POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Autonomous Docking for an Automated Robot-based Charger for EVs with Obstacle Detection

Supervisors

Prof. Marcello CHIABERGE

Prof. Marina MONDIN

Candidate

Gianluca RAPAGLIA

December 2022

Summary

In a world moving towards automation, research and development of new solutions are becoming more and more important. Nowadays, electric vehicles are playing a key role in the development of new cutting-edge technologies in the automation field. In particular, the topic that has been discussed throughout the last 10 years, is the charging solutions for EVs. For this reason, Innotech Systems in Los Angeles, California, in collaboration with the Cal State University of Los Angeles, decided to invest in the development of an Automated Robot-based Charger for EVs. Their idea is to project a prototype while learning about this innovative technology that has the potential to simplify and automate certain aspects of life. The aim of this thesis is to design a charging station prototype using a robotic arm upon a detailed market investigation to understand what other companies all around the world are up to. The first two chapters will go through why an automated charging station is needed, similar projects carried out by other companies and all the theoretical background necessary to create such a system. Instead, in the last two chapters the main work will be presented going through both software and hardware components, and various tests and results will be discussed.

Acknowledgements

Ho lasciato questa parte di tesi per ultima, ma non perchè fosse la meno importante, anzi, giusto perchè in queste settimane di scrittura, organizzazione festeggiamenti mi è capitato spesso di navigare fra i ricordi condivisi con le persone che mi sono state accanto, per capire come ogni singolo step sia stato raggiunto grazie a voi. Non vi nascondo che, andando indietro e ripercorrendo questi 5 anni, molte volte mi è capitato di commuovermi ed accorgermi quanto sia affezionato a chi mi sta attorno.

Innanzitutto, vorrei iniziare ringraziando chi mi ha permesso di volare negli States e di lavorare su questo progetto, il prof. Marcello Chiaberge e chi mi ha permesso di svolgere il tutto, prof. Fred Daneshgaran, Kash Olia e la prof. Marina Mondin. Ringrazio due volte prof. Mondin per essere stata durante la mia esperienza all'estero la mia seconda mamma. I sorrisi, i consigli e la lasagna che tanto mi faceva sentire in Italia.

Vorrei ringraziare i miei genitori che durante questi 5 anni hanno fatto sempre completo affidamento su di me. "Mamma, Papà secondo me è meglio prendere questa strada", "Fai tu Già, hai sempre il nostro supporto". Mio fratello Flavio che anche con una chiamata per avere un consiglio riusciva sempre a farmi sorridere e a farmi tornare piccolo per un momento. A Zio Calo per le chiamate infinite e il piacevole Natale a Verona. A Patrizia, la mia ragazza ma anche la mia migliore amica, sì migliore amica, perchè tutto è nato così e lo è ancora oggi. Grazie per

avermi sostenuto nei momenti più difficili, per aver gioito (spesso più di me stesso) ai miei traguardi più importanti e per aver sempre creduto in me. Grazie anche a tutte le volte che mi hai fatto da babysitter, prima o poi ricambierò, lo giuro. Grazie a Peppe, conosciuto dai miei amici torinesi come Peppe Sessa. Ti conoscono tutti perchè è impossibile non parlare a chi ti sta intorno di un fratello. Se qualcuno dovesse raccontare tutta la mia vita quella persona saresti tu, ma ti prego non farlo ma continua a mandarmi "Buongiorno bro" tutti i giorni come fossi la tua fidanzata. Ad Alfonso per i compiti di latino che cominciavano sempre con un "Ou attruatu a versioni ndi internet?". A Michele per i bei traguardi festeggiati insieme e per esserci ritrovati nell'altra parte del pianeta partendo 5 anni fa da un paesino chiamato Avola. A Iano Big capace di farmi sorridere anche quando sto giù con una delle sue innumerevoli avventure. A Vincenzo e Riccardo, sono stato fortunato a conoscervi. Ricordo quando entrai in camera a Villa, Io: "Ma si sicilianu?" Riccardo: "Se" e la corsa ad abbracciarti perchè mi sentivo un pò più a casa. O quando abbiamo conosciuto Vincenzo in autobus in ritardo per la partita, incazzati, ma lui con il solito sorriso " Uè Wagliu" non consapevole di aver fatto 3 km a piedi verso una piazzuola deserta per raggiungere la fermata quando la fermata stava sotto casa. La nostra fratellanza è stata sancita a Madrid tornando da 3 nazioni diverse, alzate l'anello al cielo. A Carletto, per molti mio fratello, perchè uscire con lui e non essere presi per fratelli è diventato impossibile, fra se vuoi la famiglia Rapaglia è pronta ad accettarti. A Ciro per essere stato studente modello di siciliano e per avermi fatto da mental coach nella ricerca di un lavoro. Ad Antonio che invece di siciliano non capisce nulla, per guardarmi ogni volta che parlo siciliano come un serbo-croato guarderebbe un egiziano. A Giovanni, fra prima o poi ti porterò a Los Angeles. Ad Alessia per avermi insegnato come mantenere la calma in situazioni critiche grazie alle sue doti da infermiera e per mandarmi messaggi quando sto male. A Ragazzo per aver condiviso ogni

lezione e aver affrontato ogni esame insieme per 5 anni, ricordo ancora la sudata a freddo di Fisica 1. A Mattew, per avermi fatto capire che anche i fan di ciclismo si rompono guardando le gare. A Troiano, adesso so come combattere l'apatia. Ragazzi non voglio essere noioso e non voglio tenervi fermi ad ascoltare tutti i miei ricordi e pensieri anche perchè di storie da raccontare ne avrei troppe e di persone da ringraziare altrettante, vedi Simone per Vlahovic o Giggino per fare sempre brutte figure o Lorenzo, Alvin e Mattia per il surf e per il piacevolissimo road trip. Ma prima di terminare volevo rigraziare delle persone che nonostante la lontananza mi sono sempre state vicine e che anche con una chiamata sono sempre riuscite a tirarmi su. Grazie per tutte le candele accese prima di ogni esame o evento importante della mia vita, ogni preghiera perchè tutto andasse bene, ogni chiamata sperando sempre di vedermi e di abbracciarmi. E' per questo che dedico questo elaborato a voi.

Ai miei nonni,
Totò e Rosalba e a zia Angela.

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

ROS

Robotic Operating System

ARBC

Automated Robot-based Charger

DOF

Degrees of Freedom

EV

Electric Vehicle

DH

Denavit–Hartenberg

Chapter 1

Introduction

During the past ten years, the worldwide market for electric vehicles has expanded significantly, and in the coming years, an exponential growth is expected. Indeed, taking a look to the industry predictions for the incoming years, this is just the beginning. Even though 2020 did not show a growth in the electric vehicle market share due to the COVID situation, 2021 has broken the records. As a matter of fact, the number of EVs sold in a week was higher than EVs sold during 2012 [1].

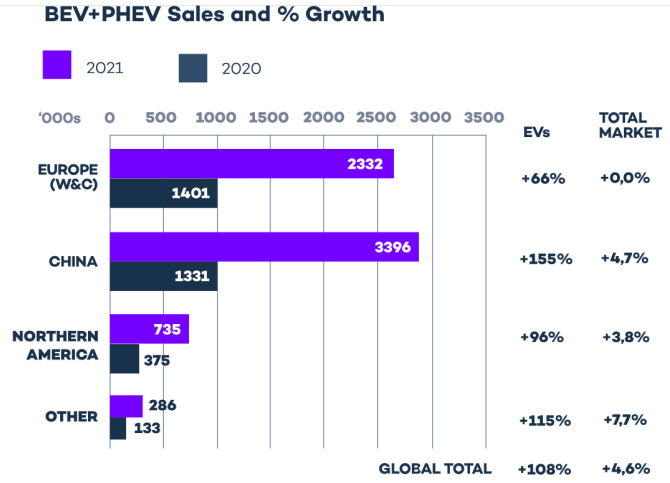

Figure 1.1: EVs sales and % growth

To this extent, the charging topic becomes a relevant aspect. In a world where autonomous driving and electric vehicles are growing day by day, a new charging technology has to be implemented. This is the idea carried out by InnoTech system company in collaboration with California State Univerisity of Los Angeles.

1.1 The Need for Automated Charger

One of the main reasons why the electric vehicles are not becoming so widespread is the waiting time to charge the vehicle. Indeed, for a high customer benefit, large driving ranges combined with quick recharge intervals are crucial. Obviously, the faster is the recharge the more powerful should be the charging station. In Fig. 1.2 is shown how many kilometers can be covered per hour load capacity and it is quite noticeable that the covered distance increases using more powerful charging stations [2]. The problem of such stations is the weight of the plug to be connected to the socket of the car to start the charging phase. Indeed, inductive systems are not able to provide such high charging capacities. Strong DC charging currents must be realized, hence the wire diameter must increase. In turn, this makes the cable bulky, inflexible, and difficult to handle, making it difficult for people to charge their vehicles. This is the main reason why automated charger stations are spreading rapidly nowadays, to make the charging procedure as easy as possible to human beings.

Vehicles and automated chargers provide fresh alternatives for the future. This method might provide a wealth of new services in a society that is striving to become as automated as possible. For instance, people with disabilities would not have to worry about traveling alone, and those running errands might keep their car running while they recharge it at the next automated charging station.

Throughout the last decade, several companies tried to design an Automated

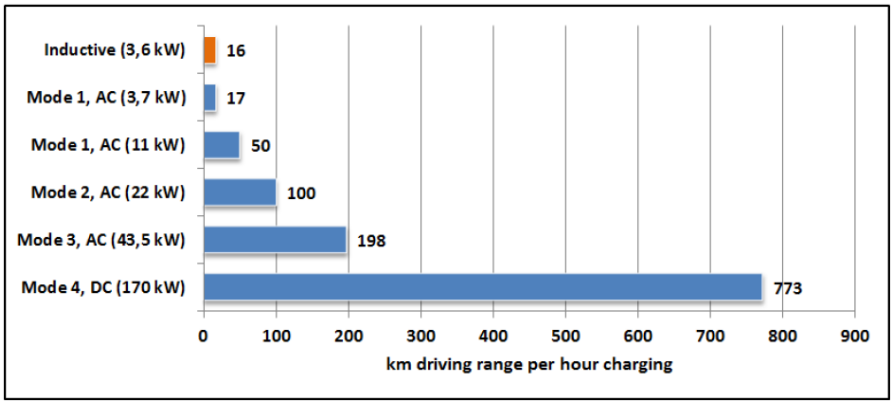

Figure 1.2: Kilometers covered per charging hour depending on the technology

Robot-based Charging station for EVs, however up until this point, no producer has introduced a series product to the market. Furthermore, all the solutions proposed by the companies are compatible just with their electric vehicles. Volkswagen company developed an Automated Robot-based Charger suitable for an e-Golf using a LBR iiwa manipulator by KUKA company with high charging capabilities. Seven driving axles on the robot and built-in torque sensors provide an accurate, force-sensing, and dependable connection. Basically, the charging station communicates with the car telling it exactly where to park to start the charging process. After that, a camera mounted on the top part of the robotic arm gripper, computes the goal to reach till the last centimeter. Finally, the software will tell the car to move on upon finishing the charging phase to allow other cars to be charged [3].

Also Tesla developed its own solution to the autonomous charging problem, coming over with a metal snake able to reach every position of the workspace with every pose. Though, the product has not been released on the market yet, so no further technical details have been published. The advantage is that all Tesla cars have the same position of the socket and plugs to charge the vehicles. This enables a certain degree of system adaptation to any Tesla vehicle [4].

Figure 1.3: Volkswagen solution to autonomous charging

ALanE is a project that the Technical University of Dortmund launched. Automated Charging System for Sustainable Electric Mobility is what the abbreviation stands for. The project's goal is to charge an electric car while it is parked without requiring the driver to put in the charging connection.

Figure 1.4: Tesla solution to autonomous charging

There is no need for the driver to manually intervene. As a result, as compared to known charging methods, the ALanE system significantly improves comfort, which stands out as a distinct selling feature when compared to conventionally powered automobiles. The TU Dortmund's charging system includes a stand-alone

energy source and a Wallbox that is upgraded with a reasonably priced and small connecting module. An app for a smartphone may be used to connect and disengage [2].

Figure 1.5: TU Dortmund solution to autonomous charging

1.2 Robot Manipulator

As shown by different projects developed by companies all around the world, to design an Automated Robot-base Charger (ARBC) the concept of Robot Manipulator (RM) becomes fundamental. A robot manipulator's mechanical structure is made up of a series of rigid bodies (links) joined by articulations (joints); a manipulator's arm enables movement, its wrist adds dexterity, and its end-effector completes the work demanded of the robot. The mobility is ensured by the presence of the joints used as connecting item between links. The joint can be either revolute or prismatic, and each of them generates one degree of freedom (DOF) if it comes to open kinematic chain structures (the one that will be used in this thesis work). The revolute one, from the word itself, gives to the links it is attached to the capability to rotate relatively, while the prismatic one allows a relative translational motion between them. As far as degrees of freedom are concerned, whenever a task requires

the robotic arm to be positioned in a certain position with a certain pose in the workspace of the robot itself, then six DOF's are required, three for positioning and three to orientate the end effector. However, the robot could have more DOF's than six, in such case the manipulator is known as redundant from a kinematic point of view. Another fundamental concept regarding RM is the workspace. Basically, it is the environment the manipulator can reach and it depends on the structure of the manipulator in terms of links and joints. As shown in Fig.1.6, the workspace of the anthropomorphic manipulator is a sphere cut at the base of the manipulator itself [5].

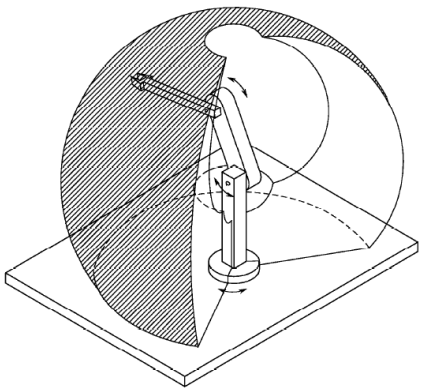

Figure 1.6: Workspace of an anthropomorphic robotic arm

This kind of manipulator is realized by three revolute joints allowing the robotic arm to be the most dexterous one. The second joint is known as the shoulder joint because to its resemblance to the human arm, and the elbow joint, which joins the "arm" to the "forearm," is known as the third joint.

Every task to be accomplished in the robotic field needs the execution of a specific motion. This can be done only if a mathematical model describing the robotic system and a complete knowledge about actuators and sensors is fully achieved. Therefore, the first important step to deal with is to model our robot manipulator in order to study and simulate all the different situation it could be involved into

the work environment.

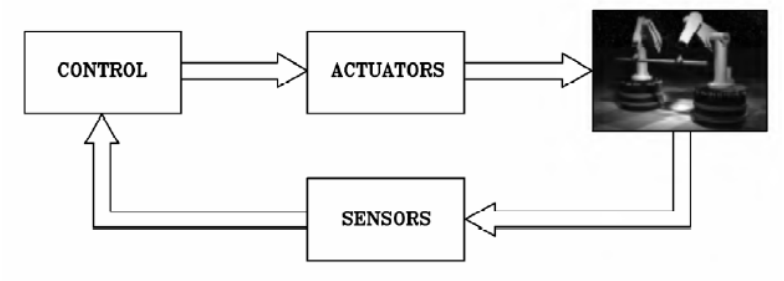

Figure 1.7: Components of a robotic system

Such a goal can be reached through different design phases: *Modelling, Planning* and *Control*.

Modelling Modelling is the first step of designing a robotic system, it should be performed through a kinematic analysis of the system to work with. Such analysis allows to study and solve two fundamental problems in robotics : the direct kinematics problem and the inverse kinematics one. The former is useful to understand, assigning a value in terms of angle or translation motion, depending on the kind of joint to be used, the prospective position in the space of the end-effector. The latter is exaclty the opposite: knowing a desired position the robotic arm is requested to reach, the joint values have to be defined. Either problems have to be solved using some linear algebra tools. Additionally, the kinematics analysis is the initial step in a thorough investigation of the manipulator's dynamics, enabling us to develop an effective control scheme and make the optimal actuator selections.

Planning Planning is of fundamental importance for a manipulator since a certain motion should be specified either to the joints or the end-effector. Obviously, it really depends on the kind of application the manipulator has to be used. If a material handling task is considered, just the pick-up and drop-off locations have to be specified and a point-to-point motion can be used. In the other hand, whenever the end-effector has to follow a desired path, a path motion technique has to be

used for trajectory planning. Furthermore, in presence of obstacles we address the problem as motion planning.

Control Control is fundamental for the manipulator to perform trajectories and to impose velocities and acceleration properties. Actuators and sensors must be used in order to realize the motion indicated by the control law. The motion control system of the mechanical structure uses the computed trajectories as reference inputs. Finding the timing behavior of the forces and torques that must be given by the joint actuators in order to guarantee the execution of the reference trajectories is the crux of the robot manipulator control problem.

1.2.1 Industrial Robotics

The field of industrial robotics, which deals with robot design, control, and applications, has advanced to the point that its products are now considered mature technologies. The main feature about industrial robots is to work in a structured environment whose primary a priori knowledge is of its geometrical or physical features. The first industrial robots were designed to manage the handling of radioactive materials in a remote fashion and they were characterized by:

- *versatility* : the robot had to be capable to adapt to different end-effectors depending on the task to be done,
- *adaptability* : the robot should have managed unknown situations by a clever usage of sensors,
- *positioning accuracy* : exploiting control techniques to stabilize the position of the end-effector,
- *execution repeatability* : if different operations have to be executed then a certain programmability has to be ensure.

Reduced manufacturing costs, increased productivity, improved product quality standards, and, last but not least, the potential to eliminate harmful or unappealing tasks for the human operator in a manufacturing system are the primary factors that have led to the spread of robotics technology in an increasingly wider range of applications in the manufacturing industry.

Industrial robots are stiff mechanical structures to ensure accuracy and repeatability, similar to human arms with wrist and the end-effector carrying out the work that usually a human hand should do. Usually, five or six DOF's are used for such a kind of robots, including proprioceptive joint sensors. As it works surrounded by human beings, the workspace has to be strictly safe and mostly of the time the working environment is well-known and quasi-static.

Figure 1.8: Industrial robots in a strictly safe environment

1.3 The Goal of the Thesis

As anticipated in the previous sections, the rise in the use of electric cars and society's desire to replace labor-intensive manual labor with automated systems have prompted several firms to invest money in the research for innovative charging solutions. Among these companies, Innotech Systems located in California, decided to invest in this prospective cutting-edge technology moving forward towards

automation and gaining experience in this field. In particular, the idea is to design a prototype of an automated charger for electric vehicles capable to start the charging phase of the EV in a fully autonomous fashion. A robotic arm is used to construct the automated charger. This arm must be able to recognize an AR tag using a depth camera and move towards its destination using a docking algorithm while detecting and avoiding obstacles. Numerous concerns must be addressed in order to create such an automated robot-based charger, and the task may be separated into several sections:

- Robotic Arm Design
- Path Planning
- Docking Algorithm
- AR Tag Identification through Depth Camera
- Hardware & Software Integration
- Obstacle Detection
- Obstacle Avoidance

As the robotic arm is already available, the first two bullets point do not need to be developed. As far as the other points, a docking algorithm is needed to decide how to reach the socket (goal) based on what the depth camera is able to detect in the surrounding environment. Indeed, it should be able to detect the goal, potential obstacles and to transmit the information to the software. The Hardware & Software Integration part is a key point as the software part enables the connection of different hardware parts to create a single usable system. Finally, the obstacle detection and avoidance will be used in both fashions, when the robot is steady and when it is moving. When the robot is steady the software is still

trying to make a decision about whether approaching or not the goal basing on the surrounding environment. In this part, both depth camera and ultrasonic sensors will be used. Instead, when the robot is moving the obstacle detection will rely just on the information given by the ultrasonic sensors.

1.4 The Organization of the Thesis

The thesis will be broken up into four chapters plus a fifth one for the conclusion, in an effort to evaluate every single project-related element. The first one is intended to analyze the general problem giving answers to the reader about the choice of developing such a project, going through the historical improvements and growth of this technology and a brief introduction about basic concepts behind it.
The second one is entirely devoted to the state of the art; first, a few actuators and sensors employed in the robotics sector will be described, and then it will go through the theoretical considerations involved in the development of such robotic systems.
The third chapter will explain the project carried out and the choices taken based on both adaptability to the available items and hardware components memory limitations. It will discuss the necessity for an additional DOF to widen the charging station's workspace as well as various geometrical and algebraic considerations regarding the established docking algorithm.
The fourth chapter, will resume the tests and results obtained while in the conclusion chapter potential future improvements that could be carried out, deepening the difficulties encountered and how they might be solved, will be addressed.

Chapter 2

State of the Art

The whole background theory necessary to comprehend the project undertaken and some current methodologies will be covered in this chapter. But first, it will explore the core sources of robotics sensors and actuators. Indeed, every robotic system must have sensors and actuators. The first ones make it possible for the program and the designer to both obtain measurement data on a certain quantity or characteristic that has to be controlled. Each control loop anticipates the presence of a number of sensors in order to stabilize a certain aspect of the system. Actuators are the actual mechanical components that allow a robotic system to move. During the process of the robotic system's operation, the control system must select what input to give to such actuators at each moment. Finally, the robotics concepts already involved in the robotic arm that the project is based on will be developed, the theory inherent obstacle recognition by the depth camera (Point Cloud and K-d Tree theory), and finally the concepts underlying a Software Processes Analysis to prevent any race condition issues.

2.1 Actuators and Sensors

In this section, two fundamental basic elements in robotics are covered: *actuators* and *sensors*. The first ones make it possible for the robot to move and steady it, which prevents the structure from shaking. They are only considered joints in the kinematic analysis, but if the robot is to be evaluated dynamically, their weight must be taken into account as part of the structure. On the other hand, the information supplied by sensors enables to determine if the motion produced by the robot using a well designed control architecture is the one required or not. It is important to classify the sensors into *proprioceptive* sensors that provide to the user information about the internal state of the robot (joint position, velocity, torques) helping out to understand whether the system is tracking the desired behaviour or not, and *exteroceptive* sensors providing the user information about the surrounding environment (force sensors, vision sensors, etc.).

2.1.1 Motors

An actuator is a device that transforms energy and signals into the system to generate motion. It may generate either rotational or linear motion. As the name suggests, linear actuators generate linear motion. This means that linear actuators have a fixed forward or backward travel distance on a linear plane before they must come to a halt. On the other side, rotary actuators generate rotational motion, which involves the actuator rotating on a circular plane. In the world of robotic manipulator rotary actuators are more spread. In particular, *servomotors* are the most used ones and depending on the kind of input in terms of power, they can be classified into three different groups:

- *Pneumatic motors* making use of the pneumatic energy supplied by a compressor and convert it into mechanical energy using pistons and turbines.

- *Hydraulic motors* which, using the appropriate pumps, convert the hydraulic energy contained in a reservoir into mechanical energy.
- *Electric motors* whose the principal source of energy is the electricity provided by the electrical distribution system.

They should all adhere to certain basic standards. In fact, they should be able to create large accelerations while having a low inertia to minimize their impact on the mechanical structure. Additionally, they must have a broad velocity range and great placement precision.[5]

Figure 2.1: Servomotors used in robotics

2.1.2 Stereo camera

Stereo camera, unlike standard cameras, is able to provide fundamental information such as depth of a detected object. From the definition of Stereo camera : *"A camera with two or more lenses and a separate image sensor or film frame for each lens is called a stereo camera."* By simulating human binocular vision, stereo photography, a technique that enables the camera to record three-dimensional

images, is made possible [6]. Obviously, the resolution of the cameras and the lighting conditions have an impact on how well measurements are performed. Even though they are directly related to the computational weight, it offers a ton of functions that allow the user to see the objective to be reached or even identify obstacles to avoid along the road. In fact, such a camera is capable of generating a point cloud of the observed objects and recognizing pre-established pictures using a fairly computationally intensive machine learning process. Nowadays, stereo camera is becoming more and more famous, mostly for self-driving. Indeed, the Tesla CEO Elon Musk, during an interview highlighted the capability of stereo camera of having a clear picture of what is going on in the surrounding environment, outlining the superiority over the LiDAR technology for self-driving applications as camera describes exactly the way we perceive reality.

Figure 2.2: Stereo camera

2.1.3 Ultrasonic Sensor

A widely used sensor in robotic field is the ultrasonic sensor. It is a piece of technology that uses ultrasonic sound waves to detect a target object's distance and then turns the sound that is reflected back into an electrical signal. The speed of audible sound is greater than the speed of ultrasonic waves (i.e. the sound that humans can hear).

Figure 2.3: Ultrasound sensor

The transmitter (which generates sound using piezoelectric crystals) and the receiver are the two major parts of an ultrasonic sensor (which encounters the sound after it has travelled to and from the target) [7]. Whenever one wants to use this sensors to compute the distance from an object, the equation 2.1 taking into account the speed of the sound should be considered. The distance D can be computed as follows:

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

where:

- T is the time for the sound to bounce back.
- C is the sound speed 343 m/s.

It is worth noticing that this kind of sensor does not provide information about the position in the space of an object but just how close it is. This aspect is clearly understandable from Fig.2.4 taking a look at the propagation of sound waves. In fact, it is frequently employed to provide the robot the ability to determine if it will collide with an item on the way to the destination. Furthermore, ultrasonic sensors in proximity sensing applications are less prone to interference from smoke, gas, and other airborne particles than infrared (IR) sensors are (though the physical components are still affected by variables such as heat).

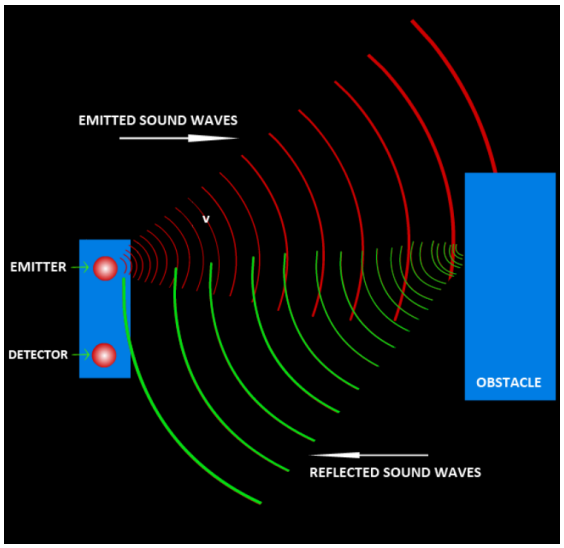

Figure 2.4: Ultrasound sensor functioning

2.2 Kinematics

As introduced in the previous chapter, a robotic manipulator can be fully described by a kinematic chain. This is crucial to study and simulate the system to work with and to control. Indeed, the kinematic analysis represent the first step towards the software simulation of such a system and the development of a dynamic model itself. In particular, this chapter will go through the general approach based on linear algebra to derive the *direct kinematics equation*. By means of such equations, it is possible to compute, given the joint angles of the manipulator, the final position and orientation of the end-effector. Then, the *inverse kinematic* problem will be analyzed, a key concept to understand, knowing a position in the space the robotic arm has to reach with a certain pose, which joint angles to assign. It is worth noting that in order to go through all these concepts an efficient way to represent the position in the space of the end-effector should be presented. To this extent, *Euler angles* and *Quaternion* will be introduced in order to represent the robotic tip pose in the space. Furthermore, a systematic way to derive the direct kinematic equations of the robot is to be defined. For this purpose, the *Denavit-Hartenberg Convetion* is introduced, enabling to define the relative position and orientation of two consecutive links and recursively to compute the kinematic equations.

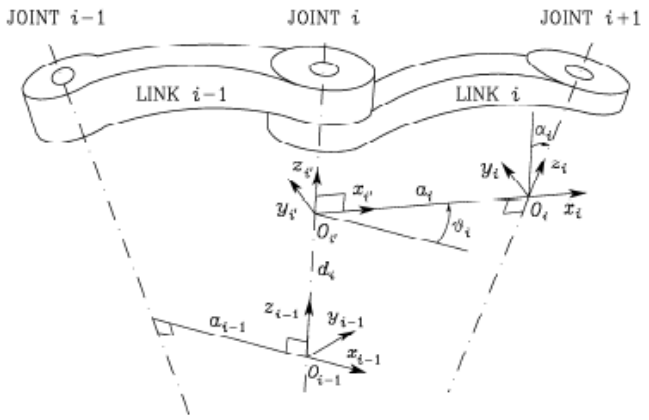

Figure 2.5: DH convention kinematic parameters

Taking into account that Axis i in Fig.2.5 is the axis connecting *Link i-1* to *Link i*, *Frame i* can be described exploiting DH convention as follows:

- Select axis z_1 along *Joint i-1* axis.
- Find the origin O_i at the point where axis z_i and the common normal to axis z_{i-1} and z_i intersect. Additionally, locate the origin $O_{i'}$ at the intersection of axis z_{i-1} and the common normal.
- Select the axis x_1 along the common normal to axes z_{i-1} and z_1 with direction from *Joint i* to *Joint i+1.*
- Finally, select y_i axis to complete the right-handed frame.

Then, once the link frames have been chosen, the position and orientation of *Frame i* with respect to *Frame i-1* can be fully specified using the following parameters

- a_i distance between O_i and $O_{i'}$
- d_1 coordinate of $O_{i'}$ along z_{1-1}.
- α_1 angle between z_{i-1} and z_1 about axis x_1.
- θ_1 angle between x_{i-1} and x_1 about axis z_1.

Among the parameters described above, some of them are always constant as they depend just on the geometry of the connection between the joints. Indeed, a_i and α_1 will always depend on the geometry of the robot, while the variable in our representation will be just one between θ_1 and d_1. It will be the former if the joint is *revolute* while it will be the latter if the joint is *prismatic.*
In order to mathematically describe the relation between the different frames, a matrix representation can be exploited. In particular, roto-translation matrices are to be used such that

$$A_i^{i-1}(q_i) = \begin{pmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\theta_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

where c_{θ_i} and s_{θ_i} are respectively, the cosine and the sine of the theta variable, while q_i is the joint variable.

2.2.1 Euler Angles and Quaternion

Here the problem of finding an efficient way to represent the attitude in the space of the end-effector is going to be addressed. Indeed, rotation matrices have nine parameters that can be reduced to three as they are related on each other. To this aim, *Euler angles* can be introduced as three independent parameters $\phi = [\varphi\, \theta\, \psi]^T$ constituting a minimal representation of the attitude of a rigid body in the space. There are different types of Euler Angles used nowadays, but the most used one is the *Roll Pitch Yaw* defining the *RPY Euler Angles*.

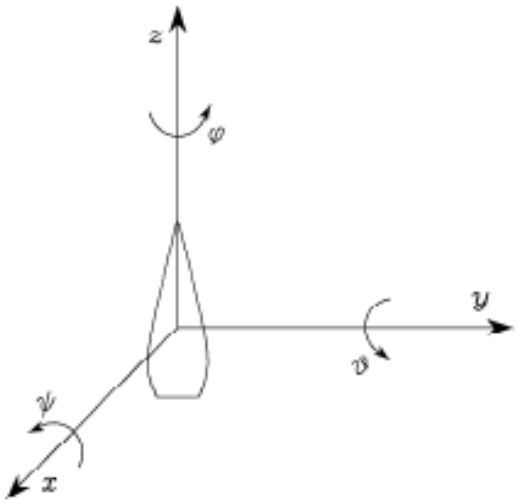

Figure 2.6: Representation of Roll Pitch Yaw angles

For such a representation, a rotation matrix can be computed taking into account the three indipendent paramemters

$$R(\phi) = R_z(\varphi)R_y(\theta)R_x(\psi) \tag{2.3}$$

and computing the multiplication matrix by matrix it is possible to get the following general matrix form

$$R(\phi) = \begin{pmatrix} c_\varphi c_\theta & c_\varphi s_\theta s_\psi - s_\varphi c_\psi & c_\varphi s_\theta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\theta & s_\varphi s_\theta s_\psi + c_\varphi c_\psi & s_\varphi s_\theta c_\psi - c_\varphi s_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{pmatrix}$$

As stated before, another way to represent the attitude of a rigid body with a minimal representation (using just three parameters) is through *quaternions*. A *quaternion* is a vector composed by a real part q_0 representing the magnitude of the rotation, and by a vectorial part $[q_1\ q_2\ q_3]$ representing the vector around which the rotation has to happen. The base of the 4D linear space in which the quaternions are defined is $[1\, i\, j\, k]$, where $i\ j\ k$ are *hypercomplex* numbers satisying the following *anticommutative* multiplication rules

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = -ji = k$$
$$jk = -kj = i$$
$$ki = -ik = j$$

A quaternion can be defined as a linear combination of the base as follows

$$q = q_0 1 + q_1 i + q_1 j + q_2 k \tag{2.4}$$

Noting that the following formulas can always be used to compute the corresponding quaternion whenever a rotation matrix is available

$$q_0 = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}} \tag{2.5}$$

$$q_1 = \frac{1}{2}\, sgn(r_{32} - r_{23})\sqrt{1 + r_{11} + r_{22} + r_{33}} \tag{2.6}$$

$$q_2 = \frac{1}{2}\, sgn(r_{13} - r_{31})\sqrt{1 - r_{11} + r_{22} - r_{33}} \tag{2.7}$$

$$q_3 = \frac{1}{2}\, sgn(r_{21} - r_{12})\sqrt{1 - r_{11} - r_{22} + r_{33}} \tag{2.8}$$

where *sgn* is the sign function and $r_{i,j}$ is the element of the matrix already shown in 2.3.

2.2.2 Kinematics of Anthropomorphic Robot Arm

As anticipated, this subsection will go through the derivation of the *direct kinematics equations* useful to describe the end-effector pose and attitude in the space. Every study addressing robotics has to deal with this fundamental part. In this case, the anthropomorphic arm is to be study as it will be exploited throughout this paperwork.

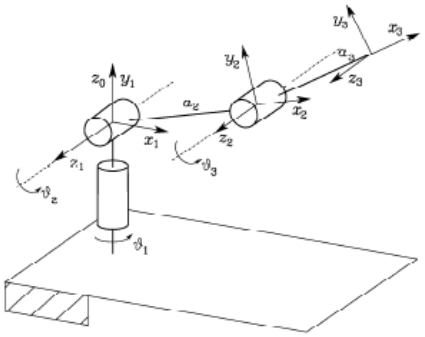

Figure 2.7: Anthropomorphic arm

To this aim, the Denavit-Hartenberg convention, that has been already presented, will be used. The anthropomorphic arm is modeled after a human arm and is made

up of three joints connected by regular links. In fact, the shoulder, elbow, and wrist joints may be seen by looking at Fig.2.7 from the ground up, following the structure. Therefore, it is useful to spot the four fundamental parameters of the DH convention to compute the direct kinematics function. The correct choice for such parameters is the one shown in Table 2.1.

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

Table 2.1: DH parameters for anthropomorphic arm

The computation of the matrices for each link, as stated by 2.2, yields to the direct kinematics function:

$$T_3^0(q) = A_1^0 A_2^1 A_3^2 = \begin{pmatrix} c_1c_{23} & -c_1s_{23} & s_1 & c_1(a_2c_2 + a_3c_{23}) \\ s_1c_{23} & -s_1s_{23} & -c_1 & s_1(a_2c_2 + a_3c_{23}) \\ s_{23} & c_{23} & 0 & a_2s_2 + a_3s_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

In this case, having just revolute joints in the mechanical structure of the robotic arm, among the parameters the variable is always θ_i.
As it will be discussed later, it is worth to describe the *spherical wrist* kinematic structure made up by three revolute joints whose rotation axis intersect in one single point. This last feature is what makes the wrist *"spherical"*.

Link	a_i	α_i	d_i	θ_i
4	0	$-\pi/2$	0	θ_4
5	0	$\pi/2$	0	θ_5
6	0	0	d_6	θ_6

Table 2.2: DH parameters for spherical wrist

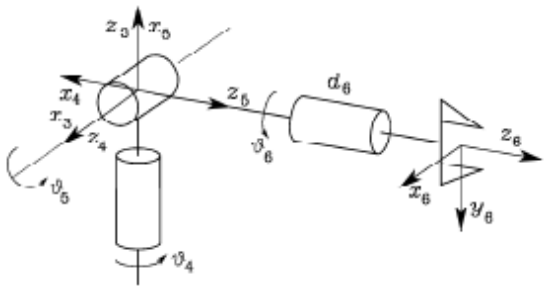

Figure 2.8: Spherical wrist

As before, also in this case it is possible to compute the direct kinematics equation by means of homogeneous matrices as follows

$$T_6^3(q) = A_4^3 A_5^4 A_6^5 = \begin{pmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 & c_4s_5d_6 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 & s_4s_5d_6 \\ -s_5c_6 & s_5c_6 & c_5 & c_5d_6 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.10}$$

It is useful to think to attach the spherical wrist to the end-effector of the anthropomorphic arm presented before. This feature would allow to get really nice features either for dexterity and solvability of the *inverse kinematic problem*.

2.2.3 Inverse Kinematic Problem

As it has been discussed so far in Section 2.2, direct kinematics equation allow to establish a relation between the joint variables in the *joint space* and the end-effector position variables in the *operational space*. Here the problem of determining the joint variables with the knowledge of the end-effector position in the space will be addressed. This is what *inverse kinematics* is all about. Basically, it answers the question "Which values should I assign to reach that particular position with the end-effector?". It is not trivial to go through the solution of such a problem for many reasons:

- There might be infinite solutions to a particular end-effector pose in the space, it is the case of *redundant manipulators*.
- No admissible solution is available studying the kinematic structure of the manipulator.
- Generally, nonlinear equations is to be solved implying sometimes no *closed-form solution* for such a problem.

There are several approaches to come up with a solution for the inverse kinematics problem. An inverse kinematic problem solver should have a complete knowledge of the mechanical structure of the manipulator in order to have some *geometrical/algebraic intuition* to get a closed-form solution. In the majority of applications, *numerical solution techniques* are used even though they do not allow the computations of all possible solutions to the inverse problem.
In this subsection a possible algebraic solution for the anthropomorphic arm will be assessed. As anticipated before, the goal is to determine the variables $\theta_1, \theta_2, \theta_3$ once the position of the end-effector to be reached in the operational space is chosen. Before starting with some algebraic consideration taking to the final solution, it is worth defining the position of the end-effector into the base frame of the anthropomorphic arm shown in Fig.2.7. Hence, it follows

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = T_3^0(q)\, \vec{p_3} \tag{2.11}$$

where $\vec{p_3}$ is the position that is to be reached in the operational space of the arm expressed in the end-effector Frame 3, while $\vec{p}$ is the same position but expressed in the base Frame 0. Since the problem is to position the end-effector into a particular point in the space let the $\vec{p_3}$ vector be

$$\vec{p_3} = \vec{i_3} + \vec{j_3} + \vec{k_3} \tag{2.12}$$

From 2.11 it follows

$$p_x = c_1(a_2c_2 + a_3c_{23}) \tag{2.13}$$

$$p_y = s_1(a_2c_2 + a_3c_{23}) \tag{2.14}$$

$$p_z = -a_2s_2 + a_3s_{23} \tag{2.15}$$

Now, as introduced before some algebraic techniques will be applied. First, squaring and summing 2.13, 2.14, 2.15 yields to

$$p_x^2 + p_y^2 + p_z^2 = a_2^2 + a_3^2 + 2a_2a_3c_3 \tag{2.16}$$

where just the c_3 unknown appears, and so

$$c_3 = \frac{p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{2.17}$$

and knowing that c_3 has always to hold the following inequalities $-1 \leq c_3 \leq 1$, it follows that

$$s_3 = \pm\sqrt{1 - c_3^2} \tag{2.18}$$

and this yields to the computation of the first unknown joint variables θ_3

$$\theta_3 = Atan2(s_3, c_3) \tag{2.19}$$

that for the well-known property of angles yields to two different solutions

$$\theta_{3,I} \in [-\pi, \pi] \tag{2.20}$$

$$\theta_{3,II} = -\theta_{3,I} \tag{2.21}$$

With the knowledge of θ_3, the computation of θ_2 can be addressed considering 2.13 and 2.14

$$p_x^2 + p_y^2 = (a_2 c_2 + a_3 c_{23})^2 \tag{2.22}$$

that yields to

$$a_2 c_2 + a_3 c_{23} = \pm\sqrt{p_x^2 + p_y^2} \tag{2.23}$$

and the solution coming out from 2.23 and 2.15, taking into account the two different possible values that θ_3 can assume is

$$c_2 = \frac{\pm\sqrt{p_x^2 + p_y^2}(a_2 + a_3 c_3) + p_z a_3 s_3}{a_2^2 + a_3^2 + 2a_2 a_3 c_3} \tag{2.24}$$

$$s_2 = \frac{p_z(a_2 a_3 c_3) \pm \sqrt{p_x^2 + p_y^2} +}{a_2^2 + a_3^2 + 2a_2 a_3 c_3} \tag{2.25}$$

and finally, as before for θ_3 the two possible solution can be computed for θ_2 as well

$$\theta_{2,I} = Atan2((a_2 + a_3 c_3)p_z - a_3 s_3^+ \sqrt{p_x^2 + p_y^2}, \tag{2.26}$$

$$(a_2 + a_3 c_3)\sqrt{p_x^2 + p_y^2} + a_3 s_3^+ p_z) \tag{2.27}$$

$$\theta_{2,II} = Atan2((a_2 + a_3 c_3)p_z + a_3 s_3^+ \sqrt{p_x^2 + p_y^2}, \tag{2.28}$$

$$-(a_2 + a_3 c_3)\sqrt{p_x^2 + p_y^2} + a_3 s_3^+ p_z) \tag{2.29}$$

where these two are the solutions picking s_3^+ as stated in 2.18, and

$$\theta_{2,III} = Atan2((a_2 + a_3c_3)p_z - a_3s_3^- \sqrt{p_x^2 + p_y^2}, \tag{2.30}$$

$$(a_2 + a_3c_3)\sqrt{p_x^2 + p_y^2} + a_3s_3^- p_z) \tag{2.31}$$

$$\theta_{2,IV} = Atan2((a_2 + a_3c_3)p_z + a_3s_3^- \sqrt{p_x^2 + p_y^2}, \tag{2.32}$$

$$-(a_2 + a_3c_3)\sqrt{p_x^2 + p_y^2} + a_3s_3^- p_z) \tag{2.33}$$

that are the solutions picking s_3^-.

Finally, the θ_1 computation is possible after easily managing 2.23, 2.13 and 2.14

$$\theta_{1,I} = Atan2(p_y, p_x) \tag{2.34}$$

$$\theta_{1,II} = Atan2(-p_y, -p_x). \tag{2.35}$$

It is worth noticing that from 2.35 the following two solutions are possible

$$\theta_{1,II} = \begin{cases} Atan2(p_x, p_y) - \pi & p_y \geq 0 \\ Atan2(p_x, p_y) + \pi & p_y < 0. \end{cases} \tag{2.36}$$

The algebraic procedure studied so far, allows to come up with four different solutions to the inverse kinematics problem as follows:

$$(\theta_{1,I}, \theta_{2,I}, \theta_{3,I})\ (\theta_{1,I}, \theta_{2,III}, \theta_{3,II})\ (\theta_{1,II}, \theta_{2,II}, \theta_{3,I})\ (\theta_{1,II}, \theta_{2,IV}, \theta_{3,II}) \tag{2.37}$$

The robotic arm different poses, depending on the chosen solution, are illustrated in Fig.2.9.

2.3 Trajectory Planning

This section will go through the *trajectory planning* allowing the robotic arm to receive the correct inputs in order to reach a fixed point in the space with some

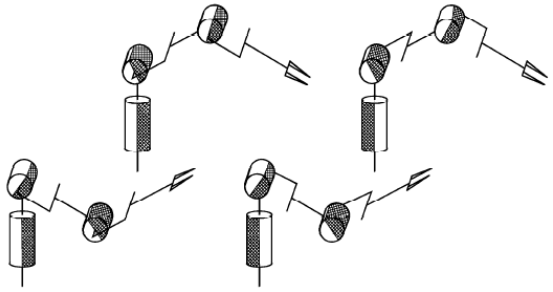

Figure 2.9: Four possible solutions to the inverse kinematics problem for anthropomorphic arm

predefined features. There are different techniques to accomplish this aim. Two of them are *point-to-point motion* and *motion through a sequence of points*. Just the first one will be addressed throughout this section.

Whenever a robot has to reach a final assigned posture, saturation limits have to be taken into account. There could me a limitation either on the applied force or on the velocity that the robotic arm can reach. Furthermore, the trajectory planning should take into account possible obstacles or path description. As the actuators send inputs directly to the joints, it is easier to operate into the joint space. Nevertheless, mostly of the times, whenever a particular path to follow has to be specified, operational space is an easy way to express it. To this purpose, an inverse kinematics problem, as described in the previous section, can be addressed to understand which values the joints have to move to.

Furthermore, the user has the chance to choose intermediate points of the trajectory, mostly of the times to express the maximum reachable velocity/acceleration. It is worth noticing that having smooth trajectories is better for the sake of stability of the robotic arm itself.

2.3.1 Trapezoidal Velocity Profile

As anticipated before, the point-to-point motion will be discussed, focusing first on the trajectory generated solving an optimization problem and then on the *trapezoidal velocity profile* widely used in industry field nowadays.
In particular, in the problem that will be deepen, the robotic arm has to move from an initial to a final point. Obviously, such a problem would take to an infinite number of solutions. To avoid this problem, an optimization problem can be solved and the optimization variable can be freely chosen by the user. In this case, the torque τ is chosen as optimization variable to design a trajectory of a body with inertia I moving from an initial angle q_i to a final one q_f.
From the Newton's second law of motion it follows that

$$I\dot{\omega} = \tau \tag{2.38}$$

and since the robotic arm has to move from q_i to q_f

$$\int_{q_i}^{q_f} \omega(t)\, dt = q_f - q_i \tag{2.39}$$

such that the following expression is minimized

$$\int_0^{t_f} \tau^2(t)\, dt. \tag{2.40}$$

The solution of the proposed optimization problem is

$$\omega(t) = at^2 + bt + c. \tag{2.41}$$

It follows that in the joint space the following *cubic polynomial*

$$q(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \tag{2.42}$$

resulting into a parabolic velocity profile

$$\dot{q}(t) = 3a_3t^2 + 2a_2t + a_1 \tag{2.43}$$

and into a linear acceleration profile

$$\ddot{q}(t) = 6a_3t + 2a_2. \tag{2.44}$$

From the equations just described, four parameters have to be fixed. Having two conditions to impose on the initial and final joint angle, other two constraints have to be chosen. Usually, a typical choice is to set the initial and final velocity to zero, $\dot{q}_i(t) = \dot{q}_f(t) = 0$. This choice takes to the following system of equation allowing to determine the trajectory unknow parameters

$$a_0 = q_i \tag{2.45}$$

$$a_1 = \dot{q}_i \tag{2.46}$$

$$a_3t_f^3 + a_2t_f^2 + a_1t_f + a_0 = q_f \tag{2.47}$$

$$3a_3t_f^2 + 2a_2t_f + a_1 = \dot{q}_f \tag{2.48}$$

If a trapezoidal velocity trajectory has to be employed, imposing a cruise velocity, a constant acceleration in the beginning phase and a constant deceleration in the final phase of the trajectory, the optimization variable has to be taken off. In this case the problem would be addressed imposing

$$\ddot{q}_c t_c = \frac{q_m - q_c}{t_m - t_c} \tag{2.49}$$

where q_m and t_m are the joint variable and time at the average point of the trajectory, while $\ddot{q}_c, q_c, t_c$ are the acceleration, joint angle and time when the cruise velocity is reached. Being the position trajectory parabolic, it follows that

$$q_c = q_i + \frac{1}{2}\ddot{q}_c t_c^2. \tag{2.50}$$

Considering 2.49 and 2.50, it yields to

$$\ddot{q}_c t_c^2 - \ddot{q}_c t_f t_c + q_f - q_i = 0 \tag{2.51}$$

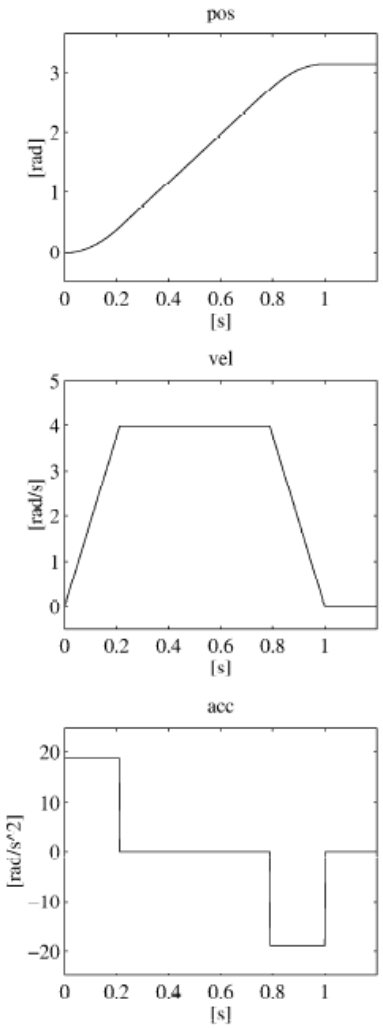

Figure 2.10: Position, velocity and acceleration generated trajectories for the analyzed case

and assigning $\ddot{q}_c$ with the following constraint

$$sign(\ddot{q}_c) = sign(q_f - q_i) \tag{2.52}$$

and, as anticipated, giving t_f, q_i and q_f it is possible to find the relation

$$t_c = \frac{t_f}{2} - \frac{1}{2}\sqrt{\frac{t_f^2\ddot{q}_c - 4(q_f - q_i)}{\ddot{q}_c}}. \tag{2.53}$$

The acceleration is then subject to the following inequality

$$|\ddot{q}_c| \geq \frac{4|q_f - q_i|}{t_f^2}. \tag{2.54}$$

Assigning t_f, q_i, q_f and a desired $\ddot{q}_c$ and solving 2.53 and 2.54, it yields to the following position trajectory

$$q(t) = \begin{cases} q_i + \frac{1}{2}\ddot{q}_c t^2 & 0 \leq t \leq t_c \\ q_i + \ddot{q}_c t_c(t - \frac{t_c}{2}) & t_c < t \leq t_f - t_c \\ q_f - \frac{1}{2}\ddot{q}_c(t_f - t)^2 & t_f - t_c < t \leq t_f. \end{cases} \tag{2.55}$$

Just to give an example and to show the trajectories in function of the time, some random values can be assigned to see what outcome will be obtained. Assigning $[q_i, q_f, t_i, t_f, \ddot{q}_c] = [0, \pi, 0, 1, 6\pi]$, and solving the equations described above, the result is the following

$$t_c = 0.12$$

$$q(t) = \begin{cases} 3\pi t^2 & 0 \leq t \leq 0.12 \\ 0.72\pi(t - 0.06) & 0.12 < t \leq 0.88 \\ \pi - 3\pi(1 - t)^2 & 0.88 < t \leq 1. \end{cases}$$

and the produced trajectories of position, velocity and acceleration are shown in Fig.2.10.

2.3.2 Jerk Controlled Motion

As described in the previous subsection, constant acceleration profiles are widely used in industry nowadays for real-time applications. This part of the work will explain the benefits of using a *jerk controlled motion*. This technique is also called *S-curve motion planning* and it turns out to have better performances than constant acceleration. Thinking about it and looking at Fig.2.10, it is worth noticing the "jump" that the acceleration trajectory has to do to pass from zero to the maximum value [8]. This feature reflect on the performances of the robotic arm in terms of vibration and accuracy. Indeed, with an S-curve profile, the robotic arm has the chance to reach the final position in a smoother way. Among the benefits, some of them are:

- Better surface quality - without abrupt changes in speed that cause momentary micro-positional problems.
- Less chatter and skip due to decreased machine vibration and resonance.
- Smoother and cooler motor operation.
- Steadier movements for precision operations.

Although numerous studies on s-curve motion profiles have been conducted, no comprehensive analysis of the general model of polynomial s-curve motion profiles is taken into consideration. Here, a formulation about defining the trajectory in the case of an S-curve model of degree n will be presented. As described, in the trapezoidal velocity profile formulation, depending on the picked degree that has been chosen for the model, different constant trajectory segments to be connected will be generated. In the generic case of a model of degree n, 2^n segments have to be connected as shown in Fig.2.12.

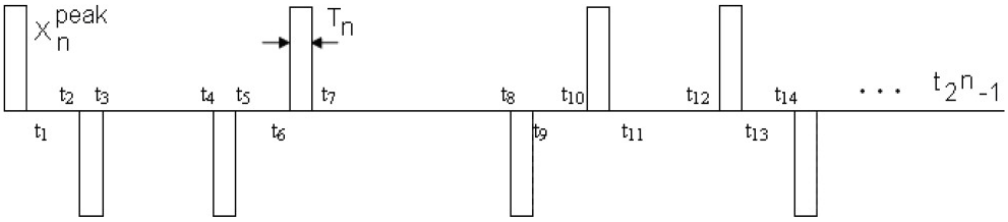

Figure 2.11: Trajectory with constant segments and n-th derivative of position

It is worth noticing that for every n-th S-curve motion trajectory, there will always be a derivative with constant segments. Basically, the goal is to compute T_p that is the time period the trajectory is constant.

Algorithm 1 n-th order s-curve profile

for p = n to 1 **do** $T_p = 0$

for p = n to 1 **do**

$X_0 = \frac{X_n}{2^n} \prod_{i=1}^{n} [(\sum_{j=0}^{i-1} 2^j T_{n+1-i+j}) + T_{n+1-i}]$

for q = 1 to (p-1) **do**

$X_q^{max} = \frac{X_n}{2^{n-q}} \prod_{i=1}^{n-q} [(\sum_{j=0}^{i-1} 2^j T_{n+1-i+j}) + T_{n+1-i}]$

if $X_q^{max} > X_q^{peak}$ **then** Recalculate Tp from

$X_q^{peak} = \frac{X_n}{2^{n-q}} \prod_{i=1}^{n-q} [(\sum_{j=0}^{i-1} 2^j T_{n+1-i+j}) + T_{n+1-i}]$

end if

First, the T_p is set to be zero. Then, it is computed through $T_p = 0$. Then, it is computed through the X_0 equation and used to pull out the value of X_q^{max}. This value is then compare to the input peak value and if it is larger then another value of T_p has to be selected. The algorithm keep going on till a value of T_p allowing not to exceed the input peaks is chosen [9].

2.4 Obstacle Detection

Here, an important feature in robotics will be discussed: *Obstacle Detection*. This allows to detect obstacles along the path bringing to the final position, alarming the software to avoid them. Nowadays, there are several techniques used to spot obstacles. It is enough to think about the automotive field, where mostly of the cars developed right now, are equipped with such technology, helping out people not just to park or to do tough maneuvers but also to avoid dangerous accident. Just few techniques will be detailed in this section, just the ones which are going to be exploited during this work.

LiDAR A key component in autonomous vehicles is *LiDAR* (Light Detection and Ranging). A typical lidar sensor releases pulsed light waves into the surrounding area from a laser. These pulses return to the sensor after bouncing off nearby objects. The sensor determines the distance traveled by each pulse by measuring the time it takes for it to return to the sensor. A real-time 3D map of the environment is produced by repeating this procedure millions of times per second. This 3D representation of the surrounding area can be used for navigation by an onboard computer [10].

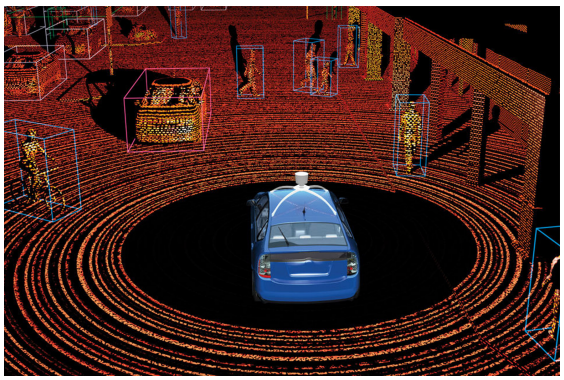

Figure 2.12: Mapping with LiDAR in autonomous vehicles

Ultrasound Sensor Another fundamental cheap component is the *ultrasound sensor* that has been widely discussed in Subsection 2.1.3. It is worth noticing that with such a sensor (using just one of it) it is impossible to fully define the position in the space of an obstacle, as for the nature of the ultrasound waves that are spherical, only the information about how far the obstacle is will be provided. This feature can be clearly seen from Fig.2.3. The propagation of spherical waves in three dimensions is described by the following equation

$$\frac{1}{v^2}\frac{\partial^2 q}{\partial t^2} = \frac{\partial^2 q}{\partial x^2} + \frac{\partial^2 q}{\partial y^2} + \frac{\partial^2 q}{\partial z^2} \tag{2.56}$$

where x, y and z are the Cartesian coordinates and q is the displacement $q(\vec{r}, t)$ of the oscillator-medium at the point labeled $\vec{r} = (x, y, z)$ at time t [11].
As already discussed, the ultrasound sensor provides the distance of the detected object. With this information, the equation of the sphere in the 3-D space to identify the object can be defined as follows

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2 \tag{2.57}$$

where (x_0, y_0, z_0) is the position in which the ultrasound sensor is located and R is the distance provided by the sensor itself. It is worth noticing that with such a piece of information, it is impossible to define the (x, y, z) position of the obstacle in the space. A way to shrink the possible locations of the obstacle from every position defined by 2.56 to few solutions is to use 2 or more ultrasound sensors and solve the system of equations to find the intersections between the spheres. Though, also in this way the unique solution is granted just when the spheres intersect in one point in the space.

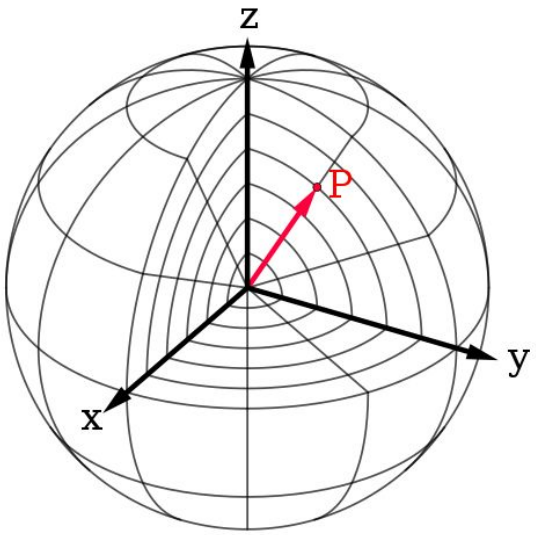

Figure 2.13: Sphere in the 3-D space

Depth Camera *Depth Camera* plays an important role in automotive nowadays. Also Elon Musk, CEO of Tesla, released an interview talking about the importance of such a sensor to detect and map the surroundings. Indeed, thinking about it, all what a car driver uses to sense and get information about the environment around is the sight. With the same principle, depth camera is able to map exploiting the vision features. The usage of this sensor has been widely described in Subsection 2.1.2.

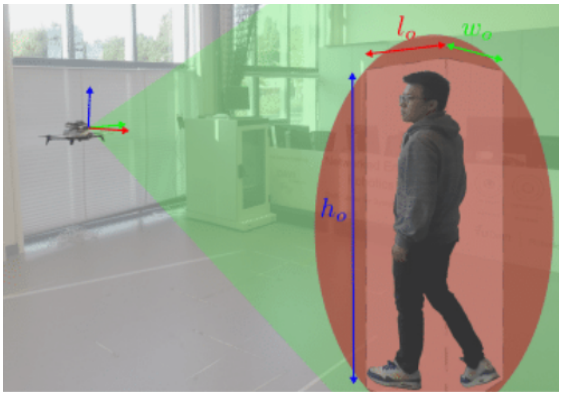

Figure 2.14: Example of human being detection by a depth camera

As shown in Fig.2.14 the depth camera has the capability to recognize a specific object in the space through a well designed *neural network* [12]. Besides of this, the depth camera is also able to generate a *point cloud* that will be discussed in the following subsection.

2.4.1 Point Clouds

As anticipated before, *point cloud* is fundamental for a depth camera to recognize the detected objects. In particular, whenever an object is caught by the camera, a set of points in the 3-D space will be generated to identify such an object. This set of points is basically a point cloud. Each point inside the mentioned set, has its own position of Cartesian coordinates in the space $[xyz]$. Point clouds are typically created by 3D scanners or photogrammetry software, which collect a lot of data from the surrounding objects' external surfaces. Point clouds are a byproduct of 3D scanning operations and have a wide range of uses, including the creation of 3D CAD models for manufactured parts, metrology and quality inspection, and several visualization, animation, rendering, and mass customisation projects. The focus of this project, involving the usage of point clouds, is about visualization and obstacle detection [13].

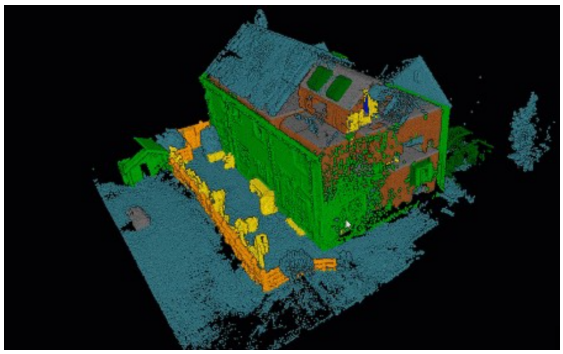

Figure 2.15: Point clouds of an environment

Obviously, using point clouds to detect obstacles allows to choose among different

options. For instance, it is possible to choose how many points are to be used to describe an element in the picture or the precision we want the software to create the point cloud map with. For sure, the higher will be the precision and the number of points used to describe the object, the more complex will be our software from a computational point of view. A fundamental tool that is directly related to point clouds and that allows the differentiate among different objects close each other, is *optimization.*

2.4.2 K-d Tree Algorithm

K-d tree is an optimization algorithm that allows to understand which points among the data set of the point clouds belong to the same object. More specifically, a Kd-tree (sometimes referred to as a k-dimensional tree) is a space-partitioning data structure that organizes a collection of k-dimensional points into a tree form that facilitates effective range searches and closest neighbor searches. When working with point cloud data, nearest neighbor searches are a fundamental operation that may be used to define the immediate vicinity of a point or points as well as to identify correspondences between groups of points or feature descriptors [14].

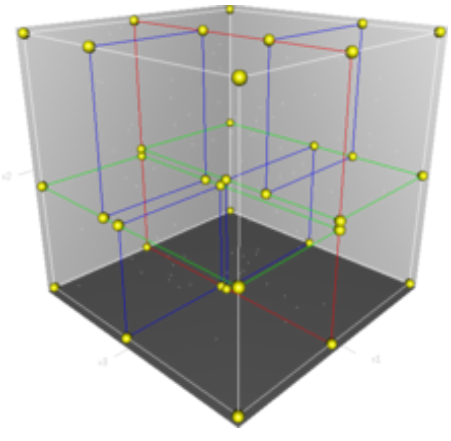

Figure 2.16: Example of a k-d tree algorithm application

The k-d tree is a binary tree in which each node represents a point in k dimensions. It is possible to consider any non-leaf node as inherently generating a splitting *hyperplane* that splits the space into two equal halves, or *half-spaces*. The left subtree of that node represents points that are to the left of this hyperplane, while the right subtree represents points that are to the right of this hyperplane. Every node in the tree is connected to one of the k dimensions, with the hyperplane perpendicular to that dimension's axis. This method of choosing the hyperplane direction So, for instance, if the "x" axis is selected for a given split, all subtree points with "x" values that are smaller than the node will be in the left subtree, and all points with bigger "x" values would be in the right subtree. In this scenario, the x value of the point would determine the hyperplane, and the unit x-axis would serve as its normal.
Just to give an example of such an algorithm, it is possible to explain the procedure showing an image. Looking at Fig.2.16, it can be seen that the root cell (white) is first divided (by the red vertical plane) into two subcells, and then each of these subcells is divided (by the green horizontal planes). The four blue vertical planes finally divide four cells into two subcells. The final eight are known as leaf cells since no further splitting occurs.

Nearest Neighbour Search The goal of the nearest neighbor search (NN) algorithm is to locate the tree node that is closest to an input point. By swiftly removing significant chunks of the search space utilizing the tree attributes, this search can be completed effectively.
The procedure of applying the NN algorithm in a k-d tree problem can be described as follows:

1. The procedure descends the tree recursively from the root node, just as it would if the search point were being inserted (i.e. it goes left or right depending

on whether the point is lesser than or greater than the current node in the split dimension).

2. The method checks the node point at each leaf node it reaches and saves that node point as the "current best" if the distance is less than the "current best".

3. The steps that the algorithm takes at each node to unwind the tree's recursion are as follows:

   (a) The present node becomes the present best if it is closer than the present best.

   (b) The algorithm looks for spots that might be closer to the search point than the present top point on the opposite side of the splitting plane. The splitting hyperplane and a hypersphere with a radius equal to the current nearest distance intersect at the search point to achieve this. Given that all of the hyperplanes are axis-aligned, this is implemented as a straightforward comparison to determine whether the separation between the search point's splitting coordinate and the current node is smaller than the separation (overall coordinates) between the search point and the current best.

      i. If the hypersphere crosses the plane, the algorithm must move down the opposite branch of the tree from the current node in search of closer points, using the same recursive procedure as the entire search. If the hypersphere crosses the plane, there may be closer points on the opposite side of the plane.

      ii. The method walks up the tree and eliminates the entire branch on the other side of that node if the

4. The search is finished when the algorithm completes this step for the root node.

Generally, the algorithm described above, going through the comparison between the current distance and the bast calculated one, uses squared distances avoiding to compute the square roots for matter of computational cost. Moreover, a comparison variable is usually created, holding the squared current best distance [14].

2.5 Obstacle Avoidance

Obstacle avoidance is fundamental nowadays in many sectors. In the autonomous driving it is used to design a obstacle-free path and to make decisions upon the detection of an object. Also in robotics, the system should be able to find out whether the path to the final position is obstacle-free or not. In Section 2.3, the path planning problem has been addressed. Obstacle avoidance is typically thought of as being distinct from path planning because the former typically entails the pre-computation of an obstacle-free path that a controller will later use to direct a robot along.
Before going through a possible method to implement obstacle avoidance, it is important to find a way to model the obstacle along the path. Here the circumscribed ball method will be described. Basically, it is possible to model every obstacle following these steps:

1. Find the smallest cube wrapping the obstacle.

2. Create the cube's circumscribed ball when the long side is no more than three times the short side. Otherwise, a plane that cuts through the middle of each long side divides the cube into two halves.

3. The two portions' wrapped information is disclosed, and each is dealt with as an impediment on its own. In other words, until the obstruction is entirely encircled by the circumscribing ball, steps I (ii), and (iii) are repeated.

4. Obtain the centers and radii of each circumscribed ball after the obstacle has been transformed into one or more of them.

The manipulator's linkages have a specific cross-sectional area. Each link is enclosed in a cylinder to conveniently build a collision detection system. The radius of the manipulator's connection serves as the unit of measurement for the cylinder's section radius. The radius of each encircled ball of the barrier is added to the radius of the connection of the manipulator in order to simplify the calculation. At this point, the obstacle's positional relationship with the manipulator's link is reduced to a straightforward ball-to-wire interaction. The collision detection procedure goes like this:
A two-point equation can be established whenever the information about the two ends of the link are available (x_i, y_i, z_i) and $(x_{i+1}, y_{i+1}, z_{i+1})$. It yields to the following equation

$$\frac{x - x_i}{x_{i+1} - x_i} = \frac{y - y_i}{y_{i+1} - y_i} = \frac{z - z_i}{z_{i+1} - z_i} = t \tag{2.58}$$

that yields to the following system of parametric equations

$$x = t(x_{i+1} - x_i) + x_i \tag{2.59}$$

$$y = t(y_{i+1} - y_i) + y_i \tag{2.60}$$

$$z = t(z_{i+1} - z_i) + z_i \tag{2.61}$$

As anticipated, the obstacle is modeled as a ball. To this extent, its equation is defined as follows

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2. \tag{2.62}$$

Substituting 2.59, 2.60 and 2.61 into 2.62, a quadratic expression about t is found. The value of t tells a lot about the avoidance of the obstacle. Indeed, whenever

$0 \leq t \leq 1$ the intersection is within the range of the line segment, otherwise it is not. Thus, through the evaluation of the t parameter the intersection/position of the ball (obstacle) with respect to the i-th link of the robotic arm can be judged. In order to accomplish the goal of reaching the final position without interfering with any obstacles along the path, the *potential-field method* will be discussed. The artificial potential-field method is a crucial path planning technique. High operability and efficiency are two of its standout benefits. The controlled object is placed in an environment with an artificial potential field that is abstract in the artificial potential field approach. The impediment and the target point in the environment exert "repulsive force" on it, respectively, and the combined force of the two forces propels the controlled object in the desired direction.
To this extent, an *attractive potential field* function has to be introduced

$$U_{att}(X_j) = \frac{1}{2} k \rho^2(X_j, X_g) \tag{2.63}$$

where the variable $X_j = (x_j, y_j, z_j)$ is the position of the controlled object to be taken into consideration at the j-th step of the algorithm evaluation, $X_g = (x_g, y_g, z_g)$ is the position coordinate of the final target, k is the coefficient determining the attractive force as in the well-known equation for the potential energy of a spring and $\rho(X_j, X_g) = ||X_j - X_g||$ is the shortest distance between the the position coordinate of the object to be headed to the final destination and the final destination itself.
In order to define the attractive force bringing the object to the final target, the negative gradient has to be defined as

$$F_{att}(X_j) = -\nabla(U_{att}) = k\rho(X_j, X_g). \tag{2.64}$$

The attractive force can be decomposed along the three main coordinate axes

$$F_{att_x}(X_j, X_g) = F_{att}(X_j, X_g) \times \cos\alpha_{j,x} \tag{2.65}$$

$$F_{att_y}(X_j, X_g) = F_{att}(X_j, X_g) \times \cos\alpha_{j,y} \tag{2.66}$$

$$F_{att_z}(X_j, X_g) = F_{att}(X_j, X_g) \times \cos\alpha_{j,z} \tag{2.67}$$

where the angles $\alpha_{j,x}, \alpha_{j,y} and \alpha_{j,z}$ are the angles between the controlled object X_j and the center of the obstacle object considering the axes x, y and z, respectively. As stated before, the obstacle is modeled as a sphere. In order to define the *repulsive force* that the obstacle exerts on the controlled object, the minimum distance between the sphere and the segment is defined as

$$e_i = \frac{(x_j - x_{0i}, y_j - y_{0i}, z_j - z_{0i})}{\sqrt{(x_j - x_{0i})^2 + (y_j - y_{0i})^2 + (z_j - z_{0i})^2}} \tag{2.68}$$

so the closest point to the object of X_j is defined as

$$X_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) = (x_{0i} + r_i a_i, y_{0i} + r_i b_i, z_{0i} + r_i c_i) \tag{2.69}$$

where $[a_i\ b_i\ c_i]^T$ is defined as

$$\begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} = \begin{bmatrix} \frac{x_j - x_{0i}}{\sqrt{(x_j - x_{0i})^2 + (y_j - y_{0i})^2 + (z_j - z_{0i})^2}} \\ \frac{y_j - y_{0i}}{\sqrt{(x_j - x_{0i})^2 + (y_j - y_{0i})^2 + (z_j - z_{0i})^2}} \\ \frac{z_j - z_{0i}}{\sqrt{(x_j - x_{0i})^2 + (y_j - y_{0i})^2 + (z_j - z_{0i})^2}}. \end{bmatrix} \tag{2.70}$$

At this point, the repulsive potential field function can be defined as

$$U_{rep}(X_j) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho(X_j, X_{i,j}) - \frac{1}{\rho_0}} \right)^2 & \rho(X_j, X_{i,j}) \leq \rho_0 \\ 0 & \rho(X_j, X_{i,j}) < \rho_0 \end{cases} \tag{2.71}$$

where η is the gain coefficient of the repulsive force, $\rho 0$ is the distance of the obstacle and $\rho(X_j, X_{i,j}) = ||X_j - X_{i,j}||$ is the shortest distance between X_j and $X_{i,j}$). In order to find the repulsive force, as for 2.64, the gradient can be defined as

$$F_{rep}(X_j) = -\nabla(U_{rep}) = \eta \left(\frac{1}{\rho(X_j, X_{i,j}) - \frac{1}{\rho_0}} \right) \frac{1}{\rho^2(X_j, X_{i,j})}. \tag{2.72}$$

The repulsive force can be decomposed along the three major axes as follows

$$F_{rep_x}(X_j, X_{i,j}) = F_{rep}(X_j, X_{i,j}) \times \cos \beta_{j,x} \tag{2.73}$$

$$F_{rep_y}(X_j, X_{i,j}) = F_{rep}(X_j, X_{i,j}) \times \cos \beta_{j,y} \tag{2.74}$$

$$F_{rep_z}(X_j, X_{i,j}) = F_{rep}(X_j, X_{i,j}) \times \cos \beta_{j,z} \tag{2.75}$$

where the angles $\beta_{j,x}, \beta_{j,y}$ and $\beta_{j,z}$ are the angles between the object X_j and the center of the obstacle object considering the axes x, y and z, respectively.
Considering the body X_j, the forces applied to it are both repulsive and attractive and combining them, it yields to

$$F_{jx} = F_{att_x}(X_j, X_{i,j}) + \sum_i F_{rep_x}(X_j, X_{i,j}) \tag{2.76}$$

$$F_{jy} = F_{att_y}(X_j, X_{i,j}) + \sum_i F_{rep_y}(X_j, X_{i,j}) \tag{2.77}$$

$$F_{jz} = F_{att_z}(X_j, X_{i,j}) + \sum_i F_{rep_z}(X_j, X_{i,j}). \tag{2.78}$$

Once we computed the forces applied to out body and with the knowledge of the step size l, a prediction about the next position that our body will assume can be done in the following way

$$x_{j+1} = x_j + l \times \frac{F_{jx}}{F_j} \tag{2.79}$$

$$y_{j+1} = y_j + l \times \frac{F_{jy}}{F_j} \tag{2.80}$$

$$z_{j+1} = z_j + l \times \frac{F_{jz}}{F_j}. \tag{2.81}$$

It is worth noticing that exploiting the theory explained above, it is possible to build more complex obstacle avoidance algorithms [15].

2.6 Software Processes Analysis

Another fundamental part to be addressed is about *Software Processes*. Indeed, whenever a robotic system has to be designed and simulated, there is always a software running into a piece of hardware doing something. The term "Software Analysis" refers to any processes that assist in translating requirement specifications into actual implementation. All functional and non-functional expectations for the program are laid forth in requirement specifications. These requirement specifications are presented as texts that are legible and intelligible by humans, and with which computers have no interaction.

The intermediary stage of software analysis and design assists in converting human-readable requirements into actual code. Here, the importance of *processes* will be discussed as they play a key role in the correct execution of the final task to be accomplished.

As shown in Fig.2.17, as a process executes, it changes state. Each state can be described as follows:

- **New:** the process is being created.
- **Running:** instructions are being executed.

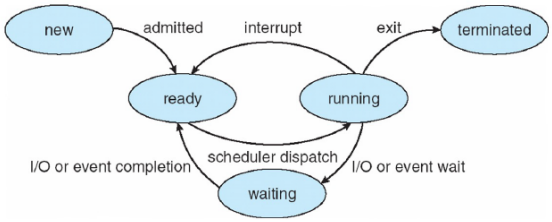

Figure 2.17: Process State Diagram (PSD)

- **Waiting:** the process is waiting for some event to occur.
- **Ready:** the process is waiting to be assigned to the processor.
- **Terminated:** the process has finished the execution

Obviously, the CPU has to be kept always busy as it would be a waste of time keeping it doing nothing. Considering N processes and one processor, only one process can be in the running state. The other ones can be kept either *waiting* for a resource to become available or *ready* to be executed, as shown in Fig.2.18.

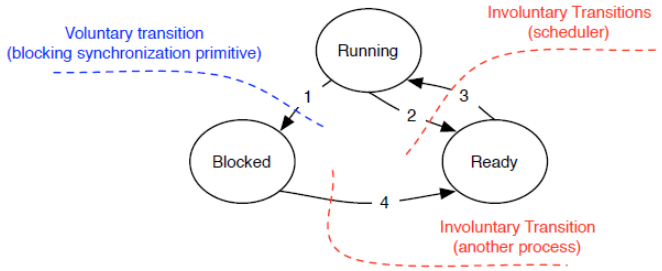

Figure 2.18: Possible states of N processes in the case of one processor

An important operation that could occur inside the CPU is the *preemption.* Preemption happens when a task takes over another task that passes from the running state to the ready one. This phenomenon can happen for different reasons, among them the priority of the preempted task can be lower than the other task. Whenever preemption happens, a *context switch* takes place and it is intended to save all the information about the two tasks.

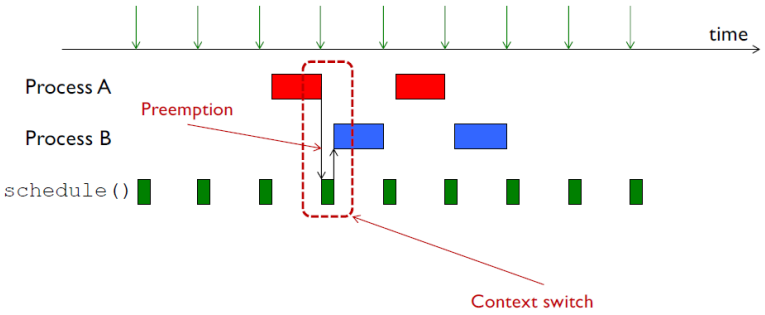

Figure 2.19: Preemption phenomenon between two processes in one processor

2.6.1 Race Condition Problem

The state of an electronics, software, or other system when the behavior of the system is dependent on the timing or sequencing of other uncontrollable events is known as a *race problem* or race hazard. When one or more of the potential actions are undesired, it turns into a bug.

Before getting into this problem, it is worth explaining what a *critical section* is. Basically, the critical section is a piece of software that accesses to a shared exclusive resource. Among the peculiarities of the mentioned resource, it is shared between more tasks and it is protected against concurrent accesses by different tasks [16].

The main problem a shared exclusive resource could cause is the race problem that will be discussed here. Just to give an example, suppose that two tasks want to change the value of a shared exclusive resource as shown in Fig.2.20.

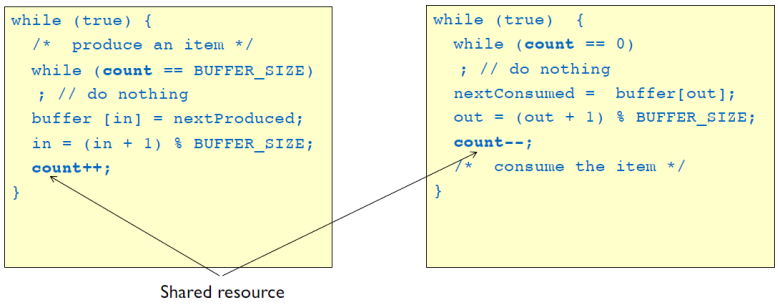

Figure 2.20: Two tasks sharing a shared exclusive resource

In this case the shared exclusive resource is the variable *count* and the problem is that the operations shown in Fig.2.20, $count++$ and $count--$ are not atomic that is to say it does not happen in just one command. To be clear, Fig.2.21 shows what happen inside the CPU whenever those operations are called.

```
register1 = count
register1 = register1 + 1
count = register1
```

```
register2 = count
register2 = register2 - 1
count = register2
```

Figure 2.21: Implementation of command count++ (right side) and count– (left side)

In this case race condition is generated by tasks whose critical sections are executed concurrently giving back a wrong result as shown in Fig.2.22.

```
S0: producer register1 = count
S1: producer register1 = register1 + 1
S2: consumer register2 = count
S3: consumer register2 = register2 - 1
S4: producer count = register1 → count = 6
S5: consumer count = register2 → count = 4
```

Figure 2.22: Wrong result due to race condition problem

In order to solve race condition, the synchronization of the tasks shall be used. The operating system should ensure that whenever the critical section is executed by a task, preemption by another task sharing that resource can not happen.

2.6.2 Multithreading Concept

In systems with multiple processors or CPU cores it could be useful to split the work onto different software *threads*.
Basically, the smallest group of programmed instructions that may be independently controlled by a scheduler, which is often a component of the operating system, is known as a thread of execution. Operating systems implement threads and processes differently, although a thread is typically a part of a process [17]. As discussed previously, if just a CPU is available it is only possible to execute one task at a time.

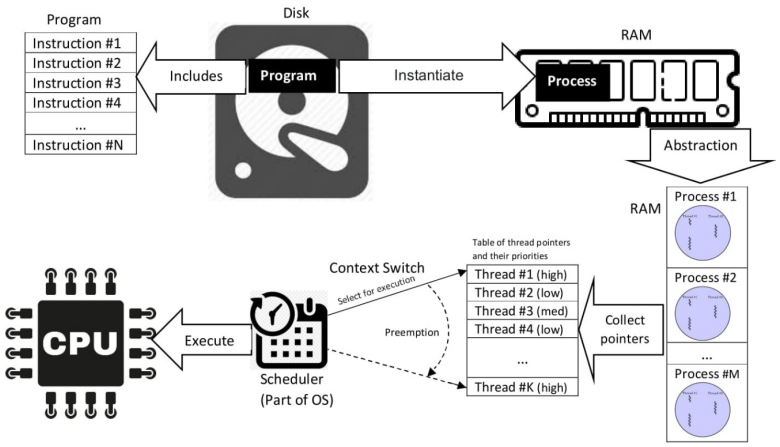

Figure 2.23: Demonstration scheme of how threads work

This is where *multithreading* comes into play. In order to increase the speed of the processor core without changing the frequency, multithreading can be implemented to have the CPU process several tasks at once. Thus, in the case of multithreading, multiple threads are processed more or less simultaneously [18]. It is important to remark that in the case only one CPU is used, it is still possible to do multithreading among multiple programs. Indeed, threads of different programs will run sequentially and as fast as it appears that they are executed concurrently.

Chapter 3

Project Development

In this chapter the whole project will be discussed going through every single part of it. As explained in the introduction, an Automated Robot-based Charger for EVs has to be developed, taking care of both the hardware and software parts. The first two sections will cover the software and hardware tools that have been used to carry out this project, giving a general description of them and going through why they have been chosen. Then, a more theoretical part comes into play into the third section, diving deep to explain why a DOF has been added to the system to accomplish the goal, exploiting and recalling some fundamental theoretical concepts already discussed in Section 2.2. Finally, the software implementation and the logic of the algorithm will be presented, also going through the explanation of some mathematical and geometric considerations to come up with an efficient docking algorithm. Moreover, in this last section, the obstacle detection implementation will be addressed analyzing how it is done and which the advantages and limitations of the implemented technology are. At the end of the section the obstacle avoidance will be discussed as well, even though it is been implemented just into a simulation environment.

3.1 Software Tools

As discussed previously, developing such a system, the only theoretical concepts are not enough. Indeed, also a deep knowledge of the software components has to be taken into account. It is fundamental to choose the proper software to carry out the right activity. To this extent, before getting into the project itself, the software tools that have been implemented will be presented, explaining the choice as well. Basically, the three main software tools that have been used throughout this project are: *ROS*, *RVIZ* and *Arduino*. Nowadays, the first two are widely used in the robotics field to control and make the robot execute different tasks and to simulate the behaviour, respectively. Instead, Arduino is more or less spread in every sector due to its compatibility with different Hardware and Software components.

3.1.1 ROS Environment

ROS stands for *Robotic Operating System* and, as anticipated before, it is widely used in the robotics field. ROS is a collection of open-source robotics middleware. Despite the name, it is not an operating system (OS), ROS is a collection of software frameworks for the creation of robot software. As such, it offers services like hardware abstraction, low-level device control, the implementation of frequently used functionality, message-passing between processes, and package management that are intended for a heterogeneous computer cluster. A graph architecture is used to describe the running sets of ROS-based processes, with processing taking place in nodes that can receive, post, and multiplex messages relating to control, state, planning, actuators, and other topics.

In order to allow users to customize their software stacks to meet their robot and application area, ROS was built to be open source. Users may choose the configuration of the tools and libraries that interacted with the core of ROS. As a result, aside from the overall framework in which programs must exist and interact,

very little is fundamental to ROS. In some ways, message passing and nodes are made possible by ROS. In reality, ROS is more than just that infrastructure; it also includes a wide range of robot-agnostic capabilities offered by packages and a larger ecosystem of ROS-related enhancements.

Although low latency and responsiveness are crucial for robot control, ROS is not a real-time operating system (RTOS). This is a fundamental aspect from a responsiveness point of view since operations are not executed in real time as in the automotive sector. Real-time code can, however, be integrated with ROS. Though, for the development of this project, processes are not *hard real time*. Indeed, waiting some seconds more for the system to start the docking phase towards the vehicle's socket would not take any problem to the final aim. The only operation to be treated with particular attention is the obstacle detection, where a certain responsiveness is required.

Nodes Getting back to the way to operate of ROS, as described briefly before, it is based on *nodes*. Each node has inside its own source code to execute and it could run either waiting for an information from outside or not. Mostly of the times, it waits for a message or an event to raise and this exchange of information between nodes happens through *topics* [19].

Topics Topics are named buses over which nodes exchange messages. Since topics use anonymous publish/subscribe semantics, the creation and consumption of information are separated. Nodes typically have no idea with whom they are interacting. Nodes that are interested in data instead subscribe to the pertinent topic, while nodes that produce data publish to the pertinent topic. A topic may have numerous publishers and subscribers [20].

Figure 3.1: ROS functioning: nodes, topics, messages

As shown in Fig.3.1, the nodes can be recognized having an oval shape. In particular, the */teleop_turtle* node (blue) communicate the information to the */turtlesim* one (green) through the */turtle1/command_velocity* topic (red arrow).

3.1.2 RVIZ Environment

RVIZ stands for ROS Visualization and is a 3-D visualization software that allows the user to check the surroundings of the robotic system is being developed. This dynamic visualization environment allow not just to check out around the robot exploiting sensors, but also to simulate a possible system in order to understand what could be possible outcomes of the implementation.

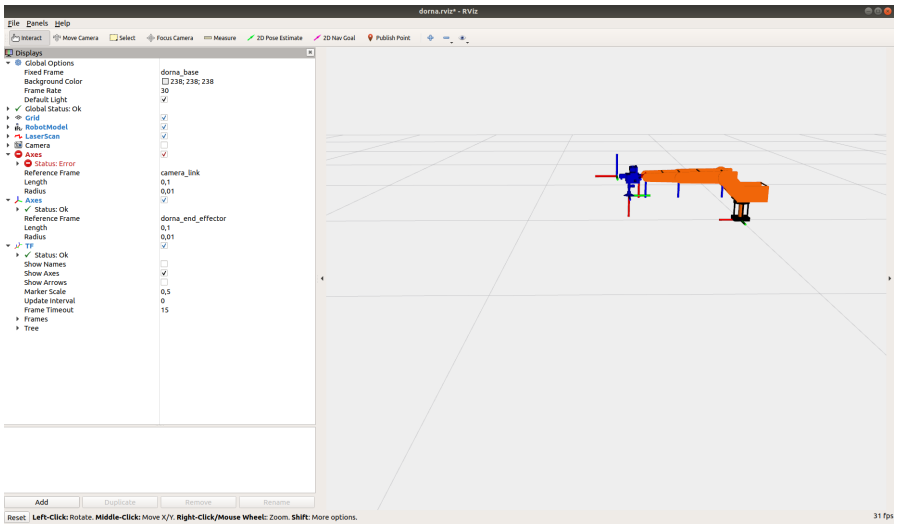

Figure 3.2: Dorna 1 robot visualization in RVIZ environment

As shown in Fig.3.2, the robotic arm that has been used for this project is visualized in RVIZ. Obviously, this visualization software allows to understand what the manipulator is able to do and what tasks it is able to manage. Furthermore, RVIZ allows to visualize the coordinate frames that, in the case shown in Fig.3.2, are placed at each motor plus at the gripper. Beside of this, also ROS is able to manage different coordinate frames and all the transformations between them that have been described in Section 2.2.

As anticipated, through RVIZ it is possible to check what is going on around the robot. It could be done both in simulation, adding a fake sensor to see the functioning and in "real-time" to check out how the robot see the surrounding environment and whether there are some problems related to the sensor or not.

For this project a depth camera has been used and, as a matter of fact, RVIZ has been fundamental to understand problems at sensor-PCB communication level. As shown in Fig.3.3, the *AR_tag* simulating the vehicle's socket, is correctly visualized and no errors should arise considering the sensor part.

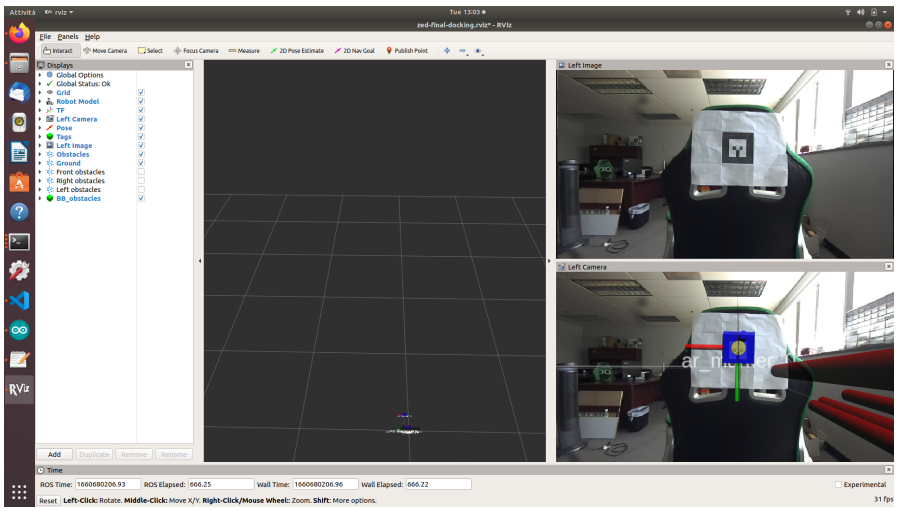

Figure 3.3: Visualization on RVIZ of the AR tag with respective reference frame

Furthermore, a blue square and the relative coordinate system have been attached to the tag to make clearer the visualization.
Moreover, RVIZ gives the possibility of creating dynamic environment inserting objects, obstacles, creating various environment tailored on the kind of application of the robot.

3.1.3 Arduino Environment

The open-source Arduino platform is used to create electronics projects. With Arduino, users can write and upload computer code to a physical programmable circuit board (commonly called a microcontroller) using a piece of software called the IDE (Integrated Development Environment), which runs on a computer or a PCB.

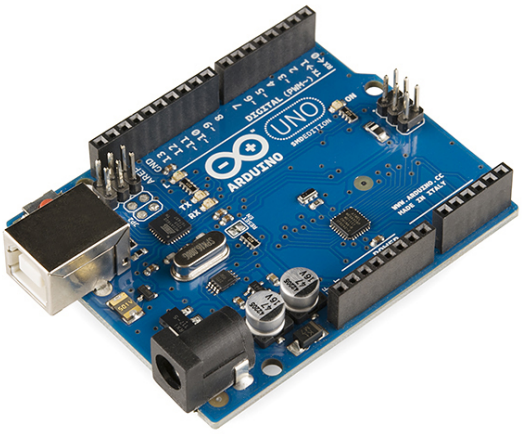

Figure 3.4: Arduino microcontroller

With those just getting into electronics, the Arduino platform has grown rather popular, and for good reason. The Arduino does not require a separate piece of hardware (referred to as a programmer) in order to load fresh code onto the board;

instead, users can do so by using a USB cable, in contrast to the majority of earlier programmable circuit boards. Additionally, the Arduino IDE employs a condensed form of C++ that makes learning to program simpler.
In this project, Arduino has been used as both microcontroller and IDE, to control the *stepper motor*, which utility will be discussed later, and *ultrasound sensors* used for the obstacle detection.

3.2 Hardware Tools

Here the importance of the *hardware* components will be addressed. Indeed, once the software is ready, it has to be employed into a piece of hardware to run and execute tasks. Moreover, mostly of the times, source codes are developed to perform actions on physical components and to get information about a particular quantity of interest of the system. Here, hardware-based *sensors* come into play, providing the proper needed data to accomplish the final task.
In this section, the hardware components that have been used will be described in detail. First, it will go through the *Dorna 1* robotic arm used to realize the automated charger station, describing all the main features such as resolution, repeatability, limitations on speed and payload. Then, the sensors and PCBs used to implement the algorithm and to get the information about the robot will be addressed.

3.2.1 Dorna 1 Robot

To carry out the project, one of the most important parts is the robotic arm that has been employed. Indeed, as described several times throughout writing down the project, the automated charger is robot-based. To this extent, a *Dorna 1* manipulator has been used. Among the features characterizing this robotic arm, it

is equipped with five motors that means it has five DOF. This plays a key rule in the positioning of the manipulator in the space that will be addressed in Section 3.3.

Figure 3.5: Dorna 1 robotic arm

It is intended to be used in both industrial and scientific applications, thanks to its high precision, high speed and high payload capability for a robotic arm of its dimension.

Number of Axis	5
Payload	1.1 Kg
Horizontal Reach	497.1 mm
Vertical Reach	607.9 mm
Resolution	0.1 mm
Repeatability	0.025 mm
Max Base Speed	300 deg/sec
Body Weight	5.4 Kg

Table 3.1: Main features of the Dorna 1 robotic arm

As shown in Table 3.1, the robotic arm ensures a good repeatability and

resolution and quite high speed limits with a light body. This features allow to use the robot in every environment for various applications.
Furthermore, a controller box is attached to the Dorna robot to control the stepper motors so that every movement is successfully accomplished. It is also possible to connect further sensors and actuators and make them work coordinated using the available API through simple Python source code.

Figure 3.6: Dorna 1 controller box

It is worth noticing that, as written into the controller box in Fig.3.6, the controller is able to support a further stepper motor. The communication between the controller and the computer is USB-based and it is possible to send commands through ROS and get information about the status of the robot itself.

3.2.2 Sensors and PCBs

As anticipated in Section 2.1, sensors are fundamental to get information of the surroundings. To develop this thesis work, ultrasound sensors and stereo camera have been used. The first ones guaranteed the obstacle detection during either the initial phase, in which the environment is scanned and the robot is trying to understand whether the docking phase can be started or not, and executing the path to the final destination.

Ultrasound sensor The model of the employed ultrasound sensor is the one shown in Fig.2.3, that is the HC-SR04. The functioning of such a sensor is the same already described in Subsection 2.1.3 and it is fully compatible with Arduino board.

Operating Frequency	40 kHz
Max Range	4 m
Min Range	2 cm
Measuring Angle	15°

Table 3.2: Main features of the Dorna 1 robotic arm

As shown in Table 3.2, the sensor ensures a good working range with the only limitation of not detecting obstacles below 2 cm.

ZED 2 camera Instead, to detect the AR tag in RVIZ environment and to get the information about its position, as anticipated in Subsection 3.1.2, a ZED2 stereo camera has been used and its functioning has been described in Section 2.1. This powerful depth camera comes with a SDK, software development kit that is a set of tools for a specific platform, including different code libraries to be used to implement a great variety of functionalities. The working principle of the whole system ZED 2 camera plus SDK is shown in Fig.3.7.

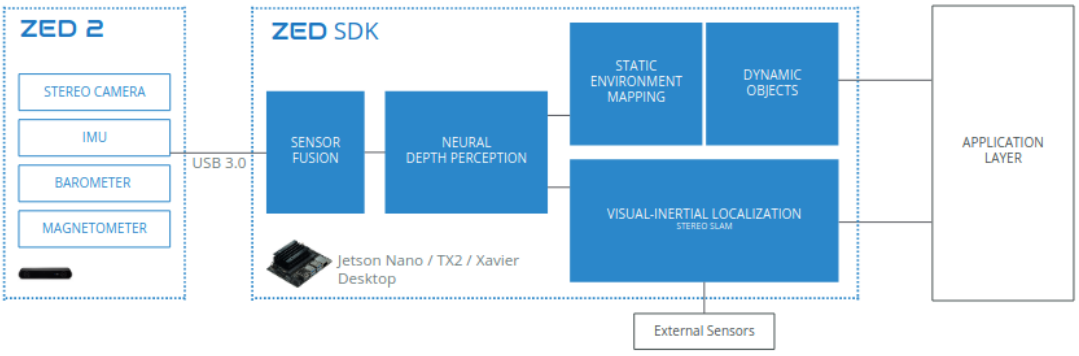

Figure 3.7: Functional SDK diagram

In particular, it is fully compatible with ROS and is able to provide the following data [21]:

- Left and right rectified/unrectified images.
- Depth map.
- Colored 3D point cloud.
- Visual odometry: Position and orientation of the camera.
- Pose tracking: Position and orientation of the camera fixed and fused with IMU data.
- Spatial mapping: Fused 3d point cloud.
- Sensors data: accelerometer, gyroscope, barometer, magnetometer, internal temperature sensors.

Not everyone of this feature has been exploited to carry out the project. Indeed, just the ones needed to build the 3-D depth map and the point cloud have been implemented, not considering the simple visualization capabilities of both the right and left cameras.
As far as the PCBs are concerned, two different printed circuit boards have been used: *NVIDIA Jetson Nano* and *Arduino UNO*.
NVIDIA Jetson Nano NVIDIA Jetson Nano is an embedded system-on-module (SoM) and developer kit from the NVIDIA Jetson family, including an integrated 128-core Maxwell GPU, quad-core ARM A57 64-bit CPU, 4GB LPDDR4 memory, along with support for MIPI CSI-2 and PCIe Gen2 high-speed I/O [22]. With this PCB it is possible to run multiple neural networks in parallel for many applications, all of this with a power consumption as low as 5 watts.

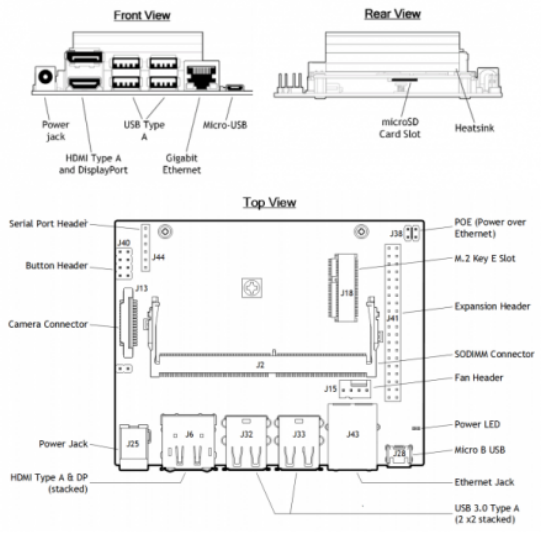

Figure 3.8: NVIDIA Jetson Nano hardware scheme

Another important aspect to be taken into consideration is that Jetson Nano board comes with Linux Operating System, the optimal choice if ROS has to be implemented. This makes this PCB the ideal board to carry out projects in the robotics field, begin able as well to interact with many different sensors. Indeed, in this case this board is fully compatible with the ZED 2 camera that has been used to visualize the environment.

Arduino Mega 2560 board The other PCB implemented for this project is Arduino , that has been already widely described as software in Subsection 3.1.3. Arduino Mega 2560 is an exemplary development board dedicated for building extensive applications as compared to other maker boards by Arduino. The board accommodates the ATmega2560 microcontroller, which operates at a frequency of 16 MHz. The board contains 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a USB connection, a power jack, an ICSP header, and a reset button [23].

The Arduino Mega 2560 is perfect for this project as it is capable of handling

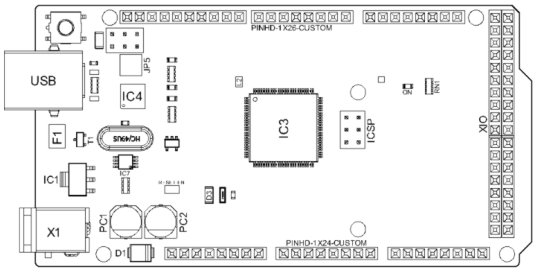

Figure 3.9: Arduino MEGA 2560 hardware scheme

a wide range of robotic applications thanks to its high processing capacity. It can operate many motors simultaneously thanks to compatibility with the motor controller shield, making it ideal for robotic applications. Numerous robotic sensors can be accommodated by the numerous I/O pins. In particular, this board has been chosen because it has to control a stepper motor and several ultrasound sensors. For such an application a quite good amount of memory is needed and this board can provide that. Unfortunately, Arduino boards are not able to run processes simultaneously yet. Though, it is enough for this project since an high responsiveness is not required.

3.3 Automated Charger System

In this section the importance of implementing a further DOF to the robotic system will be deepened. Indeed, as described in Subsection 3.2.1, the robot has just five DOFs that are not enough to choose freely the position and the orientation of the gripper. This is a fundamental aspect to be analyzed as the charging system at the end of the docking phase should be completely able to start the push phase that has not been carried out in this project. To this extent, the kinematic studies about the robotic arm and the whole system will be discussed in the next subsections. After a theoretical part, recalling some important concepts described in Section

2.2, the hardware architecture will be explained, motivating every choice that has been taken.

3.3.1 Kinematic Model of the Robotic Arm

Here the problem of adding a degree of freedom more to the robotic charge will be addressed studying the kinematic model of the robotic arm. In Section 2.2 the importance of carrying out a mathematical study of the system has been addressed. In particular, the homogeneous matrix has been presented and how it is possible to understand the influence of the different joint variables to fix the position and attitude of the robot.

For the Dorna 1 robot and being coherent to the DH convention, it is possible to choose the reference frames as shown in Fig.3.10.

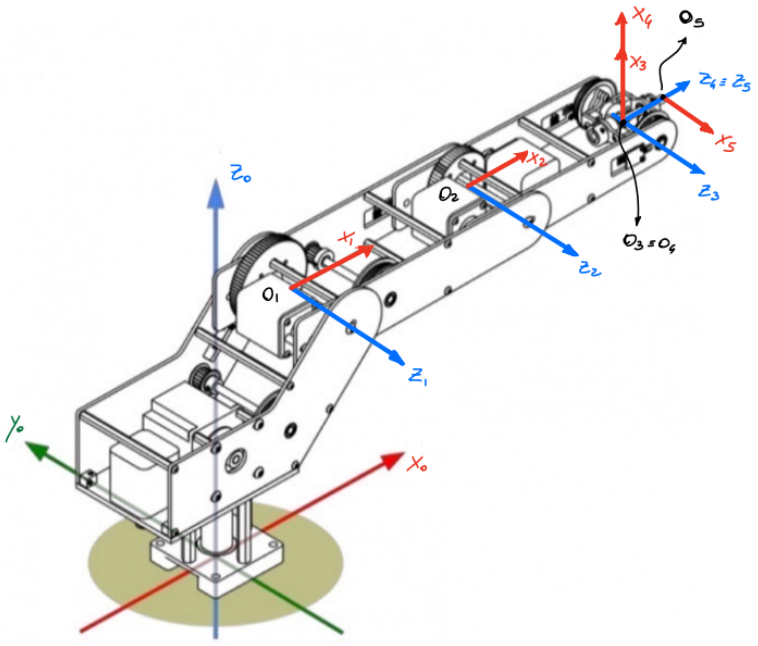

Figure 3.10: DH convention for Dorna 1 robotic arm

For such a choice of the reference frames, the table of the DH parameters can

be defined as in Table 3.3.

Link	a_i	α_i	d_i	θ_i
1	l_1	$\pi/2$	d_1	θ_1
2	l_2	0	0	θ_2
3	l_3	0	0	θ_3
4	0	$\pi/2$	0	θ_4
5	0	0	d_5	θ_5

Table 3.3: DH parameters for Dorna 1 robot

Thus, the homogeneous matrix or transformation matrix allowing to express the coordinates of the end-effector frame into the base frame of the robot can be computed through the multiplication of matrices as follows

$$T_5^0(q) = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 \tag{3.1}$$

$$= \begin{pmatrix} s_1 s_5 + c_1 c_5 c_{234} & s_1 c_5 - c_1 s_5 c_{234} & c_1 s_{234} & c_1(l_1 + l_3 c_{23} + d_5 s_{234} + l_2 c_2) \\ s_1 c_5 c_{234} - c_1 s_5 & -s_1 s_5 c_{234} - c_1 c_5 & s_1 s_{234} & l_1 s_1 + s_1(l_3 c_{23} + d_5 s_{234} + l_2 c_2) \\ c_5 s_{234} & -s_5 s_{234} & -c_{234} & d_1 + l_3 s_{23} - d_5 c_{234} + l_2 s_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In 3.1, the rotational part of the end-effector frame with respect to the base frame is represented by

$$R_5^0(q) = \begin{pmatrix} s_1 s_5 + c_1 c_5 c_{234} & s_1 c_5 - c_1 s_5 c_{234} & c_1 s_{234} \\ s_1 c_5 c_{234} - c_1 s_5 & -s_1 s_5 c_{234} - c_1 c_5 & s_1 s_{234} \\ c_5 s_{234} & -s_5 s_{234} & -c_{234} \end{pmatrix} \tag{3.2}$$

while the displacement can be expressed through the vector

$$\vec{d} = \begin{bmatrix} c_1(l_1 + l_3 c_{23} + d_5 s_{234} + l_2 c_2) \\ l_1 s_1 + s_1(l_3 c_{23} + d_5 s_{234} + l_2 c_2) \\ d_1 + l_3 s_{23} - d_5 c_{234} + l_2 s_2 \end{bmatrix}. \tag{3.3}$$

Just to verify the correctness of 3.5, the determinant of the matrix and the magnitude of the column vector x_0^5, y_0^5 and z_0^5 have been computed. Correctly, $det(R_5^0(q)) = 1$ and $||x_5^0|| = ||y_5^0|| = ||z_5^0|| = 1$.

3.3.2 The Need of Adding a DOF more

The inverse kinematic problem can be addressed as shown in Subsection 2.2.3 adapting to five angle variables instead of three. Though, even without assessing such a problem it is possible to notice that five unknown angle variables to be fixed are not enough to freely choose either position and orientation in the space of the end-effector to reach the final goal.

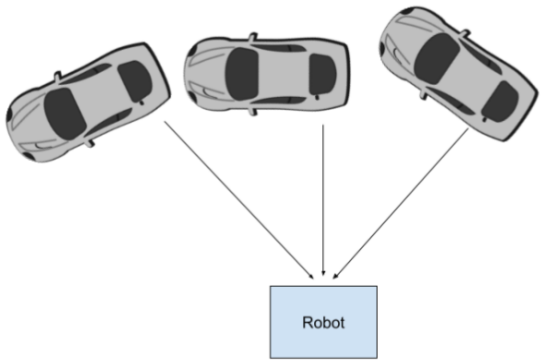

Figure 3.11: Car configuration

This is fundamental for this application as for starting the plug-in phase of the charging cable into the vehicle's socket, the gripper (or end-effector) holding the cable has to be aligned to the socket. Indeed, if just the robotic arm has to be employed, the gripper is able to reach the socket completely aligned just in the cases sketched in Fig.3.11.

In particular, five degrees of freedom are enough whenever the vector perpendicular to the socket plane of the car intersects the z_0 axis of the robot reference frame shown in Fig.3.10. Obviously, the car can not get always those position when it

needs to be charged. Because of this, other cases have to be addressed and the employment of a degree of freedom more has to be discussed.
Just to be clear, throughout this paper in order to simulate the system, the gripper shown in Fig.3.12 will be treated as a power cable for the EV.

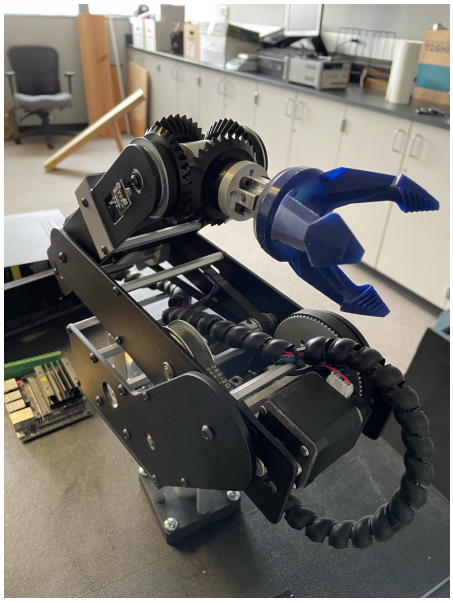

Figure 3.12: Dorna 1 gripper simulating the plug

To solve the position-orientation problem, two solutions have been proposed, such as:

- *Pen-stand principle*: attaching the gripper (or power cable) to the mechanical mechanism of the pen stand to exploit a nonmotorized spherical wrist to be oriented through magnets.
- *Railway*: mounting the robotic arm on a railway motorized by a stepper motor.

Among the two solutions, the second one has been chosen as there is no risk of electromagnetic influence on the charging system. Furthermore, picking the second choice, there is the chance to extend the workspace, as shown in Fig.3.13, of the system that is an important feature for this kind of application. Indeed, it is

impossible to think about the car to stop always at the same point in front of the robotic arm.

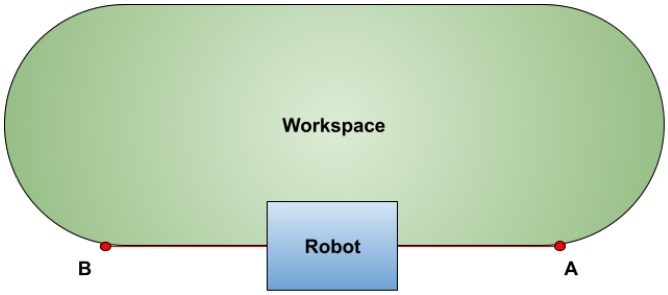

Figure 3.13: Workspace top view of the robotic arm mounted on a railway

In Fig.3.13, the red line represents the railway and A and B are the starting and ending points, respectively. It is worth noticing that the obtained workspace is the one shown in Fig.1.6 but repeated for every single position along the railway. In this case, the inverse kinematic problem gives a solution to reach the socket choosing position and attitude. It is worth noticing that not every point inside the workspace can be reached with every orientation. Indeed, in order to reach the socket with the needed orientation, the vector perpendicular to the socket plane has to intersect the red line representing the railway. This last topic will be discussed later when the software algorithm will be addressed.

3.3.3 HW & SW Integration

In order to implement the system that has been discussed throughout this chapter, Hardware & Software Integration is needed. Indeed, as already mentioned, different pieces of hardware running software have been employed to carry out this project. Basically, the core part was the NVIDIA Jetson Nano running ROS through Linux Ubuntu operating system. Thanks to ROS it was possible to run different nodes taking care of the recognition of the vehicle's socket, the docking algorithm and the obstacle detection algorithm. There was an additional node running on ROS

but written through Arduino IDE, managing the obstacle detection part and the sliding of the robotic arm along the railway. So an Arduino board was used to control the stepper motor that through a thread made the robotic arm move along the railway and to get information about possible obstacles from the ultrasound sensors. Finally, the Dorna 1 robotic arm was used as manipulator and the ZED 2 camera to spot the socket. All of them were attached together as shown in Fig.3.14.

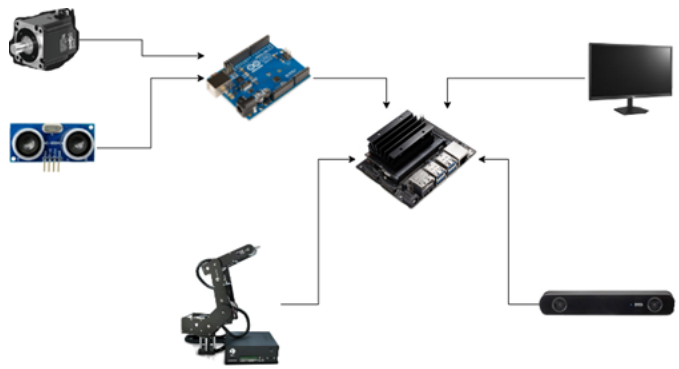

Figure 3.14: Hardware scheme of the Automated Charger System

Going more into detail, there were two different fashions of implementing the system basing on the ZED 2 camera position: *rover* and *static*.

Rover In the rover configuration, the ZED 2 camera was mounted on top of the gripper using a metal rack as shown in Fig.3.15.

The camera reference frame moved along with the gripper one so that a static transformation was enough to express the coordinates in the ZED frame into the gripper frame. Referring to the reference frame choice for the robotic arm shown in Fig.3.10, the ZED frame was chosen as shown in Fig.3.16.

In this case, the transformation matrix transforming the ZED coordinates into the end-effector ones is

Figure 3.15: Rover configuration of the ZED 2 camera

$$R^5_{zed}(q) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

where d_x, d_y and d_z are the displacements of the ZED camera with respect to the origin of the gripper frame.

Static configuration In the static configuration the ZED 2 camera was fixed to the moving platform of the railway and it was static with respect to the robotic arm.

In this case the transformation matrix expressing the ZED coordinates into the base frame ones is static

$$R^0_{zed}(q) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

Among the two configurations, the static one was implemented since in the rover to express the coordinates from the ZED 2 camera frame to the base frame 3.1 has to be used. Indeed, the transformation matrix of the robot brings some uncertainties on both the angle variables and the length of the robot links, implying an uncertain position of the object detected by the camera.

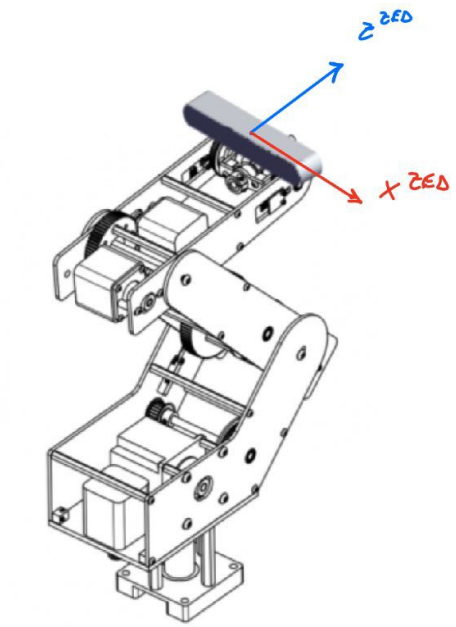

Figure 3.16: Reference frame choice for the rover ZED 2 camera

Finally, the ultrasound sensors were attached at the top of the gripper to check out the front part along the path, at the bottom part to check out whether the docking phase can start or not and on both sides of the manipulator for possible side

obstacles. Though, the number of the ultrasound sensors employed can be changed depending on the application thanks to the scalability of the code managing them.

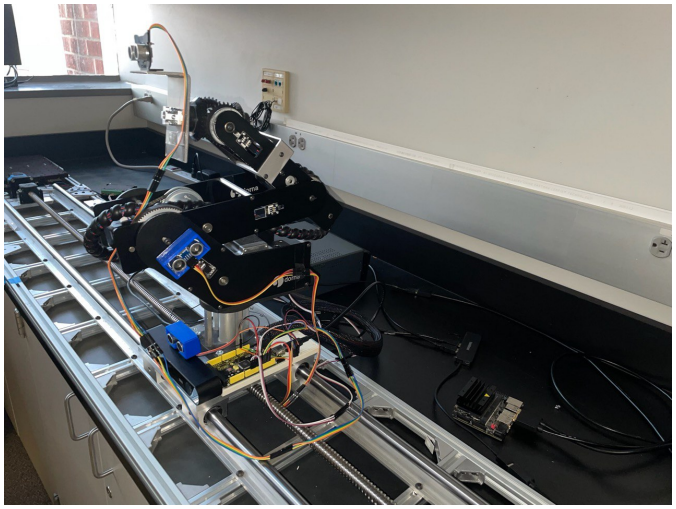

Figure 3.17: Automated Charger system

3.4 Software Development

As already mentioned the software is fundamental to accomplish the final goal of the project and once the hardware components are defined, the *Software Development* part can be addressed. In particular, first the model of the vehicle's socket used for simulation purposes and how to get the information about the position and the orientation inside the workspace will be discussed. Then, the algorithm logic will be explained for both the *docking* and *obstacle detection* algorithm. Also the multithreading, presented in Subsection 2.6.2, will be presented and how it was applied to the project to get the useful information of the robotic arm without interrupting any other process in motion. Moreover, as far as the Software Processes management, the global variables were used very carefully to avoid the race condition problem. Indeed, whenever a global variable was used and modified inside a function, in the others it was set to readable fashion

only. In the final part of this section, an obstacle recognition algorithm through ZED 2 camera that could support an *obstacle avoidance* algorithm will be addressed.

3.4.1 Socket Detection

Here, the problem of modelling and detecting the vehicle's socket will be discussed. The idea to spot the vehicle's socket was to use an *AR tag* that is a 3D marker used in computer vision for alignment and position tracking [24]. So from now on, whenever AR tag will be mentioned it will refer to the socket or the final position to be reached. As far as the detection part is concerned, the ZED 2 camera along with the *ar_track_alvar* package were used.

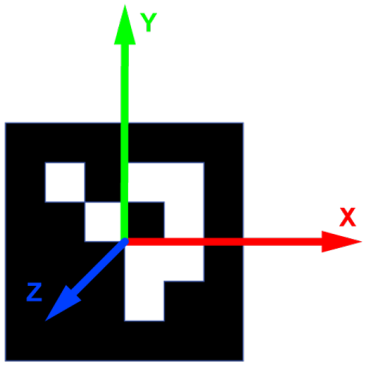

Figure 3.18: AR Tag with the respective coordinate frame

Ar_track_alvar package Ar_track_alvar package is an open source library and a ROS wrapper widely used for tracking objects. This powerful package has different functionalities such as:

- Creating different sizes, resolutions, and data/ID encodings for AR tags
- Locating and tracking each individual AR tag's pose, with the option of

including depth data from the kinect (if one is available) to improve pose predictions.

- Recognizing and tracking the position of "bundles" made up of several tags. This enables tracking of multi-sided objects, robustness against occlusions, and more reliable pose estimates.
- To avoid having to manually measure and enter tag locations in an XML file in order to use the bundle capability, camera images might be used to compute the spatial relationships between tags in a bundle [25].

As discussed in Subsection 3.1.1, in ROS there are several nodes communicating one each other through topics. Starting the ZED 2 node and exploiting the ar_track_alvar package it was possible to get the position and orientation of the socket through the published *ar_pose_marker* topic. The latter contained a message which structure is shown in Fig.3.19.

```
Header header
uint32 id
uint32 confidence
geometry_msgs/PoseStamped pose
```

Figure 3.19: AR tag message structure in the ar_pose_marker topic

3.4.2 Filtering Data

In order to filter the data received from the ZED 2 camera about the AR Tag position and orientation, a specific node was designed. This is useful to understand whether the system has to reach the final position or not, but also if it is reachable or not by the robotic arm. Indeed, this node is intended to stay in the middle between the station detection node and the docking node. Before getting into the logic scheme of the implemented algorithm, it is better to sketch all the possible

cases that could happen in a real-case application. In fact, as already mentioned, it is impossible for a car to stop always in the same position oriented facing exactly the robot. As well as, it is impossible that a car is going to stop always parallel to the railway.

Before addressing all the possible cases that could arise, it is worth presenting a general situation with all the variables that were used to implement the algorithm and to differentiate the many possible situations.

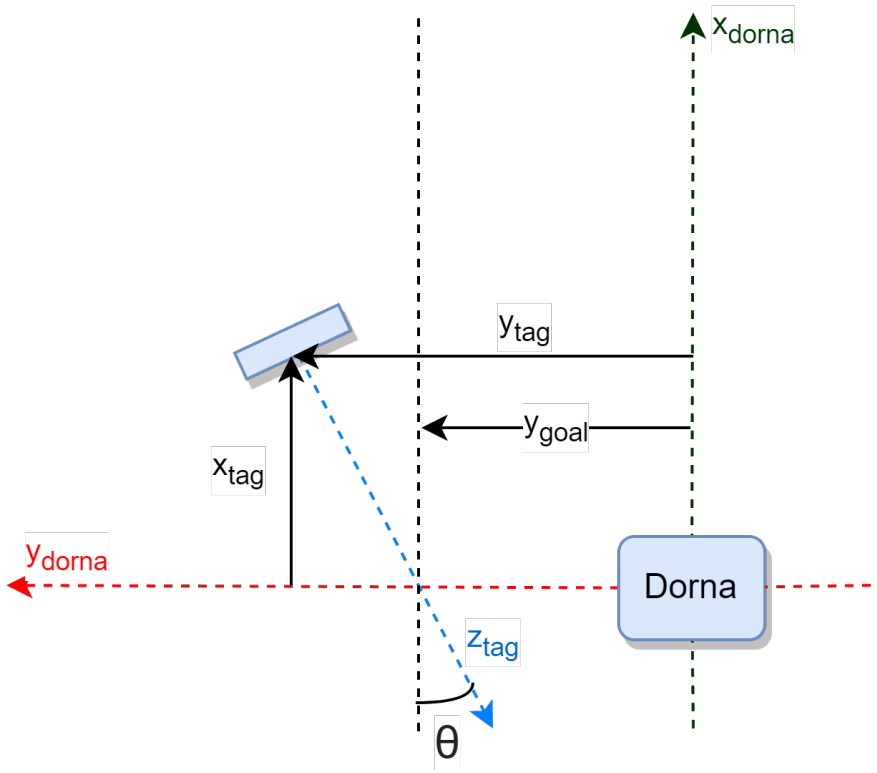

Figure 3.20: General possible situation and all the useful variables

Referring to Fig.3.20, the robotic arm is placed in O_{dorna} that is the origin of the reference frame $(x_{dorna}, y_{dorna}, z_{dorna})$, x_{tag} and y_{tag} were used to locate the AR tag in the space and were provided by the ar_track_alvar package through the message shown in Fig.3.19 and θ is the angle between z_{tag} and x_{dorna} axes. In order to compute θ, the information about the quaternions was provided by the ZED

camera node in the form $q = [q_0\ q_1\ q_2\ q_3]$. Then, through a proper function the respective rotation matrix was computed as

$$r_{11} = 2(q_0^2 + q_1^2) - 1 \tag{3.6}$$

$$r_{12} = 2(q_1 q_2 - q_0 q_3) \tag{3.7}$$

$$r_{13} = 2(q_1 q_3 + q_0 q_2) \tag{3.8}$$

$$r_{21} = 2(q_1 q_2 + q_0 q_3) \tag{3.9}$$

$$r_{22} = 2(q_0^2 + q_2^2) - 1 \tag{3.10}$$

$$r_{23} = 2(q_2 q_3 - q_0 q_1) \tag{3.11}$$

$$r_{31} = 2(q_1 q_3 - q_0 q_2) \tag{3.12}$$

$$r_{32} = 2(q_2 q_3 + q_0 q_1) \tag{3.13}$$

$$r_{33} = 2(q_0^2 + q_3^2) - 1 \tag{3.14}$$

where $r_{i,j}$ is the element of the quaternion equivalent rotation matrix of the i-th row and j-th column. Then, the z_{tag}^{dorna} vector, that is z_{tag} expressed in the base Dorna frame, was extract from the rotation matrix being the last column so that

$$z_{tag}^{dorna} = \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix} \tag{3.15}$$

and finally θ was computed as follows

$$\theta = \arctan(\frac{r_{23}}{r_{13}}) \tag{3.16}$$

Instead, in order to compute y_{goal}, that is the quantity the robotic arm should move thanks to the railway in order to finally have the system gripper-tag aligned, the following relation was exploited

$$y_{goal} = y_{tag} - x_{tag} \tan(\theta) \tag{3.17}$$

After the description of all the variables involved, it is possible to present various situations that could arise. Different cases and subcases were found and studied for the filtering node algorithm development:

- **Case 1**: the car is parallel (or almost) to the railway.
    1. **Case 1.a**: there is no need to use the railway and the docking is based on the current detection.
    2. **Case 1.b**: there is the need to use the railway and the docking is based on the current detection.
- **Case 2**: the car is not parallel to the railway.
    1. **Case 2.a**: there is no need to use the railway and the docking is based on the current detection.
    2. **Case 2.b**: there is the need to use the railway and the docking is based on the current detection.
    3. **Case 2.c**: there is the need to use the railway and the docking is based on the previous detection.

Case 1.a In this case the car is parallel (or almost) to the railway and there is no need to move the railway. It is worth noticing that "almost" is used since it is rare in a real-case application for the software to compute $\theta = 0\,deg$. To this extent, a certain tolerance $\epsilon = \pm 5\,deg$ was applied to define whether or not the tag was parallel to the railway. Then, to decide whether it was needed to use the railway or not, another small variable $\delta = 200\,mm$ to set the tolerance along the y axis was defined. Thus, this situation was addressed whenever $|\theta| < \epsilon$ and $|y_{goal}| < \delta$.

Case 1.b In this case the car is still parallel but in order to reach the tag with the gripper to start the charging phase, it is needed to use the railway. Though, in order to not waste energy in useless moves of the railway, the inverse kinematic problem is computed upfront to understand whether reaching y_{goal}, then the robotic arm is able to reach the tag. Mathematically, this situation arose whenever $|\theta| < 5\,deg$ and $|y_{goal}| > \delta$.

Case 2.a Instead in such a situation the tag is not parallel to the railway but there is no need to use the railway though. This case arose $|\theta| \geq 5\,deg$ but $|y_{goal}| < \delta$.

Case 2.b Finally, in this case the robot is not parallel to the railway and there is the need to use the railway moving the platform by the quantity y_{goal}. Also here as for Case 1.b the inverse kinematic problem were solved to understand if it was worth activating the stepper motor and using the railway. Thus, this situation was addressed whenever $|\theta| > \epsilon$ and $|y_{goal}| > \delta$.

From now on, Cases 1.a, 1.b, 2.a, 2.b will be referred as *standard cases* and the logic algorithm scheme is presented in Fig.3.21. THETA, Y_GOAL and Y_AR in the scheme are the absolute values of θ , y_{goal} and y_{ar} respectively. It is worth noticing that whenever the *move the rail* block was reached, the software was waiting for another detection of the tag to start send the message of starting the docking to the docking node. Furthermore, whenever the railway was used and the tag was not parallel to y_{dorna}, a *flag* variable was activated and the position of the tag was saved into memory, to be sure that whether Case 2.c arose, it could be addressed.

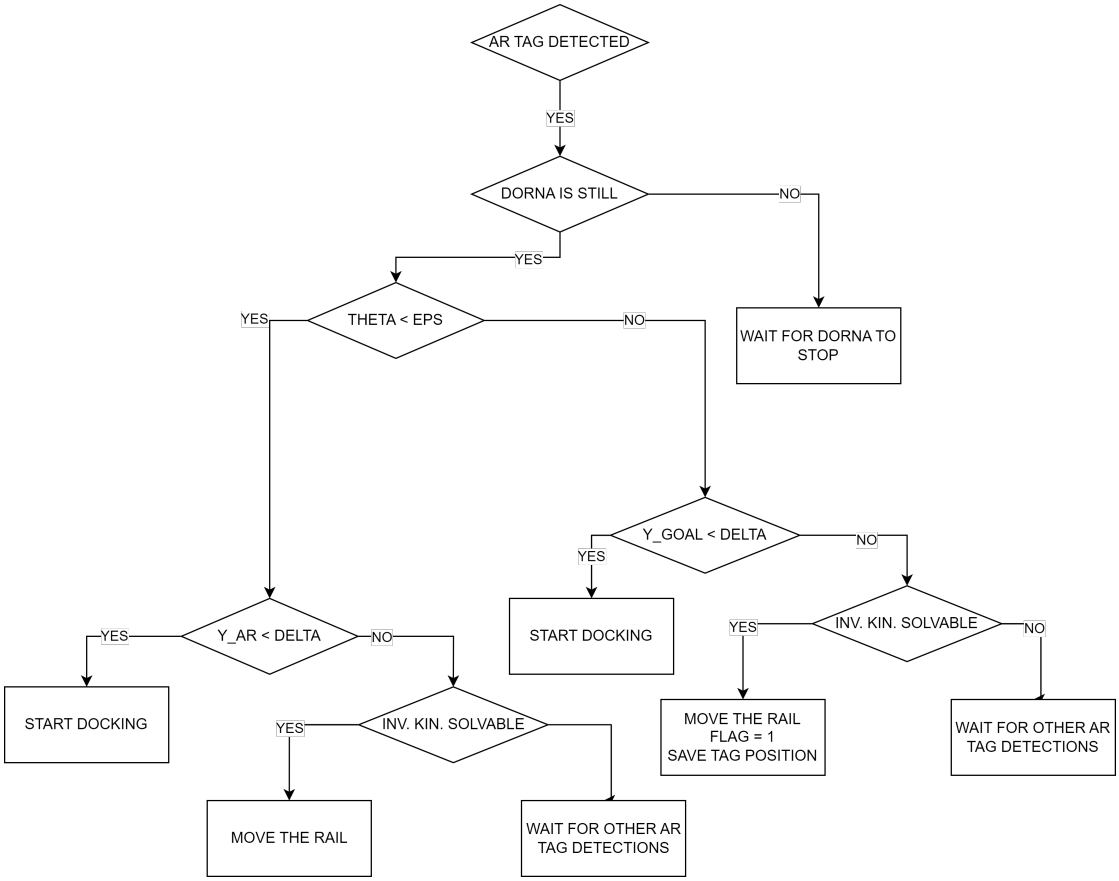

Figure 3.21: Filtering data algorithm for the standard cases

Case 2.c Here, the worst case was experienced as the docking was based on the previous detection of the tag. This happened because upon moving the platform, the tag was not detected anymore by the ZED camera. To spot the so called "worst" situation, a *flag* variable was activated whenever there was the need to move the platform and the car was not parallel to the railway so that, after reaching y_{goal} if no tag was detected, the docking was based on the last saved position of the tag itself. Mathematically, this situation arose whenever $|\theta| > 30\,deg$ and $|y_{goal}| > \delta$.

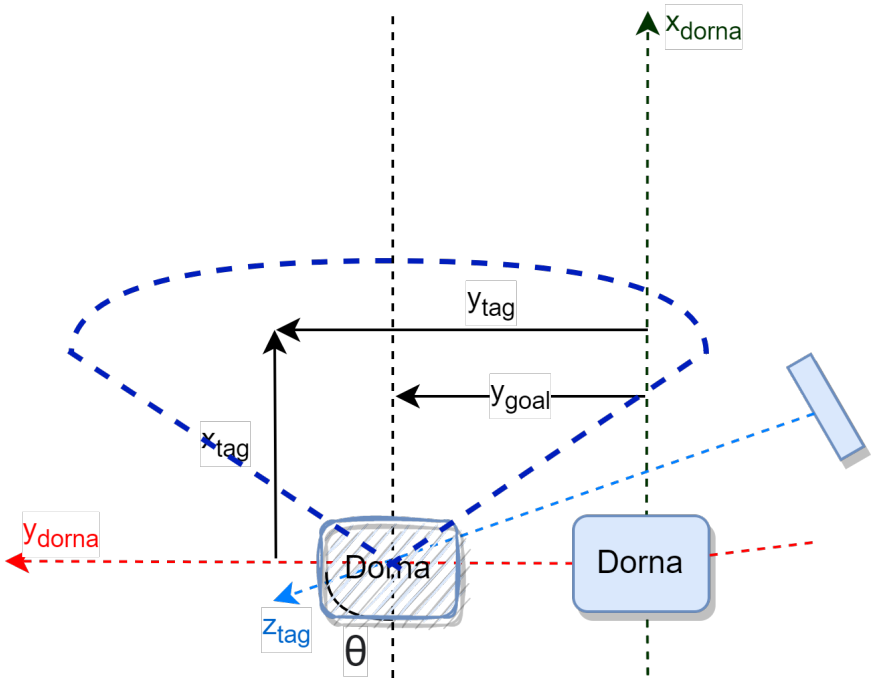

Figure 3.22: Worst case that could arise

For the sake of clarity, in Fig.3.22 the worst situation is illustrated, where the dotted blue line represents the field of view of the ZED camera that is 120 deg. In this case, the transparent Dorna block is the final position of the platform upon moving along y_{dorna} by y_{goal}. Finally, the missing part of the algorithm scheme can be shown in Fig.3.23.

To sum up and to show how in ROS the filtering node was connected to the other ones and which kind of information it exchanged through the subscription-publishing mechanism, Fig.3.24 illustrates the connections.

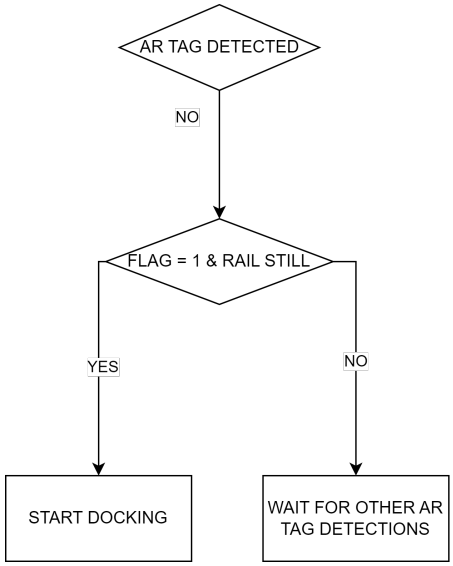

Figure 3.23: Worst case algorithm

It is possible to list all the used topics such as:

- **/zed_pose**: carrying a *PoseStamped* message giving position and orientation of the ZED camera, used to understand whether the rail was moving or not.
- **/ar_pose_marker**: carrying a *AlvarMarkers* message which structure is illustrated in Fig.3.19, giving position and orientation of the tag.
- **/fitered_ar_ypos**: carrying a *Float32* message providing the information of y_{goal} to the Arduino node to activate the stepper motor.
- **/starting_docking**: carrying a *Boolean* message giving the permission to Dorna Docking node to start the docking phase.
- **/filtered_ar_pos**: carrying a *Float32MultiArray* message providing to Dorna Docking node the information about x,y and z position of the tag in the space.
- **/dorna_status**: carrying a *Bool* message carrying the information about the robot status: *True* it was moving, *False* it was not moving.

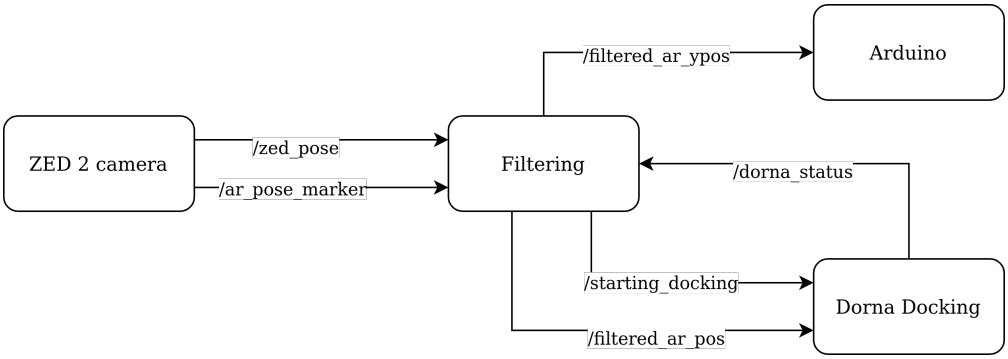

Figure 3.24: ROS graph-like representation of filtering node publications and subscriptions

3.4.3 Dorna Docking

The *Dorna Docking node* was the core of the automated charger system as it was responsible of managing the commands to move the robotic arm. In particular, whenever the filtering data node, described in the previous subsection, checking out the solvability of the inverse kinematic problem and addressing one of the presented cases, published the *ar_marker_pos*, the Dorna Docking node started running trying to move the robotic arm to reach the vehicle's socket. To be clear, referring to Fig.3.21 and Fig.3.23, the docking node received the message about the position and orientation of the tag from the *ar_marker_pos* topic whenever the algorithm reached the "Start Docking" block. In particular, before starting the docking phase, this node checked out whether the car was still or not tacking two consecutive messages from the filtering node and checking that:

$$|x_{ar}(k) - x_{ar}(k-1)| \leq \epsilon \tag{3.18}$$

$$|y_{ar}(k) - y_{ar}(k-1)| \leq \epsilon \tag{3.19}$$

$$|z_{ar}(k) - z_{ar}(k-1)| \leq \epsilon \tag{3.20}$$

where k indicates the time instant and $\epsilon = 0.254\,cm$ is a small tolerance constant

thanks to the really good precision of the ZED 2 camera. Whenever 3.18, 3.19 and 3.20 were satisfied, the *docking* function was called and the robotic arm started moving passing the command as a JSON format in the form show in Fig.3.25, where x_ar_d, y_ar_d and z_ar_d are $x_{ar}(k), y_{ar}(k)$ and $z_{ar}(k)$ respectively.

```
cmd = {"command": "move", "prm": {"path": "joint", "movement":0,
       "speed": 1000, "x":x_ar_d,"y":y_ar_d,"z":z_ar_d}}
```

Figure 3.25: Robot command in JSON format

Along with this functionality, this node was responsible for publishing all the information of the Dorna robotic arm status. Indeed, as described in Subsection 3.4.2, the filtering data node needed to check whether or not the robot was still. To provide such a piece of information, multithreading was exploited as it is important for the system to be as responsive as possible. To this extent, different threads were used to manage both the function providing the robot information and the one responsible of the robot movements.

Moreover, this node was responsible for stopping the robot whenever an obstacle was detected. This functionality was implemented through three different tasks activating whenever a message about the detection of an obstacle was published by the Arduino node managing the ultrasound sensors. These functions were intended to run a command to stop the robotic arm and to erase the movement command shown in Fig.3.25. For the detection of side obstacles, *obstacle_detection_left* and *obstacle_detection_right* functions were used so that if any obstacle was detected in a range of 7 cm the robot had to stop. For the front obstacle the reasoning was a bit more tricky as it could be either an obstacle or the car itself. Starting with the assumption of modelling the rear part of the car with a perpendicular plane to the ground, and with the knowledge of the ultrasound sensor providing the smallest distance to the obstacle, the robotic arm had to stop whenever $dist_{obst} \leq 7\ cm\ \&\ dist_{ar-tip} \geq 7\ cm$. In the last condition, $dist_{obst}$ was provided by

the Arduino node while $dist_{ar-tip}$ was computed exploiting the euclidean distance between the tip of the robotic arm provided by the odometry of the Dorna robot and the position in the space of the tag provided by the filtering node.
To sum up and to show how in ROS the Dorna Docking node was connected to the other ones and which kind of information it exchanged through the subscription-publishing mechanism, Fig.3.26 illustrates the connections.

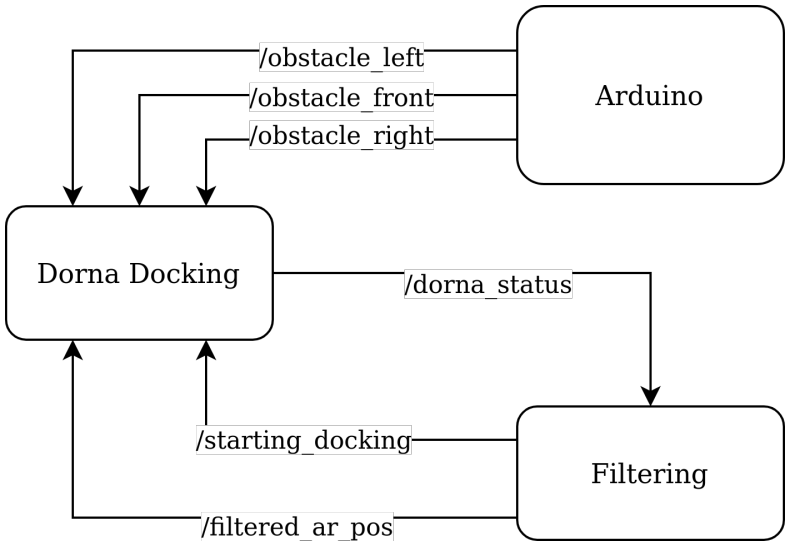

Figure 3.26: ROS graph-like representation of Dorna docking node publications and subscriptions

It is possible to list all the used topics that has not been discussed before, such as:

- **/obstacle_front**: carrying a *Range* message giving the shortest distance to the front obstacle.
- **/obstacle_right**: carrying a *Range* message giving the shortest distance to the right-side obstacle.
- **/obstacle_left**: carrying a *Range* message giving the shortest distance to

the left-side obstacle.

In Fig.3.27, the algorithm flow is shown. It is important to highlight that whenever either the docking was completed or an obstacle was detected, the *homing* phase was executed to bring the robotic arm to a safe position where it will be waiting for another detection and for a safer docking.

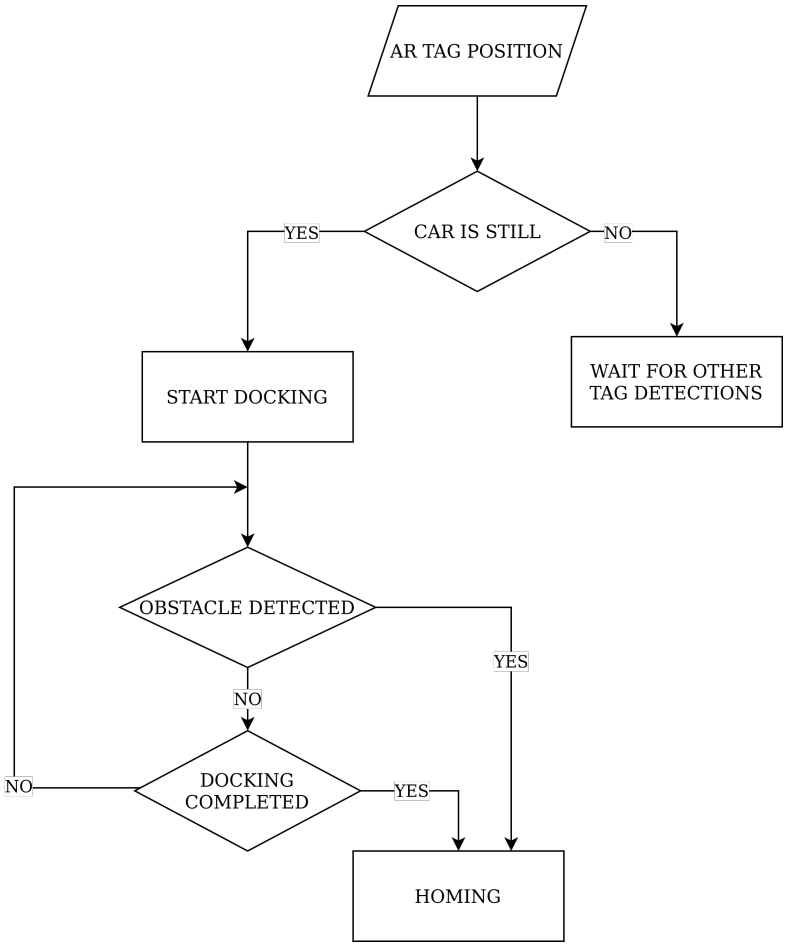

Figure 3.27: Dorna Docking algorithm flow

3.4.4 Arduino Node

As anticipated before, the *Arduino node* was responsible for controlling the stepper motor to start the movement of the platform, where the robotic arm was mounted, along the rail and to get information from the ultrasound sensors. In particular, the source code was written through *Arduino IDE* and the *rosserial node* was responsible of running it into ROS environment.

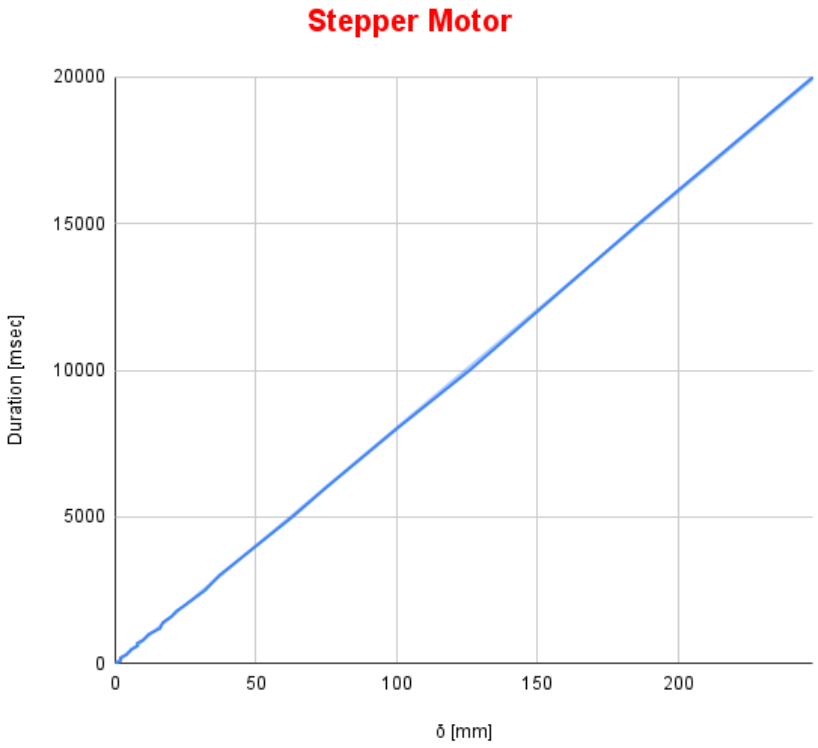

Figure 3.28: Stepper motor behaviour plot

For the ultrasound sensors management, it was responsible of switching the dedicated PIN the sensor was attached to from LOW to HIGH or viceversa. As already explained in Subsection 2.1.3, the distance of the obstacle to the ultrasound sensor was computed through 2.1. The same methodology was applied for all the

ultrasound sensors: front, right and left one. The three different functions were evaluated one after the other and not all together since the processing time of each of them was almost negligible considering the application. Moreover, to avoid any problem of responsiveness the speed of docking of the robotic arm was properly lowered.

Along with this, the Arduino node was responsible for the stepper motor moving the platform. First, a study was addressed trying to figure out which kind of control law to apply to the stepper motor to be as precise as possible. Indeed, different measurements were taken about the platform movement along the railway to understand how to minimize the disturbances as much as possible. The nut-screw system used to move the platform was subject to many inaccuracies such as the friction. Different tests were performed giving different commands to the stepper motor to understand how much time the PIN had to stay HIGH to reach a predefined position along the rail.

The results of the tests are shown in Table A.1, where *Duration* is the time the stepper motor control PIN stayed HIGH, *Starting* and *Final* positions are the starting and final measured positions of the platform and δ is simply the position displacement between the initial and final position. According to these data, it was possible to plot out the behaviour of the stepper motor and it is shown in Fig.3.28. It is clear that the control law to set the time the pin should stay HIGH is almost linear with the displacement δ and the relation that was used by the Arduino node was chosen as

$$t_{on} = 80.5\, y_{goal} - 11.5 \tag{3.21}$$

where y_{goal} is the position to reach moving the platform whose information was got through the */filtered_ar_ypos* topic and $k_1 = 80.5$ and $k_2 = -11.5$ were estimated using a Least Square Estimator (LSE). To sum up the connections with

the other ROS nodes and which kind of information was exchanged Fig.3.29 can be shown.

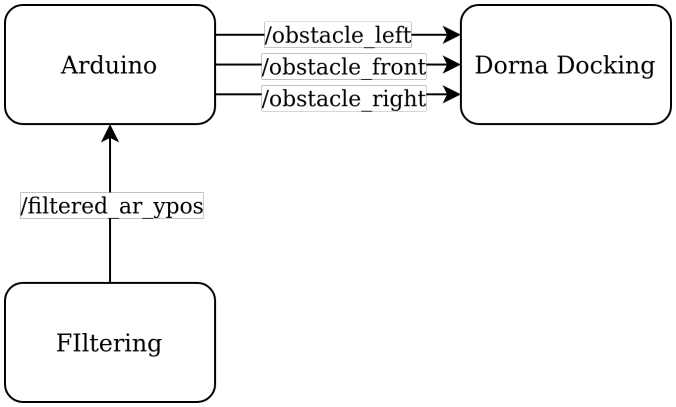

Figure 3.29: ROS graph-like representation of Arduino node publications and subscriptions

3.4.5 Obstacle Recognition

As described before, a possible implementation of *Obstacle Recognition* for implementing *Obstacle Avoidance* was developed. This algorithm has been just tried in simulation and through the usage of the depth camera sensor to understand whether it could work or not. In particular, the algorithm used for obstacle recognition went through different processes. The first one was the *Voxel Grid* allowing the PCB processing the images to save memory. Indeed, thanks to this approach it was possible to downsample an image, provided by the depth camera using the PointCloud library, dividing it in different voxels. The VoxelGrid class overlays the input point cloud data with a 3D voxel grid, which basically is a collection of tiny 3D boxes in space. The remaining points will then be downsampled with their centroid in each voxel. While it takes a little longer, this method more accurately captures the underlying surface than using the voxel's center to approximate them [26].

After this process, the usage of the *pcl::EuclideanClusterExtraction* allowed to differentiate inside the Point Cloud among different obstacles. An unorganized point cloud model P needs to be partitioned into smaller pieces using a clustering method in order to drastically reduce the processing time for P as a whole. An octree data structure, or more generally, a 3D grid subdivision of the space using fixed-width boxes, can be used to create a straightforward data clustering strategy in a Euclidean sense. The data in each resulting 3D box (or octree leaf) can be roughly approximated with a different structure, making this specific representation helpful in cases where either a volumetric representation of the occupied space is required.

From a mathematical point of view different clusters can be defined such as $O_i = p_i \in P$ and $O_j = p_j \in P$. Then the point evaluation can be addressed such as if

$$min||p_i - p_j||_2 \geq d_{th} \tag{3.22}$$

then p_i and p_j belong to different clusters. So basically, d_{th} can be considered as the radius of a sphere centered into one of the two points under examination. However, in a more general sense, we can use nearest neighbors and apply a clustering method that is basically equivalent to a flood fill algorithm [27]. In order to be as clear as possible, the algorithm flow will be presented in steps as follows:

1. Get a PointCloud P from a vision component like ZED 2 camera.

2. Create a K-d tree representation for the PointCloud P received from the sensor. The K-d tree method, a neighbours search algorithm, has been discussed in Subsection 2.4.2.

3. Declare a list of clusters C and a queue Q of the points that need to be checked.

4. Then a recursive algorithm has to be implement to manage the upcoming points $p_i \in P$ such that:

   - Add p_i to the Q queue to be analyzed.
   - For every point $p_i \in Q$ the following steps can be performed:
     - Search for the P_i^k of point neighbors of p_i in a sphere with radius $r < d_{th}$.
     - For every neighbor $p_i^k \in P_i^k$ check if the point has been already processed, if not add it to Q.
   - Add Q to the list of clusters C once the list of all points in Q has been processed, then reset Q to its initial state.

5. The algorithm stops when all the points provided by the sensors through PointCloud representation have been set into a list of point clusters C.

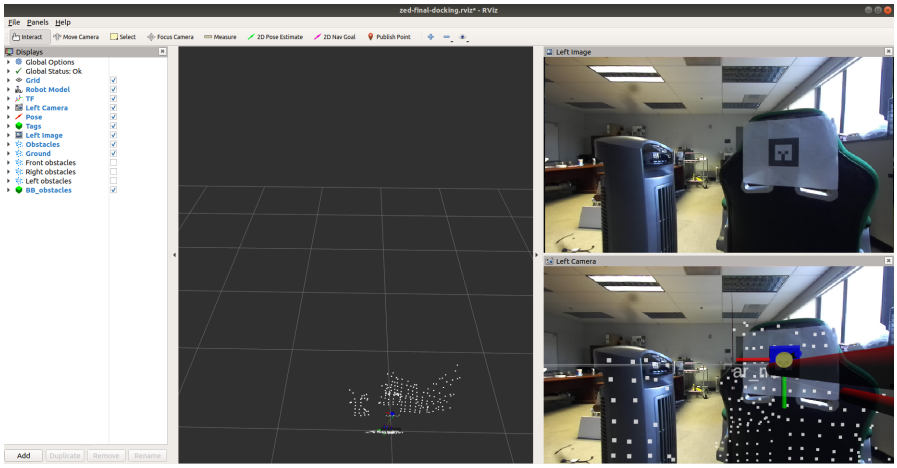

Figure 3.30: Obstacle recognition by the ZED 2 left camera in RVIZ through Point Cloud

In Fig.3.30, it is possible to appreciate the point cloud generated by the left camera of the ZED 2 camera, the recognition of two different objects thanks to the previously explained algorithm and the localization of the AR tag.

Chapter 4

Tests and Results

In this chapter some tests and results will be presented. Tests are fundamental to understand if the system respects the requirements fixed at the beginning and to see whether the software works in a real-case application. Indeed, mostly of the times some simulations are carried out without taking into account the undesired effects in real-case applications. In order to carry out some tests the automated charger systems has been mounted as shown by the hardware schema in Fig.3.14. All the pieces of hardware that have been used for the automated charger have been mounted as already explained in Subsection 3.3.3. Different factors could change in a real-case application and one has to ensure that it is going to work in every situation. Regarding the project, for instance, so far just the case in which the tag plane is perpendicular to the ground has been described. So basically, an interesting situation was to test the automated charger system changing the angle with respect to the ground of the tag and to see whether the ZED 2 camera was still able to detect it or not. It is worth noticing that changing the inclination of the tag does not mean that the socket is inclined too. Indeed, in all these simulations the assumption was to keep the socket perpendicular to the ground and centered at the center of the tag. Thinking about the real-case application, the tag could be

put wherever close to the vehicle's socket, such as in the above part that mostly of the time can not be approximated by a perpendicular plane to the ground.
The idea was to perform two different tests:

- Test A: here all the cases presented in Subsection 3.4.2 changing the angle θ illustrated in Fig.3.20.
- Test B: here all the cases will be shown as well but showing how the results change as the angle ϕ changed. The just mentioned angle is shown in Fig.4.1.

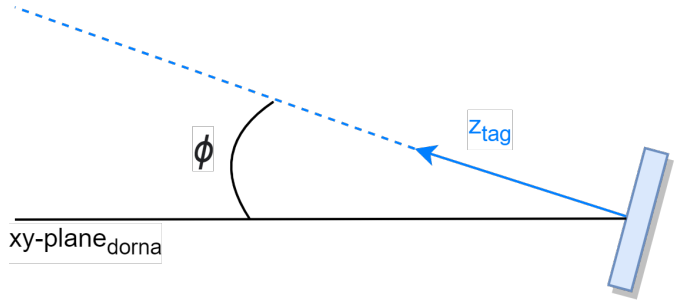

Figure 4.1: ϕ angle between the z_{tag} and the xy_{dorna}-plane

Test A Performing Test A, the angle between the z_{tag} and the $xy_{dorna} - plane$ was $\phi = 0\,deg$, so the tag plane was completely perpendicular to the ground plane. The displacement along the three main axes was measured between the end-effector reference frame described in Subsection 3.3.1 and the center of the AR tag.

As shown in Fig.4.2, the results were quite good reaching a maximum displacement of $4\,cm$. It is worth noticing that the figure has been divided into three part showing as for $\theta \geq 30\,deg$ the worst case arose. Though, it is clear that the worst results were obtained when the worst case was performed. This is what one should expect as in this situation upon moving the platform, the ZED 2 camera was not able to detect the AR tag and no current location of the socket was provided.

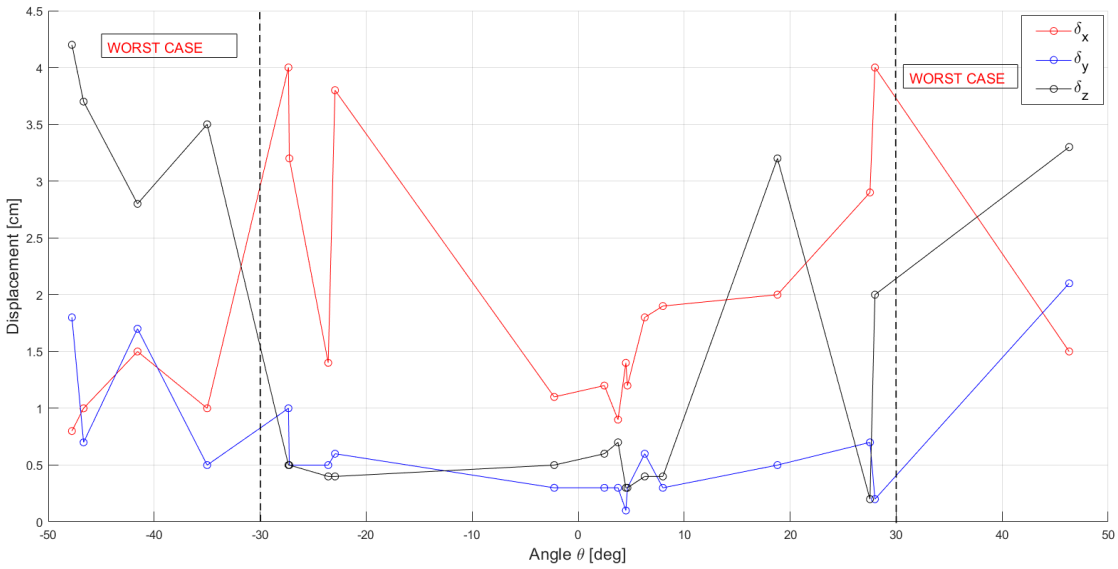

Figure 4.2: MATLAB plot showing the displacements along the three main axes as θ changes

It is possible to show better the performances of the system computing

$$\delta = \sqrt{\delta_x^2 + \delta_y^2 + \delta_z^2} \tag{4.1}$$

and the results can be shown in Fig.4.3.

Test B Instead in Test B, different values for ϕ were tried. Here it is important to highlight an important aspect about the chosen values. Indeed, it is important to simulate and test just the situation that could arise in the implementation of such a technology. To this extent, as the AR tag could be put above the socket and that part of the car if approximated by a plane, would face either the railway staying perpendicular to the ground plane or would face a bit up with a certain ϕ angle, no negative angles were tested. The results can be shown in Fig.4.4

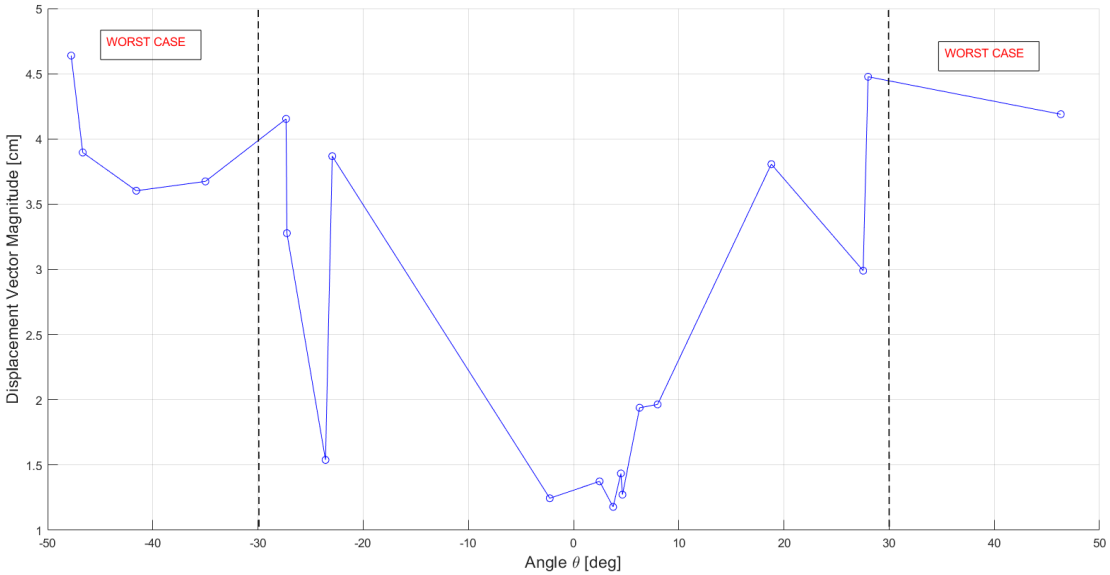

Figure 4.3: MATLAB plot showing the displacement vector magnitude as θ changes

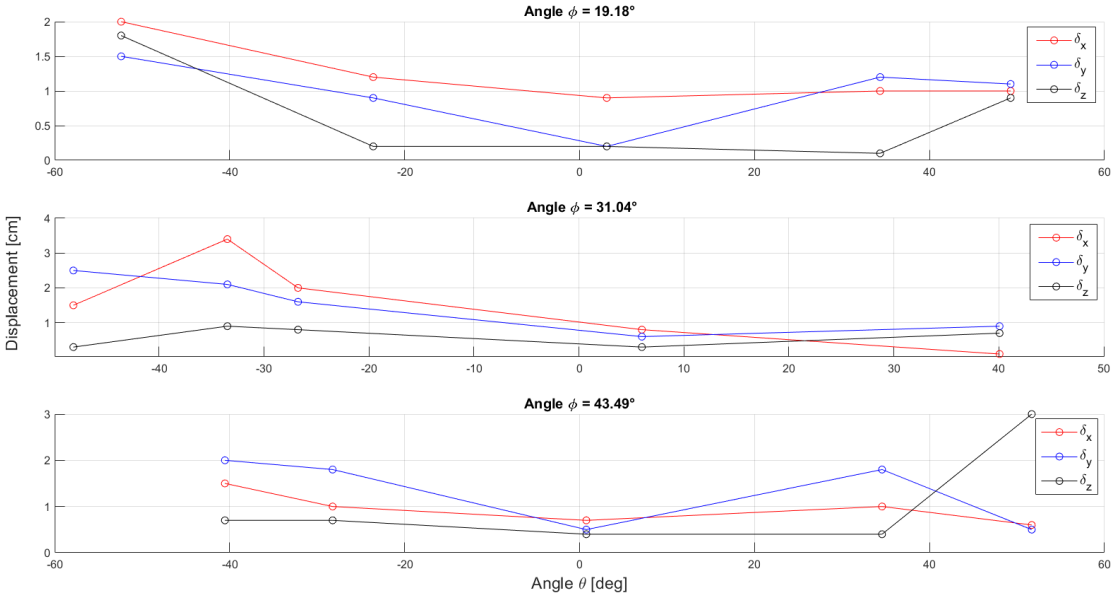

Figure 4.4: MATLAB plot showing the displacement vector magnitude as θ and ϕ change

Also in this case the performances seem to be better when $\theta \approx 0\,deg$ that is when the socket is almost parallel to the railway.

Chapter 5

Conclusions

In this thesis, the initial studies on the development of an Automated Robot-based Charger for EVs has been described. This is just the beginning part of the ambitious project that InnoTech Systems start-up has decided to invest on. With the future heading to a more and more autonomous world this can be considered as a starting point to get the right knowledge about the automation field and finding new technologies to make people's life easier.

The main goal of the project, as mentioned in the very first chapter of this work, was to develop an autonomous docking system intended to charge an EV without any human help. At the end, this goal has been achieved building from scratch both the hardware component and the software. To sum up, the robotic arm has been implemented as core part to accomplish the docking and it has been integrated with the use of a motorized railway proposing an efficient alternative to the spherical wrist. This aspect allowed avoiding the use of a 6-DOF robot, reducing the cost and developing knowledge about adding a degree of freedom to a robotic system. Then, as far the software part is concerned, the algorithm has been able to manage different pieces of information coming from multiple sensors and to process them in order to understand which position in the space had to be reached.

As conclusion of the described development, several future improvements may be suggested. As far as the hardware part is concerned, the robotic arm has some limitations in finding the connection while from a software point of view, a more detailed analysis of the process management and of how to properly organize the timing of the different tasks could be addressed. Moreover, the accuracy of the system could be improved, for instance using a UWB technology to understand how the socket is placed with respect to the plug. This would allow to create a closed-loop system taking the displacement to zero.
All considerations and this work as well, will hopefully one day become reality improving sustainability and automation.

Appendix A

Test Measurements

Duration [msec]	Starting Position [mm]	Final Position [mm]	δ [mm]
100	182	184	2
200	184	186	2
300	186	190	4
400	190	195	5
500	195	201	6
600	201	209	8
700	209	217	8
800	217	227	10
900	227	238	11
1000	238	250	12
1200	250	266	16
1400	266	283	17
1600	283	303	20
1800	303	325	22
2000	325	350	25
2500	350	382	32
3000	382	419	37
4000	419	469	50
5000	469	532	63
6000	532	607	75
8000	607	707	100
10000	707	833	126
15000	833	1019	186
20000	1019	1267	248

Table A.1: Measurements for stepper accuracy

θ [deg]	δ_x [cm]	δ_y [cm]	δ_z [cm]
-2.26	1.1	0.3	0.5
28	4	0.2	2
-27.25	3.2	0.5	0.5
4.65	1.2	0.3	0.3
3.76	0.9	0.3	0.7
-35	1	0.5	3.5
18.8	2	0.5	3.2
-23.58	1.4	0.5	0.4
46.32	1.5	2.1	3.3
-27.33	4	1	0.5
7.99	1.9	0.3	0.4
-41.57	1.5	1.7	2.8
4.5	1.4	0.1	0.3
2.47	1.2	0.3	0.6
6.28	1.8	0.6	0.4
27.53	2.9	0.7	0.2
-22.94	3.8	0.6	0.4
-46.66	1	0.7	3.7
-47.76	0.8	1.8	4.2

Table A.2: Automated charger system accuracy for $\phi = 0\,deg$

θ [deg]	δ_x [cm]	δ_y [cm]	δ_z [cm]
3.11	0.9	0.2	0.2
34.35	1	1.2	0.1
-23.59	1.2	0.9	0.2
49.27	1	1.1	0.9
-52.4	2	1.5	1.8

Table A.3: Automated charger system accuracy for $\phi = 19.18\,deg$

θ [deg]	δ_x [cm]	δ_y [cm]	δ_z [cm]
6	0.8	0.6	0.3
40.07	0.1	0.9	0.7
-33.48	3.4	2.1	0.9
-26.75	2	1.6	0.8
-48.15	1.5	2.5	0.3

Table A.4: Automated charger system accuracy for $\phi = 31.04\,deg$

θ [deg]	δ_x [cm]	δ_y [cm]	δ_z [cm]
0.77	0.7	0.5	0.4
34.57	1	1.8	0.4
-28.23	1	1.8	0.7
51.68	0.6	0.5	3
-40.54	1.5	2	0.7

Table A.5: Automated charger system accuracy for $\phi = 43.49\,deg$

Bibliography

[1] *The Global electric vehicle market overview in 2022: statistics & forecast.* URL: https://www.virta.global/en/global-electric-vehicle-market (cit. on p. 1).

[2] B. Walzel, C. Sturm, J. Fabian, and M. Hirz. «Automated robot-based charging system for electric vehicles». In: 2016. URL: https://www.researchgate.net/publication/299392348_Automated_robot-based_charging_system_for_electric_vehicles (cit. on pp. 2, 5).

[3] *e-smartConnect: Volkswagen is conducting research on an automated quick-charging system for the next generation of electric vehicles.* URL: https://www.volkswagen-newsroom.com/en/press-releases/e-smartconnect-volkswagen-is-conducting-research-on-an-automated-quick-charging-system-for-the-next-generation-of-electric-vehicles-1512 (cit. on p. 3).

[4] Fred Lambert. *Tesla's robot snake charger is not dead yet.* URL: https://electrek.co/2020/10/09/tesla-robot-snake-charger-not-dead/ (cit. on p. 3).

[5] L. Villani B. Siciliano L. Sciavicco and G. Oriolo. *Robotics : Modelling, Planning and Control.* Springer (cit. on pp. 6, 14).

[6] Wikipedia. *Stereo camera*. URL: https://en.wikipedia.org/wiki/Stereo_camera (cit. on p. 15).

[7] Danny Jost. *What is an Ultrasonic Sensor?* URL: https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor (cit. on p. 16).

[8] Rob Giseburt. *Jerk Controlled Motion Explained*. URL: https://github.com/synthetos/g2/wiki/Jerk-Controlled-Motion-Explained (cit. on p. 34).

[9] Teck-Chew Ng Kim Doang Nguyen and I-Ming Chen. *On Algorithms for Planning S-Curve Motion Profiles*. URL: https://journals.sagepub.com/doi/full/10.5772/5652 (cit. on p. 35).

[10] Velodyne Lidar. *What is lidar?* URL: https://velodynelidar.com/what-is-lidar/#:~:text=Lidar (cit. on p. 36).

[11] Charles G. Torre. *09 The Wave Equation in 3 Dimensions*. URL: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1009&context=foundation_wave (cit. on p. 37).

[12] Hai Zhu Jiahao Lin and Javier Alonso-Mora. *Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments*. URL: https://www.researchgate.net/figure/obstacle-detection-based-on-depth-image-and-the-u-depth-map-a-schematic-of-the-camera_fig3_339228265 (cit. on p. 39).

[13] Wikipedia. *Point cloud*. URL: https://en.wikipedia.org/wiki/Point_cloud (cit. on p. 39).

[14] Creative Commond Attribution 3.0. *Point Cloud Library (PCL)*. URL: https://pointclouds.org/documentation/group__kdtree.html (cit. on pp. 40, 43).

[15] Ming Zhao and Xiaoqing Lv. *Improved Manipulator Obstacle Avoidance Path Planning Based on Potential Field Method*. URL: https://downloads.hindawi.com/journals/jr/2020/1701943.pdf (cit. on p. 48).

[16] Massimo Violante. *Process Management - Basic concepts* (cit. on p. 50).

[17] Wikipedia. *Thread (computing)*. URL: https://en.wikipedia.org/wiki/Thread_(computing) (cit. on p. 52).

[18] Digital Guide IONOS. *Multithreading: More performance for processors*. URL: https://www.ionos.com/digitalguide/server/know-how/multithreading-explained/ (cit. on p. 52).

[19] ROS.org. *Understanding ROS Topics*. URL: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics (cit. on p. 55).

[20] ROS.org. *Topics*. URL: http://wiki.ros.org/Topics (cit. on p. 55).

[21] STEREOLABS. *Getting Started with ROS and ZED*. URL: https://www.stereolabs.com/docs/ros/ (cit. on p. 63).

[22] eLinux. *Jetson Nano*. URL: https://elinux.org/Jetson_Nano (cit. on p. 63).

[23] Arduino. *Arduino® MEGA 2560 Rev3*. URL: https://docs.arduino.cc/static/bd49a88997b4590197ac00ed9393d951/A000067-datasheet.pdf (cit. on p. 64).

[24] Wikipedia. *ARTag*. URL: https://en.wikipedia.org/wiki/ARTag (cit. on p. 75).

[25] ROS.org. *ar_track_alvar*. URL: http://wiki.ros.org/ar_track_alvar (cit. on p. 76).

[26] *Downsampling a PointCloud using a VoxelGrid filter*. URL: https://pcl.readthedocs.io/en/latest/voxel_grid.html (cit. on p. 90).

[27] *Euclidean Cluster Extraction*. URL: https://pcl.readthedocs.io/en/latest/cluster_extraction.html (cit. on p. 91).