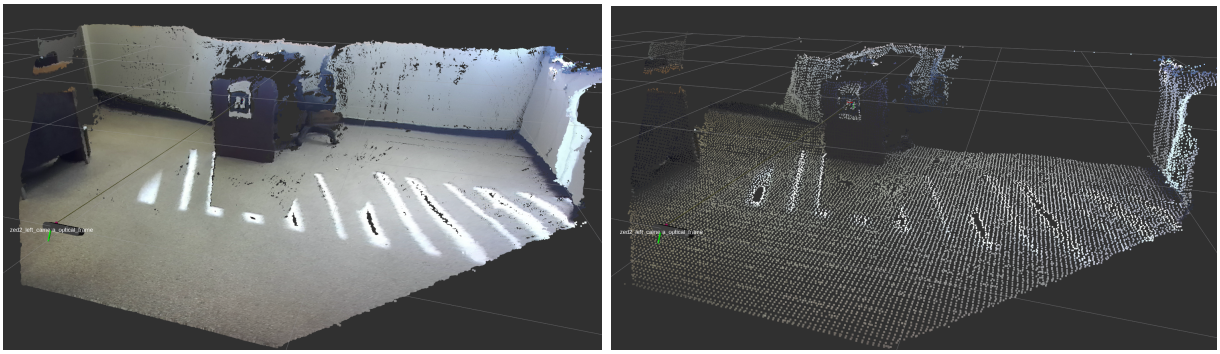

Figure 3.16. Example of voxel grid effect: at left the native point cloud and at right the point cloud after voxel grid filtering

Obtained a lighter point cloud representation there are just two final steps for obstacle detection: filter out the ground and extract different obstacles from the obstacle point cloud.

Filter out the Ground

Extract obstacles and ground from a point cloud is a challenging task which is solved by the RTAB-Map ROS package. The latter can be used in different scenarios, such as generating 3D point clouds of the environment or creating a 2D occupancy grid map for navigation. Among its main functions there is a nodelet called *rtabmap_ros/obstacles_detection* [47], which has the exact goal we are looking for. It is important to point out that in order to guarantee a high level of reliability the camera must see the ground. Since in a real world environment the ground is not even, the latter is segmented by normal filtering. It means that all

points with the normal in the $+z$ direction ($\pm$ fixed angle) are labelled as ground and all the others are labelled as obstacles. In the Figure 3.17 it is possible to see a ZED 2 camera point cloud representation where obstacles points are white and others are colored.

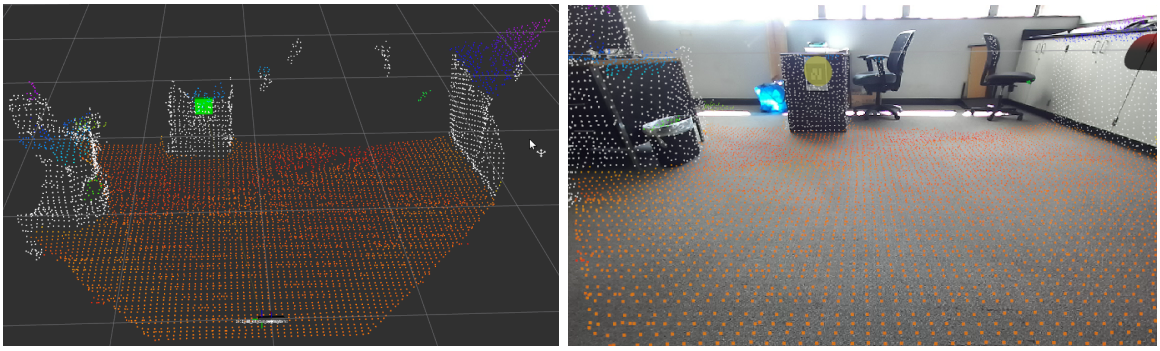

Figure 3.17. Point cloud example using a voxel grid approach and the *rtabmap_ros/ obstacles_detection* nodelet.

From this last example it's evident the drawback of this approach, in fact, all the parallel surfaces with the floor are labeled as ground. Nevertheless, it remains a very good method for the obstacle extraction.

Euclidean Clustering Extraction

The last part of the obstacle detection process is the identification of different objects in the obstacle point cloud. For this purpose, clustering methods which divide an unorganized point cloud dataset P in smaller parts are usually employed. In most cases they rely on spatial decomposition techniques based on a proximity measure that allows the data to be grouped together. This measure, which represents the similarity between points, is usually the Minkowski norm with its most popular instances such as Manhattan (L1) and Euclidean (L2) distance metrics. The system, which is in charge of recognizing the individual objects, needs to understand what is an object point cluster first and what differentiates it from

another point cluster.

From the mathematical point of view a cluster is defined as follows. Let $O_i = \{p_i \in P\}$ be a distinct point cluster from $O_j = \{p_j \in P\}$ if:

$$\min ||p_i - p_j||_2 \geq d_{th} \tag{3.10}$$

where d_{th} is a distance threshold and p_i and p_j are a set of points. The above equation states that, given two set of points belonging to the same point cloud model P, if the minimal distance between them is larger than an imposed threshold, then the points in p_i are set to belong to a point cluster O_i and the ones in p_j to another distinct point cluster O_j. In other words, this approach is based on the concept of nearest neighbors in order to assign points to the same point cluster and it works in a similar way to a flood fill algorithm. The algorithmic steps would be:

1. create a k-d tree representation for the input point dataset P for finding the nearest neighbors. K-d tree is a special case of a binary tree that organizes points in a k-dimensional space and every node is a k-dimensional point.

2. set up an empty list of clusters C, and a queue of the points that need to be checked Q;

3. then for every point $p_i \in P$, perform the following steps:

   - add p_i to the current queue Q;
   - for every point $p_i \in Q$ do:
     - search for the set P_i^k of point neighbors of p_i in a sphere with radius $r < d_{th}$;
     - for every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q;

- when the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list;

4. the algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point cluster C.

If the Euclidean distance metrics is used to define the proximity measure then we speak about the Euclidean clustering method [48].

In summary, using clustering techniques, it is possible to identify different objects in the same point cloud. These results allow the measurement of the distance between objects and the computation of the obstacle range. The latter is defined as the portion of the ZED 2 camera field of view that is occupied by the obstacle (Fig. 3.18).

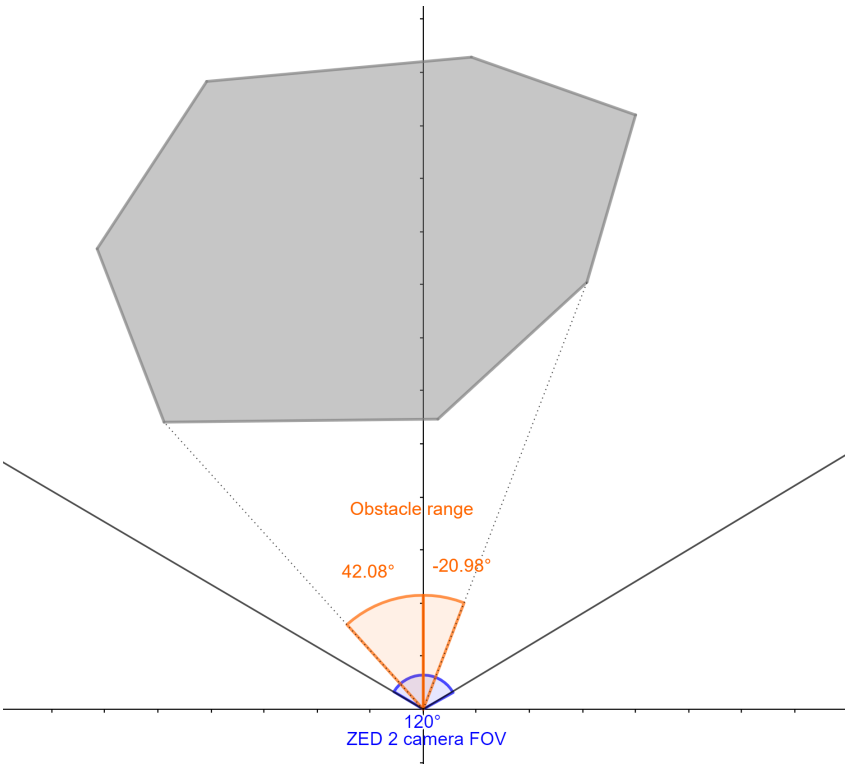

Figure 3.18. Example of obstacle range [-20.98°, 42.08°].

The algorithmic steps that need to be taken could look as follows:

1. extract all the detected obstacles from the point cloud;
2. select the two closest objects;
3. compute the distance between them. If the two detected obstacles are too close and the robot is not able to move through them, they are seen as a single object and the two clusters are combined in a single one. In the other case it focuses its attention only on the closest obstacle.
4. compute the obstacle range and the relative distance from the point cloud extracted at the previous point.

In Figure 3.19 it is shown a result obtained during ZED 2 camera tests. In this case the obstacles are not distant enough to allow the robot passage and the obstacle range is computed considering them a single objects.

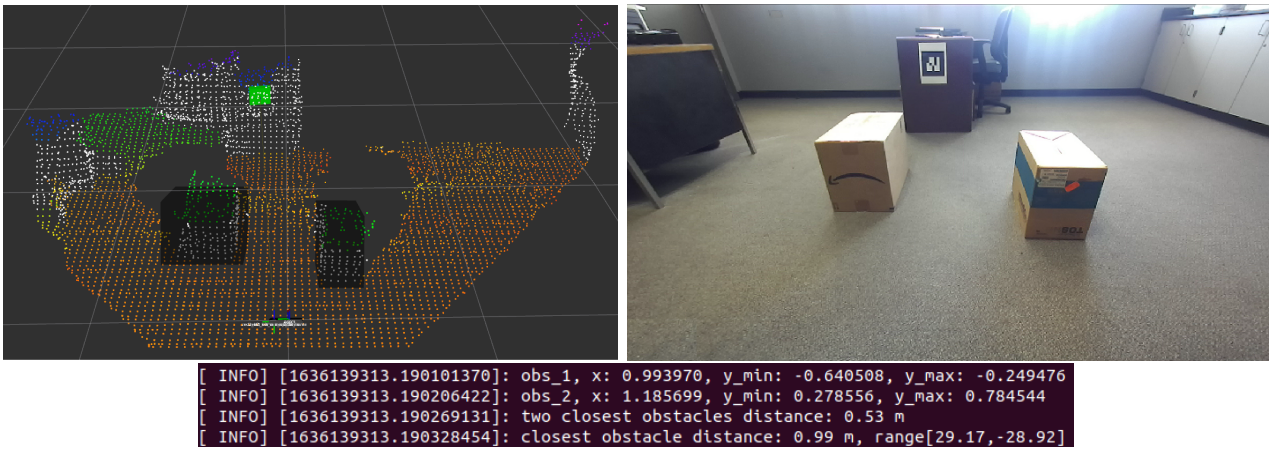

Figure 3.19. Example of computing distance between the 2 closest obstacles.

Later, the obstacle avoidance algorithm will exploit the information about detected objects and the distance measurements provided by the proximity sensors in order to prevent collisions.

3.6 Obstacle Avoidance

Currently, the docking algorithm can drive the mobile robot to the docking station following an ideal path that is based on the concept of reaching as soon as possible the station perpendicular line and then adjust its orientation. This process is based on the hypothesis that in the surrounding environment there are no obstacles. Unfortunately, in the real world this condition does not hold. Therefore, it is needed to integrate the docking algorithm with a function that based on the current information gathered by robot sensors is able to avoid the obstacles with an optimal strategy to complete a correct docking.

The 180° field of view of the robot sensors is divided in three different areas: front, left and right. The latter two are a combination of point cloud data and distance measurements provided by proximity sensors. Thus, it is possible to know only the presence of possible objects, unlike the more complex information based on obstacle range provided by the point cloud.

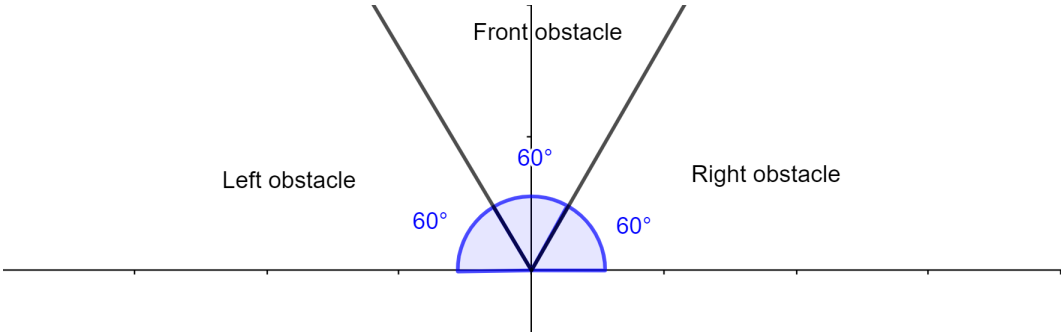

Figure 3.20. Obstacle avoidance algorithm

When the robot detects an obstacle it focuses its efforts on avoiding it, then, as soon as possible, it resumes the docking process. The main purpose is to develop an algorithm which takes in consideration the station position while avoiding objects in order to facilitate the next docking stage. Before presenting the algorithm it is

necessary to made some assumptions:

- The mobile robot has no prior knowledge of the environment, therefore it doesn't have a map of the surroundings.
- It always knows its position with respect to the map reference frame. For example, the current position can be provided by the GPS in outside environments or SLAM technology in an indoor environment.
- It knows the position of the charging station.

The knowledge of robot and station position always allows us to compute the optimal direction to reach the target. Thanks to this information when an obstacle is detected in front, it can take the best decision based on the AR tag position. In figure 3.21 are shown three possible scenarios where the red angles (θ_1, θ_2 and θ_3) are the angles between robot and station, the orange angles are the obstacle ranges detected by the ZED 2 camera and the blue angles (ϵ_1, ϵ_2 and ϵ_3) are the orientations between robot and station.

In order to be able to make the optimal direction choice the robot has to know the obstacle range related to the station reference frame. Therefore, its orientation is summed to the obstacle range obtaining the latter in the station reference system. Later, it avoids the obstacle following the direction closer to the optimal path represented by the red angle. For instance, in the case (b) in Figure 3.21 the robot avoids the obstacle at right. Concerning the left and right obstacles, the mobile robot keeps them at a minimal distance during the docking approach.

This type of solution (Fig. 3.22) is developed for a complete integration with the docking process that works thought AR tag detection system. It does not rely on a path planning or local map base approach, therefore it can be subject to some drawbacks. For example, the obstacle avoidance function considers only the most recent sensor readings. This condition, in the case of very complex situations

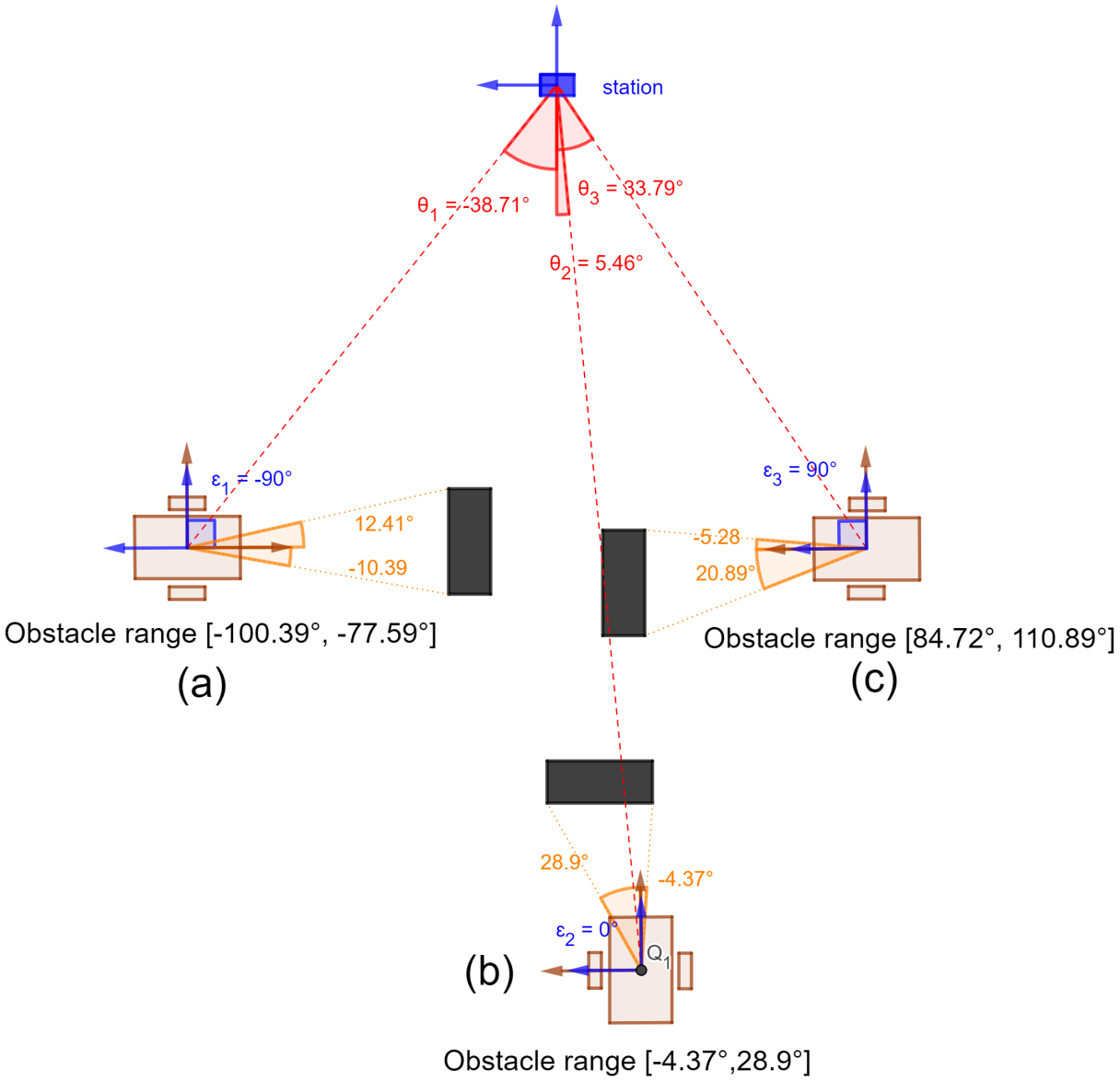

Figure 3.21. Obstacle avoidance strategy.

where there is a high presence of obstacles, could drive the robot in possible local minima points or infinitive loops.

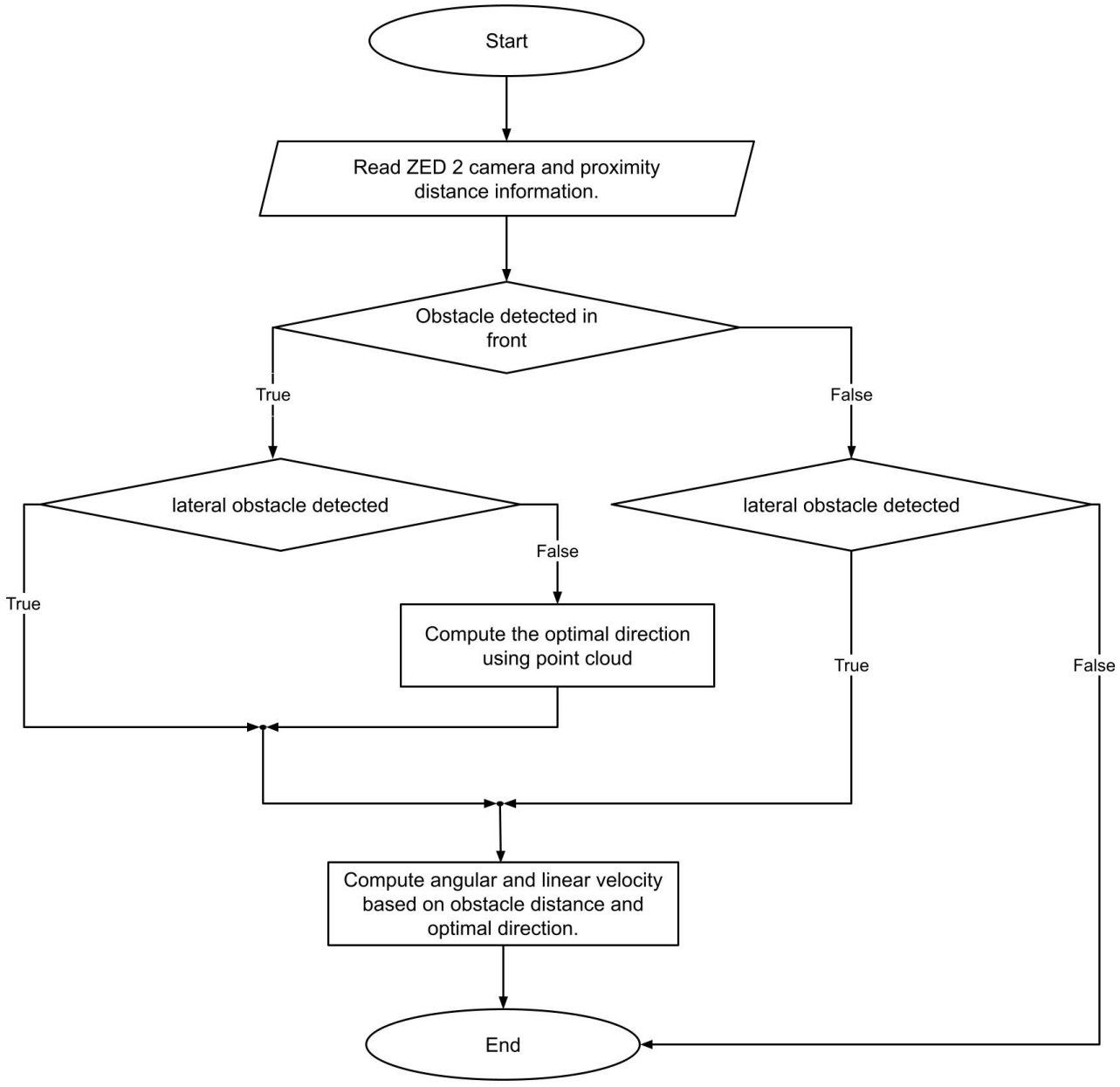

Figure 3.22. Obstacle avoidance flowchart.

Chapter 4

Simulation and Tests

4.1 Gazebo Simulations

In the previous chapter we highlighted the importance of a simulation phase in a robotic project to test the developed code and the possible robot behavior. In the present section it is described the world created in the Gazebo environment and some of the tests performed to validate the functionalities of the system.

4.1.1 Gazebo Environment

Basically the simulation world consists of three parts: the model of the robot, the model of the AR tag and an environment with possible obstacles to simulate real-world scenarios.

Robot URDF

The model of the robot is built to represent the most relevant and salient features of the real vehicle, and likewise, it is kept as simple as possible. The main element are:

- Chassis
- Standard wheels (x2)
- Coaster wheel
- Laser scan
- Depth camera

The robot model and the position of each component is shown in the Figure 4.1. Principally, to approach charging station the robot needs two types of sensors: the ZED 2 camera and proximity sensor. The former is placed in front of the robot in order to recognize the AR tag and also to detect all possible obstacles. The latter is modeled as a laser sensor which is needed in the simulation stage for two main reasons. Firstly, it allows to create a map of the environment and it can be used to provide the robot position using the Monte Carlo localization. Secondly, it works as a proximity sensor in the field of view where the depth camera cannot work.

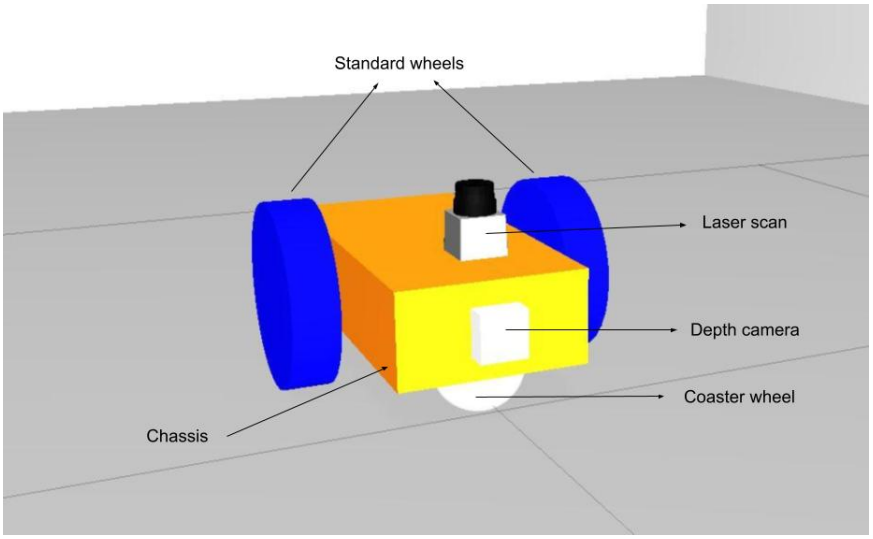

Figure 4.1. Robot model

Once the structure of the robot has been defined, three different plugins are needed to interact with it and simulate sensors:

- Differential drive plugin: allows control of the 2 standard wheels to drive the robot in the simulation environment.
- Depth Camera plugin: simulates the stereo camera sensor publishing the point cloud topic and the camera topic that is exploited by the ar_track_alvar function.
- Laser Scan plugin: simulates a 2D Lidar that provides distance information of possible obstacles.

At this point, the robot model is fully controllable and is able to simulate the sensor readings required for the docking algorithm.

AR tag

The insertion of the AR tag in the Gazebo environment required the creation of the mesh, which must be connected to the URDF file. The mesh (Fig. 4.2) is designed with the help of Blender software, a free and open source 3D creation suite, and through the texture generated by Ar_Track_Alvar. During this process is fundamental that the size of the tag must match to the size defined in the ar_track_alvar node.

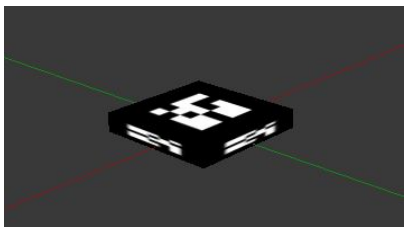

Figure 4.2. AR tag created with Blender.

Simulation world

The simulation world is designed in order to represent a possible indoor scenario, such as for example the airport terminal. The walls divide the environment in different rooms and create some areas designated to the docking process. In addition, the intensity of some light sources had to be calibrated due to the difficulty of the ar_track_alvar node to detect tags. In figure 4.3 is shown a possible example of the docking area.

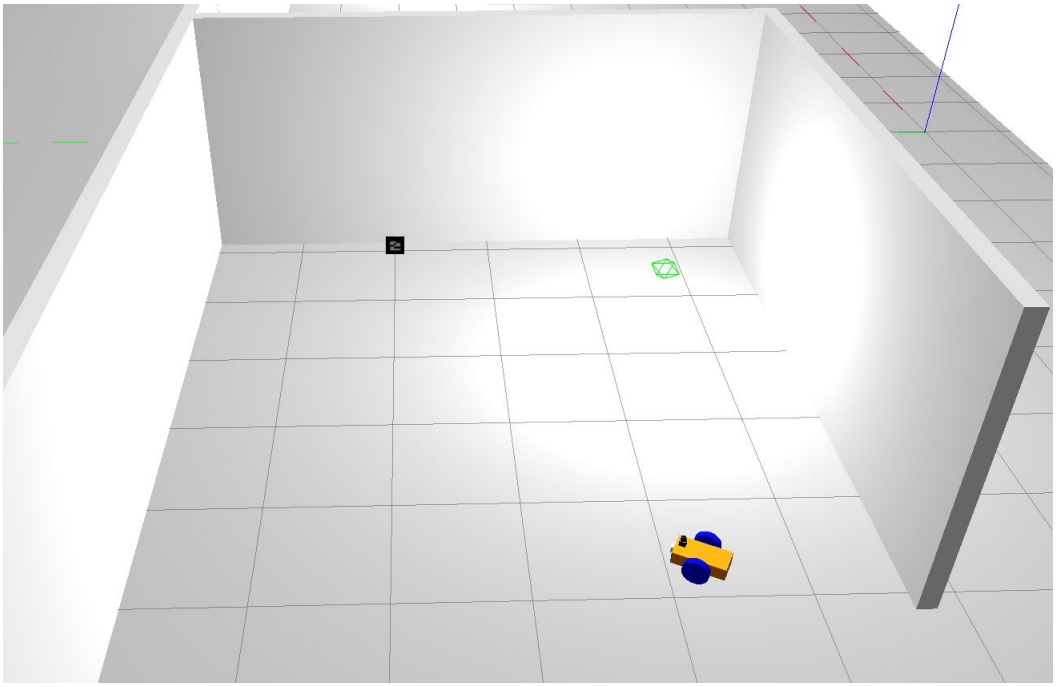

Figure 4.3. Gazebo simulation environment.

4.1.2 Simulation Tests

The test phase has been fundamental to develop a good docking algorithm and the obstacle avoidance function. In the simulation world the AR tag has a size of 0.09x0.09m which implies that it could be detected at a maximal distance of 1.5m.

When the depth camera cannot detect the AR tag, the algorithm must rely on the robot position. The odometry position provided by Gazebo is too precise and not representative of reality. For this reason, the Monte Carlo localization has been used exploiting the data provided by the laser sensor. This type of localization is one of the most adopted solutions to localize a robot in an indoor scenario and it is usually exploited in the SLAM mechanism. It requires creating a map of the environment in order to provide an accurate position estimate.

During simulations the robot was positioned in different parts of the room with and without the presence of obstacles. Then, its behaviour was studied and the algorithm was modified in order to guarantee the docking in all the situations. In Figure 4.4 is presented an example of the path followed by the robot in one of the possible scenarios.

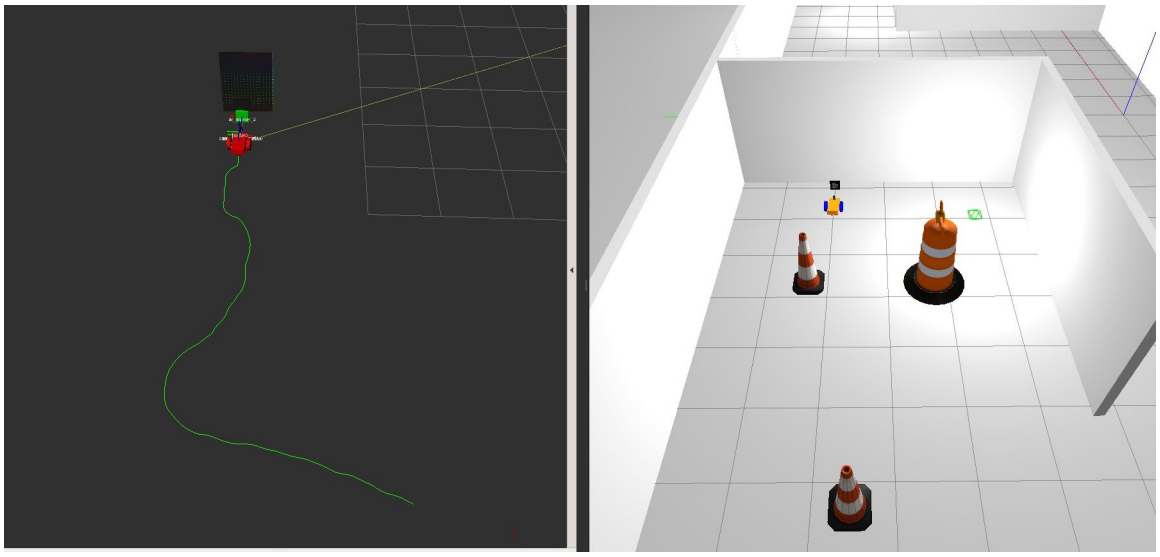

Figure 4.4. Example of docking path with obstacle avoidance, at left the RViz environment and at right the Gazebo world.

4.2 Tests in a Real Environment

The entire work of this thesis has been evaluated testing the algorithm in a real environment, that, unlike simulation, introduces many other factors due to the imperfect nature of the real world and the limitation of the hardware. However, the InnoTech robot (Fig. 4.5) was still under construction. Therefore, in collaboration with another project under development a new mobile robot was built. In particular, this prototype was used to perform some preliminary tests of the work carried out so far and allowed other researchers to evaluate a new localization system in an outdoor environment based on sensor fusion.

Figure 4.5. InnoTech robot body.

This section will describe the new robot prototype and the reasons at the base of some design choices. In the end will be presented the performance results obtained with this robot hardware and the developed algorithm.

4.2.1 Robot assembly

The new designed robot has to satisfy the sensor requirements for the docking tasks and localization goals. In particular, the ZED 2 camera sensor must be mounted at a minimum height of 0.7 meters in order to detect the obstacles and filter out the ground. Given this request, the design of the robot chassis has been driven by the need for a specific structure that allow the stereo camera to be mounted at a fixed height (Fig. 4.6).

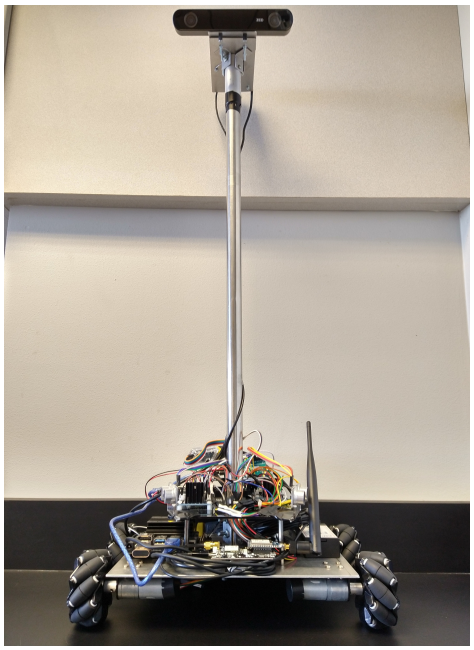

Figure 4.6. Robot.

The robot prototype is a differential drive vehicle endowed with 4 metal gear motors, hall encoders and omnidirectional wheels which guarantee a greater stability compared to 3 wheels mobile robots. It is divided in three levels, in the first one the power supply batteries and the main boards have been mounted.

- Power bank for the NVIDIA Jetson Nano supply.
- 2 batteries for motors and the Arduino micro-controllers.

- NVIDIA Jetson Nano board.
- GNSS board.

Instead, at the second level we found micro-controllers, drivers and remaining sensors:

- Arduino Mega2560 board.
- Arduino Uno board.
- IMU sensor.
- 2 motor drivers.
- 2 ultrasonic sensors with a receipt angle on 30° located in front of the robot at ±75°.

The Arduino Mega2560 reads the encoder data and communicates with the NVIDIA Jetson Nano from which it receives the relative commands for the motor drivers. Instead, the Arduino Uno board is in charge to read and send ultrasonic sensor measurements to the main board (Fig. 4.7).

At the last level it is installed the ZED 2 camera in order to satisfy the specific requirement already described and the GNSS antenna for improving the localization performances.

4.2.2 Results

The last section of this work is focused on testing the robot behaviour driven by the developed algorithm in order to verify if it could be a possible implementation for a docking applications.

The tests have been executed in an indoor environment with the tag that identifies the station hung on the wall. The AR tag is 0.18x0.18 meters and it can be

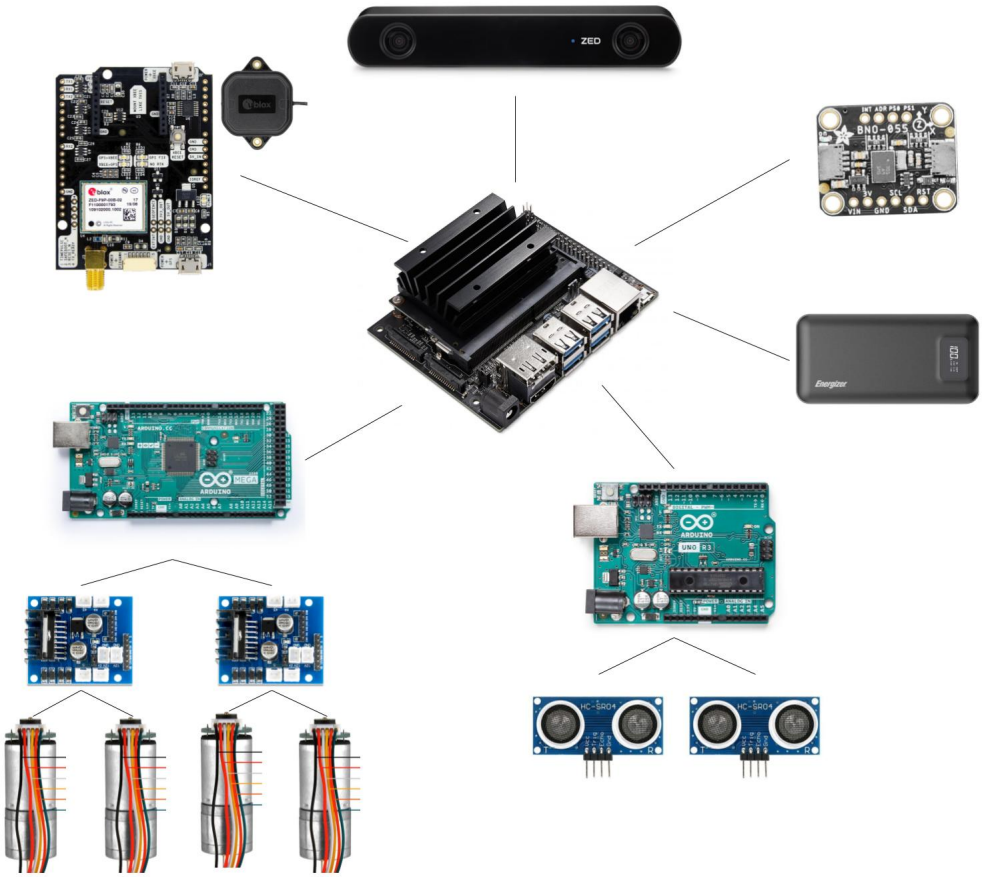

Figure 4.7. Robot hardware schema.

detected at a distance of 1.5 meters (Fig. 4.8). It is important to point out that, the accuracy and the maximum distance at which the tag is detected strongly depend on its size but also on the inclination and height at which the ZED 2 camera is mounted on the robot. Moreover, the performance of docking operations are affected by external factors not correlated with the developed code. In particular, the robot is a differential drive prototype which has some limitations. Firstly, the 6V motors are not powerful enough to permit precise movements at slow speeds. Secondly, as mentioned above, the docking process needs the robot position when the tag is not detected. However, this prototype is not equipped with a SLAM

system, therefore, a combination of IMU sensor and of wheel encoders provide the indoor position. Unfortunately, these two sensors provide position with low accuracy for medium and long navigation path. This limitation leads to executing all the tests by placing the robot at a maximum distance of 4 meters,.

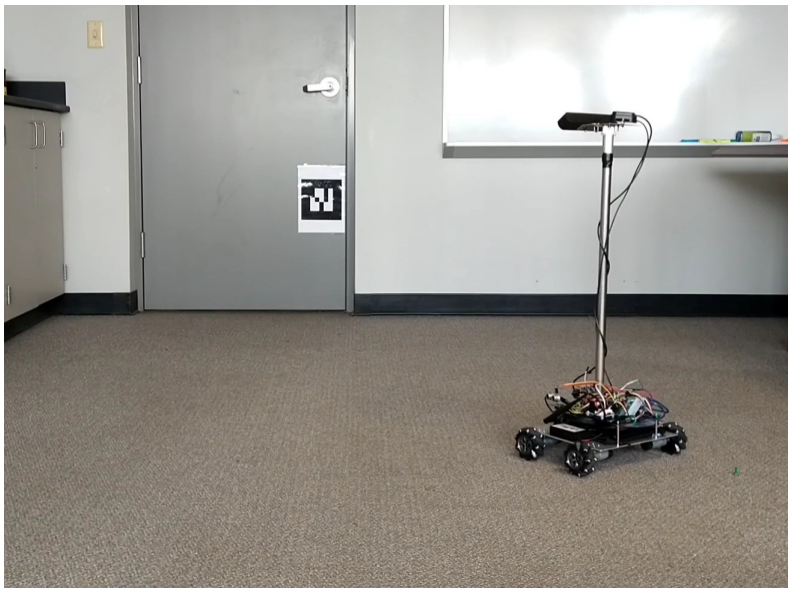

Figure 4.8. Test environment.

Ideally, a docking is considered correct when it allows the battery charging process to begin. Unlike other projects, in this case, the charging station and the correlated robot docking mechanism have not been designed. Therefore, three measures have been taken in consideration to evaluate the performance of the algorithm: the distance at which the robot stops in front of the station, the orientation and the angle with respect to the station perpendicular position.

At first, we only focused on the docking process without obstacles in the surrounding area in order to show its strength and its weaknesses. As described in the section 3.4.2, it is divided in two phases where the goal of the first is to drive the robot to the station perpendicular line as soon as possible. According to the station distance, it can drive the robot to the target position with an 80° or 45°

angle. This choice creates two different regions in front of the station, one where the docking will succeed and another in which the docking will probably fail (respectively green and red regions in Figure 4.9). This is due to the fact that, if the mobile robot is in the red area, it cannot reach the perpendicular position to the station before the end of the docking. In the test session it has been decided to stop the robot when the distance from the station is lower than 0.5 meters and to use the 45° angle approach for the first phase within 3 meters. The robot was placed in four different initial positions (Fig. 4.9):

- TestA: it starts in the red zone. It means that probably it will not be able to satisfy the minimum requirements for a successful docking. In that case, the retry docking phase drives the robot in front of the station at a distance of 2 meters (Retry docking point).
- TestB and TestC: they are very similar because the distance from the station is the same but the orientation of the robot is different. It was tested if the performance remains comparable even if the robot starts with different orientation in a position that is relatively close to the station.
- TestD: this test wants to verify that, even if the robot starts far from the station, if its localization is accurate the docking remains precise.

In Figure 4.10 are reported the mean value of the angle and the orientation for each test type. In order to understand these graphs, it is important to give some more information about the docking procedure. The algorithm dedicated to the retry docking is divided into two parts. If the angle value is acceptable, it adjusts only the robot orientation rotating on itself. Instead, in the other case, because of the nature of the differential drive model, the robot cannot change its angle with a linear velocity equal to zero. Therefore, it is driven to a location at 2 meter distance in front of the station which is optimal to start a new docking attemp.

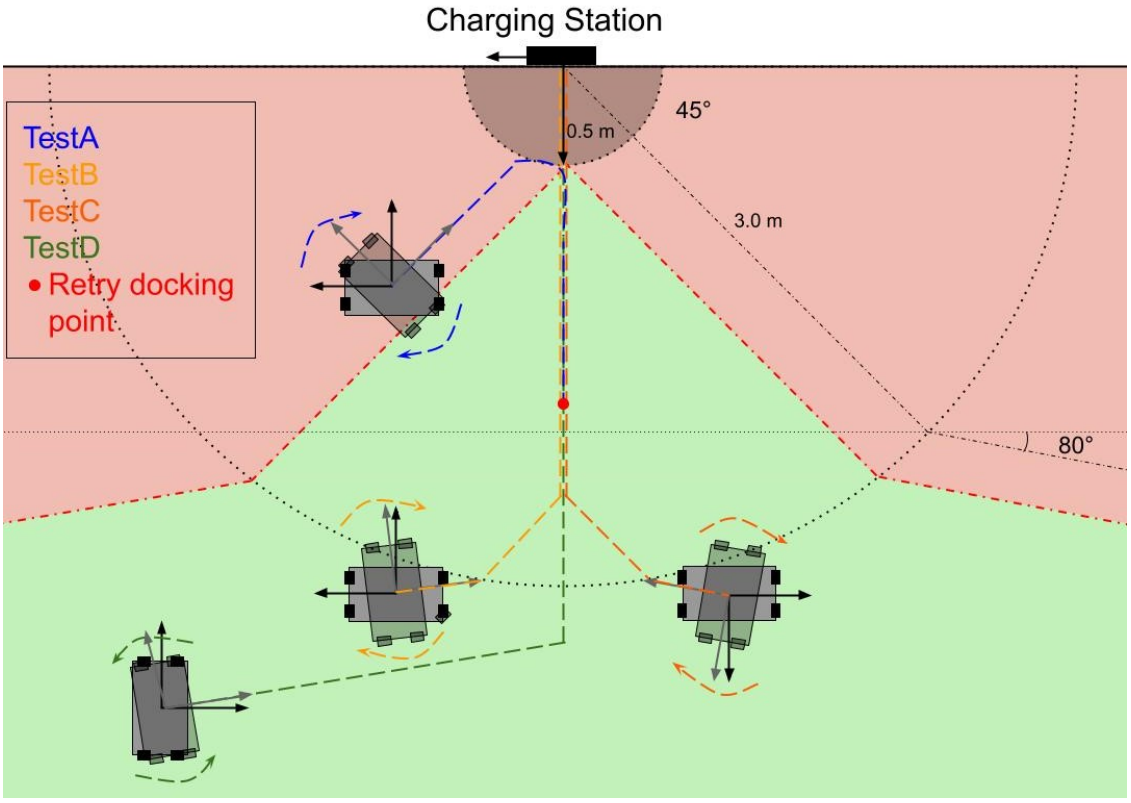

Figure 4.9. Docking tests.

During these tests, all the angle values have been considered valid, then, the robot can adjust only the orientation. This choice allowed us to show how good the algorithm is from the angle point of view and how much the precision decreases when the robot starts in the red zone. All dockings with a maximum orientation error of 5° have been considered acceptable, otherwise the robot is rotated on itself until the requirements are met.

Analyzing the test data we may notice that TestA angle is in line with the expectations, instead, the orientation results are worse than other tests despite the retry docking process. It can be explained taking in consideration that the more the sensor is perpendicular to the AR tag the higher ar_track_alvar precision is. The starting position of the robot never allows it to be perfectly in front of the

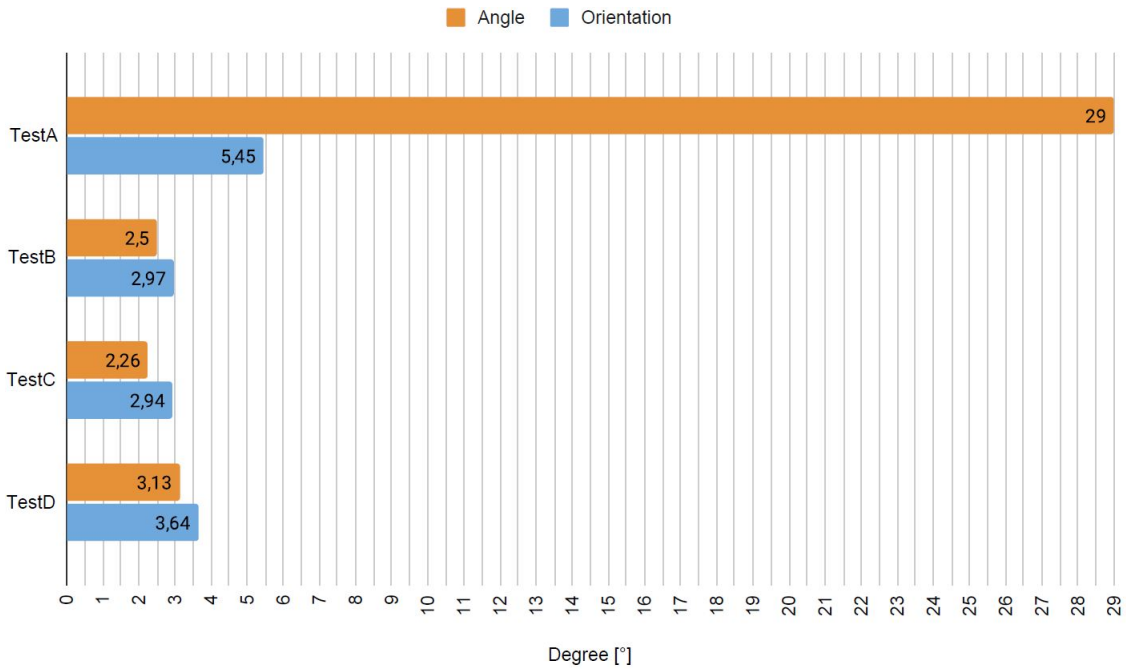

Figure 4.10. Angle and Orientation test results.

marker, therefore the orientation and the distance data cannot be very precise. Given the algorithm nature, an average angle of 29° is the maximal accuracy that we can obtain maintaining an angle of 45° during the first phase. In the event that TestA angles are considered not acceptable, the robot should be driven in the retry docking point in order to complete a successful docking.

The TestB and TestC achieved the same results with few variations. This shows that although the robot can start with different orientations to the AR tag, it is able to carry out the docking phase with similar performances.

The TestD outcomes are slightly worse than the TestB and TestC due to a longer docking process based on the robot's position, which, not being very accurate, drives the robot with a bigger angle error before detecting the AR tag.

Previous results have shown that we can reach an average precision of about 3° for both the angle and the orientation when the robot starts in the green area. Otherwise, the docking may fail and the robot can be driven to an optimal position to retry the process. This level of accuracy cannot be reached relying only on the robot position provided by the IMU sensor and Encoders. This is evident analyzing the comparison between the TestD results obtained with and without the help of the AR tag (Fig. 4.11). The error in the angle is about 10 times greater than the docking angle reached using the AR Track Alvar function.

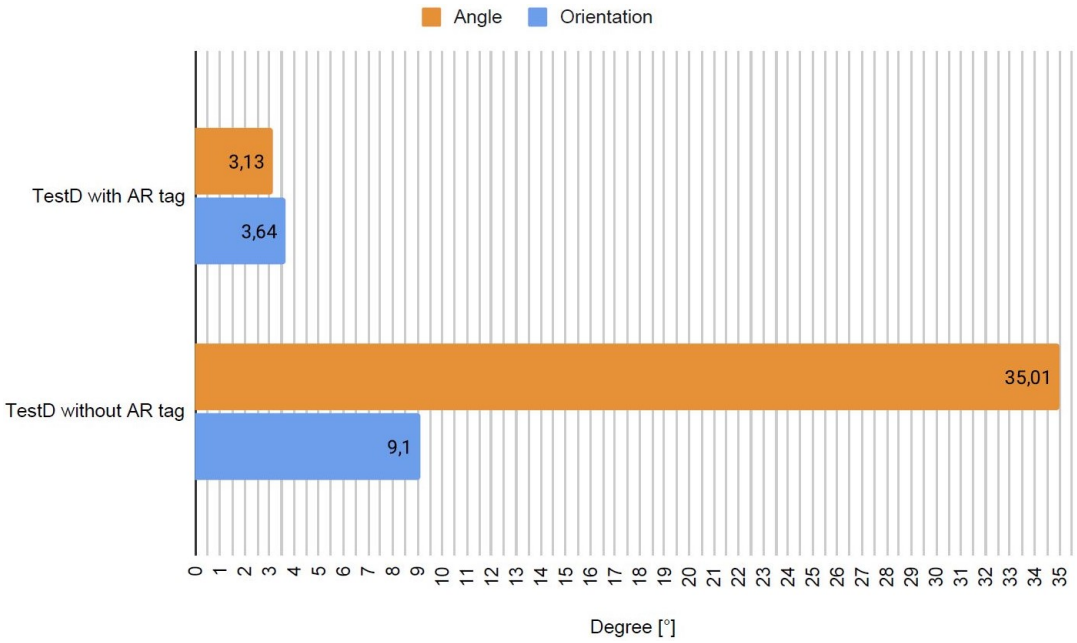

Figure 4.11. Comparison between Angle and Orientation results with and without the AR tag.

The last part of this chapter is dedicated to the evaluation of the obstacle avoidance function during the docking process. The minimal distance detected by the ZED 2 camera is 0.3 meters, therefore, for safety reasons, the obstacles were maintained at a distance of 0.4 meters. Moreover, the robot was equipped with 2 ultrasonic sensors to detect lateral objects.

Three most relevant scenarios have been taken into consideration in order to test the main functionalities of the algorithm.

a. Obstacles during the first phase.

b. Obstacles during the second phase and far enough away to get through them.

c. Obstacles during the second phase but too close to get through them.

In all the previous situations the robot was able to complete a successful docking avoiding collision with objects. In Figure 4.12 are illustrated the paths that the robot followed driven by the obstacle avoidance algorithm.

These obstacle configurations has been chosen to verify if the algorithm was able to select the most promising direction to avoid collisions with obstacles. In the case b, for example, the optimal direction is to get through them, instead, in case c, the obstacles are too close and they are considered as a single entity. Other tests have drawn attention to the fact that the presence of possible obstacles in an area of 1.5 meters around the station can affect in a very negative way the docking process, because of the difficulties to avoid objects and complete a correct docking in a confined space. At the same time, under certain conditions, this project can be a possible solution for the autonomous recharging issue.

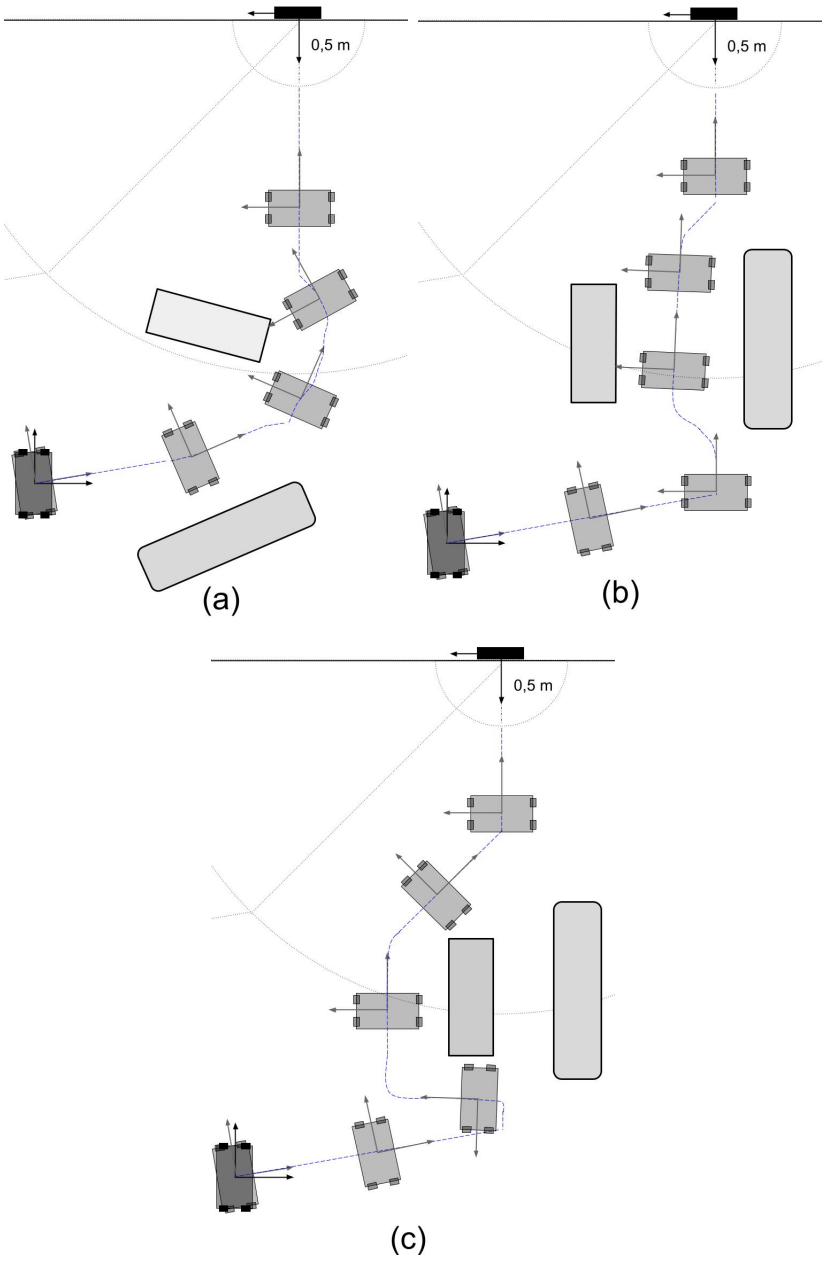

Figure 4.12. Obstacle avoidance tests. At top left the test (a), at top right test (b) and at bottom the test (c).

Chapter 5

Conclusion

Coming to the conclusion it is possible to sum up the work done so far. In this thesis an overview on the docking projects has been performed as well as a description of the most used obstacle avoidance algorithms. The goal of providing a docking system endowed with an obstacle avoidance function to the InnoTech project has been achieved. Currently, the robot can perform different tasks to complete the docking phases. At first, it can detect the charging station through the AR tag and the ar_track_alvar node which extracts the information about its position. If the tag is not detected, the robot knows the charging station position and orientation and it can compute the same information with a lower accuracy. Then, with the extracted data and driven by the docking algorithm, it can complete the docking phase.

Tests performed in a real environment have demonstrated that it is possible to complete the docking with an average precision of approximately 3° for both the angle and the orientation. In the case of eventual obstructions during operations, the robot can firstly detect the obstacles creating a point cloud of the environment and secondly it can compute the optimal direction to avoid them in order to facilitate the subsequent docking phase. The results of the tests carried out can

be considered satisfactory taking into account the limitations of the built robot. Furthermore, with a more performing robot, such as the one being developed by the InnoTech System, it would be possible to achieve an higher level of accuracy thanks to more powerful motors and better controllers.

However, this is only the initial part of the autonomous charging project and other improvements are needed. One of the weaknesses of the developed system is the obstacle avoidance function that considers only the most recent sensor readings. This condition can drive the robot in possible local minima points or infinitive loop making docking impossible. In addition, the detection performance of the ZED 2 camera sensor is not reliable in certain lighting conditions. Storing the path followed by the vehicle could be a possible solution for the local minima points and infinitive loops since it allows to understand if the robot has already made that choice in that specific location [49]. Moreover, a combination of LiDAR and ZED 2 camera for the obstacle detection task needs to be employed in order to achieve a reliable obstacle avoidance system.

Regarding the future of the autonomous recharging project, the developed docking system requires the design of the recharging station with the related charging mechanism and the integration with an energy awareness function which must decide when the robot needs to replenish his batteries.

Bibliography

[1] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. «A review of mobile robots: Concepts, methods, theoretical framework, and applications». In: *International Journal of Advanced Robotic Systems* 16.2 (2019). DOI: 10.1177/1729881419839596.

[2] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. USA: Bradford Company, 2004. ISBN: 026219502X.

[3] Mary B. Alatise and Gerhard P. Hancke. «A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods». In: *IEEE Access* 8 (2020), pp. 39830–39846. DOI: 10.1109/ACCESS.2020.2975643.

[4] Balajee Kannan et al. «The Autonomous Recharging Problem: Formulation and a market-based solution». In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 3503–3510. DOI: 10.1109/ICRA.2013.6631067.

[5] M. Ben-Ari and F. Mondada. *Elements of Robotics*. Cham, Switzerland: SpringerOpen, 2018.

[6] *IR sensor image*. URL: https://solarduino.com/infrared-ir-sensor-module-with-arduino/.

[7] *Ambarella LiDAR and stereovision*. URL: https://www.ambarella.com/blog/a-closer-look-at-lidar-and-stereovision/.

[8] R.C. Arkin and R.R. Murphy. «Autonomous navigation in a manufacturing environment». In: *IEEE Transactions on Robotics and Automation* 6.4 (1990), pp. 445–454. DOI: 10.1109/70.59355.

[9] S. Yuta and Y. Hada. «Long term activity of the autonomous robot-proposal of a bench-mark problem for the autonomy». In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*. Vol. 3. 1998, 1871–1878 vol.3. DOI: 10.1109/IROS.1998.724869.

[10] A. Zelinsky S. Oh and K. Taylor. «Autonomous Battery Recharging for Indoor Mobile Robots». In: 2000.

[11] M.C. Silverman et al. «Staying alive: a docking station for autonomous robot recharging». In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 1. 2002, 1050–1055 vol.1. DOI: 10.1109/ROBOT.2002.1013494.

[12] S. Yoona K. W. Leea H. S. Ryub C. K. Woob K. H. Kima H. D. Choia and Y. K. Kwaka. «Development of Docking System for Mobile Robots Using Cheap Infrared Sensors». In: *1st International Conference on Sensing Technology, Palmerston Noth, New Zealand. November 21-23,* 2005.

[13] D. V. Kaushok Kariappa K. Varun Raj Kiran Patil and Amit Madhav Jakati. «A Beacon-based Docking System for an Autonomous Mobile Robot». In: *13th National Conference on Mechanisms and Machines (NaCoMM07), IISc, Bangalore, India. December 12-13,* 2007.

[14] Nathalie Mitton Roberto Quilez Adriaan Zeeman and Julien Vandaele. «Docking autonomous robots in passive docks with Infrared sensors and QR codes». In: *International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCOM)*. Vancouver, Canada, June 2015. URL: https://hal.inria.fr/hal-01147332.

[15] Florin-Dan Secuianu and Ciprian Lupu. «Implementation of a home appliance mobile platform based on computer vision: self-charging and mapping». In: *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*. 2018, pp. 464–468. DOI: 10.1109/ICSTCC.2018.8540685.

[16] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[17] M. Nosrati, Ronak Karimi, and Hojat Allah Hasanvand. «Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches - TI Journals». In: *World Applied Programming* (2012).

[18] J. Minguez, F. Lamiraux, and J. Laumond. «Motion Planni 35 . Motion Planning and Obstacle Avoidance». In: 2011.

[19] R.B. Tilove. «Local obstacle avoidance for mobile robots based on the method of artificial potentials». In: *Proceedings., IEEE International Conference on Robotics and Automation*. 1990, 566–571 vol.1. DOI: 10.1109/ROBOT.1990.126041.

[20] J. Borenstein and Y. Koren. «The vector field histogram-fast obstacle avoidance for mobile robots». In: *IEEE Transactions on Robotics and Automation* 7.3 (1991), pp. 278–288. DOI: 10.1109/70.88137.

[21] J. Minguez. «The obstacle-restriction method for robot obstacle avoidance in difficult environments». In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 2284–2290. DOI: 10.1109/IROS.2005.1545546.

[22] D. Fox, W. Burgard, and S. Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[23] *ROS introduction*. URL: http://wiki.ros.org/ROS/Introduction.

[24] Leon Jung Darby Lim Yoonseok Pyo Hancheol Cho. *ROS Robot Programming (English)*. ROBOTIS, Dec. 2017. ISBN: 9791196230715. URL: %5Curl%7Bhttp://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51%7D.

[25] *ROS Concepts*. URL: http://wiki.ros.org/ROS/Concepts.

[26] *Rviz*. URL: http://wiki.ros.org/rviz.

[27] *URDF in Gazebo*. URL: http://gazebosim.org/tutorials/?tut=ros_urdf.

[28] *Gazebo plugins*. URL: http://gazebosim.org/tutorials?tut=ros_gzplugins.

[29] *Jetson Nano eLinux wiki*. URL: https://elinux.org/Jetson_Nano.

[30] *Jetson Nano Developer kit*. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[31] *STEREOLABS ZED2*. URL: https://www.stereolabs.com/zed-2/.

[32] *ZED2 camera datasheet*. URL: https://www.stereolabs.com/assets/datasheets/zed2-camera-datasheet.pdf.

[33] *ar_track_alvar, attachment:artags.png*. URL: http://wiki.ros.org/ar_track_alvar.

[34] *Alvar*. URL: http://virtual.vtt.fi/virtual/proj2/multimedia/index.html.

[35] *Alvar user manual*. URL: http://virtual.vtt.fi/virtual/proj2/multimedia/media/ALVAR_v2_User_Manual_v1.1.pdf.

[36] *ar_track_alvar*. URL: http://wiki.ros.org/ar_track_alvar.

[37] *ar_track_alvar_msgs/AlvarMarker Message*. URL: http://docs.ros.org/en/kinetic/api/ar_track_alvar_msgs/html/msg/AlvarMarker.html.

[38] *tf/tfMessage Message*. URL: https://docs.ros.org/en/kinetic/api/tf/html/msg/tfMessage.html.

[39] *geometry_msgs/PoseStamped Message*. URL: http://docs.ros.org/en/kinetic/api/geometry_msgs/html/msg/PoseStamped.html.

[40] *geometry_msgs/TransformStamped Message*. URL: http://docs.ros.org/en/kinetic/api/tf/html/msg/tfMessage.html.

[41] *tf package and frames*. URL: http://wiki.ros.org/tf.

[42] Larry Matthies. «Obstacle Detection». In: *Computer Vision: A Reference Guide*. Ed. by Katsushi Ikeuchi. Boston, MA: Springer US, 2014, pp. 543–549. ISBN: 978-0-387-31439-6. DOI: 10.1007/978-0-387-31439-6_52. URL: https://doi.org/10.1007/978-0-387-31439-6_52.

[43] Anca Discant et al. «Sensors for Obstacle Detection - A Survey». In: *2007 30th International Spring Seminar on Electronics Technology (ISSE)*. 2007, pp. 100–105. DOI: 10.1109/ISSE.2007.4432828.

[44] *Point Cloud Library (PCL)*. URL: https://pointclouds.org/.

[45] *PointCloud type*. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/basic_structures.html#basic-structures.

[46] *PointCloud2 type*. URL: http://docs.ros.org/en/api/sensor_msgs/html/msg/PointCloud2.html.

[47] *RATB-Map ROS*. URL: http://wiki.ros.org/rtabmap_ros#rtabmap_ros.2Fobstacles_detection.

[48] Radu Bogdan Rusu. «Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments». PhD thesis. Computer Science department, Technische Universitaet Muenchen, Germany, Oct. 2009.

[49] Hoc Thai Nguyen and Hai Xuan Le. *Path planning and Obstacle avoidance approaches for Mobile robot*. 2016. arXiv: 1609.01935 [cs.RO].