Federal University of São Carlos– UFSCar
Center for Exact Sciences and Technology– CCET
Departamento de Engenharia Elétrica– DEE
Programa de bacharelado em Engenharia Elétrica

Rafael Gomes da Silva

Obstacle detection and avoidance for a mobile robot

São Carlos
2020

Rafael Gomes da Silva

Obstacle detection and avoidance for a mobile robot

Report of the project done for my Undergraduate Thesis, submitted to the Bachelor of Electrical Engineering program at the Center for Exact Sciences and Technology of the Federal University of São Carlos, as part of the requirements for obtaining a Bachelor of Electrical Engineering title.

Specialization area: Robotics and Complex Systems

Advisor: Faïz Ben Amar
Co-advisor: Roberto Santos Inoue

São Carlos
2020

Confidentiality Notice

This present document is not confidential. It can be communicated outside in paper format or distributed in electronic format.

Acknowledgment

I would like to thank all the people involved during the development of this project, who directly or indirectly helped to acquire new and enriching skills and knowledge.

Special thanks to Mr. Ben Amar, team leader of the SYROCO "Complex Robotic Systems" group at ISIR, for allowing me to work for his team Syroco team at ISIR during my research internship, and also special thank to the Doctoral student Mohamed Fnadi for his time helping me to develop all the stages of this project. Thank all the ISIR laboratory staff who offered me the means, the structure, and a quality internship in the intelligent systems area, in which I always wanted to deepen my knowledge. And also thank you to professor Roberto Santos Inoue who reviewed my work to be presented for my undergraduate thesis presentation at UFSCar.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Resumo

Os robôs *off-road* são veículos complexos usados em diversas aplicações e são capazes de operar em diversos tipos de terrenos, incluindo terrenos acidentados. Por isso, sua aplicação vem crescendo cada vez mais nos dias de hoje.

Com a crescente importância do estudo e aplicação de veículos autônomos em áreas insalubres ou perigosas por razões de segurança aos usuários desses veículos, as pesquisas sobre esse tipo de veículo aumentaram nos últimos anos. Já foram feitos trabalhos com o objetivo de propor algoritmos para controladores preditivos não lineares, para melhorar a estabilidade do veículo, para seguir trajetórias e também para estimar a rigidez das curvas de contato em tempo real utilizando observadores não-lineares, mas há também a questão de como proceder se o robô encontrar um obstáculo que possa obstruir seu caminho enquanto ele estiver rastreando um caminho a ser seguido.

Assim, este trabalho tem como objetivo propor um algoritmo que permita a um robô móvel de quatro rodas, rápido, *off-road* e com dupla direção detectar os obstáculos presentes no terreno em tempo real usando o mapeamento dinâmico do ambiente, permitindo o desvio de obstáculos seguindo um caminho local criado usando uma curva de Bézier composta, otimizada com base na curvatura máxima que o robô pode executar.

Para os experimentos foram utilizados diversos sensores de posicionamento e percepção incluindo um sensor Lidar (*Light Detection And Ranging*) *Velodyne HDL-32E*. O tratamento da nuvem de pontos fornecida pelo mesmo foi tratada utilizando principalmente a biblioteca *PCL*. Por motivos de tempo do estágio, os testes realizados foram feitos principalmente em ambiente virtual considerando diferentes tipos de trajetória a serem seguidas pelo *SPIDO*, com obstáculos posicionados no caminho. Os resultados finais obtidos se mostraram satisfatório com relação ao esperado, concluindo assim a validade do algoritmo proposto.

Palavras-chave: Nuvem de pontos, Evitamento de obstáculos, Curvas de Bézier, Robôs *off-road*

Abstract

Off-road robots are complex vehicles used in a variety of applications and are capable of operating over rough terrain, and its application has been growing more and more nowadays.

With the growing importance of the study and application of autonomous vehicles in rough areas for the users' safety reasons, researches concerning this kind of vehicle have increased. Works were already done to propose algorithms for non-linear predictive controllers, for the improvement of the stability of the vehicle, for path following and also for a nonlinear observer to estimate the contact cornering stiffness in real-time, but there is also the question of how to proceed if the robot encounters an obstacle that could obstruct its path while it is tracking a path.

Thus, this work aims to propose an algorithm that allows a four-wheel, fast off-road, double-steering mobile robot to detect obstacles from the terrain in real-time using the dynamic mapping of the environment, and that also allows the robot to avoid obstacles by following a local path created using a composite Bézier curve, optimized based on the maximum steering that the robot can perform.

For the experiments, a sensor for position and perception were used, including the Lidar (Light Detection And Ranging) *Velodyne HDL-32E*. The treatment of the point cloud provided by it was treated using mainly the *PCL* library. For reasons of internship duration, the tests performed were done mostly in a virtual environment considering different types of trajectory to be followed by the *SPIDO*, with obstacles positioned along the way. The final results obtained were satisfactory concerning the expected, thus concluding the validity of the proposed algorithm.

Keywords: Point cloud, Obstacle avoidance, Bézier Curves, Off-road robot

List of Figures

Contents

Chapter 1

Introduction

1.1 The internship

The present project was developed during the research internship that I did as part of the curriculum of the French university in which I did my double degree program (ENSTA Paris), from May 13^{th} 2019 until July 31^{st} 2019. This internship aims to present to the students the research and development world by working in a project that involves initiative, innovation, and inductive reasoning.

I did my research internship at the Institute of Intelligent Systems and Robotics (ISIR), a joint research unit in the Pierre and Marie Curie University (UPMC) in Paris. In this context, my work was done within the SYROCO team, which focuses on the development of design and control methods for mobile robotic and manipulation systems.

1.2 Scientific context

Off-road robots are complex vehicles used in a variety of applications, such as military, agricultural, and leisure applications. This type of vehicle is capable of operating over rough terrain in a variety of weather conditions without a great compromise of its structure.

The term off-road means that the vehicle must have a structure that allows it to be able to ride in a location without any urban structure, that means, have a structure that allows it to drive even in unpaved and difficult to reach places.

Thus, works were already done in the design of non-linear predictive controllers to ensure the mobile robot to move in a natural environment at high velocity and following a reference path [1], the improvement of the stability of fast rover moving in a high-speed

[2], path following LQR controller with good accuracy for high-speed motion [3] and also a nonlinear observer designed to estimate the front and rear contact cornering stiffnesses in real-time [4].

However, besides the question of how to efficiently perform a tracking path, there is also the question of how to proceed if the robot encounters an obstacle that could obstruct its path. In this project, a study on the capture of environmental information through sensors and the use of a little-explored avoidance path generation technique is proposed, aiming to decide which decisions a robot should make when the path is obstructed by an obstacle.

Trajectory planning for robots is a broad area of study, and the main objective of this topic is to provide the robot with a reference to be followed during a given task so that it can achieve the desired objectives for its purpose. However, the choice of the path to be followed should be made selectively, as each robot has limitations concerning its degrees of freedom and constraints of movement due to constructive factors, and if these limits are not taken into account during the path planning process, it may either inefficiently follow the suggested path or may even become damaged when attempting to perform an improper movement.

Therefore, one of the most important characteristics that should characterize a planned trajectory is that the curve to be used must be smooth and continuous, and it has to have adequate curvature to the limits of the robot's steering angles and maximum speed, not presenting sharp curves to allow the robot to follow it without stopping.

Several works have studied special curves that generate a smooth and continuous path, such is the case of the study of clothoide curves done in [5], [6] or a variant of the clothoides done in [7], which generates several possible avoidance paths and selects the best curve based on optimization criteria. Some works study the use of the potential field technique to avoid obstacles, as done in [8], [9], and [10]. This technique consists of considering an imaginary force resulting from the influence of objects that draw the robot to its ultimate goal (objects that are not considered as obstacles), and objects that repel the robot (objects considered as obstacles).

This project aims to explore Bézier curves for the generation of real-time avoidance paths to verify if their use allows a robot to follow a local obstacle avoidance path. This curve when well parametrized can meet all the requirements mentioned above and can be useful to the obstacle avoidance path.

1.3 Study issues

Figure 1.1 shows the general block diagram of the main system that makes up the mode of operation of the off-road robot platform being used. As can be seen, it is composed of a low level and a high-level subsystem. The low-level part is responsible for receiving the

good steering angles and the linear speed that the vehicle must perform, necessary for the robot to continue following the proposed path. This step will not be the subject of study in this project since several works have already been done to improve and validate this stage.

The other, the high-level subsystem, is responsible for performing data processing and implementing control algorithms that return command law variables to the Spido. This subsystem has a stage of informing to the robot what is the reference path to be followed ("Reference Path" block), and it will be in this part of the system that the algorithms developed in this project will be applied. The Figure 1.2 shows more details about this block.

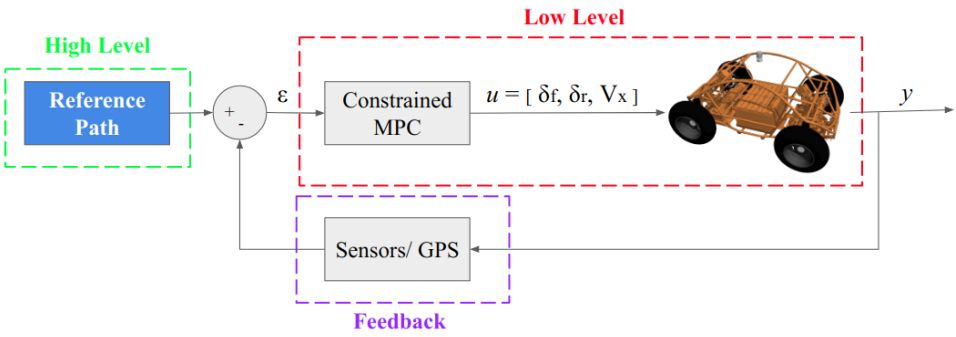


Figure 1.1 – Block diagram of the overall system

The work will be divided into different steps, including data acquisition, data down-sampling, and data processing. Initially, it is necessary to capture and process the data information of a point cloud coming from the LIDAR. This processing stage includes a re-sampling step that decreases the amount of data to be worked on and increases the algorithm calculation time.

After reducing the amount of data, it is necessary to separate the captured scene into different unit elements and to classify which of these elements are considered obstacles and which are not obstacles to the robot.

After identifying the obstacle, it is necessary to communicate to the command law control stage that an avoidance path must be generated, the algorithm proposed will compute an alternative route to avoid the obstacle if there is the possibility to avoid the obstacle, otherwise, the robot will stop.

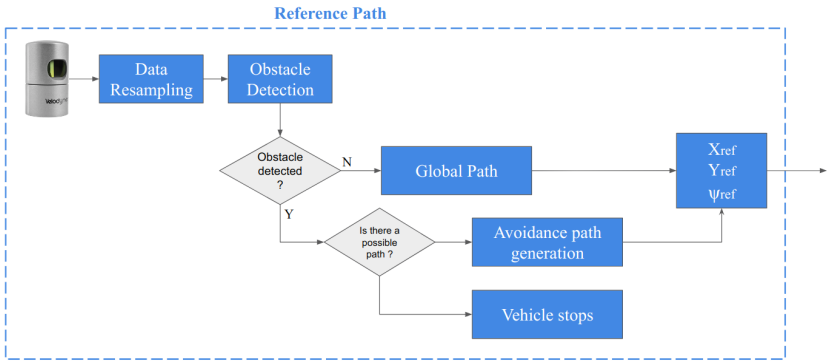


Figure 1.2 – Block diagram of the part considered on this project

The above process is iterative and performed in real-time, and uses a dynamic update of the scene to be handled rather than using a static mapping. With static mapping, there's a need to know the field of work before and store the data in the local computer, which can consume more time and more memory. On the other hand, with the dynamic mapping, the information about the scene is updated at the same rate as the data update frequency of the LIDAR.

1.4 Objectives

The objective of the present project is to create an algorithm that allows a four-wheel, fast off-road, double-steering mobile robot to detect obstacles that could block it from following a global path, and to create a local planned path using Bézier curves, by analyzing the information obtained through the vehicle's sensors. In order to do so, this work will cover the treatment of a 3D point cloud coming from a LIDAR sensor, the detection of obstacles, and the search and optimization of avoidance paths, always taking into account constraints and considerations for solving the problem proposed, such as:

- Flat ground
- Static obstacles
- Field opened and clear
- Known depth of the obstacles

1.5 Robot Operating System (ROS)

Robot Operating System (ROS) is a tool that structures, standardizes, and manages the communication in a robotic network system where there is a need to establish communication among various types of equipment. It is an Open Source tool that operates with

Linux, using mainly C ++ and Python as the programming language and that communicates between the system units through the TCP/IP protocol. Every robot network that uses ROS as a communication structure has a master machine that initializes the services necessary for the system to function properly and so that the machines can communicate with each other.

In this project, ROS was used to work with the different modules of the off-road mobile robot such as the LIDAR, GPS, IMU, in order to facilitate communication with different modules, as well as to enable the processing of data and control the steering angles of the vehicle to keep it within the desired path.

Furthermore, there are 3 important ROS frameworks to mention: Nodes, Topics, Packages.

1.5.1 Nodes

Nodes are composed of the active programs and executables used by each machine, and these nodes can send information to the network to a specific topic (publisher node) or they can read data that is coming from a specific topic (subscriber node). In this project, the use of nodes was important, for example, to interface sensors and actuators, to navigate through and map the terrain, to do the communication between various mobile robot embedded devices, and also to perform the virtual experimentation of the algorithms created.

1.5.2 Topics

Topics are responsible for storing the messages circulating through the ROS core (master) in order to allow them to be read by a subscriber node or changed by a publisher node. Topics can be understood as global network variables, which can be accessed by all the connected elements. Using the terminal, you can get different information about the messages being exchanged on the network through commands such as *$rostopic list* (that shows all system topics), *$rostopic pub* (that sends data to a specific topic) or *$rostopic echo* (that visualizes data from a specific topic).

1.5.3 Packages

ROS packages are composed of an arbitrary number of nodes that allows the users to have a more organized structure and they are used to give the network different modules with full functionality.

Although the structure of the ROS is complex to understand, it ensures that all system units will communicate in a consistent and standardized manner to avoid any

information and data conflicts. The Figure 1.3 obtained with the *rxgraph* command shows a simple example of how communication occurs between the various system nodes within the default example from ROS named "turtlesim", where the node "/ turtlesim" is the master node. With this command, it is possible to visualize which nodes are active and how they interact with each other on the network.

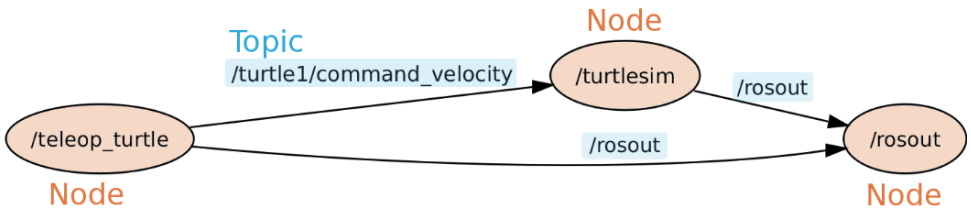


Figure 1.3 – ROS computation graph for the turtle-sim example. Source: [11]

1.6 Experimental platform

The Spido Robot (Figure 1.4) was the experimental platform used during this internship. It is a four-wheel, fast off-road, double-steering robot, having a total weight of around 700Kg and that can reach the maximum speed of 12m/s. It is equipped with embedded computers that treat the data of sensors that provide information about its localization, its orientation, and that gives information about its surroundings.

The communication between the embedded computers and the sensors is made using Ethernet cables and the protocol of communication TCP/IP. Nevertheless, it is also possible to communicate with the NUC unity via SSH protocol, in order to get the processed data and all the log files generated by the system while running the tests.

1.6.1 Embedded computers

This experimental platform has two embedded computers, that are used to process the received information and control the robot. One computer (the NUC computer, as shown in Figure 1.5) is responsible for processing the data and signals acquired from the sensors of the platform, including the LIDAR sensor and the positioning sensors. It is in this computer that the algorithms of treatment of the point cloud are stored and it is also in this computer that the information about the control of the robot is made using publisher nodes. The other computer is responsible for subscribing to the topics having the information with the command law that will assure that the platform will follow either the global path or the local path, by injecting in the robot the right steering angles and the right linear speed for the Spido.

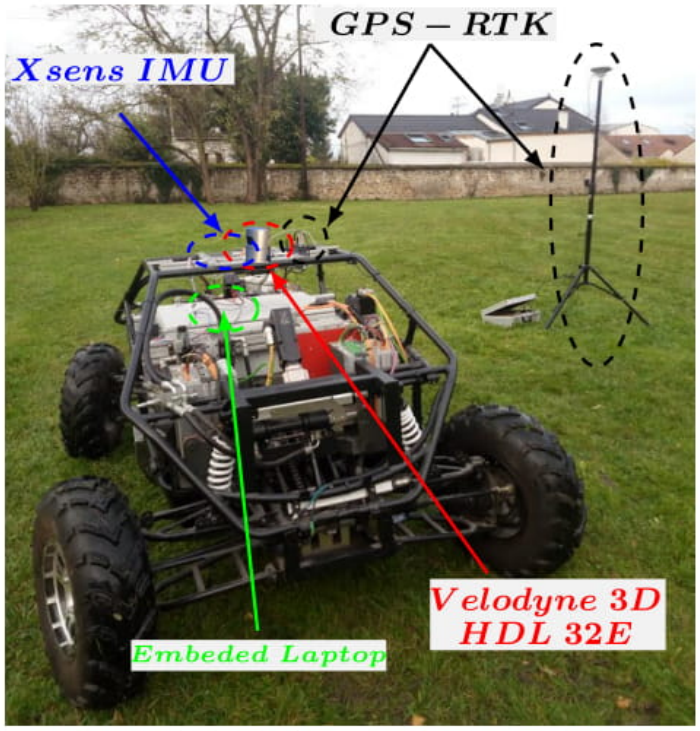


Figure 1.4 – Experimental platform with the embedded sensors

Figure 1.5 – NUC

1.6.2 Real Time Kinematic GPS (GPS RTK) and Inertial Measurement Unit (IMU)

GPS RTK is a positioning technique that provides the location coordinates more accurately when compared to the GPS we use in our daily lives, such as the ones in our cell phones. The main difference is in the way coordinates are measured: the RTK GPS has a base receiver that stays static at a known coordinate and sends positioning correction for the mobile receiver, based on the principle of triangulation. The accuracy of an RTK GPS is around 2cm, while the accuracy of a conventional GPS ranges from 5m to 10m.

The IMU is an equipment having three main sensors: accelerometer, gyroscope, and a magnetometer. With the measures obtained with the IMU, it is possible to determine

if the experimental platform is moving, rotating, or inclined.

These elements will not be discussed with more details in this report because they do not represent an important part of my internship and are mainly used by the low-level part (path tracking) and for setting the control points of the avoidance curve (which will be discussed in Chapter 4.4)

1.6.3 LIDAR Velodyne HDL-32E

The LIDAR (Light Detection and Ranging), as shown in Figure 1.6, is a sensor used in order to perceive the environment through the emission, reception, and treatment of laser beams signals, and it is used in different areas of application such as autonomous vehicles and robotic vision.

Figure 1.6 – Lidar used in this project

This type of sensor uses time-of-flight (ToF) methodology, that means, each laser beam of the equipment sends an optical signal modulated by a transmitter until it hits the scene intended to extract 3D information, and the reflected light is detected by the paired detector, which determines the flight time and the energy received. By emitting these light pulses with a certain update rate, LIDAR sensors can create a position (x, y) and elevation (z) data set, resulting in a high-density point cloud and reproducing the scanned environment in 3D with a high level of detail.

This project used a LIDAR Velodyne HDL-32E, which has the following technical characteristics:

- 32 lasers;
- Update frequency of 10Hz;
- ± 2 cm accuracy;
- Up to ∼1.39 million points per second;
- 360° Horizontal field of view;
- +10° to -30° Vertical field of view;

1.7 Point Cloud Library (PCL)

The Point Cloud Library (PCL) is an open-source C ++ language software, compatible with various operating systems (such as Linux and Windows), and contains a series of small modules that can be used separately to work with a point cloud using algorithms including filtering, feature estimation, segmentation, etc.

A point cloud is a structured data set made up of a multidimensional point collection and is commonly used to represent three-dimensional data [12]. That means, a point cloud is a set of data $P = \{p_1, p_2, p_3, ...\}$, having $p_i = \{x_i, y_i, z_i, ...\}$ as attributes, which are used to represent the 3D information of the points. In addition to the XYZ coordinate position of each point, a point cloud can have additional information such as color and intensity.

In this work, the PCL was used to process the data coming from the LIDAR sensor, in order to downsample it and develop the algorithm for the obstacle detection stage.

1.8 Virtual platform

Since the robot that is being used as an object of study in this project is a complex robot, performing all the first tests directly on the real robot could bring a risk to the integrity of the vehicle if any major error occurred during the application of the algorithms developed. Thus, before performing any experimental test it was necessary to validate in simulation each step, and once validated, the algorithm was applied to the real robot. The three main elements important for the virtual simulation were the virtual world, the robot's file description, and the sensor's visualization software.

1.8.1 Gazebo

For the simulation tests, the Gazebo virtual environment was used. Gazebo is a widely used 2D/3D environment simulator that offers the possibility to efficiently simulate various types of robots in complex environments, and its results are very close to the results obtained in the real environment, showing, thus, that one of the benefits of its use is the ease of switching between the real and the simulated world. The simulation in the Gazebo's environment takes into account many important aspects of both the robot and the environment in which it is being simulated. For instance: dynamic and kinematic modeling, soil type, interactions with the environment (such as collisions). Those are elements that are relevant for the validation of the project's simulation.

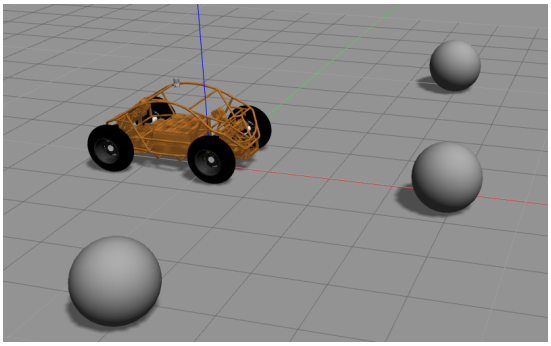
1.8.2 Unified Robot Description Format (URDF)

The model specifications are made based on a file written in a Unified Robot Description Format (URDF) file, which is a file used in a ROS environment that gives an XML specification that allows the configuration of the parameters that will perform the virtual representation of a robot. The model description consists of the definition links (which describes the characteristics of an inertial rigid body) and the set of joints (describes the kinematics and dynamics of a joint).

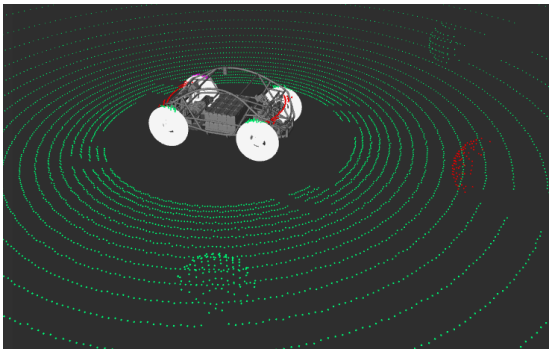
1.8.3 ROS Visualizer (Rviz)

Finally, to visualize the information and data obtained through the simulation, the ROS Visualizer (Rviz) is used. Rviz is a tool used to display various important information such as data acquired by simulation robot sensors, ROS status, and URDF model information of the robot, joint position, and can display data from camera devices, laser, 3D and 2D, including images and point clouds.

Figure 1.7(a) shows the Spido's model created to be simulated in a Gazebo world, as well as three spheres placed in the environment to interact with the sensors and the vehicle itself. Figure 1.7(b) shows the Spido's model displayed in the Rviz software, and the points in green are the point cloud detected by the virtual LIDAR present in the vehicle's model. As it is possible to see, the Rviz software and the Gazebo world can interact with each other in the sense that it is possible to visualize with Rviz the changes happening in the gazebo world.



((a)) View of the Spido on Gazebo



((b)) View of the Spido on Rviz

Figure 1.7 – View of the Spido on the virtual platforms

1.9 Contributions

During the development of this project, my main contributions were made in the high-level stages of the system, that means, the parts related to the acquisition and treatment

of the information obtained from the environment detection sensors since the low-level part related to the vehicle control had already been developed, tested and validated in previous works [3].

Thus, I was able to work on the development of a fast and real-time obstacle detection method, in which it is not necessary to have prior knowledge of the cartographic map of the environment in which the robot needs to navigate, requiring only that the test conditions were controlled and were within the presumed restrictions and restrictions. In this context, the obstacle to be detected is an object that is located in a position that will interfere with the displacement of the robot in its initially planned global path.

The development of this method was made using the LIDAR sensor point cloud, treated using messages and filters of the PCL library, and with this processed information it is possible to plan a local obstacle avoidance path using Bézier cubic curves.

Chapter 2

3D point cloud processing

2.1 State of the art of obstacle detection methods

Obstacle detection is an important step in the development of an autonomous robot since it is responsible for recognizing and processing the spatial and physical characteristics of the environment in which the robot is inserted, and without this step, it is impossible for the robot to make important decisions to interact with the environment, such as moving toward a goal point or moving away from objects or people to avoid collisions.

In order to perform such detection, a robot must be equipped with sensors capable of reading the environment, either through cameras, modulated optical signals, or any other signals that may somehow add a sensory vision function to the robot.

In addition to being able to acquire information about the surroundings of the environment, the robot must also be provided with algorithms for processing this information, so that it can retain characteristics of interest and make the necessary decisions to achieve a given goal.

In the case of an autonomous mobile robot, the main need that must be met with a vision system is to identify obstacles and make decisions about what actions to take. Currently, in the literature there are several studies related to the acquisition and treatment of the presence of obstacles for mobile robots, using several algorithms to classify objects that can be considered as obstacles, such as the convex hull algorithm studied in [13], the clustering techniques studied in [14] or [13], and also the construction of cost-maps to help the robot make decisions about where to go based on the knowledge of its surroundings, as studied in [15] and [16].

In this project, a combination of the cost-map principle of obstacle inflation and cluster techniques of classification using nearest neighbors were used to explore the filters of the

Point Cloud Library, in order to create a real-time algorithm capable of processing a dynamic point cloud data set.

2.2 Data acquisition

The first and most important part of the obstacle detection algorithm is how to acquire the data to process it and get all the necessary information needed to correctly execute algorithm. In the Figure 2.1 it is possible to see a block diagram of all the steps necessary to implement the obstacle detection algorithm.

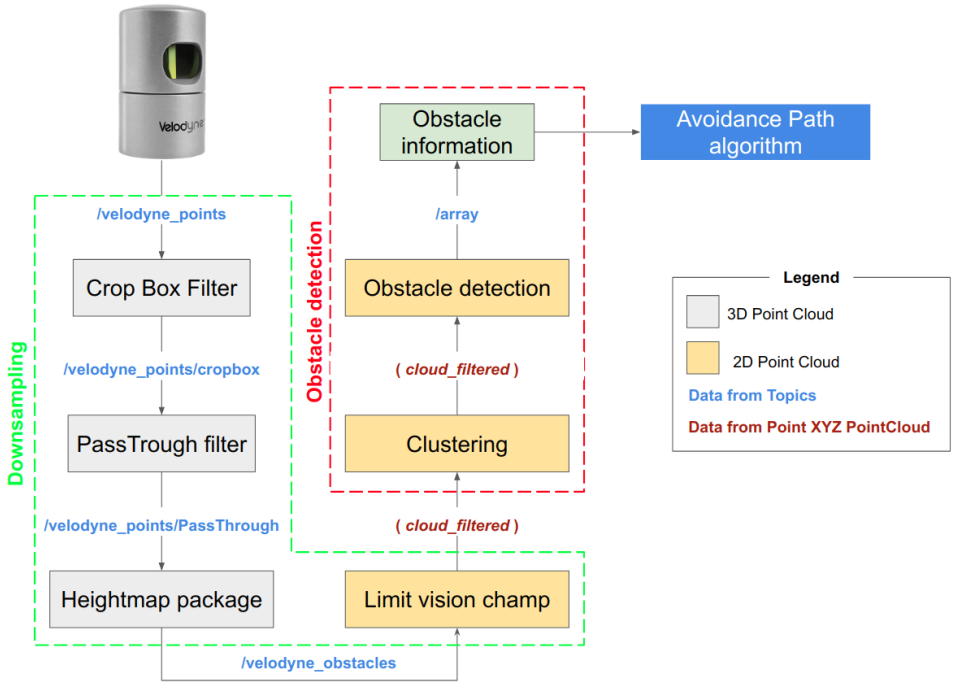


Figure 2.1 – Block diagram with the details of the Cloud processing stage

The point cloud used to recognize and the position of objects in the scene sends a message of the type *sensor_msgs/PointCloud2* from the LIDAR, which is later converted to a cloud of the type *pcl/PointCloud* to be used by PCL. The point cloud and filter acquisition steps are presented in the cloud pre-processing, which performs the point cloud preparation and reconstruction, while retaining the relevant feature information.

It is also important to note that all data obtained by LIDAR determines the position of the captured points with reference to the LIDAR's position, which can be considered equivalent to the position of the Spido robot at any given time. However, the operation of the robot command law algorithm takes into account the coordinates considering the GPS reference. Therefore, all point cloud data sets received by LIDAR must go through a reference shifting process so that the coordinates of the point detected by the sensor can be used in a harmonized manner by the other algorithms of the system shown in Figure 1.1. For this, the received coordinates are transformed with the aid of a frame

transformation matrix that considers the current position of the robot and also its current orientation, performing a translation operation and a rotation operation.

Moreover, during the whole point cloud processing stage, some hypotheses were considered to allow the algorithm development and to do the data processing in a controlled and effective way for the proposed problem. The hypotheses considered at this stage of the work were the following:

- **Flat ground**: It was considered that the terrain on which the test platform would operate would be flat, without ramps or major irregularities on the test ground.
- **Static obstacles**: The obstacles studied in the development of this project are static obstacles with definite and invariant position. Thus, their position can be found using RTK-GPS and Point Cloud XYZ.
- **Field opened and clear**: The vehicle's operating terrain was considered to be a broad terrain with no elements that would interfere with data capture or wireless communication (such as GPS and embedded computers, for example).
- **Known depth of the obstacles**: For all obstacles it is assumed that their depth is known and deep enough that the obstacle could fit inside a sphere. This way, the hidden part of obstacle that the LIDAR can not view would not influence the algorithm's result.

2.3 Data downsampling

Working with a cloud point data set allows you to build a map rich in details out a captured three-dimensional scene. However, the amount of points collected is very high (in the order of millions of points) and because of this the calculation time of each data set becomes very high if the computer handling the data is not powerful enough, such as is the case of the embedded computer used in this project. Therefore, to be able to extract only the essential information from the scene map, the pre-processing and processing steps must be performed to execute the stage called data downsampling, that is, the process used to reduce the size of a sample to store only relevant information to the desired purpose.

Thus, the downsampling process consists in the preparation and reorganization of the point cloud, removing the nonessential points and reducing the amount of points from the data-set.

In this project, filters from the PCL library were used to reduce the size of the data set obtained with the LIDAR while keeping only the characteristic information relevant for obstacle detection, as it will be shown below.

2.3.1 CropBox Filter

As described later in the Chapter 2.4, points close to the vehicle are considered as obstacles by the developed algorithm and this includes the detected points that represent the body of the Spido. Therefore, the CropBox filter was used considering the dimensions of the robot as shown in the Code 2.1.

The CropBox filter allows the point cloud filtering, using as reference a 3D box with defined dimensions. If the "negative" parameter is true, the filter eliminates the points inside the defined box; If this parameter is false, the filter eliminates the points outside the box.

Code 2.1 – CropBox filter paramters

```
<node   pkg="nodelet" type="nodelet" name="pcl_manager_crop" args="manager"
       output="screen" />
<node  pkg="nodelet" type="nodelet" name="cropbox"
       args="load pcl/CropBox pcl_manager_crop" output="screen">
    <remap from="~input" to="/velodyne_points" />
    <remap from="~output" to="/velodyne_points/cropbox" />
    <rosparam>
         #negative = true: no points in the box
         #negative = false: no points outside the box
         negative: true
         min_x: -1.5
         max_x: 1.5
         min_y: -1.5
         max_y: 1.5
         min_z: -3
         max_z: 3
    </rosparam>
</node>
```

2.3.2 PassThrough Filter

This filter is used to remove any point cloud having coordinates out of the bound set in the filter within a certain range in the direction of a given axis. In this case, this filter was used with the parameters shown in the Code 2.2 to eliminate the points that represent the ground - since one of the assumptions in the development of this project is that the soil on which the Spido will move is flat - and points that are slightly higher than the height of the robot were also eliminated, as these points would not block the vehicle's path: the leaves of a tree for example may not be taken into account at the obstacle searching algorithm as long as they are at a height high enough that Spido can

pass underneath them.

Code 2.2 – Pass Through filter paramters

```
<node pkg="nodelet" type="nodelet" name="pcl_manager_ransac" args="manager"
    output="screen" />
<node pkg="nodelet" type="nodelet" name="voxel_grid_ransac" args="load
    pcl/PassThrough pcl_manager_ransac" output="screen">
    <remap from="~input" to="/velodyne_points/cropbox" />
    <remap from="~output" to="/velodyne_points/PassThrough" />
    <rosparam>
        filter_field_name: z
        filter_limit_min: -1.3
        filter_limit_max: 1.5
    </rosparam>
</node>
```

2.3.3 Voxel Grid Filter

The Voxel grid filter is used to decrease the resolution of the point cloud. This process is done with the aid of a three-dimensional cubic grid (voxels) over the point cloud, as shown in Figure 2.2, and the size of each voxel is determined with the filter's "leaf size" parameter. With this filter, all points that belong to the same voxel are approximated to the cube's centroid. This process helps to create a simplified representation of the model, and with the appropriate leaf size value it is possible to maintain the most important cloud characteristics. That is, the smaller the size of the defined voxel, the higher the resolution (and therefore the cost) to render the necessary scene representation [17]. And as mentioned, we seek a resolution low enough to have fewer points to process, with a good representation of the environment.

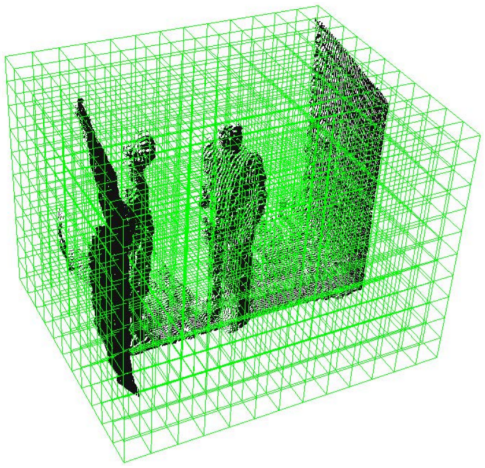


Figure 2.2 – Voxel grid filter visualization. Source: [18]

2.3.4 Heightmap

The heightmap [19] is a package that creates a nodelet that subscribes to the 3D data coming directly from the LIDAR point cloud and returns a 2D point cloud that detects objects higher than a certain height, publishing two point clouds: the *velodyne_obstacles*, which represents the obstacles found, and the *velodyne_clear*, which represents the points which are not detected as obstacles. The algorithm of this package also have implemented on it a stage of Voxel grid filtering to downsample the original point cloud.

Heightmap is a package used in the downsampling stage because with this package it is possible to take a 3D point cloud, process it and turn it into a simpler 2D point cloud with a lower resolution and with only important points. In addition to that, the heightmap package has a filtering step that uses the voxel grid filter mentioned above, to lower the original point cloud resolution.

In this case, the heightmap package was used to detect all the objects in the field that were higher than 5cm, to cluster them afterwards by the Euclidean Cluster Extraction filter and identify which objects could be considered as obstacles to the Spido Robot, bearing in mind the initial hypothesis of obstacle detection.

2.3.5 Limitation of the Vision field

Another step taken in LIDAR's point cloud downsampling process was to eliminate points that were out of a certain viewing angle. Thus, all points with coordinates such that $x < 0$ were disregarded, since the objects behind the car no longer represent a blocking element in the process of tracking the desired trajectory. The Figure 2.3 shows the limitation cited: in green we have the region in which the point cloud is considered, while in red we have the region in which the point cloud is disregarded.

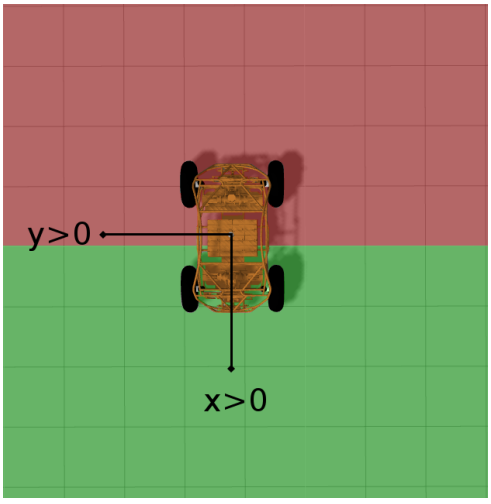


Figure 2.3 – Diagram illustrating the vehicle limited area of vision

2.3.6 Clustering

The clustering stage is an important part of this project as its main purpose is to separate the entire point cloud set into different smaller set of points that form up the representation of a specific elements, and in this way each of the identified objects is analyzed individually and separately by the obstacle detection algorithm. To perform the clustering stage, the Euclidean Cluster Extraction filter from the PCL library was used as shown in Figure 2.4, having as input the point cloud already treated using the heightmap package, that means, with the cloud being analyzed in a 2D perspective.

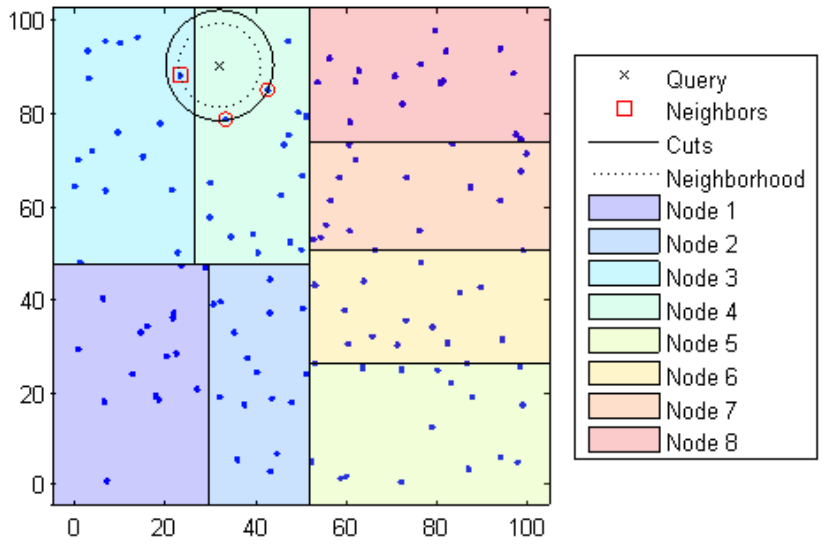


Figure 2.4 – Diagram illustrating the clustering method. Source: [20]

Euclidean Cluster Extraction uses a threshold parameter to the nearest neighbor of each point of the cloud data set to decide to what cluster a point would belong to. This search for the nearest neighbor is done using the Kd-tree method.

Kd-tree is a searching technique, and is a solution based on spatial decomposition and partitioning of the original point cloud into smaller parts to speed up the overall processing time. With the Kd-tree method, the point cloud space is divided into two parts from its midpoint and after that, each half is split in two and so on.

The Algorithm 1 shows the steps taken by PCL filter shown in the Code 2.3, to be clustered the point cloud.

Initially, a Kd-Tree object is created to represent the search method , together with a Point Indices array that contains the indexes of each detected cluster saved in the cluster_indices vector.

After this, a EuclideanClusterExtraction object with points of type PointXYZ is created. The parameters to the cluster processes are set choosing the tolerance for each cluster (SetClusterTolerance) and the maximum and minimum number of points inside a cluster (SetMaxClusterSize, SetMinClusterSize).

Algorithm 1 Clustering algorithm. Source: [21]

$P \leftarrow$ Point cloud resulting from the Kd-Tree searching method
$p_i \leftarrow$ Points of the cloud P
$C \leftarrow$ Empty list of clusters
$Q \leftarrow$ Queue of points to be checked
$d_{th} \leftarrow$ Cluster size

for every $p_i \in P$ **do**
  - Add p_i to the current queue Q
  for every $p_i \in Q$ **do**
    - Search for the set P_i^k of point neighbors of p_i in the threshold $r < d_{th}$
    - For every neighbor $p_i^k \in P_i^k$, check if the point has already been processed,and if not add it to Q;
  end for
  - When the list of all points in Q has been processed, add Q to the list of clusters C,and reset Q to an empty list
end for
- The algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters C

Finally, the Kd-tree search method is applied to the cloud and the indexes for each cluster identified are saved to the cluster_indices object.

Code 2.3 – Kd-tree filter

```
// Creating the KdTree object for the search method of the extraction
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new
    pcl::search::KdTree<pcl::PointXYZ>);
tree->setInputCloud((*cloud_filtered).makeShared());
std::vector<pcl::PointIndices> cluster_indices;

//Cluster point cloud based on the kd-tree algorithm, and returns a vector of
    indices
pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;

//kd-tree Library parameters
ec.setClusterTolerance (0.4);
ec.setMinClusterSize (2);
ec.setMaxClusterSize (25000);
ec.setSearchMethod (tree);
ec.setInputCloud ((*cloud_filtered).makeShared());
ec.extract (cluster_indices);
```

2.4 Obstacle detection

The development stage of the obstacle detection algorithm is separated into 3 steps: Data acquisition, downsampling and finally clustering and obstacle classification.

At the acquisition stage, data from the LIDAR is sent to the NUC embedded computer via the topic */velodyne_points* to be re-sampled by the PCL library filters, and finally, the reduced point cloud goes through a clustering and sorting algorithm to identify which objects could be considered as obstacles and which ones are not obstacles. The result of the obstacle detection algorithm is a vector containing information about the detected obstacle (position in the map and safety distance), and this information is published to the network through the */array* topic

To perform the obstacle detection part, a technique called "obstacle inflation" was used to facilitate the development of the avoidance path generation algorithm. In this technique, for every object detected by the LIDAR, the position of the barycenter of the point cloud related to this object is obtained. After that, the object is inflated, that is, its location in space is no longer considered only as the coordinates of its barycenter but now the obstacle is considered as a circle of radius r and centered in the barycenter of the point cloud of the obstacle.

Thus, the radius r is chosen so that the dimensions of the vehicle can be considered negligible by the avoidance path algorithm, and so that Spido can be considered as a point in the configuration space. This circle surrounding the object is named in this project as *danger_circle* and is defined as being $9m$ larger than the circle surrounding all points of a detected object. The value of $9m$ was chosen after doing empirical tests with the robot. An object is classified as an "obstacle" if the coordinates (x, y) of the robot are located within the objects *danger_circle*. If the (x, y) of the robot are located outside the *danger_circle*, the object is not considered as an obstacle.

In addition to the *danger_circle* that aids the obstacle detection determine whether an object is an obstacle or not, the algorithm uses a second circle with a radius smaller than the *danger_circle* radius, which is used to aid the avoidance path creation algorithm. This circle is defined as *bezier_circle* and has a radius of size $3m$ larger than the circle surrounding all points of an object detected and considered as an obstacle. As for the $9m$ value mentioned before, the value of $3m$ was chosen after doing empirical tests with the robot.

The Algorithm 2 shows in more detail all the steps taken to perform the obstacle detection stage, and the Figure 2.5 illustrates the idea mentioned above.

Algorithm 2 Obstacle detection algorithm

$C_k \leftarrow$ Clusters found with the Algorithm 1
$p_i^k \leftarrow$ Points of the cluster clouds C_k
array $\leftarrow$ Publisher node for the obstacles information
min_dist $\leftarrow$ Vector to store the distances of p_i^k
raio $\leftarrow$ Radius of the circle that contains all p_i^k of a obstacle cloud
raio_danger $\leftarrow$ Circle that gives a safety distance to detect an obstacle

for every C_k **do**
  for all $p_i^k \in C_k$ **do**
    - Find $x\_max, x\_min, y\_max, y\_min$
    if $(x\_max - x\_min)/2 > (y\_max - y\_min)/2$ **then**
      raio $= (x\_max - x\_min)/2$
    else
      raio $= (y\_max - y\_min)/2$
    end if
    $raio = raio + 0.3.raio$ *The 0.3 is there to increase the radius, to ensure that the points of the cluster will be inside of it
    $raio\_danger = 9 + raio$
    Loop when obstacle is found
    if $\sqrt{media\_x^2 + media\_y^2} - raio\_danger < 0.3$ **then**
      - array $\leftarrow$ flag indicating that an obstacle was found
      - $array \leftarrow media\_x$
      - $array \leftarrow media\_y$
      - $array \leftarrow raio\_danger$
      - $array \leftarrow raio$
      - publish the information of *array*
    end if
  end for
end for

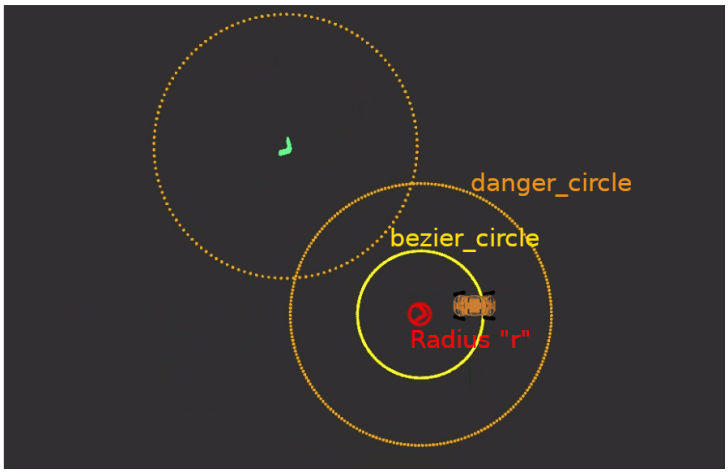


Figure 2.5 – Illustration of the obstacle detection circles

Chapter 3

Simulation and experimental validation of the 3D point cloud processing

3.1 Validation in virtual simulation

This section will present the results obtained from the simulation tests of the obstacle detection algorithm. The Figure 3.1 shows the settings used to test the algorithm: Three boxes were placed on the gazebo's grid distant from each other, and at different distances from the Spido.

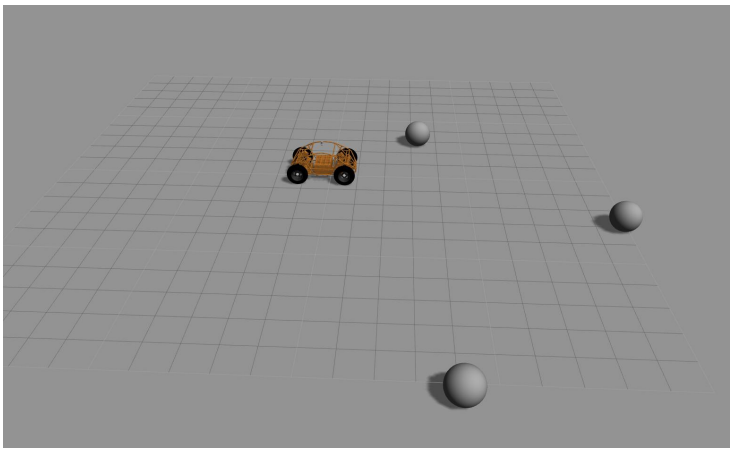


Figure 3.1 – Simulation settings for the obstacle detection part

3.1.1 LIDAR data cloud point

The Figure 3.2 shows the point cloud captured directly by the LIDAR, without passing through any data processing stage for the simulation tests.

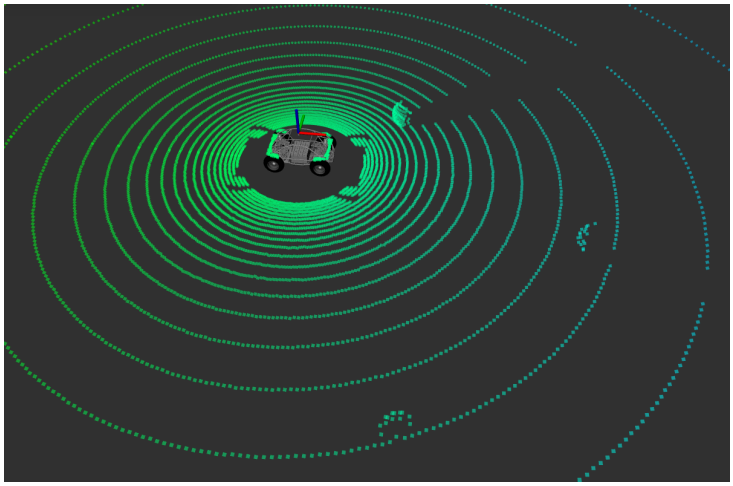


Figure 3.2 – Simulation test for the point cloud from the LIDAR without processing

In this image it is possible to visualize all the elements of the gazebo's world: ground, Spido's body and spheres placed around the car.

3.1.2 Pass Through and CropBox Filters

The Figure 3.3 shows the point cloud after being downsampled by the PassThrough and CropBox filters for the simulation tests. The colors in the image represent the distance of the points compared to the position of the robot.

Figure 3.3 – Simulation test for the cloud downsampled with PassThrough and CropBox filters

In this image is possible to visualize the filter's results: After eliminating the ground points and the Spido body's points, we can just see a 3D cloud representing only the

spheres placed on the simulation.

3.1.3 Heightmap

The Figure 3.4 shows the downsampled point cloud that was processed by the heightmap package for the simulation tests.

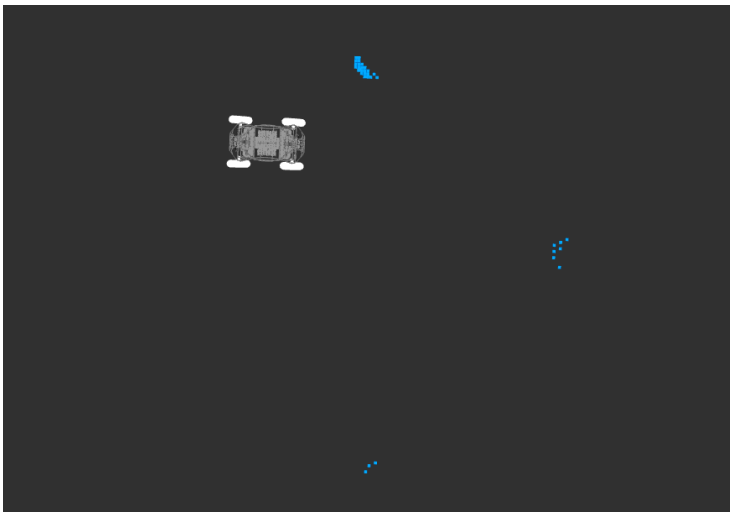


Figure 3.4 – Simulation test for the cloud after the application of the heightmap package

In this image it is possible to see that from now on all the 3D point cloud is transformed into a 2D point cloud, and the only objects detected are the spheres, which are higher than 5cm from the ground.

3.1.4 Obstacle detection

The Figure 3.5 shows the the final stage of the obstacle detection algorithm for the simulation tests.

In this image, it is possible to see the obstacle detection algorithm working: If the Spido is inside one of the danger circles, the object is shown in green, otherwise it is shown in red. Two spheres are far away from the Spido and thus are shown in green, while there is one sphere that is close to the Spido and therefore is shown in red.

3.2 Experimental validation

This section will present the results obtained for the experimental test of the obstacle detection algorithm. Figure 3.6 shows the settings used to test the algorithm: Two boxes were placed on the test terrain and the pointclouds captured by the Spido were saved using a remote computer that was communicating with the NUC via ssh protocol.

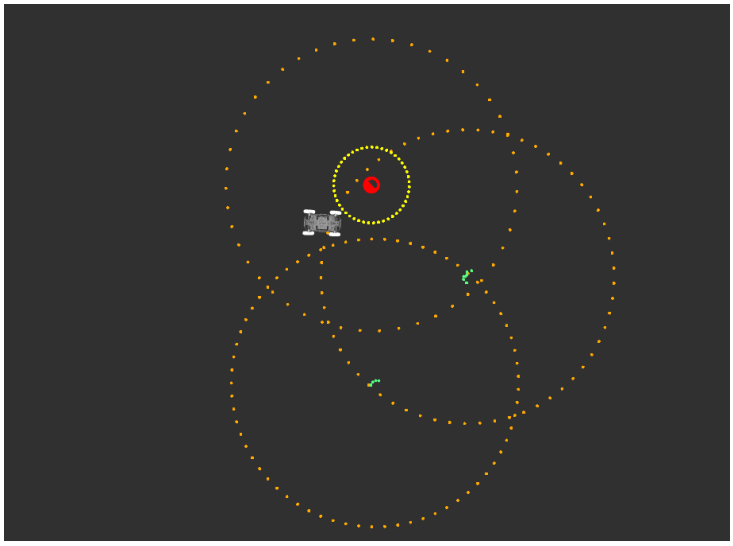


Figure 3.5 – Simulation test for the cloud after clustering and obstacle detection algorithm

Figure 3.6 – Experimental settings for the obstacle detection part

3.2.1 LIDAR data cloud point

Figure 3.7 shows the point cloud captured directly by the LIDAR, without passing through any data processing stage for the experimental tests.

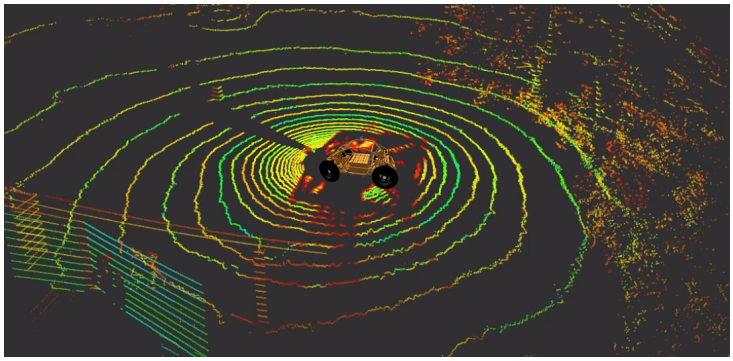


Figure 3.7 – Experimental test for the point cloud from the LIDAR without processing

In this image, it is possible to visualize all the elements of the terrain: Soil, trees, one of the boxes, and walls of the laboratory.

3.2.2 Pass Through and CropBox Filters

The Figure 3.8 shows the point cloud after being downsampled by the PassThrough and CropBox filters for the experimental tests.

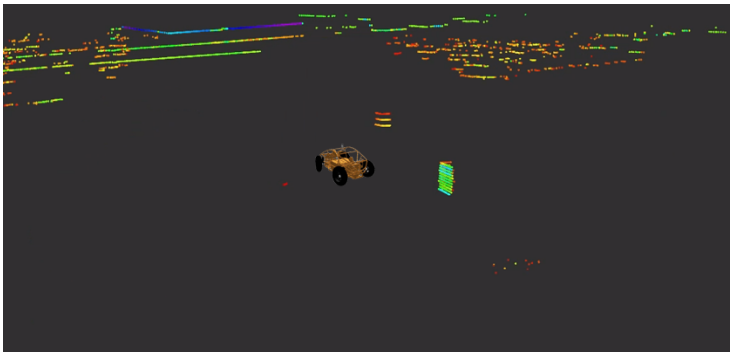


Figure 3.8 – Experimental test for the cloud downsampled with PassThrough and CropBox filters

In this stage, we can validate the two filters comparing with Figure 3.8, since here the soil is no more visible, while it is still possible to see one of the boxes on the field. In the upper part of the image it shows a part of the soil, because the terrain is not completely flat and, thus, there is a little elevation compared to the actual position of the Spido.

3.2.3 Heightmap

The Figure 3.9 shows the raw point cloud coming from the LIDAR (in white) and the downsampled point cloud that was processed by the heightmap package (in red) for the experimental tests.

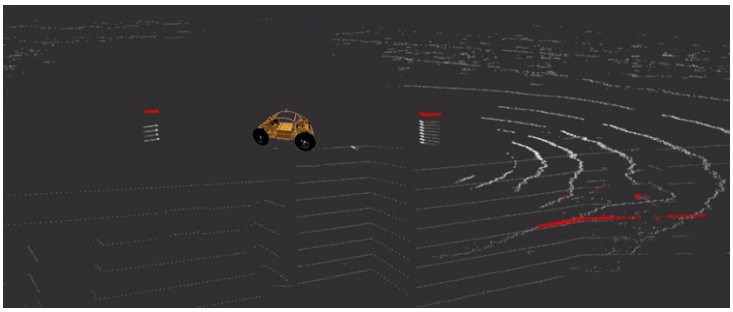


Figure 3.9 – Experimental test for the cloud after the application of the heightmap package (in red)

In this image is possible to see that from now on all the 3D point cloud is transformed into a 2D point cloud, and the only objects detected are the ones having a height above 5cm from the ground, which means, the boxes and also the walls of the laboratory.

3.2.4 Obstacle detection

Figure 3.10 shows the final stage of the obstacle detection algorithm for the experimental tests.

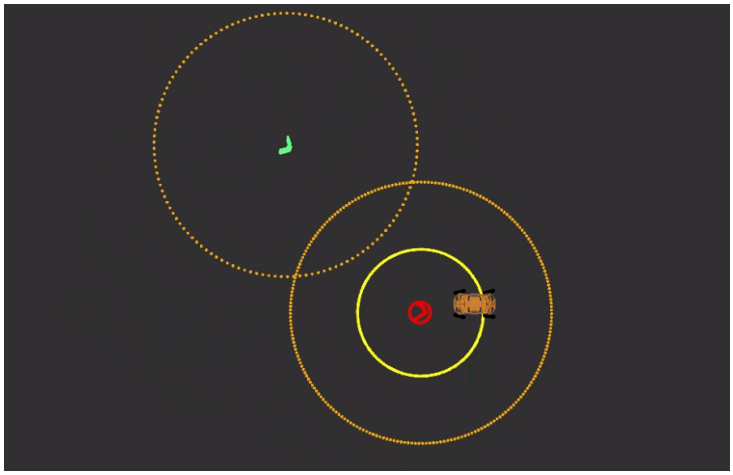


Figure 3.10 – Experimental test for the cloud after clustering and obstacle detection algorithm

In this image, it is possible to see the obstacle detection algorithm working: One of the boxes is far away from the Spido and also the Spido is not inside this box's *danger_circle*. Thus, it is not considered an obstacle and it is shown in the image with the green color. On the other hand, the other box is closer to the Spido and it is inside the box's *danger_circle*. For this, these boxes are shown in red. The yellow circle represents the circle that will be used by the algorithm that generates the avoidance path.

3.3 Conclusion and perspectives

Comparing the two sets of results, from simulation tests and experimental tests, it was possible to validate the algorithm created. One thing that can be explored with more details in future works is the shape chosen to inflate the object: instead of using a circle, it might be better to use an ellipse to wrap the objects wider like walls for example. Besides that, the algorithm showed satisfactory results.

Chapter 4

Search for avoidance trajectories and optimization

4.1 Introduction

Trajectory planning for obstacle avoidance is a crucial element in the studies for autonomous mobile vehicles and the choice of the path to be followed should be made selectively according to the robot constraints. Usually, the desired curve to be used must be smooth and continuous, and it has to have adequate curvature to the limits of the robot's steering angles and maximum speed, not presenting sharp curves in order to allow the robot to follow it without stopping. Otherwise, the robot may either inefficiently follow the suggested path or may even become damaged when attempting to perform an improper movement.

This project aims to explore Bézier curves for the generation of real-time avoidance paths and to verify if their use allows a robot to follow a local obstacle avoidance path. Bézier curves are practical to work since it is easy to define its ending and starting points for local trajectories, only having the inconvenience of being difficult to choose certain parameters.

This chapter will present an overview of the definition of this curve, as well as the algorithm used to generate avoidance paths using this kind of curve.

4.2 Global and local paths

The global path is the path given to the robot considering the starting position of the robot and the final destination. It can be seen as a global picture of the path which the robot has to follow. On the other hand, the local path is a path done by the robot considering only a small interval inside the global path.

For this work, the global path is set as the main path for the robot to follow, imagining that the path would be free of any kind of object that could potentially block the way of the robot. The local path refers to the local avoidance path generated to the robot in order to avoid an obstacle that is blocking its way.

In order to follow a global and local path, a command algorithm is run in one of the embedded computers of the platform, and this algorithm ensures that the Spido will follow the proposed path.

Command algorithm

This part is not included in our work, and it is already synthesized and validated through advanced numerical simulation under ROS-Gazebo as well as real experiments using Spido Robot (For more details, see [22]). In general, it uses a Model Predictive Control (MPC) approach to develop the cost function based on the dynamic model of the vehicle. This cost function must be minimized taken into account some intrinsic and physical constraints of the vehicle. These later are steering angles limits and tire pseudo-slippage (sliding) area bounds that should be fulfilled at each time step. To solve this problem, [22] used quadratic programming (QP) solvers. The solver used is CVXOPT to get at each time the optimal and acquired steering angles to ensure path tracking tasks.

4.3 Bézier Curve

Developed by Paul de Casteljau and Pierre Etienne Bézier, Bézier curves are parametric curves used in different applications to build smooth continuous curves, and have their shape defined by a finite number of control points. [23], and are used in different applications such as in graphic design software and CAD projects. It is an n-order class of polynomial curves expressed by the linear interpolation between the vertex of a control polygon, and its construction is based on the use of a parametric curve created by combining Bernstein polynomials with the Casteljau algorithm, which computes the Bernstein coefficients of the curve. Furthermore, an n-degree Bézier curve can be simplified by using a piece-wise cubic Bézier curves, as will be shown.

4.3.1 Parametric Curves

Parametric Curves are a very convenient way of describing the path of a particle in the space because for each instant of time t considered, we have the respective coordinates x and y that describe the position of a point P, that means, the t parameter is within the interval $[0, 1]$ and $(x(t), y(t))$ gives us the position of the point at t, which, in this case, moves in the R^2 plane. We call then $\gamma(t)$ the curve that describes the trajectory done by point P.

For this project, the concept of parametric curves was used to generate Bézier curves for the deviation path that should be performed by the robot, and it was also used to work with the development optimization algorithm for the Bézier curve parameters.

4.3.2 Bernstein polynomials

Bernstein polynomials of degree n are defined as shown in the equation (1):

$$B_{i,t}(t) = \binom{n}{i} t^i (1-t)^{n-i} P_i, \qquad i = 0, 1, ..., n, \qquad t \in [0, 1] \tag{1}$$

For a given n-degree Bernstein polynomial, there are n+1 polynomials that make up a base for the space of polynomials of degree less than or equal to n. Bernstein polynomials are a good form of interpolation compared to other existing forms because its linear combination results in a smoother interpolating curve and also offers a more uniform approximation [24].

Figure 4.1 shows an example of the Bernstein polynomials of degree 3.

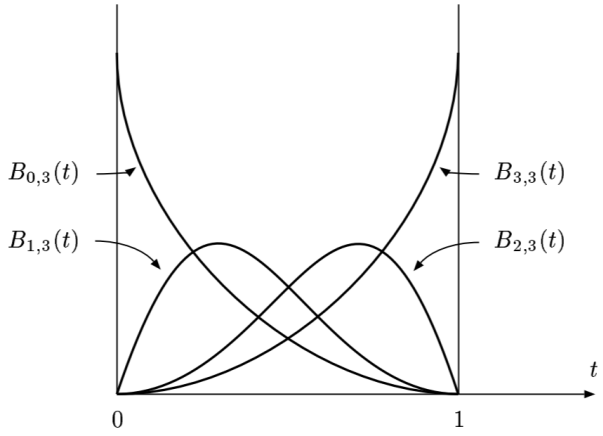


Figure 4.1 – Bernstein polynomials of degree 3. Source: [25]

4.3.3 Construction of a bezier curve

Bézier curves are polynomial parametric curves in the plane or space and are defined according to the equation (2), where the P_i are called control points, and the curve is

defined from the base formed by the Bernstein polynomials (1)

$$B(t) = \sum_{i=0}^{n} P_i B_{i,n}(t), \qquad t \in [0,1] \tag{2}$$

Among the properties of this family of parametric curves, we can highlight 3 that are considered important for the development of this work, namely:

1. The curve approximates a given polygonal curve, called the control polygon.
2. The curve tangents the first and last segments of the control polygon to the first and last vertices of the polygon.
3. The curve is contained within the convex hull of the control polygon.

In the equation (2), the P_i vertices can be seen as weights for each degree of the Bernstein polynomial, and the curve $B(t)$ resultant can be seen as a weighted combination of the control points of the curve as a function of the t parameter.

Cubic Bézier curve:

The cubic Bézier curves is a case of the Bézier curves made up of Bernstein polynomials of degree 3 and, thus, 4 control points. This is a special case for this family of curves for it can be combined in order to represent and generalize Bézier curves of an order superior to 3. Due to this special characteristic, this is the curve that will be used in this project to generate the avoidance path. Besides that, the four control points of the cubic Bézier allows the determination of the start and end direction of the curve, which is highly important to assure the continuity of the reference path at the joint points connecting the global reference path and the local reference path. The equation (4) shows the general form of the Cubic Bézier curve ($P(t)$), knowing that each vertex of the control polygon is a weight for the Bernstein polynomial of order 3 ($B(t)$) from equation (3).

$$\begin{aligned} B_{0,3}(t) &= (1-t)^3 \\ B_{1,3}(t) &= 3t(1-t)^2 \\ B_{2,3}(t) &= 3t^2(1-t) \\ B_{3,3}(t) &= t^3 \end{aligned} \tag{3}$$

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \tag{4}$$

Looking at equation (4) and the Figure 4.1, it is possible to verify that the $P(t)$ curve is a combination of the control point P_0, P_1, P_2 and P_3, and that as t goes from 0 to 1, the degree of influence of each control point in the $B(t)$ curve starts to change, that means, at the beginning P_0 has more influence into the shape of $B(t)$, then P_1, then P_2 and finally P_3.

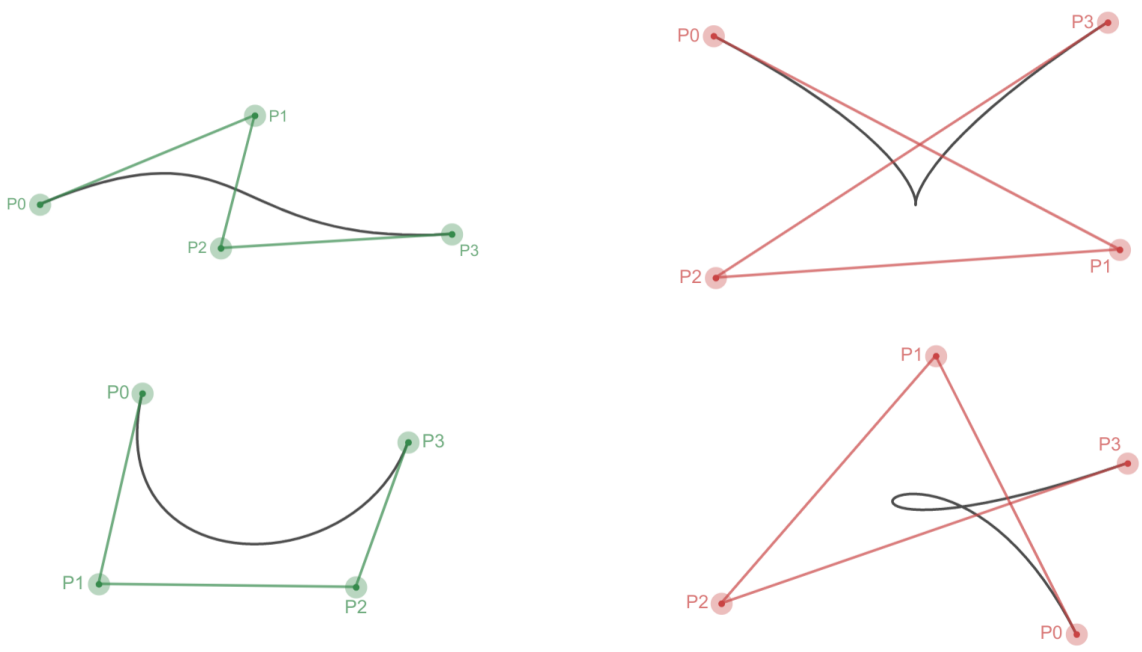
Composite Bézier curve

As stated previously, an n-order Bézier curve can be simplified by using multiple cubic Bézier curves. Bearing this in mind, an important concept needed to know for this work was the concept of continuity for curves. In order to assure a smooth and continuous curve, it has to present a certain degree of continuity.

- C_0 **continuity**: Two lines connect with each other using a common point, which means, there is no jump between the two lines.
- C_1 **continuity**: The direction of the tangent vector connecting the two curves is the same.
- C_2 **continuity**: The curvature connecting two curves are the same

Usually, the ending and starting points of a Bézier curve is easy to do, since the position of these points is defined based on the need of the application. In this case, the ending and starting point for a Bézier curve is defined as the moment when the Spido detects an object and the ending point is a point where the Spido is no closer to the object considered as an obstacle before.

However, the choice of the control points can be difficult to do, since the control points are responsible to ensure that a Bézier curve will be continuous. The Figure 4.2(a) shows a cubic Bézier where the control points were chosen in a way that the resulting curve was a C1 curve, whereas the Figure 4.2(b) shows a cubic Bézier where the control points were chosen in a way that the resulting curve is continuous, but not C1.



((a)) Good control points for a C1 Cubic Bézier curve

((b)) Bad control points for a C1 Cubic Bézier curve

Figure 4.2 – Illustration of the importance to choose well the control points of a C1 Bézier curve

4.4 Bézier curve for local trajectory planning

In the stage of the search for avoidance trajectories and optimization, the aim is to use a composite Bézier curve to generate a local path to avoid obstacles while taking into account some important constraints to ensure that the avoidance path created is an ideal path to follow. For this, two cubic Bézier curves connected by a common point will be used. The first curve is relative to avoiding obstacles: once the Spido detects an obstacle, it will follow this first curve to distance itself from the obstacle in order to avoid hitting it. The second curve is relative to the return of Spido to the initially set global trajectory. However, an important and difficult issue to adjust when using composite Bézier curves is the choice of good control points for each curve. As seen earlier, given a Bézier curve composed of several cubic Bézier curves, the choice of their control points will be responsible for determining whether the connection between the curves will have a good continuity or not. Therefore, in a second moment, an algorithm was developed to optimize the choice of the control points for the two cubic Bézier curves for the avoiding curve. Thus, the steps taken to use a Bézier curve for local trajectory planning takes into account four major important criteria:

1. **Minimum curvature:** The new path has to have the less curvature as possible, to avoid giving to the Spido robot a path where it would have to turn in a sharp angle.
2. **Closer to the reference path:** It is important to avoid an obstacle, but it is also important to ensure that the Spido will not be taken far away from the original path.
3. **No obstacle:** Last and most important, the new path generated has to give the Spido the possibility of navigating without hitting any obstacles.
4. **Longitudinal velocity V_x constant:** In this work the longitudinal dynamics are neglected

Below it will be presented in more detail the algorithm developed to construct an avoidance curve for the robot, as well as the algorithm used to find the optimal curve for a given obstacle, bearing those four main criteria in mind.

4.4.1 Avoidance path algorithm

The injection of the desired path into the command algorithm is done using a ".txt" file in which each line has three columns of information, being the second and the third one the values of x and y respectively that gives the Spido the coordinates to be followed through the experiment. The first column gives information about the angle of orientation ψ of the robot, and it is used in the control part.

The Algorithm 3 proposed to create the avoiding path was done based on the information of this ".txt" file, and at the end of the algorithm a new ".txt" file is generated

and the reference path to the robot is changed in case an obstacle is detected, as shown previously in the Figure 1.2.

Algorithm 3 Avoidance path algorithm

ref ← reference path
index ← line index (of *ref*) from the control algorithm
P ← Point with coordinates (x, y)
obstacle_circle ← circle containing the obstacle
danger_circle ← safety circle for the obstacle detection
bezier_circle ← circle used to create the bezier curves
flag_new_path ← 1
$k \leftarrow 0$
Find initial and final point for the avoidance path
if flag_new_path = 1 **then**
  flag_new_path ← 0
  k ← index
  $P_0 \leftarrow P_k$
  for each $P_k \in$ *ref* **do**
    if P_k is inside the *danger_circle* **then**
      k ++
    end if
  end for
  $j \leftarrow k$
  $P_7 \leftarrow P_j$
  Find the middle point for the avoidance path
  - *bezier_circle* ← *obstacle_circle* + 3
  - r ← line connecting P_0 and P_7
  - s ← line perpendicular to the line r, passing by the barycenter of the obstacle
  - Find P_3 in $min\_dist\{P_{di}, P_M\}$, with $di = \{1, 2\}$
  - Find the intersection of the line s with the bezier circle, returning P_{d1} and P_{d2}
  - Find the coordinate $P_M \in$ *ref*, the closest to the intersection of r with s
  Find Auxiliary points
  - Find the line t_1 parallel to r, passing by P_3
  $P_{3_{aux}}$ ← projection of P_7 on the line t_1
  $P_{4_{aux}}$ ← projection of P_0 on the line t_1
  Find Control Points
  P_1, P_2 ← run the **Algorithm 4** for the first Bézier curve
  P_5, P_6 ← run the **Algorithm 4** for the second Bézier curve
end if
Generate the Bézier curve
Create a new reference path changing the path between P_0 and P_7 by the Bézier curve created

The avoidance path is created by using two cubic Bézier curves:

- **First Bézier curve**

  This first curve will be used to leave the global path and start the local avoidance

path. In this case, we name its control points as P_0, P_1, P_2 and P_3 and they are defined as follows:

- The point P_0 is defined as the point (x, y) where the robot detected the obstacle
- The points P_1 and P_2 will be found with an optimization algorithm, as it will be described below.
- The point P_3 is located within the global path, with the distance $d(P_3, P_0) = d(P_3, P_7)$.

- **Second Bézier curve**
  The second curve will be used to leave the local avoidance path and go back to the global path. In this case, we name its control points as P_4, P_5, P_6 and P_7 and they will be defined as follows:
  - The point P_4 is located at the same position as the point P_3.
  - The points P_5 and P_6 will be found with an optimization algorithm, as it will be described below.
  - The point P_7 is located within the global path in a position outside the *danger_circle*.

Thus, this algorithm receives a message from a topic called /array from the obstacle detection stage, and in this topic, there is a flag that indicates whether an obstacle was detected or not. If an obstacle was detected and the Spido's position passes the safety circle area, the index value of the ".txt" file is saved in a variable and that position is taken as the starting point of the Algorithm 3. Once the algorithm starts, it finds the next index point that gives a point outside the safety circle area and applies some geometric steps to find the middle point that will connect the two Cubic Bézier curves. After finding the fix points, that means, the starting point, the ending point, and the middle point, the control points are determined using the Algorithm 4. The Figure 4.3 illustrates the main steps are done by the avoidance path algorithm in an S-shaped path: First, the starting and ending points of the local path are found (P_0 and P_7); then the intersection between the global path and the line perpendicular to the line crossing the starting and ending points is found; After it, the two possibilities for middle control point for the avoidance path are found, and the one closer to the reference path is chosen (P_3 and, consequently, P_4); Finally, the control points of the cubic Bézier curves are computed and the local path to avoid the obstacle is generated (P_1 and P_2 for the first curve, P_5 and P_6 for the second curve).

Among the 3 criteria cited at the beginning of this section, it is possible to verify that the "minimum curvature" criterion is met through the control points optimization algorithm, while the "closer to the reference path" criterion is met through the choice of

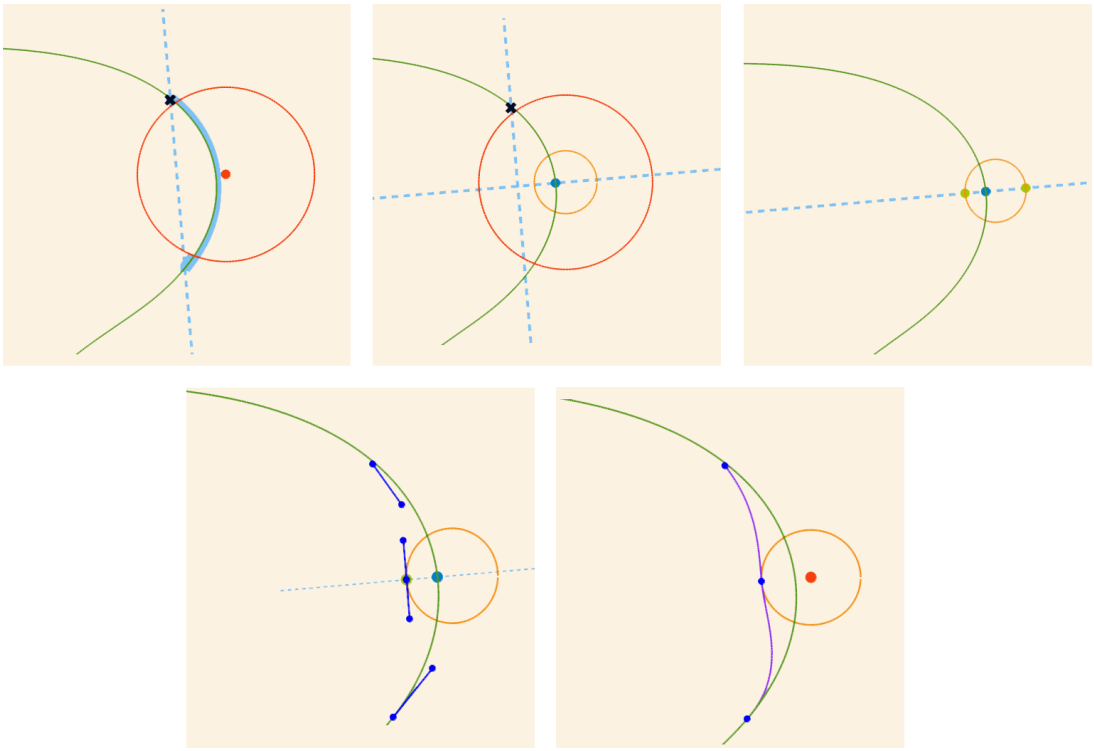


Figure 4.3 – Visualization of the steps for the Algorithm 3

the middle point. Finally, the "no obstacle" criterion is met by choosing the starting and ending points outside the safety circle.

4.4.2 Avoidance Path Optimization

As seen previously, the main challenge in using Bézier curves for the avoidance path generation stands on the choice of the unknown control points, since it is due the positioning of these points that it is possible to ensure that the generated path will present the continuity restrictions desired so that Spido can track it with stability.

To solve this problem, it is proposed the Algorithm 4 based on the principle of the choice of the clothoide tentacles used in [7], which generates several Bézier curves by combining the parameters P_1 and P_2, P_5 and P_6, and chooses the curve that has the lowest possible curvature, having as a restriction parameter the maximum curvature that can be performed by the robot, which takes into account its construction factors and hence its steering angles. In this case, the maximum curvature that can be performed by the Spido robot is $0.246m^{-1}$.

Algorithm 4 Control points optimization

$P_{0_{aux}}, P_0, P_{3_{aux}}, P_3 \leftarrow$ variables from the **Algorithm 3**
$P_{4_{aux}}, P_4, P_{7_{aux}}, P_7 \leftarrow$ variables from the **Algorithm 3**
Find first Bézier tangents
Find the line q connecting $P_{0_{aux}}$ and P_0
Find the line q_{per} perpendicular to the line r, passing by the point P_3
$P_{end1} \leftarrow$ intersection point of q with q_{per}

Find the line r connecting $P_{3_{aux}}$ and P_3
Find the line r_{per} perpendicular to the line s, passing by the point P_0
$P_{end2} \leftarrow$ intersection point of r with r_{per}
Find second Bézier tangents

Find the line s connecting $P_{7_{aux}}$ and P_7
Find the line s_{per} perpendicular to the line s, passing by the point P_4
$P_{end3} \leftarrow$ intersection point of s with s_{per}

$step \leftarrow 10$
$t_1 \leftarrow 0$
Finding P_1 and P_2
for i $<$ *step* **do**
    Vary P_1 through the vector P_0, P_{end1}, with $dist(P_0, P_1) = t_1$
    for j $<$ *step* **do**
        - Vary P_2 through the vector P_3, P_{end2}, with $dist(P_3, P_2) = t_2$
        - Compute the maximum curvature for the bézier curve using P_0, P_1, P_2 and P_3
        if maximum curvature < 0.246 **then**
            - ***bezier_candidates*** $\leftarrow$ P_0, P_1, P_2, P_3 and curvature value
        end if
        $t_2 \leftarrow t_2 + 1/step$
    end for
    $t_1 \leftarrow t_1 + 1/step$
    $t_2 \leftarrow 0$
end for
P_1, P_2 $\leftarrow$ values from ***bezier_candidates*** the set of parameters having the minimum curvature value
Finding P_5 and P_6
$P_5 \leftarrow$ Point in the vector P_2, P_3, with $dist(P_4, P_5) = dist(P_3, P_2)$
$t_3 \leftarrow 0$
for k $<$ *step* **do**
    - Vary P_6 through the vector P_7, P_{end3}, with $dist(P_7, P_3) = t_3$
    - Compute the maximum curvature for the bézier curve using P_4, P_5, P_6 and P_7
    if maximum curvature < 0.246 **then**
        - ***bezier_candidates*** $\leftarrow$ P_4, P_5, P_6, P_7 and curvature value
    end if
    $t_3 \leftarrow t_3 + 1/step$
end for
P_5, P_6 $\leftarrow$ values from ***bezier_candidates*** the set of parameters having the minimum curvature value

Return P_1, P_2, P_5 and P_6

Figure 4.4 shows the results of the optimization algorithm applied to just one cubic Bézier curve. In the image, it is possible to see that the red curve, chosen as the optimal curve by the algorithm, has a smooth curvature and also meets all the desired continuity criteria presented previously.

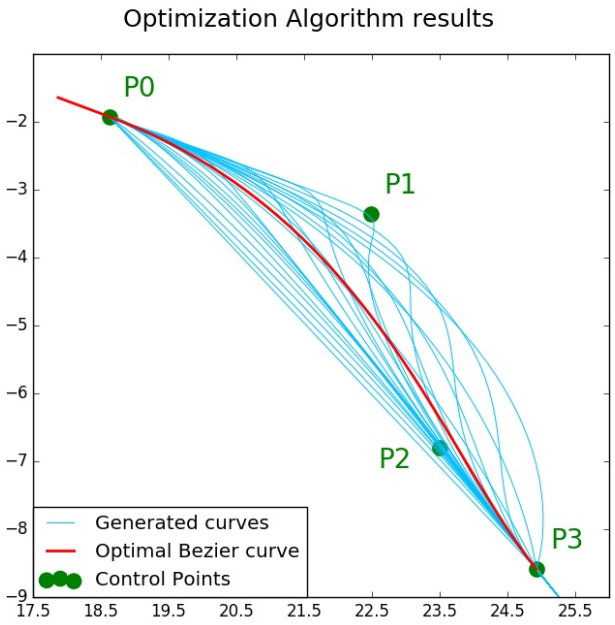


Figure 4.4 – Optimal Bézier Result

Chapter 5

Validation for the search for avoidance trajectories and optimization

5.1 Validation in virtual simulation

To validate the algorithm through virtual simulation, several tests were done considering different path tracking shapes, different obstacle positions, different speeds for the vehicle and also multiple objects.

For the experimental test part it was necessary to update some hardware of the test platform and also to adapt the simulation algorithm code to the appropriate parameters considering the actual sensors, and readjust the referential points to the rotation matrix. Unfortunately, due to technical issues during this system upgrade stage, we were unable to perform the desired experimental tests as there was not enough time to do so.

5.1.1 Rectilinear path

The figures 5.1, 5.2 and 5.3 below show, respectively, the results for the simulation tests in a rectilinear path, and the tracking error of those tests.

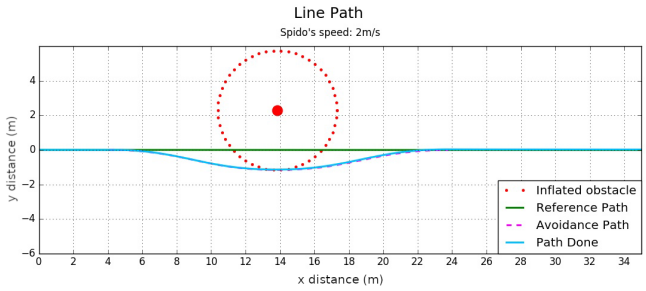


Figure 5.1 – Simulation test for the rectilinear path with a speed of 2m/s and its tracking error

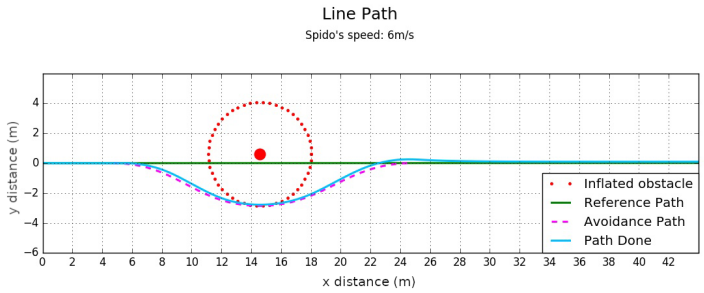


Figure 5.2 – Simulation test for the rectilinear path with a speed of 6m/s and its tracking error

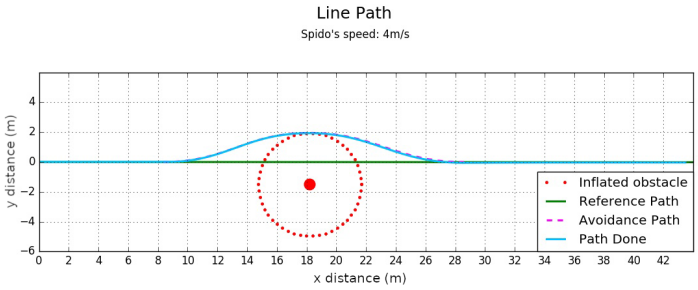


Figure 5.3 – Simulation test for the rectilinear path with a speed of 4m/s and its tracking error

5.1.2 S-shaped path

The figures 5.4, 5.5 and 5.6 below show, respectively, the results for the simulation tests in a S-shaped path, and the tracking error of those tests.

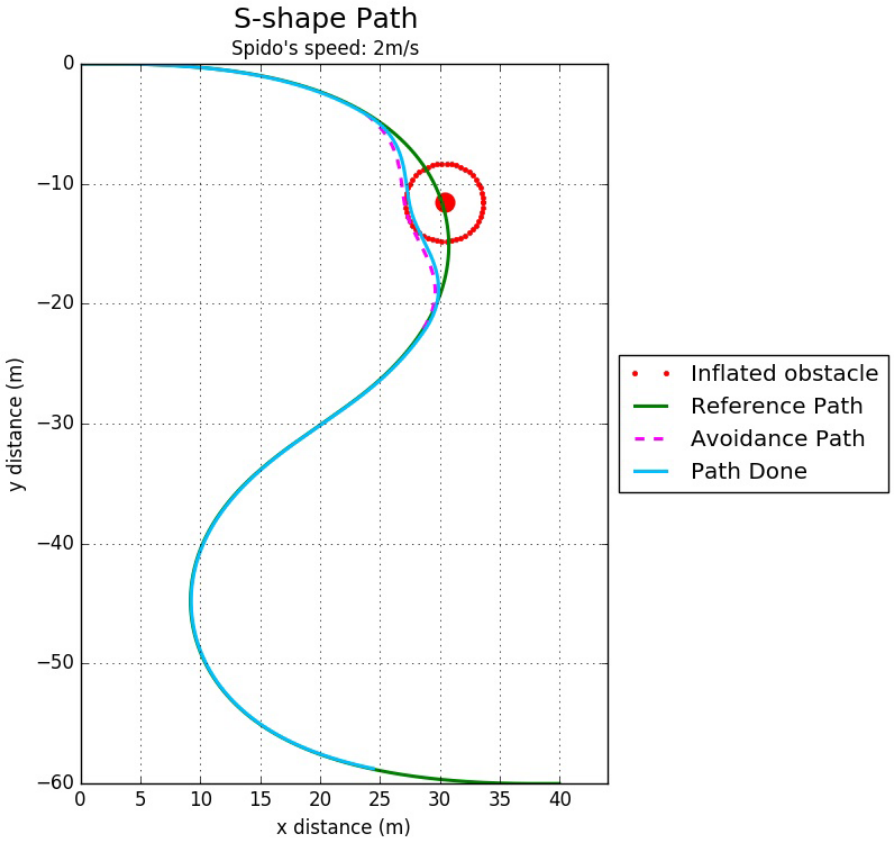


Figure 5.4 – Simulation test for the S-shaped path with a speed of 2m/s and its tracking error

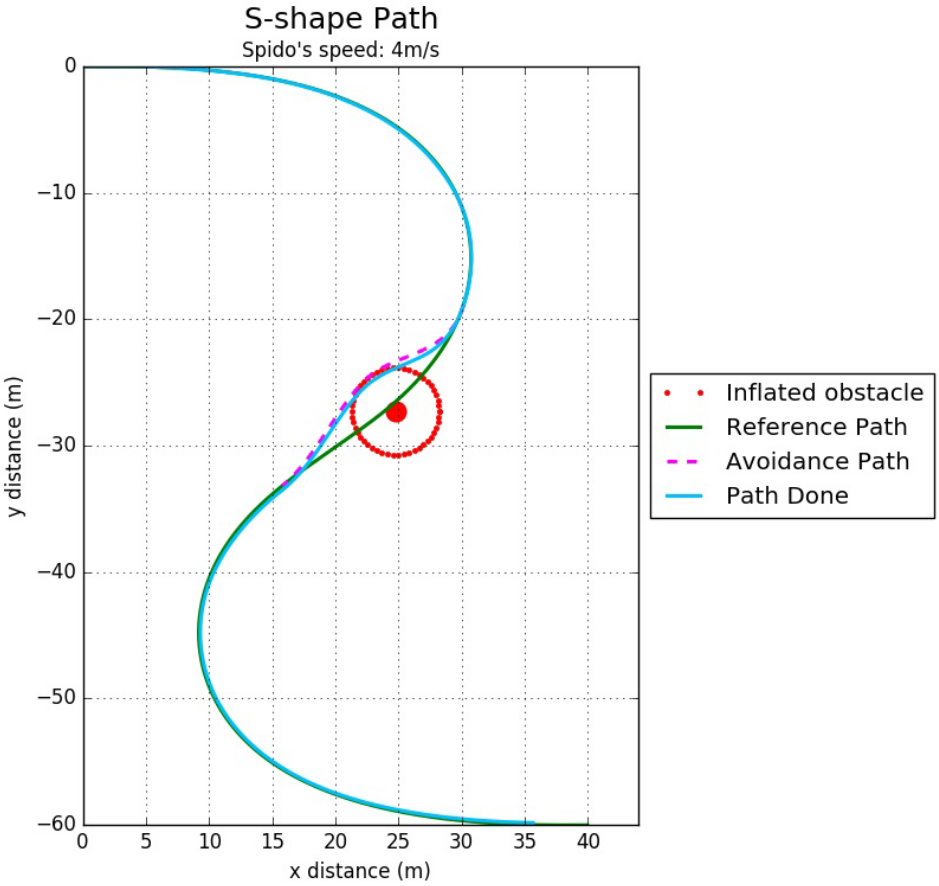


Figure 5.5 – Simulation test for the S-shaped path with a speed of 4m/s and its tracking error

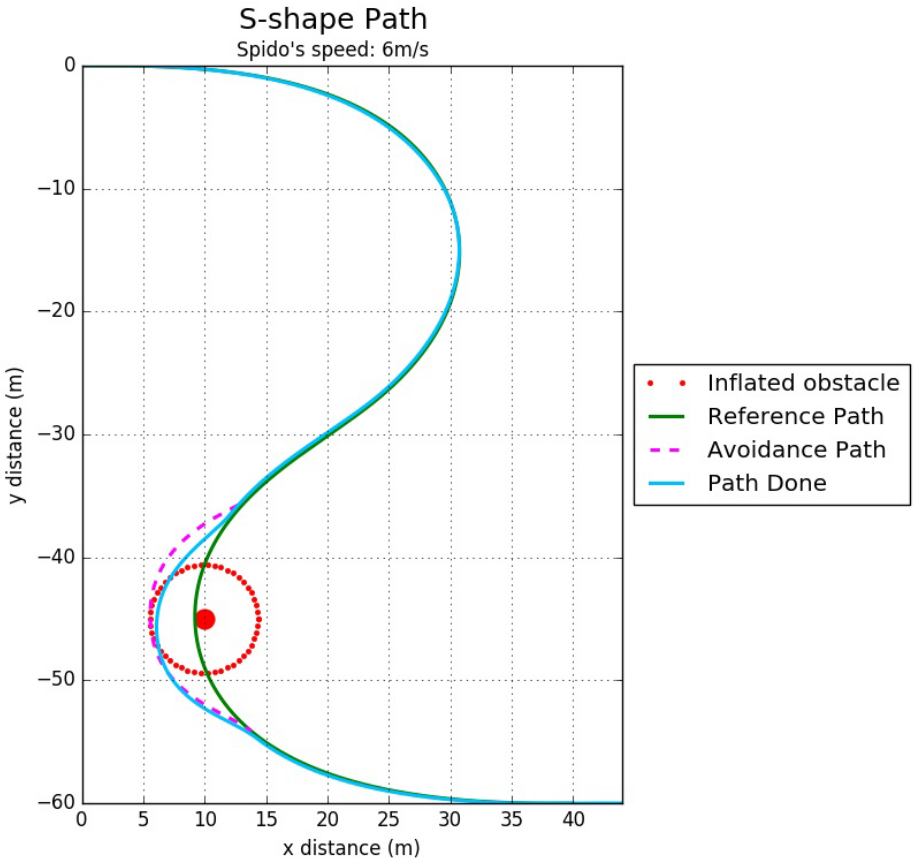


Figure 5.6 – Simulation test for the S-shaped path with a speed of 6m/s and its tracking error

5.1.3 O-shaped path

The figures 5.7 and 5.8 below show, respectively, the results for the simulation tests in a O-shaped path, and the tracking error of those tests.

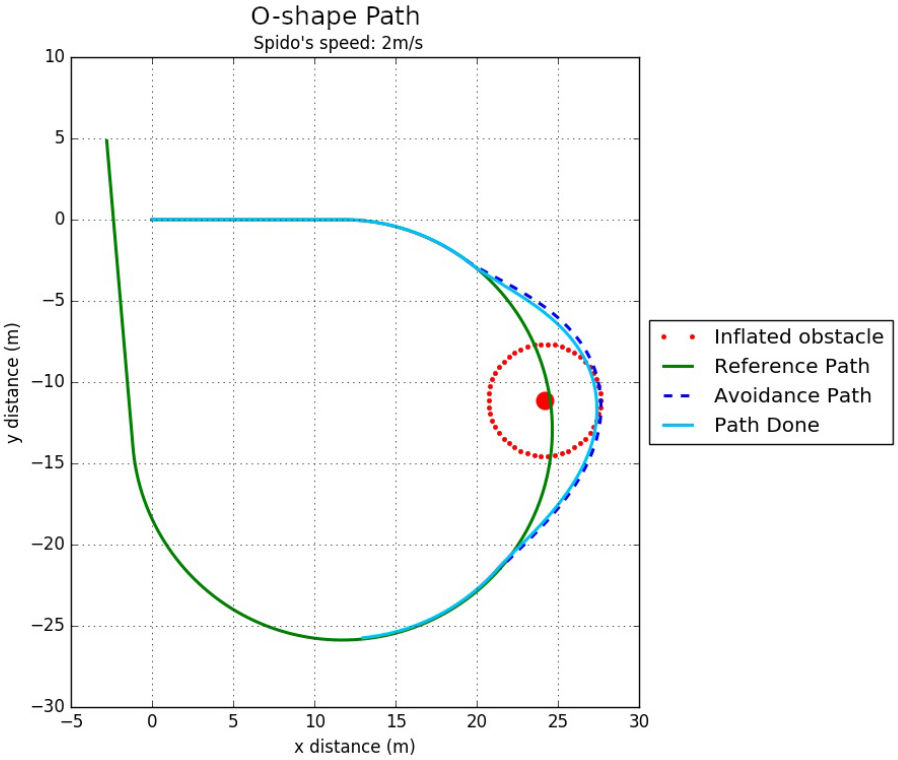


Figure 5.7 – Simulation test for the O-shaped path with a speed of 2m/s and its tracking error

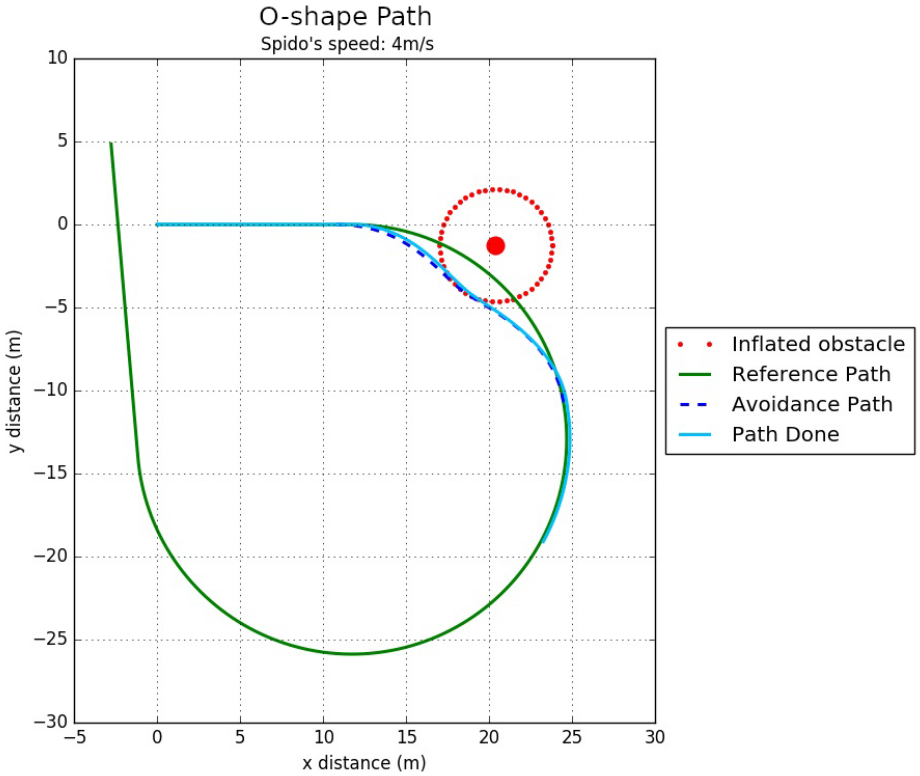


Figure 5.8 – Simulation test for the O-shaped path with a speed of 4m/s and its tracking error

5.1.4 Multiple obstacles

The figures 5.9 and 5.10 below show, respectively, the results for the simulation test for two obstacles in a line path and the simulation test for three obstacles in a S-shaped path as well as the tracking error of those tests.

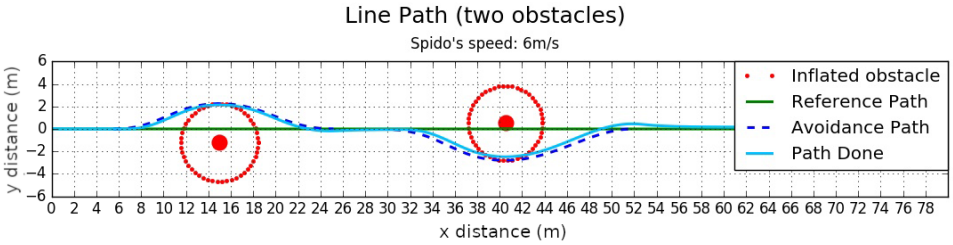


Figure 5.9 – Simulation test for two obstacles in a Line path with a speed of 6m/s and its tracking error

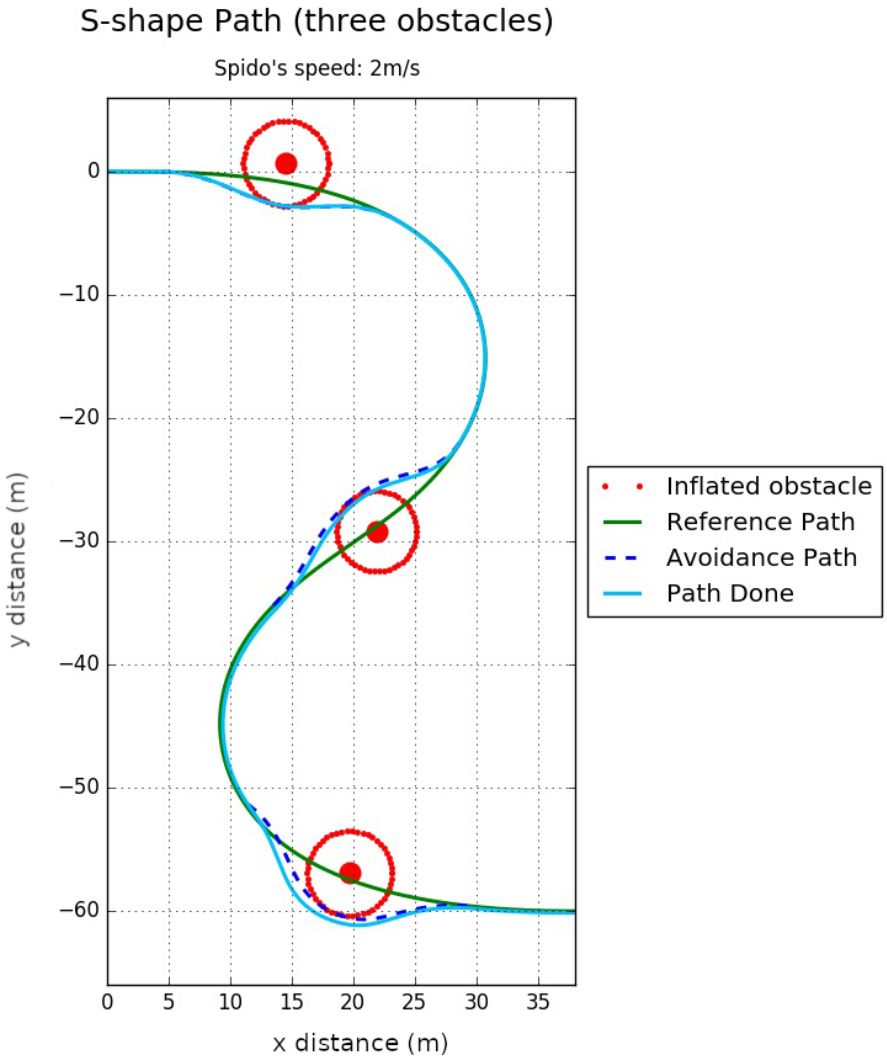


Figure 5.10 – Simulation test for three obstacles in a S-shaped path with a speed of 2m/s and its tracking error

5.1.5 Discussion of results

Analyzing the results obtained for the simulation part it is possible to verify that the obtained results show that the algorithm developed to avoid obstacles using two Bézier cubic curves demonstrates to be efficient for the required task. In the images above we see that the robot has successfully avoided one, two and even three obstacles. However, due to the simplifying hypothesis considered, in the cases of multiple obstacles the objects should be positioned at a distance so that they are not within the security circle of the other object detected by the robot.

Chapter 6

Conclusions and perspectives

Analyzing the results obtained in both virtual and experimental analyzes, it is verified that the obstacle detection algorithm and the avoidance path generation algorithm showed good results and achieved the expected objective for the conditions and restrictions proposed at the beginning of this work. However, to make result of this project to more useful in real-life problems, it is still necessary to make improvements in its functionality and structure to further generalize the operating restriction statements. For instance, we plan to control the longitudinal dynamics of the robot to improve the accuracy of the path tracking algorithms, to ensure the stability and safety of the vehicle, especially in major turnings where the curvature is minimal.

One of the first considerations made before starting the algorithms of this project was the consideration that the whole work ground is flat, without any ramps or major irregularities on the test ground. However, analyzing the Figure 3.8, It can be seen that even little ground slopes could interfere with the designed detection algorithm. Therefore, another point to be improved would be to include terrain slope conditions in the proposed algorithm.

Also, the algorithm proposed in this project considers only static objects with fixed and invariant positions. However, it is known that in more real situations, we rarely find only static objects in an environment. In this sense, the developed algorithm can be improved in future works, so that it can perform the detection of dynamic obstacles as well.

In addition to it, as mentioned before, several works have already been developed focusing on the detection and identification of obstacles and also in the techniques to generate avoidance trajectories. All these works have positive and negative points related to the use of their methods. Thus, to verify which technique is most appropriate for a

given task it is necessary to make a comparison between the various possibilities available to be used for a given problem. In future works, it would be important to compare the proposed method in this project with other proposed methods, to verify which methods present better efficiency in general, to verify the quality of the proposed algorithm.

Bibliography

[1] Mohamed Krid, Faiz Benamar, and Roland Lenain. “A new explicit dynamic path tracking controller using generalized predictive control”. In: *International Journal of Control, Automation and Systems* 15.1 (2017), pp. 303–314.

[2] Mohamed Krid and Faiz Benamar. “Design and control of an active anti-roll system for a fast rover”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 274–279.

[3] M Fnadi et al. “Path Tracking Control for a Double Steering Off-Road Mobile Robot”. In: *ROMANSY 22–Robot Design, Dynamics and Control*. Springer, 2019, pp. 441–449.

[4] M. Fnadi, F. Plumet, and F. Benamar. “Nonlinear Tire Cornering Stiffness Observer for a Double Steering Off-Road Mobile Robot”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 7529–7534. DOI: <10.1109/ICRA.2019.8794047>.

[5] Marcus Lundberg. “Path planning for autonomous vehicles using clothoid based smoothing of A* generated paths and optimal control”. MA thesis. KTH, Numerical Analysis, NA, 2017.

[6] Suhyeon Gim. “Flexible and Smooth Trajectory Generation based on Parametric Clothoids for Nonholonomic Car-like Vehicles”. PhD thesis. Clermont Auvergne, 2017.

[7] Chebly Alia et al. “Local trajectory planning and tracking of autonomous vehicles, using clothoid tentacles method”. In: *2015 IEEE intelligent vehicles symposium (IV)*. IEEE. 2015, pp. 674–679.

[8] J. Guldner et al. “Tracking gradients of artificial potential fields with non-holonomic mobile robots”. In: *Proceedings of 1995 American Control Conference - ACC’95*. Vol. 4. 1995, 2803–2804 vol.4. DOI: <10.1109/ACC.1995.532361>.

[9] N. Noto et al. “Steering assisting system for obstacle avoidance based on personalized potential field”. In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. 2012, pp. 1702–1707. DOI: <10.1109/ITSC.2012.6338628>.

[10] D. Bodhale, N. Afzulpurkar, and N. T. Thanh. “Path planning for a mobile robot in a dynamic environment”. In: *2008 IEEE International Conference on Robotics and Biomimetics*. 2009, pp. 2115–2120. DOI: <10.1109/ROBIO.2009.4913329>.

[11] Rodrigo Ventura. *Understanding ROS Topics. Available at <http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Tutorials(2f)UnderstandingTopics.html>*. URL: <http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Tutorials(2f)UnderstandingTopics.html>.

[12] Jeff Delmerico. *PCL Tutorial: The Point Cloud Library By Example*. 2013. URL: <http://www.jeffdelmerico.com/wp-content/uploads/2014/03/pcl_tutorial.pdf>.

[13] D. Ghorpade, A. D. Thakare, and S. Doiphode. “Obstacle Detection and Avoidance Algorithm for Autonomous Mobile Robot using 2D LiDAR”. In: *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. 2017, pp. 1–6. DOI: <10.1109/ICCUBEA.2017.8463846>.

[14] A. N. Catapang and M. Ramos. “Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle”. In: *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. 2016, pp. 441–445. DOI: <10.1109/ICCSCE.2016.7893614>.

[15] M. A. S. Teixeira et al. “A pose prediction approach to mobile objects in 2D costmaps”. In: *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. 2017, pp. 1–6. DOI: <10.1109/SBR-LARS-R.2017.8215314>.

[16] J. Seok et al. “Diverse multi-path planning with a path-set costmap”. In: *2011 11th International Conference on Control, Automation and Systems*. 2011, pp. 694–699.

[17] Gustavo Nunes Wagner. “Visualização Interativa de Modelos Massivos de Engenharia na Indústria de Petróleo com o Algoritmo de Voxels Distantes”. PhD thesis. PUC-Rio, 2007.

[18] Zhe Wang, Hong Liu, and Tao Xu. “Real-Time Plane Segmentation and Obstacle Detection of 3D Point Clouds for Indoor Scenes”. In: vol. 7584. Oct. 2012. DOI: <10.1007/978-3-642-33868-7_3>.

[19] Jack O’Quin, David Claridge, and Michael Quinlan. *velodyne_Height_map. Available at <http://wiki.ros.org/velodyne_height_map>*. URL: <http://wiki.ros.org/velodyne_height_map>.

[20] *Classification Using Nearest Neighbors. Available at <https://www.mathworks.com/help/stats/classification-using-nearest-neighbors.html>.* URL: <https://www.mathworks.com/help/stats/classification-using-nearest-neighbors.html>.

[21] Radu Bogdan Rusu. "Semantic 3D object maps for everyday manipulation in human living environments". In: *KI-Künstliche Intelligenz* 24.4 (2010), pp. 345–348.

[22] M Fnadi, F Plumet, and F Ben Amar. "Model Predictive Control based Dynamic Path Tracking of a Four-Wheel Steering Mobile Robot. In : IROS 2019". In:

[23] Yves Bertot, Frédérique Guilhot, and Assia Mahboubi. "A formal study of Bernstein coefficients and polynomials". In: *Mathematical Structures in Computer Science* 21.4 (2011), pp. 731–761.

[24] Heloisa B Medeiros and M Lucia Menezes. "Aproximaçao de funçoes: polinômios de Bernstein". In: ().

[25] Kenneth I Joy. "Bernstein polynomials". In: *On-Line Geometric Modeling Notes* 13 (2000).