

POLITECNICO DI TORINO

Master of Science in Mechatronics Engineering



Master Thesis

Autonomous charging system for a three wheel robot

Thesis Superadvisors

Prof. Giuseppe Calafiore

Prof.ssa Marina Mondin

Student

Luca Romanello

Academic Year 2019/2020

Acknowledgements

As first, I would like to thank Politecnico di Torino and Cal State University for giving me the opportunity to live this extraordinary experience and professors Calafiore and Mondin for dedicating their time to my work.

I would also like to thank Eliza for her infinite generosity and my friend Justin for his robot construction help. They, together with my colleagues Andrea e Matteo, greatly improved my experience with their sincerity and stimulations. Here I would like to make a summary of these 5 years.

I want to send an hug to all the wonderful people I have met in these years, I have been fed by your smiles and strength and this helps me to go ahead and give more and more. In particular, I want to give thank to Salvatore, Giacomo, Alessandro and the mechatronics guys for the entire days spent studying together and Giuseppe, Davide, Luca and all the guys I have met in college in this period. We spent so much time together, from nice to sad moments and I won't forget it.

By the way, another hug is reserved to the Frati Minori Piemonte for their endless sense of altruism and hospitality and to the Gioventù Francescana for the countless quantity of kisses and hugs. They have made my smile larger than ever, love you guys.

Then, a greeting to my long time friends, Giorgio, Andrea, Giacomo, Gianluca, Marco, Gory, Picci, Sarci, Biagino and who else who have been looking forward to go outside with me every time I showed up in Nardò or just send me a message. You are all nice people and I carry you in my hearth.

Last but not least, my family for their regardless love. My grandmother for her advices on cooking and living, my father to teach me to be more carefree and focused on what counts, my mother where I always get the example of everything and my brother, Pierpaolo, to allow me to be one step over him, always.

And my fiancée, Alessia to be from my side no matter what and to love me every day more and more.

This work is dedicated to my grandfather, Michele. Miss you nonno.

Luca

Abstract

This work is about the design, implementation and testing of an autonomous charging system for a three wheel drive service robot. The purpose is to find a way to autonomously drive the robot from a position close to the charging station towards the station itself. This was one subsection of an autonomous driving project from California State University, Los Angeles. At the beginning a research about the state of the art of this technology has been presented together with references to relative works and papers. The introduction also includes a comparison among the different charging systems employed for market and custom service robots and an overview of the different docking algorithm strategies. Then, in order to achieve a successful docking operation, it has been necessary to design the docking algorithm inspired by the state of the art and compatible for both the robot and the charging station. The algorithm takes advantage of a vision system for detecting and tracking an AR tag. The odometry information is also extracted to have a feedback of the robot pose. The development system used to test the algorithm has been the ROS/Linux framework. The first part of the work delves on the simulation stage conducted on Gazebo environment. The functionalities of the camera and the robot differential drive have been implemented in the simulation as part of the relevant elements of robot model. The camera topic is visualized on Rviz and the AR tag is tracked by the mean of a code reading ROS library. This library is capable to mark the tag and even to extract the pose from it. Then, the algorithm has been translated in C++ functions that communicate on ROS. The pose data coming from the tag are manipulated in order to extract distance, angle and orientation between the camera (robot) and the station (tag). Based on these dimensions, the algorithm provides command signals to the robot to reach the station. The second part is focused on the building of a custom robot. The robot is equipped with a stereocamera (ZED by Stereolabs) and encoder sensors to get odometry information (i.e. wheel velocity). The camera is connected to the NVIDIA Jetson Nano embedded board, that represents the core of the robot. It allows you to perform the algorithm ROS functions, computing the desired speed. The robot implements an Arduino platform that provides the encoder sensors data as output to the Jetson. At the end, a feedback control system takes as input the desired speed and the actual speed and calculate the control input voltage for the motor drivers.

Contents

List of Figures	iii
List of Tables	vi
1 Introduction	1
1.1 Autonomous Service Robots	1
1.2 Overview of the project	3
1.2.1 Mechanical design	4
1.2.2 User Interface	5
1.2.3 SLAM Depth-based	6
1.2.4 Task assignment optimization	7
1.2.5 GPS localization	8
1.2.6 Autonomous charging system	9
2 State of the art	11
2.1 Charging systems	13
2.1.1 Docking: Mechanical coupling	13
2.1.2 Wireless	16
2.1.3 Snake-like solution	19
2.1.4 Comparison	21
2.2 Docking strategies	23
2.2.1 Market sensors	23
2.3 Existing systems	29
3 Project implementation	35
3.1 Station detection	36
3.1.1 Stereoscopy	37
3.1.2 Image processing	39

3.1.3	AR tag	44
3.2	Tracking algorithm	45
3.3	Software tools	52
3.3.1	ROS framework	52
3.3.2	Gazebo environment	55
3.3.3	Arduino world	58
4	Simulation stage	61
4.1	Robot on Gazebo	62
4.1.1	Robot body	63
4.1.2	Differential drive plugin	64
4.1.3	Camera plugin	64
4.1.4	Environment	65
4.2	Software development	68
4.2.1	Extraction of parameters	68
4.2.2	Docking function	73
4.3	Testing	82
5	Custom Robot	85
5.1	Mechanical design	87
5.2	Hardware set up	88
5.2.1	Jetson Nano	88
5.2.2	Arduino UNO and motor driver	89
5.2.3	Encoder sensors	91
5.2.4	Set up and communication system	91
5.3	Velocity control system	96
5.4	Software compatibility	100
5.4.1	Sensor data acquisition	100
5.4.2	Odometry	102
5.4.3	Motors input	104
5.5	Test execution	106
6	Conclusion and Final results	109
6.1	Improvements	112

List of Figures

1.1	Differential drive configuration	4
1.2	Airport concept app	5
1.3	ZED 1 stereocamera by Stereolabs	7
2.1	Fellows Robot: Example of mechanical coupling between station and robot	14
2.2	Savioke Relay during the recharge process on the station	15
2.3	Wireless charger compatible with Apple devices	16
2.4	Graph comparing the different charging solutions by the driving range per hour of charge.	18
2.5	Witricity: an MIT implementation of magnetic resonance wireless charging	19
2.6	Staübli connector: a snake-like charging system	21
2.7	An example of infrared sensor - SHARP GP2Y0A02	23
2.8	An example of LiDAR sensor	24
2.9	Raspberry Pi 4 Camera, an example of robot implementation camera . . .	25
2.10	RFID detection system	26
2.11	U-Blox ZED-F9P GPS system	27
2.12	Station and robot design of University of Detroit implementation	29
2.13	FIT IoT Lab robot	30
2.14	Docking strategy of the FIT IoT Lab robot	31
2.15	The custom robot fruit of the Conference of Mechanisms and Machines in Bangalore	32
2.16	Bangalore robot docking station	33
3.1	Cal State Electrical department robot	35
3.2	ZED 2 stereocamera by Stereolabs	36
3.3	Triangulation problem explanation image	38
3.4	Epipolar geometry	40
3.5	Model used for the image rectification example	41

3.6	Triangulation problem explanation image	42
3.7	An example of Augmented reality (AR) tag	44
3.8	An hypothetical initial condition of docking operation	45
3.9	Parameters description of the first phase docking strategy	46
3.10	The flow chart explaining the docking algorithm	50
3.11	ROS logo	53
3.12	Gazebo logo	56
3.13	Arduino logo	59
4.1	Initial condition	62
4.2	Body of the robot on Gazebo	63
4.3	Lemon Tree House: model of the Los Angeles author apartment	66
4.4	Gazebo Building Editor main interface	67
4.5	Rviz interface showing the camera visualization and the robot and tag on the map with the relative reference frames	69
4.6	<i>ar_pose_marker</i> : transformation between reference frames of camera and tag (top), <i>tf2</i> : transformation between map reference frame and robot one (bottom).	70
4.7	Definition of <i>ar_pose_marker</i> message	71
4.8	First phase of the docking operation: on the left the orientation is lower than ϕ , while is higher on the right.	74
4.9	Twist message definition	75
4.10	Robot directions of movement	75
4.11	The graphs representing the behaviors of the linear and angular speeds with respect the distance between robot and station.	77
4.12	DistanceAngle message definition	78
4.13	First phase angle signs and definition of the areas distinguishing first and second phase of the docking operation.	79
4.14	Tolerance on the orientation. If the orientation overcomes this value, the algorithm takes action.	81
4.15	Joystick function on the simulation test	82
4.16	Caption taken during the docking operation: the robot is approaching the tag on Gazebo on the background, while the message of distance, angle, orientation one the left and of the speeds on the right are publishing. . . .	83
5.1	Differential drive robot spatial parameters	87
5.2	Jetson Nano embedded platform by NVIDIA	88

5.3	On the left the L298N driver board, on the right Arduino UNO platform . . .	90
5.4	H bridge circuit scheme	90
5.5	On the left the LM393 encoder sensor, on the right an image explaining the encoder system	91
5.6	Scheme showing the connection between Arduino, motor driver, motors and batteries	92
5.7	Communication set up between the various elements of the system	93
5.8	The front and the top view of the custom robot	95
5.9	Simple scheme of a feedback control system	96
5.10	The front and the top view of the custom robot	99
5.11	Vector3 message definition	101
5.12	Odometry message definition	102
5.13	Rviz window showing the ZED camera visualization, the track marker and the reference frames.	106
5.14	Arduino node launched is showed on the terminal above, while the terminal on the bottom is printing all the four functions messages.	107
6.1	Trace showing the path the robot made during its docking operation on Rviz.	109
6.2	Prompt printing all the four main functions messages.	110

List of Tables

2.1	Pros, Cons and Implementation method of the docking solutions	20
2.2	A comparison between the different wireless technologies on the advantages, disadvantages and applications	22
3.1	Specification of ZED 2 stereocamera	36
5.1	Specifications of Jetson Nano board	89
5.2	Bill of material of the custom robot	94

Chapter 1

Introduction

1.1 Autonomous Service Robots

Robotics is the science and practice of designing, manufacturing and applying robots. The International Organization for Standardization, with its *ISO 8373:2012* standard, defines the robots and robotics vocabulary to merge the concepts of this complex field. The definition of the robot contained in the standard is:

"Actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks"[1]

Robots can be used in different contexts and they are classified as :

- Industrial robots
- Humanoid & biomimetic robots
- Service robots
- Exploration robots

This project deals with Service robots. A service robot is considered as :

"A robot that performs useful tasks for humans or equipment excluding industrial automation applications"[2]

Service robots move and act essentially in terrestrial, underwater or aerial environments. Their structure and complexity strongly depends on the environment they work into and on the purpose they have been built for.

According to ISO 8373 robots have "degree of autonomy", which is the ability to perform intended tasks based on current state and sensing, without human intervention[2]. It requires intelligence, sensors, actuators and energy source. For service robots the degree of autonomy goes from partial autonomy (i.e. the need of a human-robot interaction) to full autonomy (when there is no human intervention).

Automation in the industry has evolved from the use of simple hydraulic and pneumatic systems to today's modern robots. Since its inception, industrial automation has made great advances among activities that were previously carried out manually. A manufacturing organization that uses the latest technologies to fully automate its processes typically sees improved efficiency, production of high-quality products and reduced labor and production costs [3].

Automated machine is not a new technology. Suffice it to think to the Ford car production assembly line in 1913, which is considered a pioneer of types of automation in this field. The production increased as the profits. The current robots are characterized by high computing capabilities, improving degree of freedom and different kind of perception devices.

In this introductory chapter an overview of the project will be presented, briefly describing the main features of each subsystem that characterizes the whole project. Such subsystems concern the common problems of design a robot. Some of them are already completed from other graduate researchers of California State University, while others are the focus of this current commitments.

1.2 Overview of the project

In recent years the investments made in research on autonomous robots and more generally on autonomous guidance systems have become increasingly high. In addition, companies producing cars with certain levels of automation and aiming at fully automated vehicles are rapidly increasing their value and power on the market, directing it towards higher forms of automation.

Major automaker companies, technology giants and specialist start-ups have invested more than USD 50 billion over the past five years, in order to develop autonomous vehicle (AV) technology and at the same time, public authorities see that AVs offer huge potential economic and social benefits. By the way, Tesla has recently overtaken Toyota to become the world's most valuable car manufacturer in the world. Furthermore, Waymo, the Alphabet's self-driving company in Mountain View, raised more than USD 2.25 billion in its first external funding raise on May 2020, despite the coronavirus crisis. Such a hit on the market has its root in the venture capitalists, that have begun to feel more confident about the future of electric and autonomous vehicles.

All these elements attracted the Department of Electrical and Computer Engineering of California State University of Los Angeles and the startup *Innotech Sys* company to invest and to gain competences on this field finding a fertile ground of investors especially in the state of California. In this regard, last year the startup joined the incubator of the San Diego Airport that helped the company to fund its developments with the purpose to build an autonomous robot. This robot should be designed to interact with the users of Airport in a continuous and effective way, being at the same time a good-looking and user friendly device.

Among the several functionalities, the robot should be capable to find the information about their flights, to help with check-in operations with the baggages, to indicate the different points in the airport to find food and beverage and to guide them to the right gate for the departures.

To accomplish these tasks, the project is divided into subsystems. Some of them are completed, but need to be improved, while others have to be start from scratch:

- Mechanical Design
- User Interface
- Depth-based SLAM
- Task assignment optimization

- GPS localization
- Autonomous charging system

The first three items were the necessary first steps for the project and the development of the mobile robot and they had been completed last year.

1.2.1 Mechanical design

The mechanical design has been developed trying to obtain the mechanical structure of robot as reliable and controllable as possible, being at the same time good-shaping and practical for the tasks the robot has to fulfill. The design process included the choice of the wheel configuration, the drive train design and the CAD modelling structure.

As far as the wheel configuration is concerned, the choice had been conducted by taking into consideration the simplicity of control, the degree of steerability, the number of driving motors and the simplicity of mounting. For the other two design choices the research had been carried out by taking into consideration the performance of the robot and the esthetical needs of the client.

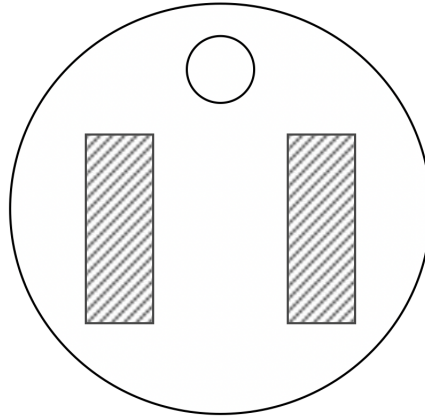


Figure 1.1: Differential drive configuration

The solution to the first problem passed through the analysis of the different kind of wheel configurations and wheel types. At the end, the differential drive configuration prevailed over the others thanks to its simplicity of realization and reconfiguration. Differential drive means that the robot has one motor for each wheel to be controlled.

It is also characterized by a faster and more reliable design and assembly than the other configurations. The number of idle wheels (empty circle in *Figure 1.1*) can be changed without affecting the kinematics of the system.

1.2.2 User Interface

The goal of this sub-project was to design an application to make the robot capable to communicate with the end-user, the client server and the motherboard. Another time, being an engineering project, the mission was to create an interface as simple as possible, intuitive and beautifully designed way.

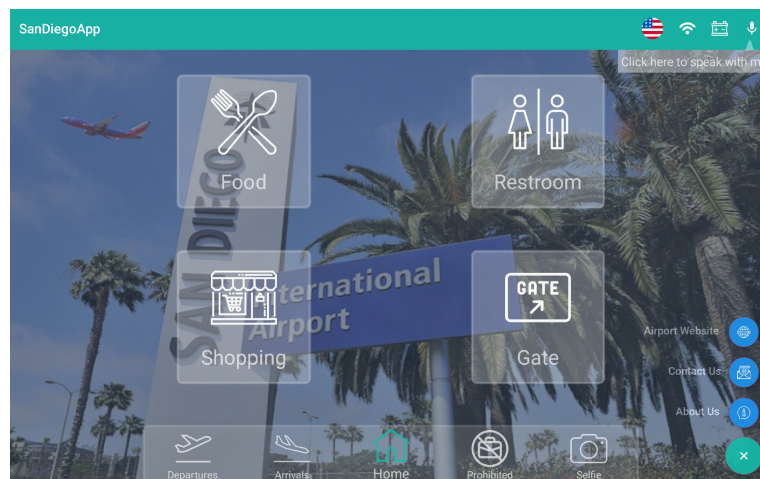


Figure 1.2: Airport concept app

In addition to the interface, a speech recognition algorithm has been integrated for simplifying to the maximum the front-end and to reduce the number of actions the end-user has to perform to reach the service, gaining time. In this connection, a case study on Hidden Markov Models has been conducted, being the most widespread algorithm in the speech recognizers.

Regarding the communication system, the goals were to address to these issues in a safe and cheap way, to not affect the security airport information and to not burden the cost of the robot.

1.2.3 SLAM Depth-based

This is the last subsection that was already done at the start of this work. SLAM stands for Simultaneous localization and mapping. It is a computational problem of constructing and updating a map in an unknown environment. To tackle this problem implies the capability to build a map of the environment while the robot is moving and at the same time to localize the robot inside that map. It enables mobile robotic systems to move inside unknown surroundings and to simultaneously locate themselves, indeed avoiding the need of a structured map.

Such a technology involves the use of depth-based sensors able to record and store 3D data of the environment. These informations are then used to compute the robot position either with respect to a global reference frame or to a relative reference frame based on the previously stored image.

Anyway, SLAM is a chicken-or-egg problem since to localize the robot in the environment is needed a map, while for mapping, the informations about the precise position are required. Thus, the two aspects have to be approached in a dependent manner.

This was the core of the project. Indeed, this algorithm had to allow the robot to take the first step to move independently in the airport environment.

SLAM algorithms take advantage of the map to update the position and orientation of the robot through either the Lidar or cameras or a combination of two. It needs data of Depth from the sensors, a localization algorithm (e.g. Monte-Carlo localization) or a localizer as a GPS system and eventually a map.

The algorithm can work in two ways. The first is without a map, navigating inside the system remotely driven and creating a map of the same by means of the depth-sensors and odometry data and at the same time localizing itself. Either with a map, previously created, finding the position of the vehicle by comparing the elements and obstacles of the updated map and the depth data. In this way, the robot can navigate in the environment according to this information.

As it was mentioned before, the depth data can be taken from cameras, whether monocular or stereocamera. In fact, they represent the base of visual Simultaneous localization and mapping (vSLAM) technique. The images are used to create a map of the environment in order to globally position a vehicle or they can be used to track the motion of an autonomous system by subsequent reference frames transformations, each of them related to the previous state as the state evolves in time.

A part from the localization contribution, the cameras are useful also for extracting informations about possible active or passive obstacles together with its distance to the robot and used in a object detection algorithm if needed. Moreover, the camera can be

instructed to distinguish one object from another improving the potential of the SLAM algorithm.

Finally, the ZED 1 stereocamera has been chosen as the main sensor for the vSLAM algorithm.



Figure 1.3: ZED 1 stereocamera by Stereolabs

The stereocamera is a device composed of two cameras sensing at different position. This kind of sensors rely on the principle of Triangulation. By comparing two images coming from the two lens, it is possible to extract the three dimensional information from two bi-dimensional images. In this way, the depth of the environment is extracted and the data sent to the SLAM algorithm to be elaborated.

The tests have been conducted in the laboratory of California State University of Los Angeles. As first, the robot was remotely driven all around the environment to capture the frames. These frames were elaborated and stored inside the database that created the three-dimensional map of the location.

Then, the SLAM algorithm was launched. The robot tried to localize itself by comparing the new depth data with the map and at the same the algorithm updates the map. As we said before, the algorithm can work also without a map. Tests have been conducted without a previous updated map without meaningful results. Due to the lack of a redundancy over the data from other sensor like IMU or Lidar, the quality of the results was not satisfactory as the one with a preloaded map.

At this point, the data of the localization were sent to the other nodes of the robots allowing eventually an autonomous navigation system.

1.2.4 Task assignment optimization

This problem refers to a fleet of robots working in the same environment that receive requests from the users or from the holder (i.e. tasks) to be accomplished as soon as they could.

This concept has been developed in the context of shared mobility. The users asking for

services from the robot have been considered as customers waiting for a pickup from a taxi rather than an Uber or Lyft vehicle. Moreover, this innovation is expanded with the ride-sharing service like UberPool that accounts for the requests of the customers who wants to share the ride with someone else.

The purpose of this work is to manage the fleet with the aim of reducing the cost derived by the transportation, satisfying as many as customers increasing the occupancy ration of the vehicles.

To address such an issue it has been designed a dynamic ride-sharing strategy for the assignment of vehicles to users that includes also a demand forecasting algorithm to predict the requests based on the different areas of map. Dynamic means that the algorithm deals also with the disposal of additional requests while the vehicle is handling the hanging ones trying to meet all the customers time requirements.

What the algorithm does is to build a so called shareability graph describing the couplings between requests and vehicles. This then generates a set of feasible trips and linear programming problem assigns these trips to the vehicles. An issue that arises from this approach is the imbalance between the demand and available vehicles that causes unserved customers. So to try to serve unassigned customers first, the approach has been improved with a rebalancing phase in order to occupy idle vehicles with these requests.

1.2.5 GPS localization

The purpose of this subsection of the project is to improve the localization functionality of the robot by implementing a GPS localization system. As an autonomous robot requires a very precise localization, such a feature significantly increase the accuracy about the information of the position of the robot especially in outdoor environments.

By the way, the SLAM node mentioned above makes use of landmarks for the localization. In an open space configuration, there could be a lack of landmarks and the function struggle to work properly. Employing a GPNSS module addresses this issue, making the system more robust.

Without high obstacles (i.e. skyscrapers, high buildings, forests), some GPNSS modules can reach even an accuracy under the centimeter allowing a more range of autonomous services the robot can provide.

The module that has been employed in this case is the U-Blox ZED-F9P receiver. The technology behind this device is called RTK (Real-Time Kinematics) and belongs to a bigger category called differential GNSS. This latter takes advantage of correction data provided by so called base stations, which as the name suggests are receivers that are

placed in specific positions and whose extracted coordinates can be surveyed and found accurately.

A fusion of GNSS data and SLAM data gives a position which is more robust and accurate in presence of external disturbances, like the presence of trees or the absence of landmarks.

1.2.6 Autonomous charging system

This topic is the theme of this work and it will be discussed further in more detail. The additional functionality of an autonomous charging system for the robot, was born out of the need to have a robot that continuously accomplishes tasks without the intervention of the humans.

This represents a step for the project that makes the robot competitive and attractive for the customer. Thanks to this solution, the service can increase its capacitance of tasks that can be admitted, allowing the formation of a queue of tasks ready to be accomplished. Moreover, the implementation of an autonomous system to charge the robot is capable to handle the problem of stopping around the robot whenever it runs out of the queue. Instead of having someone to move the robot in a open spot or to instruct the path planning algorithm to do it, the robot can reach the closest charging station and charge itself until another request from the users appears.

It really represents a good step to achieve complete automation for the robot. However, if one considers all the subjects that have to be addressed like the design of the docking strategy, the management of the sensors data to or the realization of tests with autonomous driving robot, this implementation is a truly challenging issue.

But the real challenge is the precision. It is not sufficient to make a system to move the robot from a position to another. This system has to guarantee to slowly follow a precise strategy as safe as possible. Slow, precision and safety are the keywords for this module and they have to be taken into account along its actualization.

Chapter 2

State of the art

With the increasingly research on robot in the last decades, its role became more and more important. Mobile robots are designed for interacting with human environments and for working with humans on a daily basis. The robot systems are widely employed in many applications such as transportation, factory automation, dangerous environment, space exploration, military, security services, hospitals, surgery, farmer and other services like reception, assistance and entertainment.

To be considered of any use, these robots must exhibit some form of self-sustainability. In addition, to being robust in their physical design as well as control methodology, such robots must be capable of long-term autonomy. Energy is of great concern, and without it the robot will become immobilized and useless.

In this work it is presented a method for autonomous charging of the robot, a necessity for achieving long-term autonomy.

The robots have to implement either sensors or cameras for perceiving the environment they work into.

Before going on the actual implementation of autonomous recharging system, subject of this work, let's consider first some relative significant projects.

The first works about the docking operations include [4], whose author used four IR LED emitters and one IR receiver to implement the docking operation for its robot. Instead of the work of [5], in which the robot approaches the recharging station using the map and makes contact with specially designed battery recharging system. Furthermore, some systems are using vision-based method to implement docking [4, 5, 6, 8]. The robot designed by Silverman [8] is considered the state of the art of the academic implementations in the early 2000s. It consists of an immobile docking station and the docking mechanism mounted on the robot. Vision and laser ranger-finder are the systems used to find the docking station.

Moreover, the choice of the type of charging deserves some lines about. The first works utilized a mechanical coupling with a sufficient tolerance due to the given special forms (e.g. hemisphere) for increasing the possibility to center the plug. In the last years the Wireless charging technology is making room among the other solutions thanks to its reliability, durability and safety intrinsic characteristics.

Other methods include the battery switch or even the robot switch (e.g. the widespread sharing scooters) and the snakelike robot arm, whose purpose is the opposite of the previous ones. In this case is the docking station that has to be programmed to go towards the robot, which is parked for charging the robot itself.

One significant aspect that is of great concern in the field of charging methods is the time that the station takes to recharge the batteries. This characteristic influences a lot the choice of method to be employed by the companies. The shorter the time the robots take to recharge, the longer they can take to perform more tasks and not stand still.

Now, let's have a look a bit more in deep into the different kinds of charging systems.

2.1 Charging systems

Typically, rechargeable batteries may provide few hours of peak usage. 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. Inherently, mobile robots can perform a finite amount of work in a single charge cycle which is determined by the capacity of their batteries.

Recharging is necessary before the power of the robot becomes insufficient. Each robot should have the capacity to plan its actions taking into consideration the available energy and scheduling of recharging tasks. Autonomous recharging brings with it the promise of extended runtime, reduced need for human intervention, and enhanced system performance. Current research in autonomous recharging is disjointed and focuses predominantly on addressing one of two questions: how to recharge and when to recharge.

This section and the next one will address the first question. In particular this part will tackle the research made upon the charging systems to be employed in this project as well as a comparison among them.

2.1.1 Docking: Mechanical coupling

The Docking charging system consists in a static or actuated plug, which is inserted into a socket, placed in the robot. The plug is located in a specific infrastructure, designed according to the socket and to the robot specifications (electrical and communications services).

The autonomous system of the robot usually requires a series of sensing mechanism, such as lights and marked and colored stuff on top of the station. These are features giving to the control system the capability to recognize the docking infrastructure.

The docking charging system is adopted by the main companies manufacturing mobile robots. Almost all literature relative on the autonomous charging robots employs this type of charging system. This spread of such a technology is a consequence of being an easy to design and implement system, which doesn't need too much computational cost with respect to the wireless charging system. Moreover, it represents a complete robust and reliable system which requires constant maintenance to avoid possible damage or erosion coming from dust, water, vandalism or not-proper use of the instruments.

As an instance, the solution of Fellows robot on *Figure 2.1* is an example of docking coupling. Fellows provides as service the autonomous navigation and 3D mapping software to the customers. It represents an advanced computer vision solution for inventory scan-



Figure 2.1: Fellows Robot: Example of mechanical coupling between station and robot

ning and supply chain automation. It provides advanced algorithms for label and product recognition, barcode decodification, data and position extraction. Fellows platform integrates with fixed-position cameras to do image recognition of inventory on shelves.

As regards the charging system, the station has two pins and the robot is provided with the correspondent two plugs. The coupling has a tolerance to increase the possibility of a successful connection.

This kind of plugs imply that the robot approaches the station following the perpendicular line to the station. The robot has to stick to the pins with an angle nearly null.

Actually a lot of autonomous service robot adopt a mechanical/electrical coupling for their charging system. Another example is the Savioke of Relay, an autonomous service robot employed in an hotel environment capable to deliver items to hotel guests. It is characterized by a strong integration with the hotel facilities like phone system to alert guests when delivery is ready or hotel floors and elevators map to navigate autonomously in the hotel environment. As regards the station, the coupling provides an union between station and robot. As long as the robot is mechanical attached, the station is capable to supply the robot batteries.

The reason why this technology is so widespread is the time the battery takes to recharge



Figure 2.2: Savioke Relay during the recharge process on the station

it completely. The charging time is of great concern in the recharging sector. Let's for example suppose that the robot, with full charger battery, can last $5/6$ hours. Now, let's suppose that the charging system takes 2 hours to bring the battery from a low battery level to full charge. In this case, one quarter of the operating time of the robot is spent for the recharging process. This can be inadmissible if the robot has a queue of tasks to be accomplished and it will cause delays for the customers. The mechanical coupling solution represents a fast charging mode that minimizes this relation between the charging time and the operating time.

2.1.2 Wireless

Wireless charging, also known as wireless power transfer, is a technology that is capable to transmit electromagnetic energy to an electrical load across an air gap, without any kind of wires.

Such a technology is actually attracting a wide range of applications, from earphones to high-power electrical vehicles. Pike research [11] estimated that wireless charging would be a 15 billion dollars market in 2020.



Figure 2.3: Wireless charger compatible with Apple devices

The spread of the wireless devices could lead to an elimination of the busy interconnection cords, being more reliable due to the lack of intermittent physical connections. In addition, the lack of an external infrastructure preserves from any form of erosion and damage of contacts caused especially by the dust and the water. In conditions of outdoor infrastructure or underwater stations, the wireless charging may be the best solution of all allowing a long-lasting system.

Moreover, when installed on public spaces, such a type of arrangement prevents vandalism and being noninvasive, it can not clash the surrounding environment.

Therefore, all of these features bring to a robust and safety system able to carry and harvest energy without any kind of physical connection.

However, some critical aspects must be taken into account. First of all, the wireless charging methods lead to an unbearable heat generation, so that it is necessary to find a suitable solution to mitigate such a problem. Second, the final implementation can last more time than other type of services due to the bureaucratic and legal issues. In fact, at

this time, a standard form of wireless charging is not yet realized.

Nevertheless, the most critical aspect is related to the high relative cost of prototyping and implementation. When starting a project, the total amount of the investment is not known a priori and it could become exponentially higher than the traditional ones.

Now, let's analyze the different wireless charging methods.

Classification

The development of wireless charging technologies is advancing towards two main directions, radio frequency (RF) based (or radiative) wireless charging and non-radiative wireless charging. The radiative wireless charging develops electromagnetic waves, RF waves or microwaves, as medium for delivering energy, while the non-radiative wireless charging mechanism consists of a magnetic-field coupling between two coils, being the distance among the coils the dimension for energy transmission.

However, due to safety issues raised by RF exposure, radiative wireless charging is mainly used for low power applications, such as sensor node applications with up to 10mW of power. As far as the non-radiative wireless charging is concerned, thanks to its safety implementation, is largely wide employed in our daily appliances (e.g. from toothbrush to electric vehicle charger).

Indeed, the wireless charging can be implemented in different techniques.

The *Inductive coupling*: Inductive power transfer (IPT) happens when a primary coil of a transmitter generates predominantly varying magnetic field [12] across the second coil of the receiver. The operating frequency is in the order of Kilo Hertz range, while the effective charging distance is generally within 20 cm.

The diffusion of such technology is encouraging by the easy implementation, the high efficiency in close distances and the ensured safety, so that it has been developed for mobile devices.

To enable comfortable and autonomous battery charging lot development effort is invested in inductive charging systems. Due to the advantages of the contactless method, manufactures are working on the marketability of this technique. But several challenges like energy losses, electromagnetic radiation, complex vehicle adaption or environmental impacts on humans and animals still have to be solved. Furthermore, long driving ranges together with short charging intervals are essential for a high customer benefit. Due to a significantly lower power transmission performance, high charging capacities are not feasible with inductive systems.

Figure 2.4 depicts how many kilometers per one hour of charging can be covered depend-

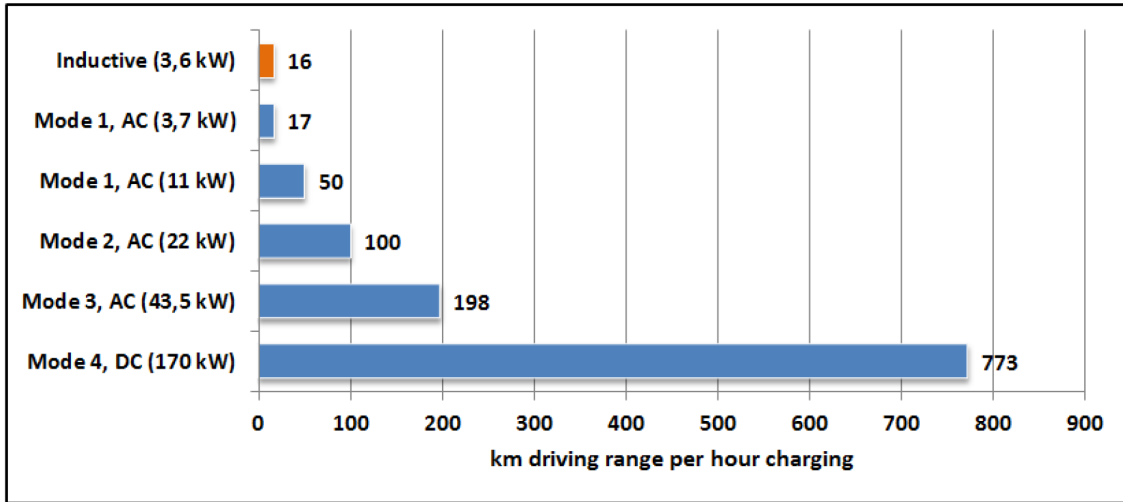


Figure 2.4: Graph comparing the different charging solutions by the driving range per hour of charge.

ing on the loading method and based on the energy consumption of an average electric vehicle. In this comparison, charge and discharge losses are not considered. As shown fast charging technologies with loading capacities up to 170 kW can be achieved with mechanical couplings and they allow to reduce the charging time significantly in comparison to the low charging methods [13].

The *Magnetic Resonance Coupling*: this coupling happens between two resonant coils, i.e. coils characterized by approximately the same natural frequency, thorough varying and oscillating magnetic fields. Its working frequency typically varies around the megahertz range.

The resonant property allows to widely reduce the leakage of field caused by the neighboring environment, so that the distances between the coils can be much greater than that of inductive. Furthermore, the line-of-sight requirements is no longer needed thereby allowing a concurrent charging of different devices.

However, the essential presence of a bulky capacitance system in the receiver doesn't allow the implementation on the mobile devices. An example of such a technology is the robot designed by a team from the Massachusetts Institute of Technology, Witricity [15].

This technology was conceived for all kind of electrical vehicles. The company, Witricity, provides tailor-made systems according to the implementation. This system is capable to provide a good power (3.6kW 11kW+) enough to be comparable to some docking first level solutions. We will discover later how the power and the runtime of a vehicle are



Figure 2.5: Witricity: an MIT implementation of magnetic resonance wireless charging

related.

The *RF Radiation*: the diffused RF/microwaves are the medium to carry radiant energy and they propagate over space at the speed of light. Their working frequencies range from 300 MHz to 300 GHz. The distance between transmitter and receiver could be several tens of meters up to few kilometers. Nevertheless, how it is mentioned about earlier, due to such a range of high frequencies, an high-power implementation is not sufficiently safe, relegating their employment only for low-power implementations.

Table 2.1 helps to summarize the most common wireless technologies together with the relative applications and features.

2.1.3 Snake-like solution

Last but not least, there is another implementation that has been taken into account during the decision-making time, the snake-like charging systems. Actually, this kind of implementation regards a mechanical coupling as in the first case. However, in this case, it's not the robot that has to reach the station, but the station has to control an arm to dock the robot.

An interested implementation of this method is built by a Swiss mechatronics company, Staübli [16]. Such company was founded in 1892 as a textile machinery manufacturing company. After the acquisition in 2002 of the majority stake of a company leading

Wireless technologies			
Wireless technique	Advantages	Disadvantages	Applications
Inductive	Safe for humans, simple implementation, suitable for mobile applications	Short distance, heating effect	From electric toothbrushes to electric vehicles (EVs)
Magn. resonance	Charging multiple devices different power (scaling), high efficiency (over 90%)	Not easy for mobile applications, complex implementation	Electrical vehicles and industrial robots (Witricity), consumer electronics
RF radiation	Long distance, suitable for mobile devices	Unsafe and inefficiency	Low-power communications applications

Table 2.1: Pros, Cons and Implementation method of the docking solutions

provider of electrical connectors and with other further acquisitions, the company became a world leader of connectors for electrical charging systems [17].

Currently, the company is engaged with a lot of contracts all over the world. As a sake of example, they won the contract with SSA Marine, which manages the terminal at the Port of Long Beach, California. The project includes a transformation of 33 diesel-powered tractors by retrofitting them with all-electric drivetrain systems [18]. Staübli will manage the charging part in collaboration with Tritium, an Australian company that designs and manufactures fast-charging solutions for electric vehicles.

This latter will install its Veefil High Power Chargers (HPCs), type PK 175kW DC to power SSA Marine's fleet of terminal tractors. Staübli's QCC system features an enclosed pin-and-socket design that is self-cleaning, touch-protected on both sides of the connector and easily corrects for misalignment.

The rollout of both the vehicles and chargers at the Port of Long Beach will bring a zero-emissions environment to the port, enabling a cleaner air environment by eliminating diesel fuel and reducing noise pollution. Moreover, this project will create the largest automated electric vehicle charging program of any port in the United States.

The Staübli connector is a compact, versatile and fully automated docking charging solution suitable for AVGs, busses, trucks and mobile robots. It is capable to reach even an high transfer power of around 300 kW, that is comparable with the best docking solutions. Such an high power transfer capability is gained by the large coaxial cable that

can implement.

However this kind of solution requires much effort for the control system, due to its reverse approach.

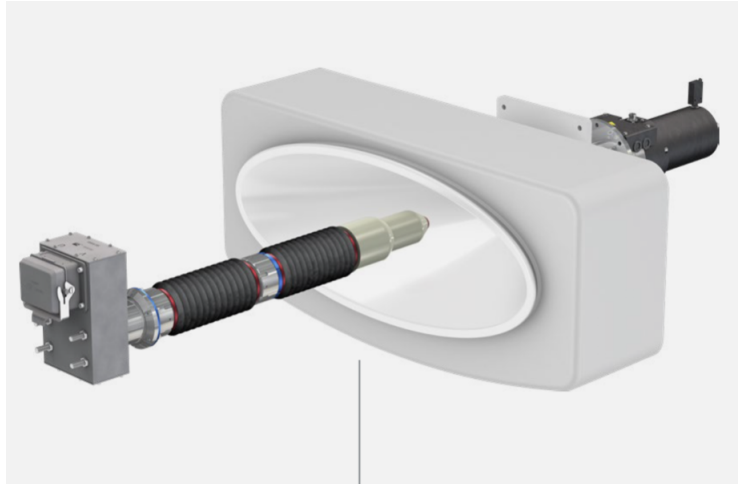


Figure 2.6: Staübli connector: a snake-like charging system

This connector represents a reliable and sustainable solution thanks to its capability of over 1 million of charging cycles. In addition, it integrates a self-cleaning mechanism and a safety system to protect the users.

At the end, this solution allows an high tolerance of the docking operation. The large socket accepts the male plug with an error up to 500 mm, increasing a lot the odds to accomplish a successful docking operation.

2.1.4 Comparison

The three main actable solutions that have been accounted in the decision-making time was: a custom mechanical coupling plug, a magnetic resonance solution such as Witricity and a connector such as the Staübli one.

The analysis to chose the best solution has benn made by taking into account the cost to design, implement and maintain the plant, the efficiency, the quickness of the charging operation, susceptibility to damages and vandalism, automation-friendly solution and the easiness of control and use.

To summarize, the following table (*Table 2.2*) helps to better figure out the advantages,

Docking solution			
Docking technique	Pros	Cons	How to implement?
Witricity	Less maintenance, charging multiple devices, high efficiency market solution, safe	High cost of computation and hardware/software implementation	Tailor-made system together with assistance and maintenance before and after the installation
Staübli	Robust and reliable market solution, quick-charging, self-cleaning system	High cost of implementation and maintenance, susceptible to damages and vandalism	Tailor-made system together with assistance and maintenance before and after the installation
Mech. coupling	Low cost of implementation, flexibility, easy to use and control	Constant maintenance and time consuming	Design of mechanical, electrical and software part from scratch

Table 2.2: A comparison between the different wireless technologies on the advantages, disadvantages and applications

disadvantages and the methods to implement them.

The choice of the charging system to be implemented in the present work fell to the utilization of a tailor-made wireless charging system. The complete safeness of the technology, its reliability and its intrinsic automation-friendly together with its increase in commitment in the service robots applications are some of the reasons that justify such a choice.

2.2 Docking strategies

This section is going to explain how the autonomous robot conducts the detection and tracking of the station. The analysis of the sensors on the market with a focus on infrared sensors, Lidar, cameras and localization sensors and how they can be coupled with other elements such as QR codes, colored marks and special shapes of the station. The analysis will include also RFID detection systems.

2.2.1 Market sensors

This summary starts with a very interesting technology, the *infrared sensors*. It is a device that measure the infrared light coming from objects in its field of view. It takes advantage of the object radiation.

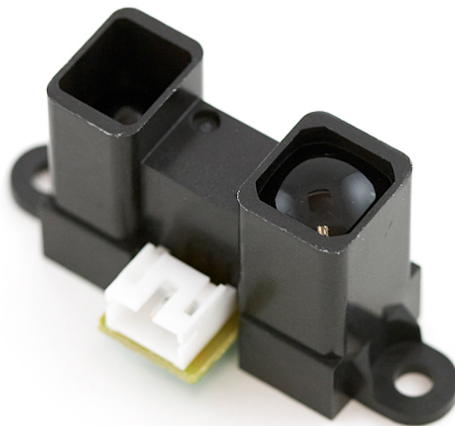


Figure 2.7: An example of infrared sensor - SHARP GP2Y0A02

They are easy to implement and their price is very low. They don't need cameras and they can work also inside a dark environment, since they don't need any particular light conditions for properly working. They are capable to detect the distance between transmitter and receiver as long as they are installed either one in correspondence of the station and one on the robot or both on the robot. They are also capable to detect obstacles and thus their distance by comparing the distance the light takes to reach the obstacle and to come back with the light velocity.

This kind of sensors are widespread on mobile robot thanks to their high cost-effective functionality.

One example is the Kobuki project robot [19], that is a low-cost mobile robot designed for education and research and represents the state of art of service custom robots. It implements three IR sensors for detecting obstacles and for the docking operation. As well as this kind of sensors are employed also in the parking detectors system of the common cars.

However, they are color-sensitive sensors and they need to be programmed to detect a certain type of color or another.

Moreover, this kind of sensors are capable to sense a little portion of the environment. Thus, to detect the shape of an object, they have to rotate themselves such to scan a larger portion of space. This is the way, for example, Roomba iRobot executes its docking operation. The vacuum robot is always rotating to perceive the indoor environment it is cleaning.

As a consequence, to manage the data coming from this kind of device is not as easy as other sensors and are not reliable as well.

Then, there is the big brother of the infrared sensors, the *LiDAR*. This latter device is an efficient method to scan surrounding environment in two or even three dimensions.

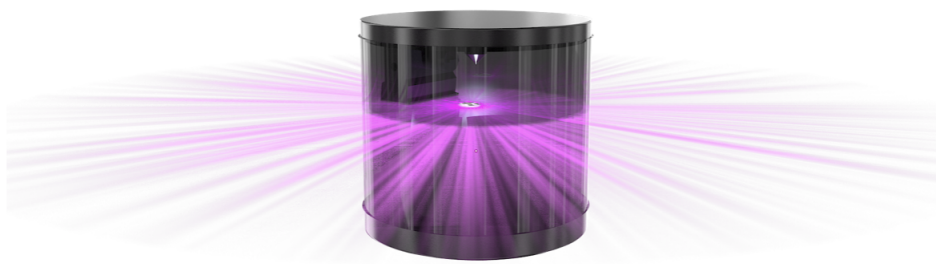


Figure 2.8: An example of LiDAR sensor

These technology works in a fashion similar to the one of the Radar except from the fact that it exploits pulses of light in place of radio waves. LiDAR, as the infrared sensors,

calculates the time it takes for the light to reach the obstacles in the environment and hit back the emitter, in this way this technology senses the distances from objects occupying the space. In case of three dimensional scanning, LiDAR can be used to build point cloud based maps from which different objects shapes can be recognized.

The way in which this device can detect a station is to give to this latter a shape such that an algorithm can recognize the difference in distance and so the station.

Another widespread technology in the field of the autonomous charging system and in general in the autonomous driving applications is the use of *cameras*.

This is a very old device (the first rudimental camera is more than 300 hundreds years old), but it keeps improving its performance.



Figure 2.9: Raspberry Pi 4 Camera, an example of robot implementation camera

The easiest way to implement a docking station detection with a camera is to place in correspondence of the position goal either a QR code, bar code or even a mark. Knowing the size of the mark, it is possible to extract the distance between the robot and the station by comparing the dimension on the visualization with the actual one of the mark. With this simple trick, a control system has the feedback to drive the robot towards it. Therefore, items like QR codes can store information like the position of the station and thanks to their insignificant cost, they are widely employed in the field of autonomous charging systems.

An advantage of the camera is that they can not only detect codes or tags but they allow a wide angle perception of the environment and therefore of obstacles. However, they need a sufficient ambient light to properly work. Even to have too much light can bring to an incorrect operation, due to the inability of the camera to maintain the focus.

However, visualizing only 2D images and without any objects of well-known size, it is impossible to calculate the distances with only a monocular camera. Different is the case instead of the stereocameras. This device is equipped with two cameras and thanks to the stereoscopy technique aimed to compare two two-dimension pictures and extract the information of the third dimension.

The main drawback of this tool is the high computational effort required when managing heavy data like images visualization.

The *RFID* is the acronym of Radio-frequency identification detection systems takes advantage of the electromagnetic fields to automatically identify and track tags attached to objects [20]. An RFID tag consists of a tiny radio transponder.

There are two kinds of tags, passive and active. The formers are powered by energy when triggered by an electromagnetic interrogation pulse from a nearby RFID reader device, while the latter has a little supply battery and therefore it can be read from the reader at larger range. Unlike a barcode, the tag doesn't need to be within the line of sight of the reader, so it may be embedded in the tracked object. The tags transmit digital data that can be an identifying inventory number, back to the reader. This number can be used to inventory goods.



Figure 2.10: RFID detection system

The RFID systems are widely used in plenty of applications in many industries. For instance, it is possible to track an automobile during its process of production simply by attaching it a RFID tag. RFID-tagged pharmaceuticals can be tracked through warehouses or even they are used in livestock and pets enables positive identification of animals.

In addition, the RFID reader can read more than just one tag per time, i.e. "Bulk reading". Bulk reading is a strategy for interrogating multiple tags at the same time. On the contrary, as tags respond strictly sequentially, the time needed for bulk reading grows linearly with the number of labels to be read, therefore the time required is greater.

The great disadvantage of such a device is the limitation on the range of detection. With passive tags, the robot could be recognized by the reader only starting from a distance shorter than about 2 meters.

Moreover, it has to be coupled with at least another sensor system such as infrared sensors to extract the information of the position and the orientation of the robot. Due to to this limitations, the autonomous robots are nearly never equipped with an RFID tag.

Last technology it is worth talking about is the GPS localization system.

A *GPS* is considered to be a Global Navigation Satellite System (GNSS), that means it is a satellite navigation system with global coverage, and it provides location, velocity and time synchronization.

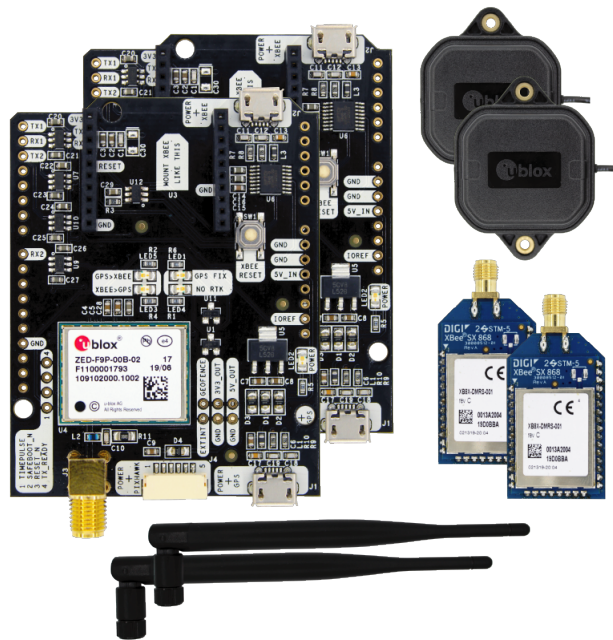


Figure 2.11: U-Blox ZED-F9P GPS system

GPS is everywhere. It helps you get where you are going, from point A to point B.

It is a navigation system that uses satellites, a receiver and algorithms to synchronize position, speed and time data for air, sea and land travel. What needed to produce a location on earth's surface are three satellites. A fourth satellite is often used to validate

the information from the other three. The fourth satellite also moves us into the third-dimension and allows us to calculate the altitude.

A single satellite broadcasts a microwave signal which is picked up by a GPS device. It creates a sphere with a radius measured from the GPS device to the satellite. The second satellite adds another sphere. The intersection with the third one to two points. The point closest to the Earth is chosen and the location can be localized [21].

The GPS systems are mainly used for localization and tracking of objects, the goals of this projects. A robot equipped with this device has a very accurate information on the position even when the robot is moving.

Nevertheless, the inability to detect active obstacles forces the robot developer to add another sensor system to address this issue.

In spite of GPS survey allows to precisely detect both the station and the robot and can easily extrapolate also the angle with respect to the station. It works also with high distance and it allows a better automation of the robot.

If the GPS localization system is characterized by an accuracy in an external environment, with an indoor environment it turns to be worsened, since the precision in this case dramatically goes down.

Anyway, to implement this system in a robotic application leads to a very huge cost, that makes the robot not competitive with others. Furthermore, such technology, more than in other systems is characterized by the formula the more is expensive, the higher is the accuracy.

This analysis of the most employed sensor systems shows off their advantages and disadvantages together with their applications and cost. In most of the location identification systems cited above, it is needed to turn away for detecting the marks or codes or the station. This movement may increase the predefined error that characterizes them. In addition, the uncertainties coming from noise or errors can affect the performances of the systems. Thus, just one system for the identification of the location is not sufficient. It is needed a sort of redundancy to improve the reliability of the system (Sensor fusion). Now let's figure out how these sensors are implemented in some academic custom robots and how they work together to accomplish an autonomous charging system.

2.3 Existing systems

The detection of the docking station is the first step to accomplish a successful docking operation. Nowadays, the robots employ more than just one sensor and strategy for having a redundancy on the perception. As a sake of example, Roomba iRobot ([22]) employs IR sensors and a strategy of turning clockwise and counterclockwise for both the detection of the docking station and the navigation. In this way the infrared sensors attached to the robot detect the stations, also communicating the distance. In addition, the robot is capable to discover obstacles in its area. iRobot is a cheap solution, but it requires a tremendous effort on research to manage the infrared sensor data. Moreover, the employment of just IR sensors restricts the potentialities of the machine, that in the case of the vacuum robot is not a limitation, but only a choice to reduce the cost of the robot.

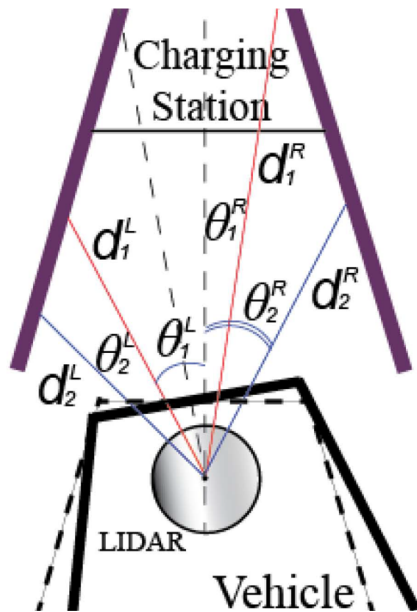


Figure 2.12: Station and robot design of University of Detroit implementation

Instead of other custom robots, such as the system designed by the University of Detroit Mercy [23]. This solution, in fact, doesn't rely on only one source of data. It couples the LiDAR sensor with a special form of the docking station. The infrared sensors of the Lidar recognize the difference in the distance and recognize the station. The Lidar

senses the distance and angle. There are three different fixed angles and for each angle the distance is extracted. Based on the distances, the robot is instructed to turn right or turn left or to go straight. If the distance of the left angles is lower than the right one, the robot has to turn to left and vice versa. The high tolerance given by the shape of both the station and the robot allows an high probability of successful docking operation.

Another solution is the one of the FIT IoT Lab. It utilizes for the docking operation both IR sensors and a camera for the detection and tracking of two QR codes. This kind of redundancy guarantees an high probability of success, even if it is characterized by an higher cost of elements.

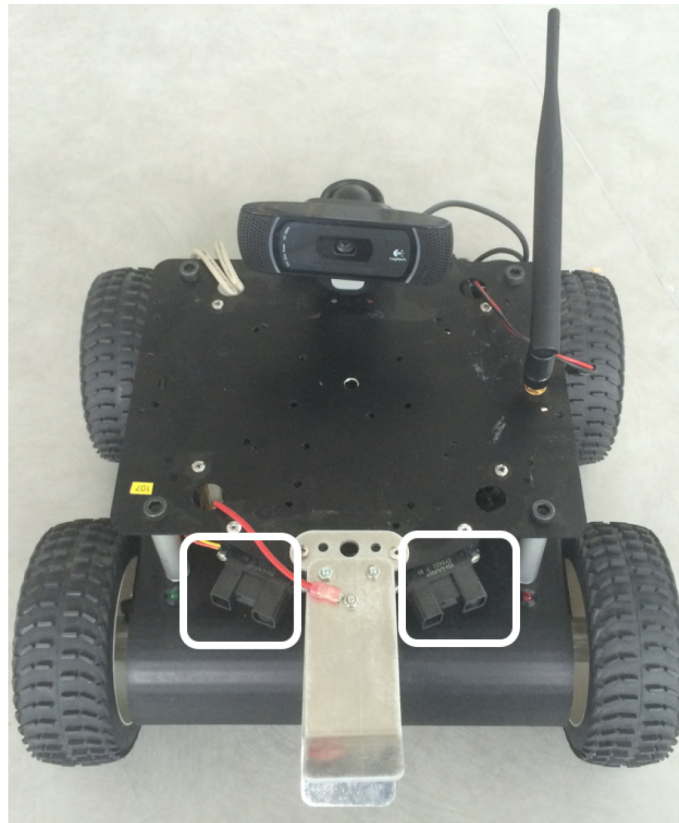


Figure 2.13: FIT IoT Lab robot

The infrared sensors are in the back of the robot as they are enlightened with white squares in *Figure 2.14*. The method benefits from two key concepts: The first is image processing using a camera module for decoding QR codes. This allows the robot to determine its position relative to the dock. The second concept is the use of infrared sensors to avoid obstacles and perform alignment with the dock.

The basic idea depicted is that the robot turns around itself looking for a QR code. It

takes a picture, decodes it, and checks whether there is a QR code with the desired information encoded. If the QR code is not found, it turns (around 30 degrees), and tries again. Once the QR code is found, the robot will align with the QR code, i.e. it will turn little by little until the symbol is centered in the middle of its image [24].

From the relative size of the lateral edges of the QR code, the robot determines whether it is in the right side, left side or centered and its position. Then, the approaching procedure algorithm considers three regions depending on the distance to the QR codes.

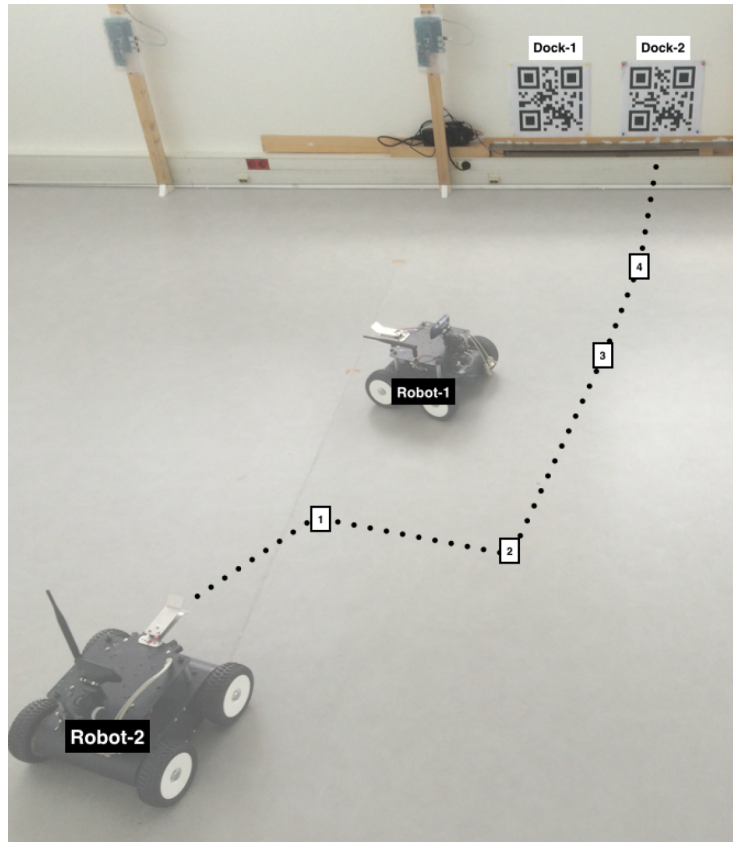


Figure 2.14: Docking strategy of the FIT IoT Lab robot

So, if the robot is *FAR* from the dock, i.e. the distance is higher than 1.5 meters, it directly approaches going straight towards the dock until it leaves this region.

When the distance becomes shorter than 1.5 meter (i.e. the robot enters in the *CLOSE* area), the robot slightly turns to correct the angle of attack and heads the QR codes until it reaches the *VERY CLOSE* zone, that means the distance is lower than 0.75 meters. At this point the robot checks whether the attacking angle is adequate. This angle estimation is done using both IR readings and the differences between the size of the lateral edges of

the QR code.

If the angle is good enough, the robot advances straight to dock till it is docked. Otherwise, it comes back to the other region adjusting its angle and repeat the other operations. Another example is the one presented in [25], which is the fruit of the studies made after the Conference of Mechanisms and Machines in Bangalore.

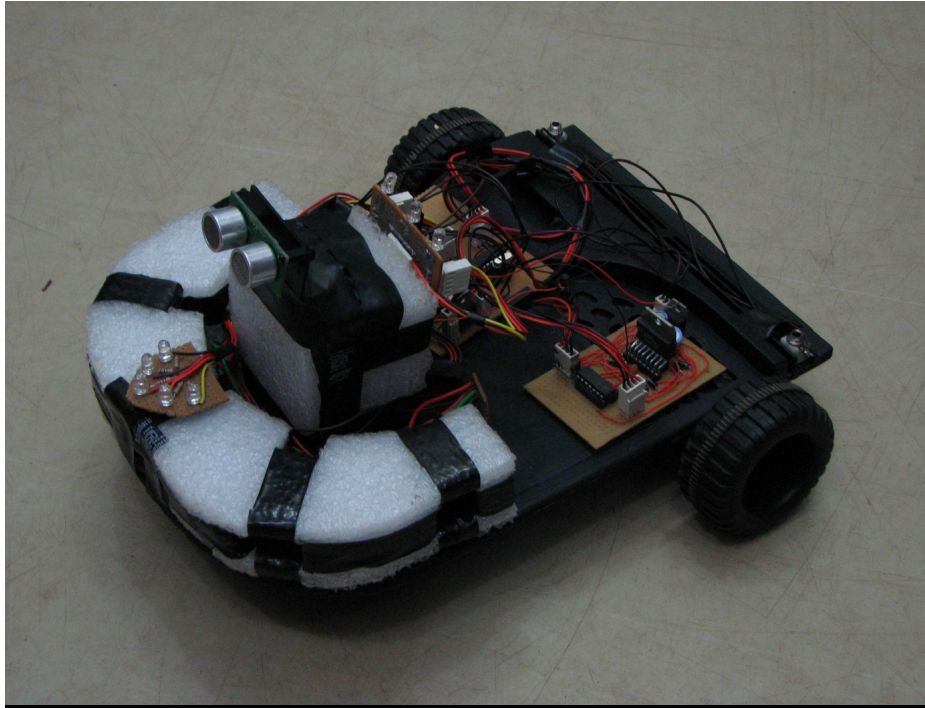


Figure 2.15: The custom robot fruit of the Conference of Mechanisms and Machines in Bangalore

It employs both Ultrasonic range sensors and infrared LED, while the docking station is provided with three beacons. These beacons are placed in such a way that the docking target forms the midpoint of the line joining the beacons. Even in this implementation, sensors data sources are got and fused to control the robot.

The purpose to use an Ultrasonic range device is to have redundancy on the LEDs information and from them extract the distance and position of the station.

The robot, using IR beacons, with the help of a range sensor (e.g. Sonar) can detect where these beacons are located. Based on the position of the beacons, using the concept of triangulation, the robot orients itself normal to the line joining the beacons. Thus it reaches the required docking location with proper orientation.

So, in the docking algorithm the robot detects the first beacon and moves towards it. The

robot will meet the 1m circle boundary (the circles in the *Figure 2.17*). As it reaches the 1m circle boundary the robots searches for the second beacon moving towards this latter. When the robot achieves the middle area, it seeks for the last beacon. To complete the docking the robot has to move in its direction until it stops in correspondence to the station.

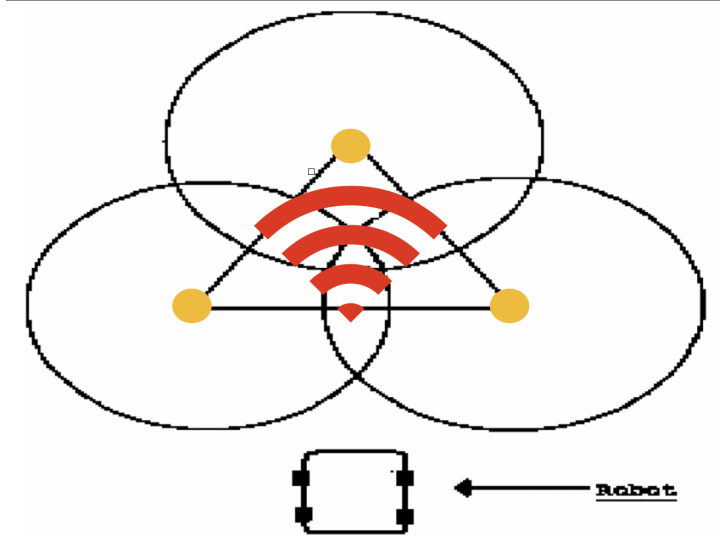


Figure 2.16: Bangalore robot docking station

This kind of approaching method is called "three-beacon convergence" and it is an extension of the homonym two beacon, a widespread method to follow a target.

Most of the cases we have seen so far have been carried on by fusing data coming from different sources. This is the key of successfully completing an autonomous charging operation.

In the present work, the robot employs a stereocamera (ZED 2 camera) for the vision recognition of an Augmented Reality tag (AR tag) placed in correspondence of the docking station.

This stereocamera implements other sensors besides the two cameras. Built-in IMU, accelerometer, gyroscope, barometer and magnetometer are some of the sensors inside the ZED 2 ecosystem, that makes this stereocamera a device full of potentials.

The tracking of the AR tag needs the odometer data to continuously know the position of the robot and to keep track of the path just completed. Then, the control algorithm consists to lead to zero the difference in the *pose* coordinates between tag and robot. The pose in ROS world indicates the position and orientation.

Chapter 3

Project implementation

This project addresses the problem of how to reach the station, as first, choosing the method of solution. This has been carrying on by taking into account the reliability and durability of the method, the quickness of charging and the effective cost of implementation and maintenance. Employing market-based sensors is the very first step for saving the cost and, as consequence, to make the robot competitive from the cost point of view.



Figure 3.1: Cal State Electrical department robot

3.1 Station detection

As it was mentioned so far, this project implements a vision-based recognition of an AR tag with the utilization of the ZED 2 stereocameras.



Figure 3.2: ZED 2 stereocamera by Stereolabs

The ZED 2 camera is characterized by the following specifications [28]:

Characteristic	Spec
Output Resolution	Side by Side 2x (2208x1242) @15fps 2x (1920x1080) @30fps
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)
Sensors	Gyroscope, Accelerometer, Magnetometer, Barometer, Temperature
Dimensions	175 x 30 x 33 mm

Table 3.1: Specification of ZED 2 stereocamera

The choice was influenced from the fact that the prototype robot of the Cal State Electrical department laboratory is already equipped with a Stereolabs ZED stereocamera as it is possible to notice from the *Figure 3.1*. Among others, it was the depth sensor responsible for the SLAM algorithm node.

Employing a vision-system algorithm, as in this case, needs a strategy to keep the contact between the camera and the QR code. If the robot loses the contact, it will result in a failing in the docking operation.

Therefore, the camera has always to have the AR tag in its field of view, otherwise the robot will never be capable to charge itself.

Moreover, during the movement and especially during a change of the orientation of the robot, the camera visualization could become blurry resulting in an impossibility to continuously detect the tag.

Thus, the test can be conducted by moving the robot inside a predefined environment and whenever it needs to be recharged, it approaches with a certain distance, close to the AR tag so that it can fulfill the detection and tracking operation in a completely safe manner.

3.1.1 Stereoscopy

The employment of a stereocamera allows to not implement another sensor for the extraction of the distance. With the term of stereo camera we refer to a camera device composed of two (or more) different lenses, one image sensor for each lens. By means of this set up, the camera may simulate the human binocular vision and eventually acquiring the ability to capture three-dimensional images.

The aim of such kind of cameras is to capture two images of the same scenario from two different perspectives and to obtain a new one capable to feel the depth sense. This 3D image sensing process is known as Stereoscopy.

The stereo cameras are classified according to the axis outgoing from the image sensors as cameras with parallel optical lens axes and the general stereo cameras with non-parallel optical axes. This thesis will only refer to the parallel axes cameras.

So, the parallel axes sensors are fixed at a constant distance called baseline. By comparing information about a scene from two vantage points of the stereo camera, 3D information can be extracted by examining the relative positions of the point in the left and right panels. Actually, the perception of the depth arises from the disparity of the projection into these two panels. The disparity is the distance between the projections of the objective point into the two panels. Taking advantage of this disparity is the trick to determine the depth, according to the triangular method.

The triangulation problem is in theory trivial when no source of errors and uncertainty are taken into consideration. To compute the depth of a point, so that you can describe an objective point P with 3 parameters, you may only need to know the X-coordinates

(or the Y-coordinates) of both the point P and the two projections in the image planes.

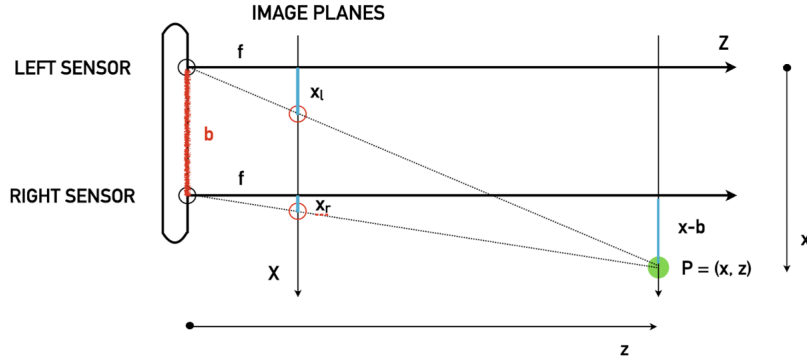


Figure 3.3: Triangulation problem explanation image

The two image planes lie in the same plane at a distance f (i.e. the focal length) from the camera. The parameters x_r , x_l are the distances from the respective axis to the actual projection of the point into the planes, while b is the measure of the baseline. The objective is to find z , the depth of the point P .

To address such an issue, the very first step is to tackle the proportions between the dimensions in *Figure 3.3*. It is possible to find out a system of equations between these dimensions by taking into account the relations between similar triangles according to the coordinate x .

$$z/f = x/x_l = x/(x_r - b)$$

The same assumptions and relations can be obtained by considering the y-coordinates. The coordinate y is entering the sheet:

$$z/f = y/y_l = y/y_r$$

At the end you will obtain the x and y coordinates of the objective point $P(x,y,z)$ as:

$$x = (x_l \cdot z)/f = b + (x_r \cdot z)/f$$

$$y = (y_l \cdot z)/f = (y_r \cdot z)/f$$

And at the end the coordinate z as:

$$\text{Depth } z = (f \cdot b)/(x_l - x_r) = (f \cdot b)/d$$

z is the depth, while the disparity d is the difference between the x coordinate of the left and the right projected points.

$$\text{Disparity } d = (f \cdot b)/z$$

3.1.2 Image processing

These are the main issues when dealing with the triangulation method of solution and have to be tackled in this order:

- *Extraction of intrinsic parameters:* The intrinsic parameters enter in the formulation of the 3D description of the objective point. Thus, it is necessary to know the exact values of both the baseline b and the focal length f to evaluate the distortion parameters introduced by the lens. Nevertheless, the common market stereo cameras are characterized by precise values for these parameters and they are equipped with camera calibration softwares, capable to adjust the sensors position.
- *Image Rectification:* The images taken from two camera sensors do not lie in the same panel. The purpose of the image rectification is to project the image panels into a unique plane, perpendicular to the cameras. This helps to simplify the problem of finding matching points between images (i.e. correspondence problem).
- *Correspondence problem:* The second issue arises from the inability to find the corresponding point (x_r, y_r) for each couple (x_l, y_l) , that is referred to as the correspondence problem.

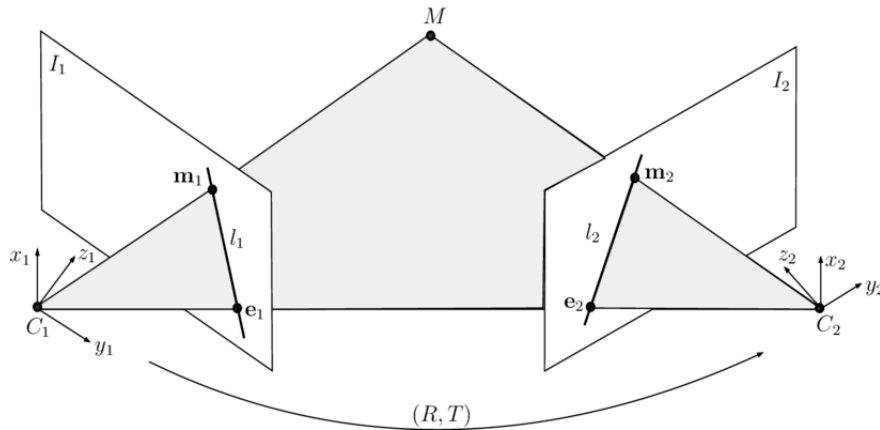


Figure 3.4: Epipolar geometry

Before going into the explanation of the resolution of such a problem, some definitions have to be clarified:

Epipole

The epipole is the point of intersection between the line joining the two cameras and the image planes. There are two epipoles (e_1 e_2 in the Figure 3.4) one per each camera.

Epipolar plane

The epipolar plane is the plane formed by the object point M and the camera optical points C_1 and C_2 .

Epipolar line

The epipolar line is the line created from the intersection between the line joining the two cameras and the image planes.

For each pixel of the original image the computer stereo vision algorithm determines the corresponding scene point's depth by first finding matching pixels (i.e. pixels showing the same scene point) in the other image and then applying triangulation to the found matches to determine their depth. Each pixel's match in another image can only be found on the epipolar line.

If two images are coplanar as in this case, then each pixel's epipolar line is horizontal and at the same vertical position as that pixel.

However, in general settings the epipolar lines are slanted. Image rectification warps both

images such that they appear as if they have been taken with only a horizontal displacement and as a consequence all epipolar lines are horizontal, which slightly simplifies the stereo matching process.

If the images to be rectified are taken from camera pairs without geometric distortion, this calculation can easily be made with a linear transformation [29].

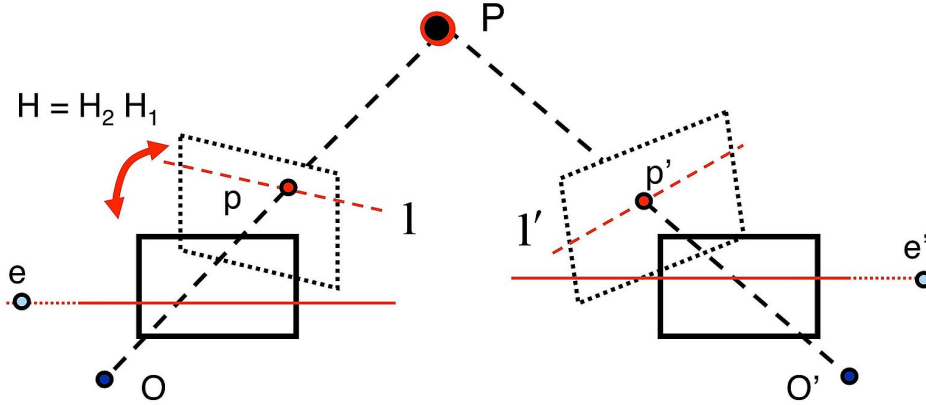


Figure 3.5: Model used for the image rectification example

In the image above, the model is based on a pair of images that observe a 3D point P , which corresponds to p and p' in the pixel coordinates of each image.

After having defined the epipole, the transformation matrices (H_1 , H_2) are computed as consequence, allowing to find out the respective projective planes. At this point, the planes finally lie in the same plane.

Now, let's find out how to solve the correspondence problem with the support of the *Figure 3.6*.

It is possible to notice, with reference to the image, that all the points along the projective line pointing to P in the left image plane map to a line in the right camera image plane. Looking inside the first line to look for matches, you will surely find that point P is just there. To follow the projective line of P is the same to follow the epipolar line.

Once the geometric problem has been illustrated, it is possible to proceed to the analytical computation of these dimensions. But first of all, it is worth introducing the concept of *Fundamental matrix* F , used to transform the geometric analysis into algebraic form [30]. It is defined as:

$$x \cdot F \cdot x' = 0$$

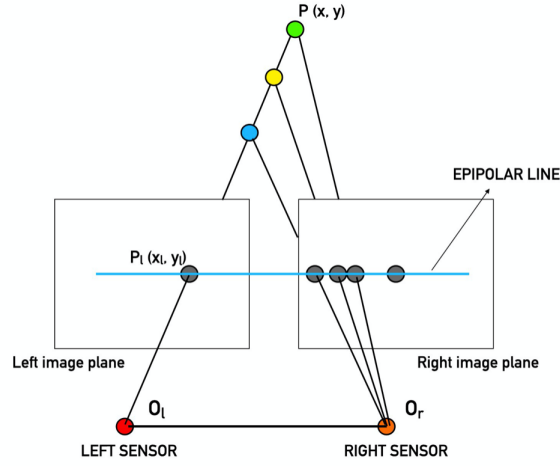


Figure 3.6: Triangulation problem explanation image

Where x and x' are matching points of the two panels of the same objective points.
 Relation between epipolar lines and epipoles.

Epipolar lines:

$$l' = F \cdot x$$

$$l = F^T \cdot x'$$

Epipoles:

$$F \cdot e = 0$$

$$F^T \cdot e' = 0$$

(e, l) and (e', l') are the epipoles and epipolar lines of respectively left and right camera sensors.

With the utilization of a sufficient number of matching points, it is possible to arrange

an equation for the computation of F . By writing the points as:

$$\mathbf{x}_i = (x, y, 1)^T \text{ and } \mathbf{x}_i' = (x', y', 1)^T$$

Prior to proceed with the formulation of F , that F represents a 3x3 mapping matrix of a two-dimensional to one-dimension transformations, hence it must have rank equal or lower than 2 ($\det(F) = 0$), generally it is 2. Thus, the transformation matrix is a 3 by 3 homogeneous matrix, hence it has eight independent ratios; however, thanks to this property it is characterized by seven degree of freedom.

With this information about F , it is possible to derive an equation with the unknown coefficients of F , by means of the *equation ..* and the description of the matching points x and x' .

$$x'F^T x = x'x f_{11} + x'y f_{12} + x' f_{13} + y'x f_{21} + y'y f_{22} + y' f_{23} + x f_{31} + y f_{32} + f_{33} = 0$$

$$Af = \begin{bmatrix} x_1'x_1 & x_1'y_1 & x_1' & y_1'x_1 & y_1'y_1 & y_1' & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n'x_n & x_n'y_n & x_n' & y_n'x_n & y_n'y_n & y_n' & x_n & y_n & 1 \end{bmatrix} f = 0$$

At this point, there are few cases of solution for the transformation matrix depending on the rank of A :

- rank = 8 : The system in this case can be easily solved, since there is a unique solution.
- rank = 9 : This case happens when the system and hence the matrix is subject to noise. The system is overdetermined and can be solved by the mean of Least square approach to approximate its solution minimizing the sum of the squares of the residuals. This kind of problem is called *8-point algorithm*.
- rank = 7 : The problem is still feasible thanks to the property of transformation matrix determinant mentioned before and the approach is called *7-point correspondences*.

The process described so far is executed by the ZED stereocamera whenever it extracts and publishes the information about the depth. With this information, the SDK tool the camera is provided can build a depth map of the environment.

3.1.3 AR tag

As regards the AR tag, it is easy to create, implement (sheet of paper) and they can be quickly detected from cameras.

It can store information about the localization, description of objects, obstacles and instructions and also some information about the docking station (e.g. the identification number of the station in case of multiple stations environment or the orientation).

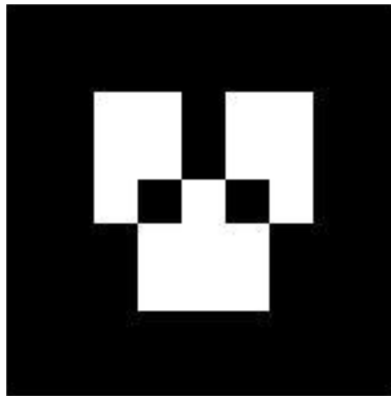


Figure 3.7: An example of Augmented reality (AR) tag

In addition, there is an high availability in terms of libraries about detection of AR tag and they are characterized by a good speed of reading.

For the implementation in this project it is a good compromise between costs and benefits. Moreover, considering the ultimate working environment of the project that can be an airport, an hotel or an office, the good lighting these latter are provided allows the robot to easily recognize the tag.

3.2 Tracking algorithm

The core of this work is the docking algorithm, that is the actual tracking planning for the autonomous charging operation. Taking time for this part allows to design the trajectory the robot should execute in a complete safe manner, avoiding collision with the station. Moreover, to design a path that is as smooth as possible preserves the camera to not lose the contact with the AR tag, resulting in a fail of the operation.

By looking at the docking operation state of art and in particular at the docking algorithm of the vacuum robot Roomba iRobot [31], a good operation could be to approach first to the perpendicular line to the AR tag in order to face it and then to slowly approach the AR tag in order to avoid collisions or accidents.

DOCKING STATION

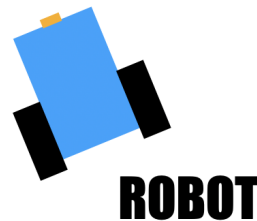


Figure 3.8: An hypothetical initial condition of docking operation

The path to be designed has very simple initial conditions. The robot, whenever the battery needs to be recharged or the tasks assignment system decides that the robot has to reach the station, is driven in a position close to the station through the path planning algorithm it is provided. In the final position of the robot, the camera must have the AR

tag in its visualization.

Someone might wonder why the robot can not be driven towards the station with a simple path planning algorithm. The answer is that, unfortunately, the docking operation requires an higher level of precision during its execution. The further away the robot docks from the point where the wireless station has the max charging effect, the less is the efficiency of the recharge.

An hypothetical initial condition is illustrated in *Figure 3.8*. The robot points towards the station and its distance is in the range of 2/3 meters.

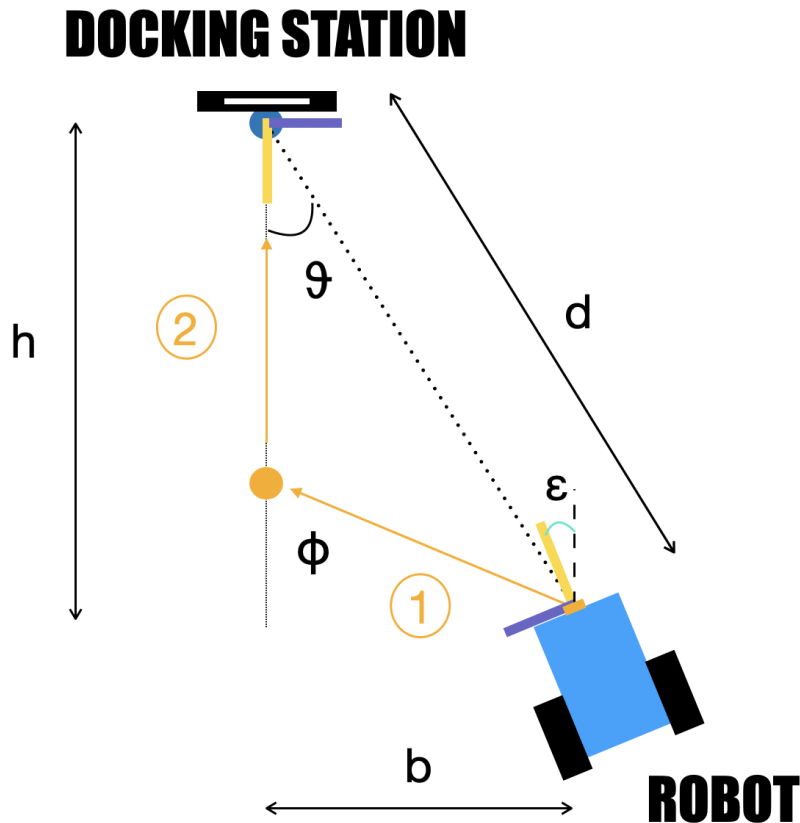


Figure 3.9: Parameters description of the first phase docking strategy

To simplify the operation, the designed strategy is divided in two phases. In the first phase the robot should approach the continuous line, as one can notice from the *Figure 3.9*. This line represents the perpendicular line to the AR tag that passes through the tag itself. The best result should be that the robot reaches the orange point the orange

arrow that starts from the robot.

The second phase is just a straight path from the first phase final position to the station. Actually, this last operation seems to be a very simple operation, but it is true only if the robot accomplishes in good manner the first phase, that the robot should complete pointing the station in the final pose. Moreover, this phase is more risky than the first from the safety point of view, since the robot approaching the station can fail the operation and collide against the station.

With reference to the *Figure 3.9*, d represents the dimension of the segment that connects the robot and the tag (i.e. *distance*), while h and b are respectively the vertical and horizontal components of the distance with regard to the station reference system. The reference frames (yellow and violet lines) are oriented pointing towards the direction of the wheels with reference to the robot and in a perpendicular manner for the AR tag (i.e. white and black rectangular).

The parameter θ is the angle describing the relation between h and b . It is referred only to the position parameters, it has nothing to do with the orientation. This work will refer to this parameter as *angle*. Instead of ϵ that is the angle representing the difference in orientation between the robot and the AR tag. In this case, this dimension has nothing to do with the position of the robot and it depends only on the angle around itself. At the end ϕ is the orientation the robot has to have to reach the perpendicular line in correspondence to the orange point.

In order to find the value of θ and of *distance*, it is necessary to extract the dimension of both h and b and it is computed with a simple trigonometry formulation:

$$d = \sqrt{h^2 + b^2}$$

$$\theta = \text{atan}(h/b)$$

However, to extract the information about h and b starting from the pose messages, it is necessary to have a feedback on the odometry data. That's why the camera provides pose data according to its reference frame and not to the one of the station. Let's take as example an orientation for the robot such that it exactly points the station. In this case, the vertical component according to the camera reference frame is practically equal to the distance, while the horizontal one is null. The odometry data keeps track of the orientation of the robot updating it while is moving. Further this work will show how the odometry data are extracted in the simulation stage and with the prototype robot and used in the algorithm.

Thus, the actual θ is composed of two components, one coming from the camera data and one from the odometry data:

$$\epsilon_{\text{real}} = \epsilon_{\text{camera}} + \epsilon_{\text{odometry}}$$

The goal of the first phase is to null the angle θ and to reach the perpendicular line so that the robot can have enough gap with the docking station in preparation to the second operation. The optimal condition is to reach an orientation close to ϕ . In this way the robot has also space to rotate itself and to point towards the station before the beginning of the second phase.

Instead, the purpose of the second phase is to lead to zero both the *distance* and the *orientation* between the robot and the AR tag so to approach the station in the right condition.

As it was mentioned before, it is necessary to accomplish the first phase as soon as possible to reach the perpendicular line having enough space for the second phase.

However, the robot isn't capable to follow a straight path like the orange arrow, since the robot executes a curve just to rotate itself to adjust its orientation according to ϕ . This causes an extension to the path that can jeopardize the success of the operation.

As far as the orientation ϕ is concerned, this parameter is computed based on the information of the distance and the angle.

The purpose of this relation is to keep the tag inside the visualization of the camera. The higher the distance, the higher will be the orientation ϕ the robot should use to reach in the second phase. This because the camera has a wider visualization and the robot can turn left or right with an higher angle. As regards the angle, the orientation depends on the sign with respect the tag. If the robot is on the right hand side of the station, it can turn clockwise (i.e. negative sign) with an higher angle, but it can turn counterclockwise (i.e. positive sign) with a lower angle because the tag is already in the right half part of the visualization.

At the end, the orientation has to depend also on a minimal orientation that should be taken with zero distance and zero angle. However, since zero distance means that the camera is attached to the paper representing the tag resulting in an impossibility to visualize the tag itself, the minimum distance the value of ϕ_{min} is represented by the minimal distance such that the camera is capable to visualize the whole tag and recognize it.

To sum up, the orientation is computed by adding three contributions, the minimal ori-

entation that sums the distance contribution and at last the angle contribution.

$$Phi = Phi_{min} + (Phi_{max} - Phi_{min}) \cdot d / d_{max} + Phi_{ang} \cdot \theta$$

Once ϕ is computed, the first thing the robot has to execute is to change its orientation. The wheels should be actuated to rotate the robot such that it reaches the value of ϕ . As long as the absolute value of the robot orientation is lower than ϕ , the robot turns to decrease the gap.

On the contrary, if the orientation is higher, the robot is instructed to invert its orientation and so to rotate to the other side.

Once the orientation is close to ϕ , the robot will oscillate about this value turning right and left so to keep itself close to this value. If the chassis exceeds this value, it has to recover turning in the opposite direction. This oscillation lasts until the *angle* θ becomes lower than a fairly small value (θ_{min}). At this point the robot should turn its direction of motion to point the station in order to get ready for the second phase.

The operation of rotation that brings the orientation to zero and the robot pointing the station is a delicate movement. The control system is in charge of execute it as smooth as possible so to not turn too much and make the camera lose the contact with the tag. If this process is well completed, the absolute value (so both if the robots came from the right or from the left) of the *angle* will be finally lower than the minimum and the robot will find itself along the perpendicular line to the station.

Now, the parameters to check are the distance and the orientation. As long as the distance has a value lower than the distance between tag and docking point (d_{min}), the robot has to execute the operation. Whenever the orientation exceeds a tolerance value (ϵ_{min}), the wheels have to be controlled to re-approach it to 0.

The algorithm can be summarized with the following loop steps:

- **Starting:** Approach the robot close to the station so the camera can visualize the tag.
- **Data manipulation:** Read pose value of the robot with respect the AR tag and manipulate this data to extract the angle θ , the orientation ϵ and the desired orientation of the first phase ϕ .

- **First phase:** Control the wheels to reach ϕ as orientation of the robot. Correct the *orientation* until the *angle* becomes lower than θ_{\min} .
- **Preparation for the second phase:** Rotate the robot as long as it directly points to the tag and the orientation goes to zero.
- **Second phase:** Follow straight path by checking *distance* and *orientation* don't exceed their bounds until the robot reach the station and the docking operation is completed.

The flow chart below can be seen as the input for the software development:

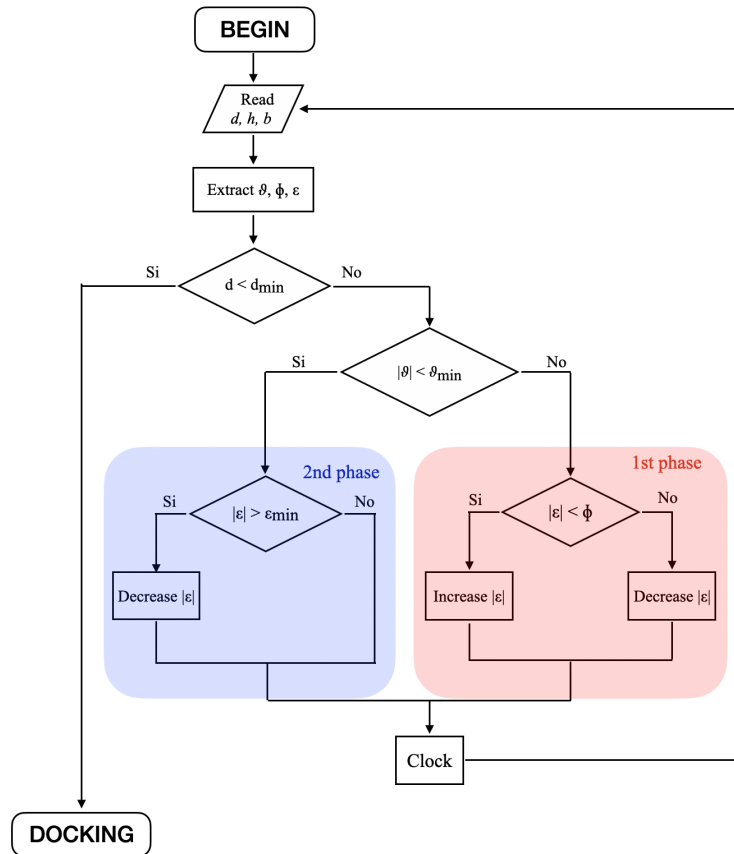


Figure 3.10: The flow chart explaining the docking algorithm

Now let's have a look on what software tools that have been utilized and how the algorithm has been developed by means of these tools.

3.3 Software tools

This section presents the different software tools and programs employed in this project. The Robotics Operating System (ROS) is a distributed framework of processes (i.e. nodes) that enables executables to be individually designed and loosely coupled at runtime. It represents the development system where nearly the entire algorithm is developed. This includes the function for the extraction and manipulation of sensors data, the program that makes the robot reproduce the docking operation explained so far, the node dedicated to the development of the motors/wheels control system.

The simulation part is taken inside the Gazebo environment. Gazebo is a 3D open-source robotics simulator. This software allows to simulate the behavior of a robotic system by adding the functionalities the robot needs to accomplish its tasks. After the creation of the URDF file representing the body of the robot, by adding the plugins that reproduce the main functionalities of the robot, the simulation on Gazebo allows to practically see the results of your algorithm in a broadly manner. This is a fundamental step, it is risk free and the time that takes this part is not comparable with the time took to change the algorithm after the implementation on the actual robot.

Further, the project will focus on the construction of a custom robot. The velocity messages sent to the robot in the simulation environment, resulting from the docking algorithm, have been transformed in voltage signals to be sent to the motors. Arduino UNO compatible platform is in charge of transferring voltage information and to transform the analog signals to digital signals. It takes advantage of the PWM signals properties to allow the variation of the velocity in a certain range.

Arduino platforms exploit own IDE (i.e. Arduino IDE) to design the functions. The function is written in this software application and then it is uploaded on the compatible board.

3.3.1 ROS framework

ROS is becoming the standard framework in the projects regarding the robotics fields. To better explain the potential of this development system it is possible to take advantage of the definition on its introductory website [32].

"ROS is an open-source, meta-operating system for robots. 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.”

The Robotics Operating System was born in 2007 thanks to two *Stanford University* PhD students, Eric Berger and Keenan Wyronek. The two students noticed that most of their colleagues were not able to implement robotics system of diverse nature, since a software developer might not have the hardware knowledge required to abstract the functionalities of electronic systems. They built a robot prototype PR1 and began to work on the software from it, while receiving fundings from the University.

The turning point occurred during the meeting with Scott Hassan, the founder of Willow Garage, a technology incubator, who shared Berger and Wyronek’s vision of a “Linux for robotics” and invited them to come and work in the incubator. The first commit of ROS code was made to SourceForge on November 2007.



Figure 3.11: ROS logo

Talking about ROS can take more than just one paragraph. However, here we will limit ourself to underline its hierarchical system and the main features.

As it was mentioned before, ROS development system allows to design executables as an apart system. The *Nodes* that shared the same purpose can be grouped into *Packages* and *Stacks* that can be easily reused, shared and distributed. The design from the filesystem level to the distributing level, enables to make decisions independent from the actual development and implementation, but everything can be brought together with ROS infrastructure tools.

The ROS Computational Graph is the peer-to-peer connections network of ROS processes that exchange data together. Its basic concepts are [33]:

- **Nodes:** It is a process executing computations. Usually, a robotic application has more than one node. For example, in this project, as regards the simulation stage, there is one node responsible for the camera visualization extraction and recognition of the AR tag, one for the manipulation of the pose data and another for the docking operation.

- **Master:** The ROS Master provides naming and registration services to every node in the ROS system. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Service:** It allows to store, update and visualize parameters of the nodes.
- **Messages:** Nodes communicate each other through the messages. A message is represented by a data structured. A ROS developer either can use standard messages or he can create his own custom message depending on the application.
- **Topics:** The topic is the name identifying the content of the message. Nodes publish messages on a given topic. A node that needs a certain data subscribes to the corresponding topic. There may be multiple concurrent publishers and subscribers for a single topic, as well as a single node can subscribe and publish also to more than one topic.
- **Services:** This concept implements request / reply interactions communication. It is defined by a pair of messages, one for the request and one for the reply. A providing node offers a service and a client uses this service by sending the request message and waiting for the reply.
- **Bags:** Service for saving and playing back ROS message data. They result to be fundamental both for storing sensors data that are difficult to collect and for having data at hand to test.

Three kinds of software can be distinguished in the ROS Ecosystem [34]:

- **Independent tools:** tools that don't depend from languages and platform/application.
- **ROS client libraries:** It provides a collection of code that lightens the developer job. It enables language-specific programmers to interacts with ROS Topics, Services, and Parameters. The main libraries of ROS that are:
 1. roscpp
 2. rospy
 3. roslisp

- **Application specific:** Application-specific packages which makes use of ROS client libraries.

The big advantage of ROS with respect the other robotics is the reusability property of the software.

Compared to the other robotics software system, ROS demonstrates a greater power and flexibility come at the cost of greater complexity. Even if in the last years ROS is getting a design path easier for the developers and especially the first-experience ones, there is still a significant learning curve.

It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. Rviz and roslaunch are tools widely employed in this project that deserve to be mentioned. The former is a 3D viewer for ROS, while the latter is a tool for easily launching multiple nodes in one shot.

Furthermore, ROS offers a complete integration of simulation platform such as Gazebo or Stage. They are well-supported and widely used in the ROS community.

An interesting project that was born 5 years ago is ROS 2.0. It addresses the lack of a support for real-time systems in ROS and adds documentation for real-time code and embedded hardware. In this work, we will refer only on ROS.

3.3.2 Gazebo environment

Robot simulation is an essential tool in the development of a product. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.

Gazebo was born as a part of *Player Project*, a project to create free software for research on robotics. Gazebo was one of the three main components together with *Player Server* and *Stage*. Then, in 2011 it became independent and supported by the same company of ROS, Willow Garage. Moreover, currently, it has been used as simulation environment in plenty of competitions such like DARPA Robotics Challenge or NASA Space Robotics Challenge.

A part from the of the main functionalities, this simulator is capable to provide a realistic rendering of the environment including textures, lightning and objects with high-quality visualizations. It can model sensors perceiving the actual environment created on Gazebo

where the robot is placed, such as laser range finders, cameras, Kinect style sensors [35].



Figure 3.12: Gazebo logo

As far as the design on Gazebo is concerned, the elements of the robot model is described in the Unified Robotic Description Format (URDF), which is an XML file format used in ROS. XML is a markup language that means it is a system for annotating a document in a way that is syntactically distinguishable from the text. This kind of file is full of tags that indicate the different document parts.

URDF can only specify the kinematic and dynamic properties of a single robot. Neither joint loops nor friction and other leaks can be specified nor lights or other objects other than the robot.

However, the URDF syntax makes heavily use of XML attributes, which makes URDF more inflexible. To address such an issue, a new format (Simulation Description Format - *SDF*) was created so to provide a description for everything from the world level down to the robot level. Thus, the URDF file has to include additional elements on code to be used in Gazebo. The required element to be added is an `< inertia >` tag within each `< link >` element of the robot file. The optional elements concern the `< gazebo >` tag and has to be inserted every time, for example, a sensor or actuator plugin is to be counted. Then, Gazebo is in charge to transform URDF file in SDF format thanks to this information.

Thus the URDF file of a robotic system is composed by consecutive links and joints. The link is the visual element, a wheel for example. It is characterized by physical properties and obstruction dimension. Through the tag `< material >` it is possible to even specify the color of each element to be visualized in Gazebo.

Each link can be a simple element like a box, a sphere or a cylinder. Then these shapes can be set up with the desired dimensions and even connected together.

To sum up, for each link, we need three main tags and relative subtags:

- **visual:**

- *< geometry >* : to set the geometry of the object to be visualized.
- *< origin >* : to set the origin of the link.
- *< material >* : to set the color of the

- **collision:**

- *< geometry >* : to set the footprint geometry area.
- *< origin >* : to set the origin of the footprint geometry area.

- **inertial:**

- *< mass >* : to set the mass of the object.
- *< inertia >* : to set the inertia along the reference frame axis.

For sure there are other tags that can be added, but these are sufficient for the present work.

The joint element connects two consecutive links. These elements are invisible to the user interface and have to be designed specifying the two connected links (parent and child links). The joint element can be created in four different types: fixed (or inflexible), continuous, revolute and prismatic.

In addition to the previous information, in order to make these elements movable, the origin of each element has to be specified and for the joint also the range of action, the maximum velocity and the applied torque/force.

- *< axis >* : to set the axis of action.
- *< limit_effort >*: to set the limits relative to the velocity, displacement and strength.
- *< origin >*: to set the origin.
- *< parent_link >*: to indicate the link which the joint should start from.
- *< child_link >*: to indicate the link which the joint should connect with.

Gazebo supports several different plugins and all of them is supported in ROS, but only ModelPlugins, SensorsPlugins and VisualPlugins can be referenced through a URDF file [36]:

- *ModelPlugins*: providing access to the physics of the model.
- *SensorsPlugins*: providing an interface to the sensors.
- *VisualPlugins*: provide access to the rendering of the model.

There is a pre-existing plugin for each of the most popular sensors on the market, based on the analysis carried out in *Chapter 2*: camera plugin, laser plugin, IMU plugin, GPS plugin, etc. These are components of the *gazebo_plugins* package.

In addition to the plugins explained above, there are also a number of 3rd party Gazebo-ROS plugins that the community shared. If one of them is considered enough useful and generic for the framework, it can be added to the *gazebo_plugins* package.

Moreover, the Gazebo community offers a complete support for the creation of a custom plugin depending on the user necessities.

3.3.3 Arduino world

Arduino is an open-source hardware and software company that designs, produces and sells microcontrollers employed to build digital system. On the market, there are plenty of Arduino boards including microprocessors and controllers. These boards are equipped with sets of digital and analog input/output (I/O) and serial communication interfaces such as Universal Serial Bus (USB) pins [37].

Arduino project was born at the Ivrea Interaction Design Institute as an easy and inexpensive tool for fast prototyping, aimed at students without a background in electronics and programming. Then, when its community got larger, Arduino board has been used as the core of complex scientific systems [38].

The Arduino systems are endowed with an easy-to-use software, tailored for beginners and an high hardware/software flexibility, perfect for the advanced users.

The way in which an Arduino platform can be programmed is with utilization of Arduino programming language, based on *Wiring* [39]. It is a very simple open source framework



Figure 3.13: Arduino logo

for microcontrollers, based on C/C++ system, characterized by limited functions, variables and structures [40].

The Arduino project provides an integrated development environment (IDE) and a command line tool (arduino-cli) as compilers. Arduino IDE is a cross-platform flexible software sketchbook that runs on both Windows, Macintosh OS X and Linux operating systems. It is based on *Programming* [41] and it provides simple one-click mechanisms to compile and upload programs to an Arduino board.

It is worth noticing that apart from the open-source characteristics, what gives flexibility to this apparatus with respect to other systems is the high compatibility with C++ libraries and the possibility to directly add to our program AVR C code, the C language designed for microcontrollers on which it is based.

Chapter 4

Simulation stage

In the present chapter this work is delving on the simulation part. This has been carrying on by means of a two wheel robot model, designed to be compatible with ROS framework and to work on the Gazebo environment.

The first part deals with the design of the robot and the implementation of its functionalities on Gazebo. In particular, the analysis will address the choice of the dimensions of the robot elements together with the application of the Gazebo plugins for abstracting the features of camera and differential drive capability.

While the second half is pointing at the explanation of the various steps made for developing the software of the docking algorithm. The main theme of this section is the issue connected with the extraction of pose parameters along with their conversion in data of *distance*, *angle* and *orientation*.

At the end the thesis will focus on the translation of the docking algorithm as ROS nodes in this simulation stage.

4.1 Robot on Gazebo

The matter of this section is the building up of the robot, part by part, with the purpose of testing the algorithm in a first step of simulation fashion.

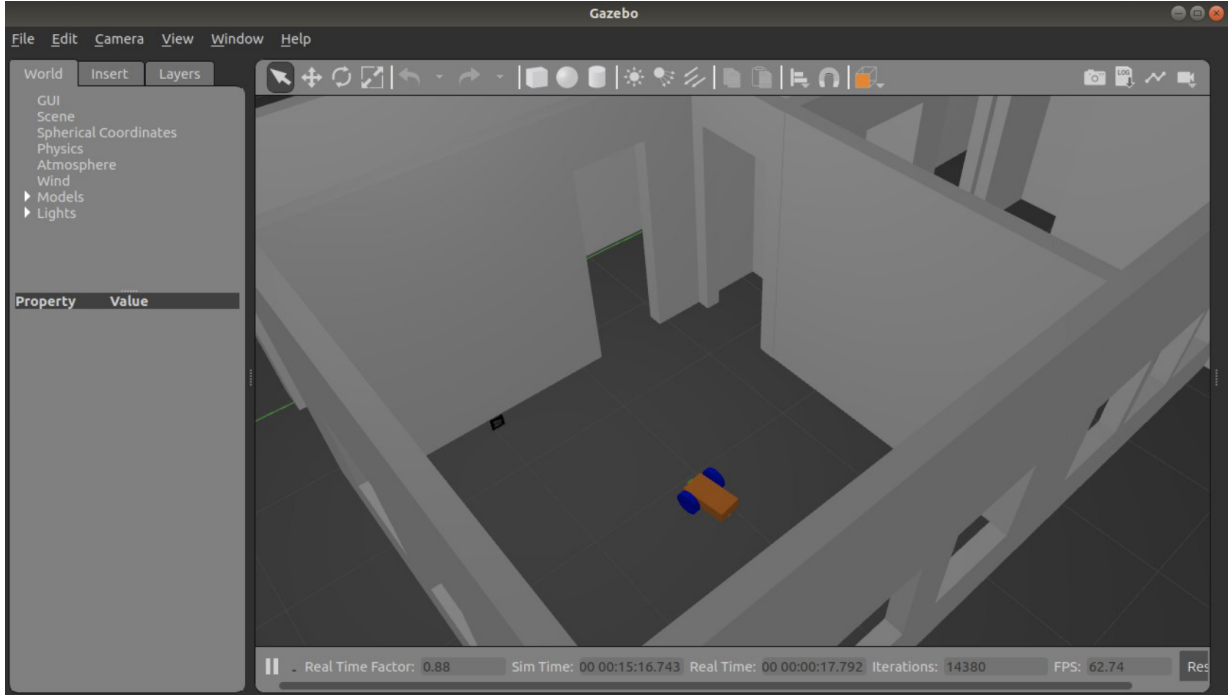


Figure 4.1: Initial condition

Figure 4.1 represents an example of initial phase of the docking operation simulated on Gazebo.

The picture has been taken including the AR tag, placed in correspondence to the wall. Actually, the tag is represented as a black box modeled using open-source rendering software Blender following a simple guide [42]. The creation of a black box is made with specific dimension such as to make the cube become a parallelepiped compatible with the dimensions of the room and the robot. The application of the tag as texture on each face of the box element follows it.

Now let's go more in deep on the design of each brick of the system and even on the coding back-part.

4.1.1 Robot body

A model, as in its nature, has to reflect the relevant selected elements of the original system properties in order to simplify the way in which the robot is controlled and the weight of data to be transferred.

The least number of link elements that allow to simulate the robot in a fairly manner is represented by:

- **chassis:** It can be representable as a simple box.
- **motorized wheels:** The cylinder is the shape to give to the left and right wheels.
- **caster wheel:** Being an idle wheel, a sphere is chosen to allow this wheel to move everywhere without obstructing the movement of the robot.

In the design of the URDF file the link and joint examples in *Section 3.3.2* was taking as reference. In addition to the above mentioned links and their corresponding joints, to complete the model of the robot, it is needed a link to implement the camera plugin on the robot. In this regard, a little box representing the camera is added to the robot URDF file on the front of the robot, so that it is possible to attach the camera sensor there.

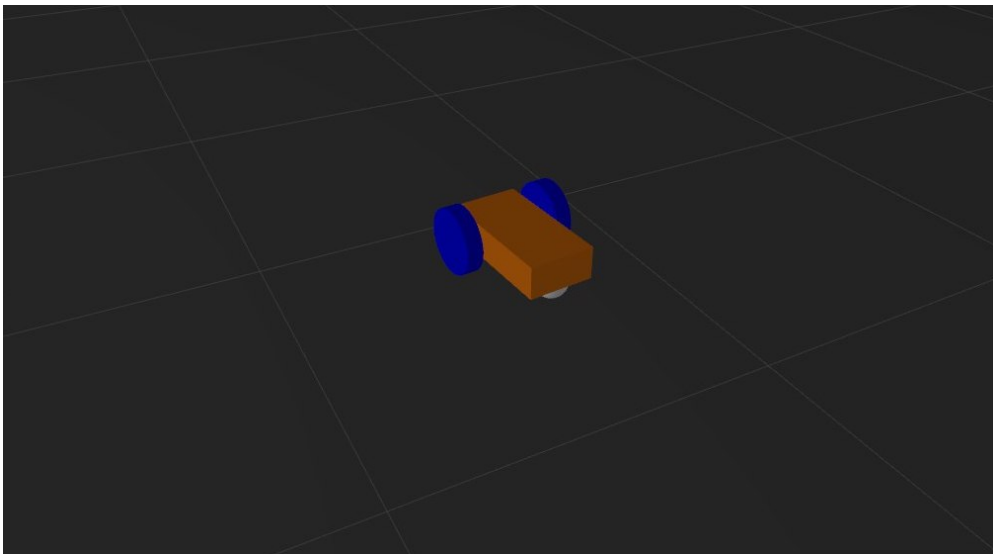


Figure 4.2: Body of the robot on Gazebo

At this point, to get the robot be interactive, we need to specify two things: Plugins and Transmissions. By the way, the differential drive plug-in and the camera plug-in are the required complements for this simulation.

4.1.2 Differential drive plugin

The differential drive plugin is one of the two main features that has to be implemented to test the developed algorithm. Such a plugin allows to control the wheel links of the robot body. In particular, by the mean of this component, the robot in Gazebo has the capability to subscribe the topic related to the velocity.

This plugin is made of different tags within the `< gazebo >` one:

- `< left_joint >` and `< right_joint >`: to indicate the robot left and right joint name.
- `< wheel_separation >`: to indicate the separation gap between the wheels.
- `< wheel_diameter >`: to indicate the wheel diameter.
- `< command_topic >`: to indicate the subscribed topic containing the speeds.
- `< odometry_topic >`: to indicate the topic name to be published containig the odometry data.
- `< odometry_frame >`: to indicate the odometry frame to attach the messages.
- `< robot_base_frame >`: to indicate the principal robot reference frame.

These are the main tags that should be added as plugin on the URDF file. Moreover, the file has to include the `libgazebo_ros_diff_drive.so`, which is a C++ pre-designed function of the package. Each tag points to a parameter of this function and it has to be set in order to make the plugin compatible with the chassis and coherent with the received velocity signal.

4.1.3 Camera plugin

To reproduce the stereocamera we take advantage of the camera plugin. Unfortunately, we are able to provide only one sensor to the simulation, due to the limitations of the Gazebo functionalities.

However, the robot is even capable to recognize the AR tag and to compute the parameters. These last ones are extracted taking advantage of the information about the real dimension of tag and the length of the sides of the tag in the camera visualization.

For sure, the implementation of this plugin doesn't allow to reproduce the real behavior of the ZED stereocamera and the precision of the system is lowered. Nevertheless, in this phase such kind of arrangement is sufficient for testing the docking operation algorithm. The camera plugin example provided in the Gazebo documentation [36] contains the tags specifying the topic and frame connected to the plugin and the characteristics of the camera. About the latter, among other we have:

- *< image >*
 - *< width >* and *< height >*: to specify the visualization dimensions.
 - *< format >*: to specify the image format.
- *< noise >*
 - *< type >*: to specify the kind of noise (e.g. gaussian).
 - *< mean >* and *< stddev >*: to specify mean value and standard deviation of noise.

The parameters has been set up according to the own physical camera hardware, as well as the reference frame and the topic names, that have been filled with the correspondent addresses.

It is worth specifying that the camera visualization can be only a square. So the width and the height of the camera visualization will have the same value. This doesn't reflect the stereocamera visualization and it represents another limitation of this kind of plugin.

4.1.4 Environment

This step has been taken only for an aesthetic purpose, just to give to the simulation a friendly working environment.

Lemon Tree House is the house model represented in *Figure 4.3*. The model is trying to recreate the structure of the apartment rented by the author in Los Angeles during the 5 months experience of collaboration with the California State University.

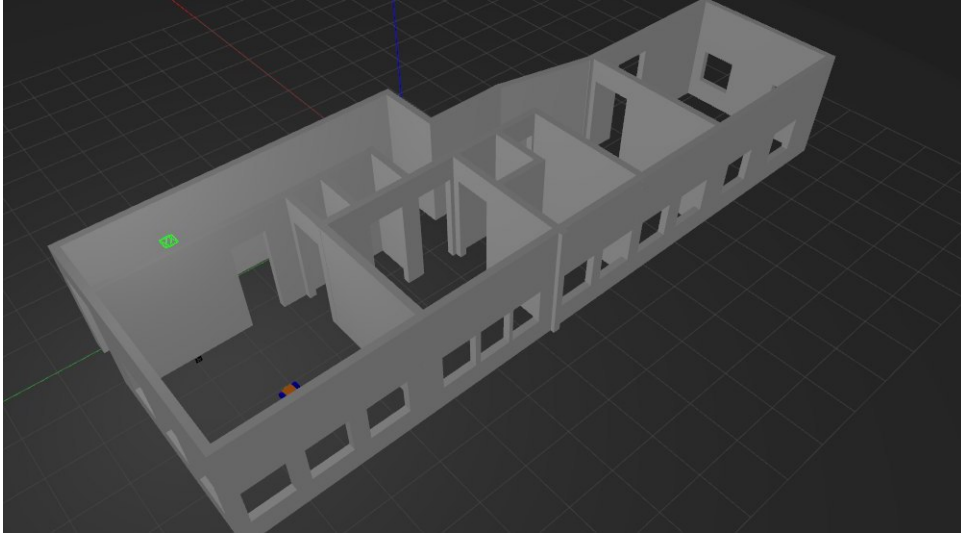


Figure 4.3: Lemon Tree House: model of the Los Angeles author apartment

This model has been built by the mean of the Building Editor tool of Gazebo [43]. The design with this tool starts by putting on the environment the walls, that represents the main feature of this tool. Then, the walls can be modified by adding *windows*, *door* or even *stairs* elements according to the plant we would give to the model. Moreover, another functionality this software is capable to provide is to chose the color and the texture of the walls.

Creating a building on Gazebo may improve the simulation from a beauty point of view and when necessary also the functionality of the simulation when, for example, it is necessary to recognize obstacles or a map. However, adding too many elements on the Gazebo environment heavily increases the computational effort of the Computer utilized for the simulation resulting in lags during the operations.

At this point, the robot is provided with the camera sensor, the differential drive capability and a setting to move inside. The information that it misses is the feedback on the movement and the robot pose, the odometry. In the simulation, Gazebo offers two nodes, *robot_state_publisher* and *joint_state_publisher*. The execution of these nodes allow to publish the state of the robot as a ROS message. The former takes the joint angles of the robot as input and publishes the robot links 3D poses, using a kinematic tree model of the robot [44]. The latter finds all of the non-fixed joints and publishes a JointState message with all those joints defined [45]. At the end, these two packages allow Gazebo to publish the odometer messages.

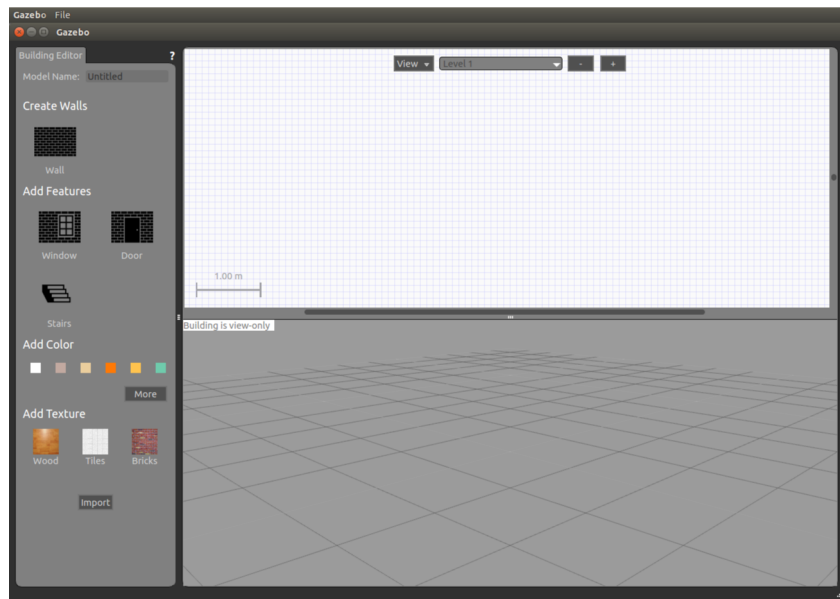


Figure 4.4: Gazebo Building Editor main interface

4.2 Software development

This section is aimed at explaining how the sensors data are extracted and manipulated and how the algorithm is developed and the robot instructed to complete the docking operation.

The first step for the software development according to the docking algorithm is to know the dimensions relative to the pose difference between camera and AR tag. This has been passed through the recognition of the tag on the Rviz visualization by the mean of the AR detector library of ROS. This package is in charge of detect and track the tag, while providing the information the algorithm needs. By continuously detecting and tracking the tag, the algorithm can keep the trace of the distance, angle and orientation.

Then, the algorithm requires to manipulate this information in order to compute the desired orientation ϕ , useful for the first phase of the strategy.

At this point, the algorithm is capable to calculate and publish the velocity signals on ROS according to the two phases of the strategy.

It is worth noticing that the main goal of the first phase is to approach the perpendicular line to the tag as soon as possible and with the maximum orientation the robot is capable to keep without losing the eye contact with the tag.

While the second phase has to be completed keeping constant null the orientation and angle, in the most safe manner.

4.2.1 Extraction of parameters

ar_track_alvar is the ROS main package among those that deal with the Augmented Reality tags [46]. The main functionalities of this package include the generation of AR tags setting the size and the data encoding, the identification and tracking of individual or bundles of tags and the calculation of the spatial relationship between bundles and camera. This latter functionality works publishing ROS messages about the pose difference between the camera and the tag.

Once the package recognizes the tag, it puts a mark on the Rviz camera visualization, as one can notice from the camera visualization on the left hand side of the *Figure 4.5*.

As a consequence, Rviz makes the tag appear on its map that is visualized on the right side of the same picture.

The tag is also provided with a reference frame in the same map, as well as the robot and the camera (i.e. *chassis* and *camera_link_optical*) and the map itself (i.e. *odom*). The package communicates the tag reference frame pose information to the package respon-

sible of the transformation between reference frames, in order to provide the geometry information.

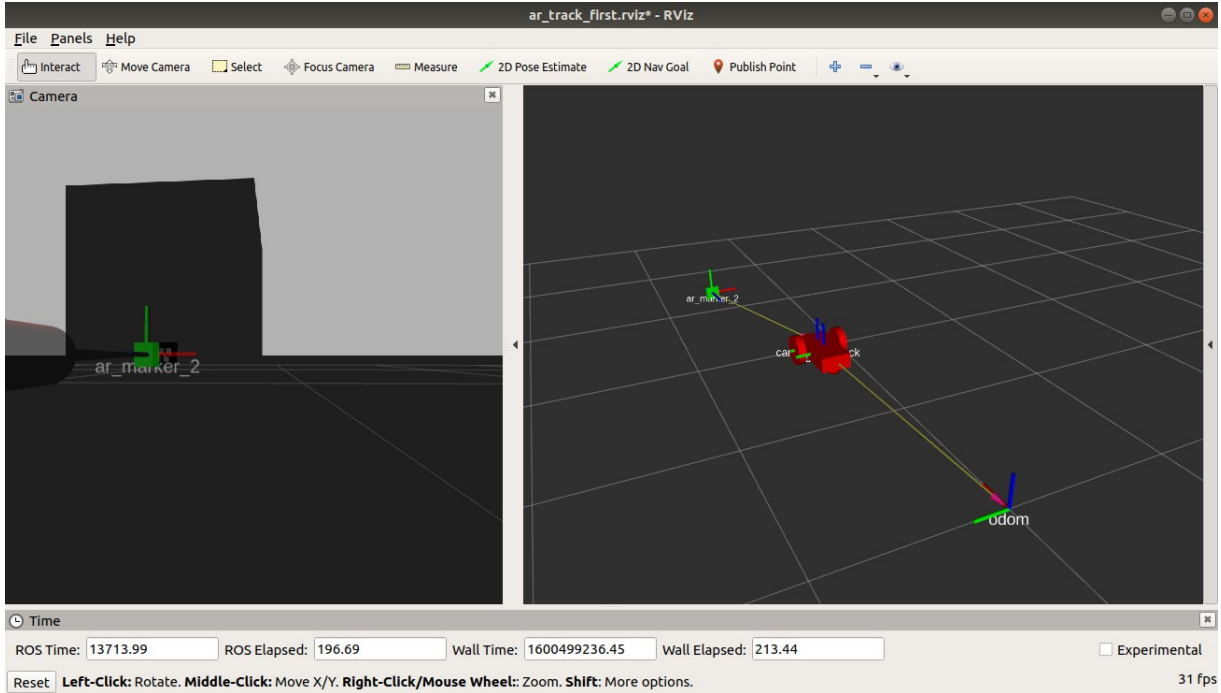


Figure 4.5: Rviz interface showing the camera visualization and the robot and tag on the map with the relative reference frames

However, there is an issue caused by the mismatch between data extracted from the tag tracking package and the dimension the algorithm needs. As one can notice while using *ar_track_alvar*, the package expresses the tag pose with the *local* reference frame of the robot, that moves with the robot. Instead of the algorithm required information, that has to be provided with the *global* reference frame, fixed with the map.

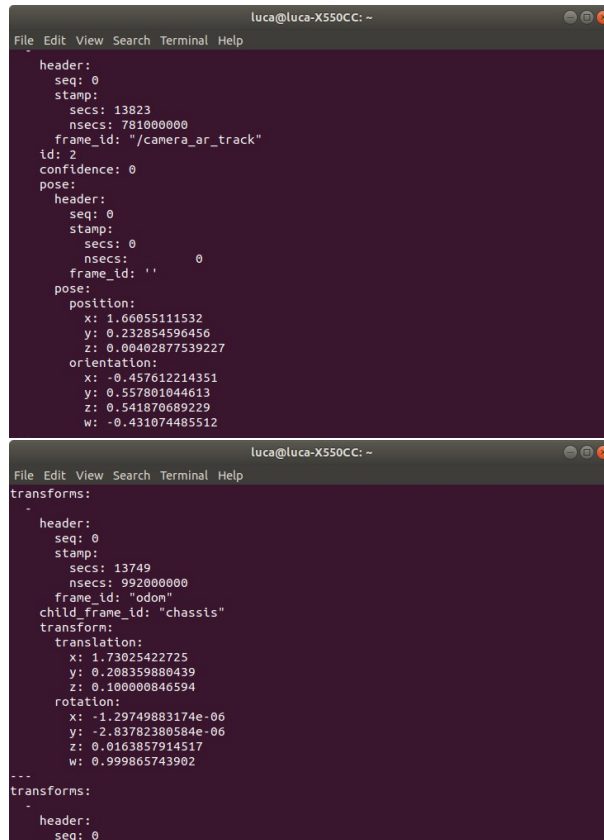
So, with reference to the picture above, the data are provided with respect the reference frame attached to the camera, but the strategy needs the tag pose data with respect the fixed map frame. This will be explained further with the definition of internal and external kinematics.

To address such an issue, the data coming from the tracking package has been coupled with the odometer information provided by the Gazebo simulation. This information tracks the reference frames transformations between objects so that it is possible to convert the local reference frame into global reference frame. The package that publishes the transformation messages is *tf2*. It lets the user keep track of multiple coordinate frames

over time. Luckily, this component is already implemented on Gazebo and no other nodes have to be connected to.

The *tf2* package shares its information about the position and orientation by the mean of a ROS message that indicates the 3D (x,y,z) translation and the rotation of one frame (i.e. child frame id) to another (i.e. frame id) [49].

The same with the other topic at stake, *ar_pose_marker*, that is the topic published by the AR tag tracking node *ar_track_alvar*. Even in this case, the content of the message is shared through an three-dimension information of translation and the two-reference-frame rotation.



```

luca@luca-X550CC: ~
File Edit View Search Terminal Help
header:
  seq: 0
  stamp:
    secs: 13823
    nsecs: 781000000
  frame_id: "/camera_ar_track"
id: 2
confidence: 0
pose:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs: 0
    frame_id: ""
  position:
    x: 1.66055111532
    y: 0.232854596456
    z: 0.00402877539227
  orientation:
    x: -0.457612214351
    y: 0.557801044613
    z: 0.541870689229
    w: -0.431074485512

luca@luca-X550CC: ~
File Edit View Search Terminal Help
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 13749
      nsecs: 992000000
    frame_id: "odom"
  child_frame_id: "chassis"
  transform:
    translation:
      x: 1.73025422725
      y: 0.208359880439
      z: 0.100000846594
    rotation:
      x: -1.29749883174e-06
      y: -2.83782380584e-06
      z: 0.0163857914517
      w: 0.999865743902
---
transforms:
-
  header:
    seq: 0

```

Figure 4.6: *ar_pose_marker*: transformation between reference frames of camera and tag (top), *tf2*: transformation between map reference frame and robot one (bottom).

The pictures represent transformations shared in *ar_pose_marker* and *tf2* topics. They are showing these packages messages by the mean of the *echo* ROS topic feature. In particular, they are shown the transformations between fixed map reference frame and the robot chassis one (bottom figure) and between camera optical reference frame (on the

robot) and AR tag frame (top) with reference to the *Figure 4.6* working conditions.

What is missed at the end is only one piece to complete the connection and to extrapolate the docking parameters between tag frame and global frame and it's just a gap between the two robot reference frames. As one can notice looking at the Rviz visualization, the camera reference frame and the chassis reference frame are just shifted in the x direction (red axis). The former is attached to the camera optical link, while the latter is placed at the origin of the chassis element. The dimension of this shift is an information contained in the URDF file.

The two message definitions relative to the topics displayed on *Figure 4.6* are grouped in the following picture.

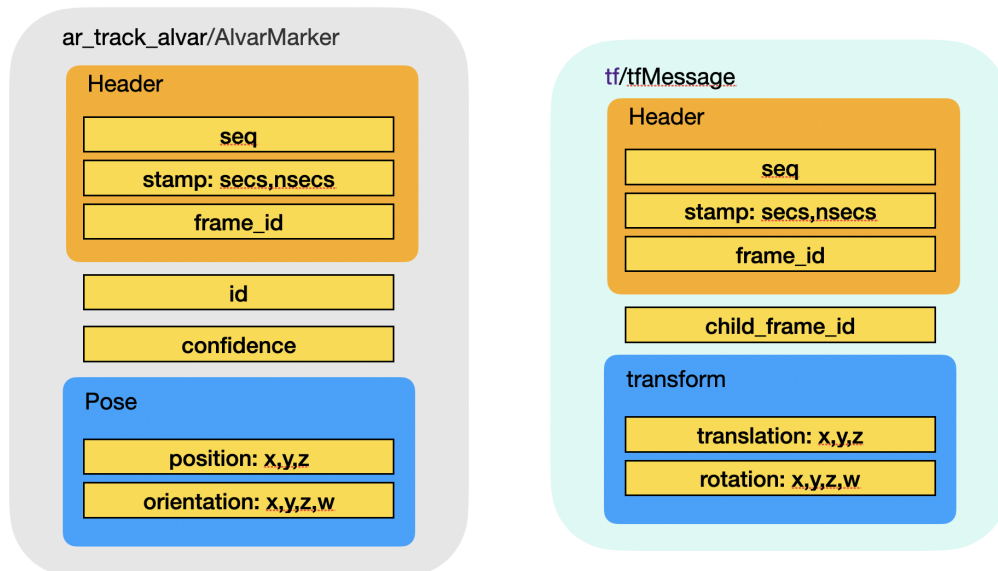


Figure 4.7: Definition of `ar_pose_marker` message

The first message is defined as `tf/tfMessage` Message in the `tf` [47], while the second is defined as `ar_track_alvar/AlvarMarker` and cited in [48]. The translation data is shared as a 3D vector. It represents the `frame_id` pose with respect the `child_frame_id` with the x-y-z triplet as reference (red-green-blue lines).

In the `tf2` message, x dimension is the largest one and it represents the distance between the frames along the red axis, while y is referred to the horizontal displacement. By considering a robot, in the case of this couple map-chassis, z is a fix dimension set by the robot building.

The rotation information is published with the convention of quaternions in both topics. So, to retrieve the RPY angles, it is necessary to apply the correspondent relations [50]:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0q_1 + q_2q_3), 1 - 2(q_0^2 + q_1^2)) \\ \text{asin}(2(q_0q_2 - q_1q_3)) \\ \text{atan2}(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

Taking as reference another time the last pictures in the terminal window below and applying the formulas above, the resulted angles are all null and it is confirmed by the Rviz visualization (the frames are only shifted).

The last step that has been carrying on is to transform this data in information of absolute distance, angle and orientation of the robot with respect the tag. Thus, the distance is computed with the Pitagora theorem applied to the x and y contributions of the tag pose data (contained in *ar_pose_marker*), as described in the docking algorithm explanation. To compute the angle and the orientation is a different story. In both cases we have to take in consideration the reference transformation topic.

As regards the orientation, in order to retrieve this information, we need to know two dimensions:

- **initial orientation:** This is relative to the orientation between the fixed reference frame (i.e. odom) and the AR tag. This dimension is constant and set prior to the test. In the tests made in this project, this orientation is taken as zero that means the tag is parallel to the global frame.
- **inline orientation:** It keeps track of the robot orientation with respect the odom frame (information contained in *tf2*). It is computed inline by using the conversion from quaternion to Euler angles formulas and taking ϕ for this data.

As far as the angle is concerned, in order to address the issues arose in the beginning of this subsection, besides from the angle computed as the arctangent of x and y pose axis of the mark, we need to add the orientation information computed before.

So, also in the case of the angle, it is composed of two contributions: the orientation and the mark angle.

At the end, the three parameters fundamental for the algorithm are computed as follow:

$$\mathbf{distance} = \sqrt{x^2 + y^2}$$

$$\mathbf{angle} = \mathbf{orientation} + \mathbf{atan}(x/y)$$

$$\mathbf{orientation} = \mathbf{initial_orientation} + \mathbf{Phi}_{robot}$$

They are translated as simple operations in the ROS nodes.

4.2.2 Docking function

Once the distance, angle and orientation are extracted the docking function is ready to be implemented in the simulation. This function should take as input these parameters during the entire duration of the operation and to give as output the wheels speeds in the form of a ROS message.

The development of the first stage of the docking operation includes the computation of the desired orientation ϕ , the development of a function for the computation and publication of the speeds under the *Twist* message and the docking operation itself.

The former has been carrying on by getting experience with the camera visualization. By setting different values of distance and angle, the maximum orientation of the robot allowing to not lose the contact with the tag is taken over and noted. Based on these orientations it has been built two linear relation, one for the distance and one for the angle representing ϕ .

According to the docking algorithm for the first phase, it has been distinguished four cases based on the position of the robot with respect the tag (left or right) and on the comparison between the robot orientation ϵ and the desired orientation.

The pictures in *Figure 4.8* only represent two cases, with the robot on the left side of the tag. The same argument can be carrying on with the right side. The two exposed situations are the only ones that can happen on the first phase. The robot orientation either can overcome ϕ or can be surpasses by it.

So what needs to be checked is the sign of the angle so to recognize the side in which the robot lays and if the orientation overcomes the desired orientation or it is under this value.

Once the side is identified, it's only a game of checking the orientation. Let's consider the case of the pictures with the robot on the left. In the left image, the orientation of the chassis is higher than ϕ . The robot, in this situation, has to be instructed to turn

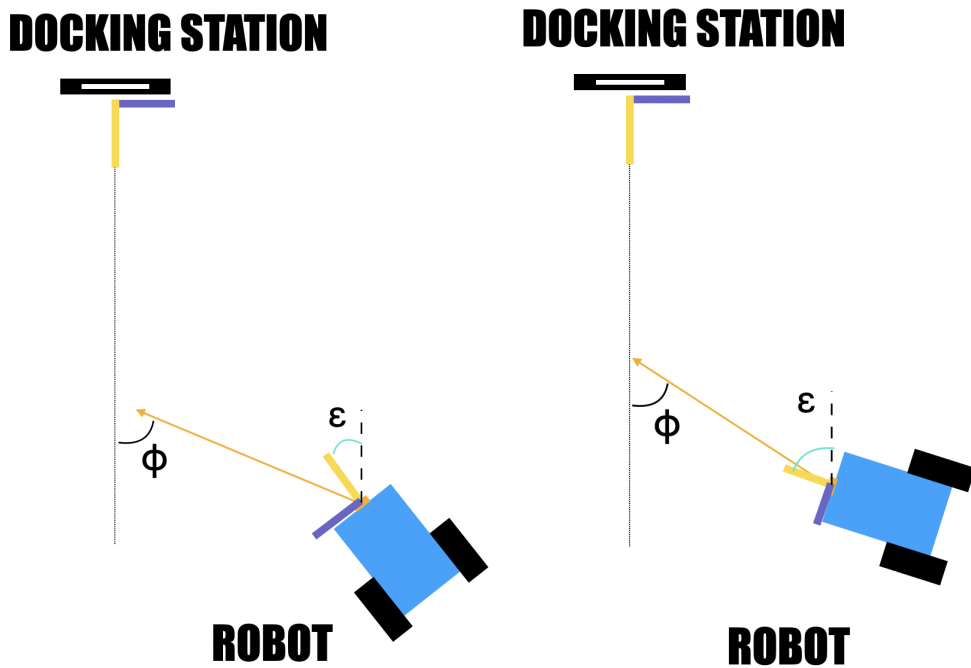


Figure 4.8: First phase of the docking operation: on the left the orientation is lower than ϕ , while is higher on the right.

clockwise. Instead of the left screenshot, in which the orientation is lower than that value. The robot should rotate counterclockwise to recover this gap and to reach the desired orientation. As regards the robot being in the right hand side of the tag, the two rotation sign has been shifted so to achieve the same result.

While distance and angle change during the movement of the robot, even ϕ changes as a consequence. So if the robot reaches the desired orientation for a while, that orientation is no more valid and the robot should adjust itself.

If the docking operation is going well, the robot will continuously oscillate around ϕ for the entire duration of the first phase. It is a kind of feedback system that adjusts the velocity of the robot in order to keep the orientation as close as possible to the desired one.

As far as the computation and publication of the speeds is concerned, the differential drive plugin implemented on Gazebo requires a *geometry_msgs/Twist* message to be communicated to the ROS Master [51]. Such a message is very simple and it is composed of two vectors with three elements each. The two vectors represent the translation and the rotation. The three elements are connected to the translation along and rotation around

the three reference axis.

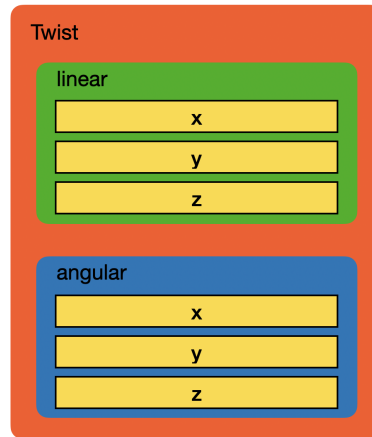


Figure 4.9: Twist message definition

Taking as reference the local reference frame of the two-wheeled drive robot, the three parameters describing the movement on a plane are the x and y translations and the rotation around the z axis. The other three are neither reachable for the robot nor part of the plane path.

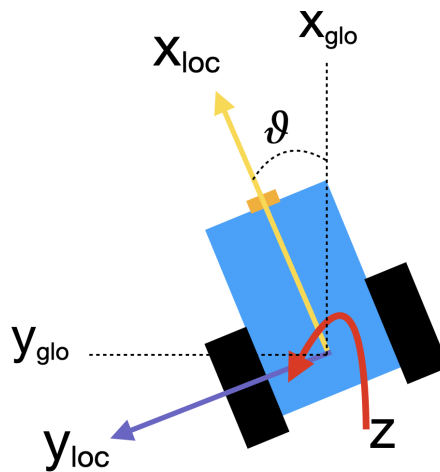


Figure 4.10: Robot directions of movement

However, how it is possible to get from the *Figure 4.10*, the robot ain't capable to move

in the direction of y axis. The robot can move either along the x axis direction or around the z axis. This is referred to us in robotics as the internal kinematics, which describes the relationship between the robot wheel rotation and how it moves. In contrast with the external kinematics, which describes the robot's position and orientation relative to a point in a global reference frame, such as a room. In this case, Y component can be non-zero, since we can drive the robot to any location in the map using a series of maneuvers. Thus, since the message is referred to the internal kinematics of the robot, the only two parameters that can be set on the ROS *Twist* message are the two connected to translation x and rotation z.

The function for the computation of the velocities has been designed as follows:

$$V_{linear} = V_{min} + (V_{max} - V_{min}) \cdot (d/d_{max})$$

$$V_{angular} = twist \cdot (d/d_{max})$$

The tactic used to set the speeds is another time the linear relation with respect the distance. It is a kind of Hook's law for the forces but applied to the velocity. The working method is simple, the higher is the distance, the higher will be the velocity. In this way, the safety is well preserved, since as long as the robot is far from the tag it can go faster, while if it is close to the station it has to keep the velocity low in order to avoid collisions with it compromising the integrity of the system.

As far as the linear velocity is regarded, it has been designed a line that starts from a minimum velocity and passes through a maximum velocity at a given distance. The minimum velocity is designed as the minimum velocity the robot appreciably moves in the Gazebo environment. Instead of the angular velocity, that is a line that begins from zero. The positive rotation is the counterclockwise one. The other values of parameters are selected according to the simulation trails.

To better figure out these two functions, one can refer to two graphs describing them, with the speeds on the y axis and the distance on the abscissas axis.

As regards the development of ROS, this computation speeds function represents a callback function that is called by the docking function depending on the robot parameters. Then, the two values of speeds are printed on the prompt and at the end published on the Master with the Twist message.

The speed function is called by the callback function of the docking node whenever a new message publishing the data of distance, angle and orientation has been shared.

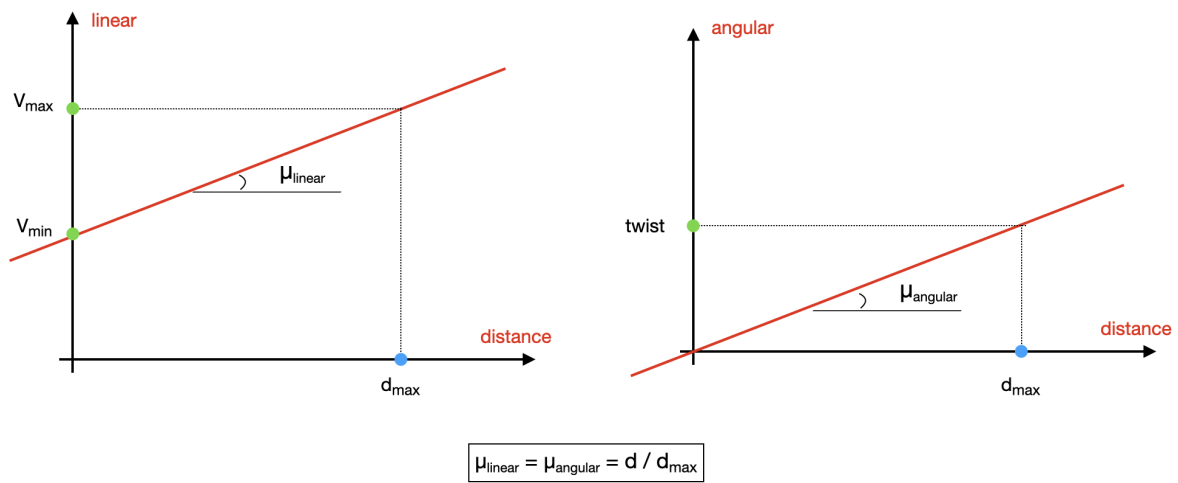


Figure 4.11: The graphs representing the behaviors of the linear and angular speeds with respect the distance between robot and station.

```

1 This function addresses the first phase of the docking algorithm
2
3 void DockingCallback(VectorArgument tag)
4 {
5     Computation of the desired orientation Phi with the use of the new
6     parameters of the tag variable
7
8     if distance > distance_min
9     {
10         // 1st phase
11
12         if (robot is on the left - check made with angle and maxangle - and
13             orientation is higher than Phi OR robot is on the right and
14             orientation is lower than Phi)
15         {
16             SpeedsFunction(tag,twist_max); //counterclockwise rotation
17
18         }
19
20         else if ( robot is on the left - check made with angle and maxangle
21                 - and orientation is higher than Phi OR robot is on the right and
22                 orientation is lower than Phi)
23         {

```

```

21     SpeedsFunction(tag,-twist_max); //clockwise rotation
22
23 }

```

This script will be continued further.

The algorithm above represents the statements used for the first phase. The *tag* element is a ROS message format defined by three float64 variables, that has been created for the parameters extraction function.

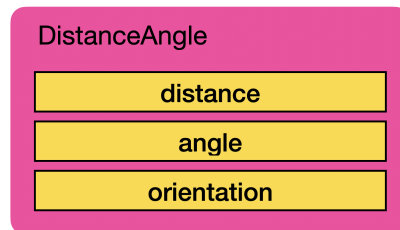


Figure 4.12: DistanceAngle message definition

This message has been built to be used by the function that spits out these parameters. Then, the docking function makes use of the message to trace them.

The *tag* element has been retrieved by the callback function. Every time the callback function is executed, the distance, angle and orientation update their value and the orientation ϕ is updated according to the distance and angle.

Then, the code includes different *if* statements to tackle the different working conditions. The first of them checks whether the distance is higher than the distance in which the robot should be docked (i.e. *distance_min*). As long as the distance is lower than this quantity, the strategy is executed in the function.

The first phase begins after the homonym comment at line 9. Here it is the moment in which the four cases cited above come into play. Since the operation to be executed are just two, either to rotate clockwise or counter clockwise, the four cases can be grouped according to the operation to do. This is done by the mean of an *OR* boolean operation inside the *if* statement. The two variables that are checked in the statement are the orientation and the angle of the robot.

The value of the twist has been chosen in accordance with the linear components, so to be as smooth as possible during the movements.

The operation that allows to recognize whether the robot is on the left or on the right of the station is the check done on the robot angle. Therefore, if the angle is higher than a

certain angle, the angle is positive so the robot is on one side than another. That certain angle is *maxangle* and it has been initialized with a small value. It defines the narrow line dividing the first phase from the second phase.

As long as the absolute value of the angle is higher than *maxangle*, the first phase piece of code is executed. If the operation is going well, the angle keeps its sign and the execution jumps from one *if* to another based on the orientation.

Figure 4.13 may help to better visualize these passages:

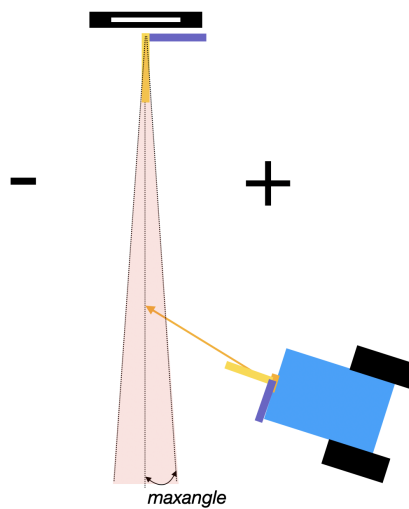


Figure 4.13: First phase angle signs and definition of the areas distinguishing first and second phase of the docking operation.

When the robot reaches that arc of circumference described by *maxangle* (pink triangle on Figure 4.12, it means the robot is approaching the second phase. This is a sensitive phase. The robot has rotate to change its orientation from a value close to ϕ to null in order to point at the tag. The changing in velocity of the wheel has to result in a rotation towards the station.

After the check on the angle whose absolute value has to be lower than *maxangle*, the twist has to be set in order to have the tag in the center of the visualization keeping the robot in the triangle.

The following is the continuation of the last script:

```
1 // 2nd phase
2
3 else if (robot is on the center - angle is between -maxangle and
```

```
maxangle)
4   {
5
6   if (orientation is out of the tolerance - counterclockwise)
7   {
8
9       SpeedsFunction(tag,-maxtwist2);
10
11   }
12
13   else if (orientation is out of the tolerance - clockwise)
14   {
15
16       SpeedsFunction(robot,maxtwist2);
17
18   }
19
20   else
21   {
22
23       SpeedsFunction(tag,0);
24
25   }
26   }
27 }
```

The value setting the *twist* component in the first two calls of the *SpeedsFunction* is different than the one of the first phase. That's because the distance in this phase is lower than in the first one, but the robot has to move in a reactive way to answer at the orientation changing requests.

It is worth noticing that if the robot goes outside the second phase area, the first phase algorithm takes action bringing back the robot to that area.

The last statement (i.e. *else*) is executed if both the robot is keeping the orientation lower than *maxorient* and the angle is in the range of the pink triangle. In this case the twist is set to 0 and the robot is ready to approach the station to complete the docking operation.

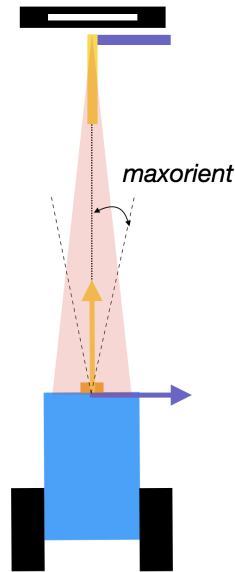


Figure 4.14: Tolerance on the orientation. If the orientation overcomes this value, the algorithm takes action.

4.3 Testing

This section is aimed at showing how the simulation tests have been developed. This will point at the execution of the functions explained so far.

But, before the functions it is needed to prepare the test environment and to launch the required nodes. A launch file has been created for accounting this and it includes the following:

1. **world**: upload the environment pre-created for the simulation.
2. **robot**: upload the URDF file of the robot.
3. **Rviz**: launch Rviz visualization.
4. **ar track alvar**: launch the tracker node.

Moreover, a pre-designed joystick function has been implemented in order to improve the test functionalities [52]. With this program the developer is able to remotely drive the robot with the use of some keyboard buttons. This will help us to bring the robot right where we need and to set the speeds of the docking algorithm according to it.

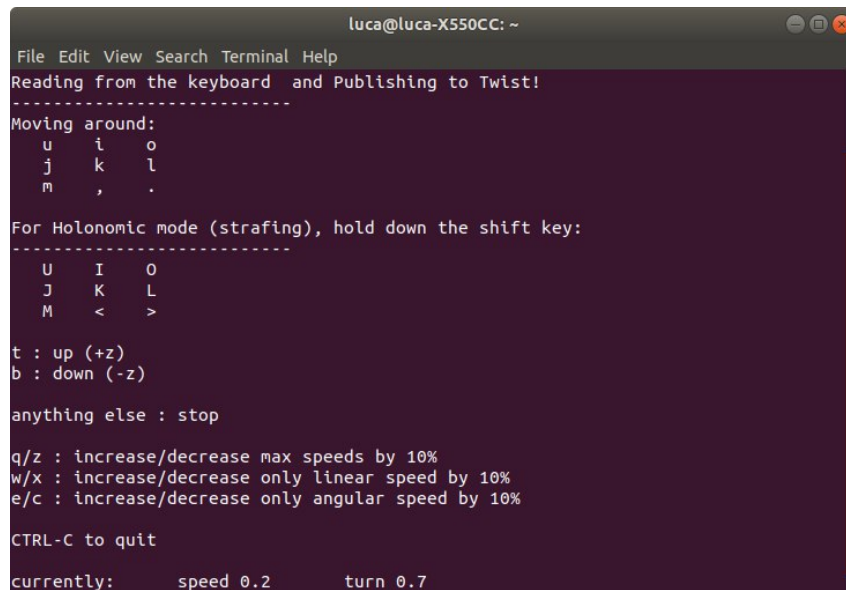
A screenshot of a terminal window titled 'luca@luca-X550CC: ~'. The terminal displays a joystick control interface. At the top, it says 'File Edit View Search Terminal Help' and 'Reading from the keyboard and Publishing to Twist!'. Below this, a dashed line separates the header from the controls. The controls are divided into two sections: 'Moving around:' and 'For Holonomic mode (strafing), hold down the shift key:'. The 'Moving around:' section lists keys u, i, o, j, k, l, m, ,, and . with corresponding movement directions. The 'For Holonomic mode' section lists keys U, I, O, J, K, L, M, <, and > with corresponding strafing directions. Below these, it lists 't : up (+z)', 'b : down (-z)', and 'anything else : stop'. Further down, it lists 'q/z : increase/decrease max speeds by 10%', 'w/x : increase/decrease only linear speed by 10%', and 'e/c : increase/decrease only angular speed by 10%'. At the bottom, it says 'CTRL-C to quit' and 'currently: speed 0.2 turn 0.7'.

Figure 4.15: Joystick function on the simulation test

Although the linear and angular speeds are constant by using the joystick buttons, it is possible to increase the maximum values of 0.2 for the linear and 0.7 for the angular (as can be seen from the terminal on *Figure 4.14*) of 10% of either the former or the latter or even both.

This function is used to drive the robot close to the station in order to make the robot pointing and detecting the tag.

Once the camera plug-in recognizes the tag and the marker is put on it in the Rviz simulation, this will mean that the *ar_pose_marker* topic (i.e. the one described in the paragraph *Extraction of parameters*) has been published. At this point, it is possible to execute the function responsible to transform this message into information about the distance, the angle and the orientation, so that the docking function can be processed. As we have seen before, this latter function publishes the Twist message with the relative linear and angular speeds. However, only the two updated speeds (the x axis linear speed and the z axis angular speed) in the picture case are printed on the prompt.

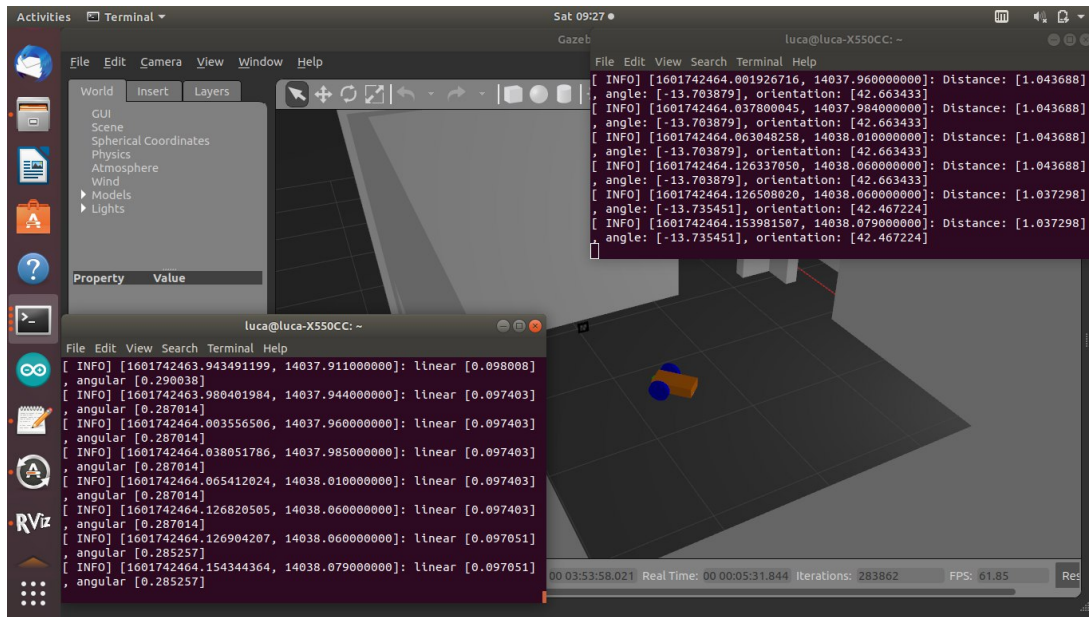


Figure 4.16: Caption taken during the docking operation: the robot is approaching the tag on Gazebo on the background, while the message of distance, angle, orientation one the left and of the speeds on the right are publishing.

Figure 4.15 shows the two terminals relative to the two cited functions publishing messages on the prompt during the maneuver of docking. With reference to the picture, the terminal on the right side is sharing information about the main parameters, while the

left side one is printing the speeds out.

As far as the parameters are concerned, they seem to reflect the what's they appear in the simulation. The angle is negative since the robot is on the left hand side of the station and the orientation is positive, as the robot is rotated counterclockwise.

The linear and angular velocities looks coherent with the speeds on the joystick function. The relation between one and the other is very similar.

Chapter 5

Custom Robot

The idea of building a custom robot was born from the impossibility to test the algorithm and the software designed for the simulation on the actual project robot. The robot developed for the project was inside the Department of Electrical and Computer Engineering of California State University Los Angeles, that was close during all the duration of the experience there. As a consequence, we can't be able to use it as prototype for this subsection.

The requirements for the robot were very basic. The robot should be a small prototype equipped with motors and wheels, as simple as possible. What is needed for this part is just a chassis to allow the stereocamera detecting the tag to move so to reach the station step by step. The chassis has to be capable to hold the hardware elements needed for the communication systems, as well as to be compatible with the economic resources of the project.

A part from the chassis, which is only an aesthetic/mechanical/economic matter of choice, the communication between the ROS nodes messages and the motors receiving this information has to be addressed.

The ROS framework can no more run on a Personal Computer due to the high weight the chassis should hold having a PC on the back. In this regard, a good solution for this issue can be the implementation of an embedded platform with acceptable performances of reactivity and reliability. This system must be equipped with a powerful processor and an high enough RAM and ROM memories so to execute all the nodes the algorithm needs to. Moreover, the hardware has to be compatible with the Linux Ubuntu open source operating system and, as a consequence, to the ROS development system.

After the choice of the platform responsible to execute the algorithm nodes, the data of speeds must communicate to a platform capable to convert and to send voltage signals the motors require.

The parameters that have been taken into account for the choice of the different elements for the construction of the robot are:

- **cost:** The prototype, at least for the time being, should only execute the autonomous charging function. Thus it is not useful in this moment of the project to buy high performance components.
- **smart design:** The purchased elements must be characterized by an ease to mount, install and connect to avoid wasting time on installation or to find special parts or tools.
- **market-compatible:** Being only a subsection of the project that was developed by the author alone and considering the electrical competences of the author, it is necessary to chose market elements, so that a community, forums and libraries are well available.

This chapter addresses the choice of the hardware elements for the robot, together with their set up and communication systems. Then, the text will explain the modification made on the software in order to add the compatibility with the hardware and to preserve the reusability of the code with a sort of abstraction layer system. At the end, it is presented the speed control system developed in this work to tackle the issue connected with the speed signals.

5.1 Mechanical design

As regard the mechanical design is concerned, the robot has to follow the guidelines of the project. Both the Cal State laboratory robot and the robot in the simulation employs the differential drive fashion with three wheels.

The differential drive robot is a mobile robot whose movement is based on two separately driven wheels placed on either side of the robot body. It can thus change its direction by varying the relative rate of rotation of its wheels and hence does not require an additional steering motion. The simplest way to use such kind of system is to equip the two back wheels with motors and keep the last wheel idle.

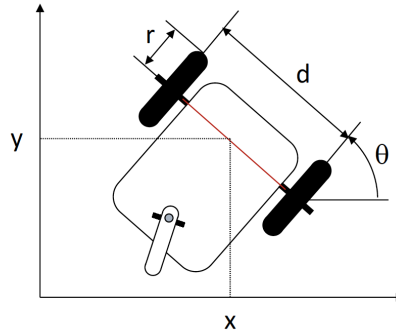


Figure 5.1: Differential drive robot spatial parameters

The physical inputs are the rotation velocities of the two wheels, w_l and w_r . There exists a direct correspondence between the input and the linear velocity v and angular velocity w that are referred to the center point between the two wheels (x, y) . With this transformation, the local reference frame parameters can be easily transformed into the global reference system by the mean of few passages:

$$\mathbf{v} = r \cdot (w_r + w_l)/2$$

$$\mathbf{w} = r \cdot (w_r - w_l)/d$$

Where d , as shown in *Figure 5.1*, represents the axis distance between the two wheels.

5.2 Hardware set up

This section is aimed at explaining the main features of the chosen hardware elements to build up the custom robot ecosystem.

After further considerations, the Jetson Nano embedded platform from NVIDIA has been chosen as the system responsible for running the ROS/Linux functions. This board is characterized by a small dimension against good performances of clock and memory.

Then, Arduino UNO compatible platform and L298N motor driver has been equipped to the custom robot in order to address the speed communication issue mentioned so far. As it was explained in the docking algorithm section, the strategy requires the odometry data, to track the robot movements. The implementation of encoder sensors tries to face the need of this data.

At the end, it has been exposed how the different hardware elements has been put to communicate between themselves. This involves explaining how data is exchanged between Jetson Nano and Arduino and between Jetson and the camera.

5.2.1 Jetson Nano

NVIDIA Jetson Nano is a small and powerful computer that lets you run applications like image classification, object detection, segmentation, and speech processing. It brings AI to a world of new embedded and IOT applications, including entry-level network video recorders, home robots, and intelligent gateways with full analytics capabilities [53].

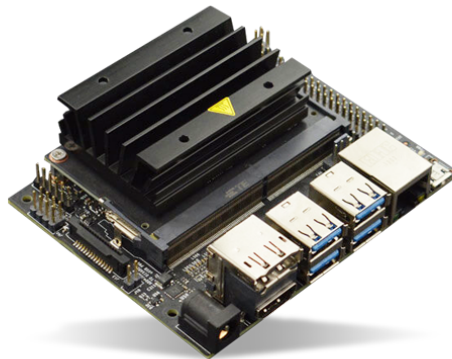


Figure 5.2: Jetson Nano embedded platform by NVIDIA

There are plenty of robot projects developed with the use of Jetson Nano, including *Jetbot* [54]. This latter is an open-source AI robot platform for developers and students. You can buy choosing among the several preassembled hardware kits or you can build your own Jetbot by following some guidelines [55].

Jetson Nano is characterized by the following specifications:

Characteristic	Spec
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
RAM	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	64GB micro SD
Connectivity	Gigabit Ethernet, Wi-fi with add. module
I/O	4x USB 3.0, USB 2.0 Micro-B, GPIO, I2C
Display	HDMI e DP
Mechanical	100 mm x 80 mm x 29 mm

Table 5.1: Specifications of Jetson Nano board

Thanks to its small dimension and low weight it can be easily implemented into a small robot. Moreover, this device doesn't need a great power to run. In our robot, this platform has been supplied by a power bank with an output of 5 Volt.

The very first step when using such a board is to flash the micro SD with Jetpack SDK to get ready to develop on it so to subsequently install tools and libraries, samples and documentation needed to jumpstart the development environment.

5.2.2 Arduino UNO and motor driver

Arduino Uno is a microcontroller board based on the ATmega328P. The UNO is the most used and documented board of the whole Arduino family [56]. It has 6 analog inputs and 14 digital input/output pins. Six of them can be used as PWM outputs. The presence of those pins resulted to be the key for the choice of such a board. They answer to the need of a regulator for the voltage signals.

This board is connected to the Jetson Nano by the mean of an USB cable. With the use

of the Arduino IDE, each time it is needed to execute a new function, the PC running the IDE has to upload the function on the Arduino board so to make it work. The NVIDIA PCB is also responsible of the supply of UNO.

Since the motors can only be controlled by varying the voltage, it is needed a system that transforms the speed analog value to a digital signal. The sending Pulse-width modulation signals allows this transformation. This kind of signal is a digital signal marked by a duty cycle, that is the value defining how much the signal is positive compared to the total time. By increasing the duty cycle, the average value of the signal increases. With this simple trick, it is possible to convert the digital signal to an analog signal by the mean of an H-bridge circuit. In this case the H-bridge is represented by the L298N driver.

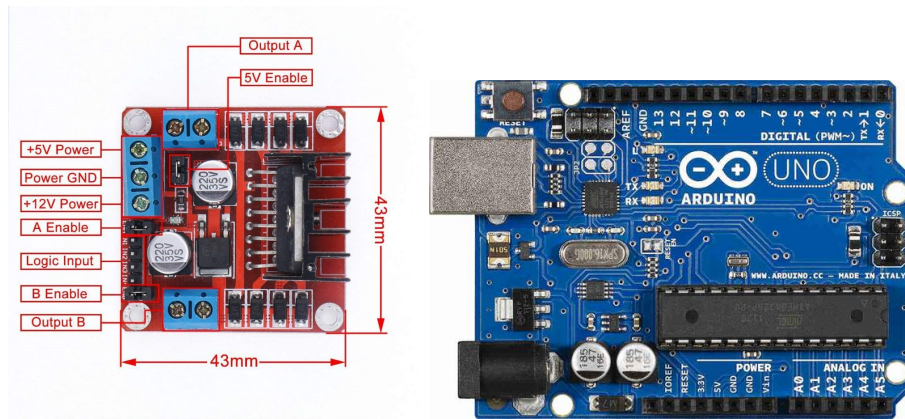


Figure 5.3: On the left the L298N driver board, on the right Arduino UNO platform

The motor driver essentially switches the polarity of a voltage applied to a load. A scheme of an H-bridge follows:

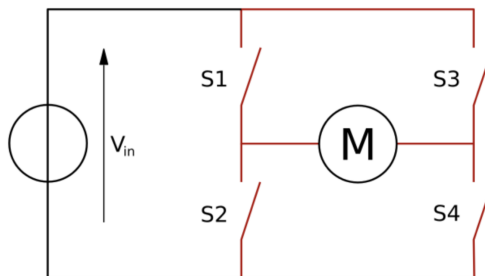


Figure 5.4: H bridge circuit scheme

The driver sends to the motors the voltage signals according to the duty cycle of the PWM wave and the input voltage. So for example, if the input voltage is 6V and the duty cycle is 50%, the motors will receive approximately 3V [57]. Actually, there is a voltage drop of 2 V with respect the supply voltage due to the leakage of the circuit. In our case, the motor driver is supplied by a AA battery pack with 4.8 V(4x 1.2V batteries), so the actual voltage on the motors is approximately 3 V.

5.2.3 Encoder sensors

The goal of this kind of sensor system is to count the amount of the rotation the wheel does during the movement of the robot. The sensor that has been picked for the robot is the LM393.

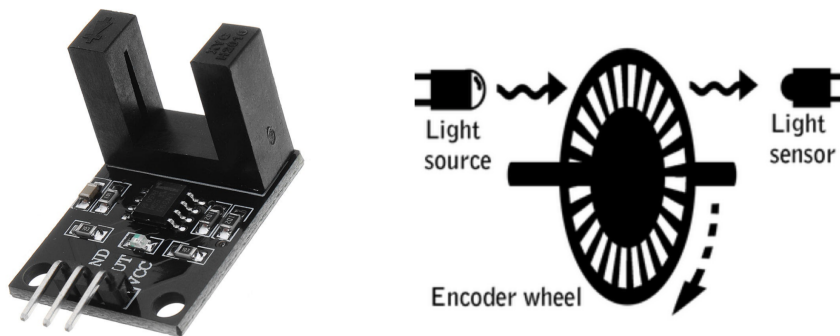


Figure 5.5: On the left the LM393 encoder sensor, on the right an image explaining the encoder system

The system is composed of two perforated discs attached to the wheels and the encoder sensor that perceives the holes in the disk. It has two light elements, one emitter and one receiver in the two columns visible in the *Figure 5.5*. The emitter sends the light and if the receiver gets it, it means that there is an hole.

The sensor sends digital signal as output. If an hole is detected, the sensor transmits 1, otherwise it sends 0.

5.2.4 Set up and communication system

The Jetson Nano is the actual core of the robot, where the algorithm is executed. It gets the camera data from the ZED 2 through an USB connection. From the Rviz visualiza-

tion and the marker detection it executes the algorithm that extracts the data of distance, angle and orientation between the camera and the marker and it publishes the velocity signals on ROS according to the docking algorithm.

The Arduino UNO function subscribes this speed message and sends the voltage PWM signal to the driver. Even the connection between Arduino and Jetson is made by an USB cable.

Eight are the signals that Arduino sends to the L298N driver through the jumpers. Two are the PWM signals related to the two motors (enA , enB), 4 are output digital signals that indicates the direction of the motion for the motors (clockwise or counterclockwise). The last two pins connected are the GND and the 5V. This latter signal is the reference signal, while the motor drive takes the supply voltage from the battery pack.

The motors are connected to the lateral sockets of the L298N driver. While, the two batteries are connected to the driver by means the wires of GND and reference voltage of the driver itself.

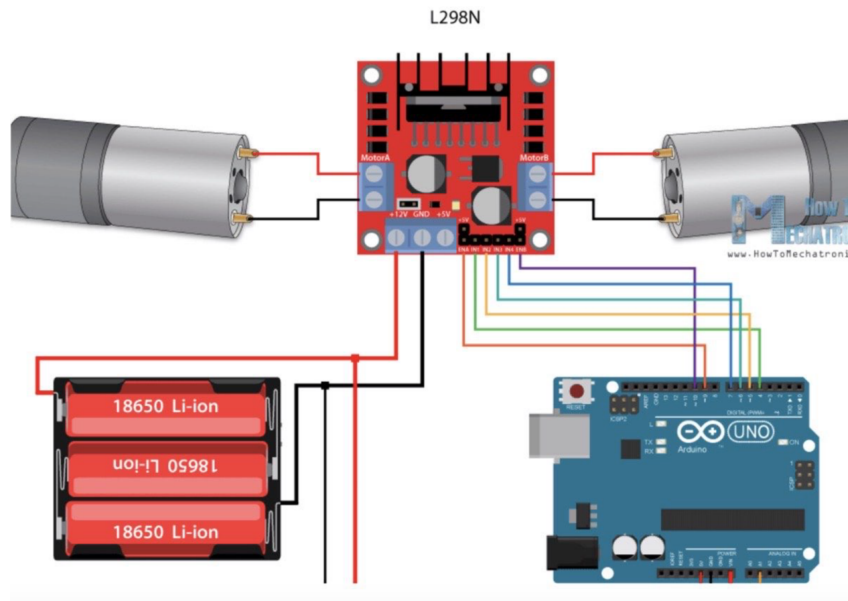


Figure 5.6: Scheme showing the connection between Arduino, motor driver, motors and batteries

Summarizing, the driver takes as input the PWM waves for the velocity commands, the I/Os for the direction of the motion, GND from both the batteries and Arduino, the input voltage supply for the motor of 3V from the series of batteries and the 5V input from Arduino.

As regards the encoders, both of the LM393 sensors are connected to the Arduino platform and each of them has 3 pins: two inputs, GND and VCC(5V) and the digital output. These latter have to be connected to digital input peripherals, so the Arduino can detect the change in the boolean value.

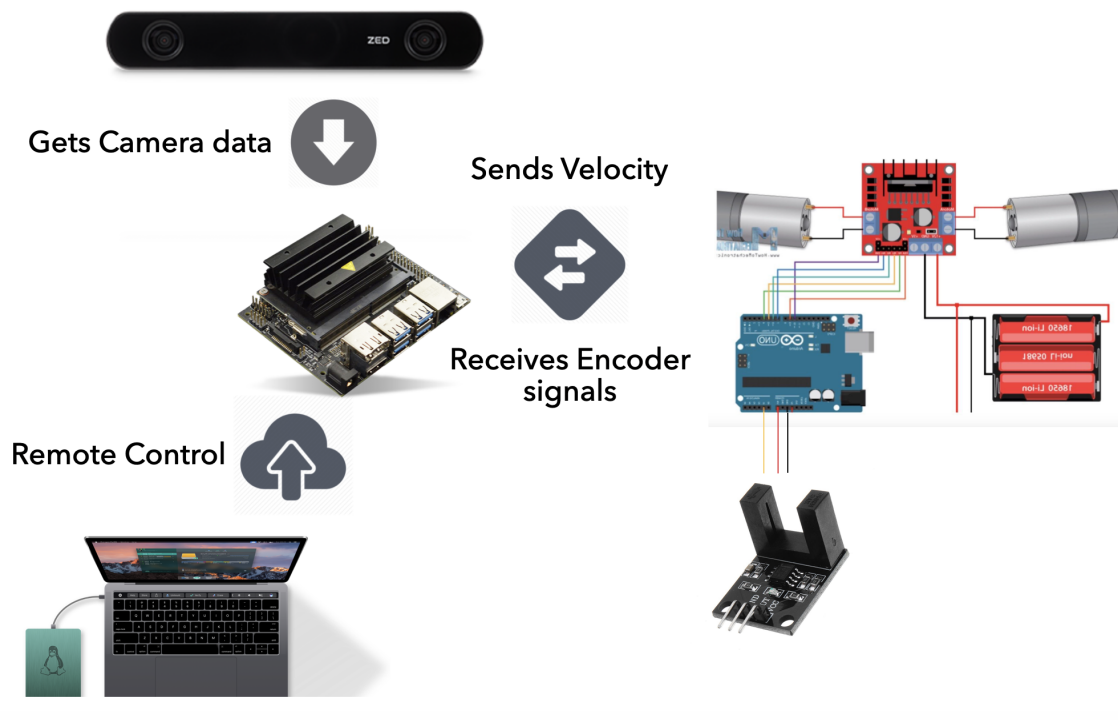


Figure 5.7: Communication set up between the various elements of the system

Figure 5.7 helps to figure out how the different hardware elements are connected together. It also indicates how the informations are exchanged between Jetson Nano and Arduino and Jetson Nano and camera.

At the end, in order to control the robot, a Personal Computer has been equipped with an application that allows the PC to visualize the output video of the Jetson Nano. This application is *NoMachine* and it has been installed on both devices. In addition, with this you can use the PC keyboard and touchpad to control the of Jetson, as for example to execute the ROS nodes.

The following bill of material lists all the components of the custom robot together with their discount cost and website shop.

Part	Quantity	Price	URL
Jetson Nano	1	99\$	[53]
ZED 2 camera	1	400\$	[58]
20 A Power bank	1	20\$	[59]
USB cable pack	1	7\$	[60]
4AA rechar. batteries	1	19\$	[61]
Wheels	2	-	[62]
Caster wheel	1	-	[62]
TT DC motor	2	-	[62]
Chassis	1	20\$	[62]
L298N driver	1	-	[62]
Arduino UNO compatible	1	-	[62]
3 Encoder sensors LM393	1	3\$	[63]
Screws, nuts and spacers	1	10\$	[64]
Wires	1	6\$	[65]

Table 5.2: Bill of material of the custom robot

By considering the Table 5.2 of material bill below, the total amount of costs to the robot is a bit less than 600 \$. The ZED 2 camera is the most expensive element and it covers the two thirds of the expenditure. The cost is justified being a powerful tool due to the sensors and feature it takes advantage of.

A part from these costs during the construction of the robot, it was necessary to make use of other tools and elements to assemble and to work the parts. The additional components employed during the operations of installation include a welder and solder for soldering the wires to the motors, sandpaper to chamfer and sand down the plastic chassis parts and a drill to make the holes for the screws and nuts.

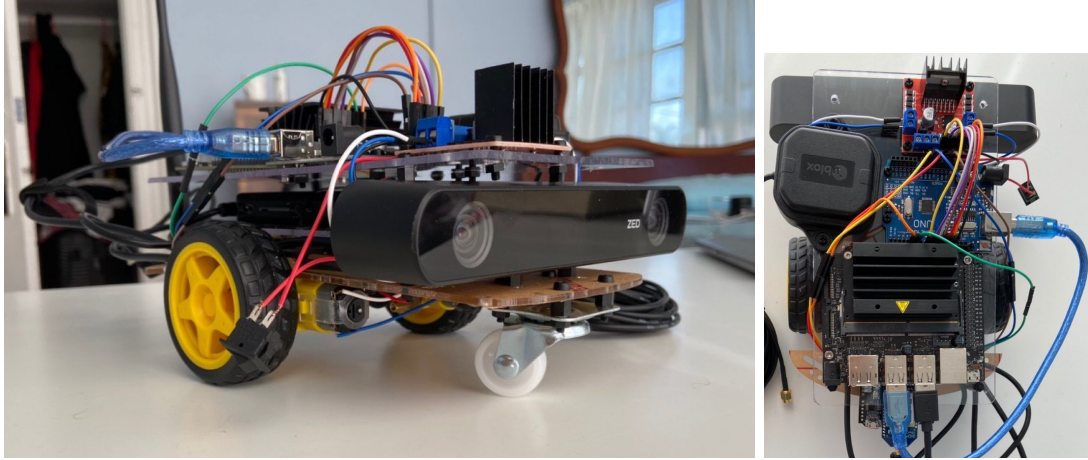


Figure 5.8: The front and the top view of the custom robot

The robot has been built over three floors. The first is composed of the motors, wheels and encoder sensors, the AA batteries on the back and the camera on the front. The second hosts the power bank and the third holds up the Jetson, Arduino and L298N driver. The final result is showed in *Figure 5.8* .

5.3 Velocity control system

At this point the path seems to be smooth like in the simulation, you give the velocity commands and the motors follow them. Although in the Gazebo simulation this is a kind of correct, because we simply are in a perfect world, the same can not be articulated with a real robot, with a real world. To have the same behavior of the simulation, the two motors should be perfectly equal, like clones. However, this is never the case and we need to have a feedback on the speed of the wheels. What is needed in this case is a control system capable to command the motors to reach the desired value of velocity.

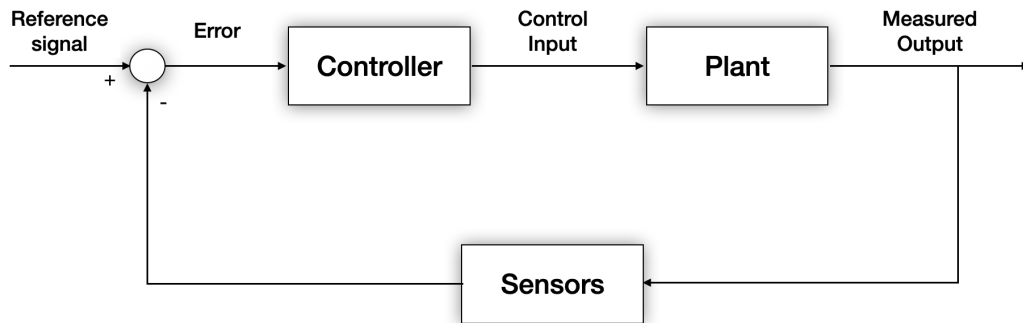


Figure 5.9: Simple scheme of a feedback control system

The simplest control system is composed of the controller, the plant and the sensors to measure the output. It continuously calculates an error value as the difference between a reference setpoint and the measured output and applies a correction based on the input computed by the controller.

The goal of the control system is to give the input to the plant, that in this case is represented by the motors, wheels and motor driver, such that the error between the desired output and the reference signal becomes null. With reference to the custom robot, the output is the speed taken by the encoder sensors, while the input is the amount of voltage to send to the system.

To design a control system is not an easy work when dealing with very cheap components like the common robot motors. Even to follow a straight path can turn to be a nightmare. Although these may seem like simple tasks it does present some programming challenges requiring control feedback loops and controllers.

To tackle this big issue it has been implemented a PID control system, that at this level it is a good compromise between computational effort required and actual effectiveness. It is the state of the art of the robotics projects that deal with the control of motors. It has been used also by a *mjwhite*, pseudonym of a Robotics Engineer in his project that shared in a post in *The Workshop* forum [66]. The author of the post implements a pose control loop, that encapsulates the speed control loop since its goal is to reach a position whatever is the path. In our case we need to follow the docking algorithm path, so we will limit our work to the design of the speed control loop.

So, PID stands for Proportional Integrative Derivative and it indicates the three different contributions of this model:

- **Proportional** : This term is like a gain factor, it directly multiplies the error. However it can not work alone since it requires an error to generate the proportional response.
- **Integrative** : It takes into account the past values of error and integrates them over time. It seeks to eliminate the residual error by the mean of the cumulation of past control actions. When the reference signal and the measured output are the same, the integrative part keeps constant its value.
- **Derivative** : It is the best estimate of the future trend of the error based on its current rate of change. The more rapid the change, the greater the controlling action.

The overall control input function is described as follows:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t')dt' + K_d \cdot \frac{de(t)}{dt}$$

The three parameters K_p, K_i, K_d are all non-negative and denote the coefficients for the proportional, integral, and derivative terms respectively. The set of these parameters is in charge of the *loop tuning*. The tuning consists in the adjustment of the parameters in order to reach the optimum values for the desired response of the control system.

A requirement for the tuning process is the *stability*, in particular the BIBO (Bounded-input bounded-output) stability. To not meet this requirements means the controlled

process input can be unstable, and as consequence its output diverges, with or without oscillation, causing damages to the plant.

The BIBO stability is defined: a linear time invariant system is BIBO stable if the zero state output response is bounded for all bounded inputs:

$$\forall u_M \in (0, \infty), \exists y_M \in (0, \infty)$$

$$|u(t)| \leq u_M, \forall t \leq 0 \Rightarrow |y(t)| \leq y_M, \forall t \leq 0$$

At this point, we pass from time domain to the frequency domain (s) through the Laplace formulation and we consider a control system with the function $K_c(t)$ representing the controller and $K_p(t)$ the plant, while the sensor has a unitary gain. We can say that the loop transfer function is:

$$H(s) = \frac{K_c(s) \cdot K_p(s)}{1 + K_c(s) \cdot K_p(s)}$$

The system is called unstable if the closed loop transfer function diverges.

Keeping in mind this, there are several methods for tuning the parameters. This work utilizes the manual tuning, that means to keep one variable constant and to vary the other in order to find the best fit value.

It has been decided during the design process to not include the derivative term because the utilization of the other two terms was enough.

The velocity control system employed for this subsection of the project is illustrated in *Figure 5.10*.

The proportional component **P** is just composed of a multiplication. While the integral component **I** is represented by a summation over all the errors times the integrative parameter K_i starting from time 0. It has been used the summation since we are dealing with discrete time contributions. The real actual output is measured thanks to the encoder sensor and transformed (we'll see how in the next section).

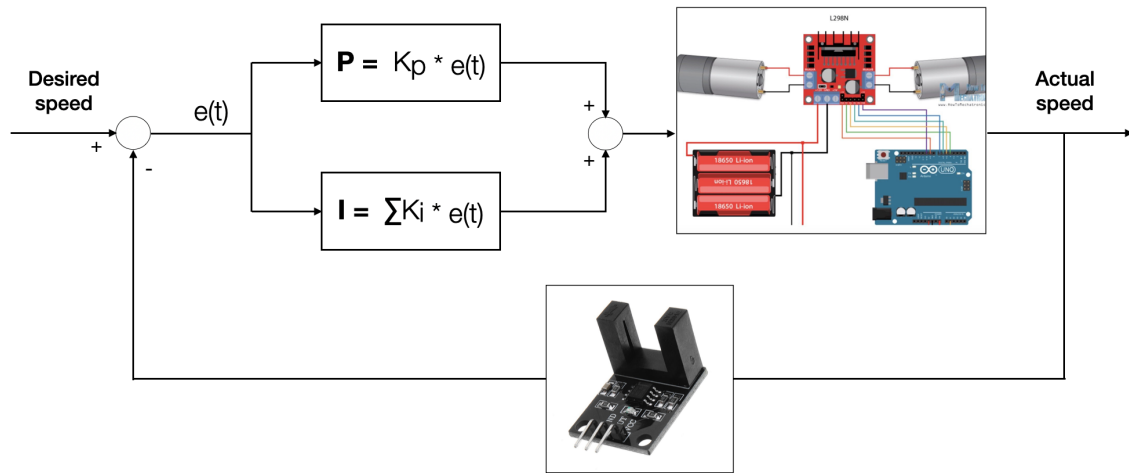


Figure 5.10: The front and the top view of the custom robot

5.4 Software compatibility

In this section, it has been explaining how the software developed on the simulation is adapted to the custom robot. The functions designed for the simulation compute and publish the robot speeds as a Twist message, leveraging the extracted Pose information of the tag. The goal is to take advantage of them to connect the different systems the robot is composed.

To accomplish such a purpose it is needed to implement some other features that tackle the implementation on the robot:

- Acquisition and publication of the **encoder** sensor data.
- Transformation and publication of the encoder sensor data in **odometry** messages.
- Subscription of the docking function to extract the desired speed and the odometer function for the actual speed. Application of the **input** resulted from the feedback control system to the motors.

This paragraph is divided in three subsections, according to the additional features. For each subsection, it will be specified the ROS type of message the data is willing to be published and the algorithm that allows to take out that result.

This work will try to design one function for each of the three points in order to decouple these new functionalities.

5.4.1 Sensor data acquisition

The aim of the software development carried out for this part is to find a way to detect and publish the amount of ticks (i.e. an hole in the encoder disks) the encoder sensors acquire during a certain amount of time.

Two are the values that have to be stored, one per each wheel. For this, *Vector3* is the type of message that has been employed to publish the number of ticks. This message is composed of three float variables x, y and z. Only the first two are updated with the ticks values within the message, while the last one is left null. This is done to make use of a standard type of message in order to avoid the creation of a new message every time

one have the necessity to do.



Figure 5.11: Vector3 message definition

The development of this functionality has been carried out on Arduino IDE, since it is Arduino UNO in charge of obtaining this data. The first thing to make is to initialize the pins variables according to the jumpers from the encoder sensor to the Arduino board. In particular, of interest are the pins receiving the digital output relative to the holes detection.

Then, the main function employs two interrupts attached to the pins and a timer interrupt set to 0.1 second. The firsts call a function that counts the amount of times the pins goes from *LOW* to *HIGH*, which indicates the encoder detects an hole on the disk. While the second publishes the two amounts of registered ticks until that moment and reset them. These steps are also exposed in the algorithm below:

```

1 This program detects the number of crossed ticks in a certain amount of
  time.
2
3 function IntCountLeft(){
4     Increase the left count value
5     end
6 }
7 function IntCountRight(){
8     Increase the right count value
9     end
10 }
11 function TimerInt(){
12     Publish the right and left values in the ROS topic
13     Reset the values of right and left counts
14     end
15 }
16 function SetUp(){
  
```

```

17   Set up the timer and the interrupt pins
18   end
19 }
20
21 {
22 In the main function
23
24   attachInterruptLeft(IntCountLeft)
25   attachInterruptRight(IntCountRight)
26   InterruptTimer(TimerInt)
27
28 }

```

5.4.2 Odometry

Once the sensor data are acquired and published on the ROS Master as a *Vector3* topic, it's the time to convert this data in an odometry information. The format of the output message in this case is the *geometry_msgs :: Odometry* [57]:

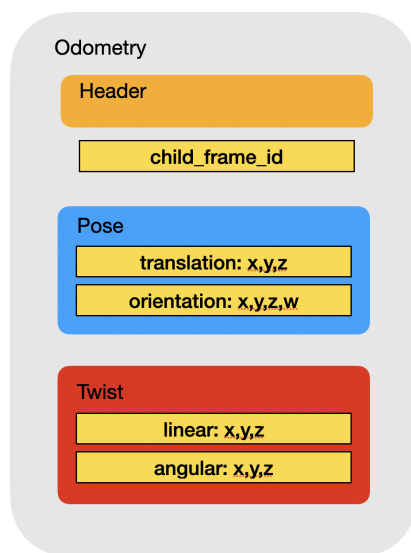


Figure 5.12: Odometry message definition

This message is defined by three other messages, Header, Pose and Twist and a string, *child_frame_id* and they are quite familiar to us. The Header is relative to the parent

reference frame and includes also the time stamp, while the Pose and Twist indicates respectively position-orientation and linear-angular velocity. The pose message is referred to the global reference frame, while the Twist to the local one.

The inputs for updating this message are only the two values (left and right wheel) of the ticks the encoder detects in a certain amount of time (0.1 s).

```

1
2 function WheelCallback(VectorArgument ticks)
3 {
4     Record ticks numbers and compute the wheels velocity multiplying
      the constant distancepercount and dividing the amount of time
5
6     Compute Twist local linear and angular speeds by applying the
      differential drive relations
7
8     Update translation and orientation variables by adding the x,y and
      theta values to their previous values
9 }
10
11 {
12 In the main function
13
14     Compute odometer Pose by deriving the delta x,y and theta. The first
      two are derived by applying the theta rotation matrix to the linear
      speeds and multiplying with the time, while the last only with the
      multiplication between angular velocity and time.
15
16     Update translation and orientation variables by adding the delta x,
      y and theta values to the message parameters
17
18     Publish the odometer data
19 }
```

As one can check from the algorithm above, it has been derived the wheels speed (V_{right} , V_{left}) from the ticks number in the specific period. As a consequence, the local linear velocity x , y and angular z (θ) are extracted thanks to the differential drive robot expedient.

$$\begin{cases} V_x = (V_{left} + V_{right})/2 \\ V_y = 0 \\ V_\theta = (V_{right} - V_{left})/d \end{cases}$$

Then, we retrieve the delta global translation Pose x and y applying the rotation matrix

with angle θ and multiplying the delta time.

$$\begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} = R \cdot \begin{bmatrix} V_x \\ V_y \end{bmatrix} dt, R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

While the delta orientation Pose z is just computed by multiplying V_θ with the delta time. The Pose parameters are updated summing these deltas up each cycle.

The odometer message is also subscribed by the docking function, as the algorithm requires an odometer sensor source.

5.4.3 Motors input

Here it has been explained how the speed control system is implemented in order to be synchronized with the other functions. The goal is to publish two integers relative to the Arduino PWM duty cycles leveraging the data of velocity from the sensor acquisition and docking functions.

Arduino can send a PWM wave to the driver with a duty cycle that depends on an analog value between 0 and 255 (8 bit). This value can be sent to the Arduino function that is responsible to transmit it to the correspondent enA and enB driver pins. Moreover, it is in charge of setting the motion direction pins accordingly.

The algorithm computing the input for the motors is divided in few steps:

1. Subscription of the two topics relative to the desired and the measured speeds.
2. Computation of the error ϵ between the two values.
3. Application of the PI control system through the definition of the Proportional and Integrative parts for each wheel.

$$P = K_p \cdot \epsilon$$

$$I += K_i \cdot \epsilon$$

4. Summation of the two parts in reference to the same wheel.
5. Publication of the resulted input value.

Even in this function, it has been made use of the same message format of the sensor data acquisition (i.e. `Vector3`) having just two values to be published. This message is then subscribed by an Arduino program that, each time a new message is published, transmits the PWM wave to the correspondent `enA` and `enB` pins with a duty cycle that depends on this analog value.

Now let's have a look on how the test has been prepared.

5.5 Test execution

The tests have been executed in the apartment which has been recreated in the simulation part. Moreover, at least for this test, the robot has been placed in the room so that it is directly pointing towards the tag.

In addition to the functions tested in the simulation part, prototype tests include additional nodes related to the extraction and publication of the encoder sensor data and the calculation of the input for the motors.

The main function is, another time, the AR tag tracking node, *zed_ar_track_alvar*. This time is already optimized for the ZED camera and it is contained in the packages released by Stereolabs in GitHub [68]. It executes the ZED wrapper that provides the messages of the camera visualization and it opens an Rviz visualization page that subscribes them. Then, as in the case of the simulation, the detecting node is launched in order to put the marker on the camera visualization and to attach the reference frame on the tag.

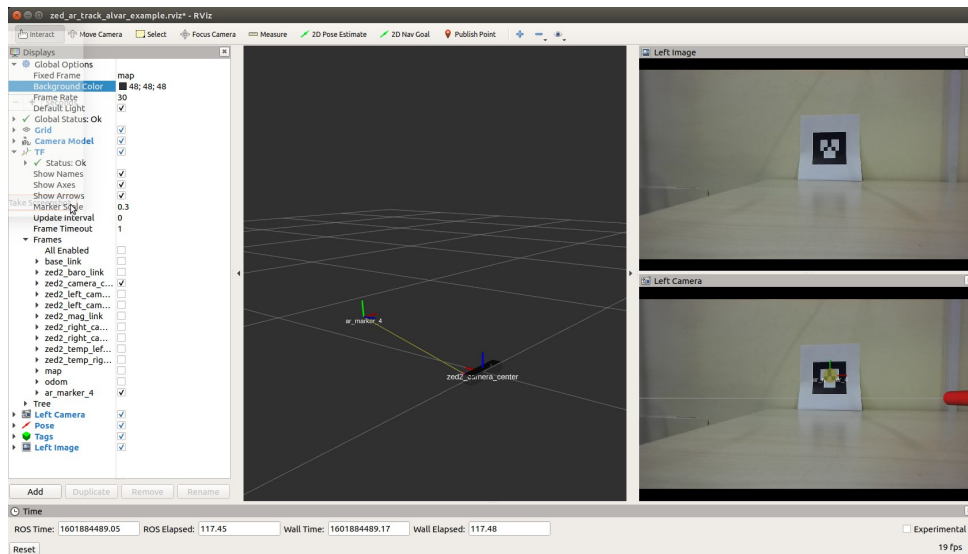
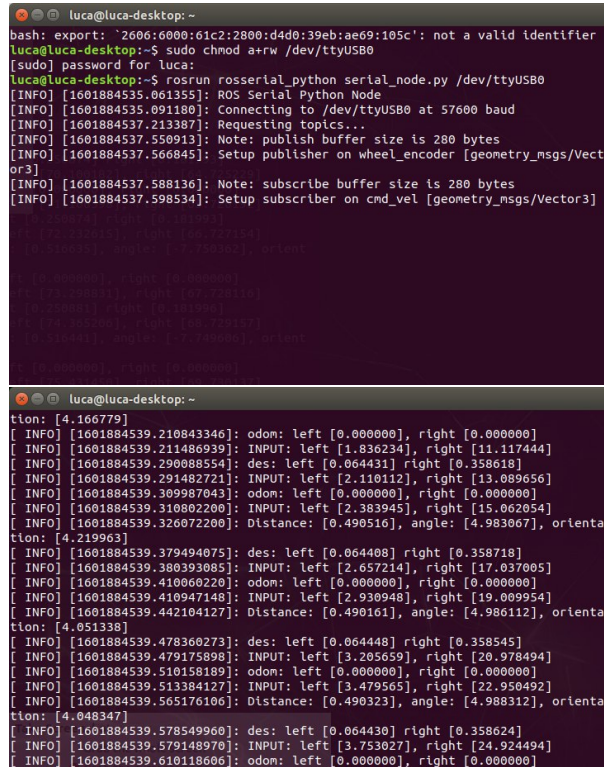


Figure 5.13: Rviz window showing the ZED camera visualization, the track marker and the reference frames.

The execution of these nodes results in the publication of the Pose message about the marker. Therefore the parameters extraction node and, as consequence, the function computing the desired wheel speeds can be executed. Actually, this latter requires the publication of the odometer data for accounting the orientation resulting from the robot movements. However, prior to run the odometer function it is necessary to connect and give the permissions to Arduino to make it communicating with ROS.

Last but not least, to complete the launching process, the function sending the input to the motors ought be called in order to make the robot to move.



```

luca@luca-desktop: ~
bash: export: `2606:6000:61c2:2800:d4d0:39eb:ae69:105c': not a valid identifier
luca@luca-desktop:~$ sudo chmod a+rw /dev/ttyUSB0
[sudo] password for luca:
luca@luca-desktop:~$ roslaunch roscpp serial_node.py /dev/ttyUSB0
[INFO] [1601884535.061355]: ROS Serial Python Node
[INFO] [1601884535.091180]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1601884537.213387]: Requesting topics...
[INFO] [1601884537.550913]: Note: publish buffer size is 280 bytes
[INFO] [1601884537.566845]: Setup publisher on wheel_encoder [geometry_msgs/Vect
or3]
[INFO] [1601884537.588136]: Note: subscribe buffer size is 280 bytes
[INFO] [1601884537.598534]: Setup subscriber on cmd_vel [geometry_msgs/Vector3]

[INFO] [1601884539.210843346]: odom: left [0.000000], right [0.000000]
[INFO] [1601884539.211486939]: INPUT: left [1.836234], right [11.117444]
[INFO] [1601884539.290088554]: des: left [0.064431] right [0.358618]
[INFO] [1601884539.291482721]: INPUT: left [2.110112], right [13.089656]
[INFO] [1601884539.309987043]: odom: left [0.000000], right [0.000000]
[INFO] [1601884539.310802200]: INPUT: left [2.383945], right [15.062054]
[INFO] [1601884539.326072200]: Distance: [0.490516], angle: [4.983067], orienta
tion: [4.219963]
[INFO] [1601884539.379494075]: des: left [0.064408] right [0.358718]
[INFO] [1601884539.380393085]: INPUT: left [2.657214], right [17.037005]
[INFO] [1601884539.410602200]: odom: left [0.000000], right [0.000000]
[INFO] [1601884539.410947148]: INPUT: left [2.930948], right [19.009954]
[INFO] [1601884539.442104127]: Distance: [0.490161], angle: [4.986112], orienta
tion: [4.051338]
[INFO] [1601884539.478360273]: des: left [0.064448] right [0.358545]
[INFO] [1601884539.479175898]: INPUT: left [3.205659], right [20.978494]
[INFO] [1601884539.510158189]: odom: left [0.000000], right [0.000000]
[INFO] [1601884539.513384127]: INPUT: left [3.479565], right [22.950492]
[INFO] [1601884539.565176106]: Distance: [0.490323], angle: [4.988312], orienta
tion: [4.048347]
[INFO] [1601884539.578549960]: des: left [0.064430] right [0.358624]
[INFO] [1601884539.579148970]: INPUT: left [3.753027], right [24.924494]
[INFO] [1601884539.610118606]: odom: left [0.000000], right [0.000000]

```

Figure 5.14: Arduino node launched is showed on the terminal above, while the terminal on the bottom is printing all the four functions messages.

Arduino is made communicate with ROS by the mean of *roscpp* package [69] as *Figure 5.14* shows.

In order to simplify the operation, all the 4 functions cited above are executed in one launch operation. They print one message each and these message are displayed in the bottom window.

When the odometer function starts to work (at the beginning the robot is stuck, the speeds will be 0), the desired speeds are computed and shared. At this point the docking function can work by publishing the desired speeds. Thanks to the designed control system, the two inputs for the motors raise their value until the robot doesn't start to move and reach that speed. The reactivity of the raise strongly depends on the control system coefficients.

By the way, in the bottom figure one can see how the speeds are increasing in order to null the error between the actual speed and the desired one.

Chapter 6

Conclusion and Final results

The results of the project until this moment can be considered quite satisfying. In the simulation stage, the robot model is capable to reach the AR tag by following the docking algorithm that has been developed for it.

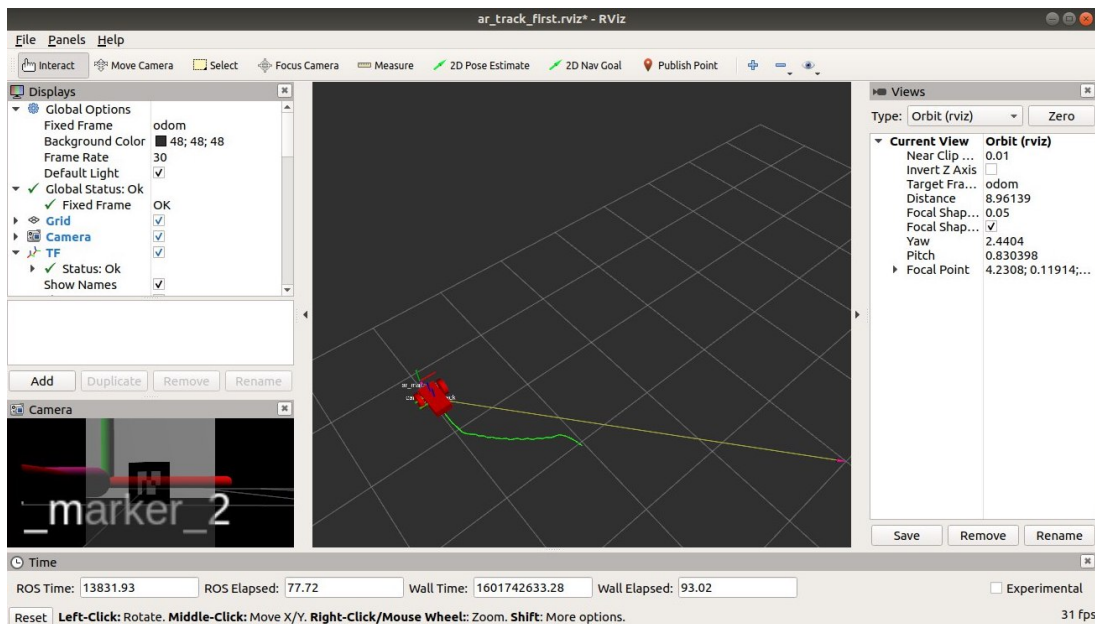


Figure 6.1: Trace showing the path the robot made during its docking operation on Rviz.

Figure 6.1 displays the trace the robot left after its passing within Rviz visualization. The trace is the path the robot followed by subscribing the velocity data coming from the docking algorithm functions. As one can notice from this path, it has broadly followed the strategy by approaching as first the perpendicular line to the tag and then by moving

straight towards it.

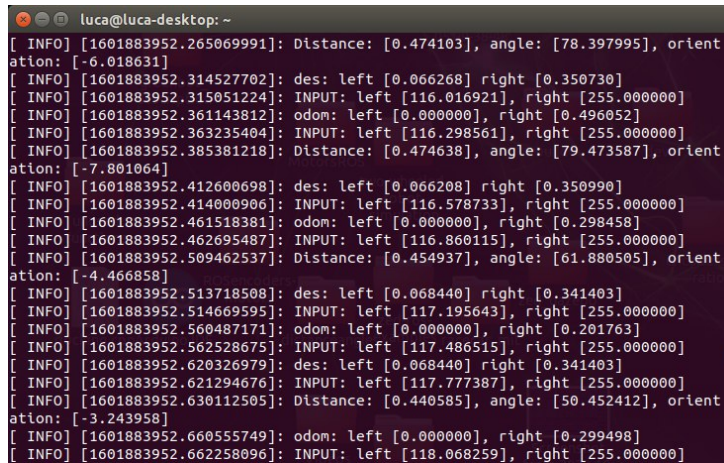
Then, the robot stops in correspondence of the tag (i.e. whenever the distance is lower than the minimum). The robot is now capable to charge itself as it is above the virtual charging station.

It is even worth noticing that the first phase of the strategy is not straight. This is easily imaginable since the desired orientation changes as the distance and angle decrease.

As far as the custom robot is concerned, we have obtained some results but many improvements are still needed.

ROS has been the right way to proceed even for the simulation and for the robot. It allows a very good communication between all the systems, from ZED to Jetson and from Arduino to Jetson. The message mechanism makes easy to communicate with each function connected with the ROS Master. Basically, the two steps, to connect a node to the master and to subscribe/publish messages, are made automatic by the means of this development system.

The encoders sensors data have been captured by the Arduino function and published on ROS. The control system takes this data and the desired speeds one and transmits the input to the robot and the motors answers accordingly thanks to Arduino-ROS communication.



```

luca@luca-desktop: ~
[ INFO] [1601883952.265069991]: Distance: [0.474103], angle: [78.397995], orient
ation: [-6.018631]
[ INFO] [1601883952.314527702]: des: left [0.066268] right [0.350730]
[ INFO] [1601883952.315051224]: INPUT: left [116.016921], right [255.000000]
[ INFO] [1601883952.361143812]: odom: left [0.000000], right [0.496052]
[ INFO] [1601883952.363235404]: INPUT: left [116.298561], right [255.000000]
[ INFO] [1601883952.385381218]: Distance: [0.474638], angle: [79.473587], orient
ation: [-7.801064]
[ INFO] [1601883952.412600698]: des: left [0.066208] right [0.350990]
[ INFO] [1601883952.414000906]: INPUT: left [116.578733], right [255.000000]
[ INFO] [1601883952.461518381]: odom: left [0.000000], right [0.298458]
[ INFO] [1601883952.462695487]: INPUT: left [116.860115], right [255.000000]
[ INFO] [1601883952.509462537]: Distance: [0.454937], angle: [61.880505], orient
ation: [-4.466858]
[ INFO] [1601883952.513718508]: des: left [0.068440] right [0.341403]
[ INFO] [1601883952.514669595]: INPUT: left [117.195643], right [255.000000]
[ INFO] [1601883952.560487171]: odom: left [0.000000], right [0.201763]
[ INFO] [1601883952.562528675]: INPUT: left [117.486515], right [255.000000]
[ INFO] [1601883952.620326979]: des: left [0.068440] right [0.341403]
[ INFO] [1601883952.621294676]: INPUT: left [117.777387], right [255.000000]
[ INFO] [1601883952.630112505]: Distance: [0.440585], angle: [50.452412], orient
ation: [-3.243958]
[ INFO] [1601883952.660555749]: odom: left [0.000000], right [0.299498]
[ INFO] [1601883952.662258096]: INPUT: left [118.068259], right [255.000000]

```

Figure 6.2: Prompt printing all the four main functions messages.

As one can notice from *Figure 6.2*, the input data are coherent with the desired speeds and the odometer data too. The robot is turning counterclockwise since its angle is positive. Even the robot parameters and in particular the angle are symbols that system is working properly, since the angle is decreasing as the robot is moving towards the perpendicular

line. We can say that the system is communicating and answering in a very good manner in this situation.

Unlike the simulation stage, here it is not possible to print the path in Rviz when using the ZED wrapper node, since, actually, in the Rviz visualization map the camera is stuck and, if the robot moves, it is the marker that approaches toward the camera and not the contrary.

However, the robot is not capable to accomplish a complete docking operation, losing completely the path. There are different causes to this behavior, from the control system to the low voltage range of input to the not full reliability on the encoders data. These all are explained in the next section within the improvements.

6.1 Improvements

Dealing with the prototype robot, decoupling the *zed_ar_track* node represents one possible improvement to make since one way to have the printed path as in the simulation can be to use two different nodes for the wrapping and the detection of tag.

Then, another limitation for the robot is represented by the analog input value varying the duty cycle of the PWM wave, or better the motors supplied voltage. The motors has a range between 3V and 8V, but the system provides only input around 3V, without taking advantage of the other voltages. This issue causes a low range of maneuverability, since the robot starts to move only after a value of 190/200(out of 255), that is too high for a control system. A good solution to tackle this problem is to add another battery pack in series to increase the controllability of the system.

Moreover, the control of the robot is even affected by the low reliability of the encoder sensors information, caused by the presence of slippages during the initial movements and from reading errors of the sensor. It is so needed to find a redundancy on this data or to make some additional software checks on them.

The improvements of both the encoders data reliability and the voltage range will allow to design a more performing control system.

Once these have been fixed during the continuation of the project, there are some features that can be implemented in order to improve the robot capabilities.

One of them is to find a way to keep the tag on the visualization autonomously and modifying the computation of the desired orientation accordingly. Another improvement to this system can be to add an object detection node in order to recognize obstacles during the path. This can be made by taking advantage of the many potentials the ZED camera have been provided.

The last step for this project is to make the robot completely autonomous by exploiting the thousands of features ROS is provided. In particular, the ambition is to master the robot by the mean of the ROS Navigation Stack package [70].

Bibliography

- [1] Robotics and robotics devices vocabulary, ISO 8373:2012 2.1,
<https://www.iso.org/obp/ui/iso:std:iso:8373:ed-2:v1:en>
- [2] Robotics and robotics devices vocabulary, ISO 8373:2012 2.10,
<https://www.iso.org/obp/ui/iso:std:iso:8373:ed-2:v1:en>
- [3] The History of Industrial Automation in Manufacturing,
<https://kingstar.com/the-history-of-industrial-automation-in-manufacturing/>
- [4] Kimon Roufas, Ying Zhang, Dave Duff, Mark Yim. *Six degree of freedom sensing for docking using IR LED emitters and receivers*. 7th Intl. Symp. On Experimental Robotics. 2000.
- [5] Yuta, S., Hada, Y. *Long term activity of the autonomous robot – Proposal of a benchmark problem for the autonomy*. 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, Victoria, pp. 1871-1878, October 1998.
- [6] R.C. Arkin and R. R. Murphy *Autonomous navigation in a manufacturing environment*. IEEE Transactions on Robotics and Automation, vol. 6, pp. 445-454, Aug. 1990.
- [7] B.W Minten, R.R. Murphy, J. Hyams, M. Micire *Low-order complexity vision-based docking*. IEEE Transactions on Robotics and Automation, vol. 17, iss. 6, pp. 922 – 930, Dec. 2001.
- [8] M.C. Silverman, D. Nies, B. Jung , G.S. Sukhatme *Staying alive: a docking station for autonomous robot recharging*. Proceedings of IEEE International Conference on Robotics and Automation(ICRA '02), vol. 1, pp. 1050 – 1055, May 2002.
- [9] Yanpeng Niu, Jirnei Zhang, Tianhua Meng, Hui Wang *Design of a home surveillance robot with self-recharging capabilities* 2010 3rd International Symposium on Knowledge Acquisition and Modeling.

- [10] N. Barnes, Zhi-Qiang Liu. *Fuzzy Control for Active Perceptual Docking*, The 10th IEEE International Conference on Fuzzy Systems, vol. 3, pp. 1531 – 1534, Dec. 2001.
- [11] Estimate on wireless chargins market.
<https://www.pikeresearch.com>
- [12] Xiao Lu, Ping Wang, Dong In Kim *Wireless Charging Technologies: Fundamentals, Standards, and Network Applications* IEEE Communication surveys & tutorials, vol. 18, No. 2, 2016
- [13] Bernhard Walzel, Christopher Sturm, J(ü)rgen Fabian, Mario Hirz *Automated robot-based charging system for electric vehicle* Institute of Automotive Engineering
- [14] Hendrik Kolvenbach, Marco Hutter *Life Extension: An Autonomous Docking Station for Recharging Quadrupedal Robots* Robotic Systems Laboratory, ETHZ, Leonhardstrasse 21, 8092 Zurich
- [15] WiTricity: Powering life, wirelessly.
<https://witricity.com>
- [16] Innovative mechatronic solutions for higher productivity.
<https://www.staubli.com/en-it/>
- [17] Staübli.
<https://en.wikipedia.org/wiki/Stäubli>
- [18] Staübli charging system installed for electric vehicles at Long Beach port.
<https://roboticsandautomationnews.com/2019/08/08/staubli-charging-system-installed-for-electric-vehicles-at-long-beach-port/24809/>
- [19] Kobuki.
<http://kobuki.yujinrobot.com>
- [20] Radio-frequency identification.
https://en.wikipedia.org/wiki/Radio-frequency_identification
- [21] What is GPS?
<https://www.geotab.com/blog/what-is-gps/>
- [22] Roomba® Robot Vacuum Cleaners — iRobot.
<https://www.irobot.com/roomba>

- [23] Chaomin Luo, Yu-Ting Wu, Mohan Krishnan, Mark Paulik, Gene Eu Jan and Jiyong Gao. *An Effective Search and Navigation Model to an Auto-Recharging Station of Driverless Vehicles* IEEE Global Engineering Education Conference, Istanbul, April 2014.
- [24] Roberto Quilez, Adriaan Zeemany, Nathalie Mitton, Julien Vandaele. *Docking autonomous robots in passive docks with Infrared sensors and QR codes* TRIDENTCOM 2015, June 24-25, Vancouver, Canada
- [25] K. Varun Raj, Kiran Patil, D.V. Kaushik Kariappa and Amit Madhav Jakati. *A Beacon-based Docking System for an Autonomous Mobile Robot* 13th National Conference on Mechanisms and Machines, IISc, Bangalore, India. December 12-13, 2007
- [26] Mamadou Doumbia, Xu Cheng, Vincent Havyarimana *An Auto-Recharging System Design and Implementation Based on Infrared Signal for Autonomous Robots* 2019 5th International Conference on Control, Automation and Robotics.
- [27] Ren C. Luo, Chung T. Liao, Kuo L. Su, Kuei C. Lin *Automatic Docking and Recharging System for Autonomous Security Robot*
- [28] ZED 2 Camera and SDK Overview.
<https://cdn.stereolabs.com/assets/datasheets/zed2-camera-datasheet.pdf>
- [29] Wikipedia: Image rectification.
https://en.wikipedia.org/wiki/Image_rectification
- [30] Wikipedia: Fundamental matrix (computer vision).
[https://en.wikipedia.org/wiki/Fundamental_matrix_\(computer_vision\)](https://en.wikipedia.org/wiki/Fundamental_matrix_(computer_vision))
- [31] iRobot Roomba - How to Dock Roomba for Charging.
https://www.youtube.com/watch?v=cCX-1KN_6GE
- [32] ROS: Introduction
<http://wiki.ros.org/ROS/Introduction>
- [33] ROS: Concepts
<http://wiki.ros.org/ROS/Concepts>
- [34] ROS: Topics
https://en.wikipedia.org/wiki/Robot_Operating_System#Topics

- [35] Gazebo, robot simulation made easy.
<http://gazebo-sim.org>
- [36] Gazebo plugins in ROS.
http://gazebo-sim.org/tutorials?tut=ros_gzplugins
- [37] Arduino Shields.
<https://en.wikipedia.org/wiki/Arduino#Shields>
- [38] What is Arduino?
<https://www.arduino.cc/en/Guide/Introduction>
- [39] Wiring, get on board.
<http://wiring.org.co>
- [40] Language Reference.
<https://www.arduino.cc/reference/en/>
- [41] Welcome to Processing 3.
<https://processing.org>
- [42] add AR tag in gazebo.
<https://www.youtube.com/watch?v=WDhIaVOUwsk&t=716s>
- [43] Gazebo: Building Editor.
http://gazebo-sim.org/tutorials?cat=build_world&tut=building_editor
- [44] robot_state_publisher.
http://wiki.ros.org/robot_state_publisher
- [45] joint_state_publisher.
http://wiki.ros.org/joint_state_publisher
- [46] ar_track_alvar.
http://wiki.ros.org/ar_track_alvar
- [47] tf/tfMessage Message.
<http://docs.ros.org/melodic/api/tf/html/msg/tfMessage.html>
- [48] ar_track_alvar/AlvarMarker Message.
http://docs.ros.org/fuerte/api/ar_track_alvar/html/msg/AlvarMarker.html

- [49] tf2.
<http://wiki.ros.org/tf2>
- [50] Conversion between quaternions and Euler angles.
https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
- [51] geometry_msgs/Twist message definition.
http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Twist.html
- [52] teleop_twist_keyboard
http://wiki.ros.org/teleop_twist_keyboard
- [53] NVIDIA Jetson Nano: Bringing the Power of Modern AI to Millions of Devices.
<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/>
- [54] Get your robot in gear with a Jetbot AI robot kit.
<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetbot-ai-robot-kit/>
- [55] NVIDIA-AI-IOT/jetbot.
<https://github.com/NVIDIA-AI-IOT/jetbot>
- [56] ARDUINO UNO REV3.
<https://store.arduino.cc/arduino-uno-rev3>
- [57] Arduino DC Motor Control Tutorial – L298N — PWM — H-Bridge.
<https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/>
- [58] ZED 2 - AI Stereo Camera — Stereolabs.
<https://www.stereolabs.com/zed-2/>
- [59] INIU Portable Charger, 20000mAh Dual 3A High-Speed Outputs.
<https://www.amazon.com/INIU-Portable-20000mAh-Flashlight-Compatible/dp/B07YCR7FR9>
- [60] USB Cable Combo Left & Right Angle Micro USB 5 Pin Male to USB 2.0.
<https://www.amazon.com/dp/B01N337FQF/>

- [61] Energizer Rechargeable AA and AAA Battery Charger (Recharge Pro) with 4 AA NiMH Rechargeable Batteries.
<https://www.amazon.com/Energizer-Rechargeable-Auto-Safety-Over-charge-Protection/dp/B00IM3P8GS/>
- [62] Geekcreit® DIY L298N 2WD Ultrasonic Smart Tracking Motor Robot Car Kit for Arduino.
<https://www.banggood.com/Geekcreit-DIY-L298N-2WD-Ultrasonic-Smart-Tracking-Moteur-Robot-Car-Kit-for-Arduino-products-that-work-with-official-Arduino-boards-p-1155139.html>
- [63] 3pcs LM393 DC 5V Optoelectronic Sensor PIR Sensor Module With LED Instruction.
<https://www.banggood.com/3pcs-LM393-DC-5V-Optoelectronic-Sensor-PIR-Sensor-Module-With-LED-Instruction-Slot-Single-Signal-p-1352781.html>
- [64] 120Pcs M3 Nylon Hex Spacers Screw Nut.
<https://www.amazon.com/Hilitchi-Spacers-Standoff-Accessories-Assortment/dp/B012G6E62I/>
- [65] 80 Piece Male to Female 4 and 8 Inch Solderless Ribbon Dupont-Compatible Jumper Wires
<https://www.amazon.com/GenBasic-Solderless-Dupont-Compatible-Breadboard-Prototyping/dp/B01L5UKAPI>
- [66] Two-wheeled robot project using ROS.
<https://forum.dronebotworkshop.com/ros/two-wheeled-robot-project-using-ros/>
- [67] nav_msgs/Odometry Message definition.
http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html
- [68] GitHub: zed-ros-examples package.
<https://github.com/stereolabs/zed-ros-examples>
- [69] roserial for Arduino/AVR platforms.
http://wiki.ros.org/roserial_arduino
- [70] Navigation Stack package.
<http://wiki.ros.org/navigation>