Autopilot Simulator Prototype for Autonomous Driving based on SimTwo

Adérito Daniel Mendes Abreu - a30968

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Professor PhD José Luís Sousa Magalhães Lima

Professor PhD Paulo José Cerqueira Gomes da Costa

Bragança

2019 - 2020

Autopilot Simulator Prototype for Autonomous Driving based on SimTwo

Adérito Daniel Mendes Abreu - a30968

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Professor PhD José Luís Sousa Magalhães Lima

Professor PhD Paulo José Cerqueira Gomes da Costa

Bragança

2019 - 2020

Dedication

To my dear godfather, thank you for encouraging me to follow this academic journey...

-This page intentionally left blank-

Acknowledgement

The work here presented, is only possible thanks to the collaboration and support of some people, to who I can not miss to give due recognition and appreciation.

To my advisors, Professor José Luís Sousa Magalhães Lima and Professor Paulo José Cerqueira Gomes da Costa, for all the time spent in favor of the realization of this project, for the orientations and critical analyzes, and for the motivation and confidence that they have always been transmitted to me over the past few months.

To my university, School of Technology and Management of Bragança, for all support and academic facilities.

To my longtime friends, who always gave me unconditional support at all times, for the true friendship and camaraderie shown over the years. Good luck to all.

To my family, for all the support provided during this journey, in particular I thank my parents for all the patience, friendship, affection, help and unity shown during my entire life.

For everything, I will be forever thankful to you all.

-This page intentionally left blank-

Abstract

The main objective of this work was to develop a control system for an autonomous vehicle that provides autonomous driving. For this, a simulation software, named "SimTwo" was used, where the actuation and sensing model was developed.

At the end of the work, a control and 3D visualization system was obtained for an autonomous vehicle capable of driving on a road, avoiding obstacles, alerting in case of danger, among others. The work was developed in a simulation environment and includes a 3D model of a road, with several real scenarios, where the vehicle moves. There are objects on the circuit that can obstruct the passage of the car, creating situations of imminent danger. This system alerts the driver in the event of danger and reacts by deflecting or stopping. This control system uses image sensors and LiDAR (Light Detection And Ranging) as inputs data sources.

Keywords: Autonomous Driving, 3D Simulation, SimTwo, Sensing, Artificial Vision, LiDAR.

-This page intentionally left blank-

Resumo

O principal objetivo deste trabalho foi desenvolver um sistema de controlo de um veículo autónomo que o dote de condução autónoma. Para tal, foi utilizado um software de simulação, SimTwo, onde o modelo de atuação e sensorização foi desenvolvido.

No final do trabalho, obteve-se um sistema de controlo e visualização 3D de um veículo autónomo capaz de conduzir numa estrada, desviar de obstáculos, alertar no caso de perigo, entre outros. O trabalho foi desenvolvido num ambiente de simulação e contempla um modelo 3D de uma estrada, com vários cenários reais, onde o veículo se desloca. Existem objetos nas bermas que podem obstruir a passagem do carro, criando situações de perigo eminente. Este alerta no caso de perigo e reage, desviando ou parando. Este sistema de controlo utiliza sensores de imagem e LiDAR (da sigla inglesa "Light Detection And Ranging"), como fontes de informação.

Palavras-chave: Condução Autónoma, Simulação 3D, SimTwo, Sensorização, Visão Artificial, LiDAR.

-This page intentionally left blank-

Contents

Acknowledgement **vii**

Abstract **ix**

Resumo **xi**

1 Introduction **1**

1.1 Context 1

1.2 Objectives and Motivation 2

1.3 Framework and Content 2

2 State-Of-The-Art **5**

2.1 Levels of Vehicle Autonomy 5

2.2 Sensorial Perception 7

2.3 Sensorial Fusion for Autonomous Driving 10

2.4 Automated Driving Solutions 12

2.4.1 Partnerships 13

2.5 Autonomous Driving Simulators 14

2.5.1 Nvidia Drive SIM 14

2.5.2 Carla 15

2.5.3 Waymo's Carcraft 15

2.5.4 LGSVL 16

2.5.5 Robotic Simulators 16

2.5.6 Robotic Simulators Comparison 17
2.5.7 Simulation Lack and Future 19

3 Sytem Architecture **21**
3.1 SimTwo Overview 21
3.2 Robot Model 23
3.2.1 Robot Body 24
3.2.2 Sensing and Perception 25
3.3 Track Modeling 27

4 Development **29**
4.1 Control 29
4.1.1 Main Program 30
4.1.2 PID Motors Control 32
4.1.3 Image Sense and Recognition 32
4.1.4 Obstacle Outline Algorithm 35
4.1.5 Red Stop Line 36
4.2 Traffic Lights 38
4.3 Scene Construction 39
4.4 Debug Interface 40

5 Results and Discussions **43**
5.1 Line Follower Algorithm Results 43
5.2 Obstacle Overtake Algorithm Results 44
5.3 Variable Values Adjust 46
5.4 Other Analyses 47
5.5 Unsolved 47

6 Conclusions and Future Work **49**
6.1 Future Work 50

List of Tables

2.1 Advantages and Disadvantages of each sensor 11

2.2 Characteristics of Commercials and Open-Source Robotic Simulation Softwares (Adapter from [39]). 18

3.1 RGB camera proprieties. 26

List of Figures

2.1 Levels of driving automation [11]. 7

2.2 Remote sensing technologies taxonomy (adapted from [2]). 8

2.3 Automotive sensors: (a) LiDAR [13]; (b) example of LiDAR point cloud [14]; (c) Sonar in a parking assistance maneuver [15] and (d) Radar application [16]. 9

2.4 Sensorial perception: (a) 3D image recreation by camera images[16], and (b) object detection with camera image. 10

2.5 Vehicle Sensor Fusion [18]. 12

2.6 Autonomous Driving Partnerships: Red, blue and green respectively represents vehicles manufactures, traditional automotive suppliers and tech companies (Adapter from [22]). 13

3.1 SimTwo software: Application windows clockwise from top left corner: spreadsheet, cameras, configuration, world view, scene editor, code editor, and chart (adapter from [29] and [42]). 22

3.2 SimTwo Scene editor structure (adapter from [29]). 23

3.3 SimTwo robot configuration (adapter from [43]). 24

3.4 Four wheels robot modeling. 24

3.5 Robot RGB camera view. 25

3.6 Robot Lidar Beam. 26

3.7 Simtwo track layout. 27

4.1 Finite-State Machine. 31

4.2 PID close loop control. 32
4.3 Camera acquired image. 33
4.4 Line Detector 34
4.5 Obstacle overtake 36
4.6 LiDAR beams read values 36
4.7 Red Stop Line 36
4.8 Visual warning about the present or not of a track obstacle. 38
4.9 Traffic Lights: (a) Red Light where the robot must stop and (b) Green Light where the robot can move forward. 38
4.10 Developer GUI as sheet style for user scenario interpretation 40

5.1 No bounder line 44
5.2 Crossing options 44
5.3 Overtake obstacle sequence 45
5.4 Velocity analyses when the robot is overtaking, where y axis represents the velocity and x axis the time in seconds. 45
5.5 Weight of each strategic line of camera analyses 46
5.6 PID Values calibration track 46
5.7 Robot X axis position adjust 47

Listings

3.1 LiDAR parameters definition 26
4.1 State machine described in pascal code. 30
4.2 "If not detected line" Pascal Code. 34
4.3 "Velocity as a function of direction" Pascal Code 35
4.4 "Red Stop Line Procedure" Pascal Code. 37
4.5 Scene XML Code 39

Chapter 1

Introduction

In this first chapter, the general overview is described and a succinct context of the theme underlying the present dissertation project performed. Subsequently, the objectives are defined, followed by a brief description of the framework presentation.

1.1 Context

Nowadays, the ability to precisely use computers to simulate a car driving environment in real-time is central in an ample range of fields, particularly in robotics and automotive industries [1]. More specifically and standing out, many applications rely on simulation analyses, from which driving assistance systems [1], autonomous driving [1], traffic control [1], robot navigation [2], games involving virtual reality [3], can be highlighted. There is a common need for accurate simulate of position, dimension or movement of real world scenarios. In the last decades, road traffic crashes are a leading cause of death in world for people aged 1-54. Throughout the world, roads are shared by cars, buses, trucks, motorcycles, mopeds, pedestrians, animals, taxis, and other travelers. Travel made possible by motor vehicles supports economic and social development in many countries. Yet each year, vehicles are involved in crashes that are responsible for millions of deaths and injuries [4]. According to Ansys research's [5], it's estimated that 12 billion kilometers worth of testing is needed to prove that self-driving cars are safer than humans,

that means almost 1 century of driving for a car and a colossal monetary investment. As a way to solve for this problem researchers need to take advantage of physics-based simulations and virtual reality simulations for autonomous vehicle testing [6]. Simulation allows for checking the behavior of autonomous vehicles in a huge number of scenarios, environments, system configurations and driver characteristics. It does not make real world tests obsolete, but it can help focusing on the necessary proving ground tests to verify the simulation results and for certification measurements. Simulation is a crucial step on the development and testing of autonomous driving software, without simulation is impossible to manage all tests and verification operations [6].

1.2 Objectives and Motivation

This thesis is incorporated in a autonomous driving simulation topic. Started with a software tool to developer an autonomous vehicle capable of self-driving and represents all the environment. At the end of the job, was expected a fully 3D environment where a self-driving car can safely drive (e.g. line follower, avoid obstacles, react in dangerous situations, etc...). To accomplish this, a software capable of world representation and accurate simulation must be elected and served as basis and started point for the development of this project, since the need of a realistic tool capable of real world behaviour representation. The central point and main motivation to address this theme of work was the rapid evolution of the automotive sector connected to autonomous driving industry, which is already a reality today and will be even more in the near future. Coupled with this topic, the ability to simulate all this autonomous world has its advantages and play an essential point to make this self-driving activity more and even more real.

1.3 Framework and Content

This dissertation is composed of 6 chapters, each one centered around a section of the developed work. The first chapter is an introduction to the thematic to be tackled in this

project. A brief background for autonomous driving current status and a initiation about why simulation is a crucial point on autonomous driving world. As an introduction chapter this division also contextualize and explains the motivation and objectives proposed for this dissertation.

In Chapter 2, the state-of-the-art for autonomous driving technologies and simulators is presented, as several sensors are explored to give an insight on the current panorama, understand common characteristics and verify if the market offer fulfils the requisites for level 4 and 5 automation.

In chapter 3, the system architecture is exposed, based on the software chosen to implement this project, also all the robot, obstacles and the track kinematic are detailed in this chapter.

Subsequently, chapter 4 includes and explain the development work created, going in detail in all algorithm developer work.

Before the last chapter, chapter 5 takes place to describe all the results and discussions obtained throughout the work.

To close chapter 6 where the conclusions of this project are drawn and some propositions for future work are disclosed.

-This page intentionally left blank-

Chapter 2

State-Of-The-Art

In this chapter, a first approach is performed to explain the current levels of existing autonomous driving levels, discriminating each of these levels. Thus, a brief explanation of the main sensors that enable autonomous driving is carried out and sensorial fusion is also explained. Afterwards, a brief market analysis is realized, culminating in the identification of the main manufacturers for the autonomous automotive industry. Later, a comparative evaluation is executed regarding autonomous diving simulation softwares, and the fundamental conclusions are stated on whether the simulators are appropriate or not for the context.

Since the market is constantly developing and the products are more and more updated, it is important to take into account that this research was carried out until March 2020, which means that some data may have already been updated.

2.1 Levels of Vehicle Autonomy

In 2014, the Society of Automotive Engineers (SAE) defines the standard J3016 [7] that classifies autonomous driving in six levels based in the amount of driver intervention and necessary alert [7]. Starting from completely human operated vehicles at Level 0 to fully autonomous driving car at level 5, is represented in Figure 2.1.

Most vehicles nowadays are level 0, they have to be controlled manually, at this level

the warning alerts of the vehicle system can momentarily intervene but the human is responsible for all aspects of driving [8]. At the lowest level of automation, level 1, most functions continue to be controlled by the driver, responsible for steering, acceleration and deceleration [9]. At level 2, vehicles are able to assist with functions like steering, acceleration, braking, and maintaining speed, although drivers still need to have both hands on the wheel and be ready to take control if necessary. [7]. Between level 2 and level 3, the vehicle takes the leading role in the driving process. At this level 3, the driver can sit back and let the car do all the driving. Also referred to as 'eyes-off' vehicles, drivers are able to focus their attention on other activities like using a mobile phone, for example.. Whatsoever, the driver's still responsibility to be alert and intervene at any time if the system fails. It is in the transition to level 4 that the challenge of autonomous driving begins to take real dimensions. At Level 4, vehicles are capable of steering, accelerating, and braking on their own. They're also able to monitor road conditions and respond to obstacles, determining when to turn and when to change lanes. [7]. Finally, level 5 being the level that presents the autonomous driving in a total format, vehicles are able to steer, accelerate, brake and monitor road conditions like traffic jams. Essentially, Level 5 automation enables the driver to sit back and relax without having to pay any attention to the car's functions whatsoever [8]. Is also expecting that in level 5 vehicles don't even need to transport passengers. As for autonomous vehicles, there is no idea on when this will become reality. Fully autonomous cars are being tested every days, also changing manufacturers predictions. For example, autonomous systems specialist NVIDIA announced that a level 5 vehicle will be on the roads by 2025 [10].

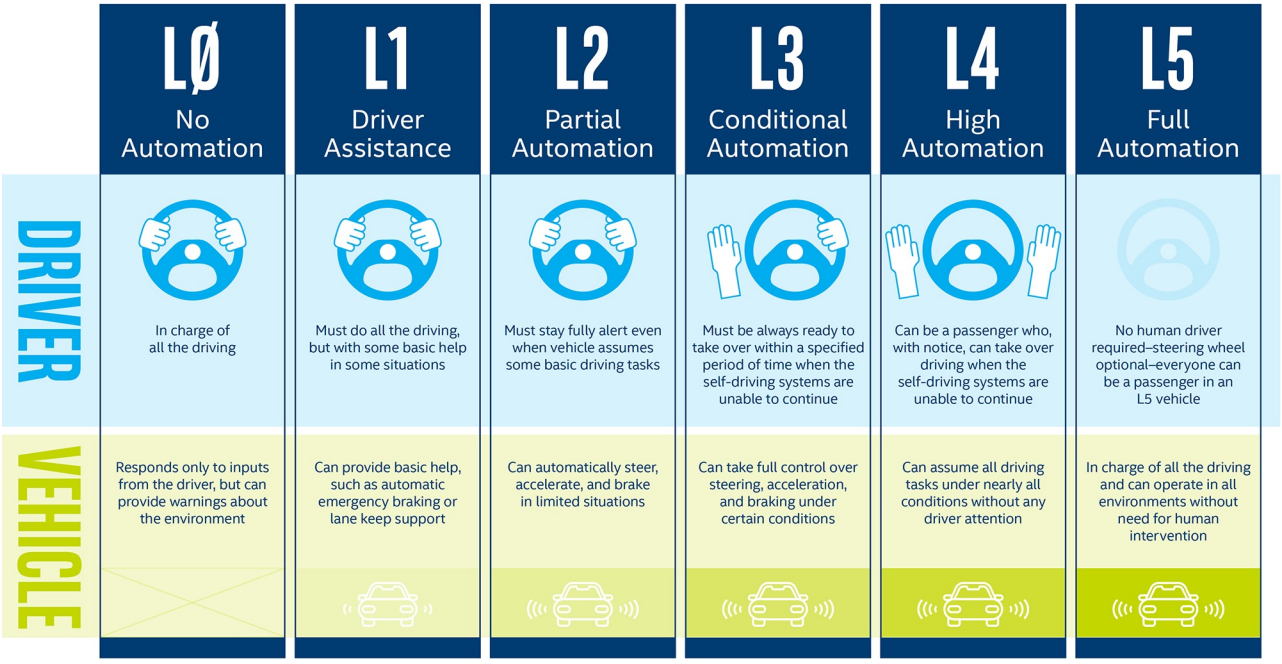


Figure 2.1: Levels of driving automation [11].

2.2 Sensorial Perception

In order to achieve higher automation levels and continuously remove the responsibility from the driver, the vehicle structure must merge a panoply of sensors as Cameras, Sonar (Sound Navigation and Ranging), Radar (Radio Detection and Ranging) and LiDAR (Light Detection and Ranging), this array of remote sensing technologies ought to be combined to provide an uninterrupted perception of the car surroundings in diverse scenarios [12]. These automotive sensors fall into two categories: passive and active sensors. Passive sensors, such as a camera, can only be used to detect energy when the naturally occurring energy is available. For all reflected energy, this can only take place during the time when the sun is illuminating the Earth. On the other hand, active technologies (Radar, Sonar and LiDAR) provide their own energy source for illumination. The sensor emits radiation which is directed toward the target to be investigated. The radiation reflected from that target is detected and measured by the sensor [2]. These

automotive sensors can be analyzed according to this criterion in the next figure 2.2.

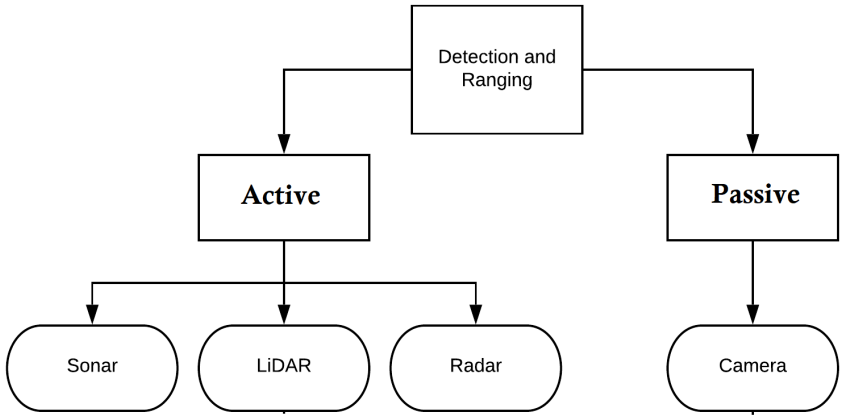


Figure 2.2: Remote sensing technologies taxonomy (adapted from [2]).

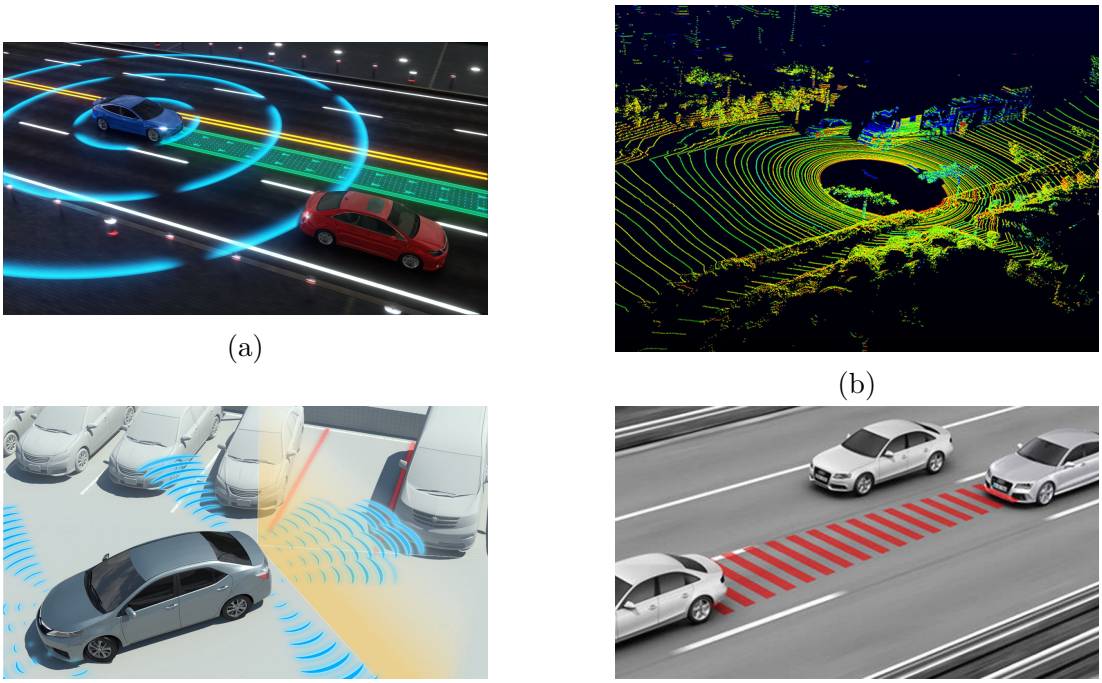
The principle of LiDAR is really quite simple, a laser source shine a small light at a certain surface within a certain FOV. The laser light emitted is transmitted to the target, when this light interact with the target a parcel of this light is reflected and returns back against the receiver, depending on the target's reflectivity (Figure 2.3a). Once the individual readings are processed and organised, the LiDAR data becomes point cloud data (Figure 2.3b). The initial point clouds are large collections of 3D elevation points, which include x, y, and z, along with additional attributes such as GPS time stamps if available.

Sonar principle is the same as LiDAR. The sensors send out short ultrasonic impulses which are reflected by barriers. The echo signals are registered by the sensors and are evaluated by a central control unit, this system is already integrated in most of the commercial vehicles for parking assistance (Figure 2.3c). Through the pulse time take to travel the sending and receiving trip, t_d, the target distance, d, can be calculated using:

$$d = \frac{t_d \cdot v_s}{2} \tag{2.1}$$

The factor 2 must be introduced in this formula since it represents the sending and returning trip that the pulse must take, the speed of sound in the middle is repented by

v_s. They're able to see through objects unlike LiDAR and are relatively cheap, however they don't have the resolution to detect small objects or multiple objects moving at fast speeds.

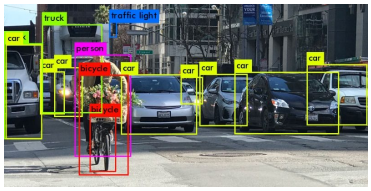


(a) (b) (c) (d)

Figure 2.3: Automotive sensors: (a) LiDAR [13]; (b) example of LiDAR point cloud [14]; (c) Sonar in a parking assistance maneuver [15] and (d) Radar application [16].

In Radar an extremely shorts bursts of radio waves in the radio-frequency (RF) spectrum are transmitted, reflected off a target and then returned as an echo, the returned fraction is detected and processed to determine the distance (Figure 2.3d), this phenomenon is called Echo Principle. Contrasted to the ToF technique, the radar can adopt frequency modulation, in which the range is estimated through the frequency shift between the emitted and received signals. Since the three remaining sensors actively send a signal to the targets, they are more suitable for distance measurement.

(a)



(b)

Figure 2.4: Sensorial perception: (a) 3D image recreation by camera images[16], and (b) object detection with camera image.

Lastly, cameras are increasingly part of today's vehicles. They are implemented on board to capture real-time images of the entire car surrounding, thus enabling a 360º view (Figure 2.4a). Precise detection of the surrounding area is a crucial basis for the successful application of autonomous vehicles. 3D cameras can also be used to enable an autonomous vehicle to precisely recognise its own position and that of the objects around it at any time in order to facilitate the accurate coordination of manoeuvres [17]. Subsequently, deep learning algorithms deal with these images, distinguishing and classifying animals, vehicles, pedestrians, among others (Figure 2.4b).

2.3 Sensorial Fusion for Autonomous Driving

There are essentially three main system functions of self-driving cars: (1) car sensor-related system functions, (2) car processing related functions that we tend to consider the AI of the self-driving car, and (3) car control related functions that operate the accelerator, the brakes, the steering wheel[18].To collect and analyze data, several detection techniques are used for autonomous vehicles such as LiDAR, Radar, Cameras and Sonars. As an autonomous management is a high responsibility task, different scenarios must be chosen and verified. Instinctively, as all sensors differ in operating principles, their characteristics will also be contrasting and each one will have a different applicability. The differences between the four remote sensing technologies are gathered in the Table 2.1.

Table 2.1: Advantages and Disadvantages of each sensor

	Advantages	Disadvantages
LIDAR	• 3D Mapping top sensor [5]; • Scan more than 150m in all directions; • Provides additional data that may be useful; • Works both day and nigh night can not be blinder by the sun, unlike a camera. • Requires minimal human supervision [3].	• Large amount of Data; • Expensive sensor [3]; • Affected by atmospheric weather conditions; • Powerful laser beams can potentially affect the human eye.
RADAR	• Able to give the exact position of an object [5]; • Can tell the difference between stationary and moving targets; • Can work in every condition; • RADAR can penetrate mediums such as clouds, fogs, mist and snow;	• Shorter range and takes more time to lock an object than LIDAR; • No exact 3D image of the object due to the longer wavelength; • Takes more time to lock an object, unlike LIDAR pulses.
CAMERA	• Not blind to weather conditions such as fog, rain and snow [5]; • Less expensive sensor systems [3]; • The best for scene interpretation (cameras can see color); • Able to see the world in the same way as humans (read street signs, interpret colors).	• Large amount of data; • Intense computational processing and complex algorithmically job; • Quite poor in the third dimension (distance), and very limited spatial resolution [5]; • Dependence on environmental lighting.
SONAR	• Excellent performance at short ranges (up to 2m); • Cheap sensor; • Aimed at detecting large objects made of solid materials.	• High affected by noise and atmospheric attenuation; • Poor angular resolution; • Short-range.

Analyzing the table above, it is concluded autonomous vehicle uses a large number of sensors to understand its environment, to locate and move and all this sensors have advantages and disadvantages, their integration in the vehicle and cooperation is therefore important in order to create a more autonomous and redundant system to prevent failures that can be fatal [34]. The figure 2.5 summarizes how the sensory fusion is crucial to create a 360º view, thus optimizing the potential of each sensor. As vehicles advance to higher levels of the SAE automated driving scale, more sensor inputs will be needed. Some architectures may use more than two dozen sensors to create the 360° view needed for fully autonomous driving.

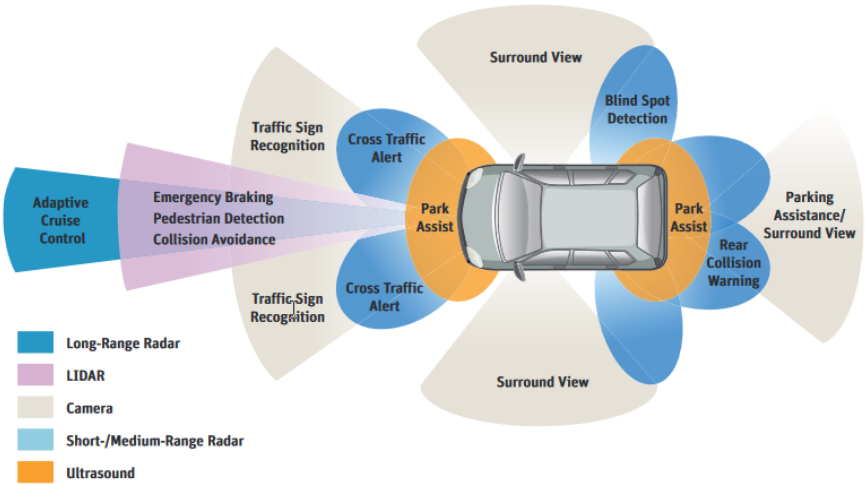


Figure 2.5: Vehicle Sensor Fusion [18].

2.4 Automated Driving Solutions

According to CBInsights data, are identified more than 40 companies developing road-going self-drivings vehicles [19], what makes impossible refer all of that works in this study. A really good example is Tesla Motors founded in July 2003, this company name is a tribute to inventor and electrical engineer Nikola Tesla. Tesla company is responsible for "Autopilot" creation, its semi-autonomous driver-assist technology, developed in 2014. Despite the title, Tesla's vehicles are only designated autonomy Level 2 by SAE [8], meaning they are capable of some autonomous maneuvers but are not considered fully autonomous. Development is currently underway on a next-generation chip that Musk, Tesla Motors CEO, claims will be "3 times better" than the current chip. Musk, also claims that the US electric vehicle maker will be able to launch fully autonomous robotaxis by the end of 2020 [19]. Side-by-Side to Tesla Motors, Google Self-Driving Car Project is one of the most iconic autonomous vehicle programs, considered by many to be the leader in self-driving car tech [20]. At the end of 2018, unlike its competitors, the company launched a commercial autonomous ride-sharing service in Arizona called Waymo One,

which allows users to ride in a self-driving van to do things like pick up groceries or go to school [21]. According to the company, until 2020 Waymo's self-driving has navigated more than about 33 million kilometers on public roads, where they had been testing driverless cars without a safety driver in the seat, across over 25 cities [20]. Many others companies such as, Ford, Toyota, Apple, Aptive, Audi, BMW, Continental, Bosch, Nissan, Volkswagen, Valeo, Samsung, are working and merging in this area and it is expected that over the years more companies will join this business area.

2.4.1 Partnerships

Collaboration in the autonomous driving area is increasingly common. Companies hope by collaborating and sharing their data it will speed up the process of developing fully autonomous transportation, besides automakers turn with fair frequency to startups and younger technology companies to supplement their in-house development efforts. Nearly every major multinational automaker and many suppliers have at least one active partnership, as described on Figure 2.6.

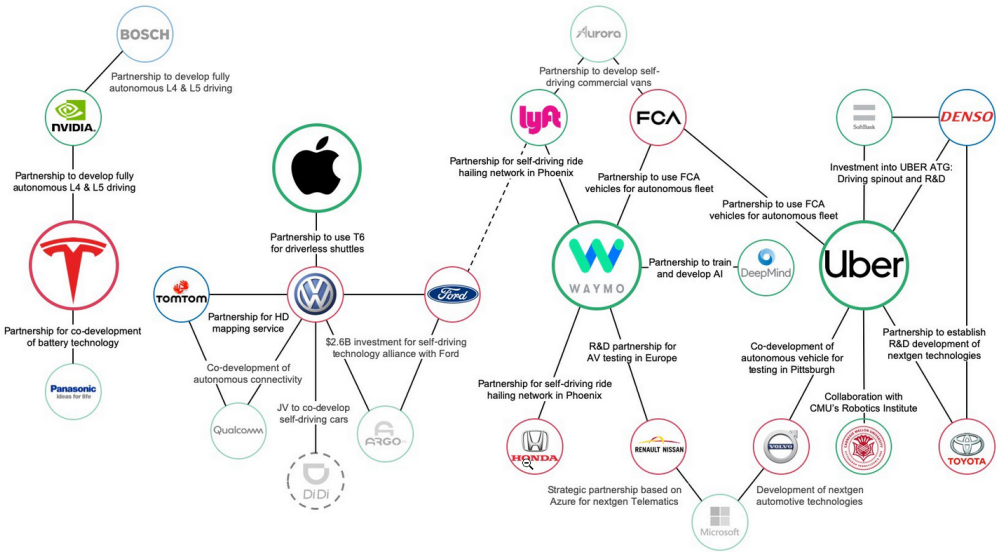


Figure 2.6: Autonomous Driving Partnerships: Red, blue and green respectively represents vehicles manufactures, traditional automotive suppliers and tech companies (Adapter from [22]).

Afterwards, single organizations or institutions cannot solve the challenges of individual mobility. Nor can these challenges be resolved by reference to a single discipline.

In the end, interested entities within industry, academia, and government need to work together, especially when addressing interdisciplinary topics in an emerging field. Academia in particular, with its focus on research and its relative neutrality, is well-positioned to take a leadership role [23].

2.5 Autonomous Driving Simulators

As mention before, driverless vehicle should be driven hundreds of millions of kilometers, and in some cases hundreds of billions, over the course of several decades to demonstrate their reliability in terms of fatalities and injuries. In order to meet this challenge, the researchers pointed to innovative testing methods such as advanced simulation technologies. The commercial models for such tools are extremely diverse, ranging from open-source simulation implementations and hobbyist platforms based on gaming engines, to professional grade commercial modelling tools and extreme-performance simulation technologies. In the context of these different toolsets and their distinct technical capabilities, it is difficult to define which tool is "state-of-the-art", as this can only be defined within the scope of the user's requirements. Nevertheless, a brief list of simulators will be describer bellow [24].

2.5.1 Nvidia Drive SIM

NVIDIA software is a simulator incorporated in NVIDIA drive constellation platform which simulates a virtual world and generates sensor output from a virtual car. The simulated sensor data is then sent to the target vehicle hardware in the drive constellation computer, which processes the data and sends driving decisions back to the simulator. Simulates edge cases and hazardous situations that autonomous vehicles could encounter in the real world, from severe weather, to difficult lighting, to risky maneuvers by surrounding vehicles. This platform enables millions of kilometers to be driven in virtual environments

across a broad range of scenarios, from routine driving to rare or even dangerous situations with greater efficiency, cost-effectiveness, and safety than what's possible in the real world [25].

2.5.2 Carla

CARLA is an open-source simulator for autonomous driving research. This simulator has been developed from the ground up to support development, training, and validation of autonomous urban driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites and environmental conditions. CARLA is used to study the performance of three approaches to autonomous driving: a classic modular pipeline, an end-to-end model trained via imitation learning, and an end-to-end model trained via reinforcement learning. The approaches are evaluated in controlled scenarios of increasing difficulty, and their performance is examined via metrics provided by CARLA, illustrating the platform's utility for autonomous driving research [26].

2.5.3 Waymo's Carcraft

Carcraft, the name of Waymo's simulator, was originally developed to replay driving situation that Google's cars experiences in real life. The simulator has since then been developed into an integral component of Waymo's efforts. According to a report in the magazine "The Atlantic" [27], which got a first glimpse into Carcraft, entire cities are now simulated with 25 thousand autonomous cars at the same time driving in it. Especially difficult situation like roundabouts or cross traffics are recreated and can be driven hundreds of thousands of times per day by the virtual cars. Every day the cars drive 8 million virtual kilometers which adds up to 2.5 billion kilometers annually. The simulator recreates the streets with high precision. Even the curb heights are measured, besides the exact position of lane markings, speed limits, or the position of traffic lights.

With that data and the vehicle logs from real world drives the perception (measurement), prediction, and motion planning can be simulated. According to Waymo-CEO John Krafcik 80 percent of improvements are now coming from the self-driving algorithms in the simulator [27].

2.5.4 LGSVL

LG Electronics America R&D Center has developed a Unity-based multi-robot simulator for autonomous vehicle developers. An out-of-the-box solution which can meet the needs of developers wishing to focus on testing their autonomous vehicle algorithms. It currently has integration with the TierIV's Autoware and Baidu's Apollo 5.0 and Apollo 3.0 platforms, can generate HD maps, and be immediately used for testing and validation of a whole system with little need for custom integration's. LG hope to build a collaborative community among robotics and autonomous vehicle developers by open sourcing theirs efforts [28].

2.5.5 Robotic Simulators

Others commercials and open source 2D and 3D simulators been created. Focus on 3D side, some examples like Gazebo, which provides a graphical user interface (GUI) for help in sensors integration into the vehicles and to develop realistic outdoor scenarios, can also be integrated with Robotic Operating System (ROS) which make this simulator very popular under Robotic Community, is not a open source while new plugins can be uploaded with the user sensor and actuator modules [29]. Other simulator example can be USARsim, designed manly for search and rescue operations, is generally used in Robocup competitions, as a disadvantage this software uses a external communication layer that doesn't support some robotics middleware without additional programming. Also Webots is a commercial simulator equipped with a full programming environment to create robots and environments. At least, Modular Open Robots Simulation Engine (MORSE) runs on a Python language and uses Blender Game Engine to interface 3D environment with external robotics software, incorporated with a lot of sensors which allow the system the

choice of different levels of realism and abstraction. Several others simulators for robotic systems can be enumerated, such as Über-Sim [30], SimRobot [31], UCHILSIM [32], Stage [33], V-REP [34], Delta3D, X-Plane, Microsoft Flight Simulator, Actin, Microsoft Robotics Developer Studio (MRDS) [35] [36] [37], among others as deescriber in [38].

2.5.6 Robotic Simulators Comparison

The following Table 2.2, describes a summary of the main characteristics of commercials and open source simulators, the table has already created one column about the SimTwo simulator, since it was the simulator chosen for the realization of this work. The software is analyzed according to the following criteria. (i) Operating system (OS); it describes which operating system is compatible with the robotic software. (ii) type of simulator; this parameter describes whether the robotic software provides a 2-D or 3-D simulation environment. (iii) Programming language: demonstrates the supported language . (iv) Documentation; it describes the level of documentation available with the robotic software and can be "high level" or "low level". "High level" means that the documentation provides only descriptions of the functions in the robotic software libraries; "low level" means that the documentation provides the code for the functions in the robotic software libraries. (v) Tutorial; describes whether examples and a step-by-step guide are provided. "Yes" means that a well-defined guide with examples is available; "limited" means that there is a guide, but not enough details and examples are provided. Finally, "No" means that there is no tutorial or other source with examples. (vi) Portability; "Yes" means that the code written for a simulation is portable to a real robotic platform. (vii) Sensors; it describes the supported sensors. Only the most requested sensors are described in the table due to space limitations. (viii) Debugging / Logging describes whether the debugging, fault tolerance, reproduction and logging features are provided by the robotic software. (ix) Graphical user interface; describes whether it is possible to modify objects and the environment during runtime and/or program functions in a development environment [39].

Table 2.2: Characteristics of Commercials and Open-Source Robotic Simulation Softwares (Adapter from [39]).

	Player/Stage	**Webots**	**Easy-Rob**	**Gazebo**	**ROS**	**Simbad**	**CARMEN**	**USARSim**	**MRDS**	**MissionLab**	**SimTwo**	**Rob.Tbx.**
OS	Linux, Mac, Win.	Linux, Mac, Win.	Win.	Linux	Linux, Mac, Win.	Linux, Mac, Win.	Linux	Linux,Win.	Win.	Linux	Linux, Mac, Win.	Linux, Mac, Win.
Simulator Type	2-D	3-D	3-D	3-D	2-D, 3-D	3-D	2-D	3-D	3-D	3-D	3-D	2-D, 3-D
Programming Language	Player(any) Stage(C,C++, Python,Java)	C, C++, Python,Java, Matlab	C++	C, C++, Python,Java	C++,Python, Octave,LISP, Java, Lua	Java	C,Java	C, C++, Java	VPL,C#, VisualBasic, JScript,Iron-Python	VPL	Pascal	C-like
Documentation	Low-Level	High-Level	High-Level	Low-Level	High-Level	High-Level	Low-Level	High-Level	High-Level	High-Level	High-Level	High-Level
Tutorial	YES	YES	YES	YES	YES	Limited	Limited	Limited	YES	Limited	YES	Limited
Portability	YES	YES	NO	YES	YES	Limited	YES	YES(using Player)	YES	YES	YES	NO
Sensors	odometry,range	odometry, range,camera, GPS	Odometry	odometry, range,camera,	odometry, range,camera,	vision range, camera	odometry, range,GPS	odometry, range,camera touch	odometry, range,camera,	odometry, range,	odometry, range, camera.	odometry,range
Debugging/Logging	YES	YES	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
GUI	NO	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES	YES

2.5.7 Simulation Lack and Future

The big question is to figure out what can be tested in the simulation software environment and what needs to be left for the physical testing.

To know how to drive, a self-driving car needs to be able to predict what other road users will do, and it needs to be able to communicate and interact cooperatively with other road users, particularly with human drivers. Predicting what road users will do requires observing them empirically, and knowing how road users communicate and successfully interact requires empirically observing how humans do those things. That's why the knowledge about how to drive is not contained in any simulation, unless that simulation is running neural networks trained on large datasets of such empirical observations [40]. To learn how to drive using imitation learning and reinforcement learning, a self-driving car may need to draw upon tens of thousands of hours of continuous driving. As with deep learning and reinforcement learning generally, the amount of empirical observation and experience needed may be massive. Almost all autonomous vehicle companies are trying to learn with a fleet of just a few hundred cars. This may not be enough to get the empirical data required. Companies should work on putting sensors, computing hardware, internet connectivity, and driver assistance software into the millions of cars that are produced every year. This is not guaranteed to work, but it is practically guaranteed to work better than relying on a few hundred vehicles to get the data is needed [41].

-This page intentionally left blank-

Chapter 3

Sytem Architecture

In this Chapter is presented a simulation environment, "SimTwo", where multiple kinds of robots and environments can be modeled, the added value of using a tool that allows simulations of the system under development and improvement is highlighted. With the simulation platform, damage to the vehicle and sensors can be avoided and it also allows experiences with the different control and decision strategies to be applied.

3.1 SimTwo Overview

SimTwo is a open source simulator developed in Object Pascal, where a multiple kind of robots can be modeled: differential, omnnidirectional, industrial, humanoid, are just a few examples. This simulator has the functionality of use specified models for characterized components, essentially sensors, bodies and motors. Even though it is not possible to upload new modules, SimTwo is capable of robot development in different programming languages and commercial software such as MATLAB and LabVIEW [29]. Decomposing a robot in rigid bodies and electric motors allows a dynamic realism to the simulator. Running in a multi-window format, where many windows as "code editor", "spreadsheet", "configuration", "scene editor", "chart", "camera" and "main view" share the process, as shown in the following figure 3.1.

The “configuration” window offers control over five important configuration sub-tabs

(Control, Graphics, Debug, Physics and I/O). It is possible, between other options, to define the values for PID control and to configure the 3D graphics world view, where camera position, shadow visibility, axis, can be highlighted. On the last sub-tab, "I/O", it is possible to external communicate through Serial Port. The “code editor” is an integrated development environment (IDE) for Pascal Language programming, viewed as the principal tool window in this simulator.

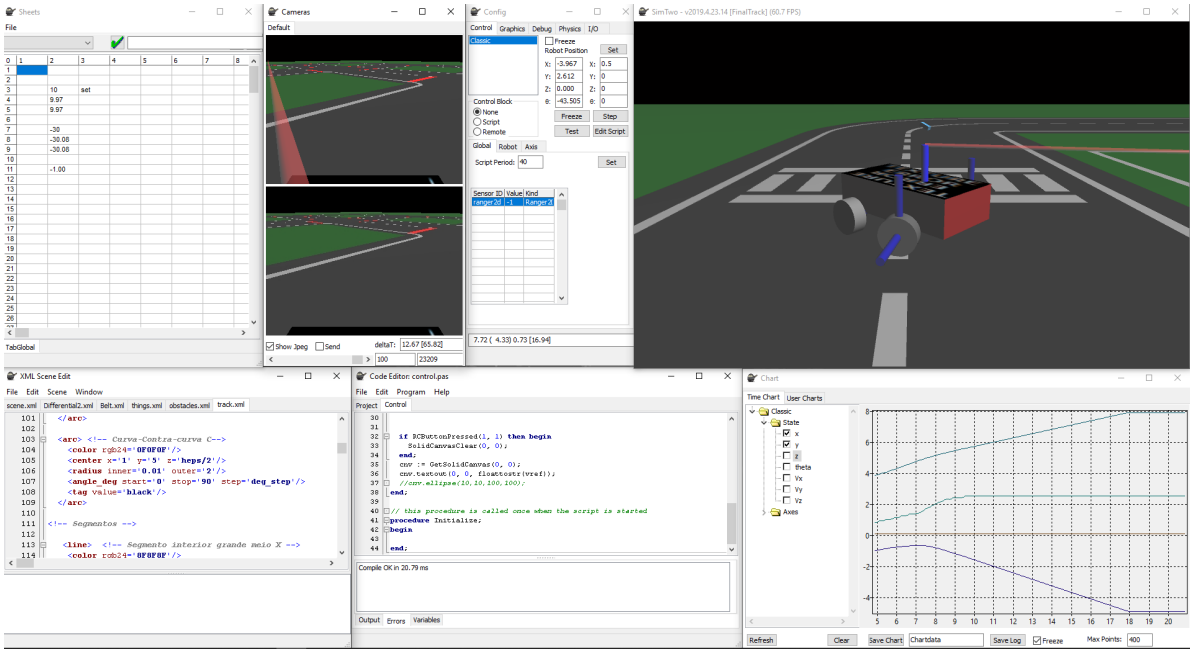


Figure 3.1: SimTwo software: Application windows clockwise from top left corner: spreadsheet, cameras, configuration, world view, scene editor, code editor, and chart (adapter from [29] and [42]).

The "Chart" and "Spreadsheet" tabs are the main windows for debugging the control algorithm. It is possible to plot many robot variables, either as position or motor speed, afterwards this variables can be saved for external use, as a ".EMF" file. In the "spreadsheet" window it is possible to create a Graphic User Interface (GUI), where "edit cells" and "button cells" can be specified for User-Machine operations, and *vice versa*. This simulator allows camera functions for data input, at that time a camera window can be used to observe the scene as a simulated camera. Lastly, edit several XML format files is possible using the "scene editor" window. SimTwo represents a scene along "robots",

"obstacles" and "things", as shown in Figure 3.3. Each robot is defined according different solids and connected thought joints. The shell elements are solids without mass and a crucial part for collision simulation. A robot can also have sensors, these provide information from the environment surrounding to the robot or the scene can have static sensors relative to the world. The “obstacles” and “things” are scene objects, these are very similar but while the “obstacles” are immovable in case of collision the “things” are not [29].

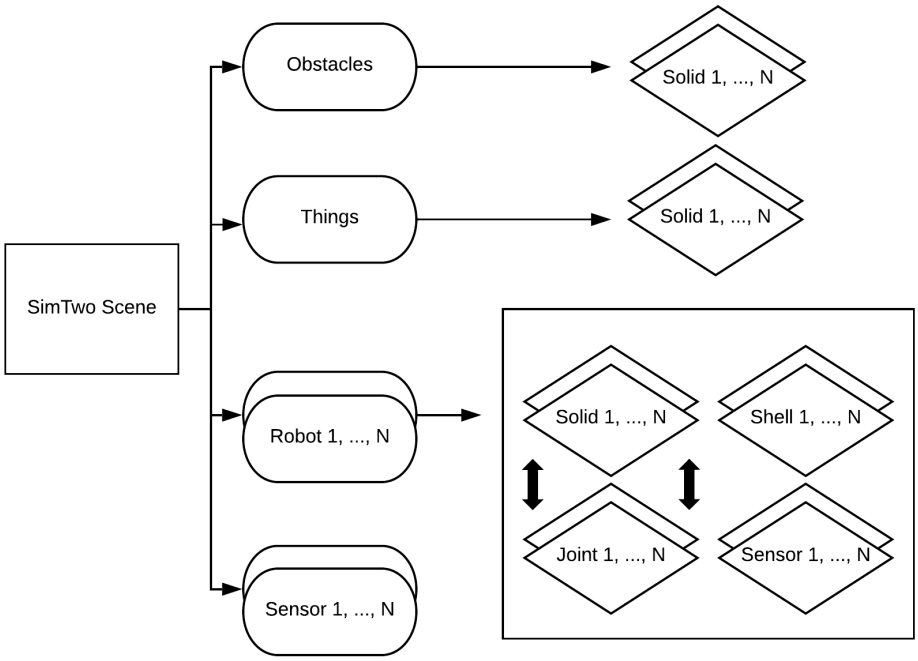


Figure 3.2: SimTwo Scene editor structure (adapter from [29]).

3.2 Robot Model

In the next subsections, all the robot physics will be described in order to better understand the robot kinematics and structures. The main sensors used by the robot to sense and visualise the environment is also explained and detailed. Finally, all the track where the robot should perform is described.

3.2.1 Robot Body

A robot in SimTwo is composed by a system of rigid bodies, electric motors, articulations, shells and sensors. The dynamics related to each of the components is simulated based on their physical properties such as, shape, mass, moment of inertia, forces from friction between the bodies, and elasticity.

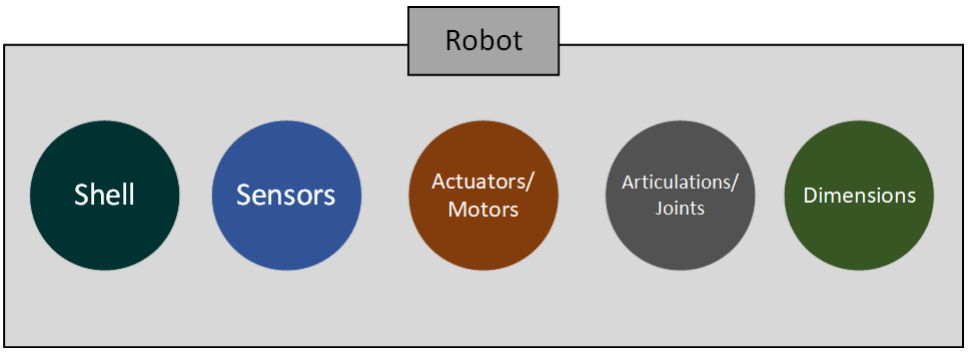


Figure 3.3: SimTwo robot configuration (adapter from [43]).

A robot is made out of combining basic shapes together, namely combination between cylinders and parallelepipeds. The robot created for this project is basic one base body, where all sensors and wheels are connected. The four wheels connect via four axis, recreating a real steering car system. The wheels are made out of four cylinders with wider radius and shorter height, they are linked through an articulation to keep them fixed. These robot shapes are defined under the "solids" section of the code (differential2.xml) and illustrated by the figure 3.4.

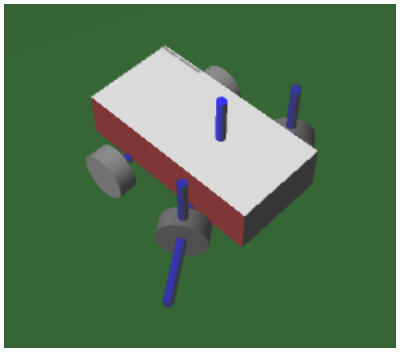


Figure 3.4: Four wheels robot modeling.

The robot dimensions are defined separately by variables, inside the "defines" section of the code (differential2.xml), to easier edit the robot. The joints and articulations are defined in "articulations" section of the code (differential2.xml) and include the motor position, which is inside the wheels. The shells section define the laser ranger and camera sensors specifications of the robot, in case of the laser other section called sensors is needed to defines the type of sensors, how many and what their layout is. All the code definition is represented in [44].

3.2.2 Sensing and Perception

The robot is equipped with a Red, Green and Blue (RGB) camera, figure 3.5, is the main sensor to see all the environment and is the main responsible for give the input to the autonomous system. This RBG camera has a 360ºx240º of horizontal and vertical resolution, respectively.

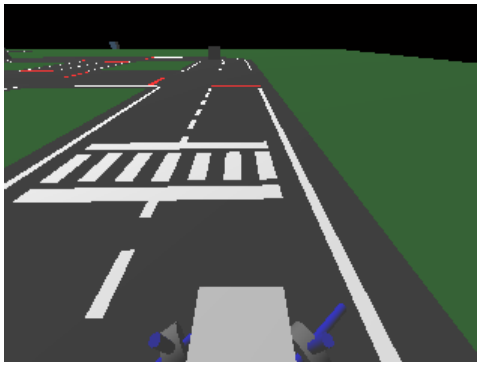


Figure 3.5: Robot RGB camera view.

On scene editor is possible to edit all RGB camera proprieties, like size, position, focal length, frame decimation, rotation angle, and even the color of the object that represent the camera sensor. This configuration should be performed on shells section of the robot XML file. There are different cameras that provide different image formats. The most known are the RGB format and the YUV format. The RGB format was adopted with the characteristics showed in the next table 3.1.

The acquired image (Figure 3.5) it's crucial for real-time control of the system. The

Table 3.1: RGB camera proprieties.

	x	y	z	Value
Size	0.01	0.01	0.03	-
Position	0.05	0	0.5	-
Focal Lenght	-	-	-	15
Frame Decimation	-	-	-	4
Rotation (deg.)	130	0	190	-
Color (RGB)				50/140/200

acquisition of the images is performed at 15 FPS. As an auxiliary sensor, the LiDAR, figure 3.6, was introduced with a resolution of 150º on the horizontal axis and with a total of 10 lasers beams. This sensor is essential for capturing additional information. As for camera,

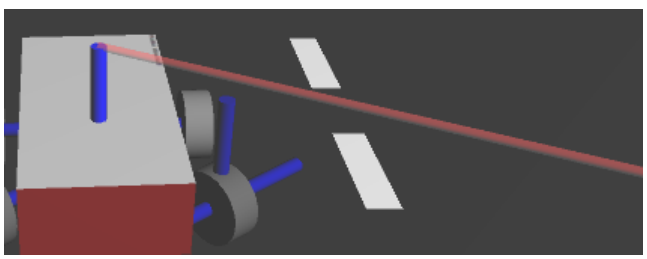


Figure 3.6: Robot Lidar Beam.

LiDAR sensor needs to be configured on robot XML file, but in a different mode than camera. For this sensor a format shape must be created as for camera, included the size, position, rotation and color but needs also a configuration for the beam on sensors section of XML code, where a 2D ranger is described. In this section is possible to describe such configuration as the beam length (6 meters in this particularly robot case) with an initial and final width (5 millimeters in bought cases were adopted), noise values, offset values, gains, period values and the color of the beam. All this parameters are defined on sensors sector of XML file, as described on the next listing.

Listing 3.1: LiDAR parameters definition

```
<ranger2d>
    <ID value='ranger2d'/>
    <beam length='6' initial_width='0.005' final_width='0.005'/>
    <period value = '0.1'/>
```

```
    <pos x='0.05' y='0' z='0.15'/>
    <rot_deg x='0' y='0' z='0'/>
    <tag value='00'/>
    <beam angle='150' rays='10'/>
    <noise stdev='0.00' stdev_p='0' offset='0' gain='1'/>
    <color_rgb r='254' g='0' b='0'/>
  </ranger2d>
```

3.3 Track Modeling

In order to recreate a real driving environment, a track with two driving lanes was created, figure 3.7, where several curves and intersections were introduced. The track layout is inspired by the track used by Duckietown Project [45] and provide several scenarios for the robot to sense and act. Obstacles and stopping points were also introduced to recreate a real road environment where several unexpected situations occur, thus making the movements and decisions that the robot must take more difficult. As shown on figure

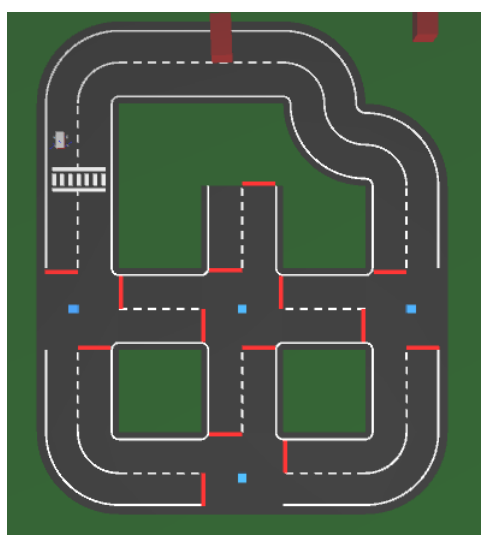


Figure 3.7: Simtwo track layout.

3.7, the track as four crossing points where four traffic lights were created in order to give permissions, about stop or go, to the robot. The color of this traffic light, either

green or red, determines the next step to the robot. An obstacle, that must be overtaken, was placed in one of the main straight lines, and a crosswalk for possible pedestrians take also place. The track presents continuous lines in all the curves and discontinued in straight lines where it is possible for the robot to safely choose the opposite lane, thereby recreating even more a realistic scenario.

Chapter 4

Development

In this chapter all the implementation work is presented and detailed, pointing the must important parts of the code development, particular difficulties encountered and technical solutions developed applied. Started with a interpretation and analyse of the main control program to sensors algorithms in order to recognize the environment and take decisions based on scene view. During this chapter is also explained the construction of the entire scene and the debug interface where the user as the possibility to intervene with the simulation.

4.1 Control

Artificial vision embedded in a Robot, sometimes replaces expensive sensors and allows a valid control. The acquisition of color image grants a distinction between different markers, traffic lights or traffic drivers. The vehicle follows, in real time, a route defined by bounding lines, namely the right bounding line of the carriageway. In most autonomous vehicle applications, location and orientation (posture) are important and a decisive criteria. Once the robot's stance is known, this robot can follow a planned path or define one. Through the model, created by the vehicle sensors, and equations that define the dynamics of the robot, it is possible to control the robot movement according to a planned route.

4.1.1 Main Program

The main program is split in eight states. When the system is initialized and the camera imagine is on, the robot starts to follow the right line of the track (state 1). In this state the system will only stop if the user decide it, otherwise the system will keep running until some red line stop is reached or some minimum distance of the laser beams is activated, jumping to state 5 or state 2, respectively. When a red line is detected the robot stop right over the limitation line, afterwards the user must decide each direction the robot should take between left (state 7), right (state 8) or keep straight (state 6). Subsequently the robot will return to first state and wait for a new instruction to move on. Even on state 1, according to robot laser sensor readings, the robot can detect some obstacle on the track and jump the control to state number 2, where a predefined operation of obstacle outline will take place. This operation contemplates the states number 3 and number 4, and as mentioned is initialized on state number 2. In these three last installments, the robot follows the left line in the state number 3, and in the other two operations it reproduces a programmed odometric behavior.The referred state machine of all system can be better analyzed on figure 4.2 and pascal code on listing 4.1 as well.

Listing 4.1: State machine described in pascal code.

```
if (state = 3) then begin FollowLeft();
end else if (state = 1) then begin FollowRight();
end else if (state = 2) then begin TurnLeft();
end else if (state = 4) then begin TurnRight();
end else if (state = 5) then begin RedStopLine();
end else if (state = 6) then begin Straight();
end else if (state = 7) then begin Left();
end else if (state = 8) then begin Right();
end;
```

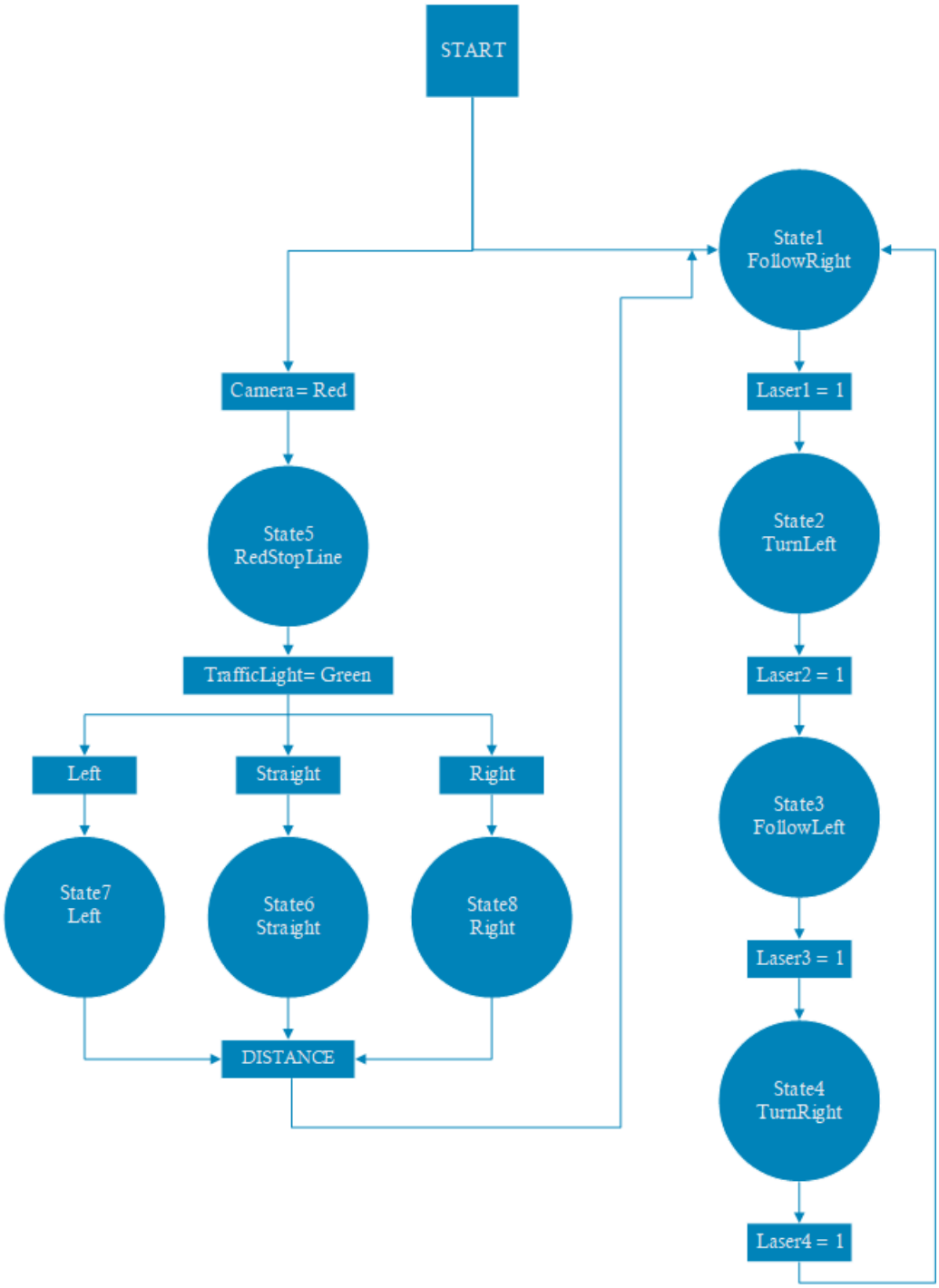


Figure 4.1: Finite-State Machine.

4.1.2 PID Motors Control

From the system control point of view, the operation is based on essentially two control speeds (W_r, W_l) that will ensure movement to the system. In fact, and in a highest level, the control is based on determine the travel speed v and rotation w, where K_1 and k_2 are the system dynamics gains. Afterwards, these speeds let on calculate the drive speeds of the right wheel W_r and left wheel W_l, as indicated in equations 4.1 and 4.2.

$$W_r = K_1 \cdot v + K_2 \cdot W \tag{4.1}$$

$$W_l = K_1 \cdot v - K_2 \cdot W \tag{4.2}$$

A PID control is implemented for each wheel in closed loop (figure 4.2).

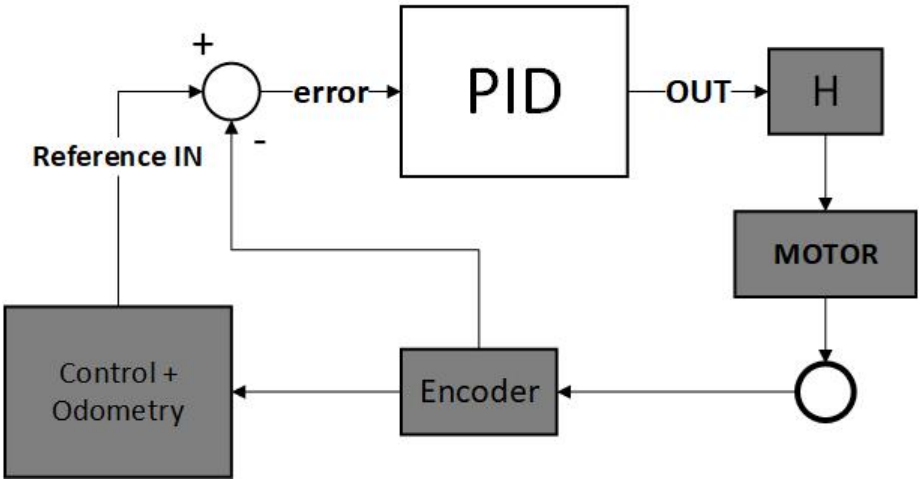


Figure 4.2: PID close loop control.

4.1.3 Image Sense and Recognition

The SimTwo simulator is able to place several cameras in various modes in the robots, including: (1) fixed camera in a certain position, (2) mechanical rotative camera (associated with a motor) and (3) fixed camera pointed at one or more mirrors. The method used in this project corresponds to the first one presented and guarantees, in this case, the desired requirements which are follow a track delimited by lines. As it is required to

the vehicle to make several stops at some crossing points, this same front image of the vehicle is sufficient for the main control, thus avoiding the placement of a second or third camera, as shown in figure 4.3. Afterwards, with each arrival of a frame, it's processed according to the algorithm and determined the linear and rotational speed to be applied to the robot at each instant of time.

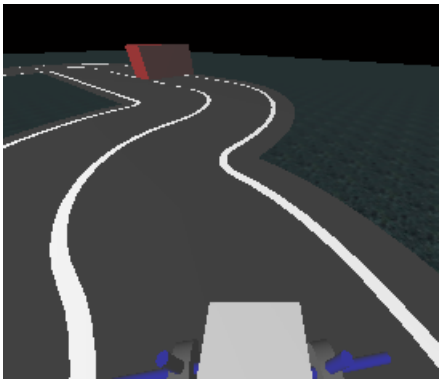


Figure 4.3: Camera acquired image.

At this point, it's intended to control the vehicle in order that follows the highway boundary line in a autonomous way.

Thus, the developed algorithm scans three horizontal lines, figure 4.4, strategically placed, and determines the three tone transition points (track boundary line $P1$, $P2$, $P3$) that allow the location and control of the robot inside the track.

The right and left engine speeds are determined in such a way that, for each horizontal line, the point in the middle of the transitions coincides with the middle of the screen. In reality, horizontal lines will not have all identical weight on the calculation. The line furthest from the robot will be less important for movement control and the closest line will be the one with greater importance, otherwise the system can represent a certain instability, however these parameters can be changed during the program itself. It could be said that the line furthest from the vehicle serves to predict or estimate the change in direction in which the robot will be subject within a short range of time, or in the near future. In case of a line does not detect the transition of the track, this is ignored and only the two others visible will affect the calculation. In the impossibility of check all

transitions the system adapts as described on the following part of the main control code [38].

Listing 4.2: "If not detected line" Pascal Code.

```
erro3:= (Tpoint3 - 169);
erro4:= (Tpoint4 - 197);
erro5:= (Tpoint5 - 222);

if (tpoint3 <> 0) or (tpoint4 <> 0) or (tpoint5 <> 0) then
w:= 0;
if (tpoint3 <> 0)  then
w:= erro3*k3;
if (tpoint4 <> 0) then
w:= w + erro4*k4;
if (tpoint5 <> 0) then
 w:= w + erro5*k5;
```

The algorithm starts to measure the error in each transition (black to white) by subtracting the founded value to the value obtained by the camera during the transition when the vehicle is stationary and perfectly aligned with the middle of the screen, in this case this values were 169, 197 and 222. Afterwards, a simple "if" conditional section increments the need to correct the robot trajectories performed, for each transition a constant (k_3, k_4, k_5) is multiplied by the error value, this value represents the weight on the calculation of each line.

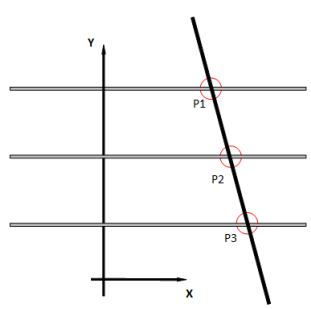


Figure 4.4: Line Detector

For example, if only one visible transition (of the three available) occur, the system tries to maintain the distance that existed from that point to the center of the image, which has been determined and memorized. Like this the system ensures to be well behaved when, in real conditions, a possible noise makes impossible to view some image area. By knowing the coordinates P1 to P3 shown in Figure 4.4, the next listing can be performed and the desired angular speed calculated.

Listing 4.3: "Velocity as a function of direction" Pascal Code

```
steer:= rad(w);
SetRCValue(2, 3 ,format('%.3g',[w]));
SetRCValue(2, 2 ,format('%.3g',[steer]));

vref := -(abs(w)$ divided by 35) + v ;
SetRCValue(3, 2 ,format('%.3g',[vref]));
```

The velocity of linear movement may be written as a function of the velocity of rotation, once the vehicle move in a turn more slowly than in a straight line, like in a real situation.

The instant begins to appear a curve, the furthest line gives its contribution and the vehicle starts to rotate slightly to the center of the curve making possible to turn with a higher speed. If this process is not made, the curve cannot be described with the desired velocity but slower. The use of a velocity variation (acceleration) within certain values, makes starting and braking appear smooth.

The mapping pixel-world becomes essential, as well like correcting image degradation acquired. Once the route is known, or by experience gained by the robot during previous routes, or by coordinates introduced [38].

4.1.4 Obstacle Outline Algorithm

In order to make the simulation more realistic and interactive, an obstacle was placed in an area of the track to make the robot change its normal direction. To this end, a process

was created that is based on three essential steps: detection, execution of the trajectory change and return to the correct lane. Thus, the robot makes use of its LiDAR sensor to detect the proximity of the object and carry out the entire lane change maneuver. As soon as the robot detects that the obstacle has been overcome, the LiDAR sensor gives the order for the robot to return its initial position, as illustrate on figure 4.5.

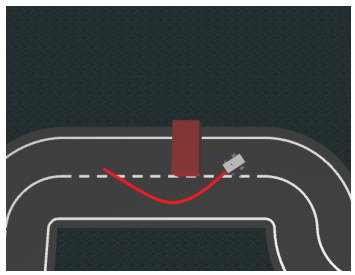


Figure 4.5: Obstacle overtake

For the LiDAR sensor point of view, the figure 4.6 represent the values read by the sensor, where -1 means the sensor did not detect any obstacle and any positive value represents the detection of an obstacle and the concrete distance to the same.

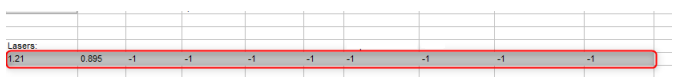


Figure 4.6: LiDAR beams read values

4.1.5 Red Stop Line

In every crossing point the robot must stop before the user take the decision to turn left, right or keep straight. To accomplished that, the camera is constantly reading a strategy line where a threshold will be activated if the red color is seen.

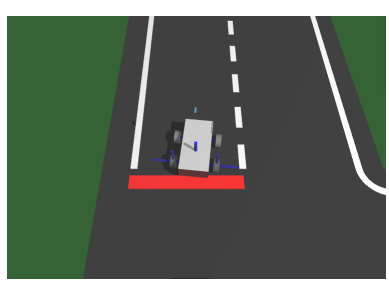


Figure 4.7: Red Stop Line

If this set point is activated the code jumps to RedStopLine procedure, see Listing 4.2, each means the robot will be stopped on the red line until the user introduce some input to the code, by pressing some button on GUI and send the code to other state. Is important to notice this stop process is not immediately reproduced. Because when the robot really understand there is a red line on the track, he really must stop, the strategic line scanned by the camera sensor is not just one line, but four in a row, each make the system redundant and more sophisticated. Meanwhile, this process is important for the robot to save the stopped position values of X,Y and theta angular for calculate the next trajectory.

Listing 4.4: "Red Stop Line Procedure" Pascal Code.

```
procedure RedStopLine;
begin
SetRCValue(2,5 ,'RedStopLine');
vref := 0;
StopXPoint:=robposx;
StopYPoint:=robposy;
StopThetaPoint:= robtheta;

if RCButtonPressed(3, 6) then
state:= 1;
if RCButtonPressed(3, 7) then
state:= 8;
if RCButtonPressed(3, 8) then
state:= 7;
if RCButtonPressed(3, 9) then
state:= 9;
```

In this stop stage and even when the robot is seeing and overtaking an obstacle a warning signal, figure 4.8, is presented on User Graphical Interface as a sign for the attention of the user. This simple flag represent an important alert which must make the user warned and ready to take control of the robot if something goes wrong.

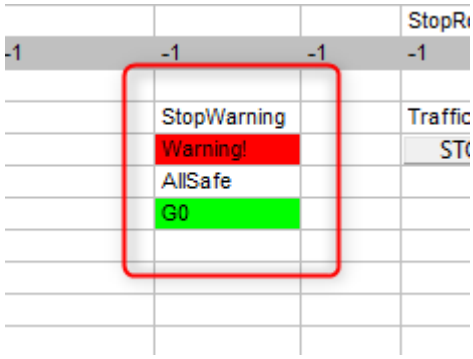
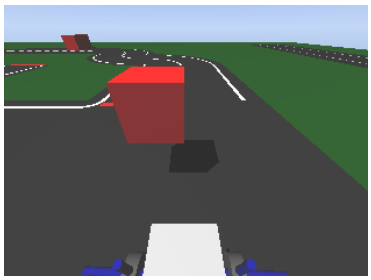


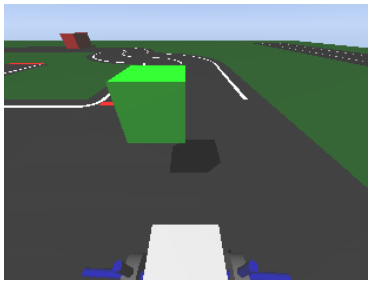
Figure 4.8: Visual warning about the present or not of a track obstacle.

4.2 Traffic Lights

In order to introduce a greater reality to the simulation at the crossing points, four cubes representing the traffic lights were introduced. These traffic lights have a binary behavior changing between two possible colors, red (figure 4.9a) and green (figure 4.9b). The green color represents the authorization for the robot to continue the route and at the red color the robot must remain on the stop line until another state comes, these changes can be visualized in figure 4.9.



(a)



(b)

Figure 4.9: Traffic Lights: (a) Red Light where the robot must stop and (b) Green Light where the robot can move forward.

To create this dynamics of color change, the pre-defined function in the simulator, $SetOsbtacleColor(N, R, G, B)$ was used, where N represents the obstacle number to be colored and the remaining values represent the equivalent value for each one RGB colors

(Red, Green, Blue). An input button has been introduced in the GUI that controls these two states as the user wishes and is explained in next chapter.

4.3 Scene Construction

The whole simulation scene is built with the help of the simtwo simulator by having a scene edit menu based on the XML markup language, which makes the creation of any component very easy and intuitive. The scene is created by join the four main files, represented in the following part of code:

Listing 4.5: Scene XML Code

```
<scene>
  <robot>
    <ID name='Classic'/>
    <pos x='4.3875' y='1' z='0'/>
    <rot_deg z='90'/>
    <body file='Differential2.xml'/>
  </robot>
  <things file='things.xml'/>
  <obstacles file='obstacles.xml'/>
  <track file='track.xml'/>
</scene>
```

The body file is where the robot is defined as explained on chapter 2. Obstacles file is where the code related with the robot must overcome the obstacle and also the traffic lights are represented. Since the traffic lights are represented by four cubes without structure to fix them in the middle of the track, these should be included in this obstacle file and not in the things file, thus avoiding the gravity force that the simulator applies to the bodies. Finally the track file is where all the robot track is defined, using functions as "arc" and "line" is possible to describe all the track.

4.4 Debug Interface

Since the simulator used allows the use of a debug interface, with a well known sheet style, to interact with the main program. A sheet was created with several inputs and outputs on the main algorithm. Thus, it is possible to know in real time the speed at the vehicle moves as well as readjust this value, to know the value of the angle of rotation that the autonomous vehicle is creating, to define initial positions of the robot for the simulation, to raise the values of the gains for each line strategically positioned. Even in this interface the user has an input on the system to decide what the robot should do when it stops at the stop points. The next figure 4.10 better represents all this sectors where the user can interact and analyse the simulation.

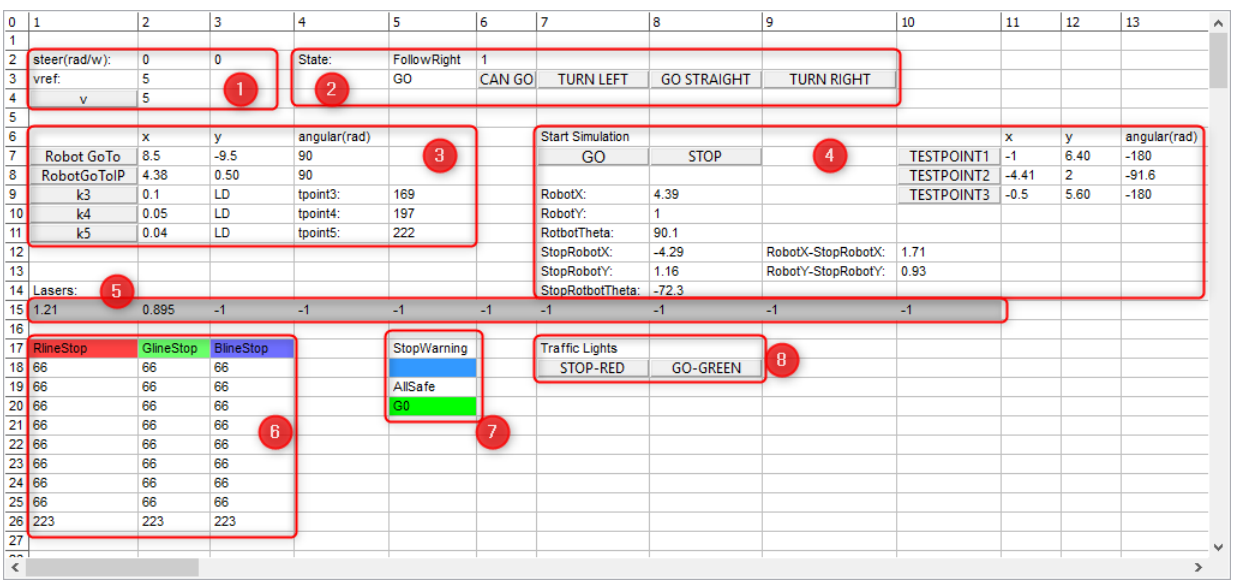


Figure 4.10: Developer GUI as sheet style for user scenario interpretation

The interface is divided into eight control blocks where the user can read and write values, thus acting in the simulation. In block number (1), the user has the option of changing the speed robot reference value, as well to see the real-time value for the steering angle of the robot wheels presented also in radians. In block (2), the current state of the simulation (number and name) is indicated, it is also in this block that the user can change between the three defined states (turnleft, gostraight, turnright) after

stopping at the red stop line. The block number (3) is separated into three sub-blocks, here the user can define two initial positions and the respective angle for the robot to start its simulation, choosing one of them at each simulation start. It is also possible to change, in real time, the parameters of the constants $K3$, $K4$, $K5$ responsible for the robot control. Finally, in this block, the possibility of analyzing whether the transition point that allows the robot to drive has been reached is also introduced, as well as the value of the pixel where this is happening.

The main and central block is the number (4), this is where the user decides whether the simulation should go forward or stop, it is also the place where is possible to analyze the entire odometer of the robot by recording also the stopping points and the distance traveled in programmed mode. Finally, as the scenario also has auxiliary tracks, the user can quickly place the robot in 3 different positions, via test points, to facilitate any type of operation. The block number (5) concerns the 10 light beams of the LiDAR sensor, here it is possible to analyze whether the robot detected an obstacle or not, being an essential part for the simulation for properly work. This block (5) connects to (7) because as soon as an obstacle is detected a flag must be activated, thus showing the warning signal to the user, as soon as the danger is no longer a concern, the flag will be cleared showing the green safety signal again. The block number (6) is merely for the user to know which value the robot is getting on the stop line, this must therefore stop when the read value represents the stop line. Finally, block number (8) is where the user can change the state/color of the traffic light, between green and red.

-This page intentionally left blank-

Chapter 5

Results and Discussions

This chapter presents the tests carried out to verify if the developed project fulfills the assumed objectives and solves, in fact, the problem described in chapter 1. For a better understanding, the results of each test are preceded by a brief description of the test performed and the expected results.

After all the work developed, several tests were carried out, to understand the functioning of the system and to improve the simulation. Next sections will detail those most relevant tests and contributed to a better analysis of the system.

5.1 Line Follower Algorithm Results

As a main objective the robot should follow the support lines and drive within the circuit. This has been successfully completed, thus observing proper behavior by the robot within the safety limits. However, and as expected due to the difficulty of the track, in some situations, as in the figure 5.1, where the support line is no longer visible, the robot behaviour tends to be unstable, and may even lose the control in an extreme case. Other point of difficult performance for the robot is on crossing points, figure 5.3. At this stage the robot starts a pre-defined program by the user but when is the moment to return to autonomous driving mode in some cases he represents a non stable behaviour. This happens mainly when the user asks the robot to turn either left (3) or right (1). As soon

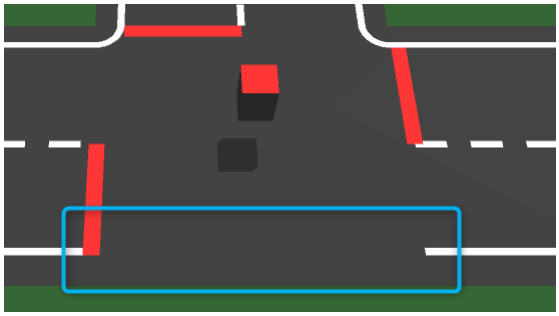


Figure 5.1: No bounder line

as the automatic process ends, the robot has difficulties in quickly detecting the support line again and if he is at an intersection, he quickly see the line of the opposite lane, which makes him misunderstand the behavior to take.

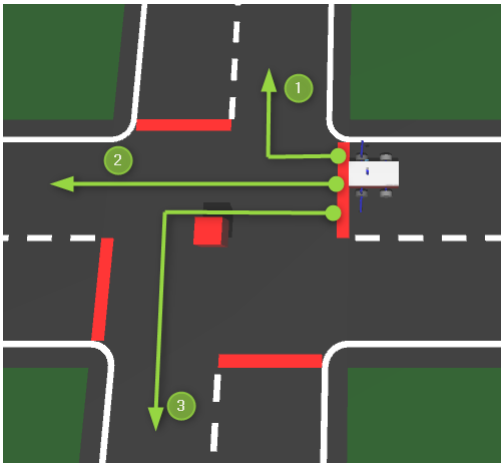


Figure 5.2: Crossing options

5.2 Obstacle Overtake Algorithm Results

Avoiding collisions with obstacles, averting them or stopping is part of the project's objectives. Therefore, the test for this type of situation involves analyzing whether or not the robot is able to make decisions correctly, overcoming the obstacle and returning to its normal carriageway without losing control. The figure 5.3 demonstrates this process divided into four phases.

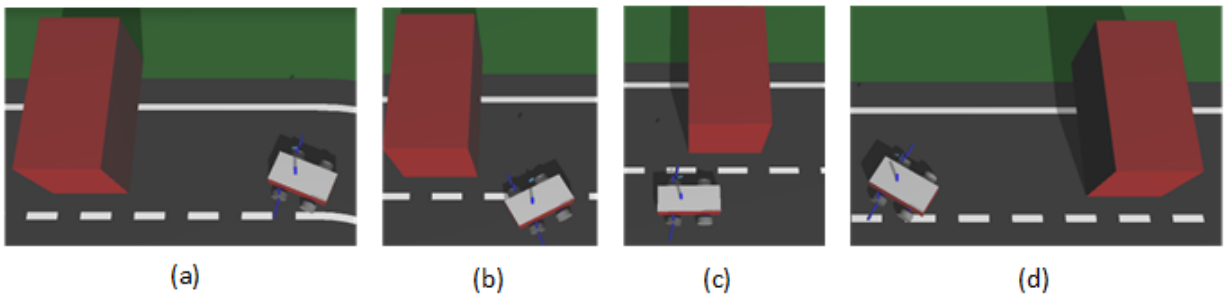


Figure 5.3: Overtake obstacle sequence

In (a) it's visible the robot obstacle approaching, here it will slow down, see figure 5.4, and start the overtaking maneuver. In (b), with the speed already reduced, the robot moves to the left lane in order to start tracking the left boundary line (c) until the LiDAR sensor gives the information that the obstacle has been passed in and that he can resume his normal lane, movement analyzed in (d).

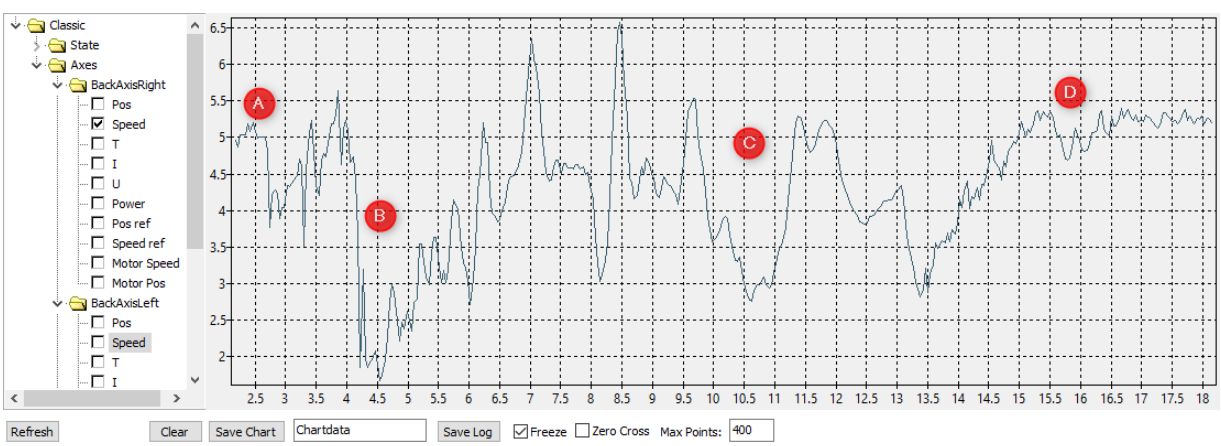


Figure 5.4: Velocity analyses when the robot is overtaking, where y axis represents the velocity and x axis the time in seconds.

After this figure 5.4 is possible to understand the correct behaviour of slow down and overtake the obstacle is successfully compiled by the robot, thus guaranteeing its safety and a correct functioning of the simulation.

5.3 Variable Values Adjust

In order to check the robot controller, a straight line track was created for the vehicle follow the boundary line while maintaining the offset to the middle of the screen. Starting the exercise at different starting points ((a) and (b) of the figure 5.6), it is possible for the user to alternate, via try and fail, the error weight of each line (k_3, k_4, k_5), as illustrated in figure 5.5.

k3	0.1
k4	0.05
k5	0.04

Figure 5.5: Weight of each strategic line of camera analyses

In (c) it is possible to see the robot taking its final position by performing a quite abrupt maneuver due to the values of the controls may be out of adjustment. At (e), unlike (d), the vehicle does not maintain the distance to the middle of the image, ending up making its way over the bounding line.

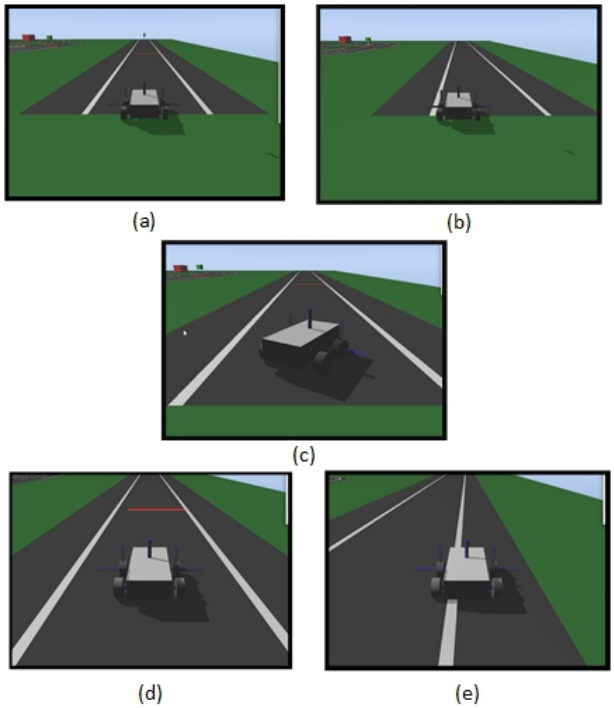


Figure 5.6: PID Values calibration track

During the course of the project, these values of K_3, K_4 and K_5 were agreed, thus allowing safe and precise driving. It should also be noted that the control must be adjusted differently for driving in a straight line than in a curve. This makes control even more dynamic. The following figure 5.7 shows a graph showing the value of the X position of the robot being adjusted, from the starting point (8.5) to the desired value (9.6) where it remains constant.

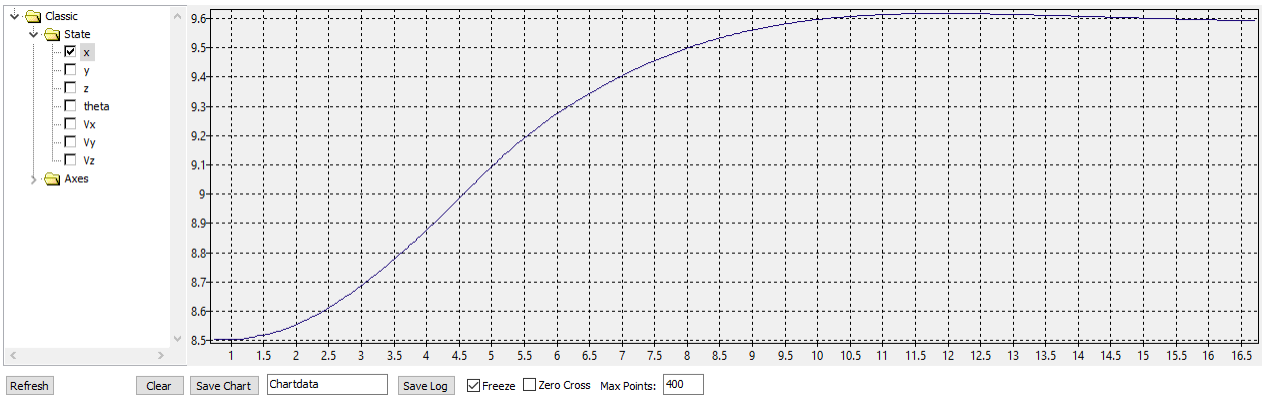


Figure 5.7: Robot X axis position adjust

5.4 Other Analyses

Some aspects went beyond what was initially thought. The creation of a simulation track much more complex than initially thought (two curves and two straights), introduced a whole new complexity to the control system, providing the creation of crossing points where the control system has a new level of complexity. Thus, is above the expected results the creation of traffic lights to control the passage or not of the robot, leading to changes in the main code, introducing new states and new points on the user interface.

5.5 Unsolved

In many circumstances the robot presents a unstable behaviour, for e.g. when the robot needs to return to the "follow right" state after a crossing point or when is performing a double curve in a opposite steering directions, or after overtake an obstacle, are just a few

examples that stand out. This unstable behaviours can cause a lose control procedure to the robot, and afterwards enter in a totally unexpected dangerous behavior. For this situation it will be important to have a security service that allows the autonomous vehicle for a case of impossibility to resume the route, to stop being safe for passengers and for the external environments, either other persons, objects, animals, etc... The full automation using the traffic lights, to stop and go the robot, is another point not totally solved for the current simulation. So far, the robot is programmed to stop at the stop points and wait for the user input to resume its normal course. It was expected that this input would be introduced in an automatic way by the traffic lights changing between the green, can move forward, or red, must wait.

Chapter 6

Conclusions and Future Work

Throughout this dissertation several steps have been taken to achieve the goal of develop a control system in a simulator for an autonomous vehicle to provide autonomous driving. Using the "Simtwo" simulator, where the actuation and sensing model was developed in order to achieve a control and a visualization system for an autonomous vehicle.

The robot/vehicle is able to drive on a road, avoid obstacles and alert in case of warning as expected.

Using camera and LiDAR as a main sensors inputs, the system is able to clearly sense and recognize possible real situations simulated by the software.

All the simulation environment was created to represent the world scenario of a car driving.

The system is able to identify and alert the in case of a imminent danger situation, and even reacts in order to avoid the obstacle.

A graphical user interface was development to help the user to understand the system control and easy interact with the robot when the driver wants or when the robot need.

In short, the main objectives of this work were achieved. Now leaving space for the continuation of the work already developed, with the introduction of new ideas and improvements.

6.1 Future Work

As next steps, it is suggested to continue this same project increasing its complexity. Beginning by fully automating the simulation thus creating a full secure and autonomous section. Immediately implementing the change of traffic light control to an autonomous solution. Afertwards, there is a suggestion to introduce more complexity into the simulation, increasing the level of realism. For this it will be essential to introduce new robots in the simulation representing new real scenarios, where for example a robot must interact with another (V2V) or even with any environment around it (V2x), at this point of complexity for the simulation a better understanding of the environment around the robots and in turn better decision making on the part of the robots will be achieved.

Bibliography

[1] E. Louay. (2017). Lidar and the autonomous vehicle revolution for truck and ride sharing, (visited on 04/01/2020).

[2] (2016). Leddar optical time-of-flight sensing technology: A new approach to detection and ranging, [Online]. Available: http://www.yellowscan.fr/products/yellowscan-mapper2 (visited on 04/01/2020).

[3] (). Yellowscan mapper ii, [Online]. Available: http://www.yellowscan.fr/products/yellowscan-mapper2 (visited on 04/01/2020).

[4] C. for Disease Control and P. (CDC). (2020). Road traffic injuries and deaths—a global probleml.

[5] S. Wasserman. (2018). Simulation is the fastest way to get autonomous cars on the market, [Online]. Available: https://www.ansys.com/blog/simulation-is-the-fastest-way-to-get-autonomous-cars-on-the-market (visited on 04/01/2020).

[6] H.-P. Schoener, "The role of simulation in development and testing of autonomous vehicles," Sep. 2017. DOI: 10.1007/978-3-658-21194-3_82.

[7] *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems*, Jan. 2014. DOI: https://doi.org/10.4271/J3016_201401. [Online]. Available: https://doi.org/10.4271/J3016_201401.

[8] Synopsys. (2020). Dude, where's my autonomous car? the 6 levels of vehicle autonomy, [Online]. Available: https://www.synopsys.com/automotive/autonomous-driving-levels.html (visited on 04/02/2020).

[9] A. Technology. (). Adaptive cruise control with stop & go function, [Online]. Available: https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/adaptive-cruise-control-with-stop-go-function (visited on 04/01/2020).

[10] T. Drive. (2017). How long, really, until self-driving cars hit the streets? [Online]. Available: http://www.thedrive.com/tech/16768/how-long-really-until-self-driving-cars-hit-the-streets (visited on 04/01/2020).

[11] INTEL. (2019). 2019 ces: Great wall motors, mobileye join forces to deliver adas and autonomous driving solutions in china and beyond, (visited on 04/01/2020).

[12] R. Thakur. (2016). Optical sensors are a key technology for the autonomous car, [Online]. Available: https://www.fierceelectronics.com/components/optical-sensors-are-a-key-technology-for-autonomous-car (visited on 04/02/2020).

[13] Nidec. (2020). For lidar scanning solution polygon mirrors and motors, [Online]. Available: https://www.nidec-copal-electronics.com/us/featuring/lidar-polygon/vs_galvo/# (visited on 04/03/2020).

[14] R. Amadeo. (2020). Google's street view cars are now giant, mobile 3d scanners, [Online]. Available: https://arstechnica.com/gadgets/2017/09/googles-street-view-cars-are-now-giant-mobile-3d-scanners/ (visited on 04/03/2020).

[15] B. S. Jahromi. (2019). Ultrasonic sensors in self-driving cars, [Online]. Available: https://medium.com/@BabakShah/ultrasonic-sensors-in-self-driving-cars-d28b63be676f,%20urldate%20=%20%7B2020-04-03%7D.

[16] J. Honkanen. (2017). Mems and sensors in automotive applications on the road to autonomous vehicles: Hud and adas 468 views, [Online]. Available: https://www.slideshare.net/Jari512/mems-and-sensors-in-automotive-applications-on-the-road-to-autonomous-vehicles-hud-and-adas-75552352 (visited on 04/03/2020).

[17] F. Markets. (2017). 3d cameras in autonomous vehicles.

[18] D. L. B. Eliot. (2017). Sensor fusion for self-driving cars, [Online]. Available: https://www.aitrends.com/ai-insider/sensor-fusion-self-driving-cars/ (visited on 09/23/2020).

[19] CBInsights. (2020). 40+ corporations working on autonomous vehicles, [Online]. Available: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/ (visited on 04/01/2020).

[20] D. Riley. (2020). Alphabet's waymo self-driving car division raises $2.25b in its first external fundraising, [Online]. Available: https://siliconangle.com/2020/03/02/alphabets-waymo-self-driving-car-division-raises-2-25b-first-external-fundraising/ (visited on 04/03/2020).

[21] C. Neiger. (2020). 3 top autonomous-vehicle companies to watch, [Online]. Available: https://www.fool.com/investing/2020/01/17/3-top-autonomous-vehicle-companies-to-watch.aspx (visited on 04/03/2020).

[22] F. VC. (2019). Decoding the autonomous driving landscape, [Online]. Available: https://medium.com/@firstmilevc/avlandscape-8a21491f1f54 (visited on 04/01/2020).

[23] D. S. A. Beiker. (2012). Legal aspects of autonomous driving, [Online]. Available: https://digitalcommons.law.scu.edu/cgi/viewcontent.cgi?article=2726&context=lawreview (visited on 04/01/2020).

[24] D. Clarke. (2019). Simulation in autonomous driving – why societal change is as necessary as technical innovation simulation in autonomous driving – why societal change is as necessary as technical innovation, [Online]. Available: https://www.designnews.com/electronics-test/simulation-autonomous-driving-why-societal-change-necessary-technical-innovation/88487843660929 (visited on 04/01/2020).

[25] (2016). Nvidiasim, [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/ (visited on 04/10/2020).

[26] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[27] A. C. Madrigal. (2017). Inside waymo's secret world for training self-driving cars, [Online]. Available: https://amp.theatlantic.com/amp/article/537648/ (visited on 04/01/2020).

[28] ND. (2017). Lg web site, [Online]. Available: https://amp.theatlantic.com/amp/article/537648/ (visited on 04/01/2020).

[29] P. Costa, J. Gonçalves, J. Lima, and P. Malheiros, "Simtwo realistic simulator: A tool for the development and validation of robot software," *Theory and Applications of Mathematics & Computer Science*, vol. 1, pp. 17–33, Apr. 2011.

[30] J. Go, B. Browning, and M. Veloso, "Accurate and flexible simulation for dynamic, vision-centric robot.," Jul. 2004, pp. 1388–1389. DOI: 10.1109/AAMAS.2004.38.

[31] T. Laue, K. Spiess, and T. Röfer, "Simrobot – a general physical robot simulator and its application in robocup," in *RoboCup 2005: Robot Soccer World Cup IX*, A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 173–183, ISBN: 978-3-540-35438-3.

[32] J. C. Zagal and J. Ruiz-del-Solar, "Uchilsim: A dynamically and visually realistic simulator for the robocup four legged league," in *RoboCup 2004: Robot Soccer World*

Cup VIII, D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 34–45, ISBN: 978-3-540-32256-6.

[33] M. Friedmann, K. Petersen, and O. von Stryk, "Simulation of multi-robot teams with flexible level of detail," in *Simulation, Modeling, and Programming for Autonomous Robots*, S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. von Stryk, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 29–40, ISBN: 978-3-540-89076-8.

[34] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, "Virtual robot experimentation platform v-rep: A versatile 3d robot simulator," in *Simulation, Modeling, and Programming for Autonomous Robots*, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 51–62, ISBN: 978-3-642-17319-6.

[35] K. Kumar and P. S. Reel, "Analysis of contemporary robotics simulators," in *2011 International Conference on Emerging Trends in Electrical and Computer Technology*, IEEE, 2011, pp. 661–665.

[36] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich, "Simulating complex robotic scenarios with morse," in *Simulation, Modeling, and Programming for Autonomous Robots*, I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 197–208, ISBN: 978-3-642-34327-8.

[37] J. Lima, J. Gonçalves, P. Costa, and A. Moreira, "Humanoid realistic simulator - the servomotor joint modeling.," vol. 2, Jan. 2009, pp. 396–400.

[38] G. Amaral and P. Costa, "Simtwo as a simulation environment for flight robot dynamics evaluation," *U.Porto Journal of Engineering*, vol. 1, pp. 80–88, Sep. 2017. DOI: 10.24840/2183-6493_001.001_0008.

[39] G. L. M. Aaron Staranowicz, "Asurveyandcomparisonofcommercialandopen-source robotic simulator software," pp. 3–4, Sep. 2017.

[40] J. Smith. (2019). Why simulation is the key to building safe autonomous vehicles, [Online]. Available: https://amp.theatlantic.com/amp/article/537648/ (visited on 04/01/2020).

[41] Y. Eady. (2019). Simulations can't solve autonomous driving because they lack important knowledge about the real world, [Online]. Available: https://medium.com/@strangecosmos/simulation-cant-solve-autonomous-driving-because-it-lacks-necessary-empirical-knowledge-403feeec15e0 (visited on 04/01/2020).

[42] T. Pinho, A. P. Moreira, and J. Boaventura-Cunha, "Framework using ros and simtwo simulator for realistic test of mobile robot controllers," in *CONTROLO'2014 – Proceedings of the 11th Portuguese Conference on Automatic Control*, A. P. Moreira, A. Matos, and G. Veiga, Eds., Cham: Springer International Publishing, 2015, pp. 751–759, ISBN: 978-3-319-10380-8.

[43] P. P. C. Emily Maggioli Motaz Ayiad, "Simtwo tutorial - tutorial 1 - getting started version 1.0 feup," 2018.

[44] A. Abreu. (2020). Github simtwo repository, [Online]. Available: https://github.com/AderitoAbreu/SimTwo (visited on 02/11/2020).

[45] D. Foundation. (2017). Duckietown.