



OBSTACLE AVOIDANCE SYSTEM STACK IN ROS

By

RAUL SERRANO MARTIN

Master in Systems and Control
UCM UNED

Supervisor:
Matilde Santos

SEPTEMBER 2021

Abstract

During this dissertation the work to create a path planner has been explained. It is a field that still has a long way to go, especially in the judicial field. The technology is already mature enough to have proof of concept, but there are still many uncovered scenarios. During this work, different autonomous driving techniques have been studied with the aim of creating a local path planner. The algorithm chosen for the path planning is Bezier curve algorithm. It can draw a path given a goal point and a goal orientation. It has been embedded into a ROS module concept called node so universities or other research organizations can take the advantage out of it. It has been tested on a global planner with the use of Gazebo.

An obstacle avoidance system has been implemented on top of it. The system detects local obstacles and traces an alternate path. With the latest enhancement added during this dissertation, 360 degrees sensors can detect obstacles on vehicle's back so the execution is halted in case it moves backwards and encounters an obstacle . Also the twirling behaviour from the vehicle can be tweaked so that the smoothness of how it rotates can be controlled. Several parameters have been added to the configuration for that purpose. Using techniques such as Support Vector Machines, Histogram of Gradients and Sliding Window Protocol, obstacles on the road can be detected with the use of Artificial Vision. Using a real video, the use of these techniques has been simulated obtaining promising results. Conclusions and proposals for future improvements have been made to continue with this project in the future.

Dedication and acknowledgements

I dedicate this work to my family for their unconditional support

Acronyms

ROS	Robot Operating System
RVIZ	ROS visualization
NHSA	National Highway Traffic Safety Administration
RNDF	Road Network Description File
WHO	World Health Organization
AVs	Autonomous Vehicles
DGT	Dirección General de Tráfico
SDC	Self Driving Car
SDCND	Self Driving Car Nanodegree
LIDAR	LIght Detection And Ranging
SONAR	SOund Navigation And Ranging
RADAR	RAdio Detection And Ranging
Qt	Q toolkit
GPS	Global Positioning Sytem
AMCL	Adaptive MonteCarlo Localization
KLD	Kullback–Leibler Divergence
MLE	Maximum Likelhood Estimation
DWA	Dynamic Window Approach
SVM	Supporting Vector Machines

Table of Contents

Page

List of Tables **xi**

List of Figures **xiii**

1 Introduction **1**

1.1 Baseline . . . 1
1.2 Historical background . . . 1
1.3 Socio-economic context . . . 3
1.4 Legal framework . . . 4
1.5 Methodology . . . 5
1.6 Budget . . . 7
1.6.1 Personnel expenses . . . 7
1.6.2 Software cost . . . 7
1.6.3 Hardware cost . . . 7
1.7 Motivation . . . 7
1.8 Objectives . . . 8
1.9 Dissertation structure . . . 9

2 State of the art **11**

2.1 Graph Search Based Planners . . . 11
2.1.1 Dijkstra's algorithm . . . 12
2.1.2 A* algorithm . . . 12
2.1.3 D* algorithm . . . 13
2.1.4 Anytime A* . . . 14
2.2 Mapping and exploration . . . 14
2.2.1 Particle filters . . . 14
2.2.2 Coverage maps . . . 15
2.3 Localization techniques . . . 15
2.3.1 Vehicle Model . . . 16
2.3.2 Occupancy grid maps . . . 16

2.3.3 LandMark-Based Maps . . . 17
2.3.4 Extended Kalman Filters . . . 17
2.3.5 Manhattan distance . . . 18

3 System design **19**
3.1 Drone system design . . . 19
3.2 Four-wheeled vehicle design . . . 20
3.3 Flowchart analysis . . . 21
3.3.1 Functional design . . . 22
3.3.2 Flowchart decomposition . . . 22
3.4 ROS navigation . . . 24
3.4.1 ROS Architecture . . . 24
3.5 Path design . . . 27
3.6 Adaptive MonteCarlo Localization . . . 27
3.7 tf Package . . . 29
3.8 Trajectory Planner in ROS . . . 32
3.8.1 Consider obstacles when escaping velocities . . . 32
3.8.2 Cost function to prevent unnecessary spinning . . . 33
3.9 Obstacle detection using Supporting Vector Machines . . . 33

4 System implementation **37**
4.1 Development in C++ . . . 37
4.1.1 Bezier function . . . 37
4.1.2 Implementation in ROS . . . 39
4.1.3 AMCL algorithm . . . 47
4.1.4 Considering obstacles when escaping trajectories . . . 48
4.1.5 Cost function to penalize robot spinning . . . 50
4.1.6 Obstacle detection using SVM . . . 50

5 Testing and Results **55**
5.1 Gazebo . . . 55
5.2 Adaptive Monte Carlo Localization and Maps . . . 57
5.3 Bézier curves in ROS . . . 58
5.3.1 Transform the curve from map to odom . . . 59
5.3.2 Behavior of the routes depending on different orientations and positions . . . 59
5.4 Avoiding obstacles in ROS . . . 62
5.4.1 Simulation in Gazebo . . . 63
5.4.2 Avoiding obstacles when moving backwards . . . 64

6 Conclusions and Further improvement **67**

6.1 Contributions summary 67
6.2 Conclusions 68
6.3 Future work 68

A Appendix: The source code **71**

Bibliography **75**

List of Tables

Table **Page**

3.1 Drone requirements table 20
3.2 Four-wheeled vehicle requirements table 21

5.1 Parameters from each curve 60

List of Figures

Figure **Page**

1.1 Graph of deaths 4
1.2 SDC World Legal situation 6

2.1 Elastic Band forces 12
2.2 Mobile robot 16
2.3 Dijkstra algorithm [1]. 18
2.4 A star algorithm [2]. 18

3.1 Drone design. 22
3.2 The Logical Decomposition phase 22
3.3 Main flowchart. 23
3.4 Pascal triangle. 24
3.5 ROS architecture 25
3.6 Data Distribution Service 25
3.7 ROS client library 26
3.8 Graph level scheme 27
3.9 Graph level scheme 28
3.10 tf tree 28
3.11 Tf transform tree 29
3.12 iCab side view 30
3.13 Drone axes of symmetry 30
3.14 PoseArray messages chart 31
3.15 PoseArray messages chart 31
3.16 PoseArray messages chart 32
3.17 Computer Vision comparison 34
3.18 Vehicle detection flow 34

4.1 Orientation calculation 40
4.2 Orientations in RVIZ 41
4.3 Transform from map to base link in RVIZ. 42

4.4 Parametrization into map grid . . . 43
4.5 Obstacle avoidance . . . 44
4.6 Obstacle avoidance abstraction . . . 45
4.8 Obstacle avoidance abstraction . . . 48
4.7 Obstacle avoidance abstraction . . . 49
4.9 Class diagram . . . 50
4.10 Class diagram . . . 51
4.11 Config parameters . . . 51
4.12 Accuracy . . . 52
4.13 First effort . . . 52
4.14 Heat map . . . 53
4.15 Heat map threshold . . . 53
4.16 Final detection . . . 53

5.1 Robot in isometric perspective . . . 56
5.2 Dimensions of P3AT robot . . . 56
5.3 Gmapping map scheme . . . 57
5.4 Hokuyo laser scan . . . 57
5.5 Hokuyo laser and map . . . 58
5.6 Bezier curve from map. . . . 58
5.7 Bezier curve in RVIZ . . . 58
5.8 Bezier transform to odom. . . . 59
5.9 Bezier transform in RVIZ . . . 59
5.10 Set of different paths with different orientations. . . . 60
5.11 Set of poses from each path. This view comes from a "PoseArray" message. . . . 60
5.12 Group A curves . . . 60
5.13 Group B curves . . . 61
5.14 Group C curves . . . 61
5.15 One obstacle avoidance abstraction . . . 62
5.16 One obstacle avoidance in RVIZ . . . 62
5.17 Two obstacle avoidance abstraction . . . 63
5.18 Two obstacle avoidance in RVIZ . . . 63
5.19 Straight line plan . . . 63
5.20 Global and Local Plan in Gazebo . . . 64
5.21 Initial position when moving backwards . . . 65
5.22 Final position when moving backwards . . . 65

CHAPTER 1

INTRODUCTION

The objective of this section is to portray the Autonomous Vehicles Market (AVM) and to show its momentum challenges to welcome a foundation on the explored content. The Legal Framework has been concentrated around the world to bring a thought of what the circumstance is in each confinement. The means followed to arrive at the outcomes are acquainted in the approach and with finish up, the time and financial costs.

1.1 Baseline

As a reference for the realization of this work, the **bachelor thesis** from the same student has been used as a starting point. New content has been added to complete the work and to further elaborate the points. In the past project an obstacle avoidance system was designed which was not capable of detecting obstacles behind the robot.

Using a broad range sensor, the planner can now identify obstacles on his back. If it identifies an obstacle that could cause the robot to malfunction, it promptly shuts down the system.

A threshold for the twirling has been added. This avoid sudden changes in the robot heading so we can control the twirling behaviour from the robot.

Through the use of Support Vector Machines and other computer vision techniques, it has been possible to detect obstacles, in this case other vehicles and it has been connected to the complete ROS system.

1.2 Historical background

In 1925 the creator Francis P. Houdina developed the main viewed self as "driving" vehicle and showed its working at New York City's public screens. Houdina introduced a reception

apparatus in the vehicle and utilized a controller from a subsequent vehicle. As the New York Times announced in those days, the vehicle turned over its motor and pivoted through its different pinion wheels "as though an apparition hand were in the driver's seat"[3].

Later in 1950, electrical motivations were utilized for far off driving to distinguish the area and speed of different vehicles just as to give data to self-driving vehicles. The issue was that they need recipients as they were assembled indicating certainty that their usage in streets will turn out to be soon accessible and tragically, they didn't. It was guaranteed that the innovation could forestall 40 percent of street mishaps, by and by, electronic streets never got famous. [4]

Subsequently, Cruise control was actualized in 1958 which permit the vehicle to keep up its speed without driver[5]. Anti-Lock Braking System (ABS) was introduced in cars some time after it was first proved in aircraft systems.

The Sure-Brake System investigates the information from the wheel speed estimations to recognize slipping and transfer orders to a pressure driven modulator. All these robotized procedures have a place with the "Level 1" of the NHTSA[1] classification. The NHTSA is an association that during the 1980s made another enactment promising vehicles less expensive to fix, more secure and less contaminating [6].

However, when we talk about autonomous vehicles, the name that usually appears is that of Norman Bel Geddes, an American industrial designer who stood out precisely for his extravagant but futuristic designs for the time.

His most recognized work is the one he did for General Motors. His commission consisted of designing a pavilion entitled *Futurama for the World's Fair* in New York in 1939. In the present case, an autonomous vehicle that worked by electricity and was controlled by radio control stands out. The car also moved by an electrical circuit built into the pavement.

Between 1940 and 1980 there were more steps in search of the definitive autonomous vehicle. However, we jump back to the 80s of the 20th century to talk about the German Ernst Dickmanns, who is considered the father of the autonomous vehicle as we know it today with the permission of Houdina, Bel Geddes and other forerunners. In addition, he is an expert in robotics and artificial intelligence.

In 1994 he would do something similar with a Mercedes, which traveled more than 1,000 kilometers through Paris avoiding, and even overtaking, other vehicles. And in 1995, a Mercedes-Benz traveled autonomously between Munich and Copenaghe. These projects were funded by the European Commission under the umbrella of the Eureka project or Eureka Prometheus Project, which awarded Dickmanns almost 800 million euros to research on autonomous vehicles. [7] In the summer of 2015, the University of Michigan launched a utility town, called MCity, where autonomous vehicles can be tested on the streets. The place has a street more than a kilometer and a half long, curves with different radii, roundabouts, traffic lights, pavements of different surfaces, etc. 16 On the other hand, there are other projects that will complement the MCity.

[1]National Highway Traffic Safety Administration: They promote vehicle safety innovations and set safety standards.

One of them is the placing of 9,000 interconnected vehicles in the large area of Ann Arbor, as well as another 20,000 on highways in the southeast of the state, two units of the first prototype of autonomous electric vehicle manufactured in that country. It is a vehicle without a steering wheel or pedals.

The first thing we have to take into account when talking about autonomous driving is the fact that we are dealing with a machine, a large computer on wheels that makes decisions based on unquestionable guidelines.

Thus, on a road with a speed limit of 80 km / h, the autonomous car will never exceed that limit unless its predefined operating parameters are altered. Ambiguity does not exist in the decision making of a machine.

If we stick to theory, the autonomous car is the perfect figure as a transport system, where its operation is based on compliance with imposed laws and guidelines. There is no interpretation, much less violation of the rules except failure. That said, the figure of the autonomous car is understood as the best advance in safety that has occurred in recent years, because its operation is extracted control and prudence impossible to achieve by a human being. [8]

Anthony Levandowski and Chris Urmson helped create the self-driving car industry as we understand it today. They were polar opposites and they fought over big issues and also over trifles. Urmson won the power struggle and Levandowski ultimately turned to Uber. Several years of infighting and indecision hampered Google's project to launch a self-driving product while Uber and others rushed to develop it themselves, though not without difficulties.

1.3 Socio-economic context

It's not just hands-free driving that self-guided cars may bring into reality. Autonomous vehicles could unleash new technological and economic development on a level not seen since the dawn of the space age. Even if autonomous cars merely succeeded in eliminating accidents caused by driver mistake and incompetence, the annual savings would be in the hundreds of billions of dollars. Because AVs allow for considerably smoother traffic flow, traffic congestion might be alleviated as a result of fewer accidents. Other experts, on the other hand, said that AVs might have the unintended consequence of worsening traffic congestion by placing more automobiles on the road.

Without upheld activity and assets, the conjecture states car accidents will turn into the seventh reason for death by 2030.

Street car accidents cost around the 3 percent of total national output to a large portion of the nations. Roughly 50 percent of those perishing on the world's streets are "weak street clients": walkers, cyclists, and motorcyclists.

A lot of the world's mortality on the streets happen in low-and center pay nations still, these nations have roughly 54 percent of the world's vehicles. The main source of death in individuals

which ages are involved somewhere in the range of 15 and 29 years is Road traffic injuries.

As distracted driving, unsafe roads and unsafe vehicles are the main factors [9].

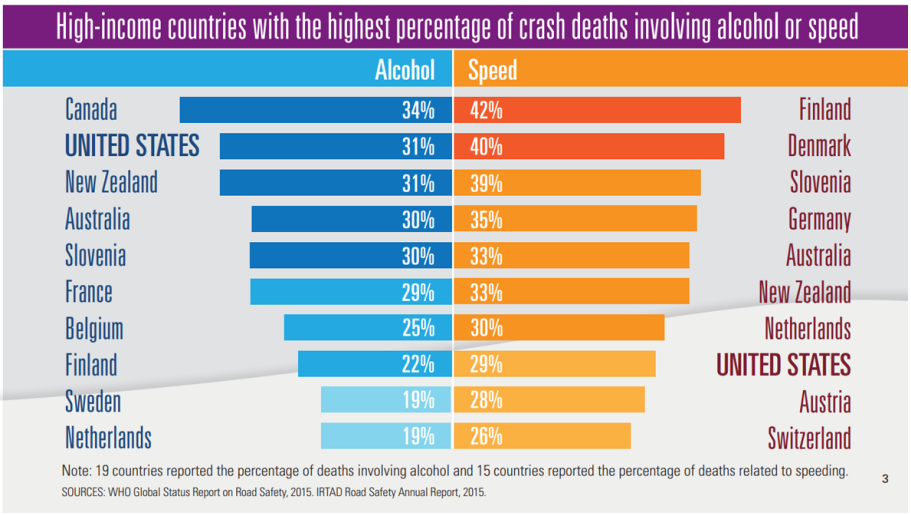


FIGURE 1.1. Graph of deaths

[10]

1.4 Legal framework

Self-driving cars have been a quick advancing field throughout the most recent fifty years. Because of this quick change, a few governments are thinking that its hard to take estimates fitted to the innovation headway. To get familiar with the lawful constraints in each domain every nation ought to be concentrated individually.

Recently in Spain there has been a few endeavors to manage the independent driving. DGT [2] and Mobileye, a company specialized in ADAS [3], have collaborated to improve driving wellbeing and to prepare to self-governing vehicles. They are essential for Vision Zero, a Swedish activity which target is to accomplish no passings or genuine wounds[11]. As far as the law, the enactment is plain. The solitary sentence the Spanish government states is: "Just producers and foundations can test vehicles without driver and in certain conditions"[12].

Regarding different nations of the European Union, Germany is the principal country ready to manage Autonomous Vehicles inside a lawful system. The German government has taken a few estimations inside the enactment on June 21st 2017. In the event of a car crash, they request the discovery record from the excursion in progress which has enlisted whether the reason for the crash was simply the human factor or the vehicle's directing system.[13].

[2]Dirección General de Tráfico. It is an autonomous body under the Ministry of the Interior of Spain responsible for the execution of the road policy.

[3]Advanced driver-assistance systems: These acronyms have been adopted by automakers to refer to systems that help the driver within driving decisions.

The legislation has made prerequisites to high level self-driving cars. In any case, the driver should be prepared to take over again if and when required. Notwithstanding this, this new methodology doesn't permit self-autonomous driving when the entirety of the occupants are just travelers. As expressed toward the start of the section, there exist various guidelines as per every country. That is the motivation behind why a few organizations are endeavoring to make a universally harmonized legitimate framework.

At the worldwide level there are numerous arrangements that close a lawful arrangement to improve every country's enactment. It should be noticed the one completed in Vienna in 1968, albeit in this year robotized frameworks have not been grown at this point and subsequently, no structure was characterized likewise. In that time it was viewed as that vehicles were just constrained by human drivers. ADAS has been permitted since an adjustment in the law in 2016 however self-driving system isn't yet conceivable. [14]

The law to advance security of driverless vehicles is to be prepared by 2021 by the Law Commission in United Kingdom [15]. As the politician Jesse Norman states, they are world pioneers in self-driving examination and improvements. One of the parts of the test is that the vehicles can work with wellbeing and proficiency in every single climate condition [16].

Outside Europe, American automakers have an incredible achievement because of their administrative system "Dissimilar to in Germany and Europe, the law in the United States as of now allows profoundly computerized vehicles to be utilized in a business setting" [17].

1.5 Methodology

The first contact in the world of autonomous driving was thanks to online MOOCs, specifically one from Udacity. It was a way of doing an exhaustive analysis of the autonomous systems industry and learning about how to model vehicles that know how to drive. It added to the elaboration of the State of Art in Chapter 2 about Path Planners.

Furthermore, by using the Amazon program called DeepRacer I have managed to put into practice some of the most up-to-date autonomous driving techniques which I have implemented in this project.

A bibliographical check through IEEExplore presumed that Bezier curves might be a compelling choice to plan waypoints ways. To check its conduct a first calculation was tried with Matlab offering a see of its advantages and impediments. As this calculation ought to be actualized in complex frameworks, we concocted utilizing ROS to facilitate the coordination cycle. In light of ROS just supports Python and C++ programming languages, C++ was picked because of it is computationally proficient.

My main idea during this work has been to implement a Global Planner in conjunction with a Local one. To find such a candidate, Github has been my reference tool. Once I had decided which planner to use, I had to investigate the different types of robots and sensors that would fit my

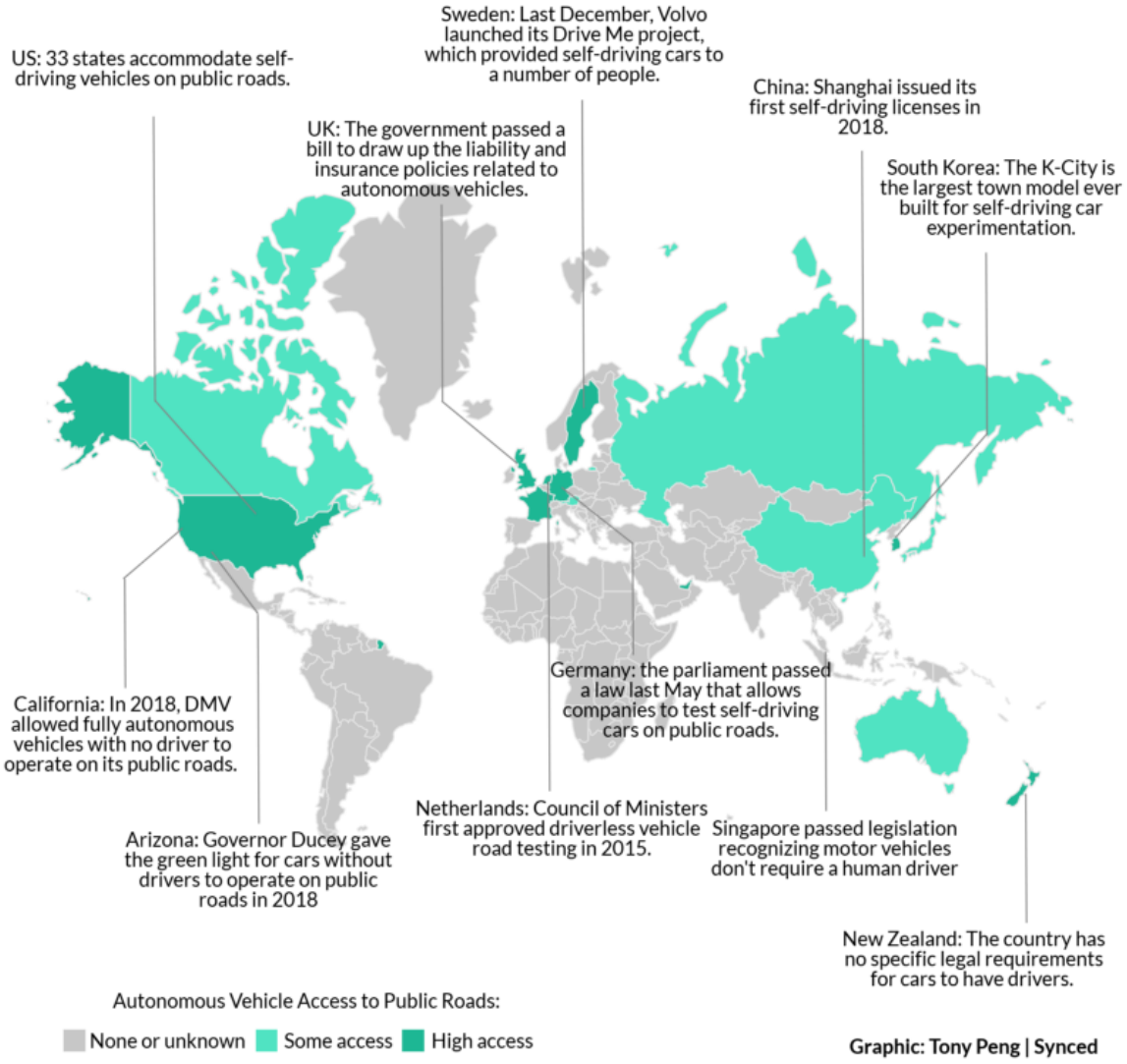


FIGURE 1.2. SDC World Legal situation. Figure taken from [18]

project.

1.6 Budget

All the costs related with the project are depicted in this segment. The last expense accounts 2216,11€

1.6.1 Personnel expenses

Following the spanish regulation at "National collective agreement of engineering companies and technical studies offices" the yearly compensation for an engineer in Spain should be 20.424,25€ each year with a limit of 1800 working hours. Then the minimum salary is calculated as 20.424,25€/1800h = 11,35€/h [19]. The compensation accounted in this venture can be determined as 11.35€/h * 120h = 1362€.

1.6.2 Software cost

Some paid licenses has been using during the development of this project as well as some open source programs. A Linux distro was installed on the desktop. Qt creator was used for the development of the code, RVIZ is the programm that allows us to see the real time simulation as is integrated with ROS. Matlab wich is not open source has an annual expense of 800€. Overleaf is the tool used for writing the thesis which amounts to a cost of 10.50€ per month.

1.6.3 Hardware cost

For the Hardware components, a Nvidia GTX 1050 Ti which costs 200€ has been used. Its devaluation period is of 8 years so it accounts to 25€ each year. The final project hasn't been tested over a real vehicle so that cost can't be computed. A desktop that costed 900€ in 2016 has been the computer machine where the tests have been executed and the code has been written having a devaluation of 9.03€

1.7 Motivation

Generally the solitary individual responsible for the vehicle could be a human. Driver help systems have been developing during time beginning from innovations like Adaptative light control to the further developed systems, for example, ADAS [4]. Those frameworks, in blend with advanced robotics may permit to precisely quantify the current status of the vehicle and system just as anticipate its behavior.

[4]Advanced driver assistance-system are mechanisms to help the driver with driving tasks

There exist numerous algorithms like A* and Dijkstra that have been actualized in many programming languages, anyway jut a couple are appropriately incorporated with ROS. Normally, these algorithms are intended for discrete circumstances, similar to lattice maps or different methods for discretization. Furthermore, some changing situations can expect consistent capture of the obstacles that may prompt a higher computational expense. An advanced vehicle focuses a stack of sensors which should be regulated. A fully connected system working in parallel may not be productive enough since certain stages just require certain information. The most ideal situation would be a system wherein the driver could focus on the thing data is useful for you without the need of gathering the information from all the sensors for the analysis. All boils down to a local planner system with ease computational endeavors that can be arranged in ROS. A few Universities are requesting an Open Source Path planner [5] structure to incorporate into its robots and begin building once again it and to satisfactory to its specific prerequisites. The hardest advance is to establish a environment which capacities are not restricted by the system plan and where to begin building. That is the reason the previously mentioned planner has been designed with a environment-friendly expectation and simple to separate structure that brings unlimited prospects.

1.8 Objectives

The motivation behind this project is to build up a structure of a Local Path-planning framework in ROS based on Bézier curves that is equipped for recognizing obstacles and evade them offering other way options. Explicit goals can be pointed as follows:

- Investigate and recognize the necessities of the required path planning framework
- Dissect and recognize the structure of a ROS framework.
- Create a structure in ROS to design Bézier curves.
- Make a Node[6] that designs a trajectory given an underlying and an objective point.
- Design that Node to various reference frames utilizing the ROS Transforms library.
- Plan a working model for the Obstacle Avoidance System and organize it so it tends to be additionally stretched out in future examinations.
- Test the Local Planner with Bézier curves integrated into a global one.
- Design and program an obstacle recognition system on the back.
- Design and program a system to smooth the twirling behaviour from the robot.

[5] It is a term used in robotics for the mechanism of dividing a route into small problems to solve
[6] It is a process that plays out a calculation in ROS

1.9 Dissertation structure

During **Chapter 1**, historical context has been provided as well as a socio-economic analysis of autonomous systems together with their regulations. At the end of it, the motivation and the objectives that are intended to be achieved during this work are also discussed. **Chapter 2** talks about the field of study of road planners. Some location techniques that have served as a reference for this project are also studied. In **Chapter 3** a high-level vision of the requirements that an autonomous system must have as well as some schematics to understand the system is offered. A lower level view is explained in **Chapter 4**. It talks about the algorithm and the methods and technologies used to achieve the proposed objectives. During **Chapter 5** these algorithms are simulated to obtain results and analyze them. Some frameworks like Gazebo are used. Finally, in **Chapter 6** the conclusions of this dissertation are collected and evaluated and therefore future improvements are proposed.

CHAPTER 2

STATE OF THE ART

Self-driving vehicles progression was extremely restricted before the 90's since organizations and governments were not placing a lot of cash into it. The advancement of the data innovation, particularly Artificial Intelligence in the most recent decade has started more interest in the subject. The primary focal point of this part is to clarify all the exploration previously done in the field of Path Planning in Intelligent Vehicles.

Path planning is applied to describe a set of problems that are related to moving an object (robot) from an initial point to an end point avoiding colliding with possible obstacles in the environment.

For the analysis of these problems, the Euclidean space in two dimensions is usually used and sometimes it is considered that the collision with an obstacle only occurs with its internal points (assuming the obstacles as simple polygons) and not with the border, although in practical terms it may not be safe for a robot to pass so close to an obstacle. According to their implementation in automated driving: **graph search, sampling, interpolating and numerical optimization**. By the end of this chapter, some graphic examples have been disposed to ease the process of understanding each planner [20].

2.1 Graph Search Based Planners

Thanks to the theory of graphs, various problems can be solved, such as the synthesis of sequential circuits, counters or opening systems. It is used for different areas such as computational drawing or in engineering areas. Once the graph is provided, its cost is minimized as much as possible. For measuring that aim, g(s) is the function used to estimate the least cost path from the origin to the target point. As it can be observed in the diagram, the arrows that points from one node to the next one contains the bounded value for each of the snaps from the trajectory. Over each of

the nodes, the value from the function g(s) is detailed [21] . The final value for that same function can be calculated as follows:

$$g(s) = min_s g(s'') + c(s'', s)$$

In the equation , s" to s is represented as any of the edges from the diagram 2.1

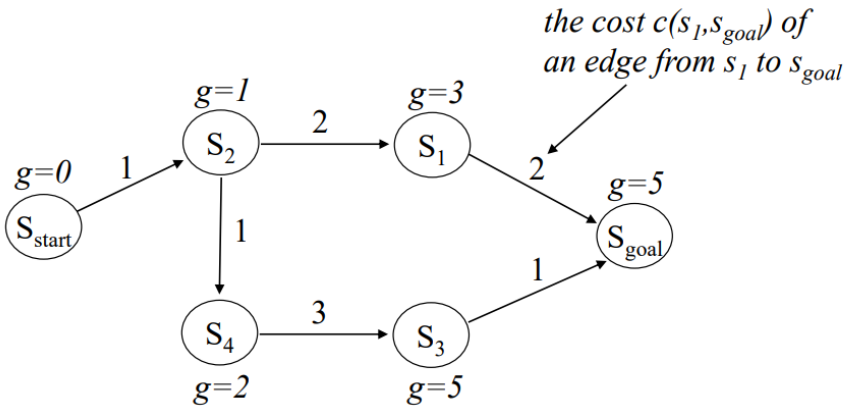


FIGURE 2.1. Graph search. Figure taken from [21]

In this case, graphs are used for searching the best route between two points. The next algorithms are going to be discussed in the following subsections: Dijkstra's algorithm, A* algorithm and State Lattices.

2.1.1 Dijkstra's algorithm

Dijkstra's algorithm, also called the least paths algorithm, is an algorithm for determining the shortest path, given an origin vertex, towards the rest of the vertices in a graph that has weights on each edge. Its name refers to Edsger Dijkstra, a computer scientist from the Netherlands who conceived it in 1956 and first published it in 1959. [1].

To carry out the formulation of a network model with the Dijkstra algorithm, it is necessary to know the starting, ending and transshipment nodes of the network, subsequently the labels that are presented in the arcs of the network are analyzed, in order to be able to determine the objective function of the problem to be formulated.

2.1.2 A* algorithm

The problem with some search algorithms in informed graphs, such as the greedy algorithm, is that they are exclusively guided by the heuristic function, which may not indicate the lowest cost path, or by the real cost of moving from one node to another (such as escalation algorithms), and it may be the case that a higher cost movement is necessary to reach the solution. It is therefore

quite intuitive that a good informed search algorithm should take into account both factors, the **heuristic value of the nodes** and **the real cost of the journey**.

A * maintains two auxiliary data structures, which we can call open, implemented as a priority queue and closed, where the information of the nodes that have already been visited is saved. In each step of the algorithm, the node that is first open is expanded, and in case it is not a target node, it calculates the f(n) of all its children, inserts them in open, and passes the evaluated node to closed.

$$f(n) = g(n) + h(n) \tag{2.1}$$

where $g(n)$ is the cost from the source node to the current node n, and $h(n)$ is the heuristic cost of the path from the node n to the destination node. Among the candidate nodes, the node n with the smallest $f(n)$ is the first to be selected to check [22].

2.1.3 D* algorithm

Another incremental search algorithm is the D*. There are three different search algorithms that can be contained into D*: The **original D*** which is an informed incremental search algorithm, **Focused D*** that collides the ideas from A* and D* helping to create a more advanced algorithm and **D* lite** that combines the ideas from A* and Dynamic SWSF-FP. [23]

- Expansion: One of the singularities from the D* algorithm is that it begins searching backwards from the goal node. When each of the nodes have been visited, they contain a pointer pointing to the next node that is leading to the target, the origin.
- Obstacle avoidance: When an impediment is found along the planned path, all impacted locations are placed back on the OPEN list, but this time with a RAISE mark. However, before a RAISED node's cost grows, the algorithm analyzes its neighbors to see if it can lower the node's cost. If this is not the case, the RAISE state is propagated to all descendants of the node, that is, nodes with backpointers to it. After that, the RAISE state is passed on to the next node, producing a wave. When a RAISED node may be lowered, its backpointer is updated, and its neighbors are given the LOWER state. The core of D* is these waves of RAISE and LOWER moods.

  A entire succession of additional sites are protected from being "touched" by the waves by this point. As a result, the algorithm has only operated on the points.
- Deadlock: This time, there is no deadlock to get around the impasse. None of the points can discover a new path to the goal via a neighbor. As a result, they continue to increase their values. Only locations outside of the channel can be discovered that can lead to a valid path to the target. When a result, two Lower waves arise, which increase as places identified as unreachable with new route information become available.

2.1.4 Anytime A*

For real-world situations, optimal search is often impossible since we only have a certain amount of time to think, and the best answer within that time should be discovered. For that reason anytime algorithms are a good approach. They start off by iterating a very sub-optimal solution and, taking into account the allocated time, they iterate to improve it until the time expires. One might execute a series of these A* searches with decreasing inflation factors to create an anytime algorithm with sub-optimality constraints. This naïve technique yields a sequence of solutions, each of which has a sub-optimality factor equal to the inflation factor. The method, on the other hand, wastes a lot of computation because each search iteration replicates the majority of the previous iteration's efforts.

The ARA* algorithm is then proposed. This is an efficient anytime heuristic search that similarly repeats A* with inflated heuristics in succession but reuses search efforts from earlier executions to satisfy the sub-optimality limits. As a result, by not recalculating state variables that were successfully calculated in earlier cycles, a significant speedup is gained. The equation looks as in 2.2

$$f(n) = g(n) + h(n) * w \rightarrow w > 1 \tag{2.2}$$

It must be ensured that h(s) is never larger than the cost of getting to the goal from n. If we make the heuristic bigger then fewer expansions are needed and the searches perform much faster.

2.2 Mapping and exploration

Learning maps with single-robot systems often necessitates the completion of three tasks: mapping, localization, and path planning. The challenge of mapping is to combine the data collected by the robot's sensors into a particular representation. The question "How does the world appear?" can be used to explain it. The depiction of the environment and the interdisciplinary elements of mapping are crucial.

Before a robot can answer the question of what the environment looks like based on a collection of observations, it must first understand where the observations were taken. At the same time, estimating a vehicle's present location without a map is difficult. Planning a route to a certain destination is also closely linked to understanding of the surrounding area as well as the information available.

2.2.1 Particle filters

A set of hypothesis is used to predict the final state of the robot. Each of them is composed by a set of states s and weighted random samples w. Each of the weights have a value greater than 0

and the sum of all of them must be 1.

The distribution can be represented as in 2.2.1

$$p(x) = \sum_{i=1}^{N} w_i \delta_i(x)$$

For using the particle filter algorithm the system works as follows, the distribution of each of the particles is estimated with respect to time. The process can be divided into several steps:

- Sampling: The next set generation S is based on the previous S_1.
- Importance Weighting: An importance weighting is computed.
- Resampling: From the set St, take N samples. As a result, the probability of drawing a particle is related to its mass. The drawn particle provides the new set S_t.

2.2.2 Coverage maps

Occupancy grid maps are based on the fact that a cell can be occupied or not occupied, binary system. But it may be the case in which there is a wall that is not symmetrical to the x-axis or the y-axis. In the case of occupancy grid maps, that cell should be either 0 or 1. Coverage maps are flexible in that sense. Coverage values range between 1 and 0 where 1 means the cell is fully occupied and 0 that there is no obstacle on it.

It is needed to convert the distance to coverage values. A map that fits to the coverage for each region would be the output from the system.

2.3 Localization techniques

When working with industrial applications, the operating environment of the robot is usually fixed over a period of time. That is why probabilistic methods are very useful in this kind of situations. A uniform distribution is generally used to start a coverage map cell in order to indicate the highest uncertainty regarding the cell's real state.

2.3.1 Vehicle Model

The differential equation that describes how the robot position and orientation evolve with time as a function of its forward and angular velocity is known as the robot motion model. The relation that exists between the measurements from the sensor and the estimation of the location of the robot in the map is called **sensor model**. The kinematics of a moving robot are like:

$$\dot{x}(t) = (v(t) + \delta v(t))cos(\phi(t)) \tag{2.3}$$

$$\dot{y}(t) = (v(t) + \delta v(t))sin(\phi(t)) \tag{2.4}$$

$$\phi = w(t) + \delta w(t) \tag{2.5}$$

Where x and y describe the position from the center of the robot at a time t and the orientation ϕ, the angle between the heading of the robot and the horizontal x. [24]

2.3.2 Occupancy grid maps

Occupancy grid maps represent a discretized view of the map where each of the cells of the grid can be represented as two states, occupied or free. If for example, there is no obstacle within the sensor range, a reflection is not received and the sensor typically reports a nominal maximum distance **dmax**.

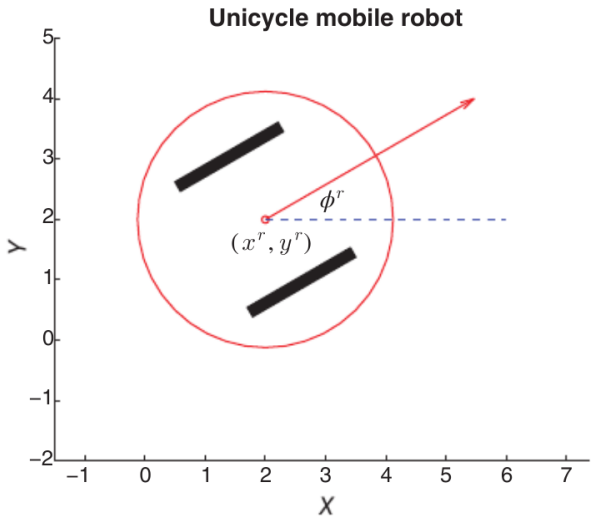


FIGURE 2.2. Mobile robot

2.3.3 LandMark-Based Maps

At each time step the robot is in motion, it will measure the distance and the angle that a straight line draws from it to a landmark in the map. The following equations show the relationship between the bearing and the distance with respect to the landmarks and the position of the robot.

$$r_{k+1} = \sqrt{(x_L - x_{k-1}) + (y_L - y_{k-1})} + w_r \tag{2.6}$$

$$\theta_{k+1} = atan(\frac{y_L - y_{k+1}}{x_l - x_{k+1}}) - \phi_{k+1} + w_\theta \tag{2.7}$$

Usually laser range finders are used for getting the range and the bearing of the landmarks.

2.3.4 Extended Kalman Filters

To summarize all the observations taken in the past and estimate the current location of the robot and the covariance matrix, Kalman Filters are used. Both the location and the covariance are updated whenever a new measurement becomes available.

$$u_k = (v_k, w_k)^T, w_k = (\gamma v, \gamma w)^T \tag{2.8}$$

Where u_k is the control and w_k is the zero-mean Gaussian process noise. Then, the future measurement at k+1 can be written as:

$$x_{k+1} = f(x_k, u_k, w_k) \tag{2.9}$$

Where f is the system transition function. The process can be splitted into two steps: Predict and Update. During the **Predict** stage, the value for the estimation and the error covariance associated to it is calculated using the laws of motion.

$$x_k = f(x_{k-1}, u_{k-1}) \tag{2.10}$$

$$p_k^- = F_{k-1} P_{k-1} F_{k-1} + Q \tag{2.11}$$

During the **Update** stage, the observations are taken into account. Predictions are compared against measurements.

Measurement residual

$$y_k = z_k - H(x_k)^- \tag{2.12}$$

Kalman Gain

$$K_k = P_k^- H_k^T (R + H_k P_k H_k^T)^{-1} \tag{2.13}$$

Update state estimate

$$x_k^+ = x_k + Kky \tag{2.14}$$

Updated error covariance

$$P_k^+ = (IK_k H)P_K^- \tag{2.15}$$



Figure 2.3: Dijkstra algorithm [1].

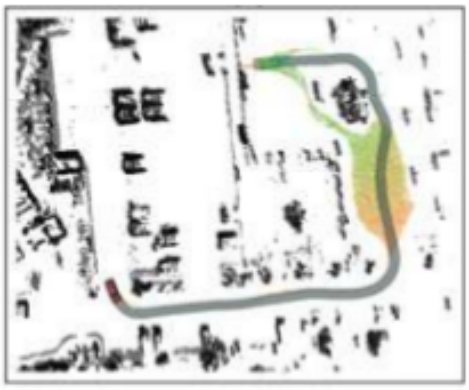


Figure 2.4: A star algorithm [2].

2.3.5 Manhattan distance

The Manhattan distance of two points can be defined as the sum of distance in each dimension in n-dimensional space.

$$d(x,y) = \sum_{i=1}^{n} | x_i - y_i |$$

The Manhattan distance is determined by the coordinate system's rotation, not its reflection around a coordinate axis or translation. All of Hilbert's axioms (a formalization of Euclidean geometry) are satisfied by taxicab geometry, with the exception of the side-angle-side axiom, because two triangles with equally "length" two sides and an identical angle between them are generally not congruent unless the angle is the same. [25]

The applications of the Manhattan distance algorithm are, among others: chess, compressing sensing and in our case, robotics.

Some comparisons have been made with other ways of measuring and scoring distances. Our suggested approach is capable of locating the best point while also exploring a large parameter field. In comparison to Euclidean and Manhattan distances in potential fields, A* distance in influence maps is extremely efficient. In a experiment done in [26] Ms. Pac-Man, Euclidean and Manhattan distances fared similarly, but A* distance outperformed both in terms of score.

CHAPTER 3

SYSTEM DESIGN

When it comes to designing a system, some considerations must be taken into account. Since a robotics system is a very complex graph of different connected nodes, we must avoid a single point of failure in case that, for example, one of the cameras used for locating the robot in the map is failing, the system can rely on the rest of the sensors that keep operating.

A good system design also cares about how it is going to evolve in the future, procuring a basic skeleton that is easy to expand. Some plug-ins are used during the development of this work for that reason.

ROS already provides a decomposition of each of its parts in the shape of "topic" and "node". A topic is a container where odometry or other information from the robot is published. If a node A publish the info in a topic, a node B can subscribe to it and work with that data. For this reason, a framework like ROS comes very handy. We can find some differences depending on the type of system we aim to design.

Next, the design of two autonomous systems is proposed based on the state of the art. In the case of the drone, a system with three symmetrical motors has been chosen, with a vertical axis in the center.

3.1 Drone system design

Following an examination of the many varieties of multirotors and their attributes now available on the market, and taking into consideration the design criteria that we wish to meet, it was determined to offer the following structure: The suggested design combines the traits and benefits of tricopters and hexacopters to create a structure that is appropriate for use.

Two motors on the same axis of each arm rotate in opposite directions, lifting the drone vertically. The increase or decrease of engine rpm is used to control the aircraft's roll or turn.

The requirements for the drone design are as follows:

Requirements
Possibility of carrying a heavier load as compared to the charging capacity of a traditional tricopter or quadcopter.
Flight that is agile and steady.
In the case that one of the thrusts fails, there is redundancy.
Multicopter is more compact than a traditional hexacopter structure.
Possibility of installing a camcorder.
A protective shell is recommended by the structural design.
For the same weight capacity, you don't require as powerful engines or big propellers.
The suggested design allows the electrical components of the airplane (cables, batteries, flight controller, etc.) to be hidden, allowing the original geometry to be perceived, resulting in simplicity and order..
Design that is both innovative and efficient.
Wing design with enough thrust to lift the system.
Telemetry to study flight parameters and allow continuous improvement.
Antenna with at least 5 bands and a minimum range of 50m.

Table 3.1: Drone requirements table

3.2 Four-wheeled vehicle design

Some of the requirements that apply directly to the vehicle design are:

Requirements
Non-holonomic robot: Non-holonomic systems are systems that have constraints that are not integrable into positional constraints.
Scalability, the adaptability and responsiveness of a system with respect to system performance as it significantly increases in size
Security requirements like sensor redundancy to avoid collisions when a sensor fails.

Communication with other nearby vehicles to synchronize
Ability to work offline in the event of a network outage.
High computing power to plan the way and avoid obstacles in real time.
Minimum autonomy of about 200 km which allows the use of the vehicle outside the urban environment.
Ability to switch from manual to automatic mode and back again without the need to stop or slow down.
The system must have the ability to be integrated with visualization systems such as RVIZ which allows adding new improvements and debugging.
New components come on the market every so often. These conform to a standard that must be the same as the one on which the autonomous system is based.
Telemetry must be collected to analyze the data in the event of a failure or inaccuracy.
Being a critical system that operates in real time, different testing phases must be established.

Table 3.2: Four-wheeled vehicle requirements table

This table allows us to get an idea of how the system has to be before starting to implement it. It is not a rigid board, it may be subject to subsequent modifications as long as they conform to industry standards. In chapter 4 the proposed requirements will be implemented

3.3 Flowchart analysis

A flowchart is a diagram that depicts a computer system, method, or procedure. They are frequently used in a variety of areas to document, analyse, plan, develop, and explain often complicated processes in clear, simple diagrams. Rectangles, ovals, diamonds, and a variety of other shapes are used in flowcharts to indicate the kind of step, as well as linking arrows that form flow and sequence. They can be as simple as hand-drawn diagrams or as complex as computer-generated diagrams describing numerous phases and pathways. When we consider all of the many types of flowcharts, we can see that they are one of the most widely used diagrams in the world, utilized by individuals with and without technical skills in a wide range of professions.



FIGURE 3.1. Drone design

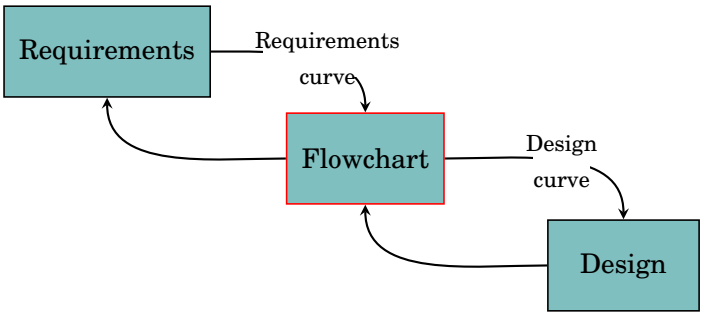


Figure 3.2: The Logical Decomposition phase

3.3.1 Functional design

To know the process that must be implemented in the code later, the flowchart establishes the logic as can be seen in the figure 3.3. It is an introduction of the model at a high level that relate how the **requirements** and the **design** interact together.

3.3.2 Flowchart decomposition

Flowcharts are useful for developing a program or algorithm and communicating it to others or collaborating on it with others since they are a visual depiction of the flow of data. Before you start coding the automated process, you may use a flowchart to describe the rationale behind it in detail. When it's time to code, they may assist create an overview and provide direction. Programmers can frequently write pseudocode, which is a mix of natural language and computer language that can be understood by humans. This can provide more information than a flowchart and can be used as a substitute for the flowchart or as the following step in the code.

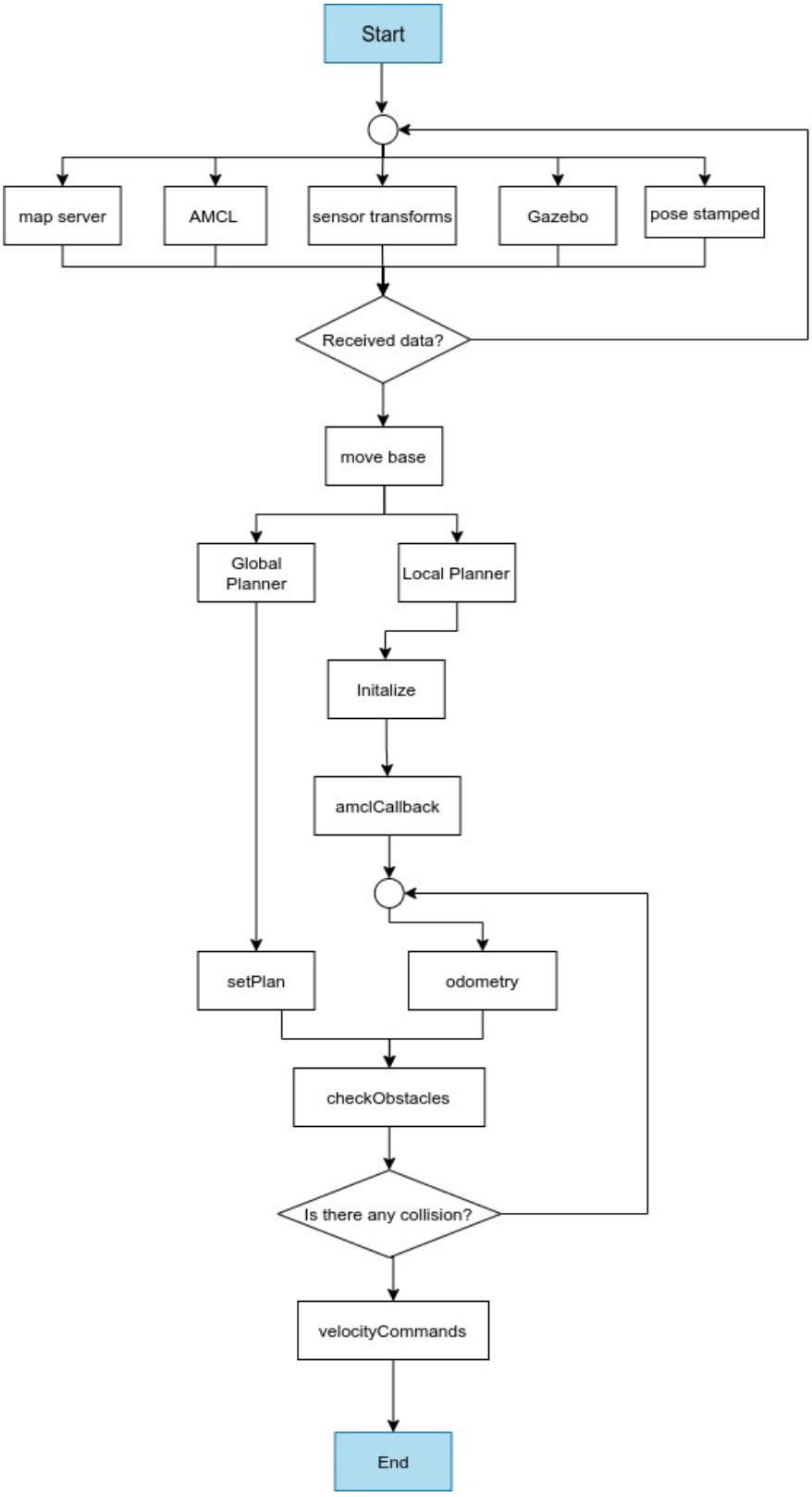


FIGURE 3.3. Main flowchart

3.4 ROS navigation

A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. Our system is taking information from the odometry of the robot, the sensors and the goal that the robot has to reach and then it outputs the velocities the robot needs to get to the goal:

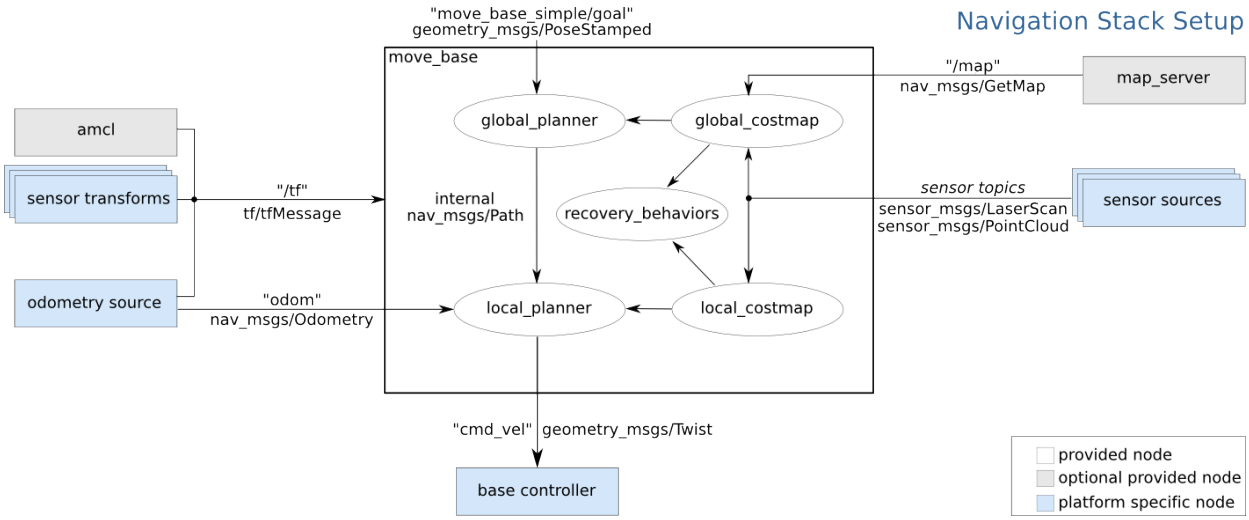


FIGURE 3.4. Graph level scheme

- amcl: As explained in depth later, AMCL is a probabilistic model used to obtain the location of the robot.
- sensor transforms: ROS keeps track of the transforms among the axes along time.
- odometry source: A transform and a nav_msgs/Odometry message over ROS that contains velocity information.
- map_server: Offers map data
- sensor sources: Topics that provides information about the environment and the localization of the robot.

3.4.1 ROS Architecture

Although the acronym "ROS" refers to an Operating System, it is in fact a middleware [1] used to communicate different parts from the robot with a computer. It is a framework that works in real

[1] It is a platform for the exchange of information between different parts of the code

time and allows to manage the dependencies among all of the packages, compile the code (either C++ or Python) and test and visualize the implemented system.

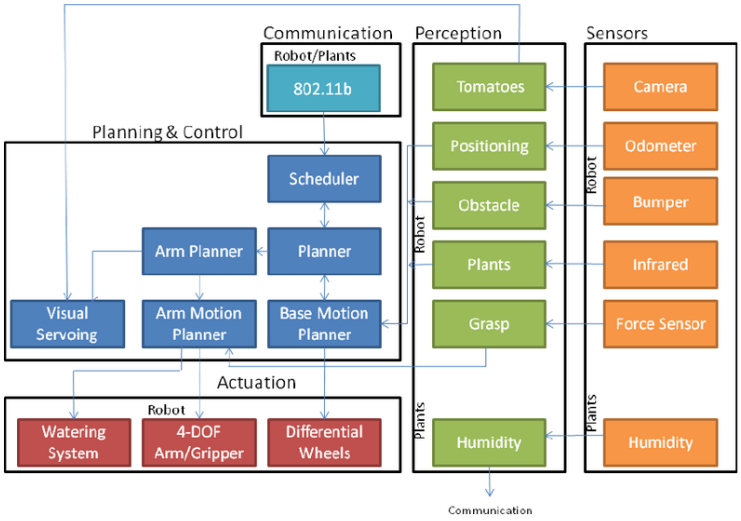


FIGURE 3.5. ROS architecture [27]

One of the main qualities of ROS is that is built on top of DDS. The DDS paradigm focuses on the exchange of data reader-publisher as in ROS a node can publish and can be subscribed from a topic as it can be seen in 3.6

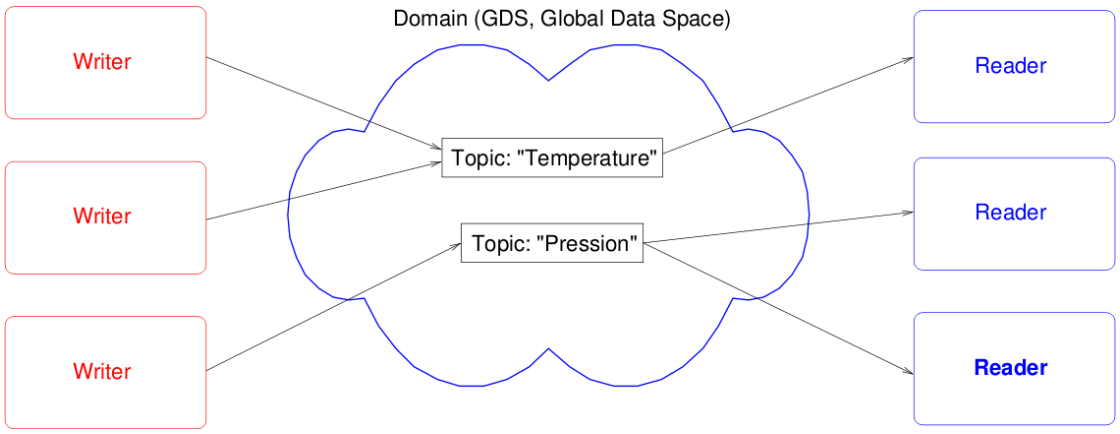


FIGURE 3.6. Data Distribution Service [28]

Another view to abstract the DDS model can be observed in 3.7

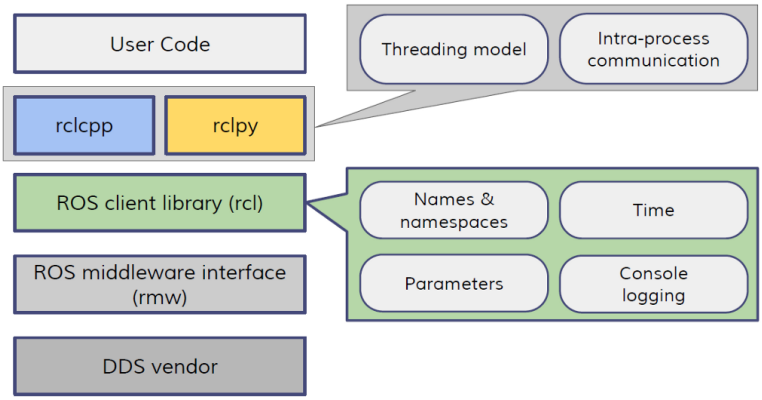


FIGURE 3.7. ROS client library [27]

The most popular ROS client libraries are roscpp and rospy. roscpp allows the implementation of a high computing language like C++. It is the most widely used library and the one who has more internet support. As for using a scripting language like python, rospy is the choice to go. Developers favorite implementations are initializacion and configuration of the code. ROS Master and roslaunch are implementented using this library which puts python in a platform dependency on ROS.

3.5 Path design

This is the scheme I am following to develop the whole system. The system takes the goal in the shape of a pose stamped point. The system also takes into account the shape of the map through a laser and the prior knowledge of the map. Through the AMCL localization system, the position of the robot is deduced. When all of the previous parameters are set, it is time for the system to compute the proper path to follow. The path is discretized in the form of points that must be translated into velocity commands. Also when the point path is provided, a curve that fits those points must be defined.

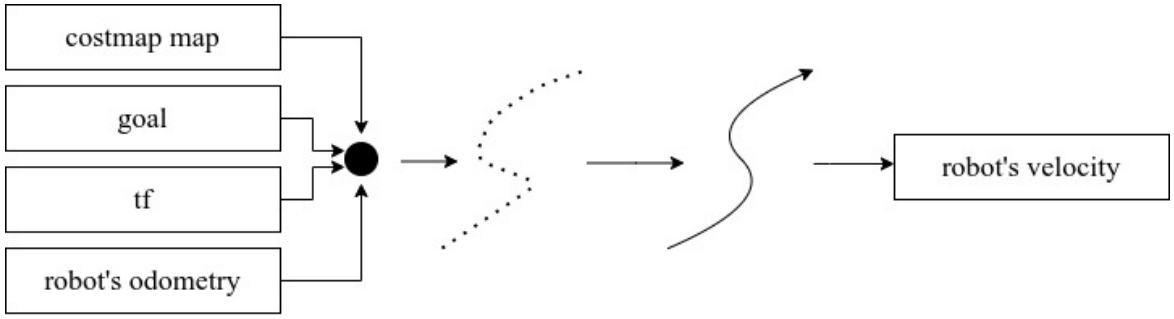


FIGURE 3.8. Graph level scheme

3.6 Adaptive MonteCarlo Localization

AMCL is a probabilistic localization framework for a robot moving in 2D. It uses a filter to locate the robot in a known map. It is based on KLD sampling. The main idea of KLD is to bound the approximation error introduced by the sample-based representation of the particle filter. At each iteration, this approach generates samples until their number is large enough to guarantee that the K-L distance between the maximum likelihood estimate and the underlying posterior does not exceed a prespecified bound. [29]

As shown in the graph 3.9, amcl takes in a laser based map, a laser scan and a tf and it outputs the pose estimate of the robot. At the beginning, it initializes a particle filter depending on how the parameters are set. If there are no parameters set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

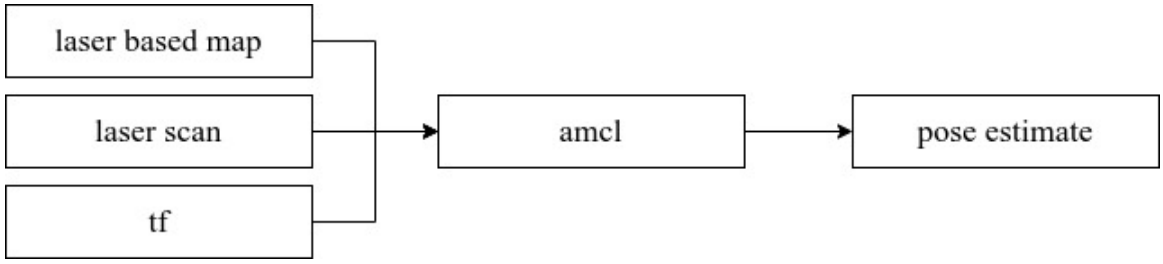


FIGURE 3.9. Graph level scheme

The transform tree is as follows

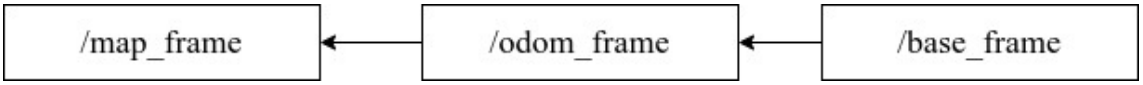


FIGURE 3.10. tf tree

At the point when a robot is needed to perform explicit tasks characterized in the base frame, there is a need for finding the coordinate transform between the kinematic base frame of the robot and the base frame. In this case we have the "base frame" which are the coordinates attached to the base of the robot. Those coordinates reference the movements from the robot as the position and orientation of the joints. The "odom frame" is used to estimate the position of the robot over time in a continuous way [2]. The "map frame", in this case matches the "odom frame".

3.7 tf Package

tf helps to keep track from each of the reference frames. The coordinate system are changing all the time. Transforms within the tf package operate in a similar way as the rest of the topics in ROS. A user can either listen to transforms or broadcast to transforms. In this special case, we are working with 3 different reference frames: map, odom and base link.

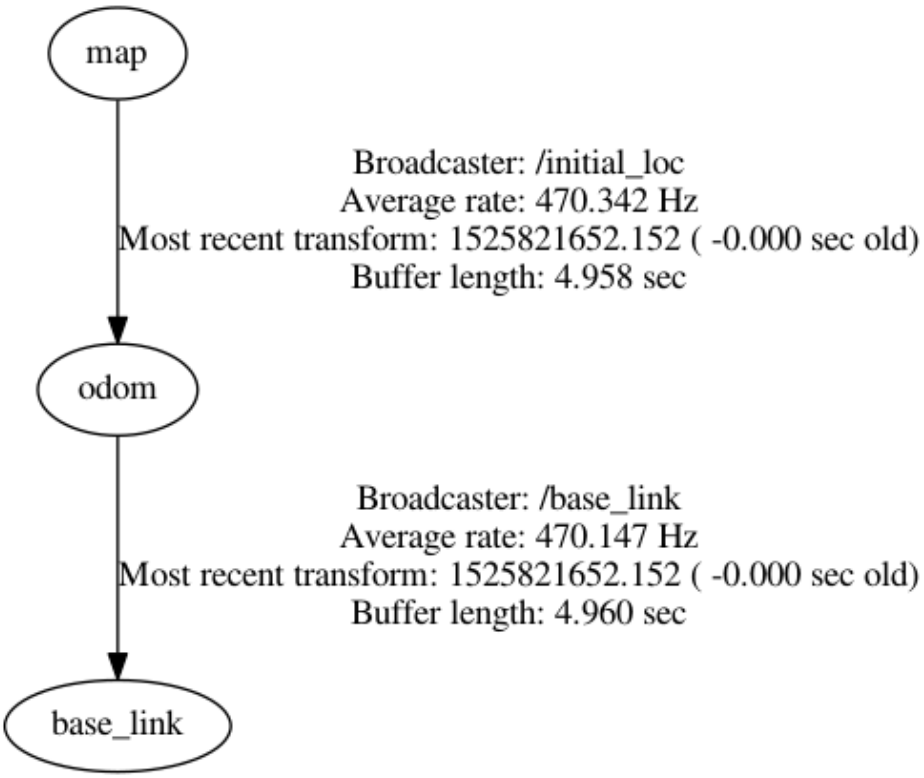


FIGURE 3.11. Tf transform tree

[2]It means, that you always have small errors, but no jumps in position. You can assume that the difference between the current and previous position estimates is approximately the distance travelled

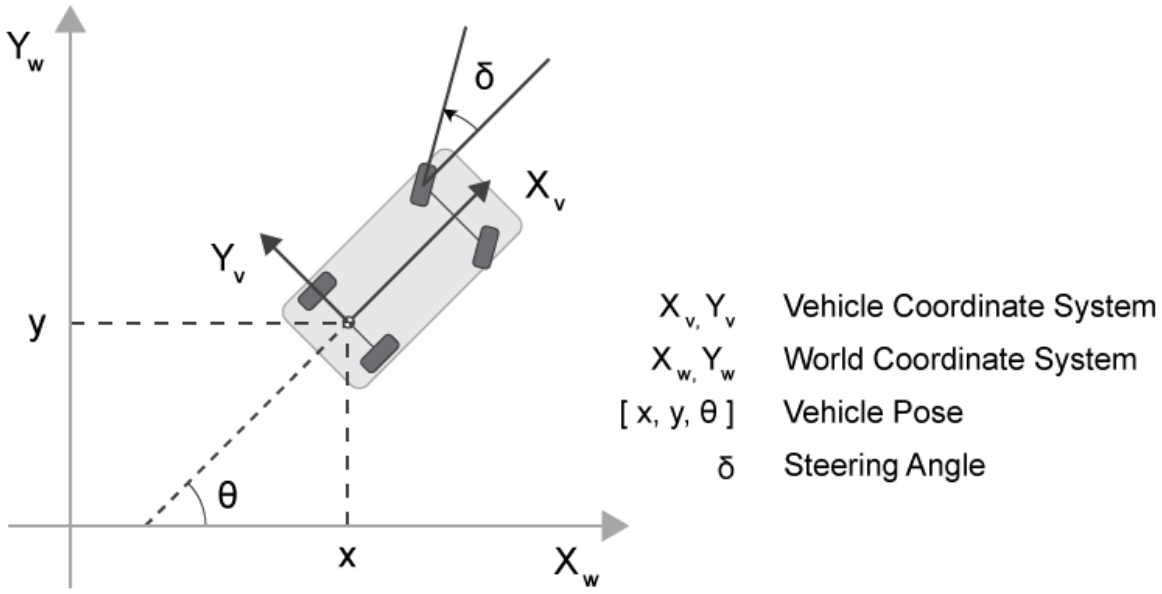


FIGURE 3.12. iCab side view

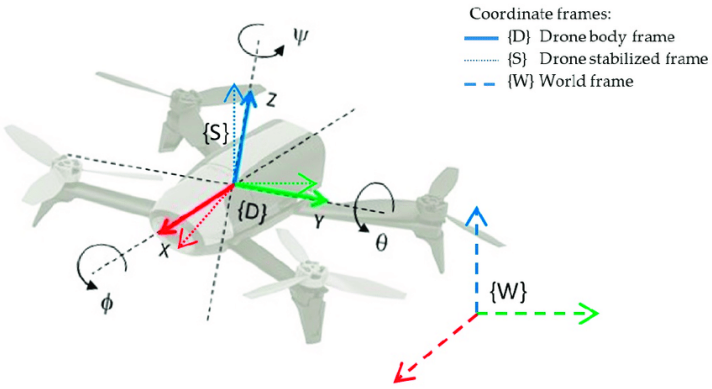


FIGURE 3.13. Drone axes of symmetry [30]

"Coordinate frames D, S and W: drone frame D is attached to the drone's body; stabilized frame S is parallel to the floor; world frame W matches the initial position of the drone". [30]

The planner implements a Dynamic Window Approach. This algorithm predicts the point at which the mobile will find itself if it follows certain given speeds. [31]

The new message I introduce is the cmd_vel message. There exists two types of Base Local Planners: Dynamic Window Approach (DWA) and Trajectory Planners. A grid around the robot is shaped for each iteration. The Global Path given by the Global Planner is given and the mapped into the grid. Those cells that lies under the Global Path and those close to the Goal Point receive

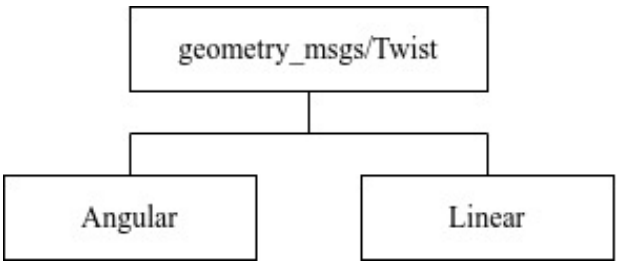


FIGURE 3.14. Twist messages chart

a value of 0. Then, the adjacent cells receive a value based on the Manhattan distance. When the Global Goal lies outside the plan, a Local Goal inside the plan is considered for calculating the Manhattan distance from the adjacent cells.

The Trajectory Planner consists of two main loops:

- scoreTrajectory: It takes in a trajectory object and scores it. A negative value means that the trajectory must be discarded whereas a positive is analyzed through the semantics of the lowest value.

- sampleGenerator: It generates a number of trajectories using either the Trajectory Planner or DWA.

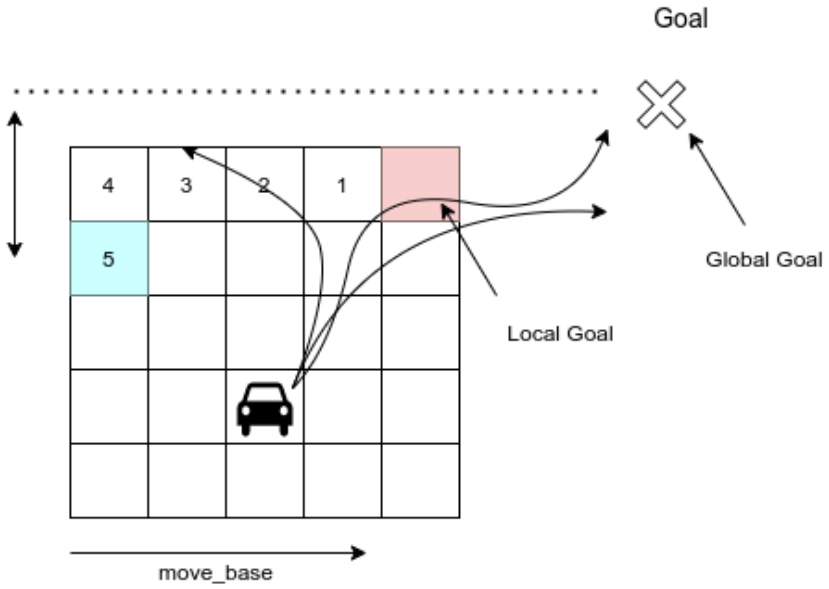


FIGURE 3.15. Map Grid in a Local Planner

3.8 Trajectory Planner in ROS

Once a path to follow is given, the mobile robot needs some kind of controller to adapt its base to it. Around the mechanism, a grid map is formed providing information about the cost of traversing those cells. Here the work from the planner consists of balancing those costs to get a value function by which dx, dy and $d\theta$ are calculated. This planner creates a kinematic trajectory following these steps:

1. Take a sample of the robot velocities
2. Make a simulation to see what would happen if the robot continues in that situation
3. Calculate the cost from each of the interactions based on **proximity to obstacles, to the goal, to the global path and the speed of the robot**
4. Take the best trajectory cost and repeat

During the Trajectory Rollout, the robot samples over the entire period of time.

3.8.1 Consider obstacles when escaping velocities

Most robots have sensors that cover up to 270 degrees. Earlier in time robots possessed a single forward-facing laser and didn't have sensors on their back. That is why checking the costmap when moving backwards was a waste of resources. With modern sensors, that is no longer the case so it is a good approach to add backwards checking.

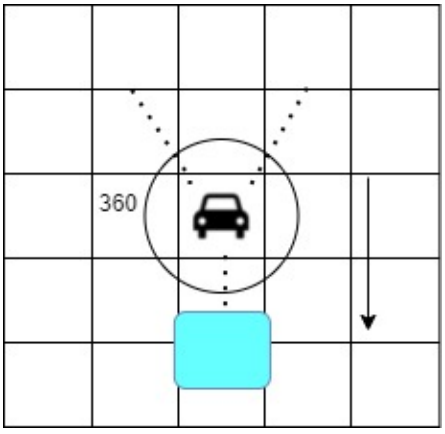


FIGURE 3.16. Obstacles when moving backwards

To find the most suitable path the following loop is taken into account:

1. Check for feasible velocity limits in the space

2. Verify if DWA should be used
3. Sample the velocities regularly
4. Track the trajectory with the best features so far
5. Check that some requirements are accomplished: Holonomic robots, rotatiting in place...
6. Check if after the previous steps we still have a legal trajectory
7. Move backwards slowly if no other action can be takento provide detections.

3.8.2 Cost function to prevent unnecessary spinning

It is relevant to add a cost function to penalize sudden changes in the robot heading. It may not be useful for diff-drive robots where changes in the heading allows it to move towards the end of the trajectory. But you can have the control of the "twirling" for holonomic robots. For doing so, there exists a parameter called "twirling_scale" which tends to make the planner strafe "towards" for higher values.

3.9 Obstacle detection using Supporting Vector Machines

For the design of obstacle detection by computer vision, in this case of other vehicles it has been decided to use **SVM**. Feature extraction, spatial sampling, and classification are examples of independent processing stages in a classic Computer Vision pipeline for object detection. For these steps, several algorithms and techniques can be used, and the settings for each stage are normally fine-tuned by hand. A Deep Neural Network created for object detection, on the other hand, can do similar tasks using a sophisticated interconnected architecture with few separate stages. Lower layers of a neural network, those closest to the input data, often conduct some sort of feature extraction, whereas higher layers can concurrently locate and classify. A graph with the comparison is depicted in 3.17

In the image 3.18 the workflow that has been followed can be observed. During the SVM stage, features are separated on a two-dimensional plane. It is intended that there is the maximum possible separation between the line that separates the features and the features, which is called "margin". The points that relies closer to the separation line are called "vectors". Their use is of vital importance since removing them modifies the separation line position.

Decision trees allow to ask multidimensional questions to split the data. The goal of the decision tree in this case is to maximize the information gain as in the following equation 3.1

$$Informationgain = Entropy(Parent) - WeightedAverageEntropy(Children) \tag{3.1}$$

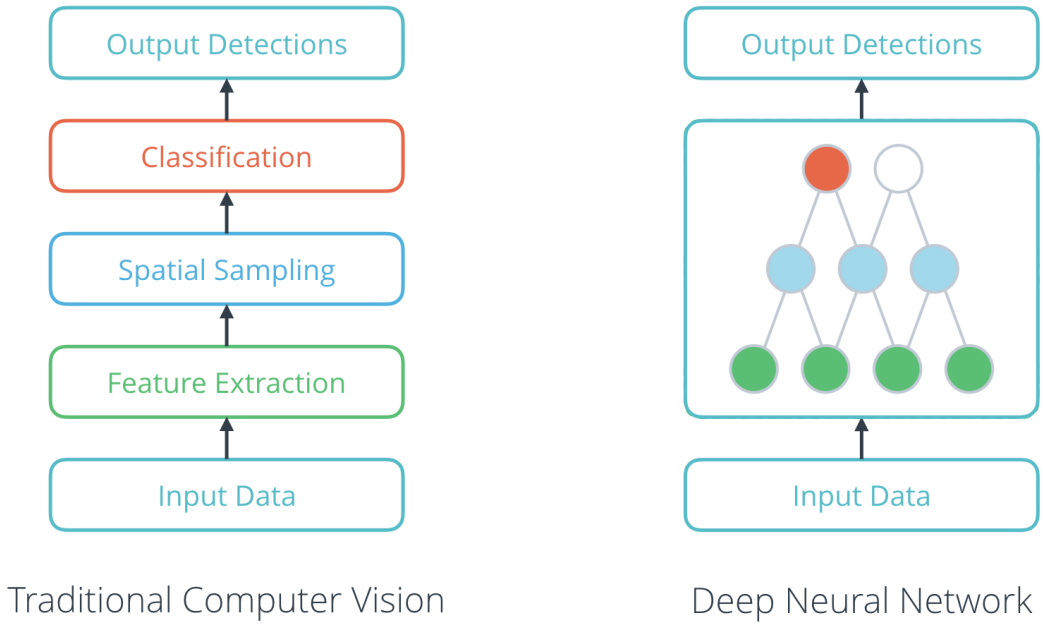


FIGURE 3.17. Computer Vision techniques comparison

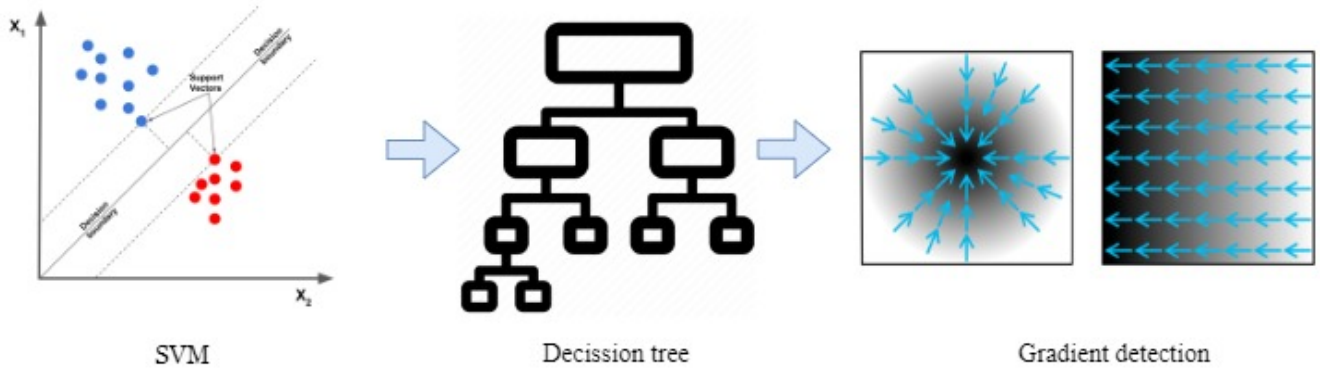


FIGURE 3.18. Vehicle detection flow

This will make the decision about what feature to split on. In the case the feature has some different values as this case, it will figure out how to make the split. Some techniques based on gradients and color have been used.

In every frame of the video, a technique called Sliding Window is used. In each of the positions of the window it will be tracked if there is any obstacle, a car in this case. Sometime the same obstacle may be detected due to overlapping windows and detections. The decision of the detection will be focus on the geometric center. The false positives will be filter out by checking which detections appear in one frame but not in the next. When a high confidence detection is found, it can be tracked by seeing how the centroid is moving.

CHAPTER 4

SYSTEM IMPLEMENTATION

The implementation part has been divided into different stages. For each of the stages a specific programming language has been used. To begin with, Matlab has been used to make a plot of how the algorithm of the Bèzier curves works given n points. Once the tests on the algorithm have been carried out, an efficient language such as C++ has been chosen. The form chosen for the Bèzier method has been the quadratic form and its explained later in 4.1.1

To help people without programming experience in the aforementioned languages, the pseudocode solution has been explained. No specific rule was followed to write the pseudocode

4.1 Development in C++

4.1.1 Bezier function

The following algorithm was used as a recommendation for the first study. To begin, create a function that produces a sequence of factorial values based on the number you provide.

Algorithm 1 Factorial

```
1: procedure C++ PROCEDURE
2:     factorialLookup ← vector from a list of 33 factorial numbers
3:     i ← number to get a factorial from
4: function:
5:     if i < 0 then return error
6:     if i > 32 then return error
7:     else
8:         factorialLookup() ← i.
9:     return factorialNumber.
```

Algorithm 2 Binomial

```
1: procedure C++ PROCEDURE
2:     a1 ← factorial(n)
3:     a2 ← factorial(i)
4:     a3 ← factorial(n-i)
5: function:
6:     Ni ← a1/(a2*a3)
7:     return Ni.
```

It is needed to calculate the Binomial $\binom{n}{i} = \frac{n!}{i!(n-i)!}$
The factorial is calculated from the numerator and denominator numbers using the method above. Now it's time to put those numbers into the binomial equation..

Now that we have the binomial component, we have all of the pieces in place to compute the Bernstein polynomial. Bernstein polynomial has the following form, as shown in the theory:

$$\vec{x}(s) = \sum_{i=1}^{n} \vec{b}_i B_i^n(s) \tag{4.1}$$

The De Casteljau's algorithm for a cuadratic function looks like [32]:

$$P = (1-t)^2 * P_0 + 2 * (1-t) * t * P_1 + t * t * P_2 \tag{4.2}$$

Then now the missing part is $t^i * (1-t)^{(n-i)}$
The factorial algorithm previously defined is the basis for the next Bernstein Polynomial.

Algorithm 3 Bernstein

```
1: procedure C++ PROCEDURE
2:     if t = 0 and i = 0 then return 1
3:     else t^i
4:     if n = i and t = 1 then return 1
5:     else (1 − t)^(n−i)
6: call to Binomial:
7:     binomial() ← i, n.
8:     return (binomial ∗ ti ∗ tni)
```

Then, for the final computation of the Bezier curve, the next function uses the algorithm established before. It determines the points on the curve that define its shape; these points will be utilized to determine how the curve appears later.

Algorithm 4 Bezier

```
1: procedure C++ PROCEDURE
2:     for i1 != cpts do
3:         p[icount] ← 0.0
4:         p[icount+1] ← 0.0
5:         for i != npts do
6:             basis ← Bernstein(npts-1, i, t)
7:             p[icount] ← basis*b[jcount]
8:             p[icount+1] ← basis*b[jcount+1]
9:             jcount ← jcount+2
10:        icount ← icount+2
```

4.1.2 Implementation in ROS

At the start, a Bézier route is provided, which includes an initial and a goal point. Rather of being derived from a topic, the beginning and target points are both defined within the main function. The magnitude of the init and target vectors must be determined ahead of time. The amount of points utilized by ROS to design the route will be the number of points in goal. The map topic is subscribed to by the Bézier node, and the path is published in the map path topic.

In ROS, a sort of function known as a "callback function" exists. Although you define the function, you do not invoke it. Passing a reference to another component of the code that will call your function whenever it is suitable is the most straightforward way of working with callback functions. Because ROS is used to link various components of the system, these routines are frequently used as message handlers. To subscribe to the message, a message handler function is defined. This function is never called by you; instead, it is automatically invoked when a message comes, and the message is handed to it.

There are three simple functions whose sole purpose is to accept and save the pointer's content.: **goalPointCallback, mapCallback and odomCallback**

- goalPointCallback: The objective point is sent to it in the form of a "posestamped" message. It contains the location and orientation of the destination point.

- mapCallback: It gets the map in the form of a "occupancy grid" message, which tells it where to go. This is a two-dimensional grid map in which each cell indicates the likelihood of occupancy.

- odomCallback: It gets the robot's current odometry, in this instance the autonomous vehicle's. The message's type is "odometry." This indicates a free-space location and velocity estimation. The utilization of data from motion sensors to determine change in location over time is referred to as odometry.

With the aid of the Bézier function, the **generate path** function creates the waypoints that define the path. There are two arrays that contain the points that will serve as a requirement to generate the trajectory, "input" and "output". The "posestamped" message stores each of the points created by the Bézier function one by one and then publishes them in two formats: points and paths. The message type is navigation message, which is also known as path message. It has an array of postures as well as a header containing the ID information.

Because the Bézier method only outputs the points and not the orientation that they must follow, this must be computed afterwards.
The following is the method I used to compute the orientation:

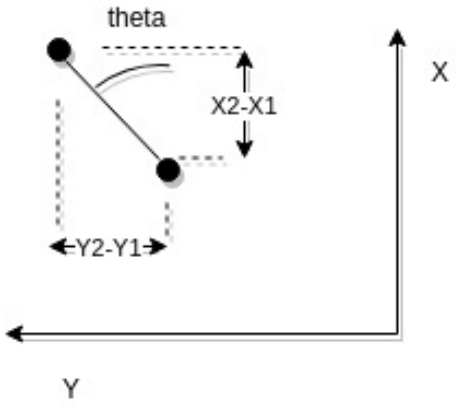


FIGURE 4.1. Orientation calculation.

Two distinct spots may be noticed in the diagram above. To aid visualization, a line has been drawn between them. The hypotenuse may be easily calculated using a square root::

$$hyp = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2} \tag{4.3}$$

After that, use trigonometry to determine the angle that line makes with the X axis.:

$$\theta = arcos(\frac{X2 - X1}{hyp}) \tag{4.4}$$

Using the aid of the quaternions, each of the orientations is transformed into Quaternions. In our case we are only rotating along the **z axis** because the plane where the vehicle is moving relies on the x and y axis, thus the **x** and **y axis** are null:

$x = 0 * sin(\theta * 0.5)$

$y = 0 * cos(\theta * 0.5)$

$z = 1 * sin(\theta * 0.5)$

$w = cos(\theta * 0.5)$

This way the orientations are drawn as follows:

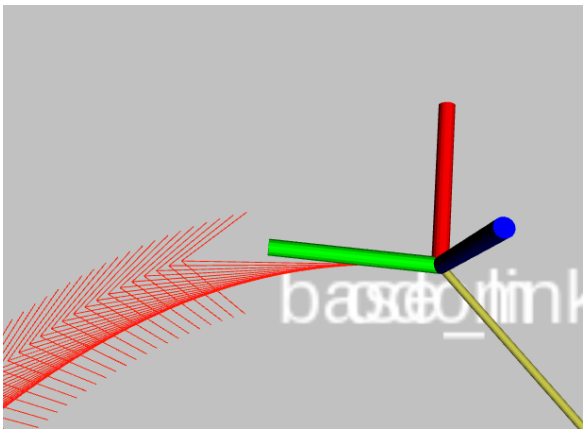


FIGURE 4.2. Orientations in RVIZ.

Each arrow indicates a line tangent to each point on the path, and it should reflect the vehicle's orientation at each step of the journey. Because it is an Ackermann, the vehicle can only adopt specific postures and orientations. [1] steering based robot.

There is also a transform process. The transform is taken from $map \Rightarrow baselink$

[1]The Ackermann model is a geometry for steering. Due to their wheeled structure, these robots are unable to turn in situ.

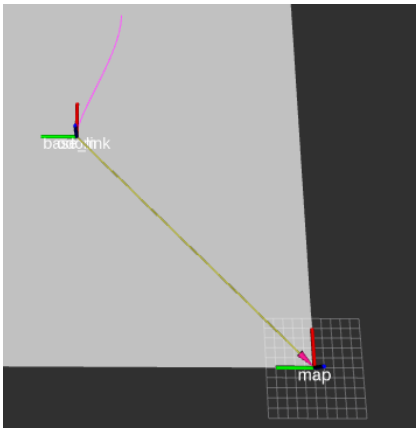


FIGURE 4.3. Transform from map to base link in RVIZ.

As a result, the route begins publishing in the base link. The next algorithm displays the pseudocode that depicts the progression of the calculations.

Algorithm 5 Generate path

```
1:  procedure C++ PROCEDURE
2:      inputs[] ← current_odom, goal_point
3:      outputs[] ← Bezier(inputs)
4:      for i < number_of_points do
5:          pose.pose.position ← outputs
6:          pose_map.pose.position ← pose.pose.position + transform
7:          pose_map.pose.orientation.x ← 0
8:          pose_map.pose.orientation.y ← 0
9:          pose_map.pose.orientation.z ← 1 * sin(0 * 0.5)
10:         pose_map.pose.orientation.w ← cos(0 * 0.5)
11:         path_map ← pose_map
12:     return path_map
```

A different form of communication is known as a "Pose array." It's an array of postures with a global reference header. The behavior is identical to that of the route, except that instead of showing the whole path that the vehicle is likely to take, it shows each of the computed postures, which is why it was so necessary to calculate the orientations beforehand. The code has a similar structure to one of the routes, but it produces a different sort of message.

Algorithm 6 Generate array

```
1:  procedure C++ PROCEDURE
2:      inputs[] ← current_odom, goal_point
3:      outputs[] ← Bezier(inputs)
4:      for i < number_of_points do
5:          pose.pose.position ← outputs
6:          pose_array.pose.position ← pose.pose.position + transform
7:          pose_array.pose.orientation.x ← 0
8:          pose_array.pose.orientation.y ← 0
9:          pose_array.pose.orientation.z ← 1 * sin(θ * 0.5)
10:         pose_array.pose.orientation.w ← cos(θ * 0.5)
11:         path_array ← pose_map
12:     return array_map
```

Once the path has been determined, obstacles must be detected in order to avoid them. Then you must build a link between the points on the path and those on the map. The square to which a coordinate belongs is computed using the map resolution. It returns the square's index. Following the figure 4.4 the explanation is easier to follow.

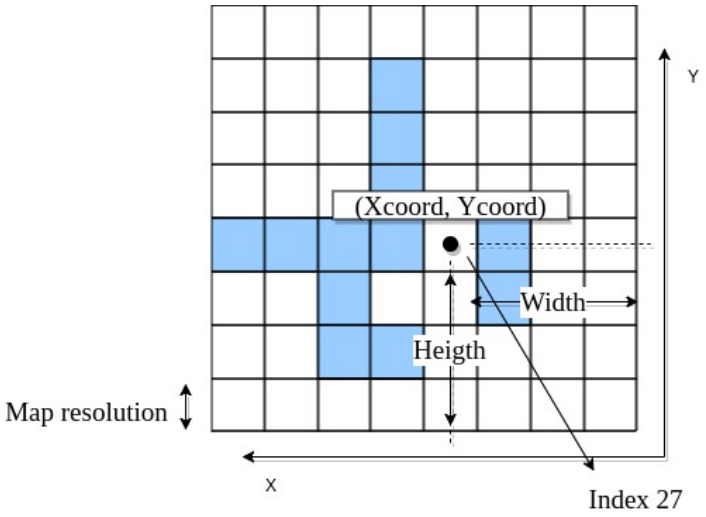


FIGURE 4.4. Parametrization into map grid.

Given the one-dimensional nature of the map message, the map grid must be seen as a one-dimensional array as well. The index from the map grid array is the total of all the squares till you reach the location.

(4.5) $$height = y_coordinate/map_resolution$$

$$width = x_coordinate/map_resolution \tag{4.6}$$

$$index = map_width * height + width \tag{4.7}$$

Once the index of a certain place on the path has been discovered, it must be verified to see if there is a barrier in the way. If there is, an iteration is supplied along the perpendicular from the line that connects the previous and next points. After that, the increment is added or removed from the perpendicular. In the figure 4.9 When the center point collides with an obstruction, it is moved along the perpendicular to a location where there are no obstacles.

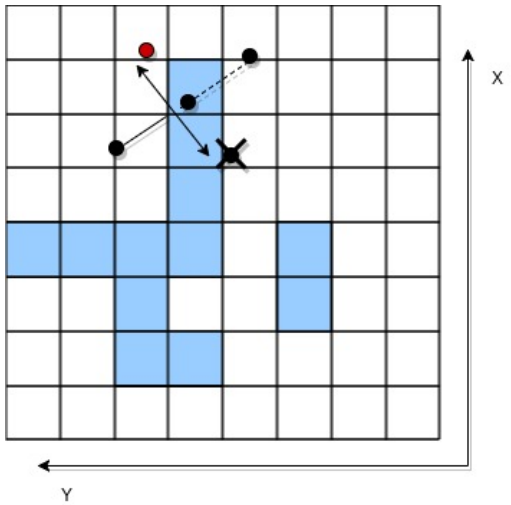


FIGURE 4.5. Obstacle avoidance.

To calculate the localization of the new point it is needed again to obtain the angle θ. The image 4.7 shows where to find that angle in the new situation.

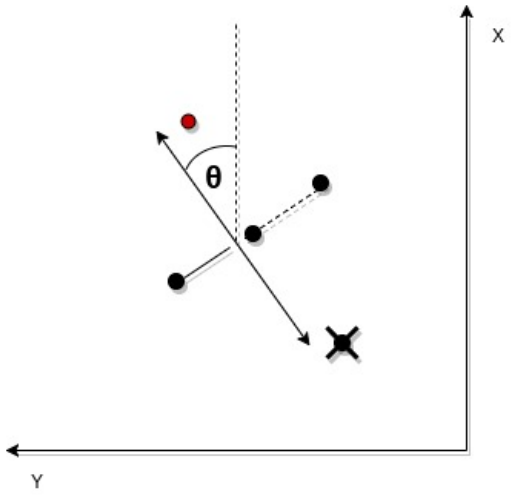


FIGURE 4.6. Obstacle avoidance abstraction.

$$tan(\theta) = \frac{y_coordinate2 - y_coordinate1}{x_coordinate2 - x_coordinate1} \tag{4.8}$$

$$pose_x = cos(\theta) * increment + \frac{(x_coordinate2 - x_coordinate1)}{2} + x_coordinate1 \tag{4.9}$$

$$pose_y = sin(\theta) * increment + \frac{(y_coordinate2 - y_coordinate1)}{2} + y_coordinate1 \tag{4.10}$$

It is feasible to find the grid cell where that point is located using the angle calculated using trigonometry. The calculus of the index is straightforward when θ is used. To get the mediatrix, take both the x and y coordinates from each of the two neighboring locations. Then, in both locations, expand the increment along it. They can be parametrized since the number of points that make up the curve isn't a set number. This value will determine the curve's accuracy as well as the computing cost of creating each route. Keep in mind that this method is designed to function on a real-time system, thus computing efficiency is an important consideration.

Algorithm 7 Check obstacles

```
1: procedure C++ PROCEDURE
2:     increment ← 20
3:     toggle ← 1
4:     for i < number_of_points do
5:         x_coordinate ← trajectory.position.x
6:         y_coordinate ← trajectory.position.y
7:         height ← y_coordinate / map.resolution
8:         width ← x_coordinate / map.resolution
9:         index ← map.width * height + width
10:        if map.data[index] ≠ 0 then
11:            x_coordinate_1 ← trajectory[i-1].position.x
12:            x_coordinate_2 ← trajectory[i+1].position.x
13:            y_coordinate_1 ← trajectory[i-1].position.y
14:            y_coordinate_2 ← trajectory[i+1].position.y
15:            tangent ← (y_coordinate_2 - y_coordinate_1) / x_coordinate_2 - x_coordinate_1
16:            θ = arctan(tangent)
17:            do
18:                aux_pose.position.x ← toggle*cos(theta)*increment + (x_coordinate2 -x_coordinate1)/2 + x
19:                aux_pose.position.y ← toggle*sin(theta)*increment + (y_coordinate2 -y_coordinate1)/2 + y
20:                height ← aux_pose.position.y / map.resolution
21:                width ← aux_pose.position.x / map.resolution
22:                index ← map.width * height + width
23:                toggle ← toggle * -1
24:            while map.data[index] ≠ 0
25:        return true
```

The **generate path** route function is then used to reshape the curve after a new point that avoids the barrier has been established. It is necessary to disclose the appropriate path once it has been discovered. Taking the advantage of a ROS Publisher [2] Once the publisher's type is defined, the message is taken outside the node and published on a topic to which the launch file will subsequently be subscribed. Because there are three distinct sorts of messages that need to be published, three separate types of publishers need to be specified.

The **launch file** instructions describe not only the transformations between reference frames, but also publish the map in the **maps folder** and order the launch of RVIZ with the settings saved in the **global plan** file. Once in RVIZ, the path's points may be shown in a visual format, making it easier for people to comprehend its meaning. The path is represented as a single line, the pose arrays as a collection of locations with a specified orientation, and the points as markers that define the route.

[2] A Publisher is a tool from ROS that allows publishing a message from a Node to a Topic. Before it obtains the message to publish, the publisher's type must be established

4.1.3 AMCL algorithm

For the location of the robot on the map, the possibility of using different algorithms has been considered, among them the Kalman filter. One of the properties of the Kalman filter is that it is parametric which means that it is based on Gaussian distributions. In the case that, for example, we are on a path in which all obstacles have a similar shape, for example streetlights based on the same model, there will be several peaks in the distribution which can make the robot lose itself in the case of use a Kalman filter.

The AMCL algorithm the system is applying is based on KLD-sampling. The equation looks like:

$$s_{t-1} = \{X_{t-1}, W_{t-1} | i = 1....n\}\ representing\ belief\ Bel(x_{t-1}) \tag{4.11}$$

Where the control measurement is u_t, observation z_t, bounds ϵ and δ.

And then sample an index j(n) from the discrete distribution given by the weights in S_{t-1}. Sample x_t from $p(x_t|x_{t-1}, u_{t-1})$ using x_{t-1} y u_{t-1}

Algorithm 1 KLD sampling

1: $w_t = p(z_t, x_t)$;
2: $\alpha = \alpha + w_t$
3: $S_t = S_t U \{x_t, w_t\}$
4: **if** $x(t)$ falls into empty bin b **then**
5: k = k+1
6: b = non-empty
7: n = n+1
8: while $\left(n < 1/_{2t} X^2_{k-1,1-\delta}\right)$
9: **for** i=1....n **do**
10: $w_t = w_t$ / α
11: **return** S_t

The objective of this algorithm is to bound the error, the distance between MLE [3] and the true posterior making them not exceed the value of Epsilon.

The main idea of the KLD-examining approach is to bound the error presented by the example based conviction portrayal of the molecule channel. At every cycle, this methodology produces tests until their number is adequately huge to ensure that the K-L distance between the most extreme probability gauge and the hidden back doesn't surpass a preindicated bound. In this way, the methodology picks few examples if the thickness is zeroed in on a little subspace of the state space, and picks an enormous number of tests if the tests need to cover a significant piece of the state space.

[3] Maximum Likelihood Estimation

The AMCL algorithm is collected in a callback function within the code. It provides a path to the system discretized into poses with covariance (positions and orientations with timestamps).

rqt_graph is a tool used for visualizing the ROS computation graph. It makes it easy to have an overview of the entire system. In the end, what the previous graph shows us is something that we can already deduce with the information we have. Move base is subscribed to the costmap of the map, a simple plan and a goal. From this information all the information on the speed of the wheels is published in cmd_vel and then it is passed to gazebo. Through the signals received by the Hokuyo scan, thanks to AMCL the robot can be located in the map.

4.1.4 Considering obstacles when escaping trajectories

During **findBestPath** a trajectory to follow is returned. With that goal, different trajectories are created inside **createTrajectories**. Using **checkTrajectory** and **scoreTrajectory**, generated trajectories can be evaluated. Finally the real trajectory is created with **generateTrajectory** which initiates a single trajectory.

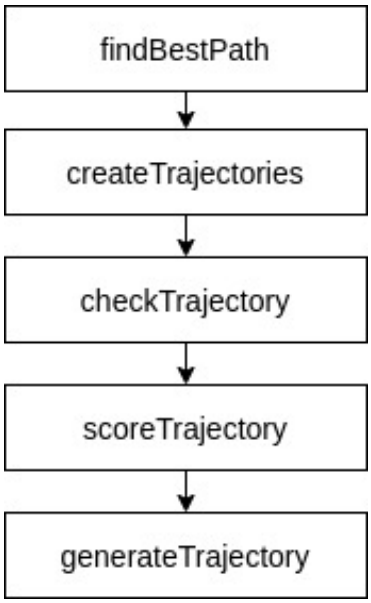


FIGURE 4.8. Obstacle avoidance abstraction.

As for start considering obstacles when escaping, a new algorithm has been implemented.

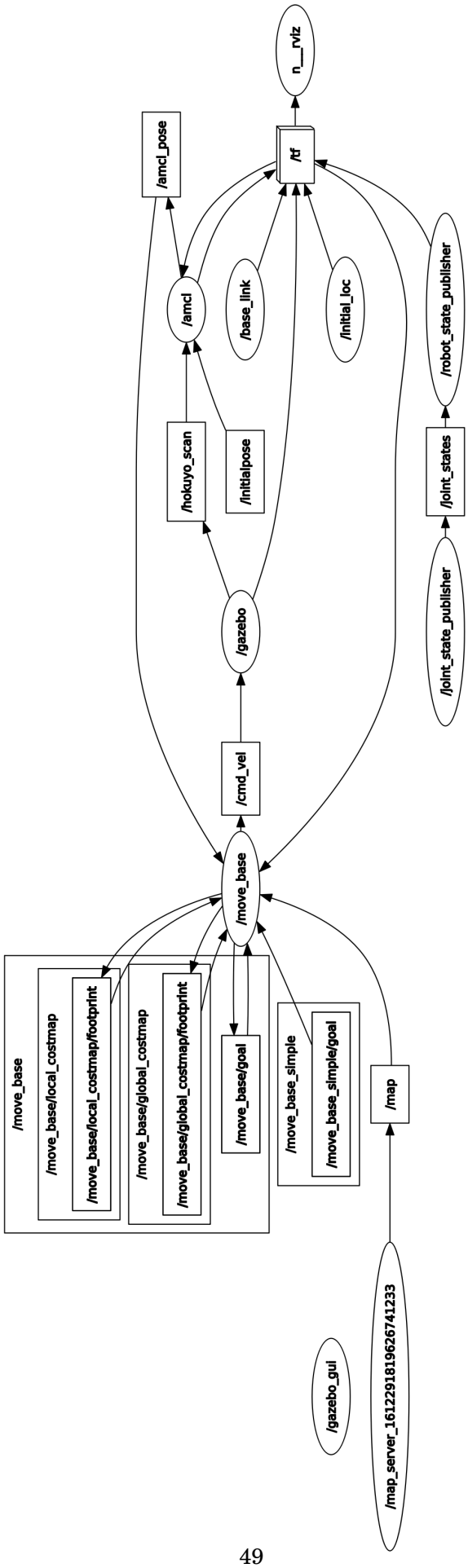


FIGURE 4.7. Obstacle avoidance abstraction.

Algorithm 2 Escape algorithm

1: $swapTrajectory = compTrajectory$
2: **for** i<swapTrajectory.size() **do**
3: $swapTraj->getPoint(i, x_r, y_r, th_r)$
4: $costmap.worldToMap(x_r, y_r, cell_x, cell_y)$
5: **if** $cost < 0.5$ **then**
6: compTraj->setPoint(i, x_r, y_r, th_r)

In this way, it is ensured that no obstacle blocks the escape and that it is possible to proceed without danger.

4.1.5 Cost function to penalize robot spinning

A new cost function has been implemented. A new file *twirling_cost_function* has been created for that purpose. Also, a new *twirling_scale* factor has been included inside DWA Planner's configuration so that it acts as a weight for penalising the robot's spinning.

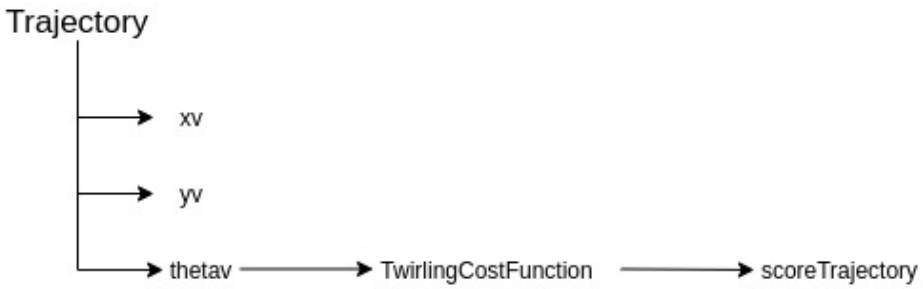


FIGURE 4.9. Class diagram

4.1.6 Obstacle detection using SVM

I started by loading all vehicle and non-vehicle image paths from the provided dataset. The following figure shows a random sample of images from both classes in the data set:



FIGURE 4.10. Data showcase

The extract_features method in the section titled *Method for extracting HOG features from an array of car and non-car images* accepts a list of image paths and HOG parameters (as well as one of a variety of color spaces from destination, to which the image is entered is converted) and produces a flattened matrix of HOG functions for each image in the list.

I then create parameters for HOG feature extraction and extraction features for the whole data set in the part named *Extract features for input data sets* and combine, define vector labels, mix and split. A label vector is defined by combining these sets of attributes (1 for automobiles, 0 for non-automobile vehicles)

In order to input into a linear support vector machine (SVM) classifier, features and labels are shuffled and split into training and test sets. The table below lists the twenty-five distinct parameter combinations that I experimented with.

Configuration Label	Colorspace	Orientations	Pixels Per Cell	Cells Per Block	HOG Channel	Extract Time
1	RGB	9	8	2	ALL	71.16
2	HSV	9	8	2	1	43.74
3	HSV	9	8	2	2	36.35

FIGURE 4.11. Config Parameters

I made my final HOG parameter selection based on the performance of the SVM classifier I created with them. I took into account not only the precision with which the classifier predicted on the test data set, but also the speed with which it could predict. The classifier must strike a compromise between accuracy and speed, and my method was to prioritize speed first, getting predictions as near to real time as feasible, and then aim for precision if the detection stream failed.

The final settings were those shown in the table above as setting 24: YUV color space, 11 orientations, 16 pixels per cell, 2 cells per block, and ALL color space channels. The following table summarizes the performance of the classifier for each of the settings in the table above:

Configuration (above)	Classifier	Accuracy	Train Time
1	Linear SVC	97.52	19.21
2	Linear SVC	91.92	5.53
3	Linear SVC	96.09	4.29
4	Linear SVC	95.72	4.33

FIGURE 4.12. Accuracy

In the section titled *Train a Classifier*, I was able to obtain test precision 98.17 table: I trained a linear SVM with the default classifier settings and only the HOG functions (I did not utilize the channel or spatial intensity histogram features).

I used the find cars method in the part titled "How to use the classifier to recognize cars in a picture." The technique combines HOG feature extraction and a sliding window search, however rather than extracting HOG features for each window separately, which may be time intensive, HOG features are extracted for the whole picture (or a selected part her). The classifier receives image characteristics that have been subsampled according to the window size. The technique applies the classifier's prediction to the HOG features for each window region and produces a list of rectangular objects corresponding to the windows with a positive prediction (car).

The first effort to utilize find cars in one of the test pictures, using a single window size, is seen in the following image:



FIGURE 4.13. First effort

A combined heatmap and threshold is utilized to distinguish the two because genuine positives are usually accompanied by numerous positive detections, whilst false positives are usually accompanied by only one or two detections. At the position of each detection rectangle, the add heat function increases the pixel value (called "heat") of a completely black picture to the size

of the original image. Heat is allocated to regions that are encircled by multiple overlapping rectangles. The heat map created by the detections in the previous image is shown below:

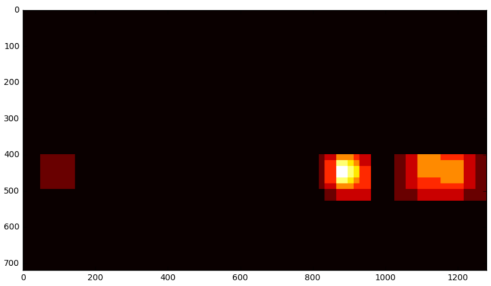


FIGURE 4.14. Heat map

A threshold is applied to the heat map (in this example, with a value of 1), setting all pixels that do not exceed the threshold to zero.

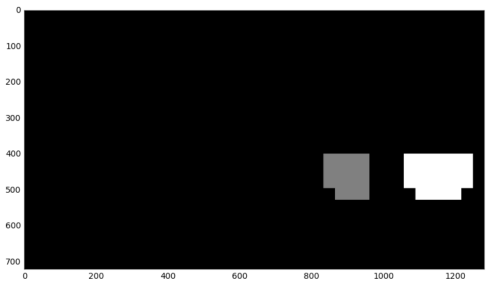


FIGURE 4.15. Heat map threshold

And the final detection area is set at the ends of each identified tag:



FIGURE 4.16. Final detection

The final implementation works very well, identifying the near field vehicles in each of the images with no false positives.

The first implementation didn't work that well, so I started by optimizing the SVM classifier. The original classifier used HOG characteristics from the YUV Y channel only, and achieved a

96.28 % test accuracy. The use of the three YUV channels increased the accuracy to 98.40 %, but also tripled the execution time. However, changing the pixels_per_cell parameter from 8 to 16 produced an approximately ten-fold increase in execution speed with minimal cost to accuracy.

Other optimization techniques included changes in window size and overlap as described above, and lowering the heatmap threshold to improve detection accuracy (higher threshold values tend to underestimate vehicle size).

CHAPTER 5

TESTING AND RESULTS

The results of the design described in the previous chapter are addressed in this section. All of the tests are presented by RVIZ, and they attempt to clarify the explanations shown throughout the project. A schematic figure has been supplied in certain situations to depict what is hoped to see in RVIZ. The reference frame for each graph has been described as the several points that make up the trajectories.

5.1 Gazebo

As not real robot is used, I take advantage of a virtual simulation environment named "Gazebo".

The Gazebo ROS package tightly integrates ROS and Gazebo. This package contains a Gazebo plugin module that allows Gazebo and ROS to communicate in both directions. Gazebo can send simulated sensor and physics data to ROS, and ROS can send actuator commands back to Gazebo. In fact, by using consistent names and data types for these data streams, Gazebo can match the ROS API of a robot to a tee. [33] The algorithm provided during this work has been developed using a specific robot, the **Pioneer 3-AT**.



FIGURE 5.1. Robot in isometric perspective

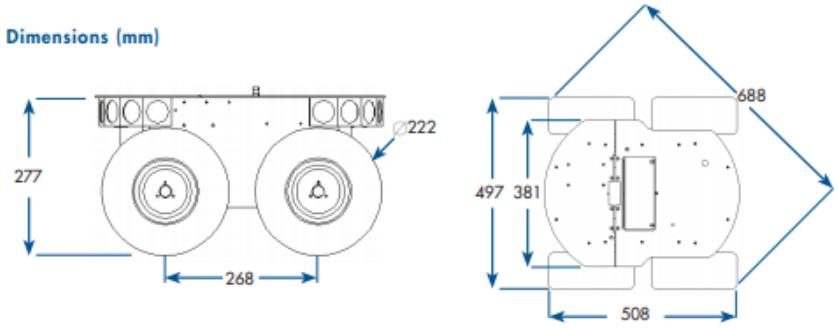


FIGURE 5.2. Dimensions of P3AT robot

Pioneer 3-AT is a small four-wheel, four-motor skid-steer robot ideal for all-terrain operation or laboratory experimentation. The Pioneer 3-AT comes complete with one battery, emergency stop switch, wheel encoders and a microcontroller with ARCOS firmware, as well as Pioneer SDK advanced mobile robotics software development package.

This is an overview of the Gazebo messages and the Gazebo architecture:

5.2 Adaptive Monte Carlo Localization and Maps

To place the robot on the map, help is needed from the AMCL. With the aid of the tool **Gmapping** the stetup showed in the image from the section above can be mapped into a map scheme as in 5.3.

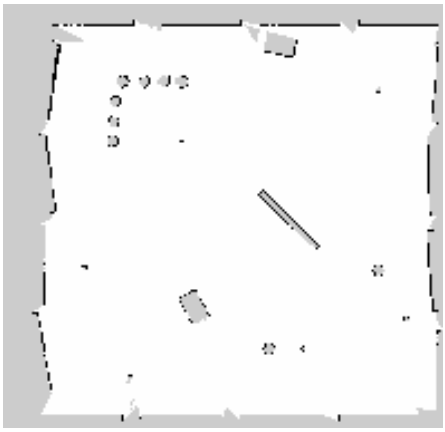


FIGURE 5.3. Gmapping map scheme

The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot. [34]

The Hokuyo URG-04LX-UG01 laser is an ideal 2D laser scanner for robotics applications with a great value for money. It is very useful to see where the obstacles are located.

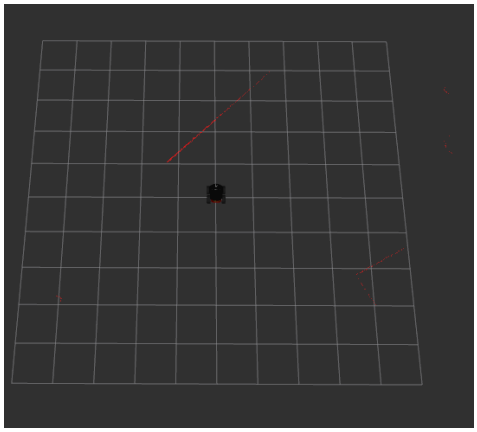


FIGURE 5.4. Hokuyo laser scan

Through the use of the laser, a laser based map is created as shown in 5.20. This map is transmitted to the AMCL system to get the localization estimate from the robot.

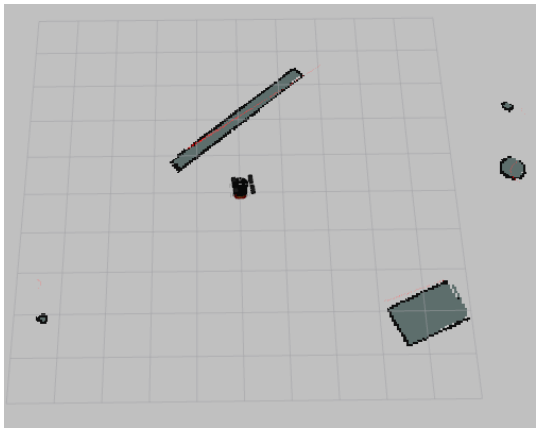


FIGURE 5.5. Hokuyo laser and map

As you can see the laser measurements match the shape of the obstacles.

5.3 Bézier curves in ROS

The main issue when drawing Bézier curves in ROS was transforming the points into ROS messages. As a test a simple curve has been drawn from the origin reference frame, in this case, the "map" reference frame. The curve is defined by 2048 points, the point in the origin is (0,0) and the goal point is variable. As it is shown in figure 5.6 the goal point orientation is perpendicular to the horizontal thus, the orientation of the vehicle stays the same in Figure 5.6.

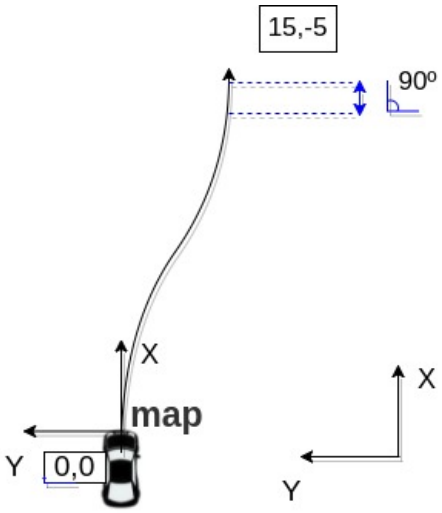


Figure 5.6: Bezier curve from map.

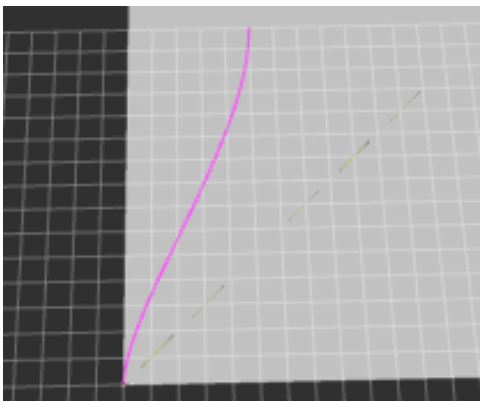


Figure 5.7: Bezier curve in RVIZ

5.3.1 Transform the curve from map to odom

The curve has been converted from the map reference frame to the odom reference frame using the transform module from ROS. The code has not been altered in any way, including the locations and orientations. The previous location has been updated with the transform matrix. The number of points that defines the curve as the goal positions and orientations are the same as in section 5.3. To see it more clearly the scheme in figure 5.8

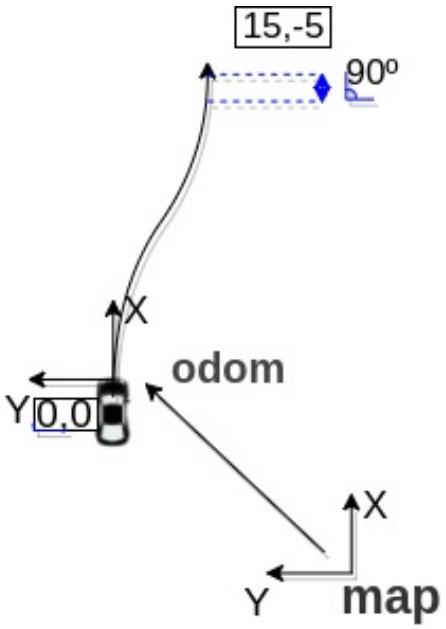


Figure 5.8: Bezier transform to odom.

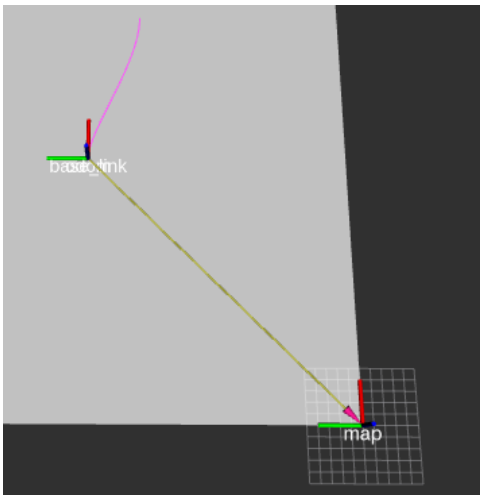


Figure 5.9: Bezier transform in RVIZ

5.3.2 Behavior of the routes depending on different orientations and positions

In the figure 5.10 s series of routes is laid out to demonstrate how varied target point orientations shape the entire journey. Three distinct pathways with various objective points are presented on the left side of the RVIZ view. Because those pathways are made up of points with varying locations and orientations, the pictures include a zoom to the origin on the right side, figure 5.11 where each of the points can be noted.

The path can be observed form RVIZ in the next images, the initial and target position and the initial and target orientation have been tuned as showed in Table 5.1.



Figure 5.10: Set of different paths with different orientations.

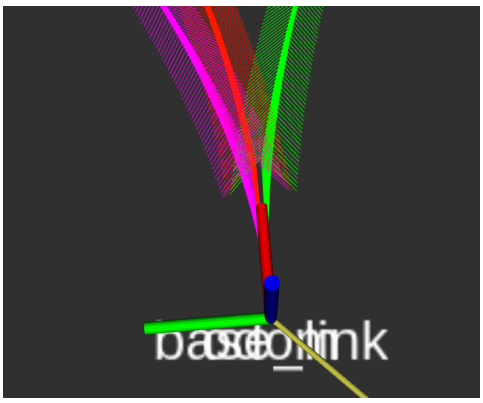
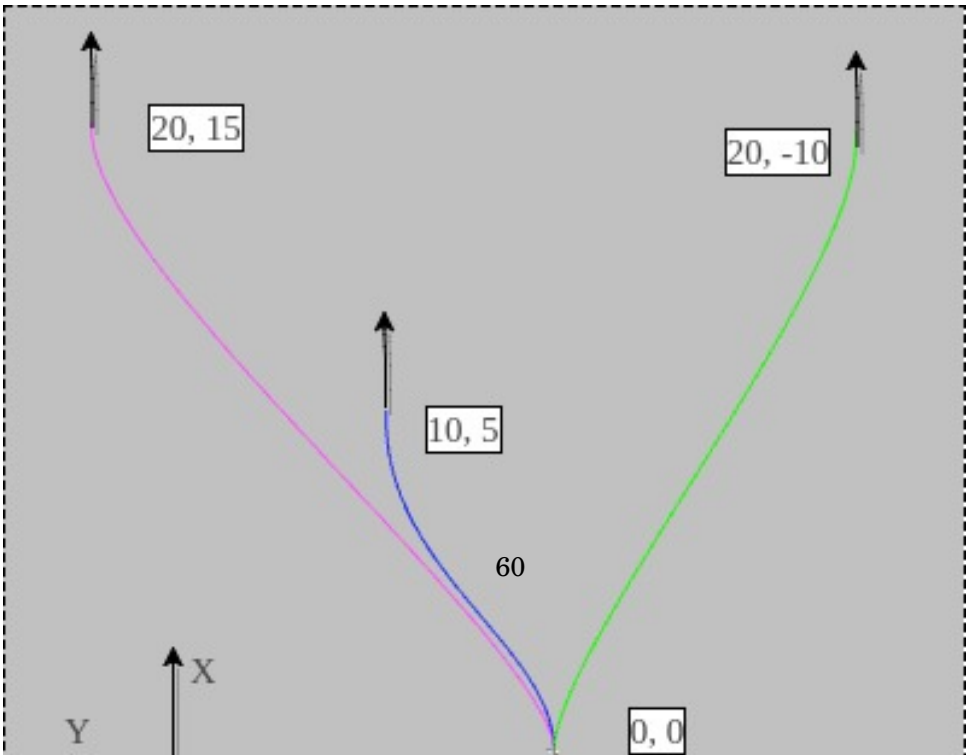


Figure 5.11: Set of poses from each path. This view comes from a "PoseArray" message.

Group	Initial orientation	Final orientation	Final position
A (pink)	90º	90º	(20, 15)
A (blue)	90º	90º	(10, 5)
A (green)	90º	90º	(20, -10)
B (pink)	90º	45º	(20, 15)
B (blue)	90º	45º	(10, 5)
B (green)	90º	45º	(20, -10)
C (pink)	135º	45º	(20, 15)
C (blue)	135º	45º	(10, 5)
C (green)	135º	45º	(20, -10)

Table 5.1: Parameters from each curve



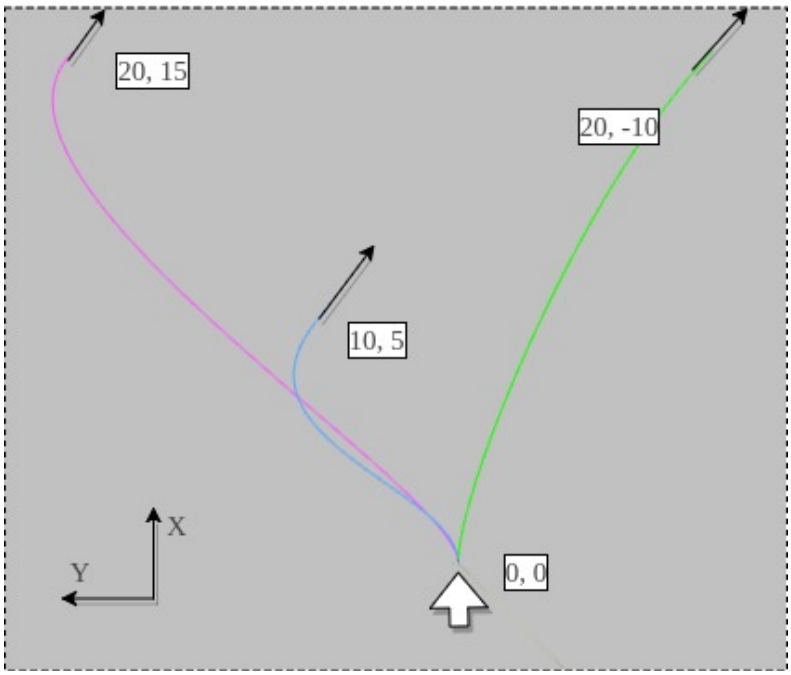


FIGURE 5.13. Group B curves in RVIZ.

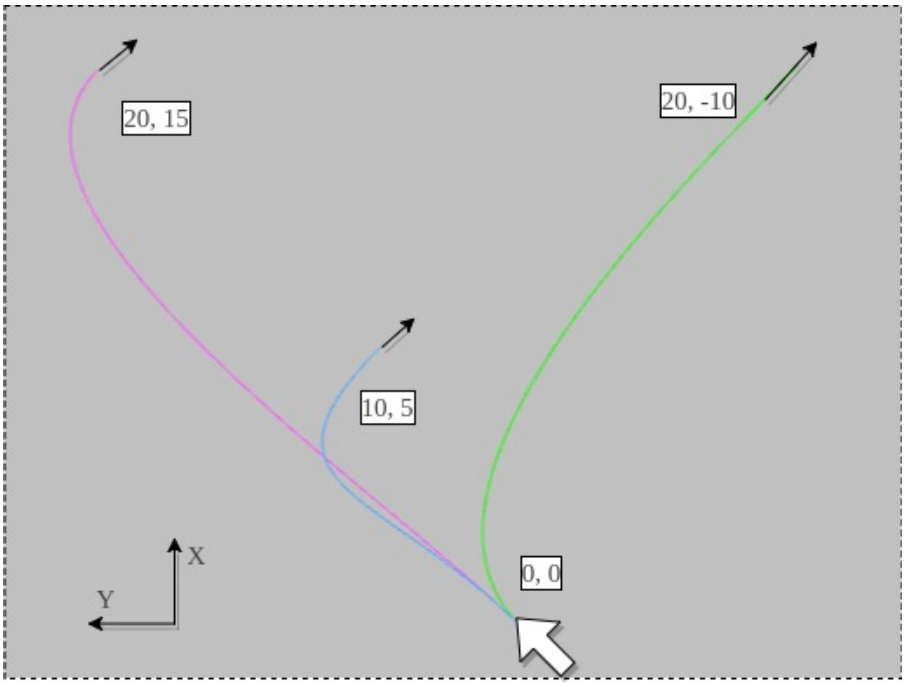


FIGURE 5.14. Group C curves in RVIZ.

5.4 Avoiding obstacles in ROS

The algorithm begins scanning the map in the x coordinate (upwards with respect to image 5.15) as if the map consists of a one-dimensional array as explained in Chapter 4. It parametrizes each of the produced path's locations in order to compare them to the map grid. When the "Obstacles Detection" function encounters an obstacle pixel, it attempts to locate the first hole inside the perpendicular to the obstruction, first in the left sense and then in the right. When it discovers an empty hole, it uses that location as an initial parameter requirement and recalculates the path.

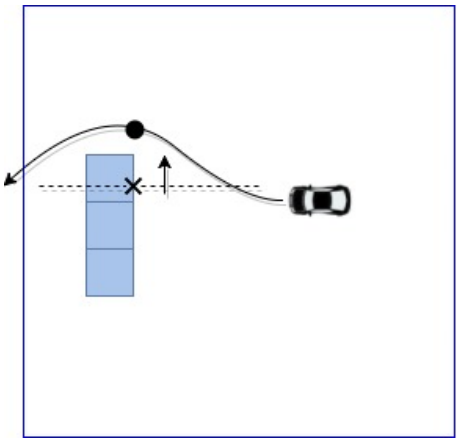


Figure 5.15: One obstacle avoidance abstraction

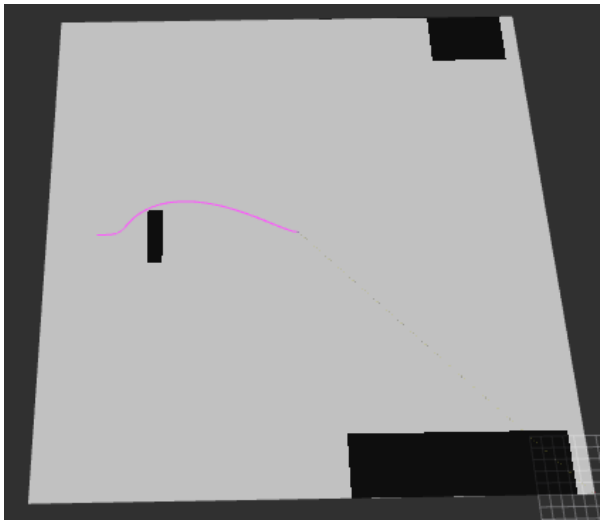


Figure 5.16: One obstacle avoidance in RVIZ

One of the issues that might emerge when using the avoidance algorithm is that the untreated holes get trapped at the end of the route, causing the curve to take on a non-smooth form in the last portion. In this scenario, the iCab [1] has a fixed minimum radius which limits the path to only work with smooth ways. Furthermore, because the iCab cannot be considered as a line-wide object, there should be a tolerable range around the obstructions. The automobile travels extremely near to the top obstacle corner, as can be seen in Figure 5.16. Even though the path does not appear to do so, depending on the proportions of the vehicle, it may create a collision with the barrier.

A map with two obstacles blocking the route is assessed as a more complicated proposal. In this scenario, there are two more vacant spots, or holes, for which memory must be allocated. Once the obstacle's location has been determined, the algorithm begins to iterate locations until no impediments are identified. The tests can be appreciated in Figure 5.18.

The curve softens in the last stretch, allowing the car to travel through that part with less difficulty, just as it did in the previous case. The car's size should be taken into account again in future tests, as it is impossible to conclude if it collides with objects at this level.

[1] Vehicle from Universidad Carlos III de Madrid used for studies in the field of Autonomous Driving

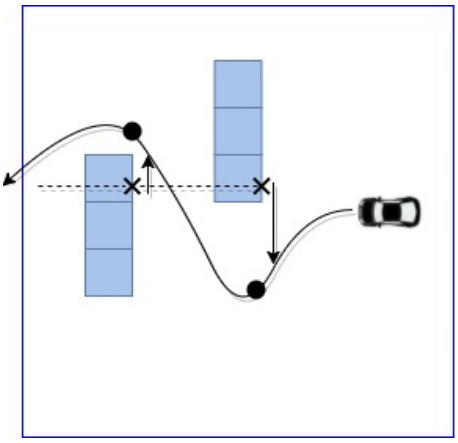


Figure 5.17: Two obstacle avoidance abstraction

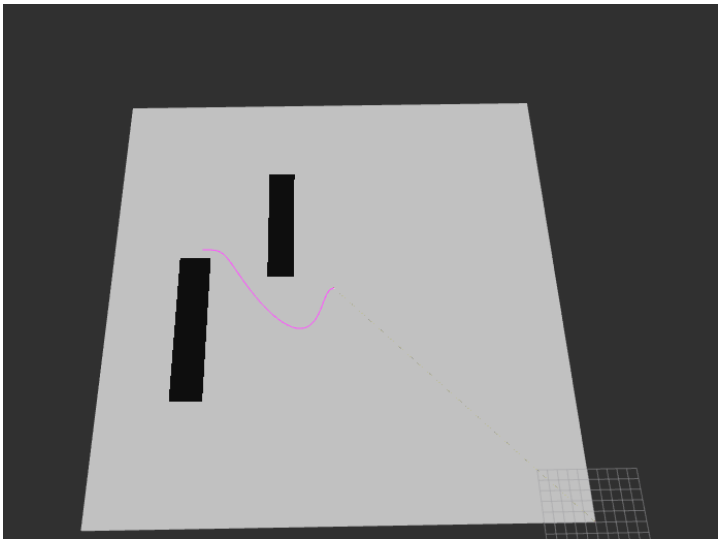


Figure 5.18: Two obstacle avoidance in RVIZ

5.4.1 Simulation in Gazebo

Now it is time to test the whole system in a simulated environment. A map with different obstacles has been chosen. The global map is able to plan the path across the obstacles (plotted in green) while the local map follows a control to be adjusted to it.

In the first example, an unobstructed trajectory has been achieved so that the obstacle avoidance system does not come into operation.

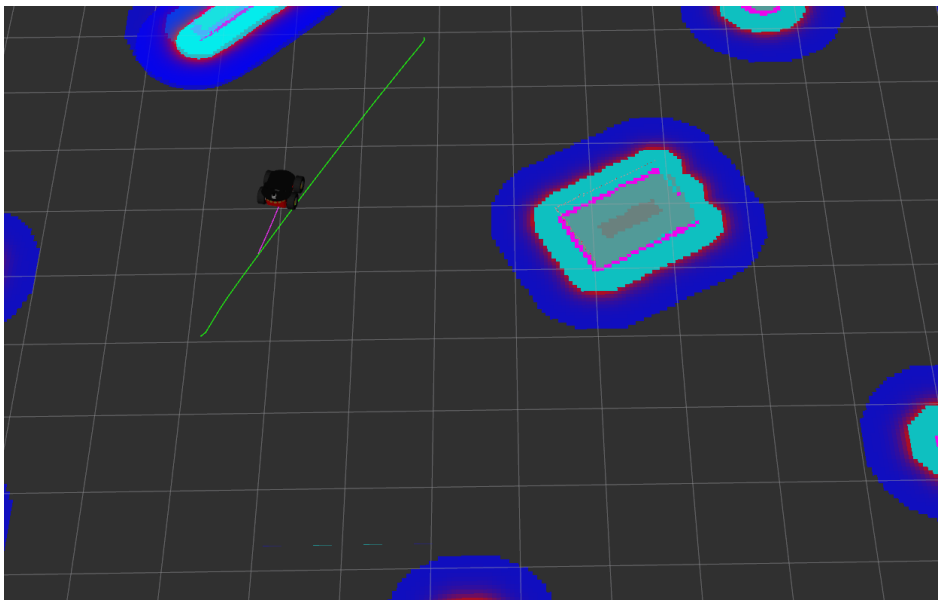


FIGURE 5.19. Straight line plan

Then a more complicated scenario comes into place. The vehicle needs to be able to avoid the obstacles in between the goal and the origin.

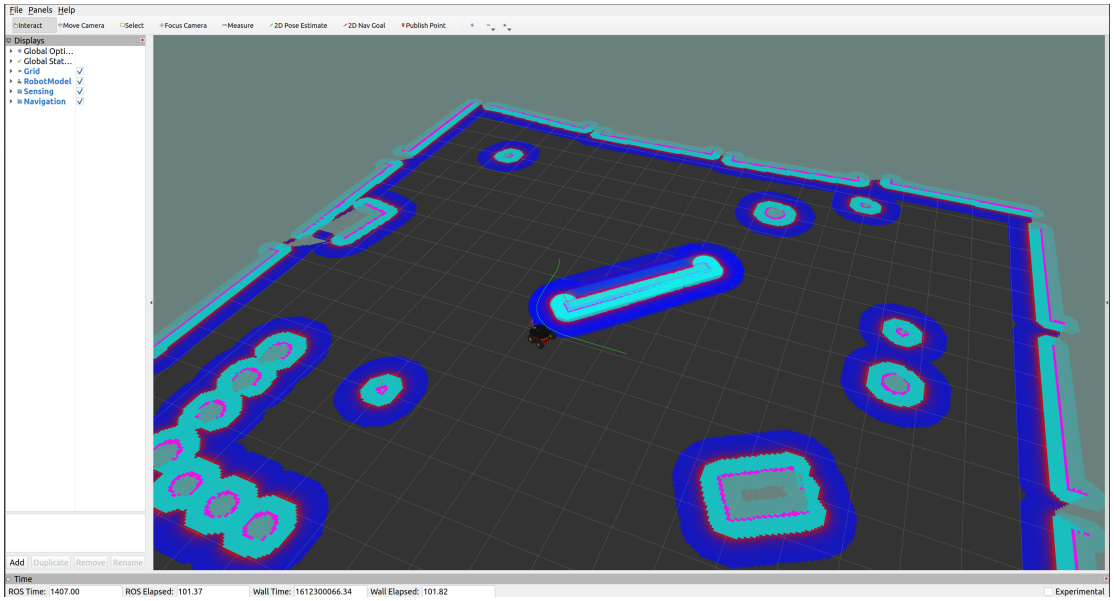


FIGURE 5.20. Global and Local Plan in Gazebo

The green line represents the **global planner** whereas the pink one references to the **local planner**. It can be seen how the vehicle avoids the obstacle without any problem.

5.4.2 Avoiding obstacles when moving backwards

For testing the conditions, the Trajectory Rollout planner has been chosen since that's the one with trouble when he moves back. The robot has been placed in the middle of the grid. The destination point has been placed behind him so that he has to start moving in reverse.

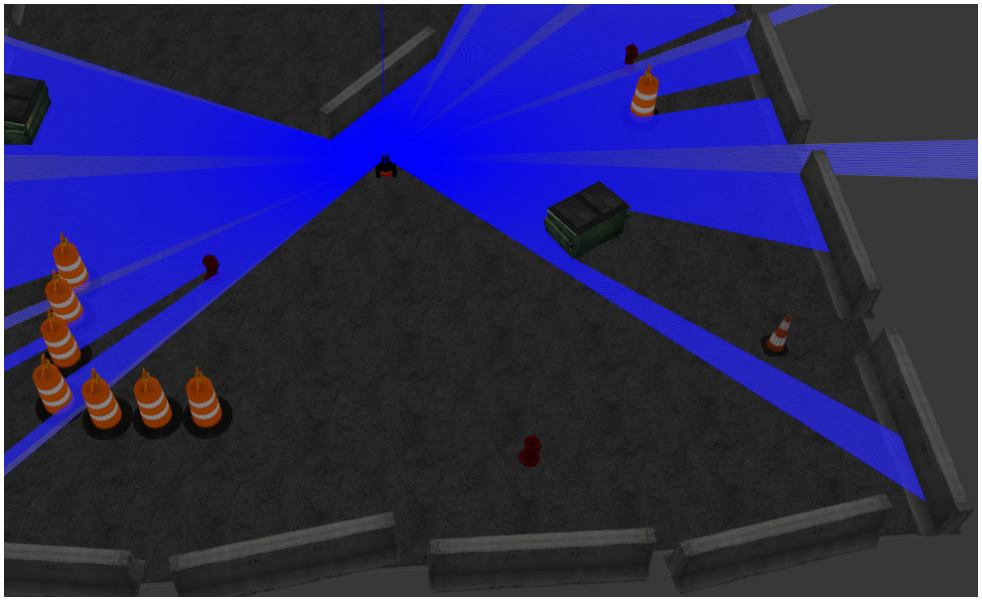


FIGURE 5.21. Initial position when moving backwards

When an obstacle is detected (measured in the code as having a cost value bigger than 0.5) the value is then not stored in the computed trajectory.

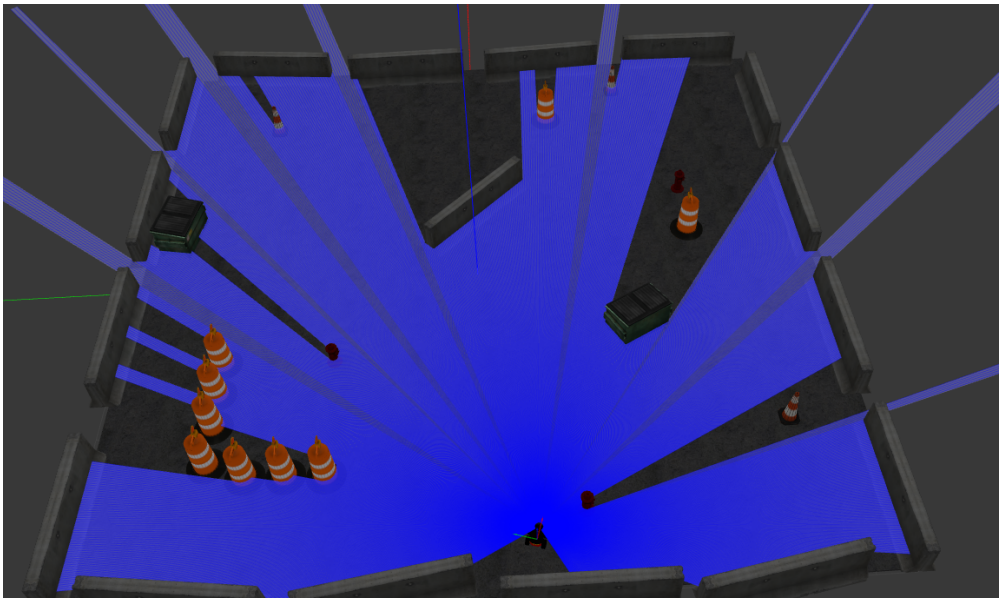


FIGURE 5.22. Final position when moving backwards

That leads to a situation where the robot stops and tries to calculate a different route.

CHAPTER 6

CONCLUSIONS AND FURTHER IMPROVEMENT

During this work, it has been possible to design a local planner capable of tracing a path given a point and an orientation of origin and destination. In addition, it has been possible to avoid obstacles not only in front but also to detect them at the rear. Some modifications have also been made to improve the twirling behavior.

6.1 Contributions summary

The main content of this project describes all the contributions to the Autonomous Driving field (Chapters 3 4). The points summarized below serve to get an idea of the content reviewed in this work:

- Using the Bezier curves, an algorithm has been created in which, given a starting point and a destination point, a path is planned in a fixed number of points. The algorithm has been written in C ++ and different alternative paths are offered as a result.
- A component has been created in ROS that isolates the behavior of the local scheduler from the rest of the system, making it modular. This results in a planner that can be improved and expanded in the future. The reference systems of the map must be taken as input and the ROS libraries are used to transform it to the end point
- The usage of the obstacle avoidance system has been extended to be used from a global planner. The coordinates maps from the physical world to the virtual map in Rviz.
- The anti-collision system has been integrated into a Global Planner. A Local Planner has been created from the ros move_base plugin. Using Gazebo, a real environment has been simulated to see how the planner reacts.

- The planner is able to detect obstacles on his back using a wide range sensor. In case that it detects an obstacle may cause the malfunction of the robot, it stops the system immediately.
- A threshold for the twirling has been added. This avoid sudden changes in the robot heading so we can control the twirling behaviour from the robot.
- Different vehicle types have been taken into account for instance, for the obstacle avoidance in the back, non-holonomic vehicles which is the representation of a standard four-wheeled vehicle has been used as reference.
- Not only four wheeled vehicles are represented in this work. Vehicles without wheels like drones can also take the advantage from the ROS autonomous system.

6.2 Conclusions

- The integration of a module designed by yourself in a system such as ROS in this case and its local planners is a long and complex process. It is a good way to work to first test the libraries provided by ROS
- There is a lot of uncertainty on when the autonomous vehicles will be ready. By the time the final version of the autonomous vehicle is ready, probably all of them will be electric.
- If you want to be a successful Robotics Engineer, you need good C++/Python skills. C/C++ are not as user-friendly as Python or MATLAB, for example. It will take a lot longer and require a lot more lines of code to achieve the same functionality in C.
- Robots' hardware evolves with time. This means that some software requirements that were taken for granted must be reviewed from time to time.
- There are different alternatives when it comes to designing a local planner. In this case, the advantages / disadvantages have been compared but a clear conclusion was not yet reached. The DWA has finally been taken over because the code is clearer and its functions are more modular.

6.3 Future work

Robotics is a technological field that is constantly evolving. That is why, as innovations occur, new planning strategies can be devised taking advantage of the technology of the moment.

- Adding a web api able that compiles all of the telemetry data and plot the robot odometry. The web site can be based on React/Javascript technologies.

- Prior to construct the trajectory in an unknown environment, use the SLAM technique to find out the shape of the map without the need to import the map scheme.
- Test the whole system on real hardware. Companies like Nvidia, Infineon... have autonomous driving platforms in the shape of System on Chip optimized for secured computing.
- Compare my local planner with other planners and train it against a neural network. There are different path planning Deep Learning methods like Q-Learning algorithm or DQN algorithm. [35]

APPENDIX A

APPENDIX: THE SOURCE CODE

Some of the files were too big to be attached to this document. That is the reason why a GitHub repository has been created. The following link redirects to the old Bachelor's repository: Click here

The prototype for checking obstacles in the back can be found in: Click here

The code related to the obstacle detection using SVM can be found in: Click here

The code related to the twirling behaviour is in the appendix below:

```
#ifndef TWIRLING_COST_FUNCTION_H
#define TWIRLING_COST_FUNCTION_H

#include <base_local_planner/trajectory_cost_function.h>

namespace base_local_planner {


class TwirlingCostFunction: public base_local_planner::TrajectoryCostFunction {
public:

  TwirlingCostFunction() {}
  ~TwirlingCostFunction() {}

  double scoreTrajectory(Trajectory &traj);

  bool prepare() {return true;};
};

} /* namespace base_local_planner */
#endif /* TWIRLING_COST_FUNCTION_H_ */
```

```
#include <base_local_planner/twirling_cost_function.h>

#include <math.h>

namespace base_local_planner {

double TwirlingCostFunction::scoreTrajectory(Trajectory &traj) {
  return fabs(traj.thetav_);  // add cost for making the robot spin
}

}
```

```
#!/usr/bin/env python
# DWA Planner configuration

from dynamic_reconfigure.parameter_generator_catkin import ParameterGenerator,
 double_t, int_t, bool_t
from local_planner_limits import add_generic_localplanner_params

gen = ParameterGenerator()


# This unusual line allows to reuse existing parameter definitions
# that concern all localplanners
add_generic_localplanner_params(gen)


gen.add("sim_time", double_t, 0, "The amount of time to roll trajectories
out for in seconds", 1.7, 0)
gen.add("sim_granularity", double_t, 0, "The granularity with which to
check for collisions along each trajectory in meters", 0.025, 0)
gen.add("angular_sim_granularity", double_t, 0, "The granularity with
which to check for collisions for rotations in radians", 0.1, 0)


gen.add("path_distance_bias", double_t, 0, "The weight for the path
distance part of the cost function", 32.0, 0.0)

gen.add("goal_distance_bias", double_t, 0, "The weight for the goal
distance part of the cost function", 24.0, 0.0)

gen.add("occdist_scale", double_t, 0, "The weight for the obstacle
distance part of the cost function", 0.01, 0.0)

gen.add("twirling_scale", double_t, 0, "The weight for penalizing
any changes in robot heading", 0.0, 0.0)




gen.add("stop_time_buffer", double_t, 0, "The amount of time that the
robot must stop before a collision in order for a trajectory to be
considered valid in seconds", 0.2, 0)

gen.add("oscillation_reset_dist", double_t, 0, "The distance the robot
must travel before oscillation flags are reset, in meters", 0.05, 0)
gen.add("oscillation_reset_angle", double_t, 0, "The angle the robot
must turn before oscillation flags are reset, in radians", 0.2, 0)

gen.add("forward_point_distance", double_t, 0, "The distance from the
center point of the robot to place an additional scoring point, in meters", 0.325)

gen.add("scaling_speed", double_t, 0, "The absolute value of the
 velocity at which to start scaling the robot's footprint, in m/s", 0.25, 0)
gen.add("max_scaling_factor", double_t, 0, "The maximum factor to
 scale the robot's footprint by", 0.2, 0)

gen.add("vx_samples", int_t, 0, "The number of samples to use
when exploring the x velocity space", 3, 1)
gen.add("vy_samples", int_t, 0, "The number of samples to use when
 exploring the y velocity space", 10, 1)
gen.add("vth_samples", int_t, 0, "The number of samples to use when
 exploring the theta velocity space", 20, 1)

gen.add("use_dwa", bool_t, 0, "Use dynamic window approach to constrain
 sampling velocities to small window.", True)

gen.add("restore_defaults", bool_t, 0, "Restore to the original
 configuration.", False)

exit(gen.generate("dwa_local_planner", "dwa_local_planner", "DWAPlanner"))
```

BIBLIOGRAPHY

[1] S. A. Fadzli, S. I. Abdulkadir, M. Makhtar, and A. A. Jamal, "Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries," in *Information Science and Security (ICISS), 2015 2nd International Conference on*. IEEE, 2015, pp. 1–4.

[2] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[3] L. Dormehl and S. Edelstein. Sit back, relax, and enjoy a ride through the history of self-driving cars. [Online]. Available: https://www.digitaltrends.com/cars/history-of-self-driving-cars-milestones/

[4] J. U. The road to driverless cars: 1925 - 2025. [Online]. Available: https://www.engineering.com/DesignerEdge/DesignerEdgeArticles/ArticleID/12665/The-Road-to-Driverless-Cars-1925--2025.aspx

[5] E. Benedito, A. Doria-Cerezo, C. Kunusch, and J. M. Olm, "Traffic flow-oriented design and analysis of an adaptive cruise control system," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[6] J. Claybrook, "Safety: The view from nhtsa: New legislation promises to make cars safer, cheaper to repair, and less polluting during the 1980s," *IEEE spectrum*, vol. 14, no. 11, pp. 87–88, 1977.

[7] Stanley, el vehículo sin conductor de la universidad de stanford, gana la carrera de 'autos-robot'. [Online]. Available: https://elpais.com/tecnologia/2005/10/10/actualidad/1128932879_850215.html

[8] "Vehículo autónomo - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Veh{í}culo_aut{ó}nomo

[9] Road traffic injuries. [Online]. Available: http://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[10] Motor vehicle crash deaths. [Online]. Available: https://www.cdc.gov/vitalsigns/motor-vehicle-safety/infographic.html

[11] R. Future. España, pionera en la regulación de los vehículos autónomos. [Online]. Available: https://future.inese.es/espana-pionera-en-la-regulacion-de-los-vehiculos-autonomos/

[12] N. López. (2017) Legislación sobre el coche autónomo en españa: queda mucho por hacer. [Online]. Available: https://www.autobild.es/reportajes/legislacion-coche-autonomo-espana-queda-mucho-hacer-179660

[13] L. Thomas and J. Irish. (2017) Germany adopts self-driving vehicles law. [Online]. Available: https://www.reuters.com/article/us-germany-autos-self-driving/germany-adopts-self-driving-vehicles-law-idUSKBN1881HY

[14] Automated and autonomous driving. legal framework. [Online]. Available: https://www.daimler.com/innovation/case/autonomous/legal-framework.html

[15] O. Bowcott. Laws for safe use of driverless cars to be ready by 2021. [Online]. Available: https://www.theguardian.com/law/2017/dec/14/laws-safe-use-driverless-cars-ready-2021-law-commission

[16] N. Lomas. (2018) Uk kicks off driverless car law review to get tech on the road by 2021. [Online]. Available: https://techcrunch.com/2018/03/07/uk-kicks-off-driverless-car-law-review-to-get-tech-on-the-road-by-2021/?guccounter=1

[17] M. Mittereder. Automated vehicles: Legal framework will decide which automakers win the race to full automation. [Online]. Available: https://www.rolandberger.com/en/Media/Automated-Vehicles-Legal-Framework-Will-Decide-Which-Automakers-Win-the-Race-to.html?country=null

[18] T. Peng. (2018) Global survey of autonomous vehicle regulations. [Online]. Available: https://medium.com/syncedreview/global-survey-of-autonomous-vehicle-regulations-6b8608f205f9

[19] Á. Melgosa Pascual, "Uav planning, autonomous tracking, and obstacle identification and avoidance," 2016.

[20] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[21] M. Likhachev, "Search-based Planning with Motion Primitives." [Online]. Available: http://www.ros.org/wiki/sbpl

[22] X. Ji, L. Liu, P. Zhao, and D. Wang, "A-star algorithm based on-demand routing protocol for hierarchical leo/meo satellite networks," in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1545–1549.

[23] "D* - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/D*

[24] J. G. Webster, S. Huang, and G. Dissanayake, "Robot Localization: An Introduction," *Wiley Encyclopedia of Electrical and Electronics Engineering*, pp. 1–10, 2016.

[25] P. E. Black, "Manhattan distance," *Dictionary of Algorithms and Data Structures*. [Online]. Available: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

[26] H. Sagar, "Comparison of A *, Euclidean and Manhattan distance using Influence Map in Ms . Pac-Man Sudip Karki Hari Sagar Ranjitkar," 2016.

[27] "Robot Operating System 2 (ROS 2) Architecture | by Huseyin Kutluca | Software Architecture Foundations | Medium." [Online]. Available: https://medium.com/software-architecture-foundations/robot-operating-system-2-ros-2-architecture-731ef1867776

[28] "Data Distribution Service - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Data_Distribution_Service

[29] D. Fox, "KLD-sampling: Adaptive particle filters," *Advances in Neural Information Processing Systems*, 2002.

[30] E. López, S. García, R. Barea, L. M. Bergasa, E. J. Molinos, R. Arroyo, E. Romera, and S. Pardo, "A multi-sensorial simultaneous localization and mapping (SLAM) system for low-cost micro aerial vehicles in GPS-denied environments," *Sensors (Switzerland)*, vol. 17, no. 4, apr 2017.

[31] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[32] "Wayback Machine." [Online]. Available: https://web.archive.org/web/20060221000535/http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf

[33] S. D. Raje, "Evaluation of ROS and Gazebo Simulation Environment using TurtleBot3 robot," no. April, 2020.

[34] Brian Gerkey, "gmapping - ROS Wiki," p. 2, 2018. [Online]. Available: https://wiki.ros.org/gmapping{%}0Ahttp://wiki.ros.org/gmapping

[35] J. Yu, Y. Su, and Y. Liao, "The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning," *Frontiers in Neurorobotics*, vol. 0, p. 63, oct 2020.

[36] Bézier curves and kronecker's tensor product. [Online]. Available: https://blogs.mathworks.com/images/graphics/bezierpost.html

[37] S. K. Gehrig and F. J. Stein, “Collision avoidance using elastic bands for an autonomous car.”

[38] P. Ranger, “Comments, with reply, on’structured flowcharts outperform pseudocode: an experimental comparison’by da scanlan,” *IEEE Software*, vol. 7, no. 2, p. 6, 1990.

[39] [Online]. Available: https://www.youtube.com/watch?v=2HvH9cmHbG4

[40] “Autonomous Vehicles to Have Huge Impact on Economy, Tech Sector.” [Online]. Available: https://www.govtech.com/fs/automation/autonomous-vehicles-to-have-huge-impact-on-economy-tech-sector.html

[41] “Normativa de Drones en España 2021: Todo lo que debes saber | One Air.” [Online]. Available: https://www.oneair.es/normativa-drones-espana-aesa/