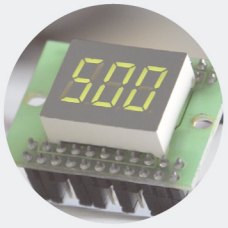
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Sistemas de suporte à condução autónoma adequados a plataforma robótica 4-wheel skid-steer: percepção, movimento e simulação

FRANCISCO JORGE SOUSA PINHEIRO
novembro de 2019

POLITÉCNICO
DO PORTO

Autonomous driving supporting systems for a 4-wheel skid-steer robotic platform: perception, motion control and simulation.

In the scope of Festival Nacional de Robótica - Autonomous Driving Competition

Francisco Jorge Sousa Pinheiro
Nº 1110395

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2019

Dissertação, para satisfação parcial dos requisitos do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Francisco Jorge Sousa Pinheiro

Nº 1110395

Orientador: Prof. Dr. André Miguel Pinheiro Dias

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

November 26, 2019

Dedicatória

”À minha mulher Elisa, que mesmo travando a batalha de todas as batalhas nunca deixou de encontrar em si força e alento para me empurrar para a frente e aos meus filhos Pedro e Maria que tornaram tudo mais colorido. Que este seja a página de um novo capítulo das nossas vidas.”

This page was intentionally left blank.

Resumo

As competições de robótica móvel desempenham papel preponderante na difusão da ciência e da engenharia ao público em geral. É também um espaço dedicado ao ensaio e comparação de diferentes estratégias e abordagens aos diversos desafios da robótica móvel. Uma das vertentes que tem reunido maior interesse nos promotores deste género de iniciativas e entre o público em geral são as competições de condução autónoma. Tipicamente as Competições de Condução Autónoma (CCA) tentam reproduzir um ambiente semelhante a uma estrutura rodoviária tradicional, no qual sistemas autónomos deverão dar resposta a um conjunto variado de desafios que vão desde a deteção da faixa de rodagem à interação com distintos elementos que compõem uma estrutura rodoviária típica, do planeamento trajetórias à localização.

O objectivo desta dissertação de mestrado visa documentar o processo de desenho e concepção de uma plataforma robótica móvel do tipo 4-wheel skid-steer para realização de tarefas de condução autónoma em ambiente estruturado numa pista que pretende replicar uma via de circulação automóvel dotada de sinalética básica e alguns obstáculos.

Paralelamente, a dissertação pretende também fazer uma análise qualitativa entre o processo de simulação e a sua transposição para uma plataforma robótica física. inferir sobre a diferenças de performance e de comportamento.

Palavras-Chave: Condução Autónoma, Visão Computacional, Controlo, Trajetória, *Obstacle Avoidance*

This page was intentionally left blank.

Abstract

Mobile robotics competitions play an important role in the diffusion of science and engineering to the general public. It is also a space dedicated to test and compare different strategies and approaches to several challenges of mobile robotics. One of the aspects that has attracted more the interest of promoters for this kind of initiatives and general public is the autonomous driving competitions. Typically, Autonomous Driving Competitions (CCAs) attempt to replicate an environment similar to a traditional road structure, in which autonomous systems should respond to a wide variety of challenges ranging from lane detection to interaction with distinct elements that exist in a typical road structure, from planning trajectories to location.

The aim of this master's thesis is to document the process of designing and endow a 4-wheel skid-steer mobile robotic platform to carry out autonomous driving tasks in a structured environment on a track that intends to replicate a motorized roadway including signs and obstacles.

In parallel, the dissertation also intends to make a qualitative analysis between the simulation process and the transposition of the developed algorithm to a physical robotic platform, analysing the differences in performance and behavior.

Key words: Autonomous Driving, Computer Vision, Control, Path, Obstacle Avoidance

This page was intentionally left blank.

Contents

This page was intentionally left blank.

List of Figures

This page was intentionally left blank.

List of Tables

This page was intentionally left blank.

List of Acronyms

4WSS 4-Wheel Skid-Steer

AADC Audi Autonomous Driving Cup

ADC Autonomous Driving Competition

ADTF Automotive Data and Time-triggered Framework

ADAS Advanced Driving Assistance Systems

BSD Berkeley Software Distribution

CAN Controller Area Network

FNR Festival Nacional de Robótica

FSM Finite State Machine

FTT-CAN Flexible Time Triggered CAN

GPS Global Positioning System

HSV Hue Saturation Value

ICR Instant Center of Rotation

IG Information Gain

INESC-TEC Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

INS Inertial Navigation System

LiDAR Light Detection and Ranging

LSA Laboratório de Sistemas Autónomos

LUT Look-Up Table

ML Machine Learning

MLL Machine Learning Library

MORSE Modular Open Robots Simulation Engine

MPC Model Predictive Control

MSE Mean Squared Error

PCA Prova Condução Autónoma

PID Proportional Integrative Derivative

PRM Probabilistic Roadmap

PWM Pulse Width Modulation

RC Radio Controlled

RGB Red Green Blue

ROI Region Of Interest

ROS Robotic Operating System

RRT Rapidly-exploration Random Tree

RVG Reduced V isibility Graph

SVM Support Vector Machine

XML Extensible Markup Language

YAML YAML Ain't Markup Language

YUV Y-Luminance, U and V-Chrominance

Chapter 1

Introduction

There is no denying the growing interest and popularity that advanced driving support systems and lately autonomous driving systems have achieved in recent years. As one of the main focuses of interest of research teams around the world, autonomous driving systems have experienced an exponential development in both complexity and robustness of the proposed solutions. This development has shown consistently positive results, anticipating in the short term their applicability in everyday life.

The development of autonomous driving systems is greatly contributed by several scientific and research groups, both in an academic context and in a more entrepreneurial perspective, groups that promote distinguished initiatives to stimulate creativity, redesigning concepts that aim to address solutions to new and more complex challenges associated to the autonomous driving. A clear example of this call for creativity is the several competitions that have emerged over the last decade like Darpa's Grand Challenge [6], Audi's Autonomous Driving Cup [7] or CARLA Autonomous Driving Challenge [8].

Some of the currently existing competitions are fundamentally oriented towards entertainment without a clear character in the development of new technologies and solutions. However, many others are science-oriented, involving more relevant research, development, and improvement of sub-systems necessary to support autonomous driving solutions. Mobile robotics competitions bring many benefits, both to academia, industry and to society in general [9].

Considering the national panorama, a relevant example of robotics competition, and particularly the autonomous driving competition, is the Festival Nacional de Robótica - Autonomous Driving Competition (FNR-ADC). This competition challenges competitors to address solutions to the following problems: designing mechanical platform,

defining and assembling hardware, designing and implementing modules like driving, parking, and vertical traffic sign identification [10]. The driving challenge is composed of four rounds with increasing difficulty, where, on each round, the robots must complete two laps within the shortest time. In the first round, robots shall complete the two laps at pure speed. In the following round, the robot shall complete the laps and simultaneously react upon signal identification, such as stopping or changing lane. In the third round, a tunnel is placed on the track and obstacles are also placed at unknown locations. Finally, in the fourth round, a road working area is added.

Regarding the parking challenge, the robots must react after the parking signal is switched on. When the signal is on, the robot must drive to one of the parking areas - parallel parking zone or parking lot - with possible obstacles strategically placed to difficult maneuvering and park in one of these places.

In the end, a special challenge is addressed to participants, where six from a group of twelve possible vertical signs are positioned along the track. The robots shall identify them successfully.

The ADC addresses relevant Educational Robotic topics, concerning important fields of mobile robotics, like localization, motion control, path planning and following, mapping, perception, computer vision, machine learning, and several others equally important and relevant.

Naturally, each of these topics feeding an autonomous driving system is vast and the plurality of different strategies and solutions enriches the vast universe of autonomous driving.

Taking as a starting point the race specifications known *a priory*, such as the track's topology, shape or dimensions of obstacles, and the signage used, this dissertation proposes to design and implement possible strategies in response to some of the challenges present in Festival Nacional de Robótica - Autonomous Driving Competition, more in particular, parking, path-following and obstacle avoidance. To this end, modules of path following, motion control, and computer vision will be developed and tested to support the autonomous driving system.

1.1 Motivation

Autonomous Driving is a popular scientific topic. Today is undeniable the presence of autonomous driving systems or as a lighter version, the Advanced Driving Assistance Systems (ADAS). Advanced systems deal with cutting-edge algorithmics and disruptive solutions, challenging the ever-increasing communities, academic, business and simple

passionate to embrace this field of robotics.

Participating in the next edition of FNR-ADC with an autonomous driving robot, and with that develop robust and suited solutions capable to accomplish all the challenges proposed by the Festival Nacional de Robótica.

1.2 Objectives

This dissertation aims the problematic of designing modules or sub-systems capable to integrate an Autonomous Driving System capable to answer successfully each of the challenges present in FNR-ADC. Therefore the development of this project requires the following topics to be fulfilled:

- Characterization of autonomous driving competitions and their scope;
- Characterization of relevant platforms hardware and software;
- State-of-the art of the main participants of previous editions of FNR-ADC;
- Analysis of several strategies supporting autonomous driving systems;
- Development and implementation of the modules of computer vision, parking lot identification, path-following and obstacle avoidance;
- Testing the modules in simulation environment designed and built with full compliance to the ADC specifications;
- Analysis of the performance and defining possible improvements.

1.3 Structure

This dissertation is organized through seven chapters. In chapter two it is presented the state-of-the-art where it is given an insight about the most advanced and relevant autonomous driving competitions worldwide. Then, a further overview about the Festival Nacional de Robótica - Autonomous Driving Competitions is also provided and this chapter ends with a full review about the solutions brought buy the most relevant participating teams.

The chapter three provides a high-level overview about the project and the problem definition. The topics discussed in this dissertations are further approached in chapter 4 with a detailed framing based theory concepts and strategy standards used in the implementation of driving systems modules.

The project object of this dissertations is then detailed in chapter five, where it is given an overview about the design strategies and functionalities. These designs and then tested, as detailed in chapter six and the dissertation ends with a brief conclusion in chapter seven.

Chapter 2

State of the art

This chapter presents examples of mobile robotics platforms, the respective competition scenarios and a brief description of the solutions presented. The section will start by listing the most recent examples at the international level and will culminate in a brief description of the context of the Autonomous Driving Competition of the National Robotics Festival (FNR), reviewing the main solutions with participation in the most recent editions of the FNR-PDC. This chapter will be the starting point for understanding the strategies used in the development of the 4W-SS mobile platform and the sub-systems that support autonomous driving.

2.1 Audi Autonomous Driving Cup

The Audi Autonomous Driving Cup (AADC) is an example of a competition in mobile robotics specially designed for autonomous driving. In this competition the participants are invited to develop systems that will be tested both in iteration with other systems developed by other participants as well as in task/mission oriented challenges. The competition evaluates the overall system performance, the software robustness, and the elegance of the solution presented by the participants.

Like other competitions of this kind, a distinguishing aspect of AADC is that participating teams compete with the same robots from a mechanical, structural and hardware point of view. In the case of AADC, these are models of real Audi AG vehicles at a scale of 1:8. The main highlight of this competition is the possibility of all participating teams compete under equal conditions in terms of hardware, thus highlighting the robustness, accuracy and reliability of the developed algorithms.

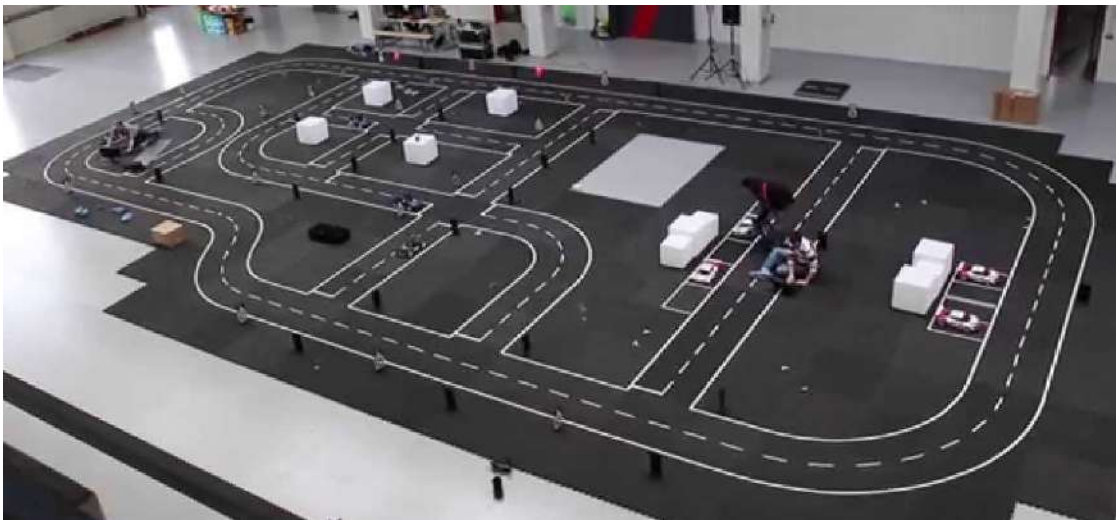


Figure 2.1: Audi-Autonomous Driving Competition - Competition scenario

The basic software for operating the robotic platforms is specified by the BSD-licensed [11] organizer (Berkeley Software Distribution, an operating system based on Research Unix). The basic software provides the necessary means for communication with sensors and actuators, but without any optimization either for control frequency or accuracy.

The competition consists of a mandatory program of autonomous driving tasks, a free style in which each team can demonstrate a special functionality or feature of the developed solution and a final stage in which the teams must perform *a priori* unknown autonomous driving tasks.

The platform is a 1:8 scale RC model of the Audi AG - Q2 range.

The robot's power supply system consists of two independent circuits. A first circuit, powered by a 22.2V 6-cell battery with a capacity of 5200mAh, is responsible for powering the computer systems dedicated to the control and acquisition of information from the sensors. This system has a high energy consumption and a battery at maximum charge ensures the operation of the system only for one hour. The second power supply system ensures the operation of the motors and speed control. This system is supported by a 2-cell battery with 7.4V and 5200mAh.

The platform is powered by a high torque brushless motor. A speed controller ensures that the motor speed in both driving directions, forward or reverse, remains constant according to the instructions of the high-level control layer. This controller also ensures braking functions in both directions.

Figure 2.2: Audi-Autonomous Driving Competition - Competition mission and challenges

Turning actions are ensured by a digital servo motor. It is characterised by its high actuation power and low reaction time or latency, which enables a fast and precise steering reaction. This turning system eliminates the need of a feedback loop of the turning angle due to an integrated control.

The robotic platforms used in AADC are equipped with a miniTX card that houses a processed Intel Core i3, 8GB RAM, a 128GB SSD disk and an NVIDIA GeForce GTX1050Ti graphics card. The miniTX card also features two Gigabit Ethernet ports, several USB3.0 ports and one USB-C port. It also has a Bluetooth, WLAN (IEEE 802.11ac) module.

The system also has an extension of communication interfaces with the robotic platform through the inclusion of a microcontroller that allows, for example, monitoring batteries charge.

The 2018 version of AADC robotic platform features a set of sensors close to those existing in real cars. One of the key innovations compared to the previous model is the replacement of the Intel RealSense r200 3D camera with the new RPLIDAR A2 applied to the front of the robot. This lists the main sensors that make up the robotic platform:

- RPLIDAR A2 (field of view >180, *range* <6m, refreshing rate 10Hz);
- Monocular front camera with 130°of field of view;
- Monocular rear camera with 80°of field of view;

Figure 2.3: Robotic platform used in Audi-Autonomous Driving Competition - exterior perspective

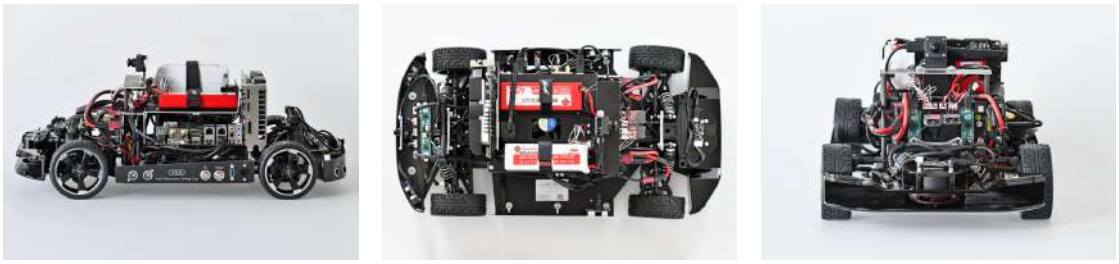


Figure 2.4: Robotic platform used in Audi-Autonomous Driving Competition - hardware

- Ultrasonic sensors (three at the rear and one at each side of the robot) with detection limit of <4m and a refreshing rate of 40Hz;
- 6-axis IMU to monitor linear accelerations and angular velocities;
- All wheel have *encoders* to measure instant velocities.

The development of the software to support the autonomous driving system is based on the environment *Automotive Data and Time-triggered Framework*. (ADTF) which represents the automotive industry standard *framework* that supports the software development process. This environment is stable and robust, used in today's Advanced Driver Assistance Systems (ADAS) with the possibility of including typical communication protocols such as CAN, FlexRAY, or Ethernet, as well as the attendance of any raw data from different sources.

2.2 Festival Nacional de Robótica - Autonomous Driving Competition

2.2.1 Description of the autonomous driving competition

The Autonomous Driving competition (ADC) of the Festival Nacional de Robótica (FNR) offers a set of challenges to mobile robotics with features that tend to replicate real driving environments on a closed track.

The structured environment of the race is composed of a *8* shaped track – in some editions in the form of *B* –, with lanes delimited by lines similar to a conventional road. The track also counts with a crosswalk, luminous signs (displayed on screen), obstacles, vertical signs, construction area and a tunnel. In addition to these characteristics of the runway and its elements, there is also a delimited parking area with two spaces, one of which will be occupied, as well as a parallel parking area.

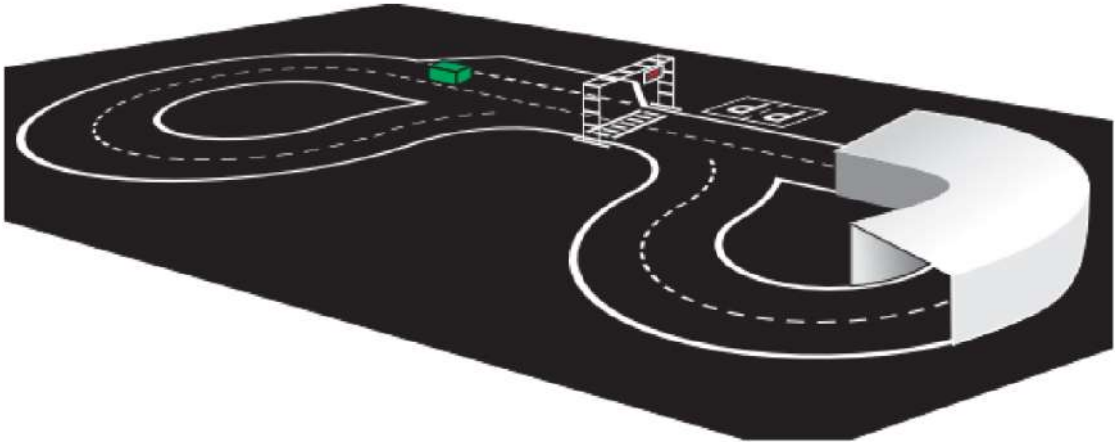


Figure 2.5: Autonomous Driving Competition scenario – Festival Nacional de Robótica

The race is composed of three runs, and the degree of difficulty is increased at each of these stages. The first run challenges the participants to make their robots drive autonomously completing the entire track. The robot must complete two laps of the circuit in the shortest possible time. Any situation the the robot exits the track can count on penalties or even disqualification of the race.

In the second run, the robot must identify the light signals displayed on the screens and act accordingly. The light panel shows five possible signals, which give order from the direction, stop, move forward or park.

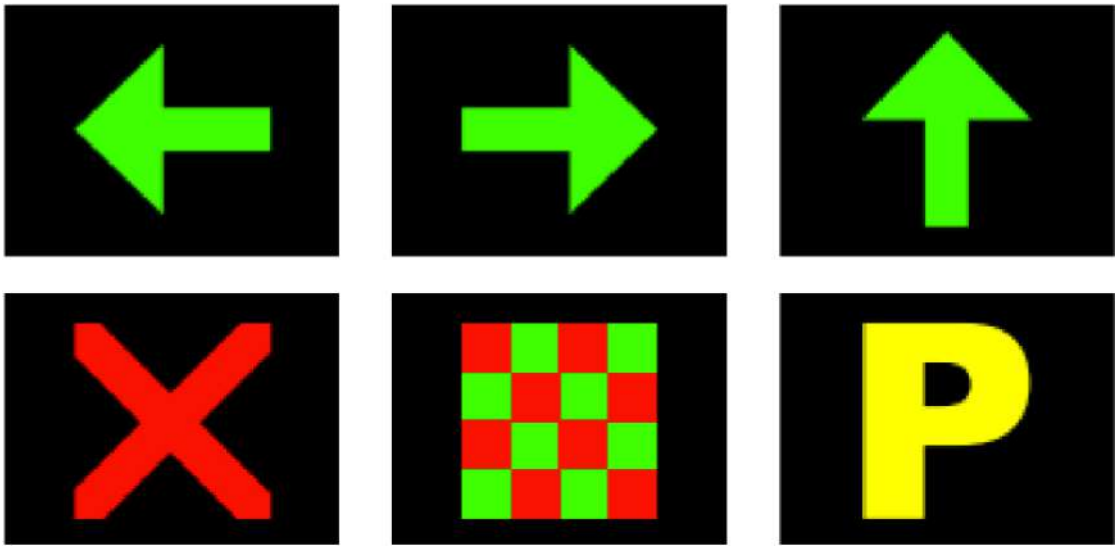


Figure 2.6: Light signals shown on the screen

The robot should, at this stage, be able to carry out the parking manoeuvre in the designated lot, with two parking areas but one them occupied, so the robot should be able to identify the free space and park there.

Another challenge in this phase is related to the placement of an obstacle on the road, simulating the presence of a vehicle. The robot should identify the obstacle and, without leaving the track, overpassing it using the lane on its left.

In the third and final phase, the difficulty is increased with the incorporation of a simulated construction area and a tunnel that influences sensors perception due to low-light conditions. The construction area delimited by traffic cones interconnected with ribbons presupposes a challenge of robustness and adaptability of the solution to identify/estimate the lane to follow when the scene faces a sudden change in its structure. The construction area is defined without prior knowledge, tracing a new route delimited by traffic cones.

2.2.2 FNR-ADC participating platforms: hardware and software

In this section a review is made about the robotic platforms with relevant participation in the last editions of *Festival Nacional de Robótica – Autonomous Driving Competition* (FNR-ADC). In the following subsections, a survey of the hardware and software options, developed by each participating team, to overcome the challenges that make up the autonomous driving competition is carried out. For this purpose, contacts were made

with the the different team managers in order to obtain the information described in this chapter (see Appendix A).

A special thank you to Eng Manuel Silva — [mss@isep.ipp.pt] — for providing some of these contacts.

2.2.2.1 N3E GT Team - Polytechnic Institute of Leiria

The N3E GT project of the team of the Polytechnic Institute of Leiria is based on an RC platform with four-wheel drive and steering system with *Ackerman* geometry – geometrical principle similar to the various steering systems seen on conventional cars –, which allows greater control in curve (see figure 2.7). Another important aspect of the platform adopted by the Leiria's team is the existence of a transmission differential, which ensures that the wheels have a different speed between them, suitable for the turning path.

Figure 2.7: N3E GT Robotic platform - Polytechnic Institute of Leiria

In terms of hardware implemented, the N3E GT robot system is supported by 8 and 32-bit PIC microcontrollers dedicated to low-level control and acquisition of information from some sensors. The low-level system is responsible for the activation of the mechanical systems of gait, steering, and the system of *pan* of the front camera using

motors and servos-motors. As mentioned, the low-level processing endows the system with a feedback loop with the introduction of acquired information from encoders into the system.

For high-level processing, the system features a conventional Intel Core i7 4700HQ eight-core 2.5GHz PC with 8GB of RAM, featuring NVIDIA GeForce GT 745M graphics card. The widely advantageous graphics card for parallel processing is based on Kepler's 384-color 800MHz architecture supporting CUDA. The high-level layer is responsible for image processing and extraction of *features*. The robot is equipped with two SCEE SLEH-00203 cameras, commercially known as PlayStation EYE. These cameras offer a resolution of 640x480 pixels with a maximum *frame rate* of 60-fps, however for a resolution of 320x240 pixels it is possible to achieve a *frame rate* of 120-fps. The optical group uses a 132.9°field of view lens, a customization made to the PlayStation EYE camera [12].

The control system developed by the Leiria's team consists of three subsystems; a first subsystem directly associated with the extraction of information through the cameras that are assembled in the robot (a camera for track analysis and another for signage); another subsystem dedicated to sending commands to actuators and extracting information from sensors; and a third subsystem dedicated to high-level control, namely decision making 2.8.

These subsystems communicate with each other through the *ROS middleware*, with the publication and subscription of topics.

As listed above, the low-level control subsystem constitutes a layer between the actuators/sensors, and the high level control. The low-level subsystem accommodates the commands received from the high level control such as: speed, wheel turn angle, camera *tilt* angle capturing the track and brake, converting these commands into actuator parameters, for example their conversion into *pulse width modulated* (PWM) signals to be applied to servo motors.

At the same time, low-level control is assigned the task of collecting information from sensors, including an optical sensor, developed by the Polytechnic Institute of Leiria, which performs the function of *encoder*, collecting information on angular movement of each wheel, relevant information given that the platform, as already mentioned, has a differential transmission. The collection of sensor data in the turning system, signal voltage, is another aspect considered in the feedback loop of the high level control system.

Tasks such as image processing and decision making are associated with the high level control layer. In the field of image processing, one of the fundamental tasks for the robot to move within the range is the perception and tracking of the track. The system

created by Leiria's team is based on the existence, always visible, of the line delimiting the strip to the right of the robot, i.e., the line that limits the exterior of the track.

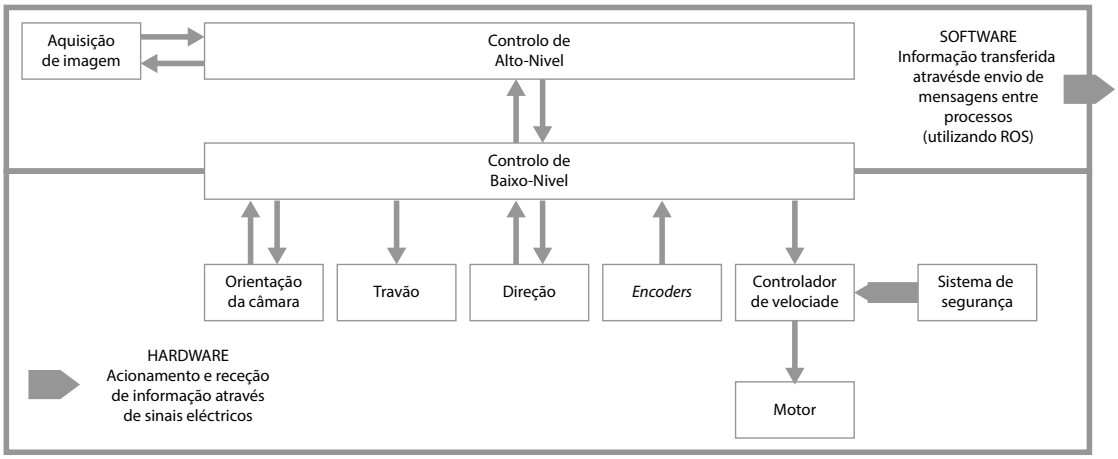


Figure 2.8: Robot N3E GT Control System block diagram**??**

The image processing starts with the definition of a *region of interest* (ROI), which is then segmented on *luminance*, this is, by a grayscale converted image. Using a *threshold*, pixel information with values below a certain *luminance* those pixels are eliminated.

Then, the information remaining in the image is grouped into a type of *cluster* in which the contours of the objects are gathered in order of proximity to the lower right corner of the image. Only contours that meet a certain size/contiguity criteria are grouped together in *clusters*, discarding smaller contours.

Once created, the *clusters* are later analyzed and retained those that have a larger dimension, that is, those that cover a larger number of lines in the image. The *cluster* with the largest dimension is considered to be lane's delimiting line. Finally, using a point on the image that Leiria's team refers to as the *center of focus*, the distance between this point and the identified contour/ *clusters* is calculated. This distance is nothing more than the distance in pixels, in the same line of the image, between the *center of focus* and the outline. Based on this distance, the robot's trajectory error is determined and the turning angle to be adopted is calculated so that the robot maintains a trajectory centered in the lane.

The crosswalk recognition is also one of the functions assigned to high-level control. The same *luminance* concept is also used in the process of identifying the crosswalk. After discarding information that does not satisfy a minimum value of *luminance threshold*, the number of white pixels in the image is counted. When the crosswalk is captured by

the camera and as the robot approaches, the number of white pixels counted in the image increases. The goal of the Leiria's team is to provide the system with a simple white pixel counter and to verify if a minimum value of pixels is counted. If this criterion is met, then the system assumes that it is in front of the crosswalk. To increase the robustness of the solution, the team included a cycle counter to check the consistency of white pixel counter. Finally, the solution developed applies the same principle of white pixel counting but with an inverse logic in the attempt to perceive when the crosswalk is transposed.

All subsystems associated with the identification of light signals uses the same concept of white pixel counter, which satisfies a minimum condition of *luminance*, and thus weigh up whether the number of counted pixels satisfies a certain condition or not. The same concept is also taken into account when identifying the screens in the captured image captured. However, in this case, the criterion used is the absence of *luminance*. However, according to a qualitative analysis done by Leiria's team itself, this method proved to be quite fragile, especially in the identification of the signals that indicates a change of direction to the robot. The vulnerability of the system to the lighting conditions is another negative aspect pointed out by Leiria's team.

2.2.2.2 ROTA Team - University of Aveiro

The ROTA team (Robot Triciclo para Condução Autónoma) is the robotic platform developed by the University of Aveiro and which has been present in FNR's recent editions (see figure 2.9). This platform succeeds another one well known as ATLAS that participated in the 2000 editions with great success [13] [14] [15].

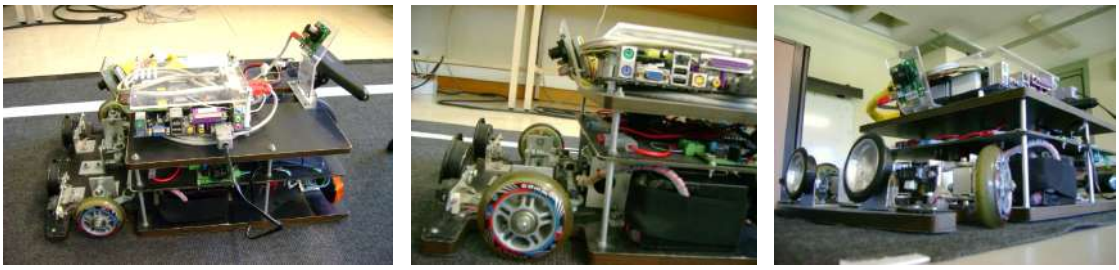


Figure 2.9: ROTA's robotic platform - University of Aveiro

ROTA's physical platform is a platform with one actuated wheel and two steered wheels through a geometry system *ackerman*. With 12 kg weight and 50x33.4x28 cm (LxWxH) in size, it is driven by a 12V/150W DC motor applied by a pinion/chain/racket

assembly to the rear sprocket and a servomotor steering the two front wheels.

The entire system is powered by 2 12V/7200mAh batteries.

The high-level control system is provided by a *single-board computer* mini-ITX board operating at 1GHz and with 512MB of RAM, which communicates with low-level control subsystems via RS-232 ports. By the other hand, the two cameras communicate through a Firewire port (IEEE 1394 type A). More recently, the mini-ITX card has been replaced by a traditional laptop-PC.

The low-level control system presents a distributed structure, based on Microchip microcontrollers of the PIC18 family, specifically PIC18F2580 and PIC18F4580 communicating with each other through a FTT-CAN bus developed by the Department of Electronics, Telecommunications and Computer Science from the University of Aveiro [16]. The low-level control system is responsible for the motor control, servo-motor coupled to the steering mechanics and activate the LEDs for lighting the track. Simultaneously, the low-lvel control system is engaged in obtaining and processing the hodometry.

The communication between the low-level distributed system and the high-level control system, is supported by a *gateway* RS-232-CAN that converts the frames of each communication protocol enabling the high-level system, through the RS-232 communication protocol with a *baud rate* of 115200 bit/s receive information and disseminate instructions to the actuators on the same FTT-CAN bus. For the implementation of FTT-CAN, the module called *Master* is implemented to control synchronous traffic and to trigger the execution of tasks recommended by the module of motor control and hodometry [1].

Regarding the vision system endowing ROTA's robotic platform, it is composed of two cameras with a resolution of 640x480 pixel and a maximum *frame rate* of 30-fps. Both cameras are connected directly to the high-level control system via the Firewire port (IEEE 1394). One of the cameras positioned at the rear of the platform has a +25° *tilted* orientation, supposedly more adequate for image acquisition parallel to the plane of the screens where the light signals can be viewed as described in the subsection 2.2.1 dedicated to the description of the FNR-ADC. The second camera is located at the front of the robot, this time with a *tilt* orientation of -25°.

As far as the control system implementation is concerned, the ROTA team from the University of Leiria considers 3 blocks: vision, high-level control, low-level control. High-level control consists of the implementation of a *finite state machine* (FSM) as shown in the diagram in figure 2.10.

Regarding the algorithm dedicated to image processing, the ROTA team had a special attention to time-constraints and algorithm optimization, trying as much as possible

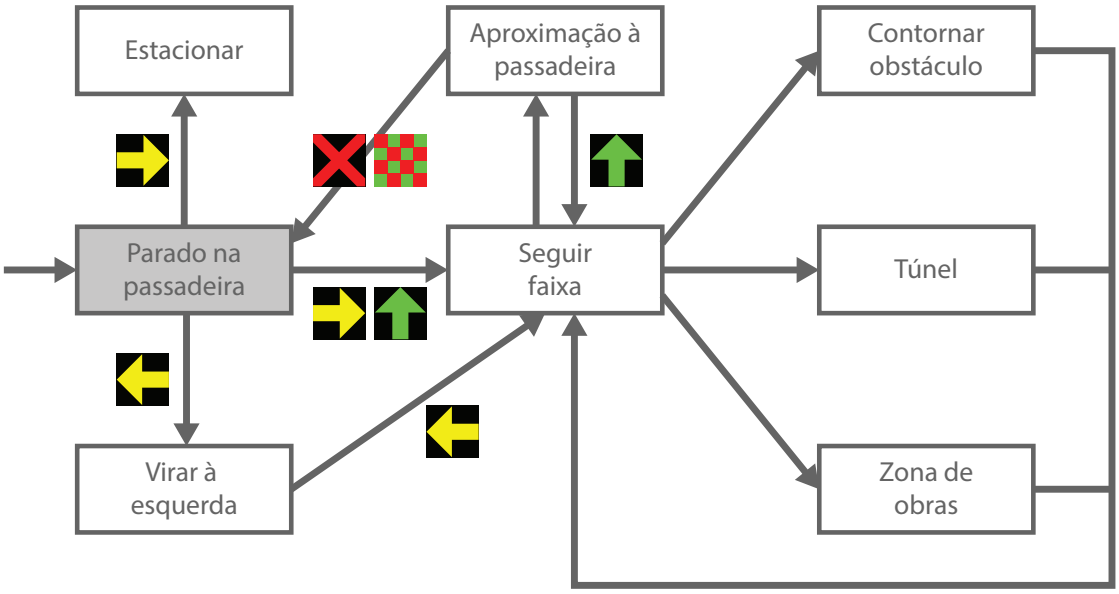


Figure 2.10: High-level control algorithm [1]

fast image processing and features extraction. To this end, the ROTA team opted for an image resolution of *320x240 px* permitting a higher *frame rate*, more suited to the computer system available. In this sense, some strategic concessions/decisions were adopted regarding the image processing. Starting by converting optical system's native color space (YUV422) to the HSV color space (*Hue, Saturation, Value*). This conversion arises from two relevant factors associated with HSV color space: color space similar to human perception of color and allowing the definition of color with only two values – *hue* and *saturation*. To turn more expedite the color space conversion, a *color look up table* was created where the conversion values for each triplet that defines the color space YUV are associated.

To minimize the impact on image processing speed, image segmentation (binary image conversion) is performed only to the most relevant chromatic values that are often present in the FNR-ADC scenario.

In terms of navigation, the ROTA team maintained the strategy of previous editions with Atlascar [17]. This consists of defining *target points* and a map of distances. In this approach, targets and distance map coordinates are calculated. The map of distances consists of a 320x240 matrix in which each position in the matrix (p_x,p_y) is an image point corresponds to a coordinate (x,y) in the world, having as reference front of the robot (in this case the center of the reference frame of the robot). It is therefore a *Look-Up Table*

(LUT) for the direct conversion of a point in the image to a point in the world in relation to the robot. All the movements contemplated in the displacement were then based on real distances. In the new approach adopted by the team, the concept of *distance map* is removed and the distances in pixel are now considered. The navigation process is based on a reactive process in which the vision system tries to evaluate the distance between robot and the centre of the lane and actuate over the steering whenever the robot moves away from that centre of the lane.

Finally, in order to obtain a fast algorithm, it is defined something similar to regions of interest *ROI* (*regions of interest*) but in a substantially simplified version. To avoid processing the whole segmented image, the identification of lane's delimiting limits is performed by scanning some image lines (*uaxis*). When doing this scanning, the algorithms search for transitions on the binary image, transitions from black to white and then to black again. As the speed of the robot increases - $v(x+k) > v(x)$ - this line (*vcoordinate*) is relocated on the image in order to correspond to a point in the world farther away from the robot, thus anticipating the need for possible changes in direction and acting on time.

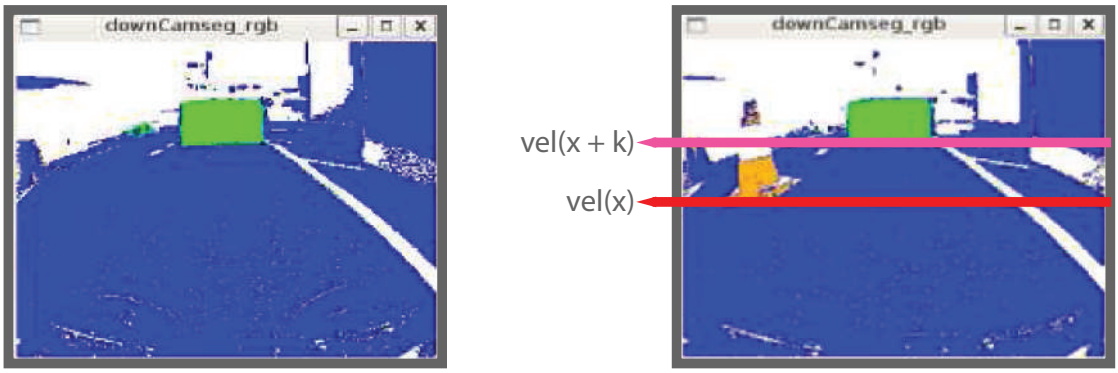


Figure 2.11: (left) Segmented image through a Color LUT; (right) Definition of lines of interest [1]

Regarding the location of the robot in the world, the method consists of calculating its coordinates *(x,y,θ)* through the combination of three variables:

- distance travelled from the crosswalk where the coordinate is considered *(0,0)* relatively to the world;
- robot's offset regarding the center of the lane;
- angular position relatively to the lane.

The distance traveled is calculated with the data from odometry. Given the track's blueprint symmetry, the system uses the crosswalk identification to restart the counting the travelled distance every half-turn. To control the count, the algorithm refers to the 21-metres that each half lap represents. In addition to the identification of the crosswalk to control the longitudinal distance travelled, the system also uses the identification of specific characteristics of the floor plan of the track to correct the positioning of the track.

Regarding robot's cross-track position relative to the lane width, the ROTA team advances with a solution they call *balancing*. This process is nothing but mediating *cross-track* error, which is the perpendicular offset of the robot in relation to the center of the track. The center of the track is identified through the analysis of image lines, sensitive to the transition of binary information from the segmented image. The transition from 0 to 255 or from 255 to 0 is assumed to have identified the lane delimiter. This process, which the ROTA team calls a *"line sensor"*. This technique performs its analysis (or "sensing") by scanning from the center to the left from the center to the right of image specific lines in $v-coordinates$. In this approach, some precautions are taken to increase robustness of the solution by dealing with some noise on the segmented image.

The ROTA team of the University of Aveiro has structured the navigation system in blocks with specific tasks associated. The tasks in a broad sense are composed of sub-blocks or basic/simple actions. These sub-blocks constitute tasks that can be as nuclear as changing lanes, progressing on the track or acquiring information on the light panel. This feature of the implemented solution allows better expandable properties of the algorithm with the integration of new features or simply greater clarity of the solution useful for debug.

As the robot progresses on the track, different driving modes are adopted:

- line search driving;
- line following driving;
- obstacle avoidance

In *line search driving* mode, as shown in figure 2.12, the system relies on the concept of "line sensor" to identify the track boundaries and, depending on the robot's cross-track positioning, calculate the turning angle to bring it to the defined strategy.

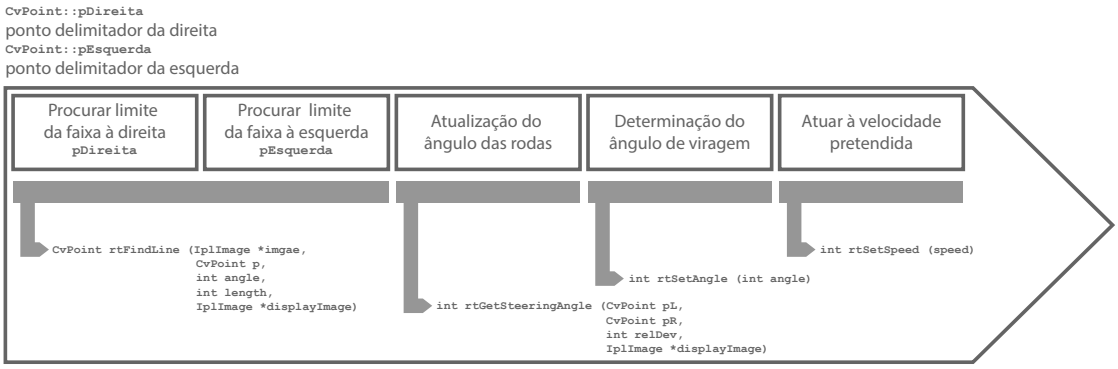


Figure 2.12: Line-searching driving mode [1]

In *line tracking* driving mode, the processing algorithm is more time performing when compared with *lane searching* driving mode because it abdicates of the *line sensors* to determine the position of lane delimiters. This driving mode is applied when continuous lines are present on the scene and uses the position information of the delimiters obtained in previous control cycles to infer their position in the next cycle. However, the *line tracking* mode requires that a continuous line is permanently seen on the right of the robot.

According to the ROTA team, the *line tracking* driving mode is mainly used in the first run of the autonomous driving test - *pure-speed* challenge. One of the challenges proposed in the FNR-ADC is the obstacle detection. The ROTA team from the University of Aveiro adopts the same concept of *line sensors* but this time these sensors are projected radially over the image with a common starting point centered on the image plane. These sensors progress in a radial way, with uniform angular spacing (see figure 2.13).

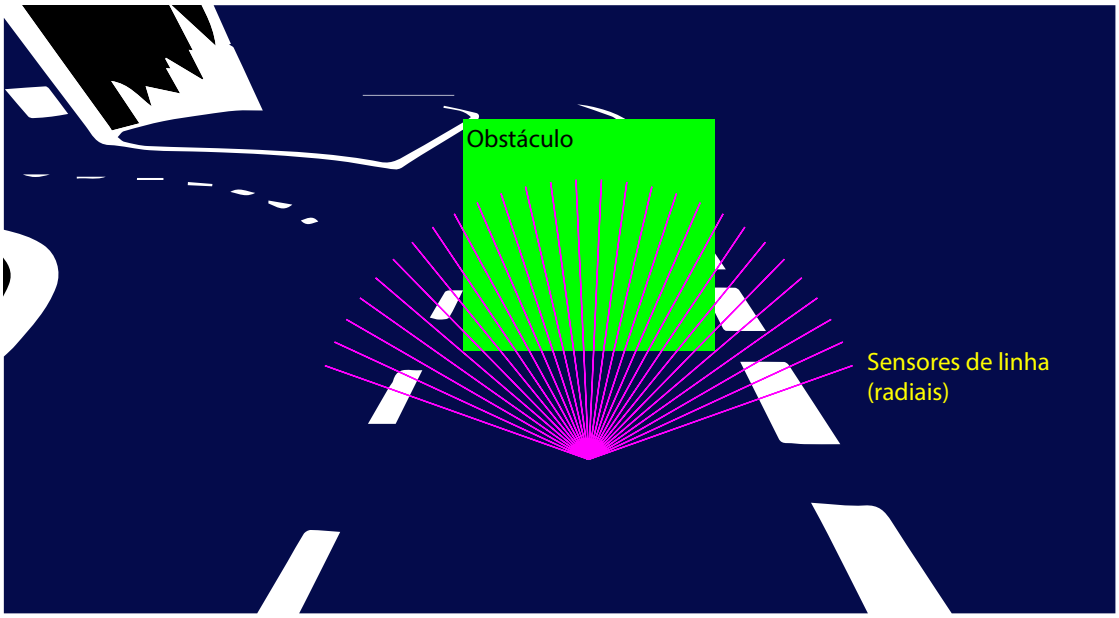


Figure 2.13: Radial *line sensors* schema

Radial *line sensors* evaluate the binary transition in the segmented image, this time with obstacle's green color segmentation. With the application of *thresholding*, the *line sensors* evaluate the transition from black to white, estimating the position of the obstacle. Depending on the evaluation made to its position, the system will return, undetected, detected and in the lane, detected and in the other lane or detected but uncertain position. As a result, the robot stops if it is too close to the obstacle, slows down or starts the obstacle deviation route. The evasive maneuver for deviation from the obstacle is performed using the *balancing* concept, i.e., the *cross-track* error is manipulated in order to force the robot to change lanes. Once the robots changes lane, the computer vision system recognizes/accepts this new lane, causing the system to stop handling the *cross-track* error. The system then counts a distance of 2 meters, defined as being sufficient to overpass the obstacle and restart the maneuver to bring the robot back into the initial lane.

2.2.2.3 CONDE team - Faculty of Engineering, University of Porto

One of the most successful robots in recent editions of FNR-ADC is the robot developed by the CONDE team from the Faculty of Engineering, University of Porto. However, despite its presence, there is little information available on the physical platform, in

particular on the hardware implemented. Nevertheless, the contacts established with Eng Armando Sousa permitted some enlightenment about their robot.

Figure 2.14: CONDE team - Faculty of Engineering, University of Porto

However, its inclusion in this state-of-the-art review is not particularly owed to their mechanical or hardware solutions but rather the algorithm developed to fulfill basic tasks expected in a robot designed to perform autonomous driving tasks, at the FNR-ADC.

As a physical structure, the CONDE robot is a basic differential steering platform with two drive wheels and a pivoting wheel at the rear. It features two commercial cameras PlayStation EYE that capture images with a resolution of 320x240 pixels at 30 fps. The optical group is used in *wide angle* mode, which corresponds to a *field of view* of 75°. However, CONDE team planned a customization of the optical group (lenses and their housing) which changes the *field of view* to a wider angle of 112°.

As for the computer system, initially low-power hardware was planned to equip the robot but they participated in the competitions with a laptop PC from which there are no technical specifications available.

In the plan of the proposed algorithm solution, two important aspects are highlighted, a fast track detection system and light signals so that it is compatible with the constraints of an online processing system, robust to the environmental variables of the test such as light conditions and precise to ensure a correct positioning and localization of the robot on the track [2].

Regarding the identification of the track delimiters, the CONDE team applies the concept of *zero copy/one pass* [18], process by which the relevant information contained in an image is obtained in the first processing pass, avoiding successive copies to auxiliary images or *kernels*. The process is organized in 6 steps as described in figure 2.15.

The process begins with the acquisition of an RGB image with 320x240 pixels. Later, points are extracted from the gray-scaled image, this is, coordinates of the image that may be associated with a range boundary line. In the next phase, the distortion caused

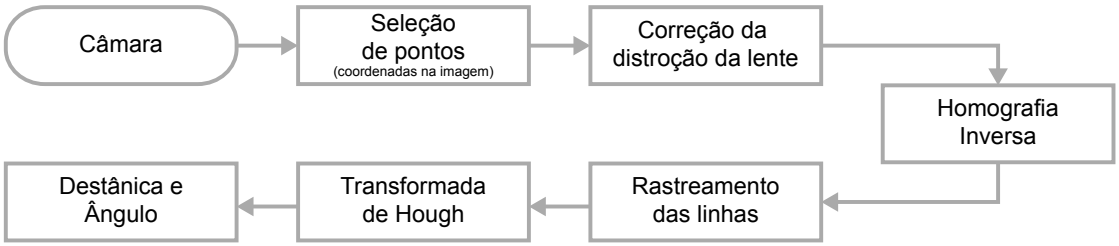


Figure 2.15: Block diagram of lane boundary tracking process

by the lens is corrected. In response to the time constraints typically on real-time solutions, the lens distortion correction is applied only to the image coordinates selected in the previous step [19].

The process of identifying the lanes continues with the Inverse Homography of the selected points and is followed by the Hough Probabilistic Transform [20]. At this stage, the tracking of lane's boundary lines is performed starting from the implementation of the Hough's Transformation available in the library *OpenCV* — `HoughLines()` and in the probabilistic version `HoughLinesP()`. The option for the Hough Probabilistic Transform is justified by using only a random selection of pixels from the image, which makes the line tracking process substantially faster when compared with `HoughLines()` version.

Proceeding to the next stage, the `HoughLinesP()` function returns a vector of detected lines. The algorithm analyzes this identified vector for the line with the shortest distance from the previously detected line and always within a tolerance limit. If no line meets the tolerance limit criteria, the line identified in the previous iteration prevails.

From the process of identifying the line that delimits the lane or track, information is extracted about the position of the robot — distance and angle — in relation to the lane in which it is supposed to move.

As mentioned, the proposed solution is based on the concept of *zero copy/one pass*, therefore the algorithm extracts this feature without the use of *kernels* as the common application of a Gaussian Filter after converting an image from RGB to *Greyscale*.

Regarding the recognition of light signals, the CONDE team equipped its robot with a second camera oriented upwards, dedicated to capture images from the signals shown by the standing screens. The proposed solution maintains the principle of *zero copy/one pass*. Starting from signalling specifications displayed on the screens (described in the section 2.2 on page 9), the solutions initiates by making a color segmentation of the image. This segmentation is done in yellow, green and red according to the equations 2.1.

$$\begin{aligned} f_R(x) &= max(0, min(x_R - x_G, x_R - x_B)/S) \\ f_G(x) &= max(0, min(x_G - x_R, x_G - x_B)/S) \\ f_Y(x) &= max(0, min(x_R - x_B, x_G - x_B)/s) \end{aligned} \tag{2.1}$$

Where:

$$S = x_R + x_G + x_B \tag{2.2}$$

After the image segmentation by color, the region of interest (ROI) is defined, taking into consideration the largest area for each of the segmented colors. Once the region of interest is defined, then the area is divided into four equal portions and counted the number of pixel inside each partition of the divided area as illustrated in figure 2.16. The information obtained feeds a decision tree that will identify the sign displayed on the screens.

Figure 2.16: The image acquisition, color segmentation and definition of region of interest (ROI) [2]

Based on this strategy, decision trees are created that evolve accordingly to the fulfilment of certain condition blocks, such as *STOP* through the set of conditions illustrated in figure 2.17.

2.2.2.4 Remarks

This chapter included a research of solutions with a successful history in the last editions of the Autonomous Driving Competition at Festival Nacional de Robótica. It will be from the solutions presented, and also presenting other approaches to answer similar challenges, that this dissertation report will continue in the following chapters. Other solutions will be addressed to respond to the constraints in real-time solutions, not only

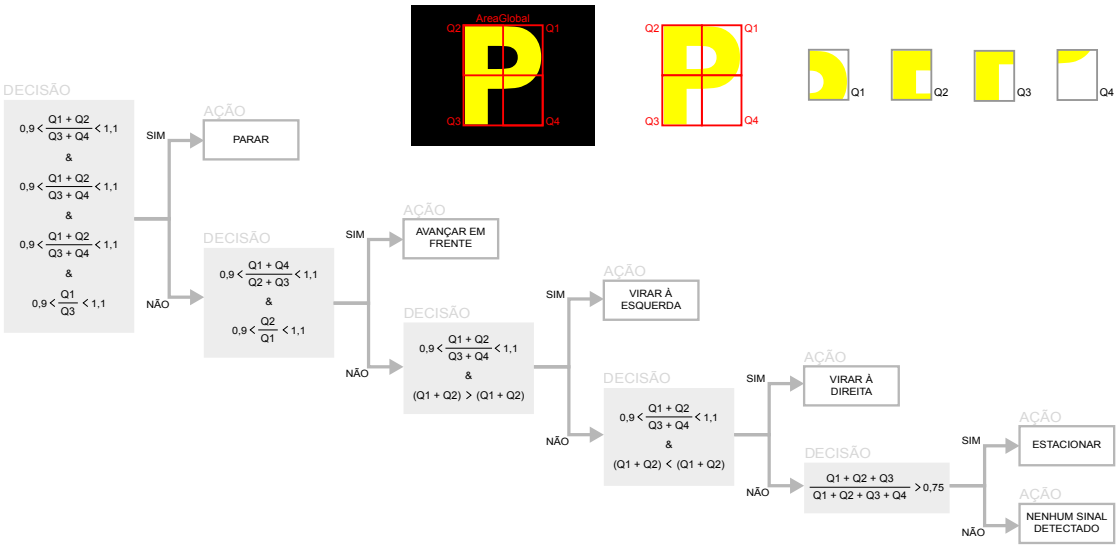


Figure 2.17: High-level control algorithm

for the tracking of lines and detection of lanes but also, trajectory planning, control and *machine learning*.

This dissertation will also tackle, in the next chapters, the importance and pertinence of making available a solid simulation environment as an instrument to develop and do preliminary testings of algorithms addressing solutions to specific autonomous driving tasks.

Chapter 3

Project and High-level Architecture

In this chapter the project is defined with a high-level overview of the systems proposed to implement. It is also given focus to the simulation tool also developed in the scope of the Autonomous Driving Competition from the Festival Nacional de Robótica.

The autonomous driving topic, beyond the mechanical and hardware issues, is a true challenge in algorithmic science, demanding solutions to extremely wide subjects such as perception, localization, camera-based localization, obstacle mapping, road mapping (creation and representation), tracking of moving objects, traffic light detection and identification, traffic signal detection and identification, pavement marking detection and identification, decision making and route planning, path-planning and motion-planning, control (path-tracking and actuators control) [21]. Disregarding mechanical strategies or how robust a platform is, an autonomous driving solution, and in particular, those solutions competing at FNR-ADC shall respond to five major blocks constituting an autonomous driving vehicle system [22]:

Figure 3.1: Major blocks forming an autonomous driving system

So the system developed in the scope of master degree in Autonomous Driving Systems, ministered by the Laboratory of Autonomous Systems / Department of Electronic

Engineering from Polytechnic Institute Engineering of Porto, can be synthesized through the following high-level architecture:

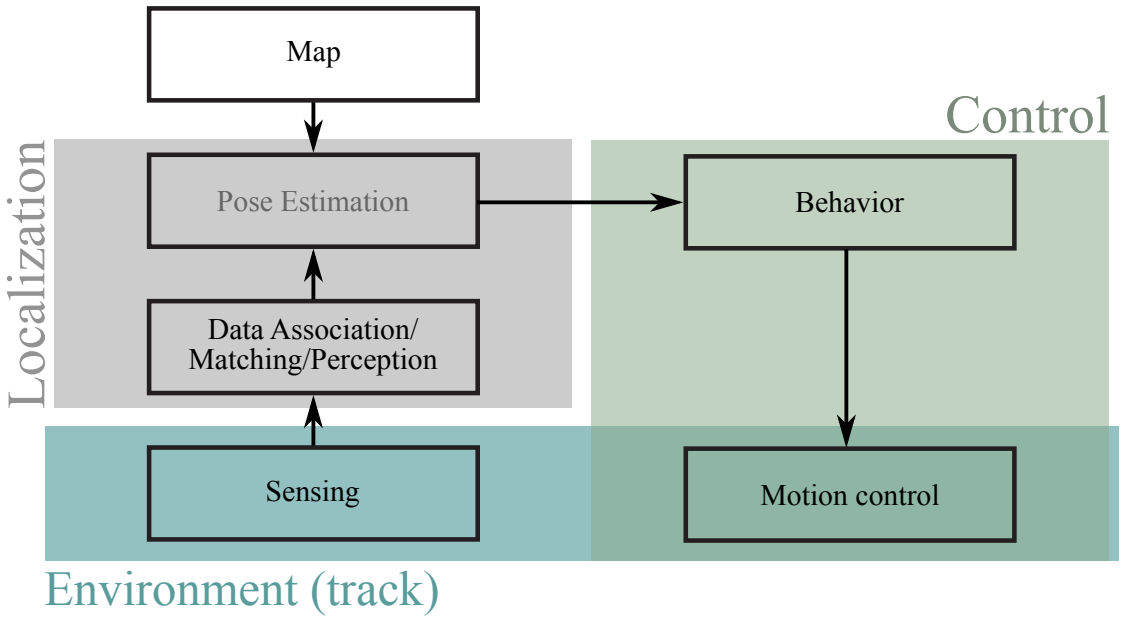


Figure 3.2: High-level architecture

The robotic platform considered in the development of this study is a low budget 4-wheel Skid-Steer (4WSS) - see figure 3.3. Due to the absence of a directional steering system, the change of heading processes by applying different velocities on the left and right wheels, resulting in high maneuverability and mobility in different terrain conditions. However, the change of heading implies slippage/skidding, which represents important challenges to obtain an accurate and reliable dynamic/kinematics model and subsequently compromising its controlability.

In this dissertation, possible solutions will be addressed answering to some of the blocks stated above like computer vision (system configuration and features extraction), trajectory-following taking into consideration the peculiar non-holonomic characteristics of the platform (tracking and obstacle avoidance), behavior and control.

At last, but equally important, the solutions here proposed were developed and tested in simulation environment especially designed as an important instrument capable to replicate on its full extent the Autonomous Driving Competition from the Festival Nacional de Robótica [23].

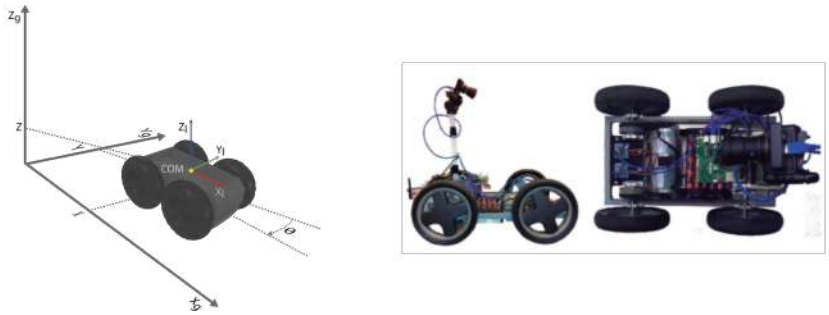


Figure 3.3: 4WSS Platform - (left) Robot model and Reference Frame; (right) Real platform

FNR-ADC Simulation Environment

The FNR-ADC Simulation Environment is built on top of Modular Open Robots Simulation Engine (MORSE), an open-source simulator engine based on Blender Game Engine [24]. The usage of MORSE brings important advantages like the possibility to model components, environments, and robots in Blender. Blender is a free and open-source 3D creation suite, supporting the entirety of 3D pipeline—modeling, animation, simulation, rendering. Figure 3.4 illustrates the work developed to simulation the real competition specifications.

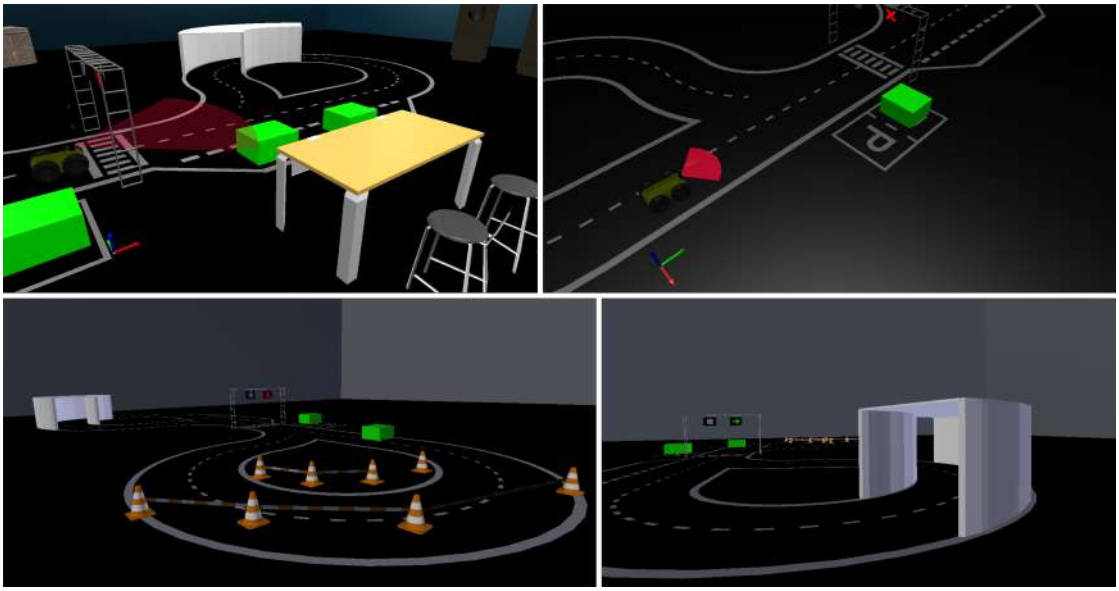


Figure 3.4: FNR-ADC Simulation Environment - Competition Specifications

Another important advantage using MORSE simulation engine is the access to Blender's 3D Game Engine (see figure 3.5). The possibility of supporting multi-robots simulations, large simulation scenarios, and better use of computational resources, are important advantages of MORSE. Finally and equally important, MORSE offers a clean installation process when compared with other simulator engines [25] [26].

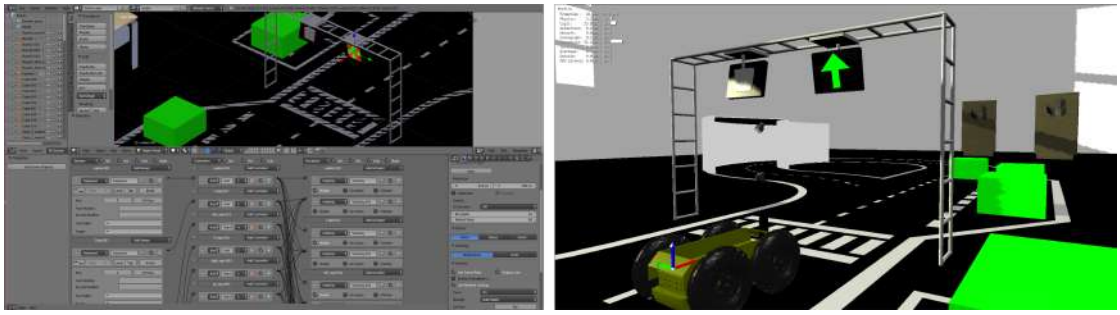


Figure 3.5: FNR-ADC Simulation Environment - Blender's Game Engine

The simulation here presented and used to test the algorithms proposed, validating the concept and its applications, as well as a test of the simulation environment as sandbox where solutions are designed, developed and tested prior their implementation to real platforms.

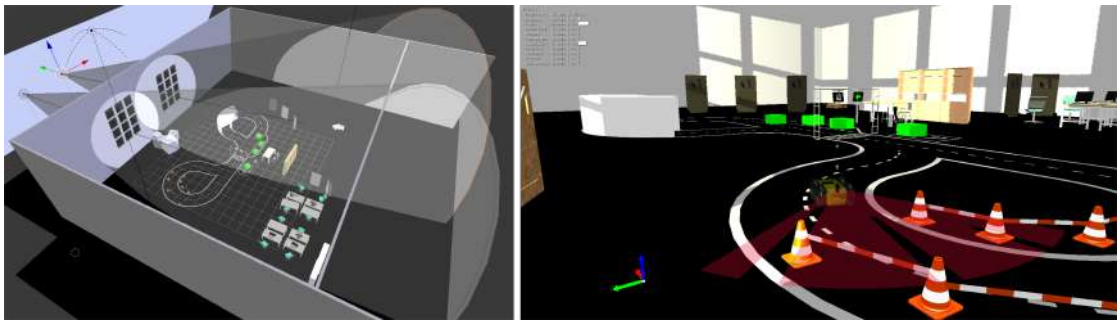


Figure 3.6: FNR-ADC Simulation Environment - full scenario set

Here beneath is described be interacting blocks that will constitute the simulated solutions on all its components. The algorithm, developed in the scope of FNR-ADC will use *Robotic Operating System* (ROS) framework and *Open Source Computer Vision Library* (OpenCV) an open source computer vision. The figure illustrates the ROS nodes subscribing and publishing information of relevant data stream that will support the algorithms output.

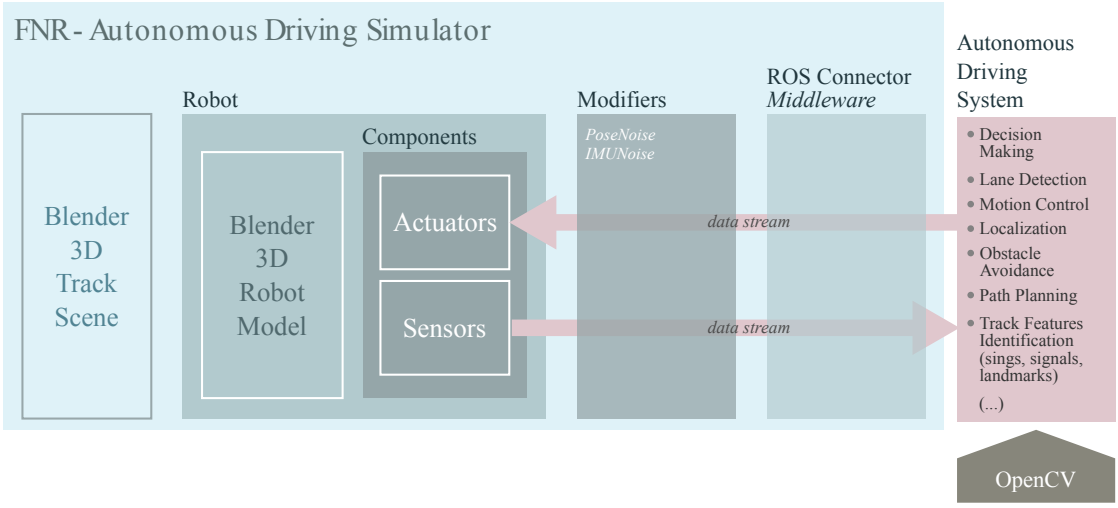


Figure 3.7: Autonomous Driving System - simulation and system interacting blocks

The following figure presents an overview from the FNR-ADC Simulation Environment ROS computation graph, with all publishing topics accessing sensors and subscribing information required to control the robot. Moreover, a leaf node available with information about the track's map, a cost map provided by ROS Map Server.

The modules proposed to be implemented will make use of those highlighted topics as illustrated in figure 3.8, with a more detailed intercommunication and relation between the topics to be developed (figure 3.9).

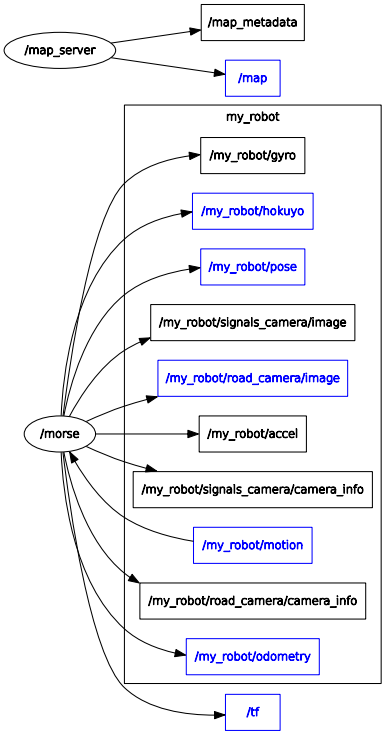


Figure 3.8: FNR-ADC Simulation Environment - data stream available (sensors and actuators)

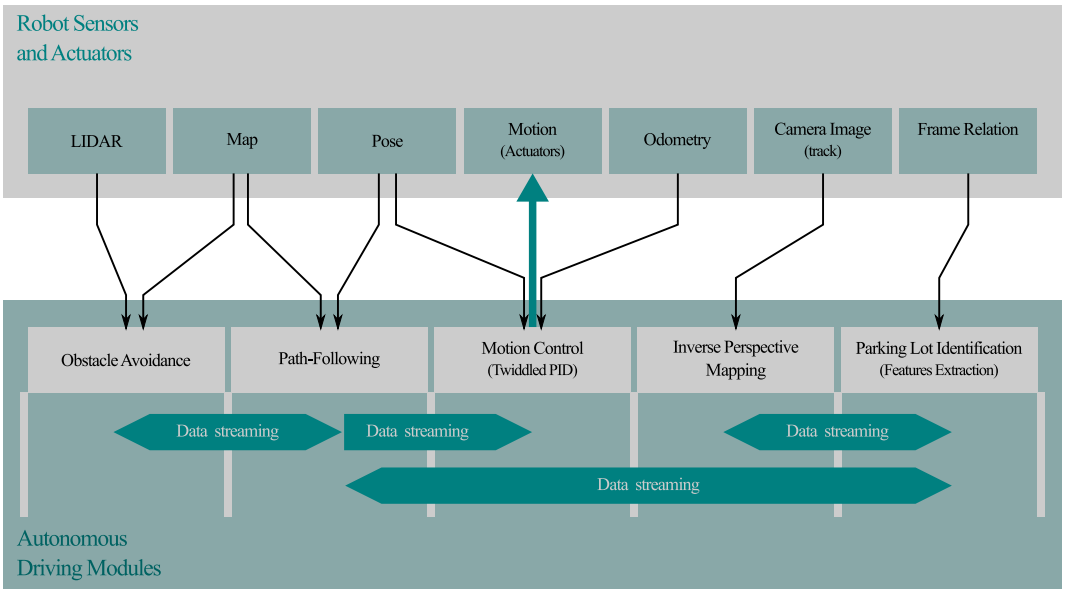


Figure 3.9: Projected blocks and communication with simulator

Chapter 4

Theoretical Concepts

In this chapter a theoretical study of important topics and modules integrating an autonomous driving robot such as, Inverse Perspective Mapping for lane-detection and land-mark detection, Classification and Machine-Learning of to detect features on the track, Motion-Control and Path-Following to define pure maneuvers, Control as a behavioral management of the robot.

4.1 Inverse Perspective Mapping

Computer vision is an essential block to any autonomous driving system. The first steps regarding the autonomous driving concept was precisely on the *lane detection* or *lane departure*. In fact these components are the support of modern Advanced Driver Assistance Systems (ADAS).

Today ADAS developed and implemented into commercial solutions are mostly based on systems focused on the forward direction of the vehicle, this means that the major feature to extract lane markings, detecting road boundaries and in exceptional cases assisting on the detection of pedestrians. David Schreiber [27] proposed a robust lane detection algorithm using a forward-looking monocular camera. Later, Joel C. McCall and Mohan Trived [28] designed a lane departure system with a road model based on parabolic approximation on a flat plane the tracking of the lane is done in Kalman filtering, estimating lane marking on the road. Making use of homography defining one-to-one relation between two coordinates system, Ho Jung [29] supported their design of an automatic parking system with a monocular rear camera. Finally Yu-Chih Liu [30] proposes a vehicle surrounding monitoring system using the bird's-eye concept which is implemented using homography transformations over fish-eye monocular cameras. The

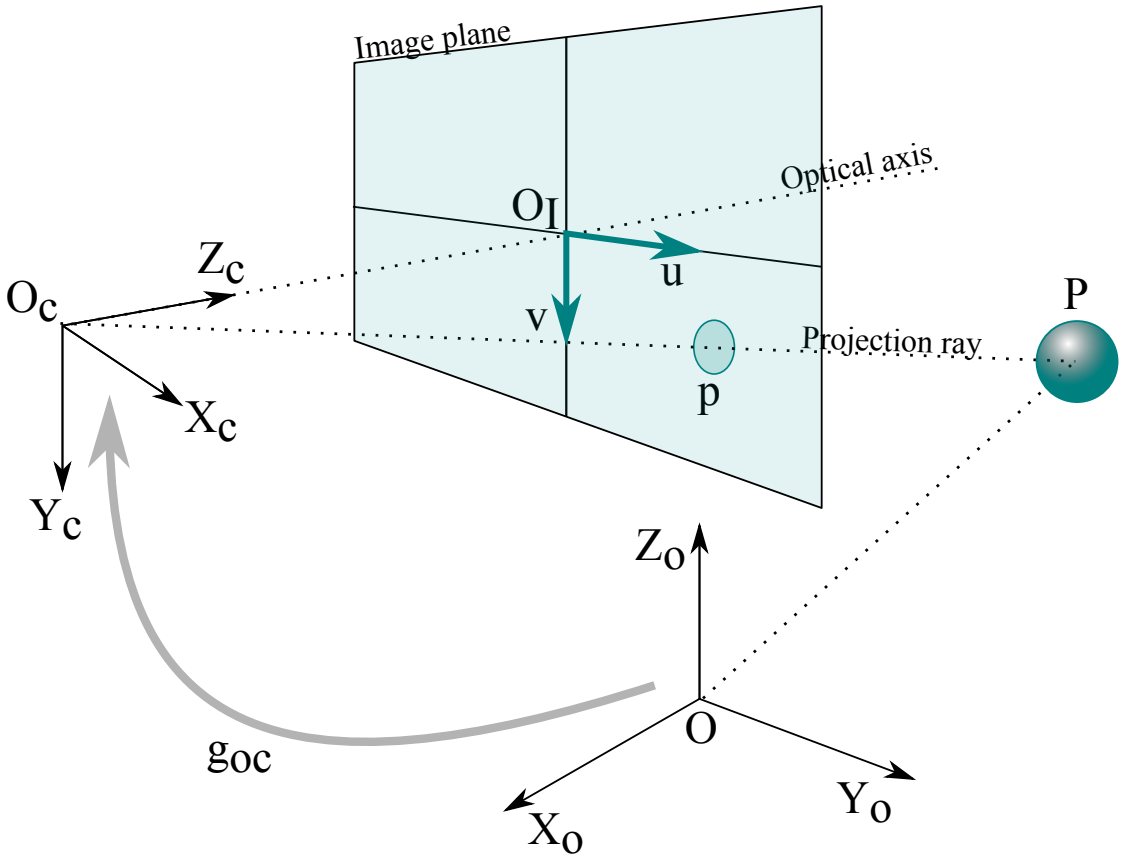


Figure 4.1: Pinhole camera model

use of Inverse Perspective Mapping (IPM) has been widely used on system designs of several ADAS.

The IPM is a technique typically used to remove perspective distortion of a 2D image captured from a 3D scene. This distorted projection, caused by a foreshortening factor, has its genesis on the perspective projection geometry. An explanation is provided based on the pinhole camera model is figure 4.1.

In the schematics above z_c in the camera reference frame is pointing forward in relation to the image plane (u, v). The captured image of an object in the world reference frame $F_w = 0_w; X_w, Y_w, Z_w$ is being projected in its inverse orientation, perpendicular to the optical axis and lies at a focal length f from 0_w.

The image plane is defined by a 2D reference frame $F_i : 0_i; u, v$ whose axis are parallel to X_c and Y_c respectively. The captured image position can be described by the expression 4.2.

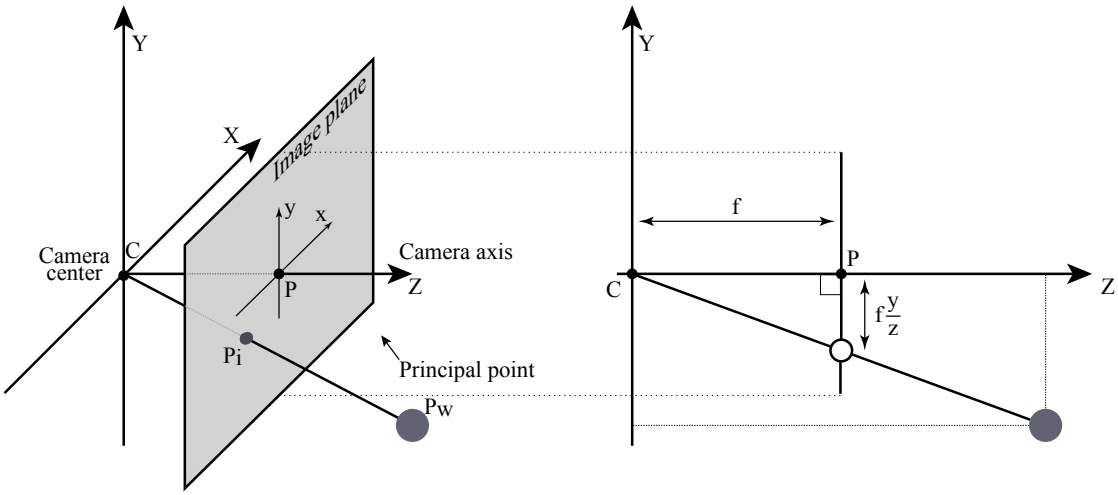


Figure 4.2: Model of camera and image frame

$$\frac{u}{x} = fz \tag{4.1}$$

Taking the expression above and expanding it to image's mathematical components, (u, v) is given by the expression:

$$(u, v) = (f\frac{x}{z}, f\frac{y}{z}) \tag{4.2}$$

The 3D pose of a moving camera's reference frame F_C in relation to the world reference frame F_O is the transformation matrix *goc* (figure 4.1).

The transformation of 3D camera coordinates into its corresponding 2D point on the image plane, both represented in homogeneous coordinates, is solved from equation 4.2, achieving a geometrical relation between these to points - *camera calibration matrix*:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{pmatrix} \mathrm{f} & 0 & 0 & 0 \\ 0 & \mathrm{f} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{4.3}$$

Not always the origin of coordinate system is coincident with the main point $(p_x, p_y)^T$, in this case the offset is considered in camera's *intrinsic* properties:

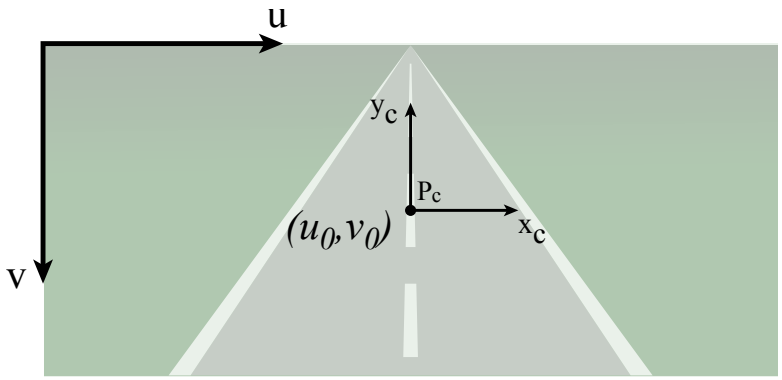


Figure 4.3: Offset of camera center and image frame

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} \mathrm{f}X + Zp_x \\ \mathrm{f}Y + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} \mathrm{f} & 0 & p_x & 0 \\ 0 & \mathrm{f} & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{4.4}$$

The relation between 3D camera coordinates and its scaling into the respective image formed is association the the pixel resolution m_u and m_v which corresponds to the number of pixels per millimeter or inch. The equation 4.4 relates a P_c expressed in millimeters into p_i expressed in pixels.

$$(u, v) = (m_u(f\frac{x}{z} + p_x), m_v(f\frac{y}{z} + p_y)) \tag{4.5}$$

The five camera's *intrinsic* parameters [31] or internal parameters are intimately related with the sensor and optical properties of the camera and these properties can be represented by the matrix K:

$$K = \begin{bmatrix} \alpha_x & \mathrm{S} & \\ 0 & \alpha_y & \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Where α_x and α_y is the product of the focal length f with size of an element of the camera image, given in pixels/millimeter m_u and m_v. Pixels are usually square and the sizes of the x and y generally are equal. The elements u_0 and v_0 represent

the misalignment between the optical center and the coordinate center in the projected image, and both are obtained by the product of offset p_x and p_y with m_u and m_v respectively. Finally $S = f_y \tan(\omega)$ represents skew-ration which, for the majority of the cameras available this parameter tends to zero.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{4.7}$$

The relation between the 3D coordinates in the camera's reference frame and the 3D coordinates in the world reference frame is given by the *extrinsic* parameters which represent the relative position of the camera to the world reference frame where the object is physically located. The *extrinsic* parameters are composed by six external parameters: three rotations in *roll*, *pitch* and *yaw* represented by the matrix R and translations in X, Y and Z represented by a translation vector t [31].

As said before the *extrinsic* parameters describe the location of the camera in the world reference frame and its attitude in relation to the world. The *extrinsic* matrix assumes the form of a rigid transformation: a $3x3$ rotation with a right block $3x1$ with the translation vector:

$$\left[R \mid t \right] = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array}\right] \tag{4.8}$$

The *extrinsic* matrix defines the transformation of 3D world coordinates into 3D camera coordinates. It is possible to consider translation vector t as the camera's position relative to the world's reference frame center. Additionally, the rotation matrix R represents world reference frame orientation, taking into consideration the camera's coordinate system.

Contrary to a more intuitive interpretation, the *extrinsic* matrix describes how the world is transformed relative to the camera reference frame.

The straightforwardness of the relationship between camera's pose and *extrinsic* parameters can be proofed as follows: taking as starting point the rigid transformation describing camera's pose in relation to the world reference frame, where the translation vector C as the camera's position relative to the world's reference frame origin, and R_c

the attitude of camera's coordinate system relative to the world we have:

$$\left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right] \tag{4.9}$$

Taking into consideration the matrix above describes camera's pose relative to the world, the *extrinsic* matrix is the inverse of matrix 4.10:

$$\left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right]^{-1} \tag{4.10}$$

Decomposing and solving the inverse matrix 4.10:

$$\left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right] \tag{4.11}$$

$$\left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right]^{-1} = \left[\left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right] \quad \left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right]\right]^{-1} = \left[\left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right]^{-1} \quad \left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right]^{-1}\right] = \tag{4.12}$$

$$\left[\left[\begin{array}{c|c} R_c^T & 0 \\ \hline 0 & 1 \end{array}\right] \quad \left[\begin{array}{c|c} I & -C \\ \hline 0 & 1 \end{array}\right]\right] = \tag{4.13}$$

$$\left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline 0 & 1 \end{array}\right] \tag{4.14}$$

So the *extrinsinc* matrix parameter are

$$R = R_c^T \tag{4.15}$$

$$t = -R_c^T C \tag{4.16}$$

If P_w represents the homogeneous coordinate of a point in the world and p_c represents the homogeneous coordinates of the projection of that same points into the camera reference frame, then these two points are related through the equation:

Figure 4.4: Object represented in Euclidean geometry (Left) and object represented in Perspective geometry (right)

$$p_c = \left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline 0 & 1 \end{array}\right] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline 0 & 1 \end{array}\right] P_w \tag{4.17}$$

The camera projection matrix – $P = K[R|t]$ takes into consideration both *intrinsic* and *extrinsic* parameters, permitting the mapping between 3D points in the world and the 2D points in the image [32]:

$$s\widetilde{m} = K \begin{bmatrix} R & t \end{bmatrix} \widetilde{M} \tag{4.18}$$

Where $\widetilde{m}$ represents the homogeneous coordinates of a point on the image plane, and $\widetilde{M}$ represents the homogeneous coordinates of that point in the world coordinates. s is an arbitrary scale factor. (R, t) are the extrinsic parameters – rotation and translation – and K represents the intrinsic parameters.

4.1.1 Perspective Image

The Euclidean geometry describes the objects as they are, whose properties such as lengths, angles and parallelism, remains unchanged independently of the rigid motion affecting that object. By the other hand, Perspective geometry describes the objects as they appear, whose lengths, angles and parallels appear "distorted" when we look at object.

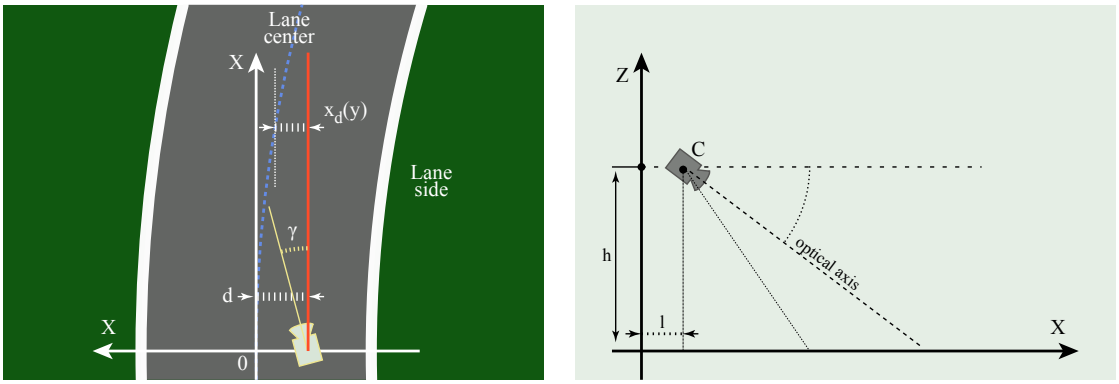


Figure 4.5: Geometrical model of camera system

Taking the example of an image capture from a road as illustrated in image 4.4, in the actual shape lanes are rectangular and their limiting lines are parallel. Dashed line dividing the track are evenly spaced along the tracks. However in their perspective appearance, lanes converge to a point on the horizon, the dashes on the dividing line become closer and closer towards the horizon.

4.1.2 Model plane

The *Direct Perspective Transformation* would relate the coordinate of a point in the world reference frame into the image plane performed by the following closed form equations.

$$u = \frac{f\big((x - X_0)\cos\Theta + (y - Y_0)\sin\Theta\big)}{-(x - X_0)\cos\Psi\sin\Theta + (y - Y_0)\cos\Psi\cos\Theta + (z - Z_0)\sin\Psi} \tag{4.19a}$$

$$v = \frac{f\big((x - X_0)\sin\Psi\sin\Theta + (y - Y_0)\cos\Psi\cos\Theta + (z - Z_0)\cos\Psi\big)}{-(x - X_0)\cos\Psi\sin\Theta + (y - Y_0)\cos\Psi\cos\Theta + (z - Z_0)\sin\Psi} \tag{4.19b}$$

4.1.3 Inverse Perspective Mapping (IPM)

The objective of the *Inverse Perspective Mapping* method (IPM) is to remap each pixel of the image in 3D space and produce a new image in 2D space. This new image represents a top-view of the track, a commonly referred to as ”bird’s-eye-view”, eliminating the perspective distortion characteristic of the perspective image.

The process to correct the image form perspective distortions using the IPM requires, beforehand, the knowledge of *camera projection matrix* as stated in equation

number 4.18 – the intrinsic and extrinsic parameters.

In the scope of autonomous driving systems, the usage of top-view images from the track is particularly useful on lane detection. Some of those benefits are the correction of the perspective effect. So, lanes that appear to converge to a vanishing point in the horizon line, are now vertical and parallel (assuming the lanes are parallel or close to that). Another important benefit is to limit the processing only to a sub-region of the input image which reduces the run-time. To obtain the IPM from the input image it is assumed the track is flat. There are additional work to apply IPM on non flat track surfaces but once the problem herein approach only contemplates the track laying flat world plane $Z = 0$.

The IPM can be described as projection of an object from a 3D Euclidean space $W = (x, y, z)^3$ into a 2D planar frame $I = (u, v)^2$ respecting the geometric properties of the 3D object is the world space. From the geometrical model representations in figure it is possible to derive into a two equation (4.20) IPM model using triangulation and trigonometry [32].

$$u(x, y, 0) = \frac{2\alpha}{n-1}\big(\psi(x, y, 0) - (\Psi - \alpha)\big) \tag{4.20a}$$

$$v(x, y, 0) = \frac{2\alpha}{m-1}\big(\theta(x, y, 0) - (\Theta - \alpha)\big) \tag{4.20b}$$

Where $\gamma = \tan^{-1}\left(\frac{y}{x}\right)$ and $\theta = \tan^{-1}\left(\frac{h}{\sqrt{x^2+y^2}}\right)$, as Θ representing Yaw angle of the camera regarding to the world frame and Ψ representing camera's $Pitch$ angle relative to the world frame. Typically these two parameters remain constant for system simplicity in detriment of its robustness [33]. Additionally α represents camera's $field-of-view$, and m and n represents camera's resolution.

Based on the flat world concept, the world reference frame is defined $F_w = 0_w; X_w, Y_w, Z_w$, the camera reference frame is defined $F_c = 0_c; X_c, Y_c, Z_c$ and the image frame is defined $F_i = u, v$. The point $C = (x_c, y_c, z_c)$ represents the camera location in the world frame F_w. The knowledge of camera geometry, estimated from a camera calibration process, among other techniques, in respect to the local planar track scene permits to recover a 2D image form the locally imaged track – $S = (x, y, 0) \in F_w$ donate the track surface and the correspondence to F_i is given by the follow mapping.

$$(x, y, 0) \in S \rightarrow (u, v) \in F_i \tag{4.21}$$

A point in the scene represented in world frame coordinates corresponding to an

image point can be calculated either by knowing the distance between the camera frame and that point in the scene, or any other point in world coordinates – ($Z = 0$).

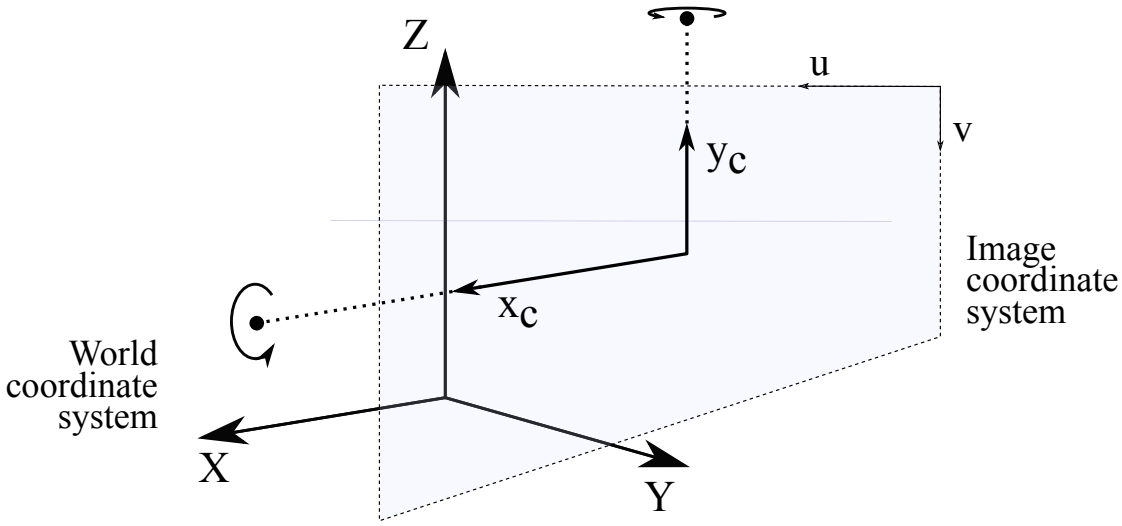


Figure 4.6: World and Image Coordinate System

So deriving from the equations in close form 4.20, and again assuming all points lies on the floor ($Z = 0$), the *Inverse Perspective Mapping* is calculated as follows.

$$u^* = X_0 - \frac{Z_0 x_c \cos\Theta + (y_c \sin\Psi - f\cos\Psi)(Z_0 \sin\Theta}{y_c \cos\Psi + f \sin\Psi} \tag{4.22a}$$

$$v^* = Y_0 - \frac{Z_0 x_c \sin\Theta + (y_c \sin\Psi - f\cos\Psi)(Z_0 \cos\Theta}{y_c \cos\Psi + f \sin\Psi} \tag{4.22b}$$

Where (X_0, Y_0, Z_0) are the camera's position in the world reference frame a the moment when the scene was taken; f is the focal length.

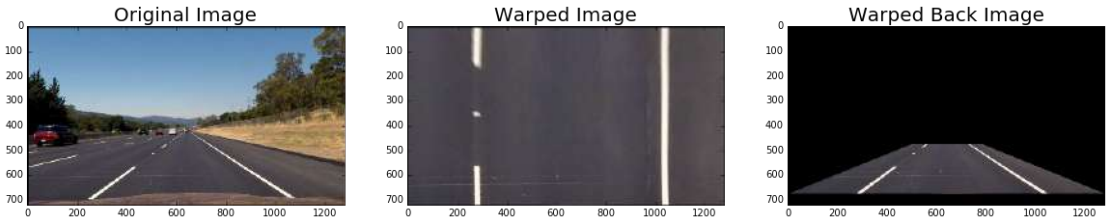


Figure 4.7: Transformation from original image to IPM [3]

4.2 Classifiers – Machine-Learning

4.2.1 Naive Bayes

The *Naive Bayes* classification method is a classification technique based on *Bayes' Theorem* and lying on the assumption of independence between *predictors*. In a simplistic approach, the *Naive Bayes Classifier* assumes that the presence of a certain characteristic or feature in a class is not related to the presence of any other kid of feature or characteristic, even if those characteristics depend on each other, or depend on the existence of third-party features. The set of all these properties contribute independently to the predictor probability [34].

Suppose we have two number generators following a normal distribution. If a number k is generated, we may be capable to predict which of the generators produced that number by predicting which class c best corresponds to the x features.

The Bayes Theorem offers the possibility to calculate the probability for a given random event, the conditioned probability for a given feature when a relevant observation is taken into account in the classification process.

$$\text{argmax } P(c|x) \ : \ c \in \{1, ..., k\} \tag{4.23}$$

This way, it is possible to infer *posterior probability* $P(c|x)$ of class c from the *predictor* of x (features), taking into consideration class prior probability $P(c)$ and the feature predictive given a certain class $P(x|c)$, normalized by the prior probability of the predictor $P(x)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{4.24}$$

However, it is frequently difficult to compute $P(x|c)$, but the *Naive Bayes* classifier simplifies the process by supposing the features are totally independent from their class.

$$P(x|c) = \prod_{i=1}^{n} P(x_i|c) \tag{4.25}$$

Naive Bayes model is simple to implement and particularly useful in scenarios involving a large volume of data to process. The model has a fast algorithm that reveals particularly suited to deal with real-time applications.

As main advantages of *Naive Bayes*:

- Simple and fast method to predict a set of data from a testing class. It also have a good performance when predicting multiple classes;
- When the the preposition of independent features prevails, the *Naive Bayes* classification model has better performance when compared with other classifiers. This model also requires less learning data;
- Fair performance when classifying categories when compared with numeric classification. In these cases it is assumed a Gaussian distribution.

Main disadvantages:

- If a classifying category among the test dataset, was not observed during the training process, that category will assume zero probability (*Zero Frequency*) and the algorithm won't be able to do any estimation. Some techniques are sued, such as the *Laplace smoothing* or *additive smoothing*;
- *Naive Bayes* estimator is poor and its information shall not be considered relevant;
- The hypothesis of true independent predictors is hard to be found in real-world scenarios, where nearly all predictors have some sort of dependency even if it is a light one.

4.2.2 K-Nearest-Neighbors – KNN

The $K-Nearest-Neighbors$ algorithm is a supervised learning classification algorithm, popular in data-mining and machine-learning systems. The classifier learning process is supported in the concept of proximity, likelihood, and *neighborhood*. It has a starting point from a set of labeled features and uses them to learn how to label other new features.

The learning dataset is composed of n-dimensional vectors and each element of that vector is a point in n-dimensional space.

In the KNN model, to determine the class of a given feature that does not appear in the learning dataset, the classifier searches in the dataset for k elements that are closer to the unknown element (more concretely with shorter distance). These elements are called $k-neighbor$ and the assigned class will be the one that appears most frequently.

There are several methods to calculate the distance between the elements: Manhattan, Minkowski or Euclidean, the latter being the most frequently used.

Consider $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ two points in $\mathbb{R}^2$

1. Manhattan distance between X and Y:

$$d(x, y) = \|x_1 - y_1\| + \|x_2 - y_2\| + ... + \|x_n - y_n\| \tag{4.26}$$

2. Euclidean distance between X and Y:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \tag{4.27}$$

3. Minkowski distance between X and Y:

$$d(x, y) = \sqrt[n]{(x_1 - y_1)^n + (x_2 - y_2)^n + ... + (x_n - y_n)^n} \tag{4.28}$$

However, Minkowski's distance is a generalization of the distance from Manhattan and Euclidean, when q=1 the distance from Minkowski represents the distance from Manhattan, and when q=2 assumes the Euclidean form.

There is also possible to each variable associate an weighting factor related to its importance, and in this sense the Euclidean weighted distance assumes the format stated through equation 4.29:

$$d(x, y) = \sqrt{w_1(x_1 - y_1)^2 + w_2(x_2 - y_2)^2 + ... + w_n(x_n - y_n)^2} \tag{4.29}$$

However, this analysis method can easy escalate in such a way that can compromise real-time applications. Assuming $N - samples$ in $D - dimensions$ the complexity of calculating the distances increases in a rate of DN^2 [35].

For performance optimization purposes, the KNN classifier offers the user a free and controllable element ($K - elements$).

Despite in certain circumstances the KNN algorithm might be computationally heavy, its performance is adequate on some real-time applications, when the range of data to classify is not wide.

In those scenarios with much data to classify and to classify with, with the implication in the loss of computational performance, some patches may help to reduce this computational effort, namely the possibility to let the user define hyper-spheres where the algorithm shall look to and compute only these elements lying inside the sphere. The risk is the possibility of getting one or more empty spheres.

There are other examples of techniques to improve computational performance by reducing the number of distance calculations. One of those improvements is the $K - DTree$ technique. Simplistically, this technique consists on a basic idea that if a point A is far from point B and close to point C then it is fair to conclude that points B and C are far one from another. This reduces the computational cost to $DN * \log(N)$ which is a significant improvement compared with the standard algorithm [35].

The main advantages of $K - Nearest - Neighbors$:

- The KNN algorithm is simple to implement;
- It is not necessary to modeling the classifier;
- Reduced number of tuning parameters;
- Versatile algorithm: classifier, regression and search.

Main disadvantages:

- Great computational effort and efficiency loss as the number of predictors and elements increase.

4.2.3 Support Vector Machine – SVM

The classifier Support Vector Machines (SVM) is a discriminating classifier specially oriented to solve problems where pattern recognition is needed. Initially introduced by Vladimir Vapnik and Alexey Chervonenkis, the concept of this algorithm lies in the definition of an optimal hyper-plane for linearly separable patterns, which categorizes new features. In 2D space, the hyper-plane is a line that divides the space into two parts attributed to each class.

The SVM algorithm has been extended in order to define patterns that are not linearly separable, by using transformations of original data into a new space through a *kernel* function.

The SVM algorithm computes a line or a plane depending if dealing with a problem with two or more dimensions. The SVM classifier is mainly a geometric approach to the classification problem [36].

Consider N training elements $\{x_i, y_i\} : i = 1, 2, \ldots, N$ where $x_i \in \mathbb{R}^M$ is a vector representation of a trained object and $y_i \in \{-1, 1\}$ its class, given an unknown probability distribution $Pr(x, y)$ from where data from training are picked.

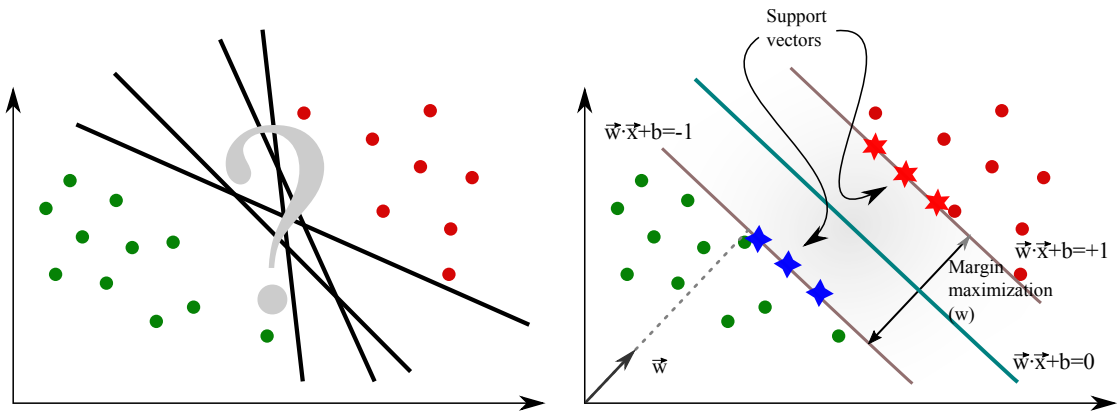


Figure 4.8: Support Vector Machine - Hyper-plan optimization

Training the classifier endows the system with the ability of mapping $x \mapsto y$ through training examples $\{x_i, y_i\}$ permitting the classification of new data that follows the same probability distributions from previously trained examples [36].

The *support vectors* consist in the data points that lie close to the hyper-plane. The $hyper - plane$ is the decision surface. The SVM finds the optimal solution for the hyper-plane that separates classes.

Support Vectors are critical elements from the training set. Those elements would change positions if the hyper-plan is removed. The process of defining the optimal hyper-plane is characterized as an optimization problem that could be solved with optimization techniques like the use of Lagrange multipliers which permits to find the maximum and minimum of a multi-variable function.

Considering D training elements $\{x_i, y_i\}$, $i = 1, 2, ..., D$ where $x_i \in \mathbb{R}$ is a vector representation, and $y_i \in \{-1, 1\}$; its corresponding class, it is possible to define an hyper-plane $w \cdot x + b = 0$ where x are elements over that hyper-plane. In figure 4.8, w is orthogonal to the hyper-plane, and $\frac{-b}{|w|}$ defines the orthogonal distance from the hyper-plane to the origin.

The perpendicular distances from the separating hyper-plane to the positive and negative side are defined by d^+ and d^-. The hyper-plane margin is defined by $d^+ + d^-$. If the training set is possible to be linearly separated, then the SVM algorithm will attempt to achieve a separating frontier whose margin is maximum:

$$x_i \cdot w + b \geq +1 \quad \text{para} \quad y_i = +1 \tag{4.30}$$

$$x_i \cdot w + b \leq -1 \quad \text{para} \quad y_i = -1 \tag{4.31}$$

Combining equations 4.2.3 it is obtained:

$$y_i(x_i \cdot w + b) \geq 0\,, \quad i = 1, 2, ..., D \tag{4.32}$$

Taking into consideration the elements that return *true* to the expression 4.30 those elements will lay over the hyper-plane defined by $x_i \cdot w + b = 1$, normal to vector $\vec{w}$ at a perpendicular distance of $\frac{|1-b|}{\|w\|}$; by the other hand, those elements that respect condition in 4.31 will be over the hyper-plane $x_i \cdot w + b = -1$, normal to vector $\vec{w}$ at a perpendicular distance of $\frac{|-1-b|}{\|w\|}$.

Therefore, through this geometrical approach, $d^+ = d^-1 = \frac{1}{\|w\|}$ and both distances represents the margin width ($\frac{2}{\|w\|}$). Recalling the distance of a point (x_0, y_0) to a line $Ax+By+C = 0$ is given by the expression $\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$. Therefore, the distance between the hyper-plane $x \cdot w + b = 0$ and $x \cdot w + b = 1$ is $\frac{|x \cdot w+b|}{\|w\|}$ which is $\frac{1}{\|w\|}$. The total distance between $x \cdot w+b = -1$ and $x \cdot w+b = 1$ is, as referred before, $\frac{2}{\|w\|}$. Under this concept the algorithm computes two hyper-planes that maximizes the margin width by minimizing the $\|w\|^2$.

The SVM algorithm ensures that if dataset is linearly separable, there is a single global minimum ($\|w\|$). Ideally the SVM should produce a hyper-plane capable to separate the vectors into two non-overlapping classes. However, on real-world scenarios such complete separation may not achieved and lead to misclassifications. There are cases that separation is only possible through a non-linear function. In these situations the SVM uses *kernels* of non-linear functions in order to map the data into a different space and from there compute the hyper-plane. In short: non-linear functions can be learned by a linear classification algorithm in a high-dimensional space while the system is controlled by a parameter that does not depends on the dimensional characteristic of the space.

The main advantages of *Support Vector Machine*:

- Algorithm particularly suited to classify uncharted data;
- Performs reasonably with unstructured or partially structured data like text, images and trees;
- The *kernel* concept is a plus and with an adequate kernel function, the system can

solve any complex problem;

- Contrarily to neural networks, the SVM is not solved for local optima.
- Performs well on high-dimensional data

The main disadvantages:

- Difficult to choose an appropriate *kernel* function;
- Requires longer training time for large datasets;
- Result on a bewildering model and therefore small calibrations may nearly impossible to do;
- Difficult to fine-tune SVM parameters as *cost* and *gama* because it is difficult to visualize their impact.

4.2.4 Decision Trees

The process the designing and implementation of computational classifying models make use on one of the following concepts:

- Top-down: getting the classifying model from information introduced by the user
- Bottom-up: getting the classifying model from information the relation between dependent and independent variables present in tagged datasets.

As the other examples before, the *decision tree* is another example of a classifier based on the bottom-up paradigm. Therefore the *decision tree* integrates the basic classifiers supporting machine learning, supervised and non-parametric. *Decision trees* are data-structures formed by a set of elements that gathers information, called in the scope of this algorithm as *nodes*. On the bottom (up) of the *decision tree* there is the *root* which is tree's starting point and has the highest hierarchical level. A *branch* is a test outcome and connects *nodes* one to another, or it connects the *root* to a *node*. A *node* that does not have any *branch* leaving is called $leaf$ which correspond the decision to be taken. A $leaf$ *node* is a terminal node that predicts the outcome and it represents itself the class label or class distribution.

In the $Decision\ Tree$ algorithm, the decision takes the path from the *root* till one of the $leaf$ *nodes* or *targets*. The creation of a $Decision\ Tree$ algorithm is directly related

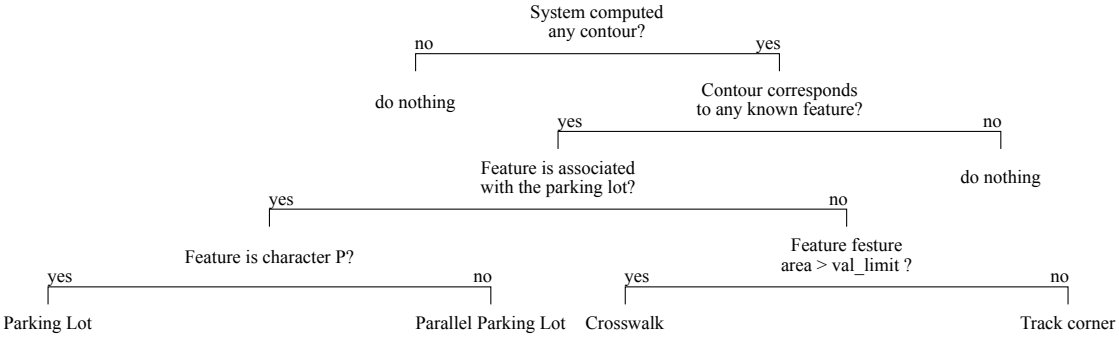


Figure 4.9: *Decision Tree* classifier example

with the type of elements that is intended to classify. These elements may assume a discrete form as the example illustrated in figure 4.9 or continuous data types. This kind of elements dictates to type of *Decision Trees* to consider: *classification tree* for discrete elements or *regression tree* for continuous elements.

A *Discrete Decision Tree* is built through a process of *binary recursive partitioning*. This is an iterative way of partitioning data recursively, splitting it into different *branches*. These *node* can be further split into new classifications levels until a maximum depth defined in order to avoid overfitting. The splitting criteria, starting form the *root* node follows the concept of *Largest Information Gain* (IG). as said before, this process is iterative so it is repeated to each child *node*. The IG ensures that the *nodes* are split at the most informative feature and this function maximizes the information gain at each split.

$$IG(D_p, f) \;=\; I(D_p) - (\frac{N_{left}}{N_p} I(D_{left}) + \frac{N_{right}}{N_p} I(D_{right})) \tag{4.33}$$

Where f represents the feature to perform the split; D_p, D_{left} and D_{right} are the data sets of the parent and child nodes, and I represents the *impurity measure*. N_p, N_{left} and N_{right} represents the number of samples at the parent and child nodes.

The concept of *impurity*, specially important on *Regression Decision Trees*, is on the base on the calculus of information gain, which is the difference between the impurity of parent node and the child nodes. The lower the impurity of child node the higher is the information gain. One the the most common *impurity* measure is the *entropy* which is defined as follows:

$$E(t) \;=\; -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t) \tag{4.34}$$

Where $p(i|t)$ is the frequentist probability of a sample belonging to a class c for a specific node t. The *entropy* is measured between 0 and 1. it may get values higher than 1 but those cases are associated with high level of impurity.

Building a *Regression Decision Tree* suited for continuous data, follows basically the same structure as modeling a $Classification/Discrete\ Decision\ Tree$. However the *impurity* measure shall be redesigned for continuous variables. In this case, the *impurity* measure assumes the form of *weighted mean squared error* (MSE), applicable on the child node only:

$$MSE(t) = \frac{1}{N_t} \sum_{i \in D_t} (y_i - \hat{y}_t)$$

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y_i$$

With N_t still the number of training samples at node t, D_t is the training subset of node t, y_i the true target value, and $\hat{y}_t$ the predicted value.

The main advantages *Decision Trees*:

- Inexpensive to construct;
- Fast classification of unknown records;
- Easy to interpret small-sized trees; Excludes irrelevant features.

Disadvantages of Classification with Decision Trees:

- Easy to overfit;
- Decision tree models are often biased on splitting features with large number of levels;
- Small changes in training data may represent important changes on the decision logic;
- Large trees can be difficult to interpret and may seem counter intuitive.

4.3 Motion and Path Planning

In modern robotics, *Path Planning* is one of the most researched topics. Several approaches have been proposed addressing particular solutions to a universe of difficulties and constraints. Generically, navigation strategies can be classified into two major approaches: *classical* approach and *reactive* or *heuristic* approach.

Classical Approach	Heuristic Approach
Potential Field [37]	Neural Network Technique [38]
Roadmap Cell Decomposition [39]	Fuzzy Logic Technique [40]
Grid Based [41]	Genetic Algorithm Technique [42]
Probabilistic Roadmap [43]	Ant Colony Optimization Technique [44]
Rapidly Exploring Random Tree [45]	Particle Swarm Optimization Technique [46]
Virtual Impedance Method [47]	Bacterial Foraging Optimization [48]
Convex Hull and Local Search Method [49]	Bee Colony Optimization Technique [50]
Divide and Conquer Method [51]	

Initial work developed in the field of mobile robotics has grounded on classical approaches. These classical methods present some shortcomings, namely: the frequent need of high computational effort, the difficulty of dealing with high degrees of uncertainty, the need for precise information on the environment often associated with the request for sensors that provide equally precise and detailed information at a suitable rate for real-time systems, and without information about the global environment, the convergence to the finish/goal point is not guaranteed by the path planner, therefore, it may be stuck at some *local minima* needing to recalculate waypoints during the movement of the mobile robot [52], and much more.

On those systems whose Path Planning techniques follow a classical approach, it remains a latent doubt if whether a solution will be found, or facing the incapacity to find a solution, the system will assume the solution does not exist [53] .

The unpredictable nature of classical approaches makes their use fragile in a context of real-time processing constraints. Despite many researchers continue to fill gaps in the classical approaches, developing more refined strategies as an "upgrade" of classical strategies, such as *Adaptative Potential Fields* and other hybrid algorithms, these strategies hardly achieve better performance, in real-time applications, when compared with reactive approaches.

Given the central characteristics of the model itself, the classical approaches are more suited for navigation in known and static environments. In opposition, reactive approaches are used in navigation contexts where the environment is unknown. Reactive

approaches reveal particularly suited to find a solution even if the limited knowledge of the environments is highly affected by uncertainty.

Typically, reactive approaches offer relatively easy implementation, have more sophisticated and efficient algorithms, are often used in real-time navigation situations, and offer better results when compared to classic approaches. Although reactive approaches have several advantages when compared with classical approaches, they still have important disadvantages such as longer computational time, complex design, the need for an initial learning process and, in some situations, substantially larger memory requirements when compared with the typical requirements to support classical approaches [54].

Regardless of the approaches listed, the overall performance of Path Planning techniques is closely dependent on available information, in particular the map.

It is not in the scope of this section to describe each path-planning technique disregarding the context of its application. Herein will be tackled the strategies specially oriented to perform path-planning in a structured, defined, known environment with the possibility to encounter with dynamic obstacles.

To allow a better understanding where $Path - Planning$ and $Motion - Planning$ system fits in a whole autonomous driving system, it is necessary to bring the line of focus far to understand its context.

Autonomous driving vehicles are, basically, autonomous decision-making systems, wheres the decision is supported by a set of information provided by sensors such as cameras, LIDAR, radar, GPS / INS, encoders as well as the knowledge of the road map, the rules governing it, the dynamics of other vehicles, etc.

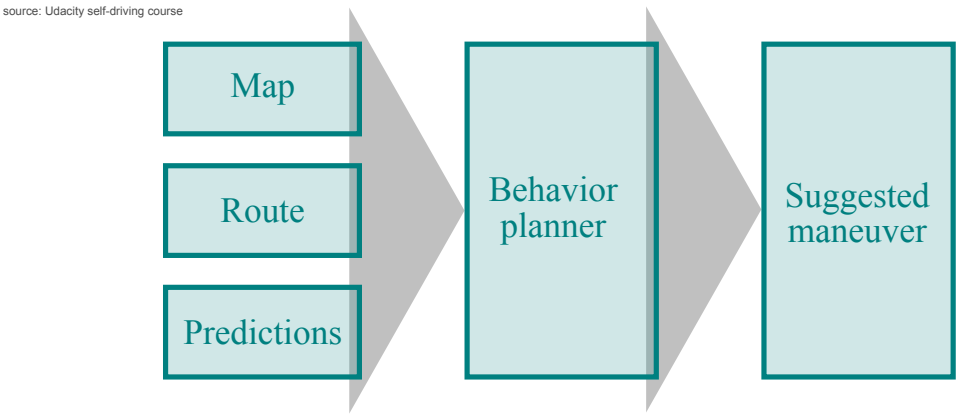


Figure 4.10: Behavioral planning overview

The behavioral planner is responsible for:

Suggest "states" or maneuvers which are:Feasible Efficient Safe Legal

However the behavioral planner is not responsible for:

- Execution details
- Collision avoidance

The approach often taken to address this decision problem is to divide and organize the perception and decision making tasks into a hierarchical structure. Information gathered from the observation systems is used by the perception system to provide an estimate of the vehicle's condition and the context in which it is inserted. Subsequently, based on the information produced about the vehicle state and its surroundings, the decision-making system monitors the vehicle to meet its defined goals.

Autonomous vehicle decision systems are generally distributed in four components:

1. Path Planning
2. Behavioral Layer
3. Motion Planning
4. Control System

At the highest level is *Path Planning*. It is considered as the process of finding the best route to follow globally from the starting point to the goal point, integrating, in more refined models, information on the current state of road traffic.

At a lower level is the *Behavioral Layer* by some authors it also referenced as *Maneuver-Choice* which defines the context-appropriate driving task at a given point in time. It is a high-level method for describing vehicle movement, taking into account its current position and speed on the road. *Behavioral Layer* focuses mainly on high-level decision making such as moving forwarding, left turn, right turn, overtaking, detour, etc., taking into account the planned route.

Then, at the next level is the *Motion Planning* component that chooses the appropriate set of configurations to execute the outlined path and act accordingly to the Behavioral Layer directives. Addressing to critical real-time issues, plans the transition from the current state to a new state, in a feasible way, respecting the holonomic and dynamic constraints of the vehicle, the specific rules in that particular section of road,

and road boundaries. In this way, the trajectory is represented as a set of vehicle states until reaching the goal point, parameterized by time, viability and speed.

Finally, at the lowest level is the *Control System* that monitors the system feedback loop, correcting the errors of execution of the planned movement.

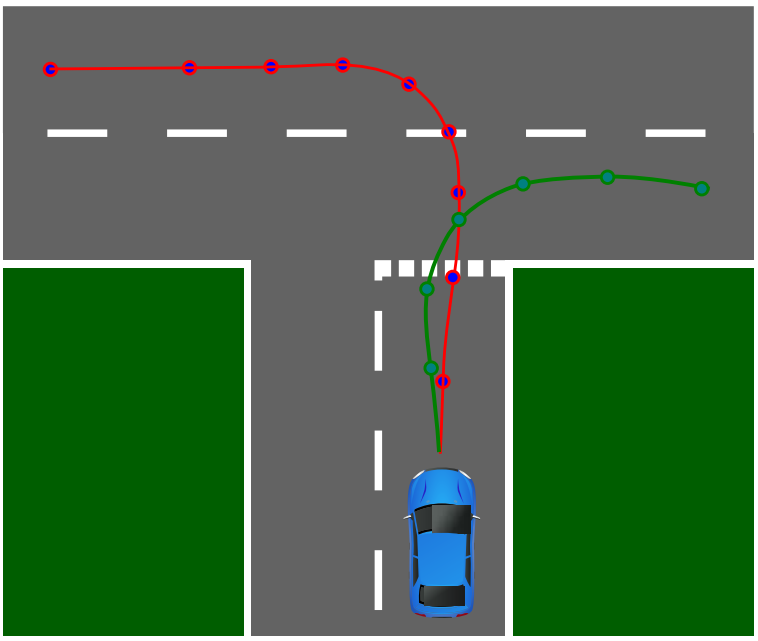


Figure 4.11: Model Predictive Controller [4]

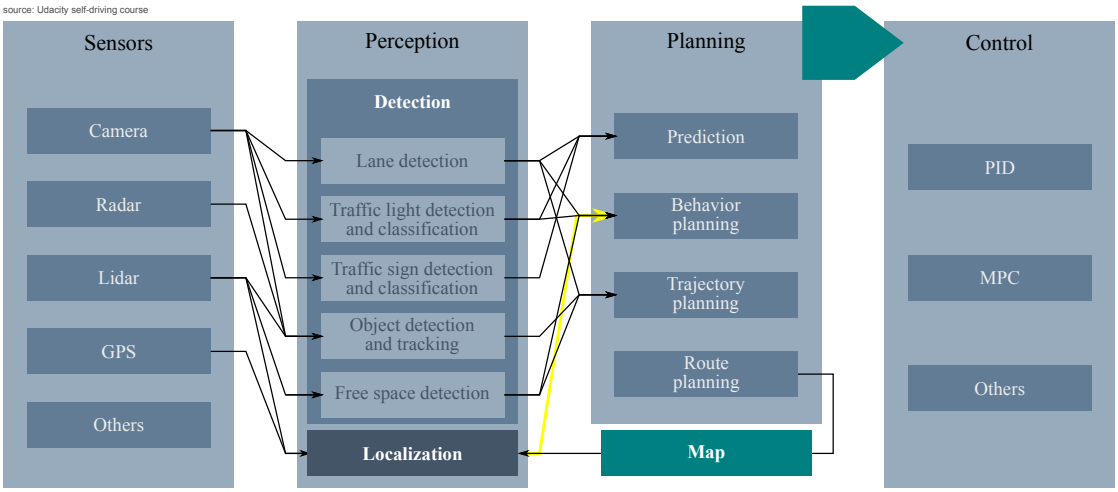


Figure 4.12: Autonomous Driving System [5]

4.3.1 Path Planning

As noted earlier, the *Path Planning* occupies at the high-level the autonomous decision system. This system has the responsibility of selecting the path through the road network from the robot's current location to the destination point (*goal*).

Typically both *Localization* and *Path Planning* uses a global Cartesian coordinate system to define location on the map (x_{robot}, y_{robot}).

Directed graphs, with route costs associated to their end, represent road maps. These costs associated with the ending extremity of each graph may, as an example, define the cost of crossing the subsequent road segment.

The route can be defined or computed based on different criteria such as minimum cost (distance, energy, holonomic constraints, etc.) on a road network chart. As referred in the preamble of this section, there are many approaches and techniques to define the optimal path. However, given the complexity of today's road network, applying classic *Path Planning* algorithms, such as Dijkstra or A-Star, becomes impracticable [55]. Route planning in road networks has attracted the interest of the scientific community and currently available algorithms are capable of calculating an efficient path at a continental scale trajectory in a fraction of seconds. [56].

The *Path Planning* of autonomous vehicles becomes possible at the moment when the structured urban environment is transformed into a digital configuration of space or a *state space*. *Path Planning* seeks to find between the available or accessible space a set of trajectories that allow the vehicle to drive autonomously avoiding collision with obstacles.

The concept of *Path Planning* shall be defined: given a current robot localization and goal/endpoint, find a path that allows the robot the reach that goal. During the *Path Following* the robot shall avoid any encountered obstacle (*Obstacle Avoidance*), initially unknown and further mapped, as long as the robot progresses on the defined path. Typically, the *Obstacle Avoidance* requires path redefinition by keeping the goal as the desired destination and yet over passing the obstacle, respecting the holonomic constraints at a minimum cost possible.

As the starting point, it is necessary to define the *configuration space* (C) [57] which translates as all possible system settings. The *configuration space* (C) is composed in two parts: the *free space* (C_{free}) part characterized by the absence of obstacles, which the robot may or not occupy, and where is frequently top-layered the defined path; and the *occupied space* (C_{obst}) normally berried space by the existence of an obstacle. When defining paths it is frequent in many techniques to reduce the configuration space by

reducing the size of the robot to a single point. To this end, the existing obstacles shall be expanded in a way that the path defined to to robot avoids leading the robot into a collision condition cause by this manipulation of the configuration space [58].

Once defined the configuration space, path planning techniques are used in order to obtain the optimal path that allows the robot to move to the desired point.

4.3.1.1 Roadmap

The *Roadmap* algorithm builds its concept on the definition of *nodes* representing localization, and *links* connecting these nodes representing possible paths between them. *Roadmaps* are links from the *initial* to the *goal* point, taking place on the *free space* - ($C_{free} = C - C_{obst}$) - through a network of one-dimensional curves. If exists a path connecting the initial point to the goal point, then this path will the combination of three *subpaths*: a first one connecting the initial point to the *roadmap* network; a second *subpath* defined by the *roadmap*; and the third linking the *roadmap* to the goal point.

This representation is specially convenient because it resumes the trajectory planning problem into a set o links between a starting and ending configuration. Generically the output of this technique is the production of graphs which will constitute support on the identification of the best route.

4.3.1.2 Voronoi Diagram

The *Voronoi Diagram* concept is a simple and intuitive technique. It consists of a set of equidistant points between two obstacles. Considering a finite number of objects in the space, all locations in that space are associated with the closet member of the object set. This results in partitioning the space into regions

So the *Voronoi Diagram* algorithm, for each discrete set of points bounder polygons encloses all the intermediate close points, one to another, herein the set. For the set of polygons originated from a point set is defined as the Voronoi diagram.

This technique permits to define a path linking those set of points. The algorithm has implicit the concept of maximizing the distance of the robot to the obstacles, hence to each bisecting point is calculated the distance to the nearest obstacle. The robot will follow these points equidistant to the surrounding obstacles until it reaches the goal. A robot equipped with range finder can incrementally build a Voronoi diagram of the unknown distance.

This technique has an important fragility: distance sensors (range finders) have a limited range of perception. Sensors with a short-range can misguide the robot.

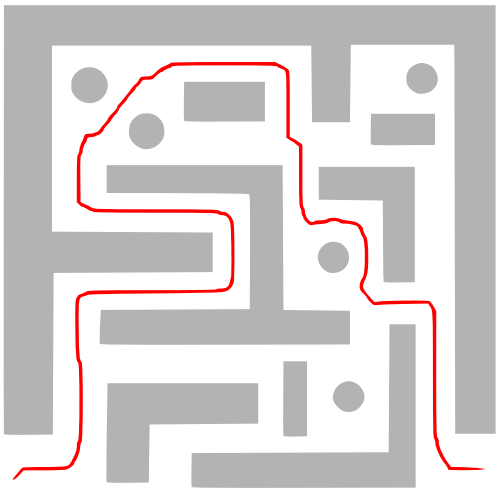


Figure 4.13: Voronoi Diagram

4.3.1.3 Probabilistic RoadMap

The planning of the path takes place in two separate stages: a *preprocessing* stage defining a roadmap in the *free-space* (C_{free}); and a second stage, the *query*, which defines a path among the mapped nodes connecting the starting point to the goal point.

The preprocessing stage, also considered a learning phase, consists of randomly distribute points over C_free, to be posteriorly linked in the next stage.

The preprocessing stage also divided into two phases: a roadmap construction phase and a roadmap enhancement phase. In the roadmap construction, random configurations are generated over the C_{free} and connected by a simple deterministic planner. The connection of the neighbor node is done by distance criteria inferior to a defined limit distance. These connections only link straight neighbor points and only if this link does not intercept the *occupied-space* by obstacles.

So the planner only tries to interconnect nodes that satisfy this maximum distance criterion. Each connection made yields itself an edge of the roadmap. Once a significant number of nodes are generated, the algorithm then performs the roadmap enhancement, which is a heuristic evaluation of space close to obstacles, generating more nodes. The second phase has the purpose to tune roadmap elements suited to the C_{free}.

The query stage aims to link the starting to the goal point trough the roadmap laying on the C_{free} built by the preprocessing stage. Then after, through a graph search algorithm, it is established a path that permits to go from the starting point to the goal point.

The query stage aims to link the starting and the goal point to the roadmap built on C_{free} by the preprocessing stage. Then, through a graph search algorithm, a path is computed from the initial node to the destination node.

In detail, the query stage specifies the beginning and goal configurations of the robot. It first connects the starting point to the closest node in the roadmap.

An important fragility of probabilistic roadmap technique is the difficulty to define a path (find a solution) through narrow C_{free} (between obstacles close to each other)

To improve the performance of this *Probabilistic RoadMap* and decrease the execution time when searching for the path, some improvements were developed such as the *Rapidly-exploration Random Tree* (RRT) which differs from the PRM on the saved node. As mentioned earlier, the PRM saves the randomly generated node, whereas in RRT the saved node is a node that stays in the direction of the randomly generated node and the nearest existing node, respecting a previously defined distance. This refinement allows a faster search as the nodes become more concentrated and therefore less or absence of redundant paths.

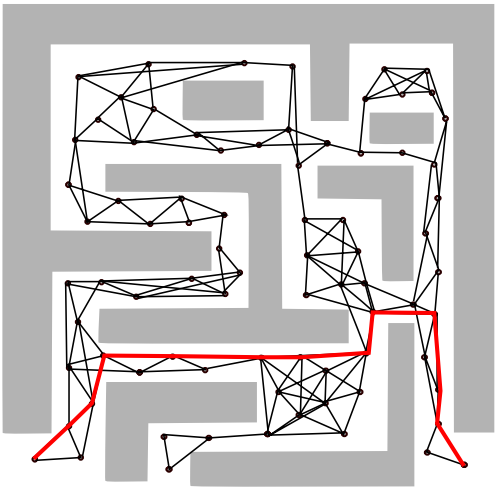


Figure 4.14: Probabilistic Roadmap

4.3.1.4 Visibility Graph

The *Visibility Graph* is a two-dimensional graph consisting of *nodes* representing the start, goal points, *vertices* defined by the obstacle configuration space, and *edges* joining nodes together describing a path.

Edges are line segments connecting two nodes belonging to each other's visible field.

All nodes and edges are defined in free space (C_{free}), so the given path will not intersect obstacles. This way, the path is formed by graphs containing edges representing the shortest distances between nodes. However, it is important to notice that edges formed by obstacles (C_{obst}) in the configuration space, and supporting the robot's traveling path between start and goal point, make the robot moving too close to the obstacles, increasing the probability of collisions.

The further techniques improved the *Visibility Graph* method by removing edges that will never be used on the path planning.

As an example is the technique called *Reduced Visibility Graph* (RVG). In RVG, new supporting line-segments are generated;

1. Supporting line-segments, tangent to two obstacles, are computed in a way that both obstacles are on the same side of this line;

2. Separating line-segments, also tangent to both obstacles but separating them one to each side of the line.

Therefore, the RVG path is formed by the supporting lines and separating lines, and those edges formed in the standard *Visibility Graph*, which are not supporting or separating lines, are removed. This technique reduces the time spent to compute the path.

4.3.2 Motion Planning

Once selected the best path and the best maneuver, the problem is now addressed to find the best motion/trajectory respecting robot's kinematic and dynamic model, ensuring simultaneously an optimal performance in comfort and safety.

The motion planning, is the task of continuously searching for a collision free path, where that path starts at an initial configuration point, and ends at a goal configuration point. The basic motion planning problem also known as the *piano movers problem* [59].

The motion-planning problem is addressed locally and does not have a global reach. Relies mainly on sensor information to produce secure orders[60]. There are several techniques to locally plan the robot's motion.

4.3.2.1 Potential Fields

This method defines an objective function that gives to obstacles the configuration of fields with great potential and the trajectory points fields with low potential[61]. The

main element on a *Potential Field* algorithm is the action vector which corresponds, in a simplified concept, to velocity and orientation of the robot at each configuration on the space. Each behavior produces an output vector. As a basic example is the *GoalSeek*, which gives the robot the task of moving towards the goal. If the robot passes through every possible cell on the configuration space, the snapshot of all those possible output vectors would resemble the figure 4.15.

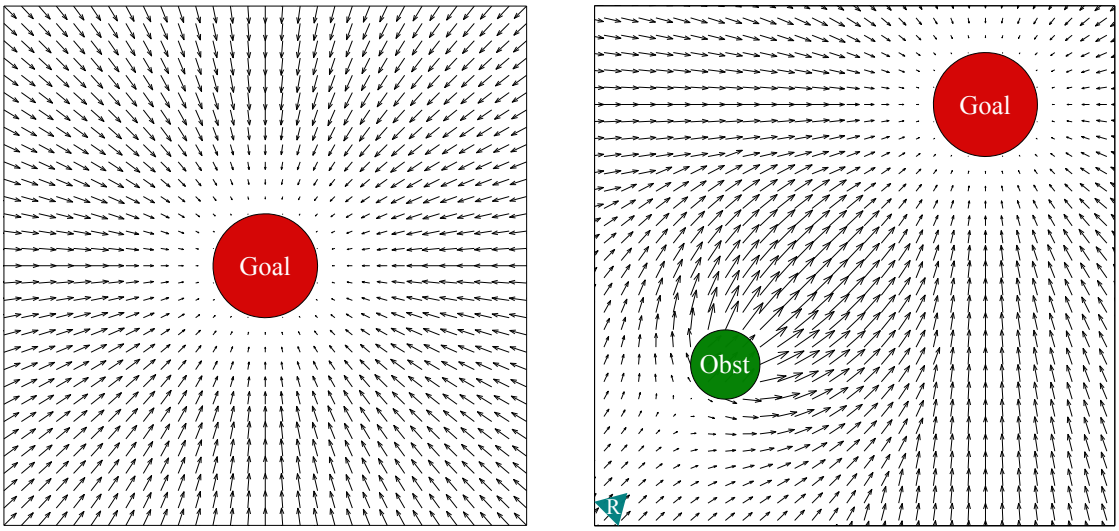


Figure 4.15: Potential Fields - (left) *Goal* attractive field force on the configuration space; (right) *Obstacle* repulsive forces and *Goal* attractive forces on a 2D configuration space

To the set of vector seen in figure 4.15[left] it is called vector field because it resemble potential energetic field that robot tries to follow.

The potential feel associated to the Goal is an example of*Attractive Potential* field because it makes to robot attracted to it (all the vectors computed on the free space aims towards the goal. Taking into consideration a 2D configuration space, the algorithm process relies on the mapping of one vector v into a gradient vector $\nabla f(x, y)$ where f is a function of v [61]:

$$v = [x, y]^T \qquad \text{and} \qquad \nabla f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T \tag{4.35}$$

- Consider (x_G, y_G) as the position of the *goal* point, with radius r_G;
- Consider $v = [x, y]^T$ the position of the robot;

- Compute the distance between *goal* and robot:

$$d_G = \sqrt{(x_G - x)^2 + (y_g - y)^2}$$

- Compute the angle between agent and *goal*:

$$\theta_G = \arctan\left(\frac{y_G - y}{x_G - x}\right)$$

- Set the values for $\nabla f(x, y)$ as follows:

$$\begin{array}{rll} \text{if} \quad d_G < r_G & & \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0 \\ \text{if} \quad r_G \leq d_G < s + r_G & \frac{\partial f}{\partial x} = \alpha(d_G - r) \cdot \cos(\theta_G) \quad \text{and} & \frac{\partial f}{\partial y} = \alpha(d_G - r_G) \cdot \sin(\theta_G) \\ \text{if} \quad d_G \geq s + r_G & \frac{\partial f}{\partial x} = \alpha s \cdot \cos(\theta_G) \quad \text{and} & \frac{\partial f}{\partial y} = \alpha s \cdot \sin(\theta_G) \end{array}$$

The process above – *GoalSeek* – describes a *goal* point with radius r_G and when the robot reaches the *goal* point no forces will act upon it because when $d_G < r_G$ the gradient vector $\nabla f(x, y)$ will be set to zero. The potential field area spreads by s and robot reaches the limit of the extent of s when $d_G = s + r_G$. When the robot is located inside the extent area, the force exerted is proportional to the distance between the robot and the goal, scaled by a factor $\alpha > 0$.

The combination of *GoalSeek* with a *AvoidObstacle* is processed as follows:

- Consider (x_O, y_O) as the position of the *obstacle*, with radius r_O;
- Consider $v = [x, y]^T$ the position of the robot;
- Compute the distance between *obstacle* and the robot:

$$d_O = \sqrt{(x_O - x)^2 + (y_O - y)^2}$$

- Compute the angle between agent and *Obstacle*:

$$\theta_O = \arctan\left(\frac{y_O - y}{x_O - x}\right)$$

- Set the values for $\nabla g(x, y)$ as follows:

$$\begin{array}{rll} \text{if} \quad d_O < r_O & & \frac{\partial g}{\partial x} = -\text{sign}(\cos(\theta_O))\infty \quad \text{and} \\ & & \frac{\partial g}{\partial x} = -\text{sign}(\sin(\theta_O))\infty \\ \text{if} \quad r_O \leq d_O < s_O + r_O & \frac{\partial g}{\partial x} = -\beta(s_O + r_O - d_O) \cdot \cos(\theta_O) \quad \text{and} & \\ & \frac{\partial g}{\partial y} = -\beta(s_O + r_O - d_O) \cdot \sin(\theta_O) & \\ \text{if} \quad d_O \geq s_O + r_O & & \frac{\partial g}{\partial x} = \frac{\partial g}{\partial y} = 0 \end{array}$$

Within the *obstacle* area, the repulsive forces are infinite, pointing out from the center of the obstacle. Outside the circle with radius $s_O + r_O$ the force is null. Within obstacle's influence area, the repulsive force grows as long as the robot gets close to the obstacle boundary ($d_O = r_O$).

$$\beta(s_O + r_O - d_O) = 0 \quad \text{when } d_O = s_O + r_O$$
$$\beta(s_O + r_O - d_O) = \beta(s_O) \quad \text{when } d_O = r_O$$

As the *attraction* field, in the *repulsive* field, β refers as a scale factor to adjust the effects of this force. The forces inside the influence extent have a negative sign ($-sign$). The combination of multiple fields the done but its sum.

$$U(x, y) = \nabla f(x, y) + \nabla g(x, y) \tag{4.36}$$

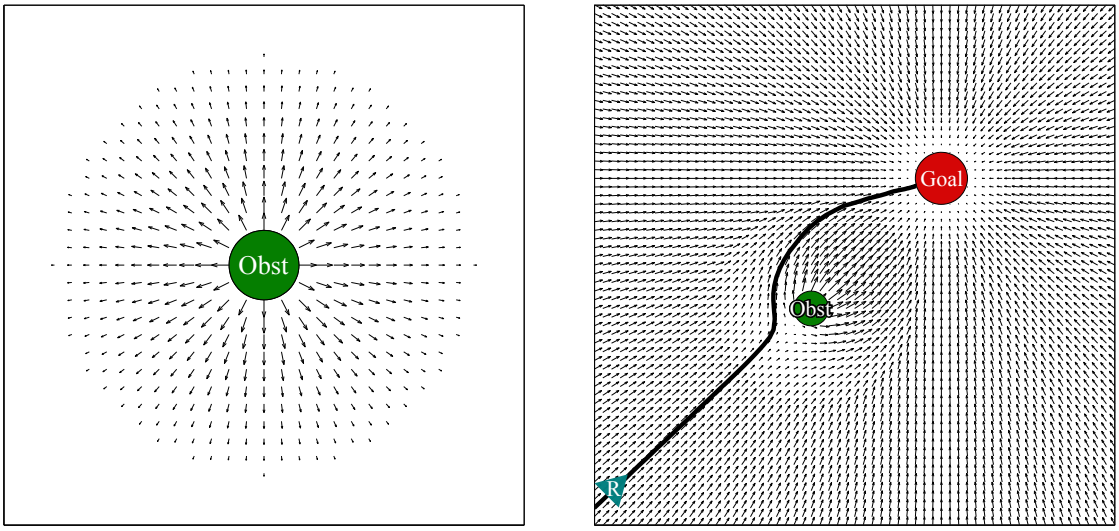


Figure 4.16: Potential Fields - (left) *Obstacle* repulsive field force on the configuration space; (right) *Obstacle* repulsive forces and *Goal* attractive forces on a 2D configuration space

In this case, the problem is limited to an optimization issue which tries to define commands that drives the robot through a local potential minimum. This technique is robust on collision avoidance maneuvers due to the use of proximity sensors once it does not require detailed knowledge about the obstacles. However, this technique requires a considerable computational effort on calculations that may compromise real-time navigation applications.

To overcome some constraints, namely the relatively heavy computation effort, some improvements are presented such as virtual force fields and vector field histograms, vector field histograms [plus] and vector field histograms [asterisk].

4.3.2.2 Model Predictive Control

This technique merges control theory elements into motion-planning modules. The *Model Predictive Control* (MPC) uses the robot's dynamic module and through it, samples control inputs regarding the vehicle's motion on the next control step. It is a feedback control method to get an adequate control input. Supported by the dynamic model and control inputs, the optimization problem to find the best trajectory is solved.

The performance of the MPC is not directly related to the number of obstacles present on the scene. However, the trajectory optimization gets more complex and difficult to perform as long as more variables are introduced to model the vehicle [62].

The MPC predicts the next N-step waypoints according to the vehicle model (dynamic and kinematic), and calculates the cost functions to reach each of these waypoints.

At each control time-step, the MPC controller makes estimations about future positions of the vehicle. To allow the robot to drive along the defined trajectory, as close as possible to the referential trajectory, the MPC redefines the controlled as an optimization problem. The algorithm attempts to minimize the cross-track errors between positions as well as the angles of steering wheel.

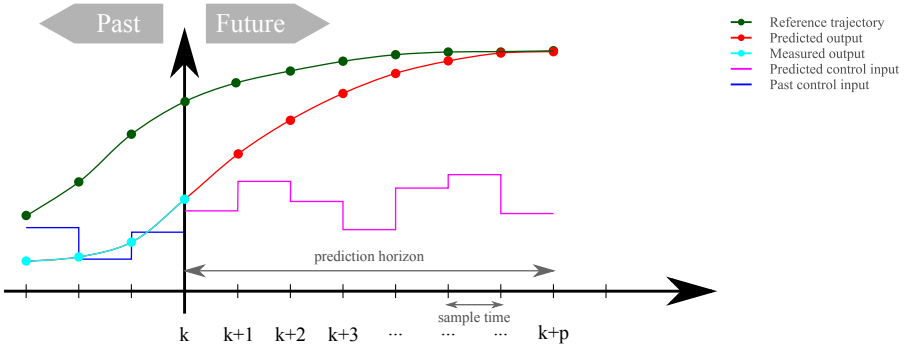


Figure 4.17: Model Predictive Control - prediction sliding window [4]

On the initial time-step (t_n), the MPC controller considers only the first configuration and disregards the further configurations ($t_{n+1}, \ldots, t_p$). On the next time-step (t_{n+1}), a new reading from the sensors is obtained and a new position is computed. The prediction horizon moves a step further and a new calculation cycle initiates to define next steering

and acceleration configurations, based on the system model:

$$x_{t_{n+1}} = x_{t_n} + v_{t_n} \cdot \cos\theta \triangle t \tag{4.37}$$

$$y_{t_{n+1}} = y_{t_n} + v_{t_n} \cdot \sin\theta \triangle t \tag{4.38}$$

$$\theta_{t_{n+1}} = \theta_{t_n} + \frac{v_{t_n}}{CC_{offset}} \cdot \delta_{t_n} \triangle t \tag{4.39}$$

Where x_{t_n} represents the X coordinate a y_{t_n} the Y coordinate of the vehicle's location at time instant t_n. $x_{t_{n+1}}$ and $y_{t_{n+1}}$ predicts the vehicle's location at the next time instant t_{n+1} based on the velocity v_{t_n} at the current time instant and the changing of heading ratio $\theta\triangle t$. The CC_{offset} is the length between the front axle of the vehicle and its center of gravity and δ_{t_n} is the steering angle. Considering the physical constraints of the vehicle:

$$\begin{aligned} \delta &\in \Big[-steer^{\circ}_{lim}, +steer^{\circ}_{lim} \Big] \\ Ua &\in \Big[-accel_{max}, +accel_{max} \Big] \\ v_{t_{n+1}} &= v_{t_n} + Ua_{t_n} \triangle t \end{aligned} \tag{4.40}$$

Where δ represents the steering limits in angles to the left and right turns, Ua represents the acceleration (from throttling to accelerating) accordingly with vehicle's limitations. The vertical axis in figure 4.17 reflects the current time (t_n). The values on the left reflects the past, and values to the right predicts the future. At t_n, the MPC controller uses the car model to simulate the car's path for the next p time-steps 4.37. The parameter p defines how far ahead the control looks into the future (prediction horizon). The MPC controller puts on the equation multiple future scenarios in a systematic way. It is at this point the optimization problem arises. When solving the optimization problem, the MPC controller tries to minimize the error between reference and predicts path.

$$\begin{aligned} \delta\epsilon_{t_{n+1}} &= \Big(\theta_{t_n} - \theta\mathrm{des}_{t_n}\Big) + \frac{v_{t_n}}{CC_{offset}} \cdot \delta_{t_n} \triangle t \\ CT\epsilon_{t_{n+1}} &= \Big(f(x_{t_n}) - y_{t_n}\Big) + v_{t_n} \cdot \sin\theta\epsilon_{t_{n+1}} \triangle t \end{aligned} \tag{4.41}$$

The cost function J of this optimization problem, includes both errors – cross-track error and steering error – and it is a weighted squared sum of the errors against the reference value.

$$\mathcal{J} = \sum_{t=1}^{n} \left(\delta\epsilon_t - \delta\text{ref}_t\right)^2 + \left(CT\epsilon_t - CT\text{ref}_t\right)^2 \tag{4.42}$$

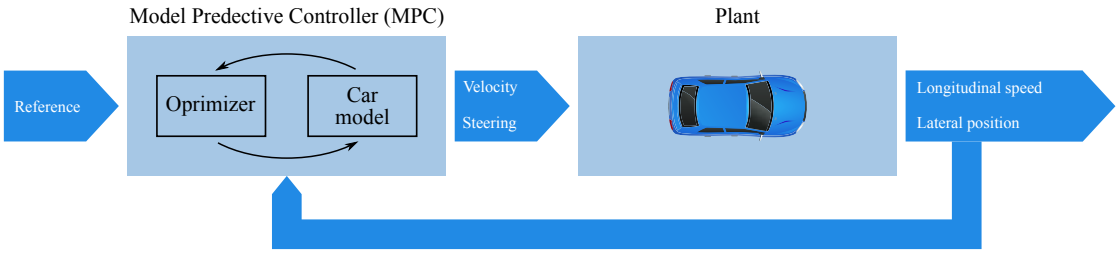


Figure 4.18: Model Predictive Control - controller

4.3.2.3 Geometric Curves

The use of *Geometric Curves* is, probably, the most frequently used technique in the implementation of autonomous driving solutions. Based on a local planning model, the techniques have its starting point on the following premise: given a local space, exists a geometric curve and a set of other adjutants defining possible trajectories. Each curve itself is a candidate route, as such assessed through a cost function with various considerations such as distance and time costs, acceleration and collision checking. Geometric representations of trajectories include polynomials, Bezier curves, spline curves, and clothoid.

There are several different algorithms of trajectories generators, optimizing a set of possible candidates to trajectories. Frequently these trajectories are generated by second-order polynomials, computed based on the error between final state and the desired final state. Vehicle's configurations (velocity and accelerations) are computed for each point. However, the first enunciations of this technique did not take into consideration the obstacles and the complexity increases when required convergence points are yet far from the initial state or unreachable due to dynamic constraints of the vehicle [63].

Other approaches on geometric curves consider higher-order polynomials to endow the system, with more information about the trajectory. In certain developments, there is the distinction of overtaking trajectories or car-following, defining to each scenario own cost functions taking into account different parameters such as safe distance, etc. [64].

The *Cubic Bezier* curves constitute another approach associated with Geometric Curves technique to define a local trajectory. In this particular technique, the cost function used considers weight parameters such as trajectory length, smoothness, and cross-track error against the reference trajectory. In this solution, the obstacles, considered as circles, are taken into account when selecting the best candidate. Making use of circle abstraction to define obstacles, it permits the algorithm to discard candidates after performing a collision inspection over the proposed trajectories. However, despite this algorithm also draw solutions for the scenario with obstacles, these are considered to be static, which is not common in a real application [65].

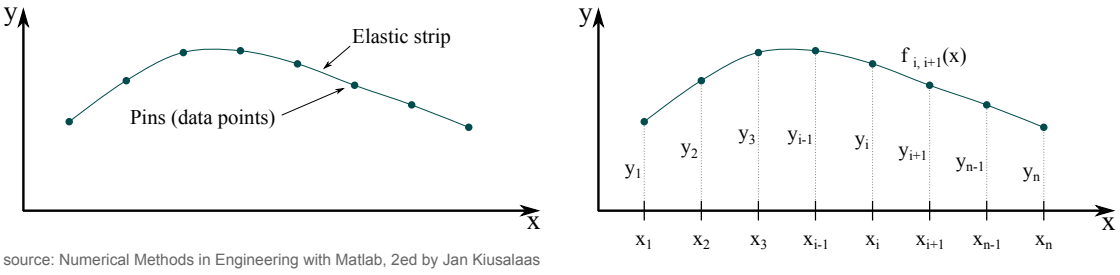
$$\mathrm{B}(t) = \sum_{i=0}^{n} {}^{n}C_n i(1-t)^{n-1} \cdot t^i \cdot P_i \tag{4.43}$$

Where n describes if the *Bezier curve* is linear, quadratic, cubic or any other higher orders. *Bezier* is a parametric curve form and t is the parameter that stays between 0 and 1. and P_i represents the anchor waypoints and the *handles* or control points.

Fourth-order *Cubic Bezier* curves are also used the generate trajectories, bring to the system more information and better results. However dynamic constraints are considered such as velocity and acceleration, the majority of the techniques proposed consider low velocities and static obstacles [66].

Another technique to produce *Geometric Curves* is the use of *Spline*. The Splines are used to generate trajectories attending to waypoints defined or provided by the high-level path-planning and taking also into account the transit signs. Each trajectory is then evaluated by its distance, time to reach the next waypoint (or control point), and collision probability [67]. Further developments driven into this concept, proposes the generation of spline trajectories between the waypoints defined by the path-planning module, but these trajectories are constrained to curvature, speed, and acceleration.

An *spline* is a polynomial of degree k that is continuously differentiable $k-1$ times. Take an example of a *cubic spline* that is two times differentiable. A *cubic spline* is considerably more stable than 2nd-order polynomial in the way it oscillates less between points.



source: Numerical Methods in Engineering with Matlab, 2ed by Jan Kiusalaas

Figure 4.19: Concept of implementing cubic *spline*

Given a n waypoints defined as (x_i, y_i) where $i = 1, ..., n$, assume that $x_i < ... < x_n$, there is an unique solution $f(x)$ to the problem that satisfies the following conditions:

1. $f(x)$ is a polynomial of 3rd-order on each interval of (x_i, X_{i+1});
2. $f(x)$, $f'(x)$, $f''(x)$ are continuous:
   - $f(x)$ — $f_{i-1,i}(x_i) = f_{i,i+1}(x_i)$;
   - $f'(x)$ — $f'_{i-1,i}(x_i) = f'_{i,i+1}(x_i)$;
   - $f''(x)$ — $f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i) = k_i$;

Where k_i is the second derivative of the segment at waypoint i. This last condition is important to make the curve described by the function to connect each waypoint of the segment.

Other alternatives of splines lie on learning methods of the human driving behavior, but these methods require an extensive volume of data capable to realistically replicate the human driving behavior [68].

The *Circular Line* fit is also a technique used in the scope of *Geometric Line*. In this technique, an optimum trajectory is defined as the example of manoeuvrer to overtake the vehicle ahead. The line fit is chosen in a trade-off between efficiency and comfort. The combination of two circular arcs generates the trajectory [69]. Regarding the probability of collision, both arcs are evaluated with a significant degree of confidence the candidate trajectories by estimating possible obstacles' trajectories.

The candidate trajectories may also be called as *tentacles* [70] because the method is inspired by the behavior of an insect that uses its antennas to perceive the environment and avoid obstacles. An important weakness of this technique is to consider all generate tentacles generated to a certain velocity as possible trajectory candidates, even if the curve represented by those tentacles does not respect the physical constraints as the steering capabilities of the vehicle.

$$r_k = \begin{cases} \rho^k R_j & :k = 0, ..., \frac{m}{2} - 1 \\ \infty & :k = \frac{m}{2} \\ -\rho^{k-\frac{m}{2}+1} R_j & :k = \frac{m}{2} + 1, ..., l \end{cases} \tag{4.44}$$

Where r represents the radius the $k-th$ *tentacle*. The exponential factor ρ defines the number of tentacles with small curvatures and a coarser distribution at larger curvatures. R_j represents the initial radius of speed set $j = 0, ..., n-1$, taking into consideration the length l of the outmost tentacle and the angle $\triangle\phi$ of the outmost tentacle at the lowest speed [70]:

$$R_j = \frac{l}{\triangle\phi(1 - \frac{j}{n-1})} \tag{4.45}$$

This page was intentionally left blank.

Chapter 5

Strategy and Implementation

In this chapter, a description of the adopted approach, with an explanation of the concept in response to the high-level architecture presented in Chapter 3, as well as, the corresponding algorithm implementation. The enunciation particularizes the system to a lower-level, describing step-by-step the process implemented on each block. Each implementation description ends with algorithm snippets.

5.1 Inverse Perspective Mapping

Autonomous driving systems rely heavily on data provided by optical sensors/cameras, whether these are configured as monocular cameras, stere-cameras/multi-camera, or omnidirectional cameras, and on the information that feature extraction algorithms can obtain from captured images. As it was referred in chapter 2 dedicated to the study of art, common solutions of lane-detection, crosswalk detection, and signals identification typically incorporating autonomous driving systems, are invariably based on the information produced by algorithms dedicated to computational vision.

One of the relevant information in computational vision is related to the measurement of distances to objects or land-marks relevant to the navigation process. In this context, a monocular camera system for image processing is often applied in vehicles to identify and determine the distance of an object. The *Inverse Perspective Mapping* (IMP) method is an implementable solution to perform this function.

Although other techniques can be used to obtain a birds-eye-view image, used in computer vision systems to correct the distortion caused by the camera perspective, such as the homography decomposition, provides an image whose image pixel match to a point in the world is not directly obtained.

To operate the system, it is necessary, first, to calibrate the camera to obtain the intrinsic and extrinsic parameters. While evaluating the proposed solution in a simulation environment, the calibration was performed using different setups of the rectangular-shaped calibration plane (checkered standard following Matlab recommendations in `https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html`) as the starting point.

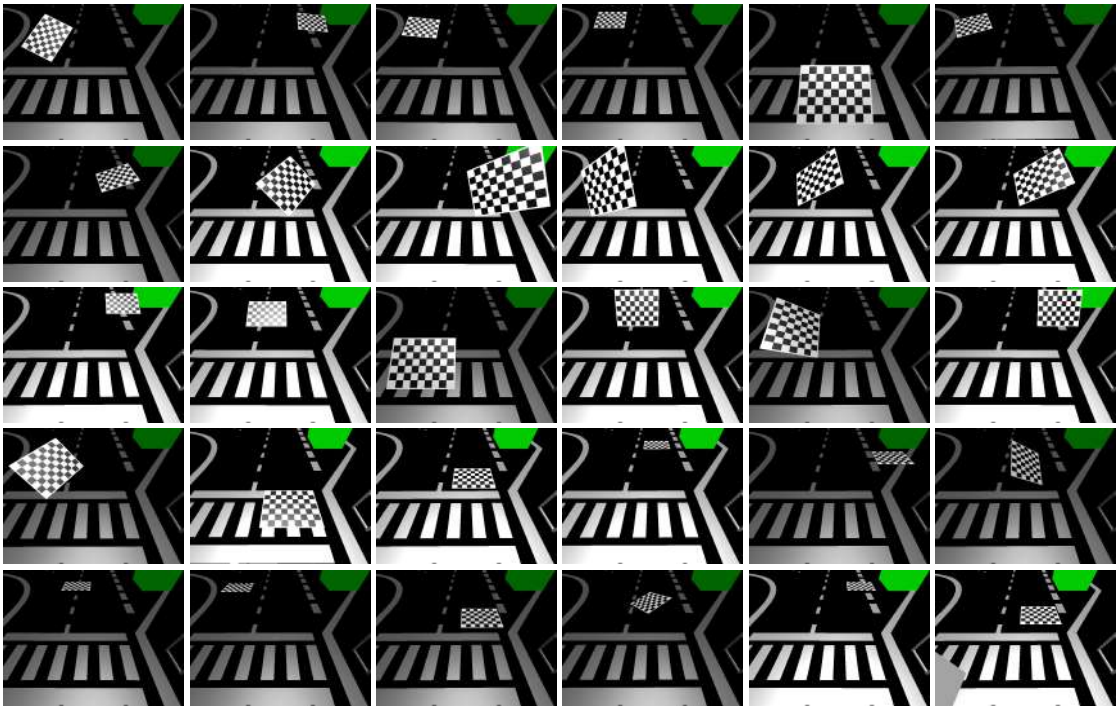


Figure 5.1: Calibration images captured from FNR-ADC Simulation Environment

The calibration process through the images acquired from the simulator is then used to obtain the calibration parameters through Matlab's Calibration Toolbox. Although both Matlab and OpenCV use essentially the same calibration algorithm, Matlab uses Levenberg-Marquardt's non-linear least-squares method [71] for the optimization process, while OpenCV uses gradient descent methods. The main advantage of using the available Matlab calibration method – `estimateCameraParameters()` – is that it minimizes the error when compared to OpenCV – `calibrateCamera()`.

Once obtained the camera calibration parameters (intrinsic, extrinsic) these parameters are compiled in a YAML file allow an easy update of the algorithm in case some parameter may change on the robot (i.e. changing the *tilt* angle of the camera).

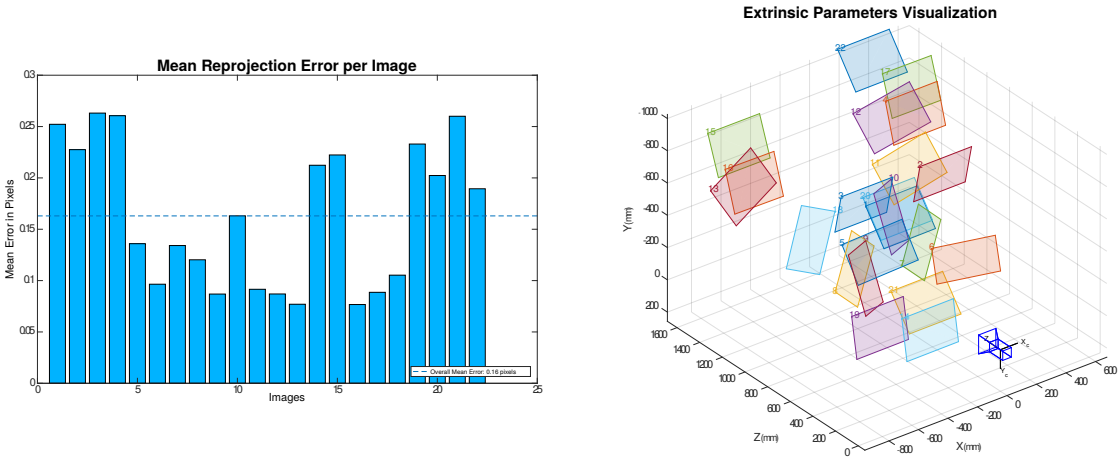


Figure 5.2: Calibration accepted plans and calibration errors

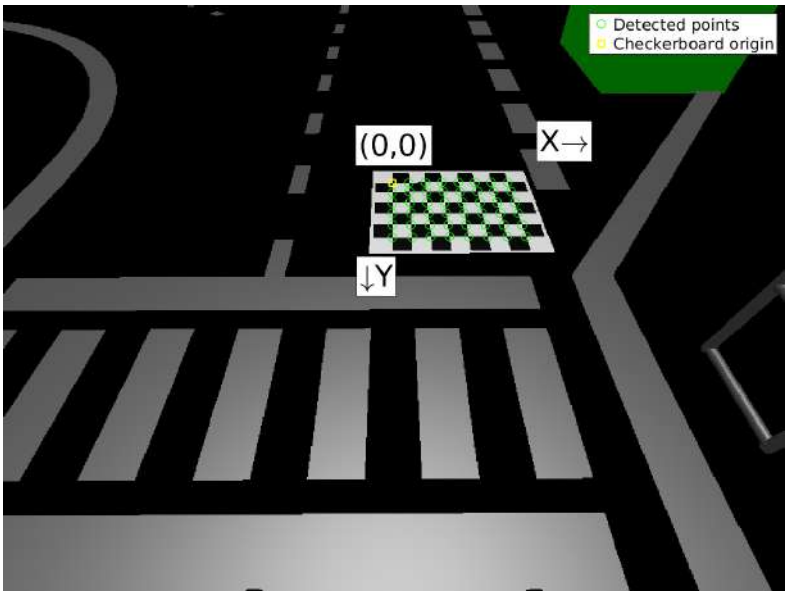


Figure 5.3: Calibration plan for extrinsic parameters

The implementation algorithm for the IPM, as the others implemented and detailed in this dissertation, was developed in *C++* making use of all potentialities of object-oriented programming. In addition, the algorithm uses OpenCV framework and ROS *middleware*. The blocks that integrates the system, work as a system library that permits an easy adaptation of the IPM algorithm to other functionalities.

The implementation algorithm for the IPM, as the others implemented deserving

```
            Standard Errors of Estimated Camera Parameters
            ----------------------------------------------

Intrinsics
----------
Focal length (pixels):    [  398.0146 +/- 0.9886      397.8414 +/- 0.9782  ]
Principal point (pixels):[  401.1549 +/- 0.3759      301.0476 +/- 0.4019  ]
Radial distortion:        [   -0.0024 +/- 0.0018        0.0009 +/- 0.0013  ]

Extrinsics
----------
Rotation vectors:
[   -0.6947 +/- 0.0014       -0.0024 +/- 0.0012       -0.0059 +/- 0.0007  ]
[    0.2250 +/- 0.0021       -0.0027 +/- 0.0019       -0.0001 +/- 0.0007  ]
[    0.2219 +/- 0.0027        0.0009 +/- 0.0021        0.0001 +/- 0.0007  ]
[    ...                       ...                      ...               ]
Translation vectors (mm):
[   -7.9168 +/- 1.1231     -353.1003 +/- 1.1795     1179.7393 +/- 2.9202  ]
[  189.5524 +/- 0.9368     -630.2315 +/- 1.1102      957.2922 +/- 2.6936  ]
[ -344.5125 +/- 0.9122     -630.0656 +/- 0.9839      957.2297 +/- 2.8028  ]
[  ...                      ...                      ...                 ]
```

further detailed explanation in this dissertation, was developed in *C++* making use of all potentialities of object-oriented programming. In addition, the algorithm uses OpenCV framework and ROS *middleware*. The blocks that integrating the system, work as a system library that permits an easy adaptation of the IPM algorithm to other functionalities.

Regarding the algorithm dedicated to perform the IPM, once it is possible that several ROS-nodes composing the autonomous driving system, in particular those blocks related to computer vision, will use this functionality, the `ImageReader.hpp` class allows the system to subscribe a a *ROS-Topic* to receive the image from the camera and provide as the output warped image corrected from perspective distortion only by making use of camera parameters loaded through YAML files.

The process of correcting the distorted image starts by obtaining the *camera-center* – (x_c, y_c, z_c) – which corresponds to the 3D coordinates of camera's reference frame center in relation to the world – (x_w, y_w, z_w).

The inverse of the camera intrinsic matrix (K^{-1}) is used to transform undistorted image points into lines from the camera center. With (K^{-1}):

$$K^{-1} = \frac{1}{f^2 \cdot \alpha} \begin{bmatrix} f \cdot \alpha & -s & p_y \cdot s - p_x \cdot f \cdot \alpha \\ 0 & f & -p_y \cdot f \\ 0 & 0 & f^2 \cdot \alpha \end{bmatrix} = \frac{1}{f_x \cdot f_y} \begin{bmatrix} f_y & -s & p_y \cdot s - p_x \cdot f_y \\ 0 & f_x & -p_y \cdot f_x \\ 0 & 0 & f_x \cdot f_y \end{bmatrix} \quad (5.1)$$

Algorithm 1 Compute IPM Transformation Matrix — preparing the image

Input: R [3×3], t [3×1], distortion coefficients, focal-length, principal point
Output: IPM Warped image size

procedure ComputeIPMMatrix ▷ Parameters loaded in class variables
    camera center ← $-R \times t$ ▷ 3D C_c coordinates
    directional vector ← $R \times K^{-1}$ ▷ Relate points in the image to vectors towards the world
    image limits ← *image size* ▷ Matrix 4×2 getting image corners
    for `image limits` **do**
        limit directional vector ← *directional vector* × *image limits*
        track′s plan Z ← *–camera center/limit directional vector*
        warped image limits ← *camera center* + *limit directional vector* × *track′s plan Z*
    return *warped image limits* ▷ Warped image new dimension

At this stage, a new dimension for the *warped image* is obtained and from this point it is necessary to compute the offset existing between the the distorted original image and the new warped image. This is necessary taking into consideration the output from the extrinsic parameters extracted when the calibration plane was laying down at an known position of the track (figure 5.3).

Algorithm 2 Compute IPM Transformation Matrix — computing offset

Input: R [3×3], t [3×1], distortion coefficients, focal-length, principal point
Output: IPM Warped image size centered

procedure ComputeIPMMatrix ▷ Parameters loaded in class variables
    warped image size ← `MaxMin`$\big($*warped image limits*$\big)$ ▷ Warped image size (u,v)
    $s_{warp} \leftarrow u_{origin}/\big(u_{warped_{max}} - u_{warped_{min}}\big)$
    return *warped image size,scaling factor*

After computing the warped image size from the warped image limits, it is possible to obtained both, *warp translation matrix* a $T_{[4\times3]}$ matrix that will correct the offset between the C_c and the warped image center. Finally, a scaling factor $S_{[3\times3]}$ is obtained that will also relate the size of perspective image to warped image:

$$T_{warp} = \begin{bmatrix} 1 & 0 & u_{warp_{min}} \\ 0 & 1 & v_{warp_{min}} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad S_{warp} = \begin{bmatrix} 1/s_{warp} & 0 & 0 \\ 0 & 1/s_{warp} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

The final step to obtain the *IPM* transformation matrix is combining the intrinsic and extrinsic camera parameters with the scaling and translation corrections that relate the the perspective image with the warped image.

The IPM transformation matrix assumes the following form:

Algorithm 3 Compute IPM Transformation Matrix — transformation

1: **Input**: R [3×3], t [3×1], distortion coefficients, focal-length, principal point
2: **Output**: IPM Transformation Matrix
3:
4: **procedure** ComputeIPMmatrix ▷ Parameters loaded in class variables
5: $Rt \leftarrow$ `Concatenate`$(R, tbig)$
6: $IPM \leftarrow K \times Rt \times T_{warp} \times S_{warp}$
7: **return** IPM

$$IPM_{[3\times3]} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \tag{5.3}$$

$$A = \frac{\frac{r_{31}\cdot(f_x\cdot p_x - p_y\cdot s)}{f_x\cdot f_y} - \frac{r_{11}}{f_x} + \frac{r_{21}\cdot s}{f_x\cdot f_y}}{s_{warp}} \qquad B = \frac{\frac{r_{32}\cdot(f_x\cdot p_x - p_y\cdot s)}{f_x\cdot f_y} - \frac{r_{12}}{f_x} + \frac{r_{22}\cdot s}{f_x\cdot f_y}}{s_{warp}}$$

$$\begin{aligned} C = \frac{t_x}{f_x} &- u_{min}\cdot(\frac{r_{31}\cdot(f_x\cdot p_x - p_y\cdot s)}{f_x\cdot f_y} - \frac{r_{11}}{f_x} + \frac{r_{21}\cdot s}{f_x\cdot f_y}) \\ &- v_{min}\cdot(\frac{r_{32}\cdot(f_x\cdot p_x - p_y\cdot s)}{f_x\cdot f_y} - \frac{r_{12}}{f_x} + \frac{r_{22}\cdot s}{f_x\cdot f_y}) \\ &- \frac{t_z\cdot(f_x\cdot p_x - p_y\cdot s)}{f_x\cdot f_y} - \frac{s\cdot t_y}{f_x\cdot f_y} \end{aligned}$$

$$D = \frac{r_{21}}{f_y} - \frac{\frac{p_y\cdot r_{31}}{f_y}}{s_{warp}} \qquad E = \frac{r_{22}}{f_y} - \frac{\frac{p_y\cdot r_{32}}{f_y}}{s_{warp}}$$

$$F = \frac{t_y}{f_y} + u_{min}\cdot big(\frac{r_{21}}{f_y} - \frac{p_y\cdot r_{31}}{f_y}big) + v_{min}\cdot big(\frac{r_{22}}{f_y} - \frac{p_y\cdot r_{32}}{f_y}big) - \frac{p_y\cdot t_z}{f_y}$$

Finally the `ImageReader.hpp` also provides three class methods: one that permits to convert a coordinate in the world into a pixel coordinate in the warped image – $(x_w, y_w, z_w) \mapsto (u_{warp}, v_{warp})$ – `WordCoordinate2WarpedImagePoint(world coordinate)`;

and another that converts a pixel coordinate in the warped image into a world coordinate – $(u_{warp}, v_{warp}) \mapsto (x_w, y_w, 1)$ – `WarpedImagePoint2WorldCoord(pixel coordinate)`.

Algorithm 4 IPM coordinate transformation

```
1: Input: homogeneous coordinate
2: Output: homogeneous coordinate
3:
4: procedure WordCoordinate2WarpedImagePoint(world coordinate)
5:     img_coord ← IPM_W2w × world_coord
6:     return img_coord
7: procedure WarpedImagePoint2WorldCoord(pixel coordinate)
8:     world_coord ← IPM_w2W × img_coord
9:     return world_coord
```

Where:

$$IPM_{W2w} = \begin{bmatrix} s_{warp} & 0 & -u_{min} \\ 0 & s_{warp} & -v_{min} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad IPM_{w2W} = \begin{bmatrix} \frac{1}{s_{warp}} & 0 & u_{min} \\ 0 & \frac{1}{s_{warp}} & v_{min} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

The third class method is refers to distance measurement `Distance2Robot(pixel coordinate`. Taking into consideration the calibration parameters estimates the world reference frame rotated in *pitch* and *yaw* in relation to the true World reference frame of FNR-ADC Simulation Environment. So the calculation process shall take this into account.

Algorithm 5 IPM image distance calculation

```
1: Input: homogeneous coordinate
2: Output: distance
3:
4: procedure Distance2Robot(pixel coordinate)
5:     C_coord ← R × t
6:     θ ← π
7:     ψ ← −1/π
8:     R_ref ← R_θ × R_ψ
9:     distance ← R_ref × C_coord + WarpedImagePoint2WorldCoord(pixel coordinate)
10:    return distance
```

5.2 Trajectory following and Motion Planning

As mention in chapter 3, the scope of this project is to propose a trajectory-planning and a trajectory tracking for a *4-wheel skid-steer* (4WSS) platform. Despite the many

physical and construction advantages, the 4WSS platform locomotion model brings added difficulties in terms of motion control and odometry.

The change of direction - *heading* - is achieved by controlling the speed of the left and right groups of wheels. But, as the wheel-set is aligned longitudinally with the sides of the platform, resembling the crawler configuration, the change of direction implies the existence of sliding at the wheel contact point (non-ideal rolling).

The *skid-steer* platform kinematics model is not intuitive, and it is not possible to accurately predict the actual movement of the robot with only the control inputs due to the strong effect of friction strongly correlated with each specific wheel-floor interface.

Assumptions such as the ideal rolling and non-slippage considered in the most typical kinematic models of non-holonomic platforms are not applicable in the *skid-steer* platform kinematic model.

In this context, an algorithm is proposed observing the kinematic properties of a *skid-steer*-type robotic platforms, based on the projection of arc-circle trajectories. The proposed solution produces a continuous trajectory, which is adjusted to a certain path defined by local parameters [72].

5.2.1 Control Inputs

One of the central problems concerning autonomous robotic systems is the definition of control instructions from which low-level commands are produced such as linear and/or angular speeds. The complexity of the *skid-steer* platform dynamic model makes it unfeasible to apply it, in real-time, to the formulation of control instructions and navigation solution based on *dead-reckoning*.

The control inputs defined for the platform are linear and angular velocity. The control inputs are then translated to angular speeds for the left and right wheel-set using the formulations of the kinematic model.

$$v = \frac{v_r + v_l}{2} \tag{5.5}$$

$$w = \frac{v_r - v_l}{2b} \tag{5.6}$$

$$v = v + bw \tag{5.7}$$

$$v = v - bw \tag{5.8}$$

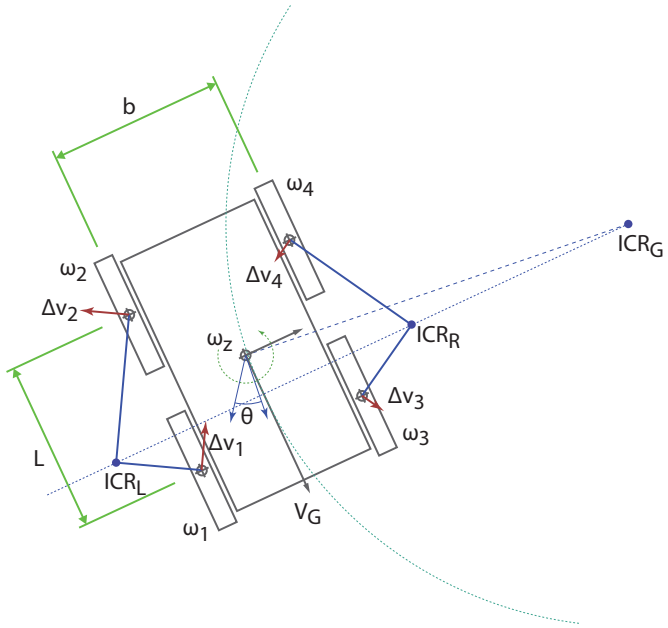


Figure 5.4: 4-Wheel skid-steer kinematic model

5.2.2 *Skid-steer* platforms kinematic model

The picture 5.4 shows the kinematic model of the *skid-steer* platform. In this model the following assumptions are considered:

- Center of mass is coincident with the geometric center of the robot;
- The two wheels on each side of the robot have the same angular speed (mechanically coupled together);
- The 4 wheels are in permanent contact with the floor.

The robot's reference frame (x, y) moves in the track's reference frame (X, Y) at a linear velocity expressed by $v = [v_x, v_y, 0]^T$ and an angular velocity expressed by $w = [0, 0, \omega]^T$. Where $q = [X, Y, \theta]^T$ is the state of the robot in the track reference system (robot configuration), $q' = [\dot{X}, \dot{Y}, \dot{\theta}]^T$ corresponds to the combination of linear and angular velocities of the robot in relation to the track reference frame system is obtained by:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{5.9}$$

By decomposing the control instructions into angular velocity of left and right pair of wheels, we have: $w_l = w_1 = w_2$ e $w_r = w_3 = w_4$. The control instructions will then take the following structure:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = f \begin{bmatrix} w_l \\ w_r \end{bmatrix} \tag{5.10}$$

When the robot is moving, Instant Rotation Centers (ICR) are set to the left and right of the traction – ICR_L and ICR_R respectively – and in relation to the robot's reference frame – ICR_G. ICR_L, ICR_R and ICR_G are on a line parallel to the X axis of the robot reference frame [72].

The change of the *heading* of *skid-steer* platforms describes circular paths. The movement described has the center of rotation (ICR_G) coincident to the center of circle. In opposition, the robot describes a straight line when the angular velocity (ω) tends to zero. In this case, there is no change in the robot's *heading*.

$$\lim_{\omega \to 0} ICR = \infty \tag{5.11}$$

The relationship between the circular movement of radius R and the velocity on each wheel group is described by the function:

$$R = \frac{b}{2} \left(\frac{vr + vl}{vr - vl} \right) \tag{5.12}$$

If the angular velocity is other than zero ($vr \neq vl$), the robot describes a circular arched-path with center in ICR. Thus, we can consider that the arc and linear movements constitute the set of primitive maneuvers of a robotic platform of the type *skid-steer* [73].

5.2.3 Arc model with dynamic adjustment

The dynamically adjustable arc model is intended, based on the primitive models of locomotion (rectilinear path and arc of circumference), to establish control instructions that allow the robot to follow a path in a continuous and fluid manner.

An arc-shaped trajectory is a consistent approach to the trajectory described by the *skid-steer* robot. Thus, basic forms of circumference produce arc-shaped trajectories defined by the centers of curvature and radii dynamically projected along the local path or path segment – *path segment*.

Given the competition assumptions, the system is initialized with knowledge of the geometric map of the track. In the same way, the missions associated to the competition runs, defines intrinsically the starting and finishing points of each run – W_{start} and W_{finish} – as well as the constraints of how the platform should reach W_{finish}, respecting one or several paths defined by the geometry of the track. These assumptions allow the definition of *waypoints* that characterize the basic path to travel [74].

In the proposed model, the path consists of *waypoints* that correspond to a set of local settings. In addition to the coordinates information in the global reference frame – X, Y – each *waypoint* integrates information relative to the robot's distance to the local target (D_n), the robot's projection distance on the path to the local target (d_n), Boolean information regarding the accessibility of the local target ($access_n$), information regarding the volatility of the target ($permanent_n$) and $index_n$ the order number of the *waypoint*.

$$waypoint_n = (x_n, y_n, D_n, d_n, access_n, permanent_n, index_n) \tag{5.13}$$

5.2.4 Path Following

The proposed *path following* algorithm is based on the dynamic arc circumference adjustment that allow to keep the robot in the path to travel. With this strategy it is possible to reduce the control instructions to the appropriate velocities – linear and angular – in such a way that minimal slippage is obtained and yet ensure a smooth and fluid path.

Two consecutive *waypoints* define a segment of the path to be traveled. Each path segment traveled consists of a starting point, $waypoint_n$ (W_n) with coordinates (X_n, Y_n) and an end point W_{n+1} with coordinates (X_{n+1}, Y_{n+1}).

A set of *waypoints* are loaded into the system at a sufficient number suitable for the $pure - speed$ run. In the presence of obstacles, the algorithm generates new *waypoints*, redefining the trajectory, and in some cases, initial *waypoints* may be considered inaccessible by the overlapping obstacles.

This initial set of *waypoints* will be adjusted in real-time as *waypoints* may get considered inaccessible as long as the system detects overlapping obstacles. Under these

circumstances, new *waypoints* – *temp_waypoints* – will be generated in the process of *collision_checker* and *obstacle_avoidance*.

The linear and rotational speeds in the workspace are described by the function:

$$\begin{bmatrix} v \\ w \end{bmatrix} = f(x_{w1}, y_{w1}, x_{w2}, y_{w2}) \tag{5.14}$$

Trajectory tracking approaches the pure pursuit model [75]. Thus a local target $P_r = (x_r, y_r)$ is defined by projecting the position of the robot $P_R = (x_R, y_R)$ at an anticipated distance l_{ahead}. The parameter l_{ahead} will affect the robot's trajectory type, such that a reduced l_{ahead} implies a more reactive trajectory, but potentially closer to the defined path, while a larger l_{ahead} implies a greater distance for stabilization and consequently a smoother trajectory, however this larger l_{ahead} may allow the robot to get more distant form the desired path.

P_r is obtained by calculating the internal product between the unit vector that describes the path segment $\vec{u} = \langle W_{n+1} - W_n \rangle$, direction of the path to follow, and the vector that describes the movement of the robot as a function of the path $\vec{a} = \langle P_r - W_n \rangle$. To the internal product is added the scale l_{ahead} which corresponds to the anticipation distance.

$$P_r = \vec{u} \cdot \vec{a} + l_{ahead} = ||a|| \cos\theta + l_{ahead} \tag{5.15}$$

It is defined as *workspace* W all the space where the robot can operate. The white area shown in the figure 5.5 represents the blocked B space and is interpreted as an obstacle. The gray area (B space) represents a hybrid zone and can be converted to free space F (in black) if an obstacle is identified.

The *workspace* W corresponds to all the space where the robot can operate. Interpreted as an obstacle, the white area shown in the figure 5.5 represents the blocked space. The gray area (B space) represents a hybrid zone and can be converted to free space F (in black) if an obstacle is identified.

In the proposed model, the robot with position P_R pursues its projection on the path segment at a distance of l_{ahead}. P_r is permanently located on the path, in particular on the path segment defined by W_n and W_{n+1} or, eventually, on the subsequent path segment, if $|P_r, W_{n+1}| < d_{n+1}$.

The algorithm updates the P_r position every time a new robot localization is received from the $Localization\ Sub - System$ and calculates the arc between P_R and P_r (see figure 5.6). This new arc is tangent to the unit vector with an equal orientation of

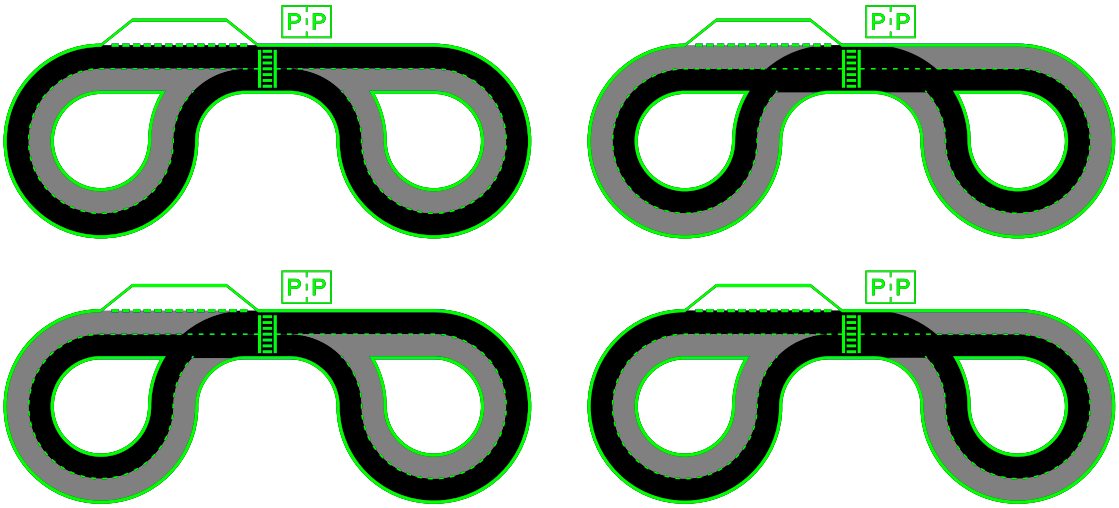


Figure 5.5: Definition of *workspace* (W) and prohibited space (B) - possible configurations

Algorithm 6 Path-Following — target

Input: P_R, W_n, W_{n+1}, l_{ahead}, $index_{seg}$
Output: P_r

procedure ComputeTarget($index_{seg}$)
    $p_{seg} \leftarrow$ Points2Segment(W_n, W_{n+1}) ▷ Converts 2 points into a segment
    $dir_{vec} \leftarrow$ Points2Segment(W_n, P_R)
    $P_{proj} \leftarrow \widehat{p_{seg}} \times dir_{vec}$
    $P_r \leftarrow P_{proj} + l_{ahead}$
    return P_r

the θ_R robot and origin at P_R and is limited by P_R and P_r. This strategy based on the primitive movements characteristic of the *skid-steer* platforms allows a smooth and continuous trajectory, with permanent correction of the θ_R as well as the *cross-track* error, this is, the distance between robot's position and its orthogonal projection on the path segment.

The robot tries to hit the target P_r, taking into consideration its *heading* given by θ_R in the W workspace. The correction of θ_R to reach P_r with the arc-circle projection that intersects P_R and P_r, tangent to the unit vector starting on P_R and angle θ_R.

The circle in which the arc P_R and P_r belongs to, has a secant with length d_{sec} (see figure 5.6). This way, the center of the circle c_{arc} will correspond to the intersecting point between the bisecting line passing through d_{arc} and the orthogonal segment to robot's direction vector and passing through P_R.

The angle formed by $\angle\, P_R\, c_{arc}\, P_r$ corresponds to the turning angle necessary for the robot to reach P_r.

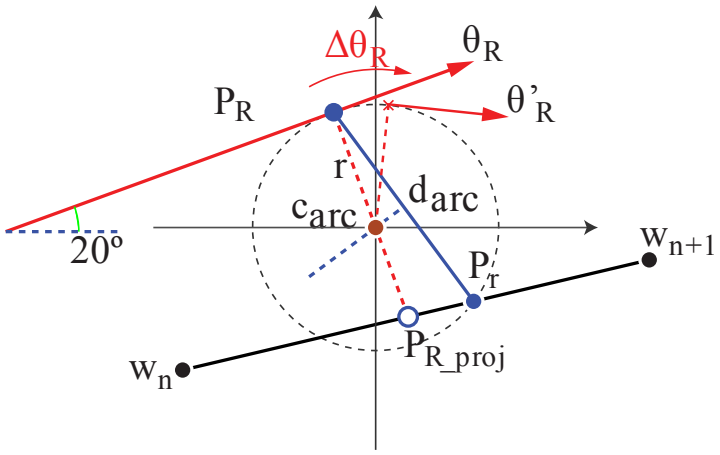


Figure 5.6: Path-Following based on arc-model with dynamic adjustment

Since the control inputs are linear and angular velocity, it is possible to determine, at each control clock cycle, the new position of the robot in the arc defined by P_R and P_r.

Algorithm 7 Path-Following — arc-fitting

Input: P_R, P_r, R_θ, R_{vel}
Output: P_{R_ω}

procedure ComputeAngularVelocity
  $d_{sec} \leftarrow$ TrajectoryFromPoint(Slope(P_R, P_r)) ▷ Segment(M,b) between robot and target
  $d_{sec_{bisect}} \leftarrow d_{sec}/2$ ▷ Bisector point of segment robot target
  $d_{sec_{orth}} \leftarrow$ TrajectoryFromSlope($-1/$Slope$(P_R, P_r), d_{sec_{bisect}}$) ▷ Orthogonal to d_{sec}
  $r_{dir_{orth}} \leftarrow$ Make1st4thQuadrant($P_{R_\omega} + \frac{\pi}{2}$) ▷ Robot's orthogonal heading – $[-\frac{\pi}{2}, \frac{\pi}{2}]$
  $c_{arc} \leftarrow$ SegmentIntersect($d_{sec_{orth}}, r_{dir_{orth}}$)
  $r \leftarrow \| P_R, c_{arc} \|$ ▷ Distance of curving center
  $P_{R_\omega} \leftarrow r = 0$?BodyCenteredRotation() : R_{vel} / r
  return P_{R_ω}

With the new position of the robot $P_R^{'}$ a new target $P_r^{'}$ is recalculated that progresses on the segment of the path defined by W_n and W_{n+1}.

Once the new robot and target positions, the $P_R^{'}$ and $P_r^{'}$ respectively, are calculated, then their are evaluated if these same locations fall inside the W_{n+1} neighborhood area. This is the same to say, if the distance of $P_R^{'}$ to W_{n+1}, and $P_r^{'}$ to W_{n+1} is less than D_{n+1} and d_{n+1} respectively (both defined in the *waypoint* configuration).

If the locations of $P_R^{'}$ and $P_r^{'}$, independently evaluated are:

$$\left|P_R^{'} - W_{n+1}(x,y)\right| < D_{n+1} \tag{5.16}$$

Algorithm 8 Path-Following — path progression

Input: P_R, P_r, W_n, W_{n+1}, D_{n+1}, d_{n+1}
Output: P_r, $index_{seg}$

procedure ComputeNeighbourhood
    $P_{r_{coord}} \leftarrow P_r \times \widehat{p_{seg}} + W_n$
    $d_{P_R} \leftarrow \| \text{Points2Segment}(P_R, W_{n+1}) \|$
    $d_{P_r} \leftarrow \| \text{Points2Segment}(P_r, W_{n+1}) \|$
    $P_r \leftarrow d_{P_r} \leq d_{n+1} \,?\, \text{ComputeTarget}(index_{seg} + 1) : P_r$
    $index_{seg} \leftarrow d_{P_R} \leq D_{n+1} \,?\, index_{seg} + 1 : index_{seg}$
    return P_r, $index_{seg}$

$$\left| P_r^{'} - W_{n+1}(x, y) \right| < d_{n+1} \tag{5.17}$$

If the neighbourhood/proximity conditions stated in the equations 5.16 and 5.17 are verified, then the trajectory segment to be considered in the following control cycles is defined by W_{n+1} and W_{n+2} until the neighbourhood conditions – D_{n+2} and d_{n+2} – are verified again.

In the transition process for following trajectory segments, the pass-through offset of P_R and P_r is given by the distance of l_{ahead}. As said before, this offset will allow a greater or lesser smoothness in the trajectory traveled by the robot.

5.3 Obstacle Avoidance

Based on the algorithm proposed for *path following*, a solution of *obstacle avoidance* was developed so that, in the presence of obstacles in the track directly interfering the path, could redefine the trajectory taking into consideration the constraints of the track and the specific challenges of the race.

As said before, the proposed *obstacle avoidance* solution is based on the *path-following* model described on the previous section, and as such respects the primitive manoeuvres of the *skid-steer* platforms – linear and arc-circle movements.

The *obstacle avoidance* process is based on the detection of obstacles through LiDAR readings (*ranges*). When a minimum number of readings – whose *range* is less than a certain limit – is collected, then the hypothesis of an existing obstacle in the vicinity of the robot raises, and the *obstacle avoidance* algorithm initiates. Once this condition is met, the algorithm starts the process of positioning the obstacle and verifies if the presence of this object interferes with the defined path or not.

According to the assumptions of the Autonomous Driving Competition of the National Robotics Festival, the obstacle is in the form of a cube with a 400 mm edge

($obst_{with}$). Based on the specifications provided by the regulations and based on the *range* measures from the LiDAR, an algorithm was developed based on geometric associations of the most significant readings to identify and estimate the position of the obstacle on the track ($obst_{center}$).

The process of obstacle fitting initiates when LiDAR's range-finder readings meet the proximity criteria, this means, if the readings, based on current's robot location, place a potential obstacle on the track and at a collision course. If this happens, an obstacle bounding estimation process starts. First, it calculates the slope of the line-segment composed by the left (rng_L) and right range (rng_R) readings with the control range (rng_C). Finally it estimates box's nearest corner using the reading with the smallest range (rng_N). A simple geometric relation described in algorithm 9 is presented as a way generate circular boundaries on obstacles.

Algorithm 9 Obstacle Avoidance — obstacle fitting

Input: $LiDAR_{data}$, $eq_{threshold}$
Output:

procedure ComputeObstacleFit($LiDAR_{data}$)
    $rng_L \leftarrow$ RangeData($LiDAR_{data}$, **left**)
    $rng_R \leftarrow$ RangeData($LiDAR_{data}$, **right**)
    $rng_N \leftarrow$ RangeData($LiDAR_{data}$, **near**)
    $rng_C \leftarrow$ RangeData($LiDAR_{data}$, **control**) ▷ 4th value between near and most distant bearing

$slp_{L\leftrightarrow C} \leftarrow$ FaceSlope(rng_L, rng_C) ▷ Slope between Range Left and Range Control
$slp_{R\leftrightarrow C} \leftarrow$ FaceSlope(rng_R, rng_C)
$slp_{L\leftrightarrow N} \leftarrow$ FaceSlope(rng_L, rng_N) ▷ Slope between Range Left and Nearest
$slp_{R\leftrightarrow N} \leftarrow$ FaceSlope(rng_R, rng_N)

    if $|slp_{R\leftrightarrow C} - slp_{R\leftrightarrow N}| \leq eq_{threshold}$ **then** ▷ $\{rng_R, rng_C, rng_N\} \in$ box face A
        if $|slp_{R\leftrightarrow C} - slp_{L\leftrightarrow N}| \leq eq_{threshold}$ **then** ▷ $\{rng_R, rng_C, rng_N, rng_L\} \in$ same box face
            $rng_{ext} \leftarrow$ RangeData($LiDAR_{data}$, $bearing_n(max)$) ▷ Distant range bearing from rng_N
            $obst_{center} \leftarrow$ ObstacleFitControid(rng_{ext} , $obst_{spec}$)
        else
            $obst_{center} \leftarrow$ ObstacleFitControid($slp_{R\leftrightarrow N}$, rng_L , rng_N , $obst_{spec}$)
    else if $|slp_{L\leftrightarrow C} - slp_{L\leftrightarrow N}| \leq eq_{threshold}$ **then** ▷ $\{rng_L, rng_C, rng_N\} \in$ face B
        if $|slp_{L\leftrightarrow C} - slp_{R\leftrightarrow N}| \leq eq_{threshold}$ **then** ▷ $\{rng_L, rng_C, rng_N, rng_R\} \in$ same face
            $rng_{ext} \leftarrow$ RangeData($LiDAR_{data}$, $bearing_n(max)$) ▷ Distant range bearing from rng_N
            $obst_{center} \leftarrow$ ObstacleFitControid(rng_{ext} , $obst_{spec}$)
        else
            $obst_{center} \leftarrow$ ObstacleFitControid($slp_{L\leftrightarrow N}$, rng_R , rng_N , $obst_{spec}$)
    else
        No obstacle centroid computed
    return $obst_{center}$

Once the position of the obstacle has been estimated, a circle is adjusted with the center coincident with the center of the obstacle $obst_{center} = (x_{obst}, y_{obst})$ and radius T_{radius} given by the expression:

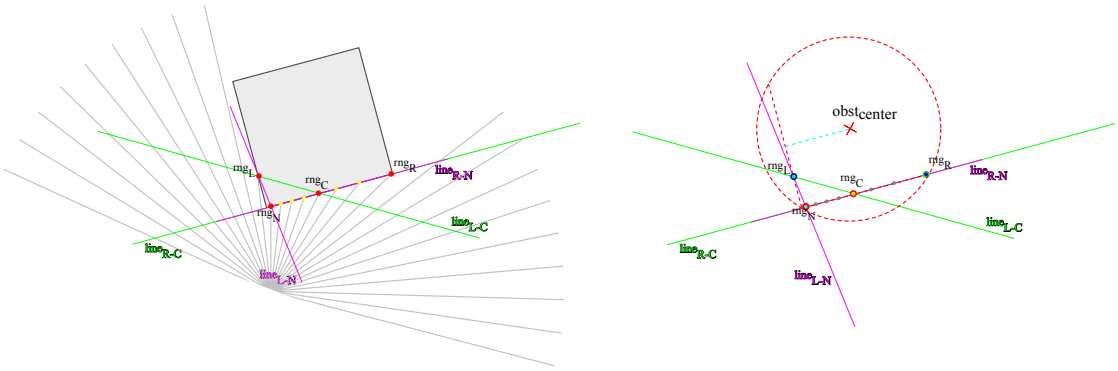


Figure 5.7: Visual solution used on obstacle's circular fit

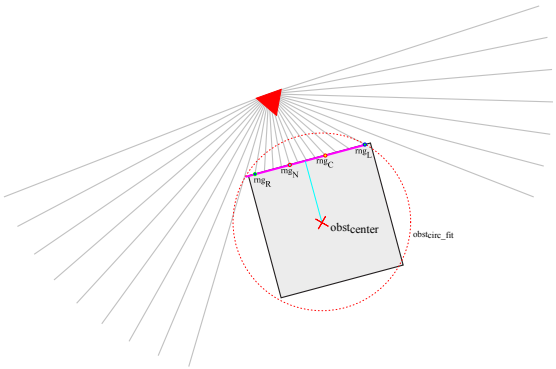


Figure 5.8: Visual solution used on obstacle's circular fit - aligned LiDAR points

$$T_{radius} = \frac{obst_{with}}{\sqrt{2}} + b \tag{5.18}$$

where $obst_{with}$ corresponds to the edge of the obstacle (cube) and b corresponds to the base of the robot wheels. The radius T_{radius} represents the turning radius or the minimum distance at which the robot must initiate an evasive maneuver.

With estimated $obst_{center}$ and T_{radius}, the following conditions are tested:

1. $Waypoints$ accessibility: If the next *waypoint* (W_{n+k}) to the current *waypoint* (W_n) is inaccessible given the presence of an obstacle in the proximity:

$$|obst_{center} - W_{n+k}(x, y)| < T_{radius} \tag{5.19}$$

   If the previous condition is verified, it implies that the path intercepts the obstacle in such a way that it prevents its transposition by the robot, according to Fig. 5.9.

2. Evaluate the interception by obstacles on path-segments: If the segment of the path defined by the current *waypoint* (W_n) and subsequent W_{n+1} (both accessible) intercepts the circle with center $obst_{center}$ and radius T_{radius} in two different points, it is concluded that the obstacle interferes with the path initially defined for the robot.

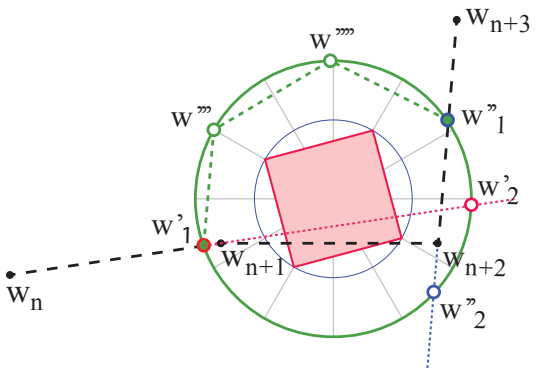


Figure 5.9: *Obstacle Avoidance* - inaccessible *waypoints*

The trajectory is redefined with the addition of four (4) new *waypoints*. Two (2) of them correspond to the intersection points between one (1) or two (2) trajectory segments and the circle describing the obstacle – $W^{'}$ and $W^{''}$. These segments have to obey the condition of an accessible *waypoint*, process illustrated by Fig. 5.10.

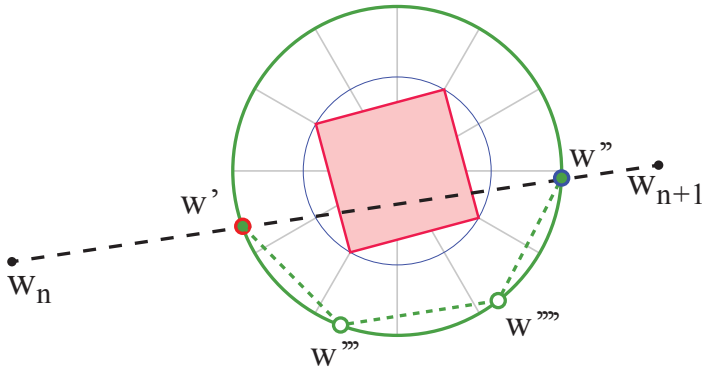


Figure 5.10: *ObstacleAvoidance* - accessible *waypoints*

Later, the smallest arc between $W^{'}$ and $W^{''}$ is defined, and two (2) new *waypoints* – $W^{'''}$ and $W^{''''}$ – are projected on it.

The trajectory is validated if the new *waypoints* $W^{'''}$ and $W^{'''}$ are located in free space F. Otherwise, it is considered the largest arc with a new projection of *waypoints*, $W^{'''}$ and $W^{''''}$.

$Path_i$	$Path_{i+1}$	$Path_{i+2}$
W_n	$W_{n+1}(access = false)$	$W_{n+2}(access = false)$
$W_{n+1}(access = false)$	$W_{n+2}(access = false)$	$W_{n+2}(access = true)$

Table 5.1: $Waypoints$ accessibility

Validated set of new *waypoints* – $W^{'}, W^{''}, W^{'''}, W^{''''}$ – these are included in the set of *waypoints* that defines the path, but these added *waypoints* take on temporary form and they are eliminated at each run ($permanent = false$).

5.4 Parking lot classification

The autonomous driving competition from the Festival Nacional de Robótica is an important mean to evaluate the capacity of competing systems to perform more complex maneuvers that show the robustness of solutions in aspects such as motion-planning or behavior/decision-making. One of the tasks is to carry out parking maneuvers. For this purpose, as already mentioned in chapter3, the competition has two specific areas for this task: parallel parking in a place adjacent to the track; another, which at the level of the track plan is detached from it, bringing with it other challenges and which will be the object of a possible solution for implementation.

To be able to park in the parking lot where, taking into consideration to track's blueprint, is located in an independent/disclosed area, it is necessary to develop specific routines that allow its identification with a high degree of confidence. Another aspect that deserves attention for the development of the proposed algorithm is related to the identification of free parking space (if any) and to estimate which change of heading is necessary to place the robot in the proper parking place and define a set of maneuvers to perform this same parking.

The strategy developed to identify the car parking lot, in particular, its location on the track map, is based on the identification of the letter P inscribed on a rectangular delimiter as illustrated in figure 5.11, as being the most distinctive factor in the identification of this characteristic of the track scenario.

For this purpose, we opted for the use of machine learning techniques and classifiers appropriate to the process of feature recognition and data analysis, in particular, the recognition of symbols. Another advantage of endowing the system with a machine learning algorithm is that it allows easily adapt the system to other contexts and other analyses, not being limited to a solution developed for a specific problem. The machine learning implemented is based on the *K-Nearest Neighbors* (KNN). The algorithm caches

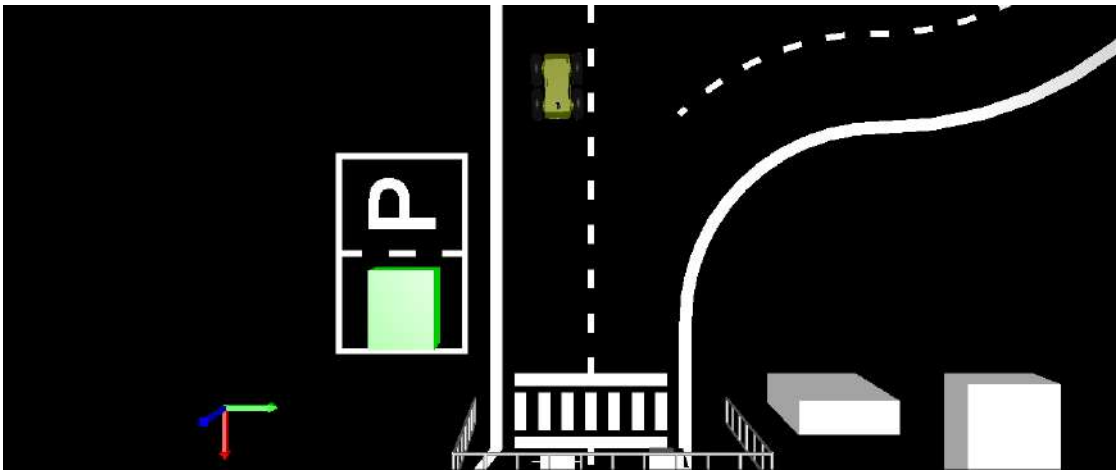


Figure 5.11: Parking lot top view from FNR-ADC Simulation Environment

all the training samples, and predicts the response for a new sample by analyzing a certain number K of the nearest neighbors of the sample, using concepts such as voting, calculating weighted sum, etc.. The KNN method is sometimes referred as "learning by example", this means that, for prediction it looks for the feature vector with a known response that is closest to the given vector.

Opting for features' identification using machine learning techniques will allow the system to be sensitive only to a set of features clearly defined and relevant to the solution. This reduces the possibility that environmental aspects like brightness or background noise, lead to false positives and therefore identifying features wrongly in other possible scenarios of the competition.

On the other hand, this process will also allow identifying which of the tow parking areas is available for parking since, according to the race regulations, it is possible that, one of the available spaces may be occupied.

In order to implement the machine learning based on the supervised learning classification algorithm *K-Nearest Neighbor*, it is necessary to develop two distinguished stages: building the training set and the classification algorithm it self to identify the parking lot.

5.4.1 Creating the training set

Suppose that for the identification of important symbolic features of the competition such as: P character, left-arrow right-arrow and up-arrow to turn left, right or move

forward respectively, including vertical signs, can be approached considering each one of this graphic signage (containing each one of them important information about the type of behaviour that is intended to be observed on the robot), that is, consider each one of these graphic elements as a character.

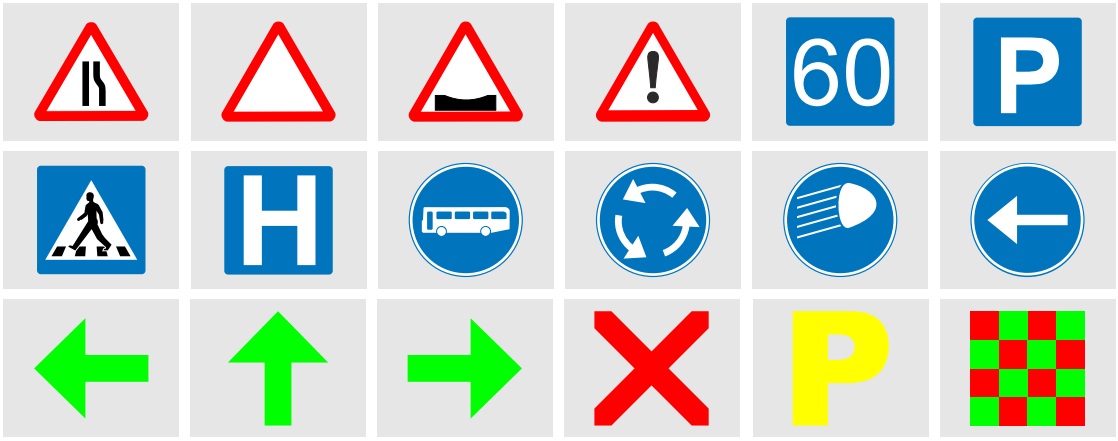


Figure 5.12: Possibles classes to identify all relevant signs present on FNR-ADC Simulation Environment

For each character proposed in figure 5.12, 50 training images of the system will be produced.

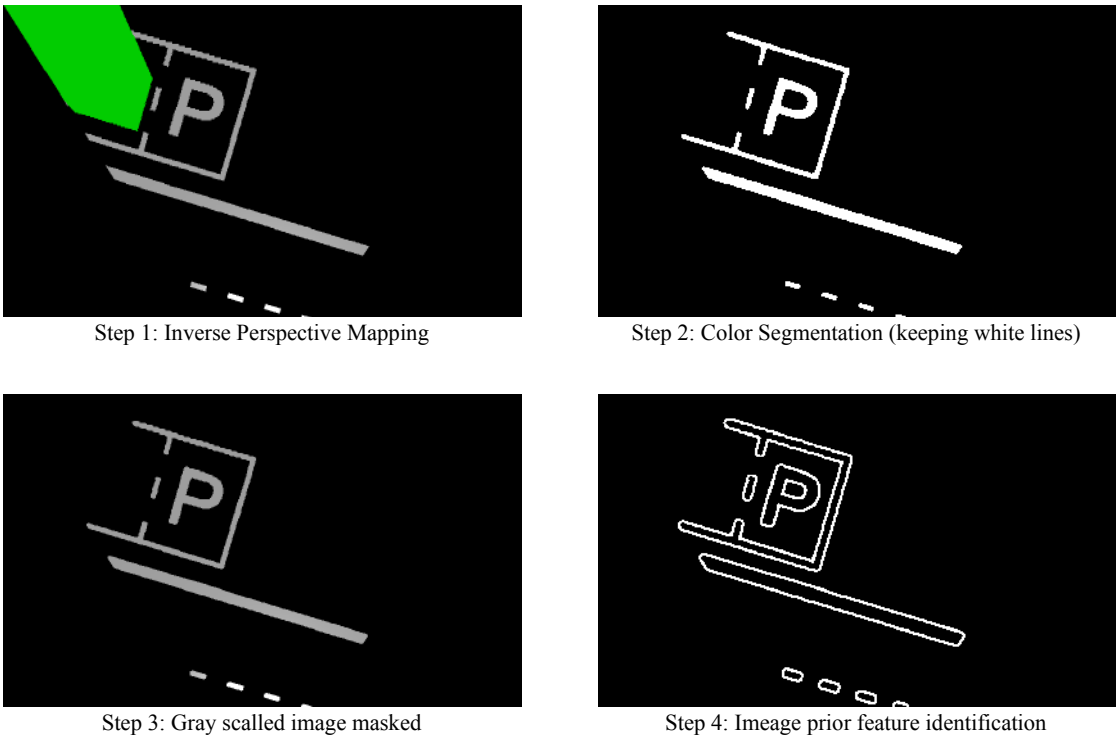
It is typified that each image that will support the classifier training will have 20x30 pixels. The training-set building process will produce two data-structures.

- a first data-set that indexes which group or classification the classified images correspond to — output values;
- a second data-set with the classified images for the knn-training process — feature-vector.

So the implementation has two distinct functionalities: a fist one to develop a training-set process to allow the classifier to train with classification examples necessary to posterior start to classify new objects; and a second functionality with the practical implementation o signs/signals identification.

Likewise the other algorithms already implemented, the coding of this makes also use of OpenCV framework and ROS middleware. The algorithm starts by using the *Inverse Perspective Mapping* focused on the previous section, and the class developed for the feature identification, *inherits* from the class `ImageReader` where is implemented all

the the class methods necessary to correct acquired images from projection distortion, as well as making available dimensional correspondence between image and real world. The `FeaturesTrainig` also *inherits* from the class `ColorSegmentation` to prepare the image to optimally extract features (illustrated through figure 5.13). Below it is transcribed in detail the training algorithm used to build the classifiers and tagging that supports the parking lot identification.



Step 1: Inverse Perspective Mapping

Step 2: Color Segmentation (keeping white lines)

Step 3: Gray scalled image masked

Step 4: Imeage prior feature identification

Figure 5.13: Preparing the IPM image for feature extraction

As mentioned before, the output of this algorithm is two data-set in *Extensible Markup Language* (XML), a markup language that defines a set of rules to describe and identify information accurately and unambiguously, in a way that computers can be programmed to interpret the information.

Listing 5.1: Output file with list of classification values

```
<opencv_storage>
    <classifications type_id="opencv-matrix">
        <rows>20</rows>
        <cols>1</cols>
```

```
        <dt>i</dt>
        <data>
            120 120 112 120 120 120 112 120 120 120 112 120 120 120 112↩
                120 120 120 112 120
        </data>
    </classifications>
</opencv_storage>
```

The *classification* are defines in a $[1 \times k]$ where k is the number o training sequences done. In this example it is 20.

Listing 5.2: Output file with list features-vector

```
<opencv_storage>
    <images type_id="opencv-matrix">
        <rows>20</rows>
        <cols>600</cols>
        <dt>f</dt>
        <data>
            154. 4. 4. 4. 4. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ↩
                0. 255. 255. 255. 255. 255. 13. 13. 13. 13. 13. 0. 0. 0.↩
                 0. 0. 0. 0. 0. 0. 0. 255. 255. 255. 255. 255. 255. ...
        </data>
    </images>
</opencv_storage>
```

Due to practicability purposes, the output of the images training process is not complete but it possible to understand its structure:

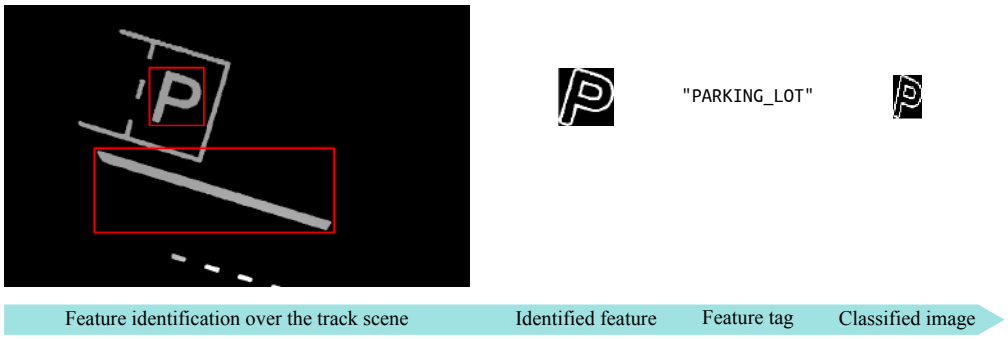


Figure 5.15: K-Nearest Neighbor learning process

Once the creation of the training-set is completed it is possible to move on to the second stage: parking-lot identification using classification methods.

5.4.2 Parking-lot identification

The method used to identify the parking-lot trains a statistical model using a set of input features-vector as illustrated in XML listing 5.2 and the corresponding output values (classes) as in the XML listing 5.1. Both input and output vectors/values are passed as matrices. As said, the input feature vectors are stored as train data rows, that is, all features of the training vector are stored continuously. The train data must have the `CV_32FC1` (32-bit floating-point, single-channel) format. The classes are also stored in a 1D row vector in `CV_32SC1` or `CV_32FC1` format.

The solution for the parking-lot identification fits the criteria associated to classification problems, therefore the responses are discrete class labels. As Machine Learning (ML) models, *K-Nearest Neighbor* is trained on a selected feature of the training-set (`kNearest->train`).

After the training has been done, it is possible the system to start identifying the parking-lot among other features captured by the cameras. The images are acquired and processes (color space, color segmentation, filtering, binarization) and thereafter checking if any feature captured by the camera matches with the parking-lot characteristics. The mathematical method described in Section 4.2 (Subsection: 4.2.2) is then used to find if there is a positive match or not (`kNearest->find_nearest`).

In case of positive mach, in this particular solution the sign on the floor **P** marking the parking-lot is found, the algorithm estimates the parking area rectangular boundaries $P_{lot}(x_n, y_n)$ and finds its center $P(x, y)$ following the expression 5.20. The center found will be used as a *temporary waypoint* (see Section 5.2 expression 5.13 in Chapter 5). Alongside with the parking-lot's center calculation, an orientation of the parking-lot based robot's position is also estimated – $\theta_{R \to P}$.

$$[h!]P(x, y) = \frac{\sum_{n \in [1,4]} P_{lot}(x_n, y_n)}{4} \tag{5.20}$$

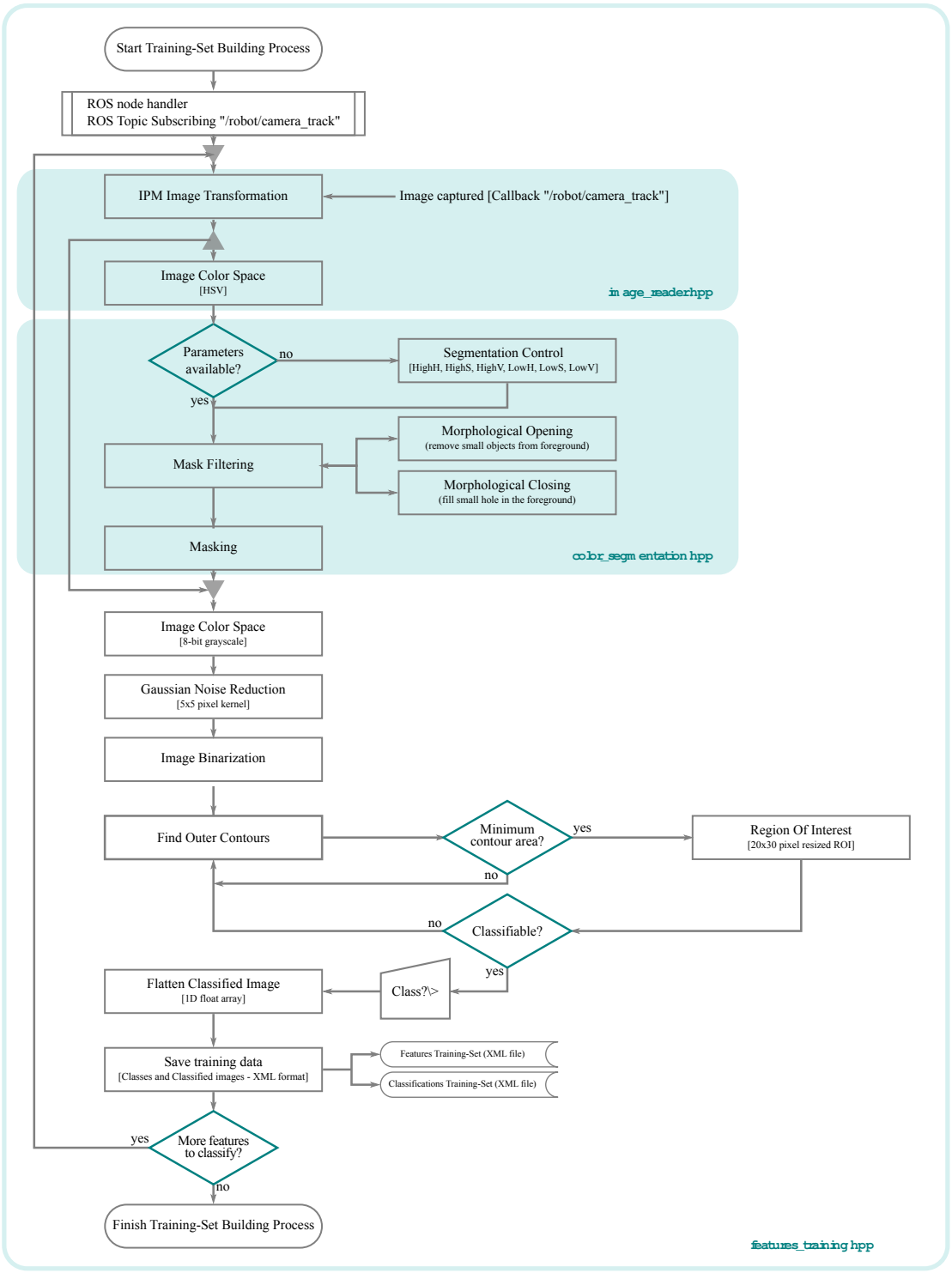


Figure 5.14: Low-level implementation of training-set build algorithm

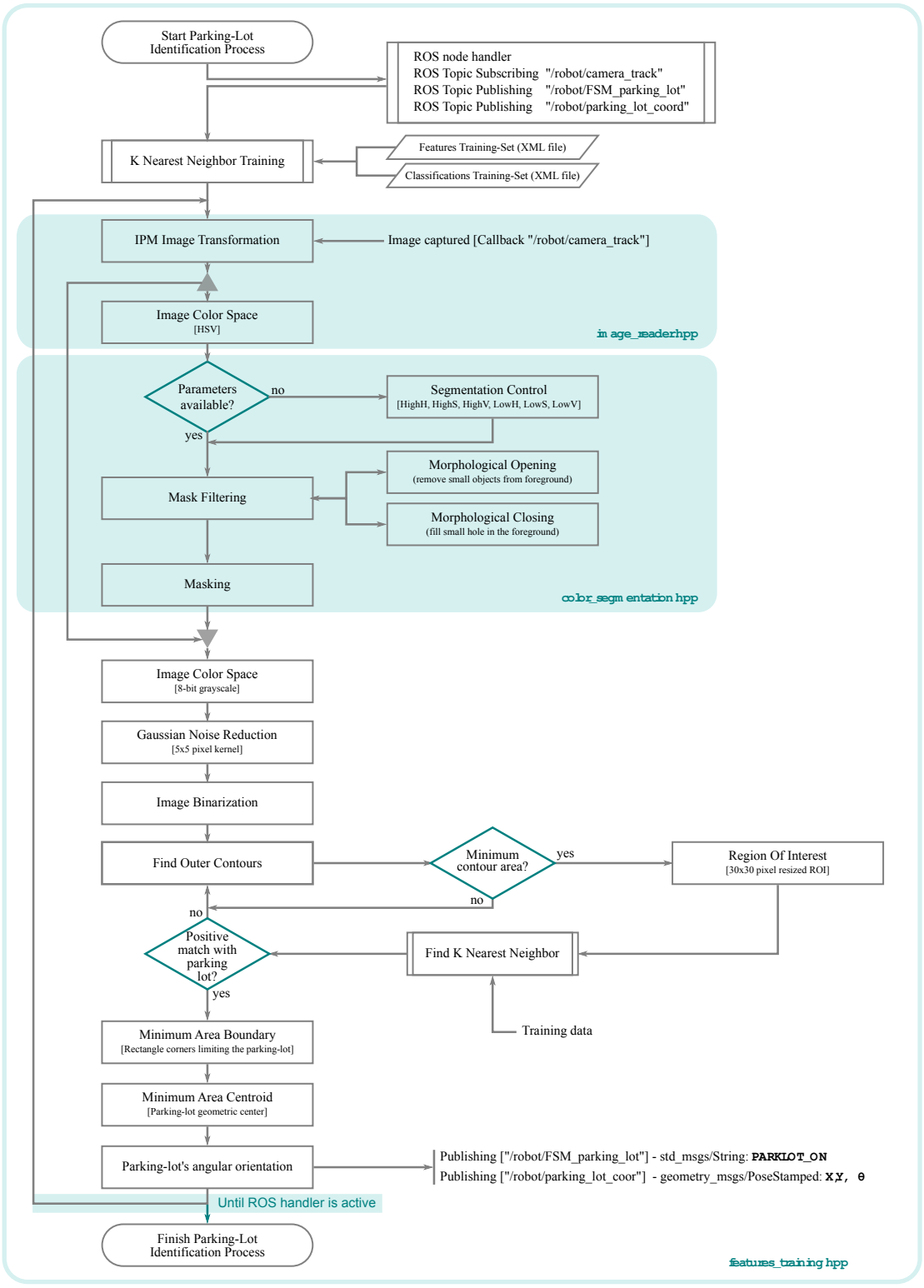


Figure 5.16: Low-level implementation of parking-lot classification algorithm

Chapter 6

Results

This chapter will describe the results obtained from the various proposals implemented in response to the problems described in chapter 3, namely: inverse perspective mapping and obtaining direct information about distances; path-following and obstacle avoidance based on dynamic arc fitting; and parking-lot identification and estimate its pose concerning robot's position. The results will be based on data with sufficient relevance for the validation of the concept as well as, whenever possible, enunciate potential weaknesses.

6.1 Path Following and Obstacle Avoidance

The current section will be dedicated to present the results regarding the performance of the *path following* and *obstacle avoidance* algorithm implemented.

To evaluate the *path following* strategy implemented, the algorithm was tested in simulation run-time, validating the capacity to running the entire track with a dynamic arc fitting technique.

Relatively to the evaluation of the *obstacle avoidance* strategy, due to computational limitations, it becomes unfeasible to simulate the algorithm with the FNR-ADC Simulation Environment. Because of the high-computation requirements, it was not possible to enable the emulated LIDAR from the MORSE simulation engine and yet having a reasonable resolution and refresh rate. To overcome this limitation, the *obstacle avoidance* algorithm was tested using MATLAB. For evasive maneuver simulation, a LIDAR and its perception block were recreated, providing the basic feature associated with this kind of sensor.

6.1.1 Path-following

The algorithm developed for the *Path-following* of a 4WSS platform, based on the dynamic arc fitting, was tested with FNR-ADC Simulation Environment at several speeds, without obstacles. The performance was observed and the validation criterion was to ensure the robot performs the whole track with an average velocity stipulated, without leaving the lane boundaries.

The following visualizations acquired from RVIZ, a 3D visualization tool for ROS, illustrates the performance of making the robot traveling at 2-meters per second on a *pure-speed* run.

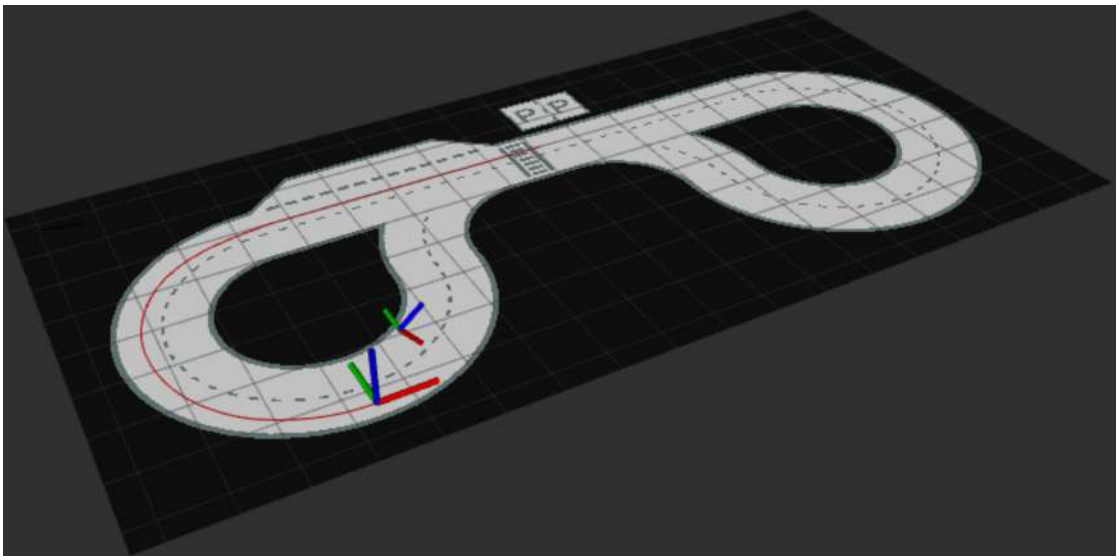


Figure 6.1: Path-following simulation - RVIZ visualization

As it is possible to visualize from the top-view visualization (figure 6.4, there is a slight deviation from the optimal course due to incorrect mapping of 2 waypoints defining the pre-established path. This is a correction to be made on the core-waypoints set.

The overall behavior is stable although, in certain situations the robot acts abnormally, maybe due to some sort of *bug* caused by the mesh of the track scene. In fact, on certain occasions, the existing bug on the simulation scenario forces the robot to flip over, a situation that does not occur with other simulation scenarios.

It was also possible to verify that it is critical the adjustment between the `Motion-Control node rate` and the `Path-Following node rate`, to turn the system capable to achieve higher velocities.

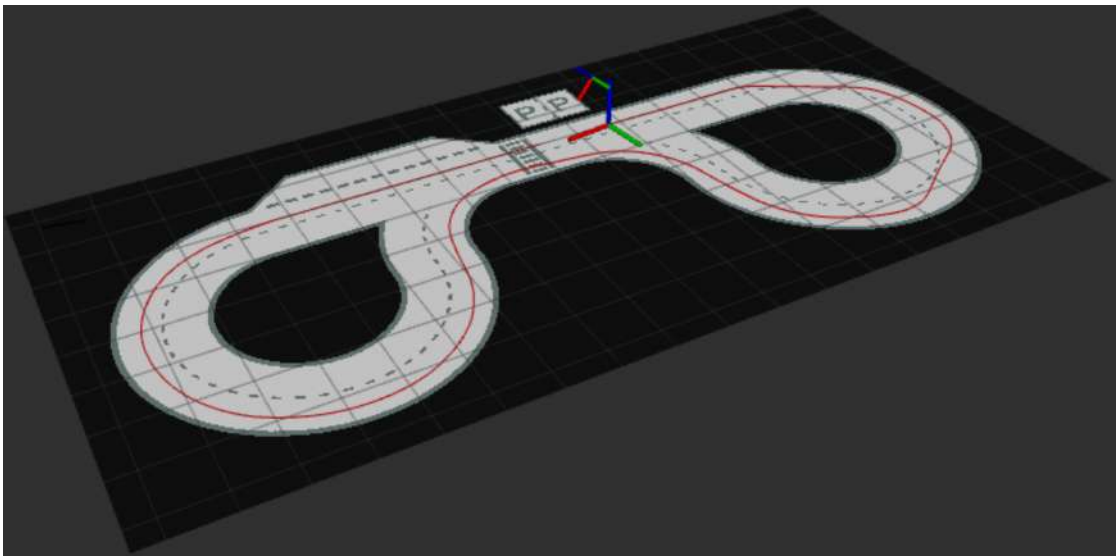


Figure 6.2: Path-following simulation - RVIZ visualization (full run)

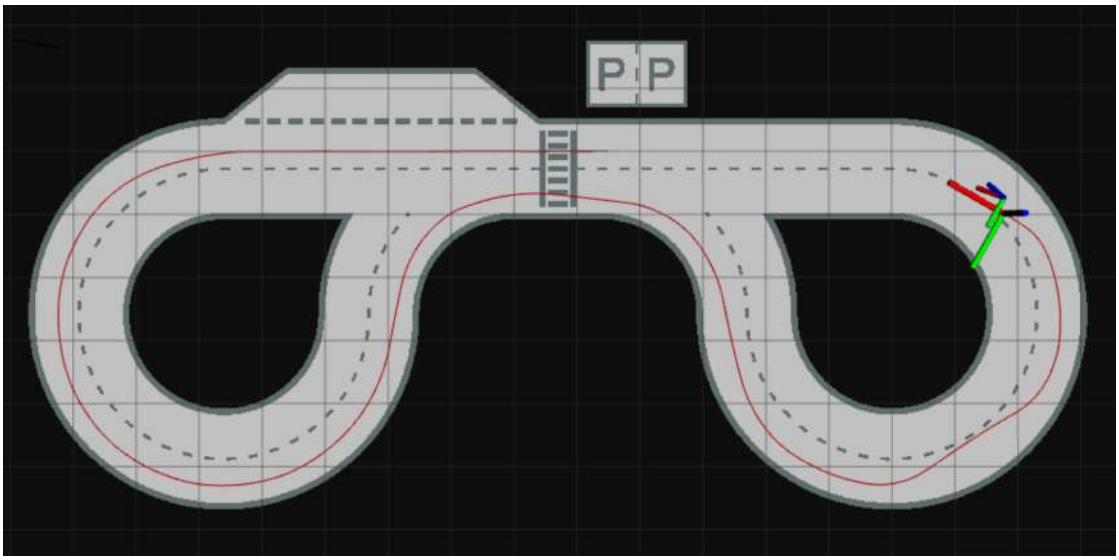


Figure 6.3: Path-following simulation - RVIZ visualization (Top View)

Likewise, the *PID Controller* responsible to control the robot's running velocity at a target set requires better tuning to ensure a more stable behavior at higher speeds.

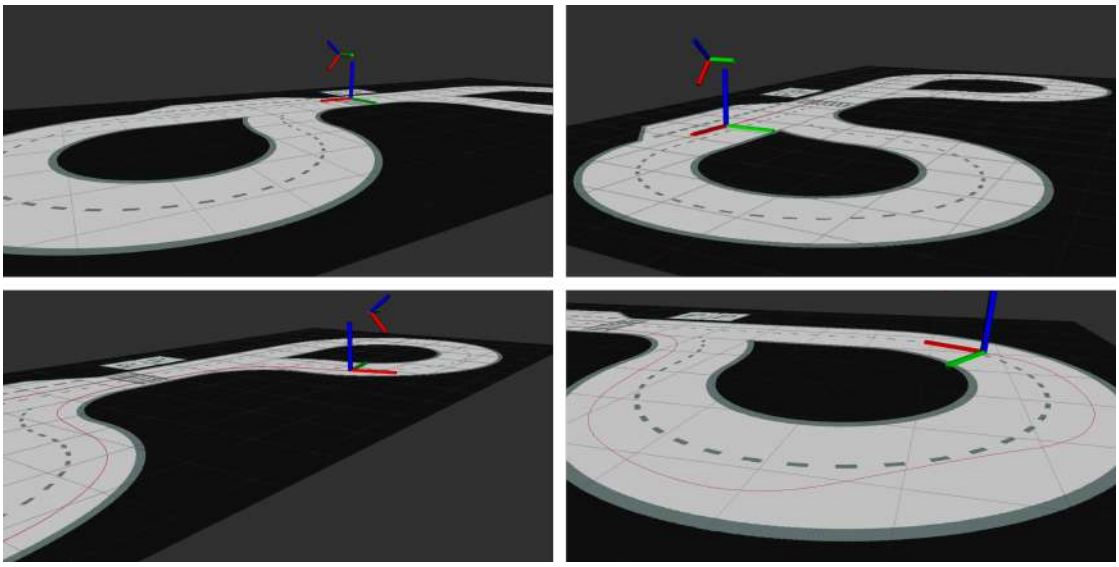


Figure 6.4: Path-following simulation - RVIZ visualization (Top View

6.1.2 Obstacle Avoidance

The following MATLAB plots illustrate the obstacle avoidance strategy getting into action. To ensure more lean behavior, the algorithm only processes LIDAR ranges inferior to 3 meters. As more readings feed the system, more accurate is the obstacle fitting. The closer the reading, the denser is the data received from the LIDAR, also making the obstacle fitting more accurate.

When the simulated LIDAR coded in *Matlab* obtains a range inferior of 3 meters it starts to estimate the position of the obstacle and to understand if it interferes with the defined path. This moment is possible to verify on figure 6.5 when the vector defining robot's heading changes color from red to blue.

In this particular example, the obstacle, complying with the FNR-ADC regulations, was placed on the track, with the condition that one of the original waypoints that establishes the path becomes inaccessible. Through figure 6.5 and with more detail in figure 6.9 , it is possible to visualize the inaccessible waypoint, close to the center of the green box, and also the newly generated waypoints necessary to overpass the obstacle and return to the original path once saved.

In these simulated conditions, the newly generated waypoints represent both, the shortest path to overpass the obstacle and yet keeping the robot inside the track.

The new waypoints shall obey to the criterion that all of them shall be located in a

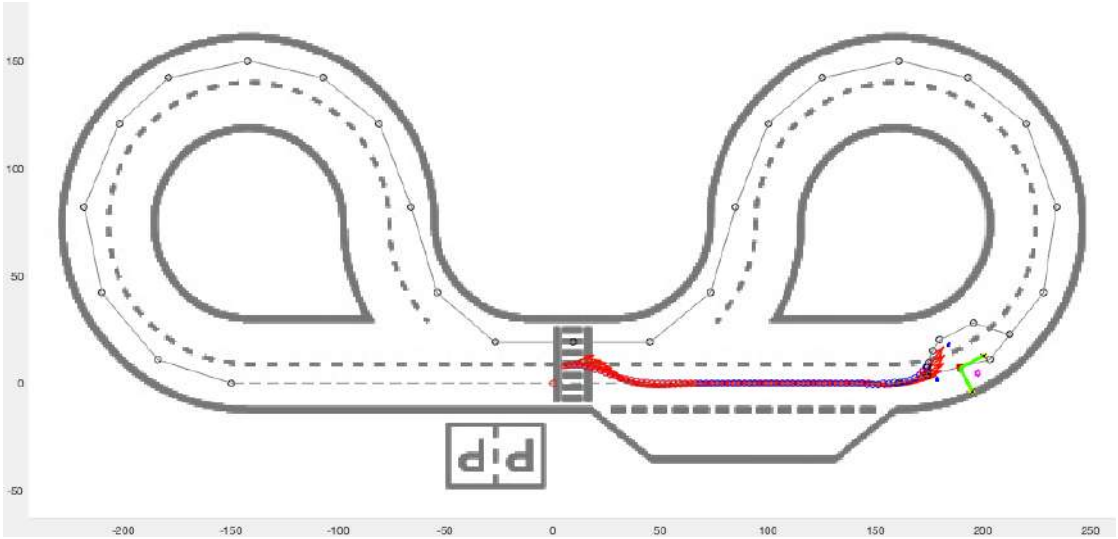


Figure 6.5: Obstacle identification and new waypoints produced

free and accessible workspace, in short, inside the track.

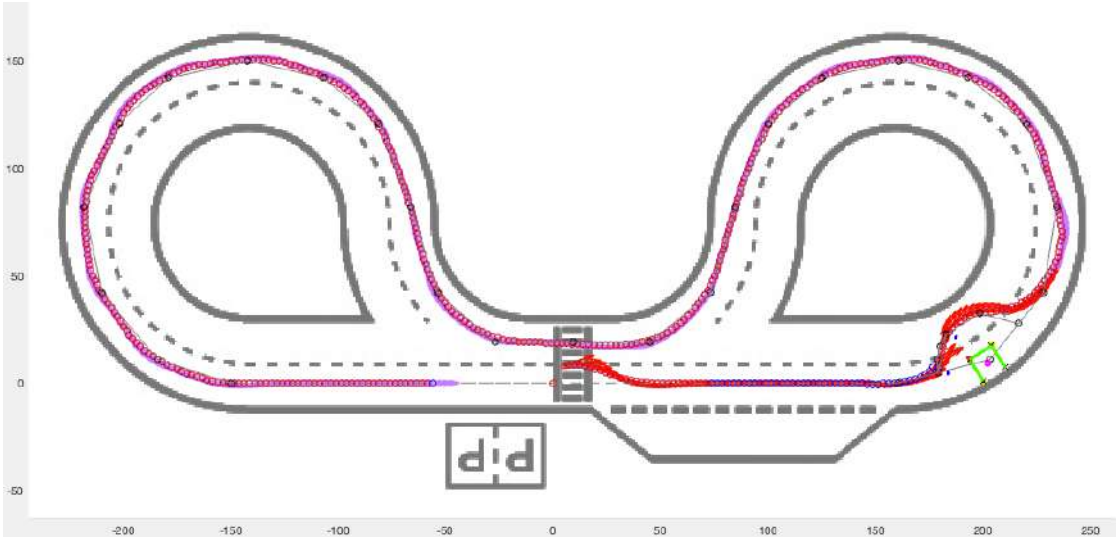


Figure 6.7: Obstacle identification and new waypoints produced

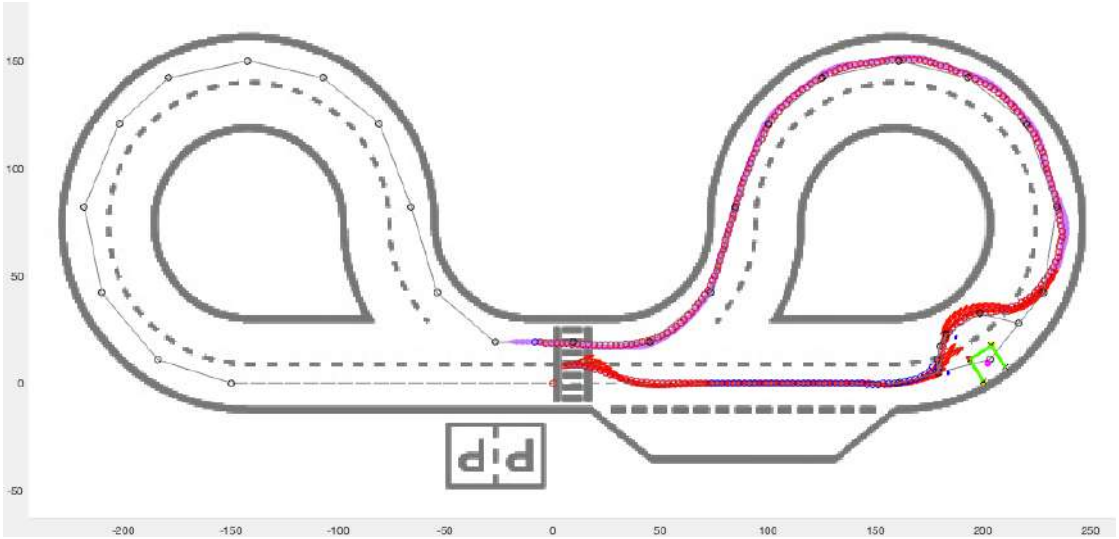


Figure 6.6: Robot over-passing the obstacle using the smooth path through dynamic arc fitting

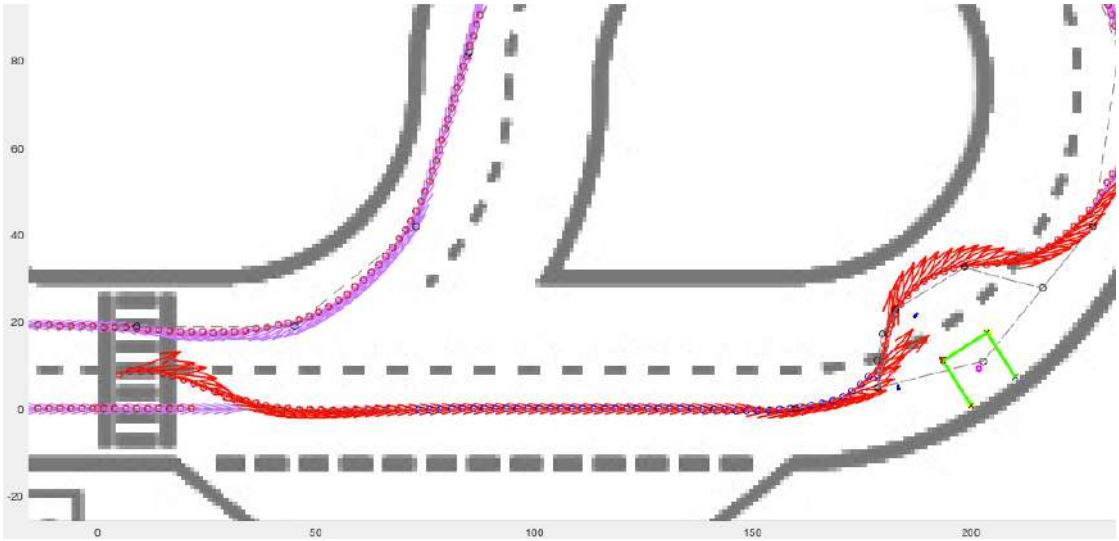


Figure 6.8: Obstacle identification and new waypoints produced

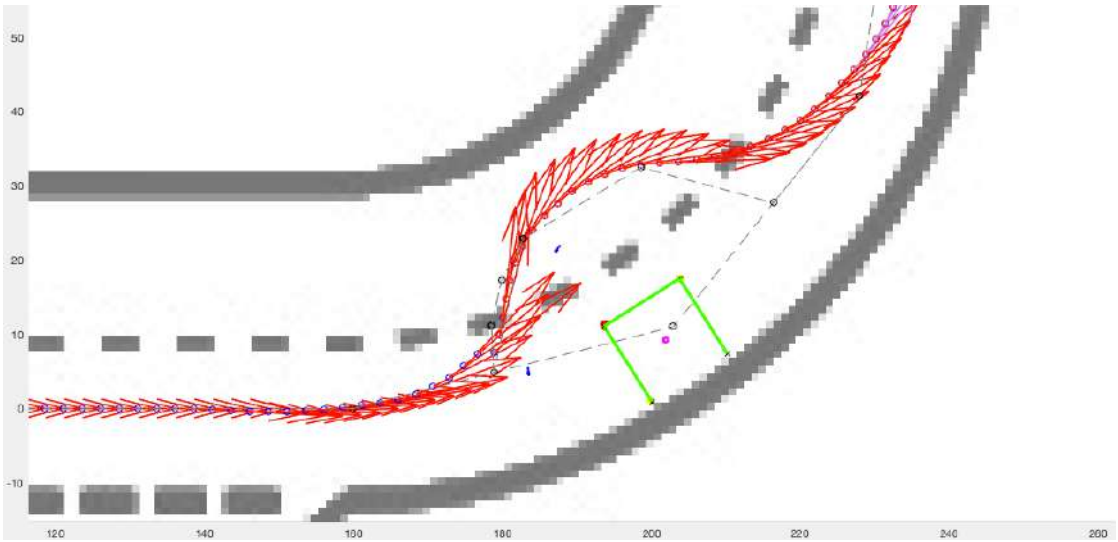


Figure 6.9: Obstacle identification and new waypoints produced

6.2 Parking Lot Identification

In this section, it is pointed out some relevant criteria for a qualitative evaluation of the parking's lot identification algorithm developed. It is also to presented results concerning those criteria. The results are obtained in a simulation procedure using the developed FNR - Autonomous Driving Competitions Simulation Environment, which allows validating the utility and versatility of the developed simulation environment. The criteria considered important for evaluating the performance of the developed algorithm are the following:

1. **consistency**: in identical scenarios they should produce identical results;
2. **speed**: ability to analyze the features and present results on time;
3. **accuracy**: coordinates and other measurements should be close to the actual references;

Concerning the criterion *consistency* of results, it expects that results produced or algorithm's output should be similar if the same characteristics of a given feature can be seen in a given set of image planes captured by the camera. The algorithm shall allow parking lot validation regardless of the perspective in which the image containing the parking symbol is acquired. This perspective is the one resulting from the robot's

position and heading on the track. However, it is plausible that there will be a moment, associated with a given robot's configuration in the work space, when the output of the algorithm will be different. In this case, it is important to assess the extent to which the system fails to correctly position the parking lot.

Regarding *speed*, the aim is to assess the extent to which the system supports a real-time processing evaluation and whether the system remains stable in the results obtained even if the robot is running. For this purpose, the ROS scheduler will be used to compare the processing time of the car park identification process. Finally, the third criterion associated with *precision* is also evaluated, comparing the results obtained in a simulation environment with the data obtained in the floor plan of the track. This comparison will provide a basis for comparison with the reliability of the obtained measurements.

With the testing of the parking lot identification algorithm in a simulation environment, it will be possible to validate the Inverse Perspective Mapping (IPM) process as a facilitating strategy for the extraction of relevant features to support the autonomous driving system, in this particular case, for an expeditious and robust process for the parking lot identification.

It will also be possible to verify the direct relationship, involving rapid algebraic manipulation, between the coordinates of the top-view image and the world coordinates.

Finally, the versatility of the algorithm proposed for the identification of other relevant features in the scenario of the autonomous competition will be explored, and the activation of such features will be based only on a classification process appropriate to each of the characteristics in question.

6.2.1 Consistency

Regarding the consistency of the algorithm, a special training set was prepared with 100 images captured from different angles, from close to 90^{o} in both directions (having the parking lot in both sides, left and right) to facing forward images.

The acquired images produced a total of 761 features. From those acquired features, only the *parking lot* feature, graphically perceived by the letter *P*, was tagged. All the remaining features were also kept in the training set but without a sensitive tagging or relevant classification. So such features are meant to be neglected by the algorithm.

During the preparation of the training set, some features were tagged as false positives and other as false negatives - in total 5 features were incorrectly tagged to test some robustness on the *K-Nearest Neighbor* and the capacity to handle the discrete evaluation

of assorted images.

The mosaic in the figure 6.10 illustrates how the algorithm is capable to detect the parking lot from different robot configurations. Nevertheless, it is possible to notice that, in some images captured, there was a situation that clear sight of the parking lot sign was not sufficient to get an identification. In one of the situations, a false match was also produced, which might reflect that errors present in the training may influence the classifications produced.

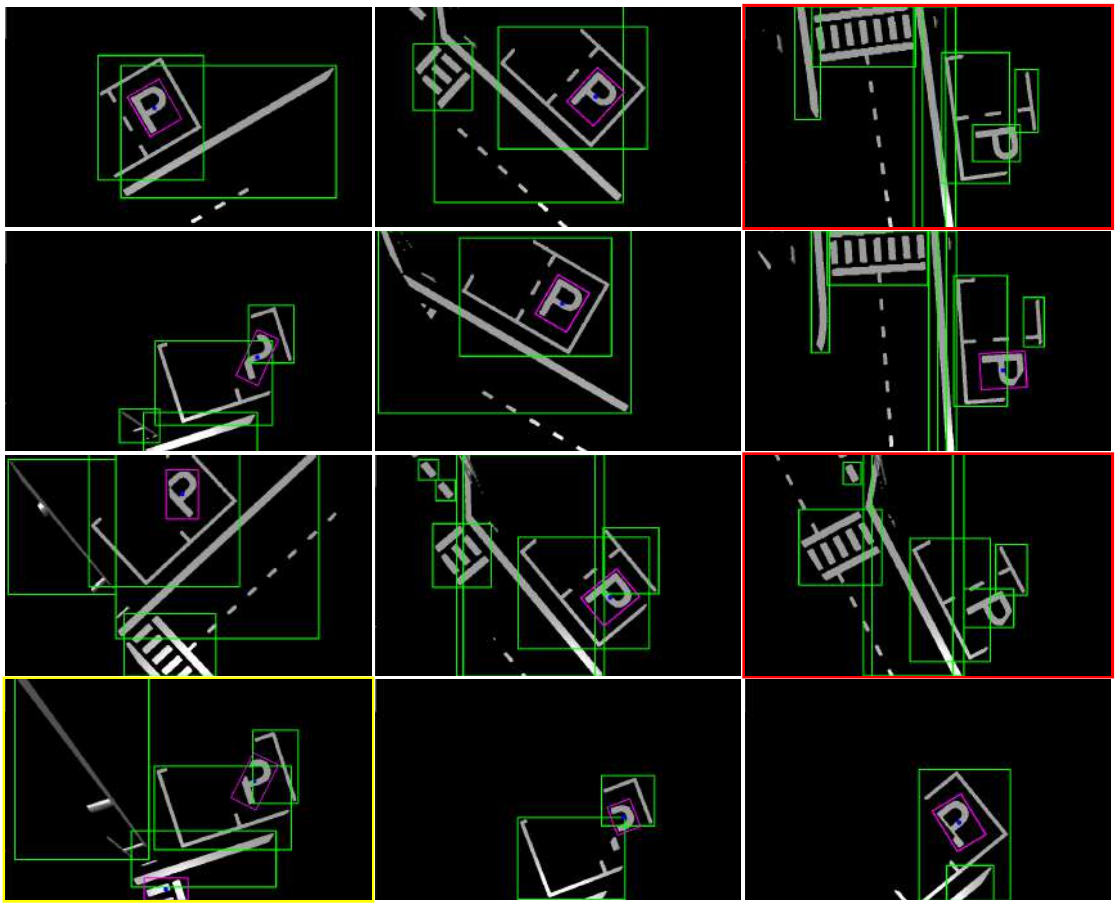


Figure 6.10: Parking lot detection with different robot configuration

6.2.2 Speed

Another criterion considered in the evaluation of the proposed solution and its applicability to real-time processing is the processing *speed*. The objective is to understand how

fast the algorithm processes the captured images and obtains an output from them.

The ROS node [`TOBOR_vision_parking_lot`] responsible for the parking lot identification, is bounded by a ROS-loop typically working at a *30 Hz* rate. This loop performs the IPM transformation, color segmentation, feature acquisition, and classification.

To measure the *speed* of the algorithm and a possible variation depending on the scene, the analysis took into consideration two possible situations: one performing the parking lot identification (centroid coordinate in world coordinate referenced to the robot and distance)in a stable scenario with the robot stopped; and another situation dynamic with the robot moving around the track.

The table 6.1 summarizes the results obtained in the experiment made. Theoretically, each loop should take 0,033 milliseconds, which corresponds to the *30 Hz* ROS loop rate defined. However, it is possible that some extra computing operations that persist no matter if visualization is active or not may implicate extra computing time directly proportional to the number of features encountered on each image.

Table 6.1: Time elapsed per each parking lot identification loop - in milliseconds

	Nr of features									
Time	1	2	3	4	5	6	7	8	9	10
Average	0,0339	0,0341	0,0356	0,0368	0,0399	0,0383	0,0384	0,0409	0,0448	0,0528
Std Dev	0,0035	0,0059	0,0060	0,0067	0,0070	0,0074	0,0076	0,0092	0,0088	0,0111
Min	0,0189	0,0187	0,0182	0,0187	0,0202	0,0211	0,0206	0,0215	0,0209	0,0228
Max	0.0603	0,0606	0,0817	0,0699	0,0721	0,0622	0,0757	0,0722	0,0678	0,0782

6.2.3 Precision

To evaluate precision, the robot is placed in a know configuration — position and heading relatively to the world — making sure the camera is capable to capture the parking lot sign, not necessarily on a straight forward position. This particular testing was made placing the robot on the track, in a plausible position to initiate parking procedures, and yet making sure the parking lot is captured even on significant angular positions.

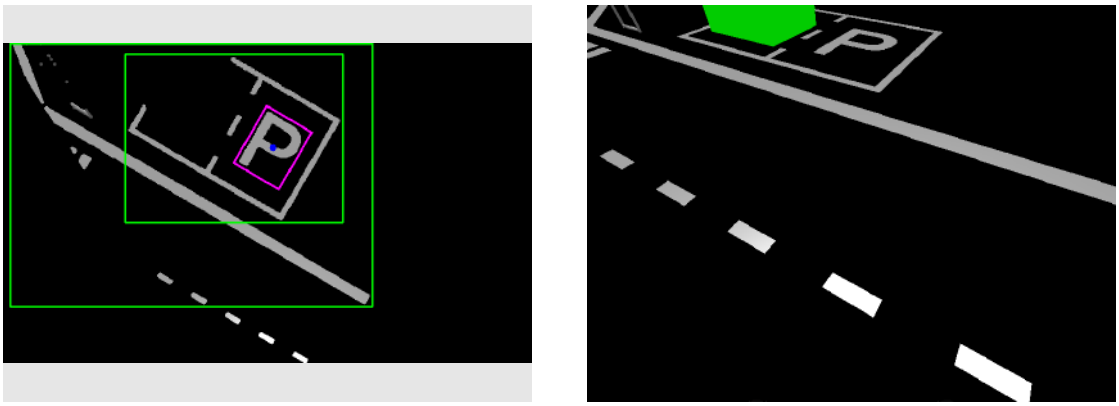


Figure 6.11: (left) Parking lot feature identified; (right) Image captured by the camera

The following PYTHON code snippet extracted from MORSE launch file illustrates the configuration given to the robot:

Listing 6.1: Robot configuration in FNR-ADC Simulation Environment

```
1 from math import pi
2 from morse.builder import *
3 # Placing the Robot right in front of the parking lot
4 my_robot.translate( (-2.4) , 0.7 , 0.0 ) # in Morse units [m]
5 my_robot.rotate( 0.0 , 0.0 , (-pi/2 + 30*pi/180) ) # heading [rads]
```

As illustrated by the code snippet 6.1, the robot was placed in the world having as coordinates $(-2400, 0700)$ [millimeters], with a heading related to world reference frame of $\frac{-\pi}{3}$.

The figure illustrates the setting prepared to evaluate the precision in the calculation of the distance between the robot and the parking lot. The figure combines the snapshot took from FNR-ADC Simulator Environment, over placed on the track's blueprint with a scale of 1:1.

Important remark: despite the configuration given to the robot through PYTHON's builder file (see code snippet 6.1), the precise location received by the FNR-ADC Simulator Environment subscribed topic "`/my_robot/pose`" is $(-2407, 722)$ [millimeters].

Despite the small difference between the robot's location settled by PYTHON builder and simulation output received in the subscribed topic, the evaluation relatively to the precision offered by the algorithm considers the robot's location received in simulation run-time. Therefore, calculating the distance between the robot's position $(-2407, 722)$ and the parking lot ground-truth center point coordinates $(-1350, -1220)$, it is possible

to calculate a distance of 2211mm.

The algorithm developed calculates distance between the camera's reference frame to a point on the image plane converted to a world coordinate in relation to the robot. The distance calculated does not take into account the offset between the camera's frame and robot's frame (correction that is made by the `tf transform` a ROS package responsible to maintain the relation between the frames composing the system - robot, camera, gyroscope, odometry, etc).

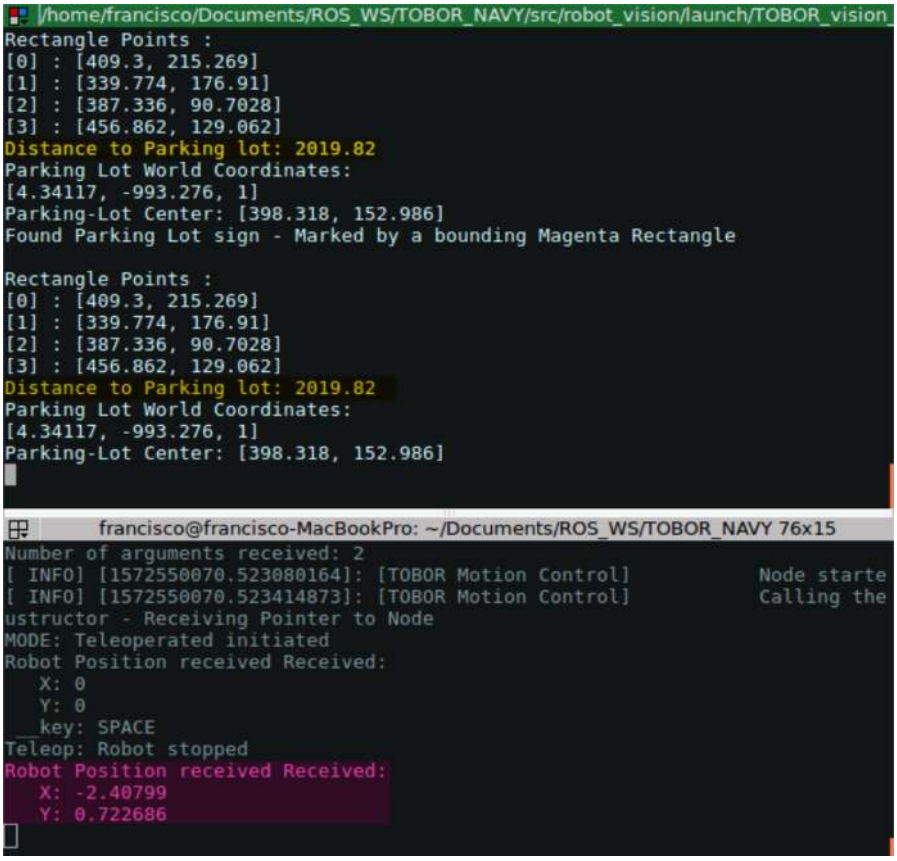


Figure 6.13: Distance measured

Therefore, to assess a correct evaluation about the precision of the system, to the distance computed by the algorithm – *2019 mm* – it shall be added the camera's offset in relation to the robot's frame $(227.895, 0)$ [millimeters] which results into a computed distance of *2246,895 mm*, a distance with an error of *35,874 mm* compared with the ground-truth.

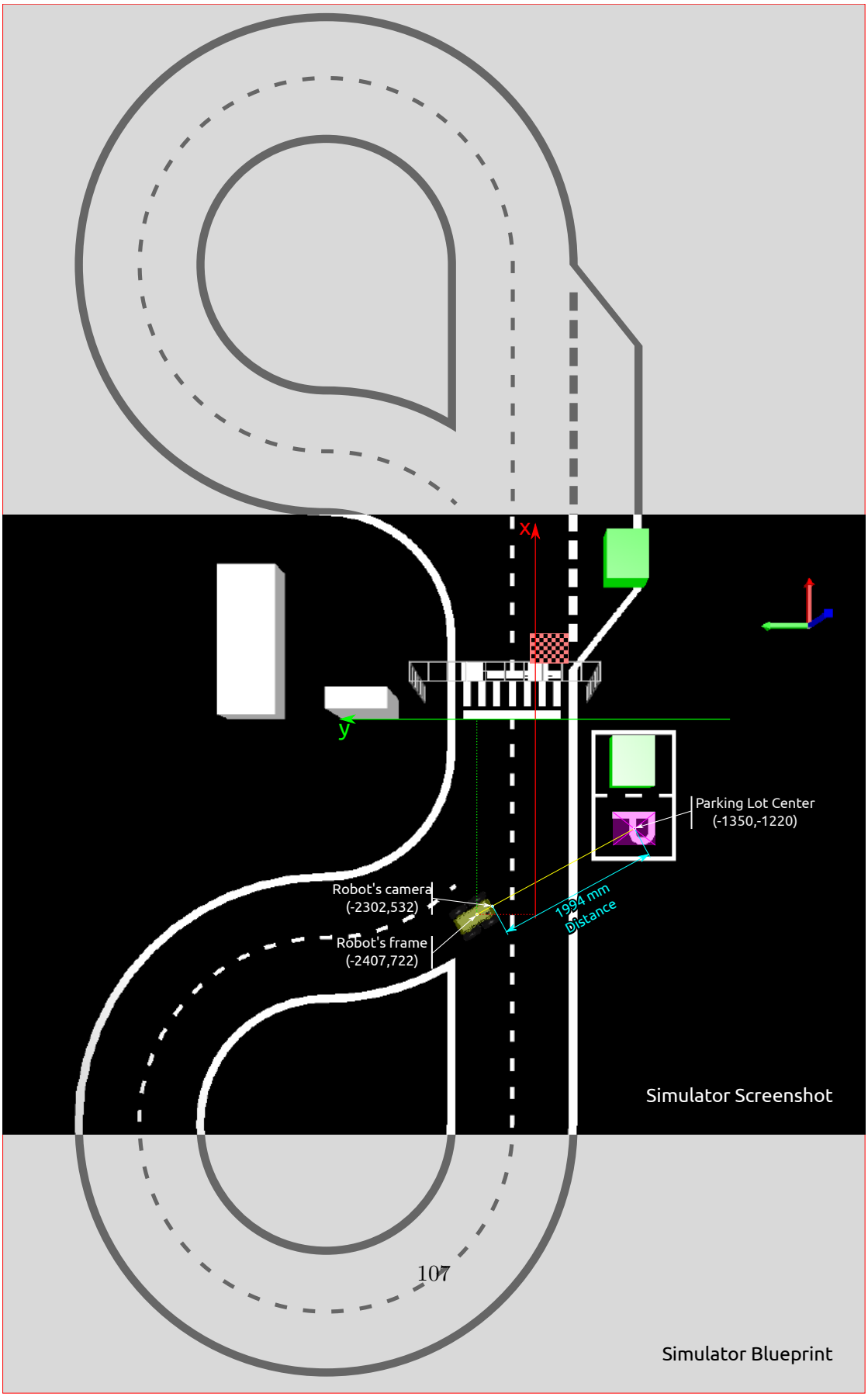


Figure 6.12: System measuring distance to parking lot

This page was intentionally left blank.

Chapter 7

Conclusions and Future Work

The work developed in the scope of master's degree on Autonomous Systems resulted in a scientific paper – *Teaching Robotics with a Simulator Environment Developed for the Autonomous Driving Competition* – published at **RiE2019**, the **10th International Conference on Robotics in Education** that took place in Vienna (Austria) [76].

It is inevitable to conclude that designing and implementing a consistent and robust Autonomous Driving System is a huge task, plenty of different challenges. It is also clear that multiple approaches can be addressed to the same problem, differing on simplicity, robustness, possibility of scaling or applicability scope. Either way, it is also clear that the level of complexity increases exponentially as long as the solution attempts to embrace wider range of different possibilities or situations present in any common autonomous driving scenario.

The **FNR-ADC Simulation Environment** proved to be a resourceful instrument to design, develop and test different systems and modules such as motion control or path planning, feature extraction, and decision making, position estimation or sensor fusion [23]. With handy high-level abstraction possibilities, it became easy to test each subsystem individually, allowing focused error debugging or performing analysis, without the risk of dispersing our attention to other modules that are not in the loop.

However, it was possible to encounter some awkward behaviors of the simulated robotic platforms, jeopardizing the simulation run-time. The observed anomalies diverge on the effects produced and occurs times-to-times, which turns difficult the task of debugging. The primary hypothesis as a potential cause for this odd behavior is the vector mesh used to design the track that might have some incompatibilities with Physics Engine or Game Engine.

Despite the punctual problems encountered in the FNR-ADC Simulation Environ-

ment, it was possible to test the developed algorithms. The Inverse Perspective Mapping proved to be useful for an undistorted feature analysis using computer vision. Despite the IPM developed is suited to FNR's Autonomous Driving Competition constraints, it may bring some fragility when applied to scenarios where the track/road has slopes (ascending or descending). This fact changes the transformation parameters between the image plane and the real-world and, under these circumstances, the IPM may not correct the perspective distortion. As future work, the Inverse Perspective Mapping shall take into account the slopes as a possible readjust of the extrinsic parameters accordingly the angled terrain. This tuning might be achieved with the introduction of concepts as Vanishing Point or even taking into account the typical parallelism of lines limiting the lane.

It was also possible to prove that Machine Learning is inevitable when developing autonomous driving systems. Despite the simplicity of the K-Nearest Neighbors algorithm, it proved to be a satisfactory option and resourceful technique when simple and discrete classification is required. The use of Machine Learning techniques is an efficient solution when it is necessary to extract complex features from computer vision systems. As future work, a possible upgrade of feature extraction using Support Vector Machine or Decision Trees, both mature algorithms with the potential to deal with discrete data with optimal performance when dealing with many classification groups.

Finally, the dynamic arc fitting proved to be a possible approach to model the local motion of a 4-wheel skid-steer platform, as its geometric principle fairly represents the primitive maneuvers of a platform of this kind. The PID controller implemented, also included a Twiddle algorithm to allow a fine auto-tuning of the Proportional, Integrative and Derivative gains. However, the results were not satisfactory, and future work must be done to ensure a more stable motion behavior. Nevertheless, the PID control only acts over linear velocity and despite the effects of the Physics Engine present in the FNR-ADC Simulation Environment, the arc fitting algorithm proved to be capable to perform a full round at a stable velocity of 2-meters per second without PID control on the change of heading. As future work, the waypoint definition shall include more detailed information about the local maximum and optimal speed which should constitute refined information to ensure a dynamic and fitted robot's motion.

Due to computational limitations, it was not possible to test the Obstacle Avoidance algorithm using the FNR-ADC Simulation Environment. The algorithm developed makes use of the LIDAR sensor to estimate the obstacle position on the track. Enabling the emulated LIDAR represented a high computational cost which turned impossible any sort of simulation due to the lag created. Therefore, the algorithm was tested in

Matlab, with both, track and sensors, coded simulated.

The approach to characterize the obstacle, obeying to the primitive maneuvers of the robot and the local motion planning created, proved to be efficient under the competition constraints. However, the approach deeply relays on an accurate definition of the obstacle and the correct positioning of the robot. However, due to the possibility of successive readjustments of the evasive waypoints created to avoid the collision, it turns the solution more flexible and less sensitive to miscalculation regarding the obstacle position. As future work, despite the arc trajectory fits the characteristics non-holonomic of a 4WSS robot, describing the arc yet represents a skidding of the platform and therefore it is not guaranteed the trajectory described is a perfect arc.

These are sub-systems integrating a more vast system which is the Autonomous Driving. As future work is the development of complementing modules in order to set a minimum autonomous driving system capable to perform all the tasks composing the FNR-Autonomous Driving Challenge.

This page was intentionally left blank.

Bibliography

[1] José Nuno Carvalho. *José Nuno da Silva Carvalho ROTA ' 2008 : um robô para condução autónoma*. PhD thesis, Universidade de Aveiro, 2008.

[2] Valter Costa. Projeto e Casos de Estudo em Robótica Educativa envolvendo Visão Tempo Real. Technical report, Faculdade de Engenharia da Universidade do Porto - 2015, 2015.

[3] Yu Huang. Inverse Perspective Mapping (Bird-eye View). `https://sites.google.com/site/yorkyuhuang/home/research/computer-vision-augmented-reality/ipm`.

[4] David Rose. Vehicle MPC Controller. `https://medium.com/@david010/vehicle-mpc-controller-33ae813cf3be`, 2017. Accessed: 18/11/2019.

[5] Sebastian Thrun. Udacity Autonomous Driving Course. `https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013`, 2019.

[6] DARPA. DARPA Grand Challenge Gallery. `https://archive.darpa.mil/grandchallenge04/contact.htm`, 2004. Accessed: 2019-11-23.

[7] AUDI. AUDI's Autonomous Driving Cup. `https://www.audi-autonomous-driving-cup.com`, 2018. Accessed: 2019-11-23.

[8] CARLA. CARLA Autonomous Driving Challenge. `https://carlachallenge.org`, 2018. Accessed: 2019-11-23.

[9] Koichi Osuka, Robin Murphy, and Alan C. Schultz. USAR competitions for physically situated robots. *IEEE Robotics and Automation Magazine*, 9(3):26–33, 2002.

[10] Sociedade Portuguesa de Robótica. Rules for autonomous driving. Technical report, 2016.

[11] BSD. Explaining BSD. `https://www.freebsd.org/doc/en_US.ISO8859-1/articles/explaining-bsd/article.html`, 2019. Accessed: 2019-11-23.

[12] Bruno Batista Neto. *Condução Autónoma – Desenvolvimento de um robô para navegação num ambiente rodoviário à escala.* PhD thesis, Instituto Politécnico de Leiria, 2014.

[13] Vitor Santos. ATLASCAR: A sample of the quests and concerns for autonomous cars. *Lecture Notes in Electrical Engineering - Springer Nature Switzerland AG 2020* -, 495:355–375, 2019.

[14] R Cancela, M Neta, M Oliveira, and V Santos. ATLAS III – Um Robô com Visão Orientado para Provas em Condução Autónoma. Technical Report April, 2005.

[15] Miguel Riem Oliveira, Jorge Manuel, Soares Almeida, and Vitor M F Santos. Modular scalable architecture for the navigation of the ATLAS autonomous robots. In *Conference: 9th Conference on Autonomous Robot Systems and Competitions, Castelo Branco, Portugal, May the 7th., At Castelo Branco, Portugal*, number October 2014, 2009.

[16] Luís Almeida, Paulo Pedreiras, and José Alberto G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, 2002.

[17] José Correia. Unidade de Perceção Visual e de profundidade para o ATLASCAR2. Master's thesis, Universidade de Aveiro, 2017.

[18] Armando Sousa, Catarina Santiago, Paulo Malheiros, Paulo Costa, and António Paulo Moreira. Using Barcodes for Robotic Landmarks. *New Trends in Artificial Intelligence. 14th Portuguese Conference on Artificial Intelligence*, pages 300–311, 2009.

[19] Valter Costa, Peter Cebola, Armando Sousa, and Ana Reis. Design hints for efficient robotic vision - Lessons learned from a robotic platform. *Lecture Notes in Computational Vision and Biomechanics*, 27(June 2018):515–524, 2018.

[20] Valter Costa, Peter Cebola, Armando Sousa, and Ana Reis. Design of an Embedded Multi-Camera Vision System—A Case Study in Mobile Robotics. *Robotics*, 7(1):12, 2018.

[21] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Ferreira Reis Jesus, Rodrigo Ferreira Berriel, Thiago Meireles Paixão, Filipe Mutz, Thiago Oliveira-Santos, and Alberto Ferreira De Souza. Self-Driving Cars: A Survey. *ArXiv - Expert Systems with Applications*, abs/1901.04407, 2019.

[22] David Silver. How Self-Driving Cars Work. `https://medium.com/udacity/how-self-driving-cars-work-f77c49dca47e`, 2017. Accessed: 10/09/2019.

[23] David Fernandes, Francisco Pinheiro, André Dias, Alfredo Martins, J. M. Almeida, and Eduardo Pereira Silva. Teaching Robotics with a Simulator Environment Developed for the Autonomous Driving Competition. *Robotics in Education*, 1023, 2020.

[24] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51, May 2011.

[25] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: MORSE. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 46–51, 2011.

[26] Farzan M. Noori, David Portugal, Rui P. Rocha, and Micael S. Couceiro. On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo? *SSRR 2017 - 15th IEEE International Symposium on Safety, Security and Rescue Robotics, Conference*, pages 19–24, 2017.

[27] David Schreiber, Bram Alefs, and Markus Clabian. Single camera lane detection and tracking. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2005:1114–1119, 2005.

[28] Joel C. McCall and Mohan M. Trivedi. Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, 2006.

[29] Gi Jung Ho, Suk Kim Dong, Joo Yoon Pal, and Kim Jaihie. Parking slot markings recognition for automatic parking assist system. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 106–113, 2006.

[30] Yu-Chih Liu, Kai-Ying Lin, and Yong-Sheng Chen. Bird's-Eye View Vision System for Vehicle Surrounding Monitoring. *Springer Robot Vision, Proceedings*, pages 207–218, 2008.

[31] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[32] Z Zhengyou. *IEEE transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[33] Massimo Bertozzi and Alberto Broggi. GOLD: A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81, 1998.

[34] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, and Constantine D. Spyropoulos. An Experimental Comparison of Naive Bayesian Anti-Spam Filtering. pages 24–28, 2000.

[35] Savan Patel. Chapter 4: K Nearest Neighbors Classifier. `https://medium.com/machine-learning-101/k-nearest-neighbors-classifier-1c1ff404d265`, 2017.

[36] Prof Patrick Henry Winston. MIT - Artificial Intelligence [course 6.034], 2019.

[37] Yong K. Hwang and Narendra Ahuja. A Potential Field Approach to Path Planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[38] A. A. Kassim and B. V. K. V. Kumar. A neural network architecture for path planning. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 2, pages 787–792 vol.2, June 1992.

[39] John Canny. A New Algebraic Method for Robot Motion Planning and Real GeolDetry s .. s ..,. *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 39–48, 1987.

[40] Costas S. Tzafestas and Spyros G. Tzafestas. Recent algorithms for fuzzy and neurofuzzy path planning and navigation of autonomous mobile robots. *European Control Conference, ECC 1999 - Conference Proceedings*, pages 4736–4743, 2015.

[41] David Šišlák, Přemysl Volf, and Michal Pěchouček. Accelerated A* trajectory planning: Grid-based path planning comparison. *19th International Conference on Automated Planning & Scheduling (ICAPS)*, (6840770038):74–81, 2009.

[42] E Kosenko, D Beloglazov, and V Finaev. Chapter Three - Vehicles Fuzzy Control Under the Conditions of Uncertainty. In Viacheslav Pshikhopov, editor, *Path Planning for Vehicles Operating in Uncertain 2D Environments*, pages 97–136. Butterworth-Heinemann, 2017.

[43] Lydia E. Kavraki, Petr Švestka, Jean Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[44] Guan Zheng Tan, Huan He, and Aaron Sloman. Ant colony system algorithm for real-time globally optimal path planning of mobile robots. *Zidonghua Xuebao/Acta Automatica Sinica*, 33(3):279–285, 2007.

[45] Steven m. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Iowa State University*, 1998.

[46] Xuan Vinh Ha, Cheolkeun Ha, and Jewon Lee. Novel hybrid optimization algorithm using PSO and MADS for the trajectory estimation of a four track wheel skid-steered mobile robot. *Advanced Robotics*, 27(18):1421–1437, 2013.

[47] Tamio Arai and Jun Ota. Recent Trends in Mobile Robots. In *World Scientific Series in Robotics and Intelligent Systems*, chapter MOTION PLA, pages 57–73. World Scientific Publishing Co. Pte. Ltd., Ohio, 1994.

[48] Santos Coelho. Bacteria Colony Approaches With Variable Velocity. *Proceedings of COBEM 2005 - 18th International Congress of Mechanical Engineering - November 6-11, 2005, Ouro Preto, MG*, 2(2002):297–304, 2006.

[49] S. Meeran and A. Share. Optimum path planning using convex hull and local search heuristic algorithms. *Mechatronics*, 7(8):737–756, dec 1997.

[50] Deepak BBVL, Jha Alok Kumar, and Parhi Dayal R. Path Planning of an Autonomous Mobile Robot Using Artificial Immune System \n. 96(11):11–16, 2011.

[51] Jonathan M. Weaver and Stephen J. Derby. A divide-and-conquer method of path planning for cooperating robots with string tightening. *Proceedings - 4th Annual Conference on Intelligent Robotic Systems for Space Exploration, IRSSE 1992*, pages 30–40, 1992.

[52] Hoc Thai Nguyen, Hai Xuan Le, and Viet Nam. Path planning and Obstacle avoidance approaches for Mobile robot. *International Journal of Computer Science Issues*, 13(4):1–10, 2016.

[53] Mohd Nayab Zafar and J. C. Mohanta. Methodology for Path Planning and Optimization of Mobile Robots: A Review. *Procedia Computer Science*, 133:141–152, 2018.

[54] Kamil F Tang SH and Khaksar W Zulkifli N. A Review on Motion Planning and Obstacle Avoidance Approaches in Dynamic Environments. *Advances in Robotics & Automation*, 04(02), 2015.

[55] Peter Sanders and Dominik Schultes. Engineering fast route planning algorithms. *Proceedings of the 6th international conference on Experimental algorithms*, pages 23–36, 2007.

[56] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[57] Choset Howie, Kevin M. Lynch, Hutchinson Seth, Kantor George A., Burgard Wolfram, Kavraki Lydia E., and Sebastian Thrun. Configuration Space. In Brandfor Book, editor, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, chapter 3. Configu, pages 39–69. MIT Press, Massachusetts, 2005.

[58] Roland SIEGWART and NOURBAKHSH Illah R. Introduction to Autonomous Mobile Robots. In Ronald C. Arkin, editor, *Intelligent Robotics and Autonomous Agents*, chapter 6- Plannin, pages 257–298. The MIT Press, Massachusetts, 1st edition, 2004.

[59] Andrew B Walker, B Eng, and B App Sc. Hard Real-Time Motion Planning for Autonomous Vehicles the degree of Doctor of Philosophy. (November):16–27, 2011.

[60] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney.

Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[61] Oussama Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Autonomous Robot Vehicles*, volume 5, pages 396–404, New York, 1990. Springer, New York, NY.

[62] David Madas, Mohsen Nosratinia, Mansour Keshavarz, Peter Sundstrom, Rolland Philippsen, Andreas Eidehall, and Karl Magnus Dahlen. On path planning methods for automotive collision avoidance. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):931–937, 2013.

[63] Vivien Delsart, Thierry Fraichard, and Luis Martinez. Real-time trajectory generation for car-like vehicles navigating dynamic environments. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3401–3406, 2009.

[64] Yanfeng Cong, Oliver Sawodny, Hong Chen, Jan Zimmermann, and Alexander Lutz. Motion planning for an autonomous vehicle driving on motorways by using flatness properties. *Proceedings of the IEEE International Conference on Control Applications*, pages 908–913, 2010.

[65] Liang Ma, Jing Yang, and Meng Zhang. A two-level path planning method for on-road autonomous driving. *Proceedings - 2012 International Conference on Intelligent Systems Design and Engineering Applications, ISDEA 2012*, (90920301):661–664, 2012.

[66] David González, Joshue Pérez, Ray Lattarulo, Vicente Milanés, and Fawzi Nashashibi. Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. *2014 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2014*, pages 1430–1435, 2014.

[67] Miao Wang, Tinosch Ganjineh, and Raúl Rojas. Action annotated trajectory generation for autonomous maneuvers on structured road networks. *ICARA 2011 - Proceedings of the 5th International Conference on Automation, Robotics and Applications*, pages 67–72, 2011.

[68] Tianyu Gu and John M. Dolan. Toward human-like motion planning in urban environments. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 350–355, 2014.

[69] Hao Sun, Weiwen Deng, Sumin Zhang, Shanshan Wang, and Yutan Zhang. Trajectory planning for vehicle autonomous driving with uncertainties. *ICCSS 2014 - Proceedings: 2014 International Conference on Informative and Cybernetics for Computational Social Systems*, pages 34–38, 2014.

[70] Michael R. Benjamin, John J. Leonard, Henrik Schmidt, and Paul M. Newman. Nested Autonomy for Unmanned Marine Vehicles with MOOS-IvP. *Journal of Field Robotics*, 29(4):554–575, 2012.

[71] Henri P. Gavin. The Levenburg-Marqurdt Algorithm For Nonlinear Least Squares Curve-Fitting Problems. pages 1–19, 2019.

[72] Camilo Ordonez, Nikhil Gupta, Wei Yu, Oscar Chuy, and Emmanuel G. Collins. Modeling of Skid-Steered Wheeled Robotic Vehicles on Sloped Terrains. *Volume 3: Renewable Energy Systems; Robotics; Robust Control; Single Track Vehicle Dynamics and Control; Stochastic Models, Control and Algorithms in Robotics; Structure Dynamics and Smart Structures;*, pages 91–99, 2012.

[73] Peter Lepej, Johannes Maurer, Suzana Uran, and Gerald Steinbauer. Dynamic Arc Fitting Path Follower for Skid-Steered Mobile Robots. *International Journal of Advanced Robotic Systems*, 12(10):1–13, 2015.

[74] Fady Ibrahim, Ahmed Ali, Ahmed Elbab, and Tetsuya Ogata. Path following algorithm for skid-steering mobile robot based on adaptive discontinuous posture control. *Advanced Robotics*, 33:1–15, 04 2019.

[75] R Craig Coulter. Implementation of the Pure Pursuit Path Tracking Algorithm. *Communication*, (January), 1992.

[76] Munir Merdan, Wilfried Lepuschitz, Gottfried Koppensteiner, Richard Balogh, and David Obdržálek, editors. *Robotics in Education*, volume 81. Springer Nature Switzerland, Vienna, 1st editio edition, 2016.

Appendix A

FNR-ADC participating Teams

List of participating teams of latest FNR-ADC editions, an corresponding leaders contacted in order to compile the design and solutions implemented with relevant outcome in the scope of the autonomous driving competition:

- **N3E GT Team - Polytechnic Institute of Leiria**
  - Eng Hugo Costelha — [hugo.costelha@ipleiria.pt];
  - Eng Carlos Couceiro Neves — [carlos.neves@ipleiria.pt];
- **ROTA Team - University of Aveiro**
  - Eng Artur Pereira — [artur@ua.pt];
  - Eng José Luís Azevedo — [jla@ua.pt];
  - Eng Ricardo Dias — [ricardodias@ua.pt];
- **CONDE Team - Faculty of Engineering, University of Porto**
  - Eng Armando Sousa — [asousa@fe.up.pt].